The Cognitive Plausibility of Deep Neural Networks with Attention

Matt Rounds

Master of Science by Research
Institute for Language, Cognition and Computation
School of Informatics
University of Edinburgh

2016

Abstract

Recent advances in the use of deep neural networks for image description have employed attention mechanisms to boost performance. We investigate whether these attention mechanisms are similar to those used by humans performing the same task. We implement and train a network that utilises hard attention to perform image description, and compare its fixation patterns to those of humans generated during an image description task with eye-tracking. Our result is primarily negative: whilst there is some correlation between the distribution of human fixations and those of the model, it is less than that between humans and a centre bias baseline. We analyse why this might be the case, and suggest avenues for further investigation.

Acknowledgements

First and foremost, thanks go to Professor Frank Keller, to whom I am indebted for his unwavering support and encouragement throughout this project.

To my CDT Cohort, who made this year one of the most vibrant and interesting of my life thus far.

To my mother, without whom I would have never made it this far.

To Kate, for everything.

This work was supported in part by the EPSRC Centre for Doctoral Training in Data Science, funded by the UK Engineering and Physical Sciences Research Council (grant EP/L016427/1) and the University of Edinburgh.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Matt Rounds*)

Contents

Chapter 1

Introduction

1.1 Motivation

The role of attention in cognition has long been a source of interest: both in philosophy (e.g. [12],[13],[61],[65]), and in psychology (e.g. [29],[19]). Attention has been argued to play a significant role in structuring conscious experience, object binding, and even introspection. In neuroscience, it has been suggested to play an important role in the process by which the brain performs inference on the causes of its sensory input (e.g. [5],[20],[27]).

To perform tasks the brain must overcome two major problems: that it lacks the processing power to fully represent its external environment, and that it lacks full synchronic access to that environment. A solution to the second would be to access local regions sequentially; naively, we might scan an entire visual environment top-left to bottom-right and then perform some computation. This runs straight into the first problem: because our scan would be undiscerning with respect to the goal at hand, we would have to store a full representation as we go. In addition, scanning takes time - time during which the environment may well have changed.

Evolution's solution to this, at least in part, is to to make the scanned locations part of the processing loop. Those visual locations judged to be most salient/relevant to the task at hand are often those most likely to be included in a saccade [20]. This means that computation can be performed online, as the brain's environment becomes just

that which it expects to be task-relevant, and the computational demand at any point in time is related only to the limited content of our attention [29],[18], and the internal dynamics of the brain.

We can ask: in tasks that humans excel at, is there something to be gained by mimicking this capacity-limited control loop? Recent work in Recurrent Neural Networks (RNNs - see (2.1.3)) seems to suggest that including some sort of attention (either soft or hard) over an image presented does indeed improve performance in image description tasks [67]. A paper by Mnih et al (2014) [44] takes this further, and shows that only allowing an RNN access to a local part of an image at a time, and giving it control over which part that is (i.e. formulating the problem as a partially observed Markov decision process [57]), allows a network to perform well both online (as a game playing agent), and as a classifier.

The aim of this thesis is to investigate whether recent models that have used attention mechanisms to make practical gains in tasks such as image description are doing so utilising policies of attention that are similar to those used by humans. If this were the case, we would be able to use them as models of human behaviour. In addition, a close similarity might suggest that there is a underlying mechanism by which constraints on access to information about an environment shape an agent's interaction with that environment, irrespective of that agent's biological or mechanical aspect.

1.2 Goals

Our goals are twofold: firstly, to implement a neural-network based model of the type described in Mnih et al (2014) [44], for the cross-modal task of image description. Secondly, if successful, we want to investigation how similar the model's attention is to that of humans: for example, we know that when humans describe images, there is an observable relationship between saccade path and resultant sentence, and between what object is attended to and what object is named [9] [6].

We have access to a (small) dataset of fixation patterns, image descriptions and images

(from Coco and Keller (2012) [9]), which would serve as a potential test set to examine the difference between how our implemented model scans an image to generate a sentence, and how humans scan the same image to generate sentences.

Ideally, we would want there to be some correlation, as the model enacts constraints that we hypothesise structure human information processing:

Hypothesis: Constraints on an agent's access to information about its environment, such as hard visual attention, structure that agent's interactions with its environment. Specific tasks require the development of specific behaviours, which must include a policy of optimal sampling. Agents that are similarly constrained should develop similar policies when presented with the same task. Therefore, **a network required to output a successful description of an image, which is constrained to 'see' only a certain part of that image at a time, and only has a limited number of timesteps to generate the description, should adopt a policy of attention similar to that of humans given the task of describing the image**.

It should be noted that the subject's constraints are not quite identical to the network's; there is no explicit time limit in the experimental set-up. However, humans have been historically subject to task-driven time constraints (we starve if we fail to find food in time, are eaten if we fail to classify a blurry patch of undergrowth as a tiger in time, etc). As such, humans are driven to examine those parts of an environment that contain the most task-relevant information/are the most interesting [40]. The network's timestep constraint is an attempt to restrict the network in a similar way and encourage it to sample from the most relevant parts of the image.

1.3 Outline of report

Chapter 1 introduces the project, describes our motivation and states the hypothesis. It also includes this outline. Chapter 2 consists of the background. It describes neural networks in detail; simple, recurrent, convolution, and stochastic. It also provides an overview of those aspects of human visual attention relevant to our investigations. Chapter 3 is an account of our methodology and implementation. It details the devel-

opment of the model, and various extensions that were explored. It also describes the metrics used to compare the model to the human test set. Chapter 4 reports the model results, and evaluates them. It contains both quantitative and qualitative analysis of the model on the original test set, and a comparison of the performance of the model to human data. Chapter 5 concludes, and discusses potential further work.

Chapter 2

Background

2.1 Neural Networks

2.1.1 Simple Neural Networks

2.1.1.1 Overview

Our notation follows, for the most part, that in Chris Bishop's excellent 'Neural Networks for Pattern Recognition'[3], to which the reader is directed for a more detailed account. We will approach neural networks from the point of view of *supervised* learning; one of the major paradigms in Machine Learning. Neural networks can be used for other tasks, but these are not within the remit of this thesis.

Supervised learning involves the prediction of some output $\boldsymbol{y} \in Y$ from some input $\boldsymbol{x} \in X$. For simplicity we will assume $\boldsymbol{x}$ and $\boldsymbol{y}$ are both vectors (either may be one-dimensional). We might be interested in using $\boldsymbol{y}$ as a discriminant function; such that we assign $\boldsymbol{x}$ to some class C_k if $y_k(\boldsymbol{x}) > y_j(\boldsymbol{x})$ for all $k \neq j$. Or we might be interested in the values of $\boldsymbol{y}$ directly, for some regression task.

In either case, we would like to learn some function $f : X \to Y$ such that $f(\boldsymbol{x}) = \boldsymbol{y}$ for all $\boldsymbol{x} \in X$. To help with this, we have a training set, $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{N}$, which we can think of as N samples drawn i.i.d from some joint distribution $P(\boldsymbol{x}, \boldsymbol{y})$ over the input and output spaces X and Y. We also have a way to measure how good some prediction is, i.e. some

non-negative real valued loss function $L(f(\boldsymbol{x}),\boldsymbol{y})$ that measures the difference between the target output $\boldsymbol{y}$ and that predicted by a particular f, $f(\boldsymbol{x})$.

For any f, we can talk about its associated *risk* [63]; namely, the expectation of the loss function under the distribution $P(\boldsymbol{x},\boldsymbol{y})$.

$$R(f) = \int d\boldsymbol{x}d\boldsymbol{y}P(\boldsymbol{x},\boldsymbol{y})L(f(\boldsymbol{x}),\boldsymbol{y}) \tag{2.1}$$

However, as we do not know $P(\boldsymbol{x},\boldsymbol{y})$, we cannot compute this quantity directly. This is where the dataset comes in: assuming it is drawn i.i.d from $P(\boldsymbol{x},\boldsymbol{y})$ we can use it to compute an estimation of $R(f)$.

$$R(f) \approx \frac{1}{N}\sum_{n=1}^{N} L(f(\boldsymbol{x}_n),\boldsymbol{y}_n) \tag{2.2}$$

In general, therefore, we want to specify some fixed class of functions $\mathcal{F}$, and select from it some particular function f^* that solves

$$f^* = \underset{f\in\mathcal{F}}{\operatorname{argmin}} \frac{1}{N}\sum_{n=1}^{N} L(f(\boldsymbol{x}_n),\boldsymbol{y}_n) \tag{2.3}$$

A neural network represents via its structure a particular set of functions, which are parameterised by some weights $\boldsymbol{w}$. This reduces the problem of finding a particular function, f^*, to the problem of finding a particular weight vector $\boldsymbol{w}^*$, that satisfies

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}\in\mathbb{R}^d}{\operatorname{argmin}} \frac{1}{N}\sum_{n=1}^{N} L(f(\boldsymbol{x}_n;\boldsymbol{w}),\boldsymbol{y}_n) \tag{2.4}$$

2.1.1.2 Structure

The simplest instance of a neural network is a one-unit net whose single dimensional output y is linear in the elements of its d-dimensional input $\boldsymbol{x} = (x_1, x_2, ..., x_d)^T$. Namely,

$$y = f(\boldsymbol{x};\boldsymbol{w}) = \boldsymbol{w}^T\boldsymbol{x} + w_0 \tag{2.5}$$

where $\boldsymbol{w} = (w_1, w_2, ..., w_d)^T$, and w_0 is the bias. For ease of notation, we will generally only write $\boldsymbol{w}^T\boldsymbol{x}$, where we have implicitly defined two new $(d+1)$-dimensional vectors $\boldsymbol{w} = (w_0, \boldsymbol{w}_{old})$, and $\boldsymbol{x} = (1, \boldsymbol{x}_{old})$. This corresponds to an additional bias input which always outputs 1 (see Figure 2.1). This single unit output can be used either for regression or for two-class classification, where we typically threshold the output at 0,

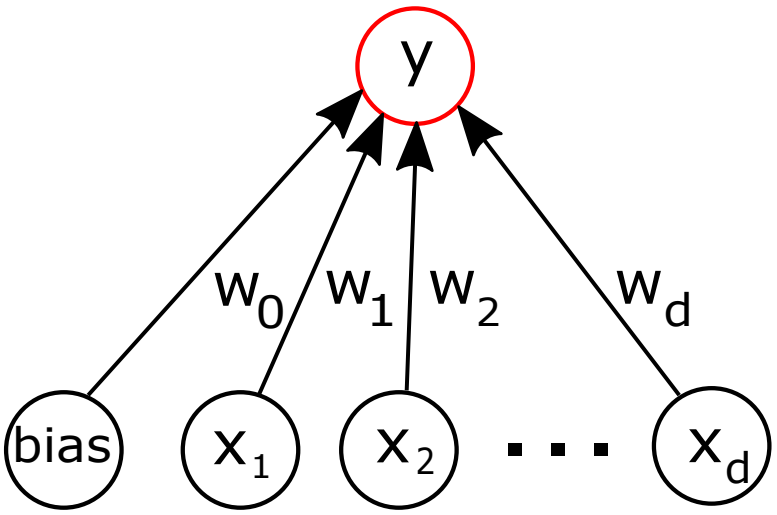


Figure 2.1: Single unit network. Arrows denote connections/paths, red circles inner products (i.e. $\boldsymbol{w}^T\boldsymbol{x}$), black circles input units.

and assign $\boldsymbol{x}$ to class C_1 if $y \geq 0$ and to class C_2 otherwise.

Naturally we can extend this architecture to a multi-dimensional $\boldsymbol{y} \in \mathbb{R}^c$, where we now have a c x $(d+1)$ weight matrix $\boldsymbol{W}$ instead of a vector $\boldsymbol{w}$ (see Figure 2.2):

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} \tag{2.6}$$

We can think of this as a 'single-layer' neural network. The red circles in the figures can be thought of as processing units; each performs some operation on 1 or more inputs and generates a single output.

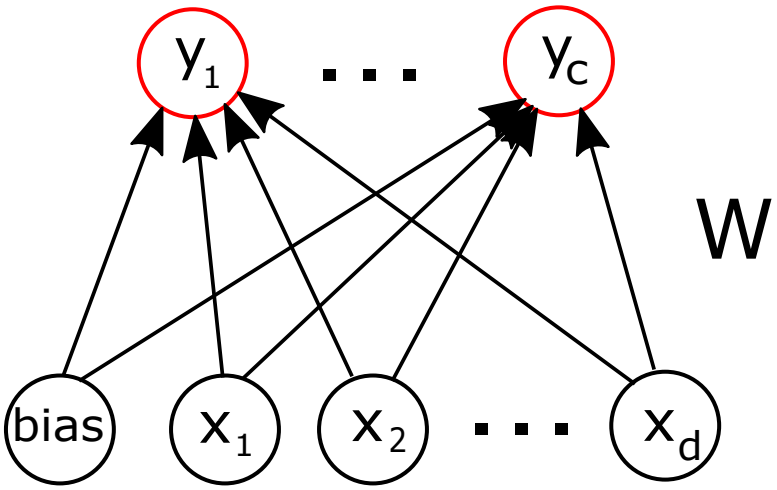


Figure 2.2: Single layer network. Arrows denote connections/paths, red circles inner products (i.e. between each row of $\boldsymbol{W}$ and $\boldsymbol{x}$), black circles input units.

The last most common structural extension is to increase the number of layers, such that the outputs of the first layer (here $\boldsymbol{z}$) serve as the inputs to the next layer, which has its own weight matrix $\boldsymbol{W}^2$ (see Figure 2.3),

$$\begin{aligned} \boldsymbol{z} &= \boldsymbol{W}^1 \boldsymbol{x} \\ \boldsymbol{y} &= \boldsymbol{W}^2 \boldsymbol{z} \end{aligned} \tag{2.7}$$

and we can keep adding additional layers. The layers not directly used as output layers are called 'hidden' layers.

The intuition behind stacking layers like this is that higher layers learn with respect to higher level features of the original input; those features generated by the layers below. This means that the network can work with more abstract features of the input data, which for complex tasks is useful. This is because the mapping from, say, pixel values to the presence or absence of a cat in an image is highly complex, whereas mapping from pixel values to low-level features (edges, bright spots) is much simpler. The next layer can then learn a mapping from low-level features to medium-level features, and so on. Spreading out the optimisation task like this makes it easier and more robust to noise. Once we introduce non-linearities and other structural modifications, networks which are 'deep' in this way can be very powerful.

2.1.1.3 Non-linearities

Normally, the linear combination of the input variables to a particular unit are then transformed by a non-linear activation function. In the networks described above, we can think of this as having been the identity. The problem with such a linear activation function is that we need a huge number of units to approximate complex non-linear functions. In addition, we may find such that networks are unstable as the transfer function is not normalisable. There is nothing stopping outputs along favoured paths increasing without bound.

To avoid such problems, therefore, the activation is often non-linear. For some partic-

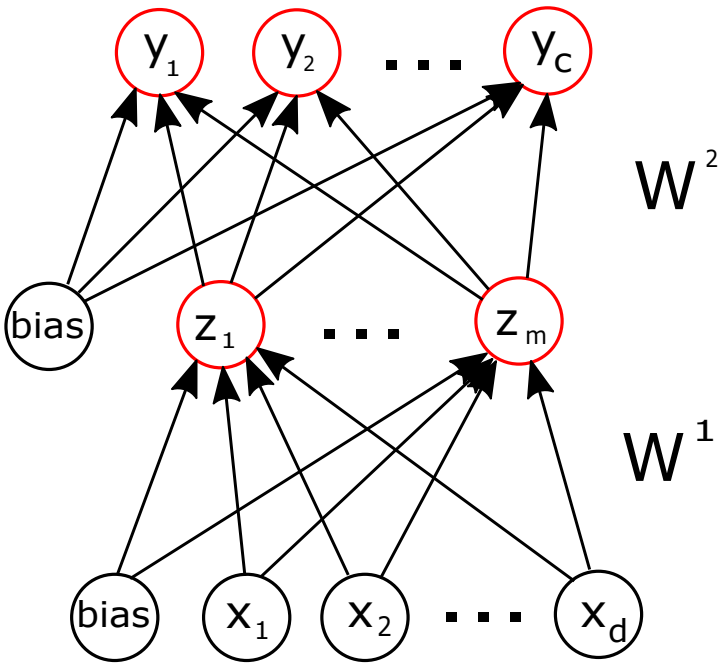


Figure 2.3: Multi-layer network. Arrows denote connections/paths, red circles inner products, black circles input units. Hidden layer has dimensionality m. Note that each layer has a bias unit, which has a continuous output of unity.

ular hidden unit, j, the activation[1] a_j is computed as follows (note parallel with (2.6)):

$$a_j = \sum_{i=0}^{d} w_{ji}^{(1)} x_i \tag{2.8}$$

where we are taking the bias to be x_0. This activation is then transformed by some nonlinear function $g(.)$ to give

$$z_j = g(a_j). \tag{2.9}$$

Commonly used $g(.)$ include the logistic sigmoid, tanh, and ReLU, where $g(a_j) = \max(0, a_j)$, which was introduced as a way to avoid the problems of vanishing gradients common to the two former [70] (where $\frac{dg}{da} \to 0$ as a moves away from the origin). Note that there is no requirement for all layers to have the same activation function, although each layer should have the same activation function, because otherwise we would be arbitrarily transforming different parts of the hidden vector into different spaces, which would not make much sense.

[1]The use of 'activation' and 'activation function' throughout the literature in this sense can be confusing. The activation of the unit, a_j, is the weighted sum of its inputs. The activation function, $g(.)$, is a nonlinear function that transforms a_j (see (2.9)).

2.1.1.4 Back-propagation

We have outlined the basic structure of simple neural networks. As discussed in Section 2.1.1.1, a particular neural network represents a fixed set of functions $\mathcal{F}$, where each specific function f corresponds to a particular weight vector $\boldsymbol{w}$. Finding an optimal function f^* is an optimisation problem equivalent to finding an optimal weight vector, $\boldsymbol{w}^*$, that satisfies

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}\in\mathbb{R}^d}{\operatorname{argmin}} \frac{1}{N}\sum_{n=1}^{N} L(f(\boldsymbol{x}_n;\boldsymbol{w}),\boldsymbol{y}_n) \tag{2.10}$$

The question this section addresses is: how do we find this optimal weight vector? Particularly in deep, multi-layered networks, if an output unit produces an incorrect response with respect to a particular input vector, determining which hidden units and thus which weights are responsible is a non-trivial task. This is known as the *credit assignment problem* ([3] p140).

The solution relies on choosing a loss function L which is a *differentiable* function of the network outputs. As the network outputs are themselves a differentiable function of the weights (assuming we also choose non-linear $g(.)$ that are differentiable), this means we can compute the derivatives of the loss with respect to the weights. We can then think of the loss as a surface above the weight space, and perform gradient descent (or some more complex first-order method -see Section 2.1.1.5) to find weight values that minimise the loss.

Computing the derivatives turns out to correspond to the back-propagation of errors through the network, hence the name. For some arbitrary hidden unit, we have defined the weighted sum of the inputs as the activation a_j (2.8), and the output of that hidden unit as a nonlinear (differentiable) transformation of this, $z_j = g(a_j)$ (see (2.9)).

Let us additionally define the loss corresponding to a particular input pattern, $L(f(\boldsymbol{x}_n;\boldsymbol{w}),\boldsymbol{y}_n)$ as L^n, such that the total loss L can be written

$$L = \sum_n L^n \tag{2.11}$$

where

$$L^n = L^n(y_1,...,y_c) \tag{2.12}$$

Assume we have presented an input $\boldsymbol{x}_n$ to the network, and computed all activations, the final output $\boldsymbol{y}_n$, and the loss, L^n. This process is often called a 'forward pass'. We are now interested in the gradient of the error with respect to a particular weight, w_{ji}, which we note only contributes to L^n via the weighted sum of inputs to unit j, a_j, to which it connects from unit i. Hence, by the chain rule,

$$\frac{\partial L^n}{\partial w_{ij}} = \frac{\partial L^n}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}} \tag{2.13}$$

We define the first part of this term as the 'delta' or 'error'; the derivative of the loss with respect to the activation of the j^{th} unit, a_j:

$$\delta_j \equiv \frac{\partial L^n}{\partial a_j} \tag{2.14}$$

This notation allows us to derive a recursive formulation that lets us to compute the gradients of the loss with respect to each of the weights by passing the loss back down the network. From (2.9) it follows that

$$\frac{\partial a_j}{\partial w_{ij}} = z_i \tag{2.15}$$

i.e. the output of the unit at the other end of the weight w_{ij}. Note for j on the first layer, $z_i = x_i$, from the input. We can now re-write (2.13) as

$$\frac{\partial L^n}{\partial w_{ij}} = \delta_j z_i \tag{2.16}$$

As we have computed all of the values for z_i during the forward pass, we only need compute the value δ_j for all units to get values for the gradients with respect to each weight. For some output unit k, we can write

$$\begin{aligned}\delta_k \equiv \frac{\partial L^n}{\partial a_k} &= \frac{\partial L^n}{\partial y_k}\frac{\partial y_k}{\partial a_k} \\ &= \frac{\partial L^n}{\partial y_k} g'(a_k)\end{aligned} \tag{2.17}$$

Where we have used $y_k = g(a_k)$ from the more general (2.9). For hidden units, we recognise that contributions to the loss from that unit correspond to all paths that connect that unit to the loss; i.e. the units k to which hidden unit j sends connections.

Again by the application of the chain rule, we can write

$$\begin{aligned}\delta_j \equiv \frac{\partial L^n}{\partial a_j} &= \sum_k \frac{\partial L^n}{\partial a_k}\frac{\partial a_k}{\partial a_j} \\ &= \sum_k \frac{\partial L^n}{\partial a_k}\frac{\partial a_k}{\partial z_j}\frac{\partial z_j}{\partial a_j} \\ &= \frac{\partial z_j}{\partial a_j}\sum_k \frac{\partial L^n}{\partial a_k}\frac{\partial a_k}{\partial z_j} \\ &= g'(a_j)\sum_k \delta_k w_{kj}\end{aligned} \tag{2.18}$$

where we have made use of (2.8) and (2.9) in the final line.

In this final form, the rule for the computation of deltas is recursive: the delta of each unit is a weighted sum of the deltas from the units to which it connects (i.e. those in the layer above), multiplied by the derivative of that unit's transfer function g. As such, we can see that applying the rule iteratively down the network is equivalent to passing, or propagating, the errors back down the network.

Finally, assuming we have several training inputs, we can simply sum the losses between them, and write

$$\frac{\partial L}{\partial w_{ji}} = \sum_n \frac{\partial L^n}{\partial w_{ji}} \tag{2.19}$$

2.1.1.5 Gradient Descent

Once we have the gradients of the error with respect to each of the weights, we want to find a $\boldsymbol{w}^*$ that corresponds to a minimum in the loss such that $\nabla L = 0$. A standard approach to this is gradient descent. This involves thinking of the loss as a surface above the weight space, where each point in weight space $\boldsymbol{w}$ has a corresponding scalar loss value $L(\boldsymbol{w})$, given our training data. The (local) gradient of the loss with respect to the weights, ∇L points in the direction of steepest slope: to find a minimum we should travel in the opposite direction. To avoid missing an optima, we should do so in small steps, iteratively.

This leads to an update of the form

$$w_{ji}^{(\tau+1)} = w_{ji}^{(\tau)} - \eta \left[\frac{\partial L}{\partial w_{ji}} \right]^{(\tau)} \tag{2.20}$$

where η is a hyperparameter that governs how far we 'step' each iteration, and τ labels the iteration step. The procedure for simple gradient descent is as follows.

1. For item $n = 1$, perform a forward pass, store the outputs z_j^n of each unit, and compute the loss, L^n.
2. Backpropagate the deltas, and using δ_j^n, l^n and z_j^n compute $\frac{\partial L^n}{\partial w_{ji}}$ for each weight w_{ji}.
3. Repeat 1 and 2 for every item, $n \in \{2, ..., N\}$, in the training set, storing the gradients cumulatively such that $\left[\Delta w_{ij}\right]_{new} = \left[\Delta w_{ij}\right]_{old} + \frac{\partial L^n}{\partial w_{ji}}$.
4. Update gradients according to (2.20) using $\frac{\partial L}{\partial w_{ji}} = \left[\Delta w_{ij}\right]_{final}$
5. Set $\Delta w_{ij} = 0$ for all i, j.
6. Repeat 1-5 until convergence.

Note we initialise the weights to small random values: this is to break symmetry, and avoid problems where two units are have identical incoming and outgoing weights, and thus can never learn to represent different features (as they will always receive the same gradient).

As specified, one of the main problems with gradient descent is that by summing all of the gradients for a training set before we make an update, we run the risk of smoothing the relationship between input and loss to the point where we do not find very good optima. One solution to this is to update after each training item, which are sampled (without replacement) uniformly at random. This is called online Stochastic Gradient Descent, or SGD, and allows the optimisation to find lower minima. It corresponds nicely with our underlying assumptions that our training examples are drawn from some underlying distribution $P(\boldsymbol{x}, \boldsymbol{y})$: by drawing uniformly from the training set, we are indirectly drawing from this distribution (see Section 2.1.1.1).

However, SGD can often be too stochastic; and additionally, performing a forward pass and a backwards pass for each training item is costly in terms of time. In prac-

tice, therefore, we subsample a small number B of training items from our dataset ($B \sim 10 - 100$) and pass these forward through the network in parallel. This means that the network input is now a matrix of dimensionality $B\text{x}(d+1)$, where each row is a single input instance. This additional batch dimension is constant for hidden states and output states, and we compute the losses and backpropagate as before.

There are many first- and higher-order optimization algorithms that have been developed and can be used in combination with neural networks; it is not relevant to this thesis to go into detail here. For our experiments we used one of two variants of SGD. Either **RMSprop** [23], which divides the learning rate for a weight by a running average of the magnitudes of recent gradients for that weight, or **Adam** [32], which modifies the paramter updates based on adaptive estimate of lower-order moments. Both have been shown to work well with deep neural networks, and with the image description dataset we are using [67].

2.1.2 Convolutional Networks

The layers described in the simple neural networks above are often described as 'fully connected', as every input is connected to every unit in the layer, and has a corresponding weight. One of the downsides of this sort of architecture is that multi-dimensional inputs (such as images) are effectively flattened when presented to the network; fully connected layers do not preserve spatial structure. This section will describe a somewhat different architecture, the convolutional layer, which does preserve spatial structure, and is effective for image classification.

Consider a 2-dimensional black and white image of size 28x28 as input $\boldsymbol{x}$ (an MNIST [36] digit, for example). Let us assume a 24x24 hidden layer $\boldsymbol{z}$ (we will keep the hidden layer 2-dimensional because we want to preserve the spatial structure for higher layers in the network). Instead of fully connecting every pixel in the input to every unit in the hidden layer, we instead choose to connect a particular unit in the hidden layer to a small group of adjacent pixels only. This small region is called the *receptive field* for that unit, and has its own bias and weights. Here we will assume that each pixel's receptive field is a 5x5 patch of the image (see Figure 2.4).

Moreover, the bias and weights for each unit in the hidden layer, $z_{j,k}$, are shared with

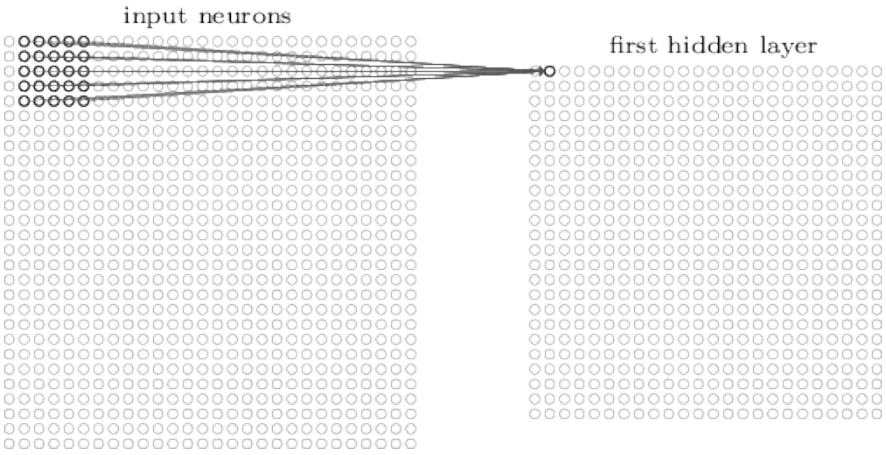


Figure 2.4: Mapping from 5x5 receptive field to single unit in the hidden layer. Figure taken from Neural Networks and Deep Learning, Chapter 6 [45]

all of the other units. One advantage of this is that it massively reduces the number of parameters in the network - in the toy example discussed here, a fully connected layer of 100 units would have $(28x28x100)+(100) = 78500$ parameters. Compare this with the $(5x5)+1 = 26$ parameters required to define the receptive field described above. This means that we can build much deeper networks for the same cost in memory. We can write

$$z_{j,k} = g(\sum_{l=1}^{5}\sum_{m=1}^{5} w_{l,m}x_{j+l-1,k+m-1} + w_0) \tag{2.21}$$

where g is the activation function of the unit, $w_{l,m}$ are the shared weights, and w_0 is the shared bias. In effect we can think of the hidden layer as the result of the convolution of the 28x28 image with the 5x5 weight matrix, which defines what we commonly call the kernel, or feature map. This latter term is due to the fact that a single kernel will only detect one particular local pattern or 'feature'; so we can think of the output of the hidden layer with respect to a particular input as mapping the presence or absence of the particular pattern detected.

In (2.21) we are assuming that the stride of the convolution is 1, i.e. we shift the kernel by 1 pixel and then recompute the weighted sum. It is possible to increase the stride so that the overlap between receptive fields is reduced. An advantage of this is that the amount of processing required to perform a forward pass is reduced, so large networks can benefit substantially.

This architecture becomes particularly powerful when we increase the number of feature maps, and the number of layers (see Figure 2.5). The description given in Section 2.1.1.2, of the first layer of a net learning a mapping from pixel values to low-level features, and the next layer learning a mapping from low-level features to medium-level features, is particularly appropriate here. In addition to the features becoming more abstract, the preserved spatial structure means each unit corresponds to larger and larger feature maps as we rise through the layers.

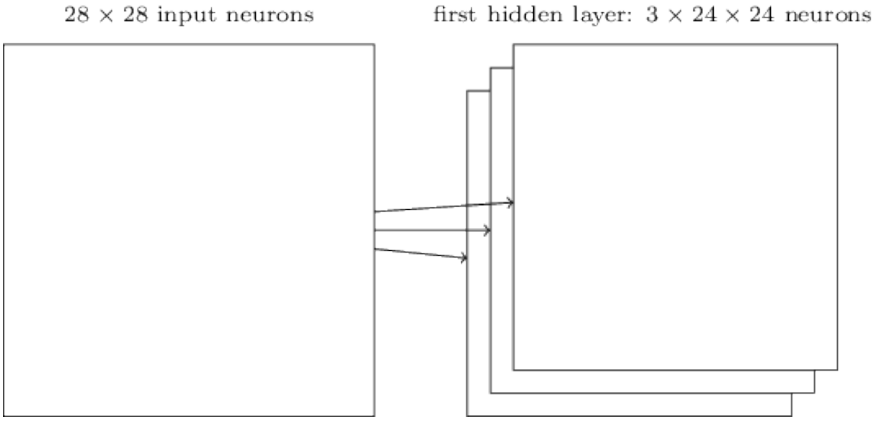


Figure 2.5: Multiple feature maps from a single input layer. Figure taken from Neural Networks and Deep Learning, Chapter 6 [45]

Note that to handle multiple feature maps as input (or an RGB image), kernels can be 3-dimensional. For example if our input image was 3x28x28, then our first layer kernels would be of size 3x5x5. For large numbers of layers with multiple kernels per layer and even moderately large input sizes (our implemented model takes input images of size 3x224x224), the computational cost of convolutional networks becomes very high very quickly. One of the ways to reduce this is to insert pooling layers after convolutional layers; these produce a reduced version of the feature maps output by the convolutional layers.

The simplest type of pooling is max pooling [55]. This takes as input some region and outputs to a single unit the maximum of that region (see Figure 2.6). For a 2x2 region, max pooling layer $\boldsymbol{y}$ can be computed by

$$y_{j,k} = \max\{z_{k,j}, z_{k+1,j}, z_{k,j+1}, z_{k+1,j+1}\}. \tag{2.22}$$

The intuition is that once we have located a particular feature in the image, we can discard some of the precise location information, but still retain information about its rough location with respect to other features. This saves us memory, but keeps the information about important features and their rough spatial distribution intact. Back-

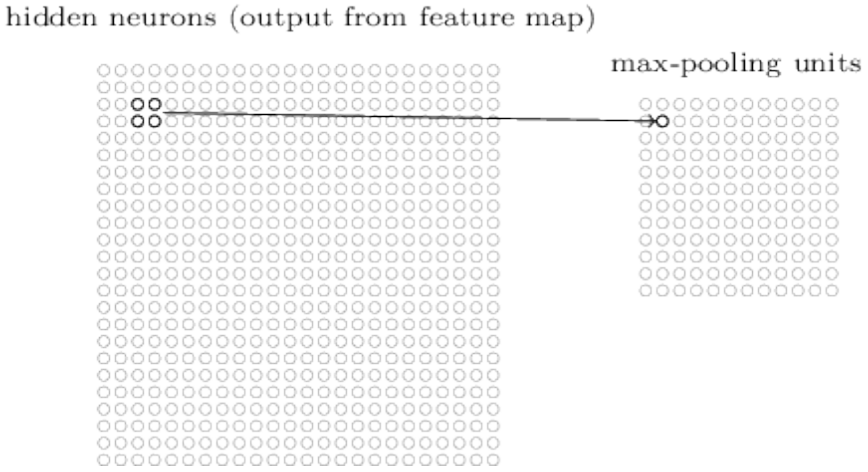


Figure 2.6: Max Pooling layer reduces the size of the feature map without losing important information. Figure taken from Neural Networks and Deep Learning, Chapter 6 [45]

propagation through convolutional layers is slightly more complicated than through fully connected layers, although still derivable from the chain rule. The difference is that we want to consider just that region of hidden units k connected to each input unit j, and sum just their (weighted) contribution to the gradient of the input unit.

$$\delta_j = g'(a_j) \sum_{k \in \text{connected to } j} w_{kj}\delta_k \tag{2.23}$$

As it turns out, if we have an mxm kernel, this is mechanically identical to zero padding the feature map with $m-1$ units all around and convolving the resultant δ matrix with the kernel rotated by 180° [49].

For max pooling layers, which have no trainable weights, we simply keep track of which units were the maximum during the forward pass, and zero all other gradients. This means that gradients from max pooling layers will be quite sparse.

For this project, we use a pretrained convolutional network to extract high-level features from our images. We used the Oxford VGGnet [56], which is a network with

16 weight layers trained for the ImageNet Large Scale Visual Recognition Challenge 2014 [52] classification. This involves classification of 1000 object categories, which means that the network should learn features that correspond usefully to the detection of objects, and thus features that are also relevant to image description.

The structure of the Oxford VGGnet is outlined in Table 2.1.

2.1.3 Recurrent Networks

The second major structural improvement to simple neural networks is to modify them to take (and output) a *sequence* of inputs. This is necessary to allow networks to model data that is explicitly temporal, such that the probability of some output is necessarily conditioned on the presence or absence of previous outputs. Natural language modelling is an example; we think of the probability of the t^{th} word in a sentence as conditioned on the previous words, i.e. $p(w^{(t)}|w^{(t-1)},...,w^{(1)})$.

2.1.3.1 RNNs

The most straightforward way to do this is to treat the hidden layer as a recurrent layer; as well as taking the input at time t, $\boldsymbol{x}^{(t)}$, we also pass the network its own hidden state at the previous timestep, $\boldsymbol{z}^{(t-1)}$. The input $\boldsymbol{x}^{(t)}$ passes through a weight matrix $\boldsymbol{W}^{(1)}$, as before. The previous hidden state is also passed through a weight matrix that we shall call $\boldsymbol{W}^{(R)}$. This is the basis of the full Recurrent Neural Network (RNN) architecture (Figure 2.7).

To account for the contribution of earlier states of the network to the loss for a particular output, we need a new learning algorithm. This is known as back-propagation through time, or BPTT. As before (2.16), for a training instance n, we can compute the gradient of the loss with respect to some weight in the net w_{ij} by

$$\frac{\partial L^n}{\partial w_{ij}} = \delta_j z_i \tag{2.24}$$

where w_{ij} connects the i^{th} unit to the j^{th} unit, and z_i is the output of the i^{th} unit. Note that here, if w_{ij} is a weight in the matrix $\boldsymbol{W}^{(R)}$, then i and j are units in the same layer

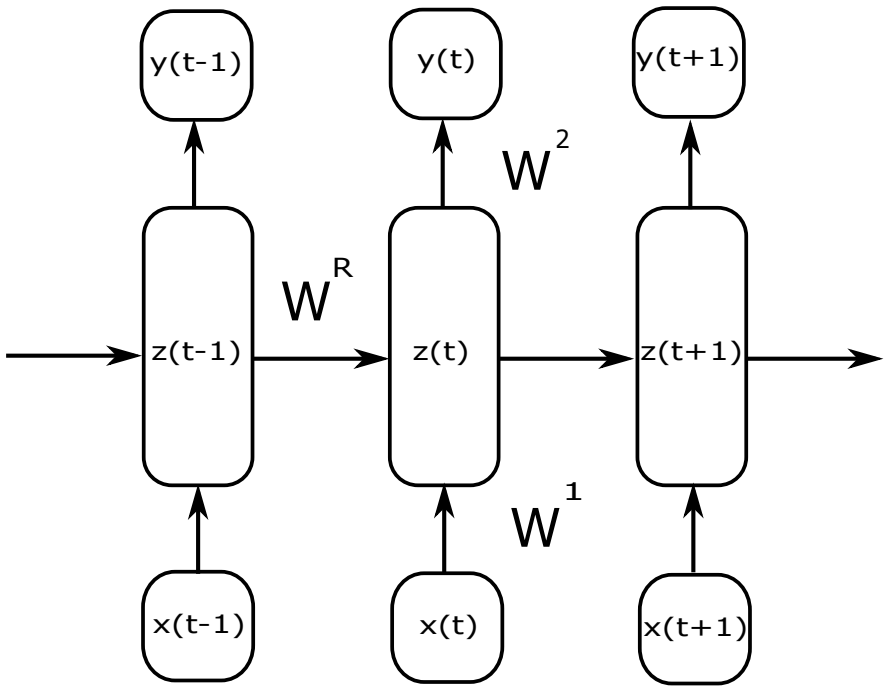


Figure 2.7: Basic recurrent neural network architecture. Arrows denote weight matrices, boxes vectors.

(and might even be the same unit), just at different time steps.

Output deltas are computed as before, but for what follows we make the time steps explicit, so we say that the delta for the j^{th} unit in the hidden layer at the last timestep before the output we are considering is $\delta_j(t)$.

$$\delta_j(t) = g'(a_j(t)) \sum_k \delta_k(t) w_{kj}^{(2)} \tag{2.25}$$

For the recurrent matrix $\boldsymbol{W}^{(R)}$ at this timestep, we can compute the gradients using the basic backpropagation formula,

$$\frac{\partial L^n}{\partial w_{ji}^{(R)}(t)} = \delta_j(t) z_i(t-1) \tag{2.26}$$

where $z_i(t-1)$ is the output of the j^{th} unit of the hidden layer at timestep $(t-1)$. Continuing back through time, we must also compute the deltas for the hidden units for the previous timestep, which are now being backpropagated via the recurrent matrix $\boldsymbol{W}^{(R)}$.

$$\delta_i(t-1) = g'(a_i(t-1)) \sum_j \delta_j(t) w_{ji}^{(R)} \tag{2.27}$$

We can continue this process for an arbitrary number of timesteps, τ. To compute the gradient updates for each parameter in a batch, we simply sum the gradients across all τ, and across all items, B.

$$\Delta w_{ij} = \sum_n \sum_t \frac{\partial L^n}{\partial w_{ij}(t)} \tag{2.28}$$

Intuitively, when we do BPTT for τ time steps, the recursive updates effectively create a deep network of depth τ. This unfolded network has far fewer parameters (and thus is less computationally costly) than a simple net of equivalent depth. Unlike in normal backpropagation, where we alter the parameters of our network to minimise our loss with respect to just one input at time t, we are now altering the parameters of our network to also take into account the effect of previous inputs $(t-1),(t-2),...,(t-\tau)$ on the loss at time t.

Certainly in the case of language modelling, one advantage is that this can improve performance [43]. This is because deep networks can be more expressive than shallow networks, and because we can learn to model structures over a sequence, rather than just using the current input to the network.

One downside is that BPTT can become trapped in local optima, which can prevent it from finding parameter combinations that generalise well.Another downside is that the gradients have a tendency to vanish as we backpropagate (as they will be < 1 for all of our standard activation functions). This means that making τ very large will provide diminishing returns, because the network will rapidly forget earlier inputs. A solution to this is the LSTM architecture (see Section 2.1.3.2).

2.1.3.2 LSTMs

The LSTM architecture solves the problem of vanishing gradients by defining a hidden layer of memory blocks, each of which contains a **cell state** and acts as a recurrent unit that is able to pass on some information from previous timesteps unchanged, by virtue of the fact that changes to the cell are passed through multiplicative sigmoidal gating units, which can zero inputs and outputs depending on what the the inputs $x(t)$ are, and what the output of the cell at the previous timestep, $h(t-1)$ was.

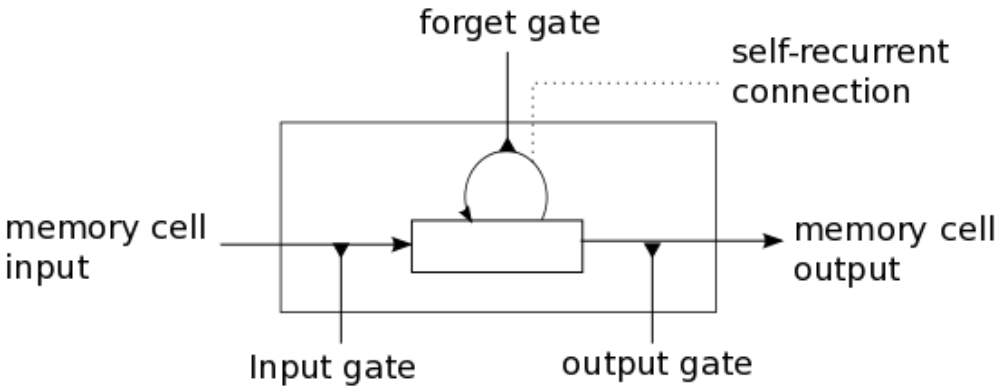


Figure 2.8: Structure of an LSTM memory cell. Figure taken from http://deeplearning.net/tutorial/lstm.html, [25]

These gates are trainable, and generally consist of an **input gate**, **output gate** and **forget gate** (Figure 2.8). The input gate $i(t)$ governs whether the input is actually passed to the cell state $C(t)$. The output gate $o(t)$ controls if the current activation of the cell state is passed out of the block. The forget gate $f(t)$ controls whether the cell state is set to zero or maintained. The cell state itself is linear, but the block as a whole has non-linear input and output activation functions, which we will call g_{in} and g_{out} (although in practice they may be the same function, often tanh).

The cell state at some time t is given by

$$C(t) = f(t) \odot C(t-1) + i(t) \odot \tilde{C}(t) \tag{2.29}$$

where $\odot$ is the elementwise multiplication of two vectors, $\tilde{C}(t)$ is the candidate update state given by

$$\tilde{C}(t) = g_{in}(W^{(C)} \cdot [x(t), h(t-1), 1]) \tag{2.30}$$

where we are concatenating the input at this timestep, the previous output of the LSTM, and the unity input from the bias unit. $W^{(C)}$ is the candidate matrix, $\cdot$ is the inner product. $f(t)$ and $i(t)$ are given by

$$f(t) = \sigma(W^{(f)} \cdot [x(t), h(t-1), 1]) \tag{2.31}$$

and

$$i(t) = \sigma(W^{(i)} \cdot [x(t), h(t-1), 1]) \tag{2.32}$$

respectively, where σ denotes the logistic sigmoid, which transforms the output into the range [0,1] and allows it to be used for gating.

The output of the whole block is the new cell state multiplied by the output gate,

$$h(t) = o(t) \odot g_{out}(C(t)) \tag{2.33}$$

where $o(t)$ is given by

$$o(t) = \sigma(W^{(o)} \cdot [x(t), h(t-1), 1] \tag{2.34}$$

The LSTM is fully differentiable, and can be backpropagated through. Note there are several different architectures possible; some include 'peepholes' between cell state and various gates (such that the gating vectors are computed directly from $C(t-1)$ rather than from $h(t-1)$). We use the architecture outlined above, without the peepholes.

2.1.4 The Reinforce Algorithm

All of the neural network architectures discussed thus far fall under the supervised learning paradigm (see Section 2.1.1.1), which is a staple of Machine Learning. The networks can be backpropagated through to find the gradient of the loss with respect to each parameter, and the expected loss, $\mathbb{E}\big[L(f(\boldsymbol{x}), \boldsymbol{y})\big]$ (see (2.1)), can be minimised using first order optimisation methods such as SGD. The units in these networks are deterministic.

However, stochastic units also exist. These often occur in an agent-based reinforcement paradigm, where an agent chooses its actions stochastically to generate appropriate behaviour, for which it is then rewarded. In this case backpropagation, which relies on the assumption that the output of the net can be fully determined from the the inputs of the net via knowledge of the local connections within, is inappropriate [66]. We still need an approximation to the gradient of the expected loss, however, and this is where the REINFORCE (Reward Increment = Nonnegative Factor times Offset Reinforcement time Characteristic Eligibility) algorithm comes in.

Let us define some stochastic layer, which takes as input a vector $\boldsymbol{x}$ from the layer below (which may or may not be the input layer), and computes some intermediate state $\boldsymbol{p}$ from its activation $\boldsymbol{a}$ deterministically by

$$p_j = \frac{e^{\boldsymbol{x}^T \boldsymbol{w}_j}}{\sum_i e^{\boldsymbol{x}^T \boldsymbol{w}_i}} \tag{2.35}$$

where $\boldsymbol{x}^T \boldsymbol{w}_j = a_j$, and we have included the unit bias in $\boldsymbol{x}$. By construction, $\sum_j p_j = 1$, so we can use the elements of this state as the parameters of a categorical, or multinoulli distribution, where $g(z = i|\boldsymbol{p}) = p(i) = p_i$. The layer then samples from this distribution $g(\boldsymbol{p})$, and outputs an integer z, $z \in \{1, .., k\}$ where k is the size of the layer (i.e. $\boldsymbol{a} \in \mathbb{R}^k$). Some architectures might output a one-hot vector $\boldsymbol{z}$, where assuming we have sampled an index, i,

$$z_k = \begin{cases} 1, & \text{if } k = i. \\ 0, & \text{otherwise.} \end{cases} \tag{2.36}$$

We want to find an approximation to the gradient of the expected reward, $\mathbb{E}[r]$, (r might simply be the negative of the loss), with respect to the layer weights $\boldsymbol{W}$ and, if the stochastic layer is high up in the net, all those other weights to which we can backpropagate via the chain rule (if the lower connections are deterministic, backpropagation will still work). For some general parameter w_{ij} in such a stochastic network, the REINFORCE update takes the form

$$\Delta w_{ij} = \alpha_{ij}(r - b)\frac{\partial \log g(\boldsymbol{p})}{\partial w_{ij}} \tag{2.37}$$

where b is a reinforcement baseline (often the model's current estimate of the expected reward), and α_{ij} is a non-negative factor (often constant for all ij). If we can differentiate $\log g(\boldsymbol{p})$ with respect to its parameters, then the REINFORCE algorithm will work. The algorithm works by measuring the relationship between variations in local behaviour (the parameters of the distribution being used to generate the randomness), and variations in global behaviour (via the reward), and uses these to compute an update estimate that will follow the required gradient [66].

In a network which is in part deterministic, and in part stochastic, we can use backpropagation through the deterministic parts, but we must use (2.37) whenever we need partial derivative information on the input side of a stochastic unit. Effectively we are backpropagating from several places; from the loss function at the top of the network,

and from each stochastic unit within the network, to which we broadcast the reward r before commencing.

Presenting each training item, sampling from $g(\boldsymbol{p})$, and averaging can be thought of as sampling from the underlying distribution $P(\boldsymbol{x},\boldsymbol{y})$, although due to our introduction of stochastic units, $P(\boldsymbol{x},\boldsymbol{y})$ is now more properly the marginal $\sum_z P(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})$. Monte Carlo such as this estimators are unbiased, but high variance: increasing the dimensionality of the probability space with respect to which we are computing the expectation increases the variance, and can cause the algorithm to suffer convergence problems. The baseline, b helps to combat this. Simple strategies to calculate this include a moving exponential average, such as suggested in Williams (1992)[66],

$$b(t) = \lambda r(t-1) + (1-\lambda)b(t-1) \tag{2.38}$$

where $0 < \lambda \leq 1$. More complex strategies can involve training the baseline as a parameter alongside the rest of the network, via a regression loss.

2.2 Human Visual Attention

2.2.1 Eye Movements

The features of the eye of interest to us are the ways in which visual information is constrained, and the way in which the eye moves to compensate for this. Light passes into the eye via the pupil, and is projected (upside down) onto the retina by the lens [28]. A small area of the retina, called the fovea, which spans less than 2°of the visual field, features a high density of cone cells. Cone cells are sensitive to visual detail, and as they are sparse across the rest of the retina, this means that to see an object clearly, we must move our eyes so that the light from that object falls directly onto the fovea. This is known as *foveating* the object.

Information from objects foveated is weighted more heavily in cortical processing. De Valois and De Valois (1980)[16] have shown that the central 2.5°of the visual scene is processed by around 25% of the visual cortex. The process of moving the eye to examine an image is not smooth. Of primary interest are *fixations*, where the eye

remains still for a period of time that can last from tens of milliseconds to several seconds. Fixations are generally considered to be a marker of (visual) attention [28], although exceptions to this correspondence exist. For the purposes of this thesis, which is concerned specifically with modelling visual attention, we will assume that a fixation corresponds to attention and hence uptake for subsequent processing. This process is what we are attempting to model using an RNN (see Section 3.1).

The eye moves from fixation to fixation by means of rapid motions called *saccades*, which typically take around 30-80ms to complete. In our implementation, which will generate a new point of fixation at every time-step of the neural network, we will approximate saccades as instantaneous. This limits our avenues of analysis. We cannot compare many temporal aspects of scanpaths, such as position duration or saccadic velocity, as our model effectively teleports to a new fixation every time-step, and remains there for just that time step. These restrictions are an unavoidable feature of the current paradigm in modern neural networks, which simply define some mapping $f : X \to Y$, and lack internal time-dependent dynamics.

However, there is still plenty of scope for *spatial* comparison of scanpaths, and it is on this analysis that we focus (see Chapter 4).

2.2.2 Scanpaths

The term 'scanpath' can be traced to Noton and Stark in the 1970s (e.g. [46]). In modern usage, it is used to denote how the eye moves through space, typically the 'series of oculomotor events' for a single participant during a single trial [28]. For our human data, because we are not making use of the temporal aspects of the data per se, when we use the term we will be referring to an ordered sequence of fixation events. These events are recorded as locations on the image in question.

When we use the term scanpath to describe the output of our model, we will be similarly referring to an ordered sequence of locations generated by the model. 'Ordered' means that we can compare some temporal details at a very rough level of granularity. Scanpath features such as backtracks (where a second saccade goes in the opposite

direction to the preceding saccade [50]), inhibition of return (the tendency to avoid refixations [48]), and local and global subscans [69][62] are just some examples of phenomena with a temporal aspect that we can infer from an ordered sequence of fixations. The presence of such in our model might be taken as some indication of a plausibly human attention mechanism (see Chapter 4).

2.2.3 Gist

Human visual understanding does not rely on fixation information alone. Attended features are supported by a gist. Gist is 'the general category of the scene' (Sampanes et al 2008 [53]). It can include information about schemas, and local and global features. We can think of it as the contextual knowledge humans bring to bear on their environment; top down expectations, verbal encodings, information distilled from earlier visual search etc.

This gist helps inform our visual system, and improves our task performance; it can be thought of compensating, in part, for the paucity of information received from the 2°of visual field we can foveate at any particular point in time. Feature recognition is easier when given external structure; object detection is faster in a coherent scene [2]. This is because the eye uses the gist to help inform its scan path, and thus to make more effective decisions about where to fixate next.

Neural networks do not have a way to model Gist directly, without some modification. As will be discussed in Section 3.1.1.6, however, we found during development that our networks performed more effectively if provided with contextual information at the commencement of description or throughout. We can think of this contextual information as acting as the gist in our model; it allows the model to perform as though it were equipped with the task- and environment-specific contextual knowledge that humans utilise to more easily parse complex visual environments.

Layer Type	No. Kernels (if conv)	Layer size
Input RBG Image	**(3)**	**3x224x224**
Convolution	**64**	
Convolution	**64**	
Max Pooling		
Convolution	**128**	
Convolution	**128**	
Max Pooling		
Convolution	**256**	
Convolution	**256**	
Convolution	**256**	
Max Pooling		
Convolution	**512**	
Convolution	**512**	
Convolution	**512**	
Max Pooling		
Convolution	**512**	
Convolution	**512**	
Convolution	**512**	
Max Pooling		
Fully Connected		**4096**
Fully Connected		**4096**
Fully Connected		**100**
Softmax		

Table 2.1: Oxford 16-layer VGGnet structure. Activation functions on Convolution layers are ReLU. All kernels are 3x3 with stride 1. Net has 14.7 million trainable parameters. Final convolution layer before last Max Pooling layer provides C_j glimpse vectors (see Section 3.1.2 Reinforce Categorical for details)

Chapter 3

Methodology and Implementation

3.1 Image description model

As our intention was to generate fixations for comparison to human data, our model utilised a hard attention mechanism as its primary source of visual information. To successfully describe an image, it had to learn both a stochastic policy over which parts of the image it should sample, and a mapping from the information provided by these visual samples to a distribution over its vocabulary from which, over successive time-steps, a high probability word sequence could be extracted.

This section will describe in detail the model implemented. The first subsection briefly covers the initial implementation, which used a bivariate Gaussian to model the sampling of fixations from the image, and which was eventually discarded due to computational limitations. These limitations motivated the final model architecture, which used a categorical distribution over vectors corresponding to specific receptive fields, and which is described in the second subsection.

3.1.1 Reinforce Normal

Our starting point was the Recurrent Attention model described in Mnih et al (2014)[44]. This model is used for classification tasks on the MNIST handwritten digit dataset, and noisy variants thereof. It comprises a recurrent layer (either a vanilla RNN layer, or an LSTM layer) which at time t takes as input the output from a 'glimpse' module,

g_t, the state of the recurrent layer at the previous timestep, h_{t-1}, and the output of a 'location' module, l_t. The recurrent layer's state, h_t serves as the input to a 'classification' module, which outputs a distribution over possible words, a_t. h_t is also passed to an 'action' module that generates a distribution over possible location co-ordinates, which is then sampled to produce x_t, the location to be foveated next. See Figure 3.1.

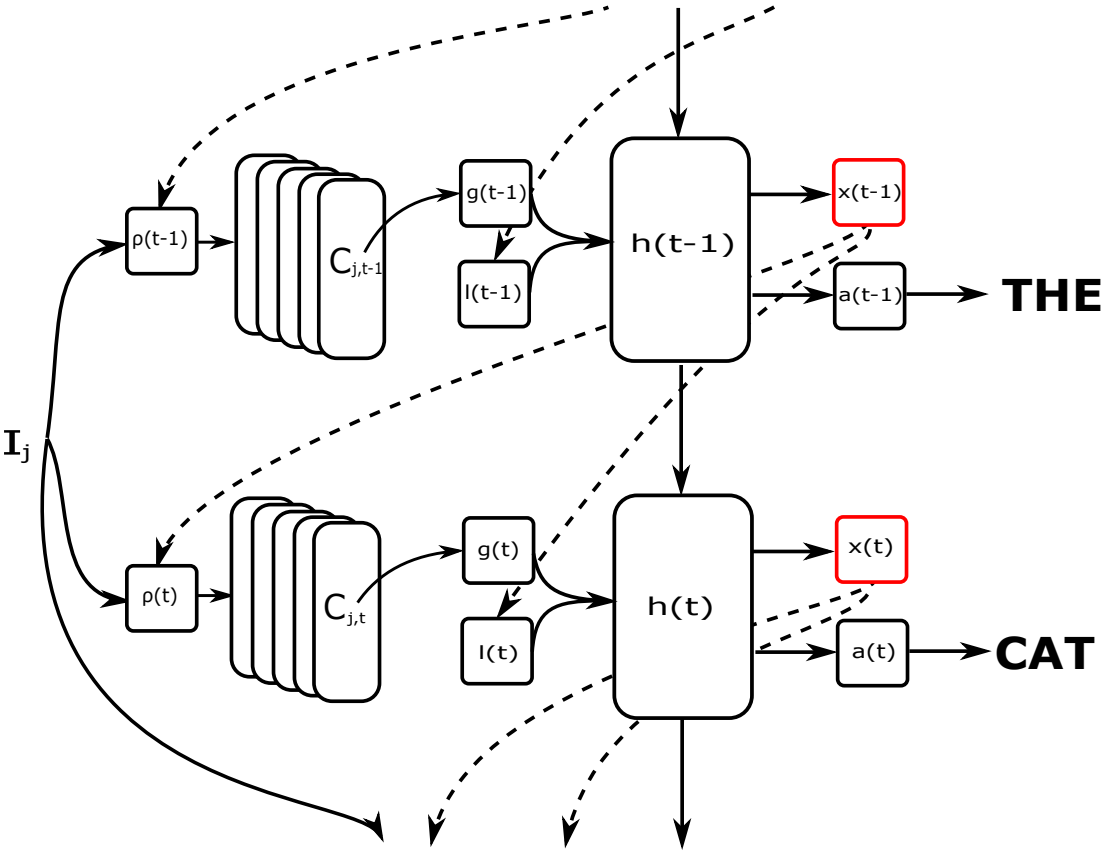
The architecture was appealing because it allows the whole net to be considered as an agent which makes a sequence of decisions about how it should interact with its environment (the image), and is then given a reward for the entire sequence. Its aim is to learn which decisions are most likely to maximise the expected reward, and given it only has a limited number of samples of small parts of the images, it must learn a policy of optimal fixation subject to this constraint. This aligns nicely with our motivating hypothesis that task-driven human behaviour is structured by the informational constraints it must overcome (see Section 1.2).

3.1.1.1 'Glimpse' module

The glimpse module takes as input the image, I_j (size 3x256x256), and the location sampled by the action module at the previous timestep, x_{t-1}. The location is a 2-dimensional vector in the range [-1,+1], which are the center coordinates of a small window (20x20 pixels, in initial experiments) extracted from the image, where (-1,-1) is the top left corner of the image. By analogy with the periphery of the human visual field, we also extract a 60x60 low resolution window around the same point, downsampled to a 20x20 tensor and concatenated with the high-resolution window to give a 20x40 snapshot, ρ_t (technically 3x20x40, as our images are RGB). Biological plausibility aside, it has also been shown by Mnih et al (2014)[44] that having a glimpse network with two different scales can improve performance.

This tensor is then passed through several convolutional layers, to extract useful features from the glimpse. During development, we used two convolution layers with 12 kernels each, which were trained from scratch by backpropagation. This was because the Oxford VGGnet [56], which we could have used as a pretrained convolution network, is trained for object classification on natural images. It was therefore unclear how such a pretrained net would perform when given low resolution image parts

Figure 3.1: Reinforce Normal Implementation: full arrows denote vector output/input within a timestep, dotted arrows denote vectors passing between timesteps, square boxes denote trainable weight layers, vertical rectangles denote LSTM units, overlapping rectangles denote convolutional layers, red squares denote that the output of the layer is stochastic, joining arrows denote vector concatenation, and letters denote the state of a layer or or LSTM unit at a particular timestep.



alongside high resolution image parts. However, training both the stochastic modules and several convolutional layers from scratch resulted in extremely slow to nonexistent convergence, and so we eventually tried passing the glimpses through the 16-layer VGGnet, to obtain a 1000 dimensional feature embedding vector $C_{j,t}$. To compensate for the VGGnet being trained for ImageNet classification [52], we allowed the net to finetune during training. This did provide some improvement in performance however, as will be discussed in the end to this subsection, the memory demands of this architecture were phenomenal; it was this limitation that eventually prompted substantial refactoring. Mnih et al [44] do not encounter the same problem as their primary dataset was the MNIST handwritten digit datset [36], which is much simpler.

Finally, the output of the convolution layers was passed through a fully connected layer and then through a ReLU non-linearity to a 256-dimensioned output, g_t, ready to be concatenated with that generated by the location module, l_t.

3.1.1.2 'Location' module

The location module takes as input the location sampled by the action module at the previous timestep, x_{t-1}. It passes this through a fully connected layer, and then through a ReLU non-linearity to a second 256-dimensioned output, l_t, which is concatenated with that from the glimpse module.

It might seem odd to transform a 2-dimensional vector into a 256-dimensional vector in this way, but the motivation is twofold. Firstly, we do not want to bias the network towards either the location or the information it receives from that location in its generation of either the classification or the action distributions at that time-step. Transforming the inputs to a similar size is approximately the same as assuming a uniform prior on their contributions. It may be that when trained on data the net learns to prioritise the one over the other, but we want this to be gained from the likelihood, rather than a prior.

The second motivation is that expanding a 2-dimensional input into a high-dimensional output allows for better representation of exactly what particular co-ordinates correspond to. The difference between (0.1,0.1) and (0.1,0.3) is about 20 pixels, which is substantial relative to our image size and task relevant features, but not that substantial when represented in just two dimensions. Once passed through a weight matrix and subsequent nonlinearity, these distinct inputs have the potential to result in much more distinct vectors, which should then make the rest of the model's job of meaningfully interpreting the location co-ordinates much easier.

3.1.1.3 Recurrent Layer

After the outputs of the location and glimpse modules have been concatenated, they are passed through a further trainable weight matrix and ReLU non-linearity. They then comprise the 512 dimensional input to the 512-unit LSTM layer along with that

layer's hidden state from the previous timestep. The hidden state, h_t, of the LSTM are output to both the classification module and the action module, and the LSTM at the next time step. See Section 2.1.3.2 for a more detailed description of the LSTM architecture. The primary motivation for using an LSTM over the simpler RNN was that this is a sentence generation task, and so we want an architecture that can handle the long range dependencies found in natural language.

3.1.1.4 'Classification' module

The classification module takes the 512 dimensional output from the LSTM, h_t, and transforms it to an $(M+1)$-dimensional vector a_t, where M is the size of the vocabulary, via a trainable weight matrix and a softmax nonlinearity, which enforces $\sum_i a_{t,i} = 1$. The +1 acts as a null token, to allow the net to train with (and output) sentences of varying length.

3.1.1.5 'Action' module

The action module takes the output of the LSTM and transfers it via a weight matrix and a tanh nonlinearity (which limits the outputs to the range [-1,1]) to two dimensions, which serve as the means of a bivariate Gaussian which is then sampled to produce the model's point of fixation for the next timestep, x_t. The variance of the Gaussian was kept constant at 0.11, which was found by cross validation to give reasonable results. This variance controls the exploration/exploitation trade-off; there is no reason why a more sophisticated model might not control this alongside the means. This might allow the model to capture its expectations of how accurate its beliefs about the optimal point of fixation were.

3.1.1.6 Gist

An initial approach was to give the net some time to scan the image prior to outputting the sentence, however in the interests of getting a working model, this search phase was shelved in favour of a 'gist' (see Section 2.2.3), which was intended to provide a low-resolution overview of the image - contextual information that in human cognition would be present in an individual's understanding of the task at hand, and additionally from their scanning across the image. In the Gaussian model, we approximated this

gist by a third, very low resolution window of size 180x180 centred on the point of fixation. This gist was reduced to a 3x20x20 tensor and concatenated with the high and medium resolution tensors extracted by the glimpse module, giving a 3x20x60 output tensor, $g(t)$.

3.1.1.7 Conclusion

As described, the developed network is not really a language model; the outputs of the classification module are not fed back in at subsequent time-steps, and so the net's states cannot be a parameterisation of $p(a_t|x_{t-1}, x_{t-2}..., a_{t-1}, a_{t-2}, ..)$, as would be the case if the net were fed some sequence of words, either those it has generated, or its target words at training time.

However, before substantial improvements could be made, it became clear that the architecture as described above was too unwieldy for substantial experimentation. The primary reason for this was that passing the glimpses through the convolution net was extremely costly in terms of time. In addition, because the pretrained nets had to be fine-tuned [1], there was a large memory cost: the Oxford VGGnet has around 14.7 Million parameters, and when training, each parameter's gradient at each time-step must be stored, which meant that on a NVIDIA Tesla K40m GPU with 11.5 GB memory, we could manage a batch size of 2 before catching an out-of-memory error. This meant that our first order optimisation algorithms (variants of Stochastic Gradient Descent - see Chapter 2) were *very* stochastic, and that training was thus very slow.

We therefore refactored the architecture of the model to take the convolutional network outside the recurrent network (see Section 3.1.2), so that we only had to forward each image through it once. This refactoring meant that our final model was closer to that of Xu et al (2015)[67] in its use of a Categorical distribution to sample locations. It was, however, quite successful: it resulted in a 24x speedup, and batches of 16 images (i.e. 80 descriptions) could be handled comfortably.

[1] Without fine-tuning, the output of the convolution net was meaningless, and the network failed to train.

3.1.2 Reinforce Categorical

Our refactoring takes advantage of the structure of the Oxford VGGnet, which is a very deep convolutional net where each convolutional layer has a kernel of size 3x3, and a stride of 1 - see Table 2.1 for a detailed outline. Every two or three convolution layers is topped by a max-pooling layer, and the number of kernels increases to 512 by the fifth set of convolutional layers.

This architecture means that the spatial structure of the image is preserved until the final fully connected layers; and we can think of the effective receptive fields of each unit in each kernel as increasing by 1 at every convolutional layer. This means that at the final convolutional layer before the fully-connected layers we have an output shape of 512x14x14 for an input image of 3x224x224 (we crop our 256x256 images as part of the preprocessing steps).

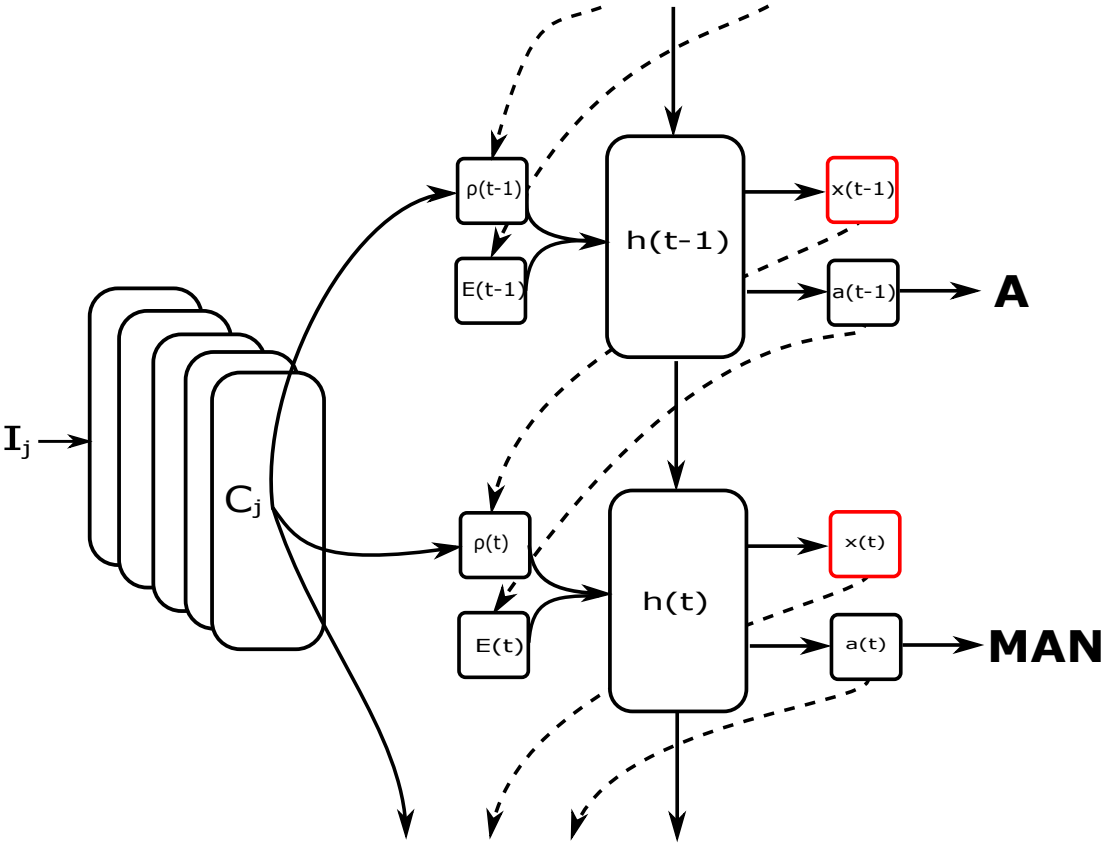
Each of the 512-dimensional vectors encodes a high-level feature description of a 27x27 effective receptive field to which they correspond. Sampling from these 196 feature vectors can then be thought of as hard attention over the image. The advantage of this is that we only have to forward images through the convolution net once per batch, and as our dataset contains 5 sentences per image, forwarding 16 images corresponds to a batch size of 80.

In short, we restructure by taking the feature vectors from the last convolutional layer of the 16 layer Oxford VGGnet, and then update the action module to sample one of these vectors conditioned on the state of the LSTM layer at each time step. See Figure 3.2.

3.1.2.1 'Glimpse' module

The glimpse module no longer extracts windows around the points of fixation. Passing each image through the Oxford VGGnet and taking the embeddings from the top convolution layer is effectively computing all possible glimpses at the start, which is tractable because we are now sampling from a discrete distribution, i.e. a finite number of fixations, unlike in the Gaussian case. Instead the glimpse module takes the one-hot

Figure 3.2: Reinforce Categorical Implementation; full arrows denote vector output/input within a timestep, dotted arrows denote vectors passing between timesteps, square boxes denote trainable weight layers, vertical rectangles denote LSTM units, overlapping rectangles denote convolutional layers, red squares denote that the output of the layer is stochastic, joining arrows denote vector concatenation, and letters denote the state of a layer or or LSTM unit at a particular timestep.



output from the action module and uses this to select one of the 196 vectors corresponding to a location on the image. This is then passed forward through a trainable weight layer and a ReLU nonlinearity, such that $\rho_t = \rho(C_j, x_{t-1})$. ρ_t is concatenated with the output of the embedding module, E_t, and input to the LSTM layer.

3.1.2.2 'Embedding' module

One of the problems noted with the first implementation (see 3.1.1.7) was that the net was not directly parameterising $p(a_t|x_{t-1}, x_{t-2}..., a_{t-1}, a_{t-2}, ..)$. To help with this, instead of solely feeding forward the output of the action module, x_{t-1}, to both input modules at timestep t, we forwarded both actions and words, a_{t-1}, to the next timestep.

Actions were forwarded to the glimpse module, as before, whilst words were passed through an embedding module.

The embedding module takes an index denoting the word output from the previous timestep t, and transforms it via a lookup table to an N dimensional vector, which represents that word in an embedding space of fixed dimension. Use of pretrained word embeddings (such as word2vec [42]) is common; these are the hidden activations of simple feedforward nets trained by word-context or context-word prediction tasks, with the idea being that if semantics can be determined by context, then the vector representations generated by a successful net should correspond to a semantically meaningful embedding space.

Initially we used pretrained embeddings by Collobert et al (2011) [10], which are only 50-dimensional, but which provided a boost to learning speed. We allowed the model to fine-tune these weights so that they would also be more closely aligned with the semantics of our dataset.

At training time, the embedding module is fed the target $t-1$ word directly. At evaluation time, the embedding module receives the word predicted by the classification module at the previous time step, either by sampling from the k most probable words, or by beam search, i.e. selecting that word which corresponds to the most probable sequence.

Later we altered the embedding matrix to 1000-dimensional (see Section 3.1.2.4), and used the output of the full VGGnet to compensate for the loss of low-dimensional gist which we used in the Gaussian architecture (3.1.1). This involved initialising the network by feeding the 1000-dimensional embedding from the top of the VGGnet (post fully-connected layers) through the recurrent module (see 3.1.2.4). Later time steps used 1000 dimensional embeddings to generate representative vectors of the word generated at the previous time step.

3.1.2.3 'Action' Module

The action module takes the hidden state of the LSTM as input and passes this through a trainable weight matrix of dimension 512x196 and then a Softmax layer, to produce p_t where $\sum_i p_{t,i} = 1$, which can be considered the parameters of the categorical distribution $f(x_t; p_t)$ from which we then sample to generate the next location x_t. The logarithm of the Categorical distribution is differentiable with respect to its parameters,

$$\frac{\partial \ln f(x_t; p_t)}{\partial p_{t,i}} = \begin{cases} \frac{1}{p_{t,i}}, & \text{if } x_t = i. \\ 0, & \text{otherwise.} \end{cases} \tag{3.1}$$

which is a necessary requirement for the reinforce algorithm to work (see Section 2.1.4).

To reduce the variance of REINFORCE, 50% of the time during training we set the output of the module to the expected value of the distribution, an approach also taken by the Xu paper [67]. Initial experiments showed that the optimisation had a tendency to collapse to very low entropy distributions (i.e the model would fixate on a particular point), which then impeded the model's ability to learn a tight relationship between output descriptions and input images, as well as being useless for our interest in the scan path behaviour of the model. To counteract this we added an entropy regularisation term to the loss function, $\lambda_e H(p_t)$, where λ_e is a hyperparameter set by cross-validation, and $H(p_t) = -\sum_i p_{t,i} \ln p_{t,i}$.

3.1.2.4 Gist

As we are passing the images forwards through the convolution net only once, we cannot extract a low-resolution window of most of the image to act as the gist. Without this, the model struggled to learn a relationship between the sampled image vectors and the descriptions. We tried two alternative approaches to give the model some information about the image as a whole. Both approaches involved the initialisation step of the model.

The first approach was to initialise the input to the recurrent layer from the glimpse module as the average of all 196 glimpse vectors, C_j, such that $\rho_1 = \frac{1}{196} \sum_i C_{j,i}$, where

i indexes the glimpse vector. This provided no real improvement on training loss, possibly because the averaging resulted in so much smoothing that all useful information was smoothed out.

The second approach was to add an extra initialisation step where we fed in the embeddings from the top of the OxfordVGG net (which are 1000 dimensional - see Table 2.1), as these have been pretrained for an object recognition task, and as such should capture the contents of the image, if nothing else. These image embeddings contained no information about object location, as the two fully connected layers at the top of the convolutional net obfuscate structural informaton. It therefore seemed more sensible to use this on initialisation as the output of the embedding module E_1. This meant that the embedding lookup table now used 1000 dimensional embeddings that the model had to learn, but produced a substantial increase in performance (see Figure 3.5).

3.1.3 Training

3.1.3.1 Data, pre-processing

To train the model initially we used the Flickr8k dataset [26], which is comprised of 8000 images each accompanied by five human-generated captions 'which provide clear descriptions of salient entities and events'. The data was split, using the standard split, into 6000 training images (30000 training captions), 1000 validation images (5000 captions), and a hold-out test set of 1000 images (5000 captions). The images were resized to 256x256 and centre cropped to 224x224 to pass through the Oxford VGGnet (the pretrained versions of which are restricted to inputs of this shape). In addition, we subtracted the pixel means from the images to zero centre the images.

The 40,000 captions were tokenised, and we replaced all tokens that occurred fewer than 3 times with a generic 'UNK' token, to prevent the model from having to learn weights corresponding to words for which it had few examples, and which could thus be considered outliers.

3.1.3.2 Net initialisation

Depending on our approach to gist, the net was initialised in one of two ways.

(1) The first fixation (x_0) was generated by passing a zero vector through the action module, as in Nicholas Leonard's implementation of Mnih et al [37]. The effect of this was just to sample from a categorical distribution parameterised by the biases of the weight matrix beneath. The first embedding output (E_1) was set to the output of the full Oxford VGGnet, the 1000-dimension embedding just before the softmax output layer.

(2) The first selected vector, ρ_1, was taken to be the average of the feature vectors, i.e. $\rho_1 = \frac{1}{196}\sum_i C_{j,i}$, and the first embedding E_1 was set to be that corresponding to the $M+1$ index (i.e. the null token).

3.1.3.3 Loss function

As discussed in 2.1.4, the main feature of nets with stochastic units is that they cannot be simply backpropagated through to update all of the weights with respect to some loss function, $-L$. Let us choose L such that we are maximising the probability of a target word sequence, $\boldsymbol{a} = \{a_1, a_2, ..., a_t, ..., a_S\}$ (where S is length of sequence), given some set of image feature vectors C_j. As we are optimising, we can take the log of this probability (as log is a monotonic function), which means we want to maximise

$$\begin{aligned} L &= \log p(\boldsymbol{a}|C_j) \\ &= \log \sum_x p(\boldsymbol{a}, x|C_j) \\ &= \log \sum_x p(x|C_j)p(\boldsymbol{a}|x, C_j) \end{aligned} \tag{3.2}$$

where x is a variable denoting the sequence of locations fixated, out of the 196 possible locations at each time step.
By Jensen's inequality [30],

$$\begin{aligned} L &= \log \sum_x p(x|C_j)p(\boldsymbol{a}|x, C_j) \\ &\geq \sum_x p(x|C_j) \log p(\boldsymbol{a}|x, C_j). \end{aligned} \tag{3.3}$$

This second term is a variational lower bound on the marginal log probability we want to maximise; and as it turns out, maximising the bound is much more tractable. We will call this term F, and note that it can be thought of as the expectation under the probability of fixations given image features, $p(x|C_j)$, of the log probability of the target word sequence given features and fixations, $\log p(\boldsymbol{a}|x,C_j)$.

$$\begin{aligned} F &= \sum_x p(x|C_j) \log p(\boldsymbol{a}|x,C_j) \\ &= \mathbb{E}_{p(x|..)}\big[\log p(\boldsymbol{a}|x,C_j)\big] \end{aligned} \tag{3.4}$$

Differentiating with respect to some model parameter θ, we find

$$\begin{aligned} \frac{\partial F}{\partial \theta} &= \sum_x \frac{\partial}{\partial \theta} p(x|C_j) \log p(\boldsymbol{a}|x,C_j) \\ &= \sum_x \Big[p(x|C_j) \frac{\partial}{\partial \theta} \log p(\boldsymbol{a}|x,C_j) + \log p(\boldsymbol{a}|x,C_j) \frac{\partial}{\partial \theta} p(x|C_j) \Big] \\ &= \sum_x \Big[p(x|C_j) \frac{\partial}{\partial \theta} \log p(\boldsymbol{a}|x,C_j) + p(x|C_j) \log p(\boldsymbol{a}|x,C_j) \frac{\partial}{\partial \theta} \log p(x|C_j) \Big] \\ &= \sum_x p(x|C_j) \Big[\frac{\partial}{\partial \theta} \log p(\boldsymbol{a}|x,C_j) + \log p(\boldsymbol{a}|x,C_j) \frac{\partial}{\partial \theta} \log p(x|C_j) \Big] \\ &= \mathbb{E}_{p(x|..)} \Big[\frac{\partial}{\partial \theta} \log p(\boldsymbol{a}|x,C_j) + \log p(\boldsymbol{a}|x,C_j) \frac{\partial}{\partial \theta} \log p(x|C_j) \Big] \end{aligned} \tag{3.5}$$

where the third line holds because $\frac{\partial}{\partial \theta} \log p(x|C_j) = \frac{1}{p(x|C_j)} \frac{\partial}{\partial \theta} p(x|C_j)$.

The form of (3.5) suggests that we could estimate this gradient by Monte Carlo; drawing samples from the distribution $p(x|C_j)$, and averaging the gradients for each sample. In practice this means that we can treat $p(x|C_j)$ as the categorical distribution parameterised by the action module, where each forward pass samples from this distribution. This aligns nicely with the process of stochastic gradient descent; forwarding each training item and sampling from the generated categorical distribution at each time step can be combined with the normal process of forwarding an item through the net and back-propagating the gradients from a loss function. As we are mini-batching, we update the gradients after N samples, as follows:

$$\frac{\partial F}{\partial \theta} \approx \frac{1}{N} \sum_{n=1}^{N} \Big[\frac{\partial}{\partial \theta} \log p(\boldsymbol{a}|x^n,C_j) + \log p(\boldsymbol{a}|x^n,C_j) \frac{\partial}{\partial \theta} \log p(x^n|C_j) \Big] \tag{3.6}$$

As discussed in Chapter 2, Monte Carlo estimators are unbiased but have high variance; as a method of density estimation, Monte Carlo can be extremely slow. One

way to reduce the variance is a formulation equivalent to the REINFORCE learning rule (Williams, 1992 [66]), where we reduce the reward (here $\log p(\boldsymbol{a}|x^n, C_j)$) by an estimation of the expected reward. This expected baseline reward was built into the net as a parameter which was trained using a MSE loss function and backpropagation alongside the rest of the model.

In addition, as discussed in Section 3.1.2.3, we added an entropy regularisation term to encourage the optimisation algorithm to find higher entropy solutions. Altogether, this gives us the following function for the gradient with respect to a particular parameter θ:

$$\frac{\partial F}{\partial \theta} \approx \frac{1}{N} \sum_{n=1}^{N} \Big[\frac{\partial}{\partial \theta} \log p(\boldsymbol{a}|x^n, C_j) + \lambda_r (\log p(\boldsymbol{a}|x^n, C_j) - b) \frac{\partial}{\partial \theta} \log p(x^n|C_j) + \lambda_e \frac{\partial H(p^n)}{\partial \theta} \Big] \tag{3.7}$$

This loss function ends up mirroring that in Xu et al [67], although we derived it separately during the development of our model. Note that to perform gradient descent, instead of maximising F, we minimise $-F$, so we update in the negative direction of (3.7). Note also that the above is a hybrid loss function: the first term applies to those θ to which we can backpropagate from the final loss without passing through the action module, i.e. parameters such that there is at least one path from final output to the parameter which does not pass through a stochastic unit (marked in red on Figure 3.2). The second and third terms are reward terms that only apply to those parameters 'downstream' of the stochastic units.

To implement this, we both backpropagate a negative log likelihood (NLL), and also broadcast the log likelihood (reduced by its baseline, b) as the reward for that batch to the stochastic modules. Backpropagation proceeds as normal, but each stochastic unit ignores incoming gradients and outputs the variance-reduced reward multiplied by λ_r and the derivative of the categorical with respect to its parameters (3.1).

In addition, it was easiest to implement the entropy term in the stochastic units, as the units store their inputs at each timestep for later use in BPTT. Thus the complete gradient passed back from each stochastic unit includes a $\lambda_e \frac{\partial H(p^n)}{\partial p}$ term.

3.2 Development

This section covers in more detail some of the design choices and investigations made during the development of the captioning model, and provides some motivation for why particular choices were made.

3.2.1 Dropout

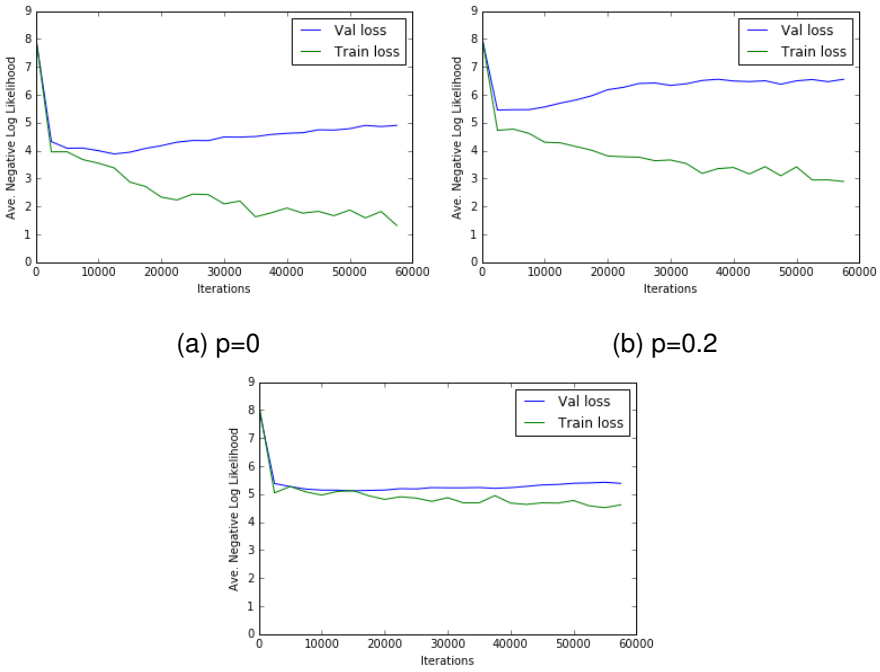
One of the primary problems faced when training any complex model is that of overfitting. This happens because we are using a finite training set as a series of examples of possible image-sentence mappings. However, the space of possible image-sentence mappings is obviously much greater than our training set, and so we are relying on the optimisation algorithm to infer rules that generalise well.

We want to find optima in the parameter space that correspond to true optima for all possible examples; however, because we can only judge the presence of optima from our limited training set, if we train for too long we will find that the validation loss begins to deviate drastically from the training loss, as the algorithm finds optima specific to our training examples that do not generalise well to other examples - i.e. statistical structures unique to our training set that are not present in the larger space of possible image-sentence mappings.

One solution to this is to use dropout [60]. At training time, this involves masking the activation of certain units (generally the input units to trainable layers) such that with some probability p, the activation of that unit is set to zero. The intuition is that this forces the net to find optima which are robust to noise in the input at different levels; to overcome potential partial activations, the net must distribute its characterisation of particular features more widely across its units. To (mis)appropriate a famous thought experiment, if there is only one unit that activates when the net is shown an image of a grandmother, and that unit is set to zero with probability $p = 0.5$, then half the time the net will fail to recognize grandmothers.

These distributed representations seem to generalise better. This is because they can

ignore small specific differences between inputs: and so images which exhibit particular features, but which have not been seen during training, should still produce patterns of activation that are recognised by higher levels of the network as the appropriate features.



(a) p=0 (b) p=0.2

(c) p=0.5

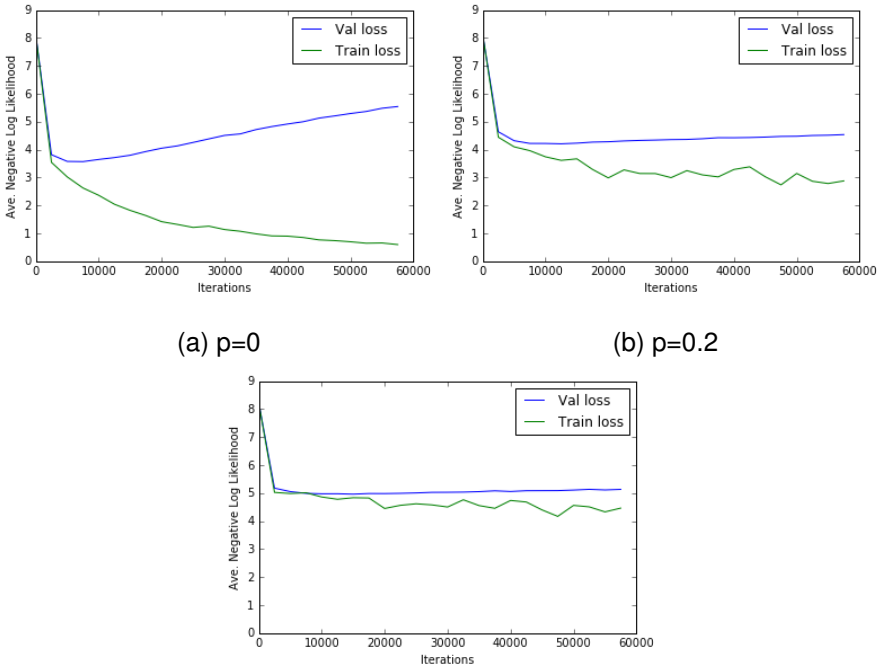
Figure 3.3: Effect of Dropout during training on validation and training loss

3.2.2 Entropy

We also experimented with a (Shannon [54]) entropy term, $\lambda_e \frac{\partial H(p^n)}{\partial p}$, on the parameters of the categorical distribution. Here, $H(p^n) = -\sum_i p_i^n \ln p_i^n$, which can be thought of as the expected information to be contained in any particular event. Highly probable events contain little information, and so high entropy corresponds to a more uniform distribution.

We found (see Figure 3.4) that adding this term to the loss produced an overall improvement in average negative log likelihood (the objective we were attempting to minimise), and as above (see Figure 3.3), dropout reduced overfitting, but did so at a cost of performance: whilst training and validation loss are much closer, the best with dropout is always higher than the best without - simply put, dropout makes the task

harder.



(a) p=0

(b) p=0.2

(c) p=0.5

Figure 3.4: Entropy with dropout during training

As well as a slight improvement in minimum NLL, and the avoidance of initial poor local optima by the REINFORCE algorithm collapsing to low entropy distributions, the advantage of the entropy term was that it encouraged the model to explore all of the space: by encouraging high entropy we encourage the distribution to be more uniform, it makes the model more likely to sample from less probable regions.

3.2.3 Soft Attention

When analysing the performance of our model (See Chapter 4), one of the things we will investigate is the underlying categorical distribution from which we sample the next location to be fixated, x_t. This is interesting because we can see directly the extent to which the model is judging particular parts of the image to be salient to the generation process at that time step.

As a comparison, we can also train the model without the sampling step; instead we can replace the 1-hot x_t (denoting the hard-sampled location) with the output of the preceding softmax p_t (see 3.1.2.3) such that ρ_t is a mixture of feature vectors C_j:

$$\rho_{t,j} = \sum_i p_{t-1,i} C_{j,i} \tag{3.8}$$

where $C_{j,i}$ denotes the i^{th} 512-dimensional feature vector of the j^{th} image, and $p_{t-1,i}$ are the i^{th} elements of the output of the action module at the previous timestep. ρ_t is thus 512-dimensional. This version of the model is fully trainable by back-propagation (as we no longer have any stochastic units to deal with), and we can think of it as a 'soft' attention mechanism; the model can now receive contributions from all locations on the image at a time-step, but can weight the information from more relevant locations more heavily.

We would expect the distributions learned by the soft attention model to be more diffuse that those learned by the hard attention model; it can receive more information about the image as a whole, and does not need to commit as strongly to particular locations. The less heavily weighted location vectors can be thought of as contributing to the overall gist. We might also expect the soft model to perform better than the hard model, as its optimisation task is arguably easier. This is because it can update all of the weights of the action module with every backpropagation, whereas the hard attention model will only update those weights corresponding to locations it has actually sampled (see 3.1), as it has no reward information about the other locations. This means that it must sample for longer to get an estimate of the underlying optimal policy (i.e. the p_t that would maximise its reward at time t).

As it turned out, the soft attention model did not perform substantially better than most of the models with 50-dimensional embeddings. We suspect this was due to the fact that the model was not initialised using any form of gist, unlike the hard attention models (see Figure 3.5). The heatmaps generated by the weights were not particularly informative (in either case, although we discuss those of the hard attention model in more detail in Chapter 4), and so we did not pursue a substantial comparative analysis between the two, preferring to focus on the hard attention model directly in the time available. Such an analysis could be the focus of future work.

3.2.4 Development Comparison

To finish this section we present the results from the development of the model; first by comparison with the models at various stages of development, and secondly the hyper-parameter fine-tuning of the best architecture.

3.2.4.1 Comparing architectures.

Table 3.1: Test Model losses for various models. Dropout $p = 0.2$.

Model	Minimum Mean Loss (NLL)
Baseline	3.9135
Baseline + Dropout	5.4534
Baseline + Entropy	3.5699
Baseline + Dropout + Entropy	4.2078
Soft Attention	4.4014
1000-dim Embeddings + Dropout + Entropy	**3.2569**

Table 3.1 shows the minimum mean test loss for the different models. We used early stopping to save the best version of each model before it began to overfit - even with some dropout, this almost always happened. All of the models seem to plateau quite rapidly, which might be an indication of the models being too simple for the task. One way to combat this would be to increase the number of hidden layers, or the number of units in each hidden layer. Both improvements would increase training time, however, and so are left to potential future explorations.

Figure 3.5 shows the validation losses for training each of the variants of our model: the baseline Reinforce Categorical model with 50-dimensional word embeddings, the baseline with dropout, with entropy regularisation and with both, the soft attention model (see Section 3.2.3), and finally the model with 1000-dimensional embedding initialised using the top layer output of the Oxford VGGnet. This latter also includes entropy regularisation and dropout $p = 0.2$.

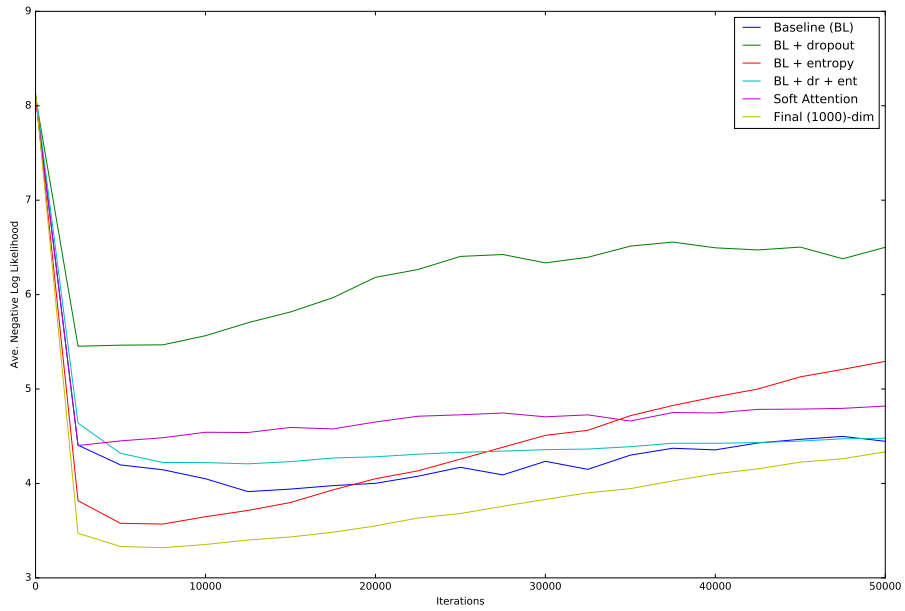


Figure 3.5: Model training validation losses. Note increase due to overfitting.

3.2.4.2 Fine-tuning 1000-dimensional embedding model

To fine-tune the performance of the model we conducted a 10-fold cross validation grid search across possible hyper-parameter settings for the model. The three main hyper-parameters we investigated were those which contributed directly to the loss function, namely λ_e, the entropy scale, λ_r, the reward scale, and η, the overall learning rate.

Table 3.2 shows the results of the grid search between λ_e and λ_r - due to prohibitive time costs, we were unable to do a full 3-dimensional grid search across all three hyper-parameters. Things to note from the table are that smaller appears to be better; given more time, we would have liked to extend the grid search in this direction. However the loss also seems reasonably robust to changes in λ_e and λ_r, and so we concluded that this was not a priority.

Table 3.2: Grid search to compare effect of λ_e and λ_r on model performance. $\eta = 0.0005$ Minimum Negative Log Likelihood reported from cross validation.

		Entropy lambda, λ_e			
		0.05	0.2	0.5	1.0
	0.05	**3.2504**	3.3378	3.2950	3.3294
Reward	0.2	3.2628	3.3561	3.2845	3.3268
Scale, λ_r	0.5	3.2601	3.3519	3.2781	3.3119
	1.0	3.2580	3.3570	3.2798	3.3241

Table 3.3 shows the results of cross-validation with a changing learning rate, η. We used the rate of 0.0005 for subsequent training, as this gives the best result. The other hyperparameters of the Adam algorithm, which we used to train the 1000-dimensional final model, were left at their recommended values (see Kingma and Ba (2014)), as these were derived empirically for deep networks performing similar tasks [32].

Table 3.3: Grid search to choose best overall learning rate η. Note for $\eta = 0.05$ loss exploded, so no value given.

Learning Rate	**Minimum NLL**
0.05	-
0.005	3.5537
0.001	3.2820
0.0005	**3.2574**
0.00005	3.6057

3.3 Comparison to Human Data

Once we have a network that generates image descriptions using a hard attention mechanism, we want to compare the scan paths generated by the network to those generated by human participants in an image description experiment, to see if the net's learned policy is human-plausible. This section will describe the ways in which comparisons will be made, and detail the nature of the human data that we will be using.

3.3.1 Human Dataset

Our human dataset is that presented in Coco and Keller's (2010) language production experiment [8]. 24 participants were eye-tracked whilst describing some subset of 48 images of 24 different scenes; each with a cluttered version and a minimal version.

The scenes were drawn from six different scenarios (e.g. bedroom, entrance, office etc), and participants were given a cue word in written form for 750ms before the presentation of each scene. The cue word referred to an object in the scene, although it was ambiguous with respect to the scene (it might refer to more than one particular object).

Participant's scan paths were recorded and divided into three distinct stages: planning, encoding and production (see Figure 3.5). For clarity of comparison in the model developed here, we are only concerned with the production phase: where the model looks whilst generating a description compared to where humans look whilst generating a description.

Figure 3.6: Example human scan path on image description task (Coco & Keller, 2012). Scan split into three distinct phases: planning (blue), encoding (green), and production (red). We are concerned with modelling the third phase. Images have been reshaped to 224x224, which is how they were presented to the model, and allows for direct comparison with images from the Flickr8k dataset. Fixation co-ordinates have been transformed accordingly.

Building a model that incorporates both planning and encoding phase is a natural extension to this project; it would involve allowing the network to sample from the image

prior to generating the rewarded descriptions. For this investigation, however, we discarded the scan path information from the first two phases, and only used that from the production phase for comparison. We also discarded scan paths with fewer than 5 fixations in the production phase, as short scanpaths give meaningless results in grid-based ranking metrics extracted from attention maps (see Section 4.2). This gives us 441 scan paths and corresponding descriptions for 48 images.

Note that there is a nice correspondence between the fact that participants were given a cue word and the initialisation of our final architecture with the 1000-dimensional top layer output of the Oxford VGGnet, which is trained on an object recognition task [52]. The initialisation vector provides a weighting over objects present in the image. The cue word is a harder version of this, but both serve to encode the presence (or expected presence) of certain objects into the gist.

To generate the model's fixations, we treated all 48 images as test images, resized them to 224x224 and used the trained model to generate a sequence of fixations and a description for each.

3.3.2 Scanpath Representations

There are many different ways to measure scanpath similarity; in the interests of time and clarity, we restrict ourselves primarily to dispersion-based measures, with some mention of scanpath events in the qualitative portion of our evaluation. Dispersion measures are those metrics that take into account only the locations fixated, and ignore temporal relationships (see Chapter 4 for an explanation of dispersion measures used). This is the simplest question we can ask of our model: **does it look in the same sorts of places as humans whilst performing the same task**? If the answer is no, then we can immediately discard our general hypothesis (see Section 1.2) and conclude that deep neural networks with attention do not attend to images in a cognitively plausible fashion.

To represent the spatial distribution of fixations (i.e. dispersion), we converted each scanpath into a Gaussian attention map.

3.3.2.1 Attention maps

Gaussian attention maps are useful for both visualisation and quantitative analysis. They excel at visualisation because they turn what can be a clutter of fixations and connection lines into a clear heatmap highlighting those areas of an image which attracted the most attention. They are useful for mathematical analysis because they provide a continuous surface above the image, which we can think of as a maximum entropy model, for some choice of variance σ^2 [51], of the probability of attention across the whole image constrained by the known fixations.

This probability need not be normalised, but can be a useful way to characterise what we know from the fixations in a principled fashion. The downside of the attention map is that it is very sensitive to our choice of σ^2. Too large, and we run the risk of over-smoothing. Too small, and we fail to smooth out any noise in our data; precise locations become vitally important. An attention map $A(x,y)$ is generated using the following formula:

$$A(x,y) = \sum_i \exp\Big(-\frac{(x-x_i)^2+(y-y_i)^2}{2\sigma^2}\Big) \tag{3.9}$$

where (x_i, y_i) are the co-ordinates of the i^{th} fixation.

It is suggested in Holmqvist et al (2011) [28] that the variance should be set so that at the half-height of the Gaussian peak corresponding to a particular fixation, the width of the peak should approximate the size of the foveal projection on the image, which is approximately $2^\circ$of visual angle. As participants for the Coco and Keller (2010) experiment sat around 60cm from the screen, we can compute this using basic trigonometry, and find that it corresponds to foveal projections of approximately $1/25^{th}$ the size of the image.

An important point is that the receptive fields corresponding to the glimpse vectors generated by the convolutional part of our model are quite a bit larger than this: at 27x27 pixels they correspond to approximately $1/8^{th}$ the size of the image. However the edges of these receptive fields will contribute less to the glimpse overall vector (due to the overlapping 3x3 kernels with stride 1 at each of the 13 convolutional levels) than the centre. This means we can think of the centre of the receptive field as that

region corresponding to the foveated region, and so σ^2 is kept the same for both model generated and human generated attention maps.

3.3.2.2 Centre bias baseline

To properly compare model dispersion with human dispersion, we need a null hypothesis. For this we use the centre bias baseline suggested by Clarke and Tatler (2014) [7], which mimics the content-independent tendency of observers to make fixations towards the centre of an image during scene viewing.

The centre bias baseline for an image of shape $[-1,1]$x$[-a,a]$, where a is the aspect ratio (0.75 for the Coco and Keller dataset) consists of a zero mean Gaussian (zero being the centre of the image) with variance given by

$$\begin{bmatrix} \sigma^2 & 0 \\ 0 & \nu\sigma^2 \end{bmatrix} \tag{3.10}$$

where σ^2 and ν are set to 0.22 and 0.45 respectively, values recommended by Clarke and Tatler from empirical evaluation of 10 recent eye-tracking datasets. ν captures the horizontal bias present in human eye-tracking data. We treated this Gaussian as an attention map for quantitative comparisons to those maps generated by our model, and from the human data.

Chapter 4

Results and Evaluation

4.1 Model Performance

The first stated goal of this thesis was to implement a neural-network based model of the type described in Mnih et al (2014) [44], for the cross-modal task of image description (see Section 1.2). Whilst the development process led us towards a different sort of implementation (see Section 3.1.2), we have produced a model that generates text when presented with an image, and does so by sampling a sequence of local glimpses of the image. This section examines how well the model performs on the test set of the Flickr8k dataset it was trained on, to decide whether this goal has been achieved.

4.1.1 BLEU, Comparison with State of the Art

Image description literature generally co-opts algorithms from the machine translation literature to estimate the quality of the descriptions generated during image description tasks. These algorithms score the similarity of a candidate sentence to a group of target sentences. One popular algorithm is BLEU (Bilingual Evaluation Understudy [47]), which was one of the first to achieve high correlation with human assessments of candidate sentence quality [11].

BLEU utilises a modified form of precision, where instead of simply computing the proportion of words in the candidate translation that appear in any reference transla-

tion,

$$p_n = \frac{\sum_{n\text{-}gram \in C} Count(n\text{-}gram)}{\sum_{n\text{-}gram' \in C} Count(n\text{-}gram')} \tag{4.1}$$

we clip the n-gram count such that it can be, at maximum, the largest count of that n-gram for any one reference translation. For some n-gram, the modified precision is then given as:

$$p_n = \frac{\sum_{n\text{-}gram \in C} Count_{clipped}(n\text{-}gram)}{\sum_{n\text{-}gram' \in C} Count(n\text{-}gram')} \tag{4.2}$$

For our purposes, we simply use descriptions instead of translations. Table 4.1 shows how our model performs against the current state of the art in image description. Two additional baselines are provided as reference: the BLEU scores for 'descriptions' generated by sampling uniformly from the vocabulary, and those scores for 'descriptions' generated by sampling from the vocabulary according to the distribution of words in the target training sentences. This is to make sure the model is not generating descriptions using words common to the dataset without utilising image information.

	BLEU			
Model	**B-1**	**B-2**	**B-3**	**B-4**
Random baseline	0.9	0.0	0.0	0.0
Random distributed baseline	28.3	5.8	0.0	0.0
Soft Attention	43.3	26.4	15.0	8.5
1000-dim embeddings	44.1	24.3	13.1	7.2
Google NIC(Vinyals et al., 2014 [64])	63	41	27	-
Log Bilinear (Kiros et al., 2014 [33])	65.6	42.4	27.7	17.7
Show, Attend, Tell (Xu et al., 2015 [67])	67	45.7	31.4	21.3

Table 4.1: Bleu scores (unigram, bigram, 3-gram and 4-gram) on Flickr8k achieved by our model compared to those achieved by current state-of-the-art in image captioning. Also provided for reference are the performance of a random baseline of sentences drawn uniformly from the vocabulary, and a random baseline of sentences drawn from a distribution matching the distribution of words across the target descriptions. Our models are bolded.

We have also provided the scores of our Soft attention model, to give an idea of the relationship between NLL and BLEU. The difference in minimum NLL between our

Soft and Hard Attention models was 1.1445 (see Table 3.1); the difference in 1-gram BLEU score is 0.9, and the Soft Attention model actually outperforms the Hard Attention model on 2-, 3-, and 4-gram BLEU metrics. This suggests that the large difference in BLEU score between our model and the state of the art would also correspond to a large difference in NLL; and so can be considered a substantive difference in performance.

In some ways, this is a little disappointing. Whilst the above models were designed to perform state of the art image description, and our model was developed with cognitive plausibility in mind, it would be nice to be closer in terms of performance. The disparity means that any conclusions we draw about the cognitive plausibility of our model can only be applied tentatively (at best) to the class of attention-based models (of the above, Xu et al [67] is the only example); if our model does not behave in a human-plausible fashion, it may still be the case that better performing models are; and we can draw no conclusions about whether their improved performance is due to more human-like attentional policies, or better optimisation algorithms, or superior architectures.

We do substantially beat the random baselines; which suggests that the model is doing *something*. We have implemented an image description model, even if it is not a very good one. In what follows, we will attempt the second goal of this thesis: investigate how similar the model's attention is to that of humans.

We will perform a qualitative analysis of the performance of the model on the Flick8k test set, to see what the model struggles with, and whether its behaviour is plausible. Following this we will compare the attention of the model over the human test set images to that of humans, and conclude with a brief discussion.

4.1.2 Qualitative Analysis

For our initial hypothesis (see Section 1.2) to hold, we want to see (1) that the model displays plausible examples of scanpath events, and (2) that the dispersion of fixations generated by the model correlates well with that of humans. First, however, given the

disappointing performance of the model when compared with state of the art, it is helpful to examine the model in action, to see if we can understand what it does well, and what it does not.

We will begin with an example of the model performing reasonably well.

Figure 4.1: Generated Caption: **a black dog running through the snow .** Figures L-R are (a) Fixations with connecting scan path, (b) Average probability of fixations, $\mathbb{E}[p(\boldsymbol{x})]$ during production, (c) Gaussian heatmap of attention generated from fixations.

Figure 4.1 (a) shows the fixations generated by the model during sentence production, connected by arrows to indicate the direction of the scan path.

Figure 4.1 (b) shows the image overlaid by the expected probability of fixation $\mathbb{E}[p(\boldsymbol{x})]$. This has been extracted from the action module during production at each time-step, and is included mostly for interest; it seems that the process of averaging across all time steps removes too much information, as there is little correspondence between the attention map generated by actual fixations and the heatmap of the underlying distribution. Note that the distribution is more finely grained than one might expect from a categorical distribution on a variable with only 196 outcomes; this is because the receptive fields from which we sample overlap, and so the probability of a particular pixel of the image contributing to the sampled glimpse vector is a combination of the sample probability of each of the glimpse vectors to which it contributes.

Figure 4.1 (c) shows the Gaussian attention map derived from the fixations. It is these maps we will use for our quantative comparison with human data in Section 4.2; for a description of how they are created, see Section 3.3.2.1.

Figure 4.1 seems to show the model working quite well; it correctly identifies a black (or mostly black) dog, and fixates primarily on the dog, with occasional saccades to either side to establish the context. That it gets the context wrong is understandable: the water is very light, almost white, and the ripples that distinguish it from a snowfield for the human observer are quite subtle (and the main ones, in the top right corner, are never directly fixated). Whether the dog is actually running is ambiguous - the human descriptions give the verb as 'walking', 'swimming', 'swimming', 'swims' and 'standing' - although we could argue that given the assumed context of snow, 'swimming' would be a greater error as far as the model is concerned.

One thing of note that is clearer from the attention map (4.1(c)), however, is that the dog does not quite seem to be the object of fixation; rather, it is the place where the edge of the dog meets the water.

Figure 4.2: Generated Caption: **a brown dog is running through the grass .** Figures L-R are (a) Fixations with connecting scan path, (b) Average probability of fixations, $\mathbb{E}[p(\boldsymbol{x})]$ during production, (c) Gaussian heatmap of attention generated from fixations.

Another dog image confirms this. Figure 4.2 shows an image that is miss-captioned; the model does not recognise that there are two dogs. However, it is more interesting because we can see that our suspicion is correct; the model is fixating regions which contain sharp contrasts (edges, changes of colour etc), rather than high-level features such as objects. What this suggests is that the model has learned a policy of fixating low level, but potentially salient features. If by doing so it happens upon high level features that are small enough to be fixated in their entirety, these may then appear in the description. For example in Figure 4.3, areas that capture the model's attention are

those which contain strong edges (such as the intersection of several hats on the right hand side of the image), and sharp changes of pixel intensity when compared with the rest of the image. These locations happen to correspond to high-level objects (but reasonably small) that appear in the generated description 'a group of men wearing a hat and a striped shirt'. It is also interesting to note that the crowd is being fixated when the word 'group' is output by the model.

Figure 4.3: Generated Caption: **a group of men wearing a hat and a striped shirt.** Figures L-R are (a) Fixations with connecting scan path, (b) Average probability of fixations, $\mathbb{E}[p(\boldsymbol{x})]$ during production, (c) Gaussian heatmap of attention generated from fixations.

Selecting edges and areas of transition as salient is quite reasonable behaviour; it has been shown that the retina and low levels of the visual cortex act as edge detectors and low level feature detectors [4], and in the days of computer vision before the advent of convolutional networks, hand crafted filters such as the LoG (Laplacian of Gaussian [22]) filter were built for robust edge detection to aid downstream image processing tasks.

Intuitively, we can think of these sorts of patches as possessing high entropy. Normally, an assumption of local consistency is a valid one to make about the relationship of adjacent patches of a visual image: we can guess that pixels near to one another will be about the same intensity and colour as each other (such an assumption underlies most image compression techniques [59][17]). Areas where there are lots of changes in intensity, such as edges, contain a lot of information - we cannot make an assumption of local consistency. As such these high entropy regions should receive a larger proportion of our attention if we are to form a correct judgement about an image.

However, it is not clear to what extent our model has learned this policy by itself. The Oxford VGGnet, which was trained to detect objects in realistic images [52], is very likely to have already learned that high-entropy regions make good low level features; as such, passing each image through it will do a lot of our language model's work for it - rather than developing a plausible policy, it merely needs to learn to attend to areas picked out for it by the convolutional network. As such, the cognitive plausibility derives more from the similarities between the low levels of the visual system and convolutional networks than from a high-level policy learned with respect to the task of image description.

In addition to these low level features, the model also leans heavily on common linguistic structures in the dataset. 22% of our training descriptions include the word 'dog' or 'dogs'; it is no surprise that the network is reasonable at spotting whether there is a dog in an image, particularly given it is also provided with a gist from an object detector.

Images which are dissimilar to most others in the training set leave it very confused; Figure 4.4, for example, which shows 'a crowd of people on the sidelines of an atv race' according to one human annotator is described by the model as 'a black dog running in the water to catch a ball', which is as interesting for the fact that it has clearly discarded most of the gist information (we can assume none of the objects mentioned were particularly highly weighted by the Oxford VGGnet), as for the fact it is actually a reasonable sentence, if it were presented alongside an appropriate image.

To conclude our qualitative analysis, it seems that the model is trying to do something reasonable. It has learned a policy of fixating high entropy regions in the images, and then combining this information with high level gist information, and the learned statistical structure of the target sentences, to output captions which are sometimes quite good, and sometimes quite bad.

It may be that the language model is too simple to learn a relationship between low-level entropy patches and high-level features such as objects; as discussed in Section 3.2.4.1, increasing the number of hidden layers might help with this problem. There

Figure 4.4: Generated Caption: **a black dog running in the water to catch a ball.** Figures L-R are (a) Fixations with connecting scan path, (b) Average probability of fixations, $\mathbb{E}[p(\boldsymbol{x})]$ during production, (c) Gaussian heatmap of attention generated from fixations.

is some evidence that high-level objects can be detected by the model if presented in silhouette, i.e. with minimal additional clutter. For example the model's description of Figure 4.5 is 'a man in a black wetsuit is snowboarding through the snow covered mountain.', which is a very good attempt, given the 1-gram probabilities of training sentences including 'wetsuit', 'snowboarding', 'mountain' and 'snow' are 0.0016, 0.0008, 0.0182 and 0.0304 respectively. The clarity of the silhouette is such that even the average distribution over fixations $\mathbb{E}[p(\boldsymbol{x})]$ (Figure 4.5(b)) maps reasonably well to the edges of the snowboarder.

Figure 4.5: Generated Caption: **a man in a black wetsuit is snowboarding through the snow covered mountain.** Figures L-R are (a) Fixations with connecting scan path, (b) Average probability of fixations, $\mathbb{E}[p(\boldsymbol{x})]$ during production, (c) Gaussian heatmap of attention generated from fixations.

4.1.3 Plausibility of Tracking

Visually inspecting test images and their corresponding heatmaps is helpful, but more statistical analyses can serve to support the conclusions we have drawn. One major feature of human scanpaths, as discussed in Section 3.3.2.2, is that over many participants over many images, they display a strong centre bias [7]: there are on average more fixations near to the centre of the image than the edges. To test to see whether our model displays the same sort of characteristics, we averaged the parameters of the categorical distribution from which fixations were sampled across all test images and across all timesteps. This gives us a much closer approximation to $\mathbb{E}[p(\boldsymbol{x})]$ than across a single image. The resultant heatmap is shown in Figure 4.6:

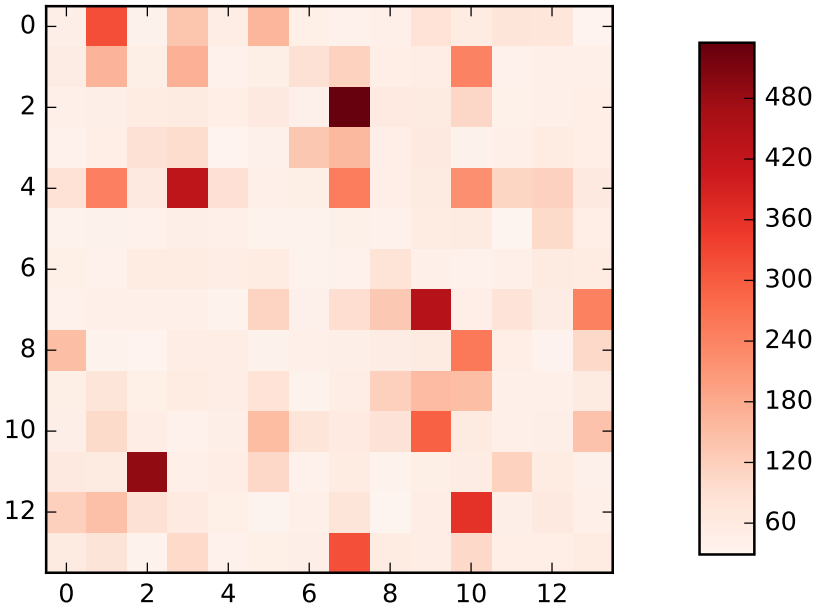


Figure 4.6: Distribution across glimpse vectors. Heatmap denotes number of fixations per 1000 images.

Note that we have not here provided the representation with respect to the pixels, but rather the glimpse vectors C_j. This allows for a clearer picture of the actual equilibrium sampling distribution of the model.

Interestingly, the model seems to end up with quite low entropy distributions despite

the entropy regularisation term (see Section 3.2.2). Additionally, the specific distribution is not structure-specific: different training runs of the same model find different (spiky) distributions. This suggests that the distribution corresponds to optima particular to that run. Disappointingly, the model shows no signs of developing a sampling distribution with central bias as one of its features. In Figure 4.6, as is typical of each training run, the spread of high-probability points is quite random. This is not very human-plausible behaviour.

4.1.4 Inhibition of Return

A second, and well documented (see, eg Posner et al (1985) [48]), feature of scanpaths is inhibition of return. Inhibition of return is the observation that attention does not directly return to the same location, within a small temporal window. The window is often quite small; it has been shown that when viewing complex scenes, the eye can return to previous fixation locations after a few seconds [58]. However, in general, some mechanism prevents immediate refixations. This has been argued to be so as to prevent inefficient returns to areas already inspected [34]. This is a temporal event, and to analyse it we will make use of the ordered nature of our fixation sequences.

During our qualitative analysis we noticed that the model tends not to fixate the same location twice. This might be due to it developing something akin to inhibition of return, or simply because there is sufficient randomness in the 196 locations, and the distribution is sufficiently flat, that it is unlikely to repeat the same location twice. To test this, for every fixated location we plotted the relative probabilities of that location being fixated at $(t-3, t-2, ..., t+2, t+3)$, see Figure 4.7.

If the model were learning a policy of subsequent avoidance, we would hope to see a substantial drop in probability post fixation ($t > 0$). Figure 4.7 shows that this is not the case; the decrease post fixation is no greater than the increase pre-fixation: all we can really conclude from the figure is that a point has a higher probability of being fixated on the time-step when it is fixated, which is trivial.

Whilst it would help the model to learn such a policy, it is probable that given the large number of locations to sample from, the model is unlikely to encounter such a repeti-

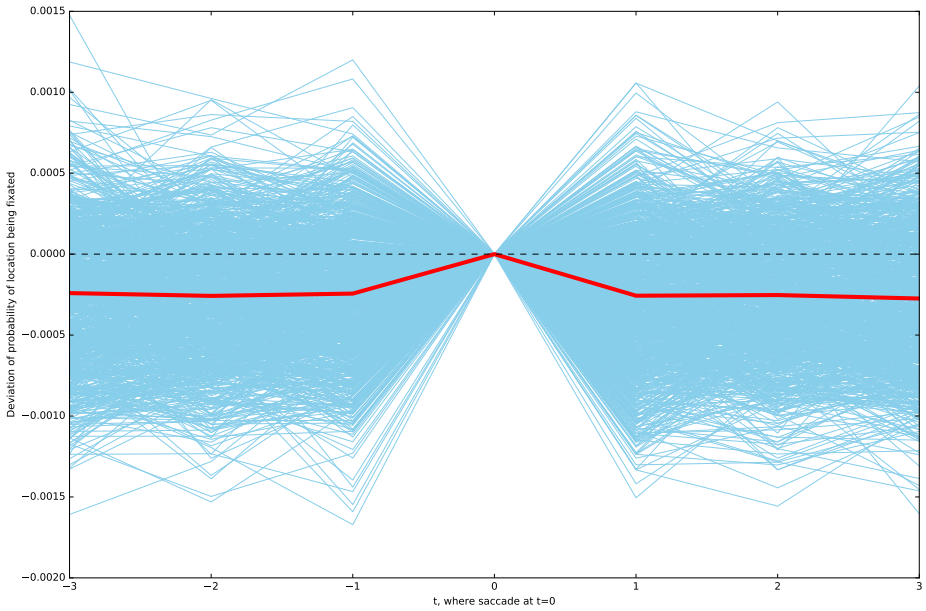


Figure 4.7: Profile of relative probabilities of fixation either side of actual fixation (at $t = 0$). Red line is average relative probability, black dotted line is zero.

tion regularly during training, and so has no cause to learn that this is bad. We can thus conclude that the model is not explicitly performing human-like inhibition of return.

4.2 Comparison to Human Data

For direct comparison with human data we used two dispersion measures: Rank-correlation and the area under an ROC curve. Given the non-dynamic nature of the network's fixation mechanism, whereby it saccades instantaneously once per timestep (i.e. once per generated word), this seemed the fairest modality of comparison. We compared the model using both the model's actual fixations, and the per image average of the distribution the model used to generate the fixations. The latter was included because in one sense it is the 'ground truth' of the model; whilst averaging across timesteps smooths some specific information from the resultant heatmap, it more directly represents the model's actual underlying attention mechanism.

As well as intra-human comparison, we included the centre bias baseline discussed in Section 3.3.2.2. For this comparison the specific version of the general hypothesis outlined in 1.2 is that **the spatial distribution of fixations produced by the model should be positively correlated with that of humans, and should be more strongly correlated with that of humans than the centre bias baseline**.

Table 4.2 details the respective scores under the two measures, along with their standard errors.

	Rank-correlation	**Area under ROC Curve**
Human ave.	**0.7848** $\pm$ 0.0018	0.5945 $\pm$ 0.002
Centre-bias	0.5817 $\pm$ 0.0071	0.5976 $\pm$ 0.002
Model (fixations)	0.2393 $\pm$ 0.0058	**0.5992** $\pm$ 0.002
Model (distributions)	0.1183 $\pm$ 0.0045	0.5461 $\pm$ 0.002

Table 4.2: Table of Average Rank-Correlation scores and area under ROC curve for 1000-dimensional embedding model compared with human average and centre bias baseline.

4.2.0.1 Rank-correlation

To compute a rank-correlation score we first converted each attention map into a 14x14 grid, and ranked each grid square according to their average spatial attention. For some pair of attention maps, x and y, for any pair of individual squares i and j, we can compute a rank correlation if we provide some x-score a_{ij} and some y-score b_{ij} such that they are anti-symmetric and form some measure of comparison [14]. The generalised correlation coefficient Γ is given by

$$\Gamma = \frac{\sum_{i,j=1}^{n} a_{ij} b_{ij}}{\sqrt{\sum_{i,j=1}^{n} a_{ij}^2 \sum_{i,j=1}^{n} b_{ij}^2}} \tag{4.3}$$

If we take the scores to be simply the difference in rank such that $a_{ij} = Rank_i^a - Rank_j^a$, after some algebra we extract Spearman's rank correlation coefficient, Γ_S:

$$\Gamma_S = 1 - \frac{6 \sum d_i^2}{n(n^2-1)} \tag{4.4}$$

where d_i is the difference in rank between x and y for the i^{th} square on the grid. Figure 4.8 shows the distribution of pairwise Γ_S for each pair human-human, human-baseline, human-model (fixations) and human-model (underlying distribution).

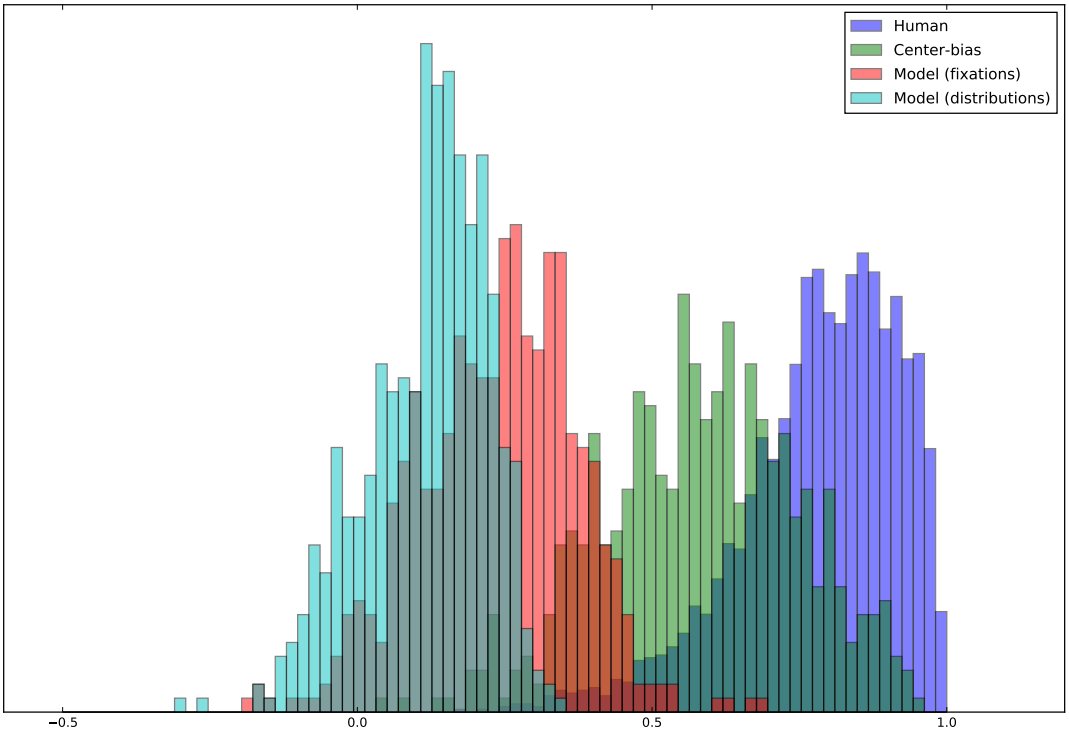


Figure 4.8: Histogram of pairwise rank correlations (horizontal-axis) for attention maps. Note that human-human has been rescaled, as there are substantially more human-human pairs that all others.

We can see that our hypothesis is partially incorrect. Whilst the vast majority of correlations with human attention maps of both the model's fixation-derived attention maps and its average distributions are positive, both are comfortably outperformed by the centre bias baseline. This is strong evidence that our model does not enact a human-like policy of attention.

Interestingly, this does confirm a result in a paper by Das et al (2016) [15], which was submitted for review during the writing of this thesis. Das et al compare the dispersion of attention in deep networks with that of humans for a Visual Question Answering task, as opposed to our use of an image description task. However, their results are similar: whilst there is some positive correlation of spatial attention of deep networks with that of humans, it is less than that between a saliency baseline [31] and

humans. To get a positive result, Das et al note that their saliency baseline is strongly correlated with centre bias, and remove all human attention maps with a positive rank-correlation with centre-bias. For the stated objective of this thesis, however, this seems to somewhat miss the point; if centre bias is a substantial feature of human visual attention, we should not simply discard it if our models do not learn a similar feature.

4.2.0.2 Area under ROC curve

ROC (Receiver Operating Characteristic) curves are commonly used as a way to measure the performance of a binary classifier as its discrimination threshold is varied [41]. Whilst our attention maps are not classifiers, it is quite simple to make them one. We grouped all human fixations into one set of locations, and denoted them as the positive instances. We then sampled points uniformly across the image's size, and denoted these as the negative instances.

The ROC curve plots the True Positive Rate, i.e. the proportion of correctly classified positive instances, against the False Positive Rate, i.e. the proportion of negative instances incorrectly classified, as the classification threshold is changed. In this case, we normalised the maximum height of the attention maps to 1, and lowered the threshold from 1 to 0 in increments of 0.01. An instance is considered classified as positive for a particular threshold th if it lies within the isoline on the attention map corresponding to a constant th.

We would expect the area under the ROC curves to be greater than 0.5 for a reasonable classifier; the closer the area to 1, the better the classifier. As the division boundary between the positive and negative instances is highly non-linear, due to our sampling strategy, we would not expect very high scores, but we should still be able to compare models.

Figure 4.9 shows the ROC curves for each of the four attention map types. To get a single curve for each of the classes of comparison, we average the TPR and FPR at each threshold increment. Interestingly, whilst the ROC curves of all four types of attention map define an area greater than 0.5, there is little to choose between intra-human, the model's fixations, and the centre bias baseline. This is unlikely to be a confirmation

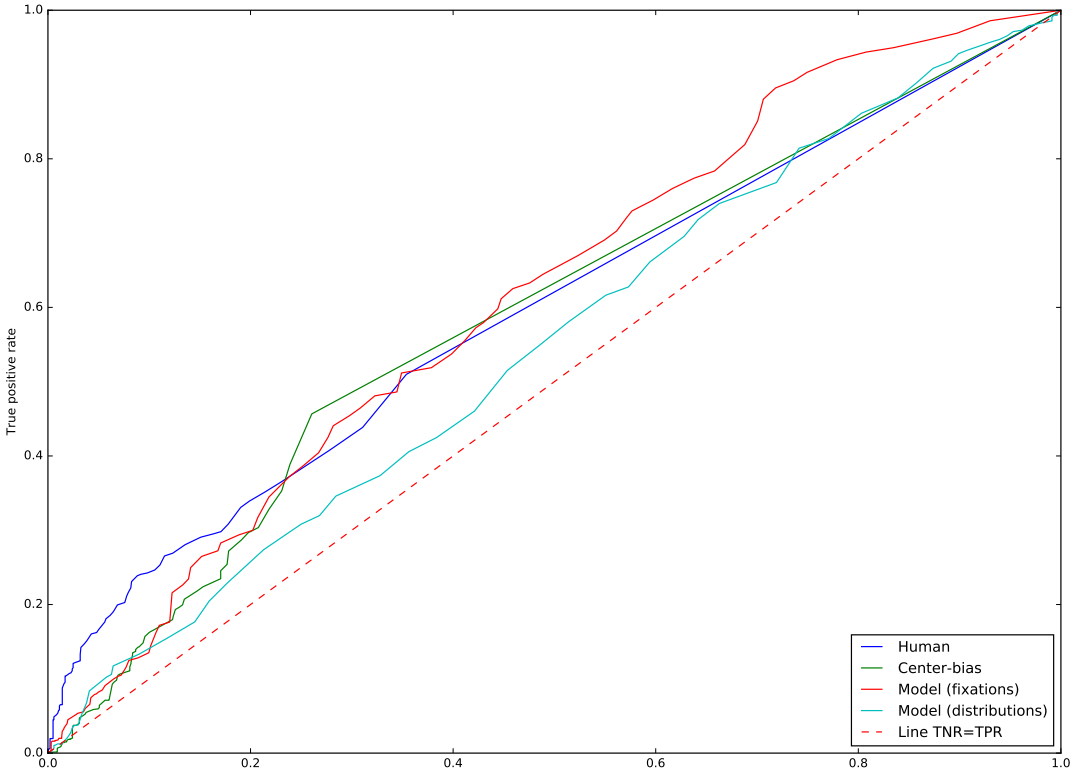


Figure 4.9: True Positive Rate against False Positive Rate (horizontal axis) as isoline threshold of classification is varied.

of our hypothesis, given the previous result in rank-correlation. Rather, it is likely an artefact of the averaging process, which smooths the relationship between attention maps and fixated locations.

It is noticeable that for very high thresholds (the bottom left corner of the graph), the human attention maps substantially outperform all of the others. This is to be expected; just below the peaks of the human attention maps, we would expect to find human fixations. Interestingly, the model fixations have a similar bump for very low thresholds (top right corner of the graph). This suggests that there are specific areas of low probability on the attention map that are higher than the minimum, and which correspond to human fixations. These might be caused by single instances where the model correctly fixates, or might be a statistical anomaly.

4.3 Discussion

Our model is not ineffective. It has learned to balance gist information and image content with the demands of grammar (for the most part), and learned a policy of fixation that involves it foveating high-entropy patches. It has not learned to attend in a fashion that takes into account high-level image features such as objects, however, and as such features are exactly what need to be robustly recognised for successful image description, it struggles in a large number of cases. This section comprises an overview of some of the criticisms that could be levelled at the model. Section 5.2 will discuss some of the directions this research could take in the future to overcome these problems.

4.3.1 Test-Training Disparity

One of the major problems with our comparison to human data is that the images in the Coco and Keller (2010) dataset [8] are unnatural collages of objects placed over a situational background (see Figure 3.6 for an example). The images in the Flickr8k dataset, on the other hand, are photographs of natural scenes.

It is unlikely, therefore, that a model trained on Flickr8k would be able to perform well on the Coco and Keller images; in our case, the descriptions the model produced were practically nonsensical. A typical example description (for the image shown in Figure 3.6) was 'a man in a black shirt and a woman in a woman in a man in', which suggests that the model has no real understanding of what it was seeing. On the other hand, the fact that there was a positive correlation between model dispersion and human dispersion suggests that, at least for reasonably rough metrics such as spatial attention, we can glean some meaningful comparison.

As such, we can interpret our results as indicating the extent to which the model's exploration of an image via its hard attention mechanism, in the absence of recognisable cues, correlates with human scanpaths of the same image. Arguably, this is a good test of the generalisability of the model's policy of attention. Additionally, as the model seems to tend to fixate low-level rather than high-level features, it may suffer less from the change of image type than a more capable (but tightly fitted) model would.

4.3.2 Lack of Temporal Features

A second criticism which might be leveled at our hypothesis directly is that neural networks do not possess the same internal time-dependent dynamics as the brain. The brain is in constant exchange with its sensory environment; a sampling mechanism such as visual attention must have evolved to take advantage of its position poised between a constantly changing environment and a constantly changing brain. A neural network, on the other hand, given some configuration of weights, the defines a mapping $f : X \rightarrow Y$.

The network's states do not update themselves independently of the presentation of some input x; even in the case of RNNs, we are artificially representing temporal dynamics by presenting the net with a sequence of inputs. It is still the case that for some input $x(t)$ there will be a specific output $y(t)$. In the brain, however, neuronal dynamics play a huge role in processing. For example, it has been hypothesised that important features such as memory are dependent in part of the attractors of the dynamical system [38], and shown that firing rates and spike rate correlations are responsible for feature binding in primary visual cortex [21]. None of these sorts of features can be modelled by a neural network.

It is therefore worth emphasising that our hypothesis is a behavioural one: we cannot expect the neural network to model the internal dynamics of the brain, and we do not. Rather, we want to know: given constraints on how much one can sample from an environment, and for how long, do models trained to perform the multi-modal task of image description manifest the same sampling behaviours as humans? If so, we would have evidence that strict adherence to internal structure is not necessary to model external behaviour, at least in the specific case of hard attention and an image description task.

Given our results, however, we might be forced to conclude that at least *some* of the dynamics of the brain are necessary for the kinds of attentional behaviour we want to simulate. Our network generates a sampling distribution by mapping an input to some output which we then treat as the parameters of that distribution; a network which had internal dynamics (such as an RBM [24]) would have to learn to parameterise an en-

ergy surface of which the sampling distribution was a marginal. It may be that this approach is more suited to modelling brain dynamics, and thus human-like behaviour, than neural networks such as RNNs.

One of the motivations for our hypothesis was that attention mechanisms have pushed the state of the art in image description further in recent years than other approaches; and attentional modifications remain a hot topic in current Machine Learning research. However, it may be simply that an attention mechanism is advantageous in all kinds of complex multimodal processing; the way in which deep neural networks utilise attention need not be anything like the way in which the brain utilises attention. In this case, our hypothesis is very unlikely to be true.

4.3.3 Scope of Results

On the one hand, we should be tentative about extending our conclusions further afield than this model. Our image description model is not state of the art (see Table 4.1), and as such it could be the case that those models which are, and which utilise attention, do so in a way our model failed to learn. However, it should not be discounted that a result in a very similar domain - that of Visual Question Answering - supports our own. Das et al (2016) [15] found that deep networks with attention correlated less with human attention than a saliency baseline, and this is reasonably compelling evidence that our result is not merely a feature of this model, but a more general one concerning models of this type.

Chapter 5

Conclusion, Future Work

5.1 Conclusion

Our opening hypothesis was that **a network required to output a successful description of an image, which is constrained to 'see' only a certain part of that image at a time, and only has a limited number of timesteps to generate the description, should adopt a policy of attention similar to that of humans given the task of describing the image**. We have implemented a neural network with a hard attention mechanism to investigate this hypothesis, and found that it does not hold: whilst there is some correlation in terms of dispersion between fixations generated by the network and those of human participants, it is less than the correlation between a centre-bias baseline and the distribution of human fixations.

In addition, several temporal aspects characteristic of human attention - centre bias, inhibition of return - are not features of our model. Whilst this is a negative result, however, we do not feel that it is an unimportant one. Modelling multi-modal human behaviour is an incredible challenge, not least because we lack a good idea of the level at which we should be pitching our models. Do we need to model the spiking of individual neurons? The interactions of brain areas? The effect of social norms? This challenge is compounded when we encounter models that perform tasks which we think of as within the domain of human behaviour. Examining how they do so, and whether they do so in a way that can inform our understanding of ourselves, is never a futile endeavour.

5.2 Future Avenues

We will finish by quickly outlining some of the directions in which this research might develop.

State of the Art: One of the problems with our model was that its performance was not state of the art. A natural follow up to this work would be to re-implement a state of the art model which utilises hard attention, and run the same tests; both qualitative and quantitative.

Increase Training Set Size: Flickr8k is a relatively small training set with a relatively limited range of types of image - as discussed in Section 4.1.2, 22% of the image descriptions contain the word 'dog' or 'dogs'. A further 17% contain the word 'man'. This imbalance may lead the optimisation algorithm to collapse more readily to local minima. Switching to a larger training set, such as Flickr30k [68] or MS COCO [39], would alleviate this problem.

Increased model complexity: As discussed in Section 3.2.4.1, the validation loss of the model seems to plateau quite quickly when training. This might be a sign that the model is too simplistic. Adding additional hidden layers might help with this, and might have the added bonus of allowing the model to connect the local low-level features it tends to fixate into higher-level features.

Perform Eye-tracking Experiments: As discussed in Section 4.3.1, one of the limitations of our approach was the disparity between the training images and the human test images. Given time, it would be nice to collect image description data using the images (or a subset of the images) we use to train the model.

Search Period: By discarding the first two thirds of our human data (that corresponding to the planning and encoding stages), we potentially miss a great deal of interesting data. Would the model learn to differentiate between an encoding stage and a production phase? Would it alter its behaviour between the two? If we gave it the choice of when to begin describing via a start-token, but slowly diminished its reward the longer it takes, how would it manage the trade-off?

Reward: Finally, we might modify the model's reward. In the current architecture, the model is rewarded at each time step by a value proportional to the log probability of that time-step's output. However, this local reward may fail to capture the more global

success of producing a grammatical sentence, or correctly mentioning the objects in the image. It might be more appropriate, therefore, to provide the model with a bipartite reward during training.

Bibliography

[1] Aditya, Somak; Yang, Yezhou; Baral, Chitta; Fermuller, Cornelia; Aloimonos, Yiannis. From Images to Sentences through Scene Description Graphs using Commonsense Reasoning and Knowledge. *arXiv preprint* arXiv:1511.03292, 2015.

[2] Biederman, I. Perceiving real-world scenes. *Science*, 177 77-79, 1972.

[3] Bishop, C M. *Neural Networks for Pattern Recognition* New York: Oxford University Press, Inc., 1995.

[4] Burr, D C; Morrone, M C; Spinelli D. Evidence for edge and bar detectors in human vision. *Vision Res*, 29(4):419-31, 1989.

[5] Clark, Andy. Whatever Next? Predictive brains, situated agents, and the future of cognitive science. *Behavioural and Brain Sciences*, 36(3): 2013. doi: 10.1017/S0140525X12000477

[6] Clarke, Alasdair DF; Coco, Moreno I.; Keller, Frank. The impact of attentional, linguistic, and visual features during object naming. *Frontiers in psychology*, 4, 2013.

[7] Clarke, Alasdair DF; Tatler, Benjamin W. Deriving an appropriate baseline for describing fixation behaviour. *Vision Research* 102: 41-51, July 2014.

[8] Coco, M. I.; Keller, F. Sentence production in naturalistic scene with referential ambiguity. In *Proceedings of the 32nd Annual Conference of the Cognitive Science Society*, 1070–1075, Portland, OR: Cognitive Science Society, 2010.

[9] Coco, Moreno I.; Keller, Frank. Scan Patterns predict Sentence Production in the Cross-modal Processing of Visual Scenes. *Cognitive Science* 36: 7, 1204–1223, 2012.

[10] Collobert, R; Weston, J; Bottou, L; Karlen, M; Kavukcuoglu, K; Kuksa, P. Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research* 12, 2493-2537, 2011.

[11] Coughlin, D. Correlating Automated and Human Assessments of Machine Translation Quality. in *MT Summit IX*, New Orleans, USA 23–27, 2003.

[12] Dainton, Barry. "Précis: Stream of Consciousness" *PSYHCE* 10: 1, 1-29, May 2004.

[13] Dainton, Barry. *Stream of Consciousness: Unity and Continuity in Conscious Experience.* London: Routledge, 2000.

[14] Daniels, H E. The relation between measures of correlation in the universe of sample permutations. *Biometrika*, 33, 129-35, 1944.

[15] Das, A; Agrawal, H; Zitnick, C; Parikh, D; Batra, D. Human Attention in Visual Question Answering: Do Humans and Deep Networks Look at the Same Regions? *arXiv:1606.03556v2*, 17 Jun 2016.

[16] De Valois, R. L; De Valois, K. K. Spatial Vision. *Annual Review of Psychology* 31, 1 (1980): 309-341.

[17] Donoho, D L; Vetterli, M; DeVore, R A; Daubechies, I. Data Compression and Harmonic Analysis. *IEEE Trans. Inf. Theory*, 44 (6):2435–247, October 1998.

[18] Duncan, J. Demonstration of capacity limitation. *Cognitive Psychology*, 12: 75-96, 1980.

[19] Franconeri, Stephen; Hollingworth, Andrew; Simons, Daniel. Do New Objects Capture Attention?. *Psychological Science* 16: 275-281, 2005.

[20] Friston, K; Adams, R. A; Perrint, L; Breakspear, M. Perceptions as Hypothesis, Saccades as Experiments. *Frontiers in Psychology* 151(3):1-20, May 2012. doi: 10.3389/fpsyg.2012.00151

[21] Golledge, H D; Panzeri, S; Zheng, F; Pola, G; Scannell, J W; Giannikopoulos, D V; Mason, R J; Tovée, MJ; Young, M P. Correlations, feature-binding and population coding in primary visual cortex. *Neuroreport.* 14(7):1045-50, May 23 2003.

[22] Haralick, R; Shapiro, L; *Computer and Robot Vision, Vol. 1* Addison-Wesley Publishing Company, 346 - 351, 1992.

[23] Hinton, G. Unpublished lecture slides: Neural Networks for Machine Learning. www.cs.toronto.edu/tijmen/csc321/slides/lecture_slides_lec6.pdf

[24] Hinton, G. E. Training Products of Experts by Minimizing Contrastive Divergence. *Neural Computation*, 14: 1771-1800, 2002.

[25] Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8), 1735-1780, 1997. Tutorial at http://deeplearning.net/tutorial/lstm.html

[26] Hodosh, M. and Hockenmaier, J. Sentence-based image description with scalable, explicit models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops* 294-300, 2013.

[27] Hohwy, Jacob. Attention and conscious perception in the hypothesis testing brain. *Frontiers in Psychology* 96(3): 1-14, April 2012. doi: 10.3389/fpsyg.2012.00096

[28] Holmqvist, K; Nystrom, M; Andersson, R; Dewhurst, R; Jarodzka, H; Van de Weijer, J. *Eye Tracking* Oxford, UK: Oxford University Press, 2011.

[29] Huang, Liqiang; Triesman, Anne; Pashler, Harold. Characterising the Limits of Human Visual Awareness, *Science*, 317 (2007): 823-825.

[30] Jensen, J. L. W. V. Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta Mathematica*, 30 (1): 175–193, 1906. doi:10.1007/BF02418571.

[31] Judd, T; Ehinger, K; Durand, F; Torralba, A. Learning to predict where humans look. *International Conference on Computer Vision (ICCV)*, 2009

[32] Kingma, D; Ba, J. Adam: A Method for Stochastic Optimization *CoRR*, abs/1412.6980, 2014. http://arxiv.org/abs/1412.6980

[33] Kiros, R; Salahutdinov, R; Zemel, R. Multi-modal neural language models. In *International conference on Machine Learning*, 595-603, 2014.

[34] Klein, R M; MacInnes, W J. Inhibition of return is a foraging facilitator in visual search. *Psychological Science* 10(4), 346-352, 1999.

[35] Kolers, P A; von Grünau, M Shape and Colour in Apparent Motion. *Vision Research* 16, 4: 329-335, 1976.

[36] LeCun, Y; Bottou, L; Bengio, Y; Haffner, P. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278-2324, November 1998.

[37] Leonard, N. Recurrent Model of Visual Attention. Blog post, September 21, 2015. http://torch.ch/blog/2015/09/21/rmva.html

[38] Li, G; Ramanathan, K; Ning, N; Shi, L; Wen, C. Memory Dynamics in Attractor Networks. *Computational Intelligence and Neuroscience*, 191745, 2015. http://doi.org/10.1155/2015/191745

[39] Lin, Tsung-Yi; Maire, M; Belongie, Serge; Hays,James; Perona, Pietro; Ramanan, Deva; Dollár, Piotr; Zitnick, C. Lawrence. Microsoft COCO: Common objects in context. In *Computer Vision–ECCV* pp. 740-755. Springer International Publishing, 2014.

[40] Masciocchi, C; Mihalas, S; Parkhurst, D; Niebur, E Everyone knows what is interesting: Salient locations which should be fixated. *Journal of Vision*, Vol.9, 25, Oct 2009. doi:10.1167/9.11.25

[41] Mason, S J; Graham, N E. Areas beneath the relative operating characteristics (ROC) and relative operating levels (ROL) curves: Statistical significance and interpretation. *Quarterly Journal of the Royal Meteorological Society*, 128, 2145-2166, 2002. doi: 10.1256/003590002320603584

[42] Mikolov, T; Chen, K; Corrado, G; Dean, J Efficient Estimation of Word Representations in Vector Space. *arXiv*:1301.3781v3, 7 Sep 2013.

[43] Mikolov, T; Karafiát, M; Burget, L; Èernocký, J; Khudanpur, S. Recurrent neural network based language model. *INTERSPEECH 2010*: http://www.fit.vutbr.cz/research/groups/speech/publi/2010/mikolov_interspeech2010_IS100722.

[44] Mnih, V; Heess, N; Graves, A; Kavukcuoglu, K. Recurrent Models of Visual Attention. In *Advances in Neural Information Processing Systems*, 2204-2212, 2014.

[45] Nielsen, M. A. *Neural Networks and Deep Learning* Determination Press, 2015. http://neuralnetworksanddeeplearning.com/

[46] Noton, D; Stark, L. Scanpaths in saccadic eye movements while viewing and recognising patterns. *Science*, 171(3968): 308-11, 1971.

[47] Papineni, K.; Roukos, S.; Ward, T.; Zhu, W. J. BLEU: a method for automatic evaluation of machine translation. *40th Annual meeting of the Association for Computational Linguistics*, 311–318, 2002.

[48] Posner, M.I.; Rafal, R.D.; Choate, L.S.; Vaughan, J. Inhibition of return: Neural basis and function. *Cognitive Neuropsychology*, 2, 211–228, 1985.

[49] Renals, S. Machine Learning Practical. *Lecture course, University of Edinburgh*, 2015. Slides: http://www.inf.ed.ac.uk/teaching/courses/mlp/2016/mlp08-cnn2.pdf

[50] Renshaw, J A; Finlay, J E; Tyfa, D; Ward, R D. Regressions re-visited: A new definition for the visual display paradigm. In *CHI '04 Extended Abstracts on Human Factors in Computing Systems*, 1437-1440, New York ACM, 2004.

[51] Rojas, R Why the Normal Distribution? *Note*, February 2010. http://www.inf.fu-berlin.de/inst/ag-ki/rojas_home/documents/tutorials/Gaussian-distribution.pdf

[52] Russakovsky, O; Deng, J; Su, H; Krause, J; Satheesh, S; Ma, S; Huang, Z; Karpathy, A; Khosla, A; Bernstein, M; Berg, A; Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3: 211-252, 2015. doi: 10.1007/s11263-015-0816-y

[53] Sampanes, A C; Tseng, P; Bridgeman, B. The role of gist in scene recognition. *Vision Research* 48 2275-2283, 2008.

[54] Shannon, C E. A mathematical theory of communication. *Bell System Technical Journal*, 27 (3): 379–423, July 1948. doi: 10.1002/j.15387305.1948.tb01338.x

[55] Scherer, D; Muller, A; Behnke, S. Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition. *ICANN*, Thessaloniki, Greece, September 2010.

[56] Simonyan, K; Zisserman, A Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv technical report*, 2014.

[57] Smallwood, R; Sondik, E The Optimal Control of Partially Observable Markov Processes over a Finite Horizon. *Operations Research* 21, 5: 1071-1088, 1971. doi: 10.1287/opre.21.5.1071

[58] Smith, T J; Henderson, J M. Facilitaion of return during scene viewing. *Visual Cognition*, 17(6-7), 1083-1108, 2009.

[59] Song, M. Entropy Encoding in Wavelet Image Compression. http://www.siue.edu/m̃song/Research/entropy.pdf [accessed 15/09/16]

[60] Srivastava, N; Hinton, G; Krizhevsky, A; Sutskever, I; Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15: 1929-1958, 2014.

[61] Tye, Michael. *Consciousness and Persons.* Sabon: The Massachusetts Institute of Technology Press, 2003.

[62] Unema, P J A; Pannasch, S; Joos, M; Velichovsky, B M. Time course of information processing during scene perception: The relationship between saccade amplitude and fixation duration. *Visual Cognition* 12(3), 473-494, 2005.

[63] Vapnik, V. Principles of Risk Minimization for Learning Theory. In *NIPS*, 831-838, 1991.

[64] Vinyals, O; Toshev, A; Bengio, S; Erhan, D. Show and Tell: A neural image caption generator. *arXiv*:1411.4555, November 2014.

[65] Watzl, Sebastian. How Attention Structures Consciousness. paper presented at *Percpetual Attention, University of Antwerp*, September 1, 2012.

[66] Williams, R.J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.

[67] Xu, Kelvin; Ba, Jimmy; Kiros, Ryan; Courville, A;Salakhutdinov, R; Zemel, Richard; Bengio, Yoshua. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint*, arXiv:1502.03044, 2015.

[68] Young, P; Lai, A; Hodosh, M; Hockenmaier, J. From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions. *Transactions of the Association for Computational Linguistics*, 2(Feb):67-78, 2014.

[69] Zangermeister, W H; Sherman, K; Stark, L. Evidence for a global scanpath strategy in viewing abstract compared with realistic images. *Neuropsychologia*, 33(8), 1009-1025, 1995.

[70] Zeiler, M; Ranzato, M; Monga, R; Mao, M; Yang, K; Le, Q; Nguyen, P; Senior, A; Vanhoucke, V; Dean, J; Hinton, G. On

Rectified Linear Units for Speech Processing. *ICASSP*, 2013. http://www.matthewzeiler.com/pubs/icassp2013/icassp2013.pdf