Attention models: Humans do not actively use all the information available to them from the environment at any given time. Rather, they focus on specific portions of the data that are relevant to the task at hand. This biological notion is referred to as *attention*. A similar principle can also be applied to artificial intelligence applications. Models with attention use reinforcement learning (or other methods) to focus on smaller portions of the data that are relevant to the task at hand. Such methods have

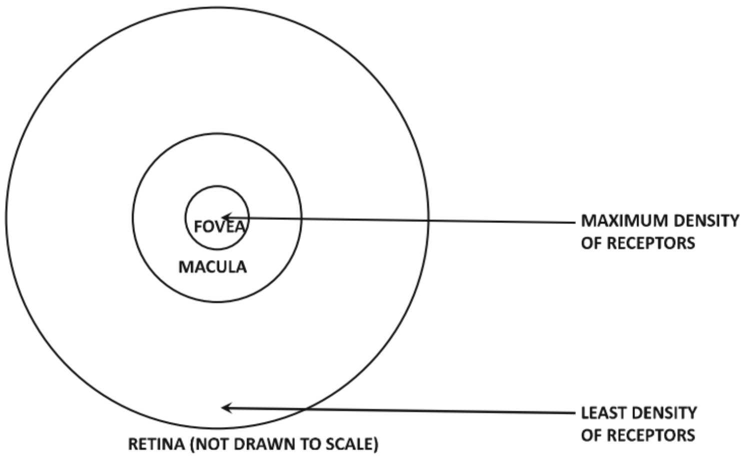

FOVEA
MACULA
MAXIMUM DENSITY OF RECEPTORS
LEAST DENSITY OF RECEPTORS
RETINA (NOT DRAWN TO SCALE)

Fig: Resolutions at different regions of Eye

either on the door or the mailbox of a house. In this process, the retina often has an image of a broader scene, although one rarely focuses on the full image. The retina has a small portion, referred to as the *macula* with a central *fovea*, which has an extremely high resolution compared to the remainder of the eye. This region has a high concentration of

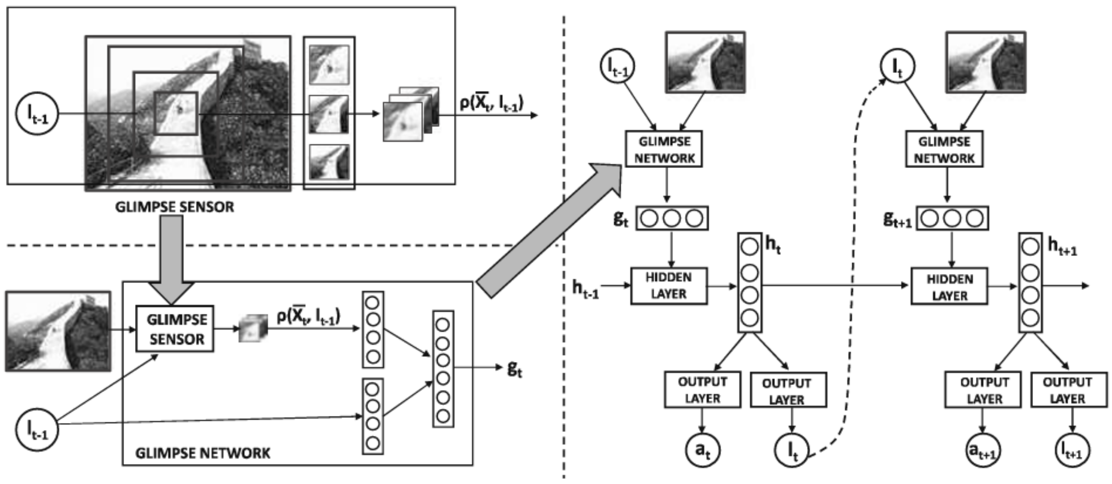
resolution compared to the remainder of the eye. This region has a high concentration of
color-sensitive cones, whereas most of the non-central portions of the eye have relatively low resolution with a predominance of color-insensitive rods. The different regions of the eye are shown in Figure 10.1. When reading a street number, the fovea *fixates* on the number, and its image falls on a portion of the retina that corresponds to the macula (and especially the fovea). Although one is aware of the other objects outside this central field of vision, it is virtually impossible to use images in the peripheral region to perform detail-oriented tasks.

The foveal region is a tiny fraction of the full retina, and it has a diameter of only 1.5 mm. The eye effectively transmits a high-resolution version of less than 0.5% of the surface area of the image that falls on the full retina. This approach is biologically advantageous, because

The eye effectively transmits a high-resolution version of less than 0.5% of the surface area of the image that falls on the full retina. This approach is biologically advantageous, because
only a carefully selected part of the image is transmitted in high resolution, and it reduces the internal processing required *for the specific task at hand.* Although the structure of the eye makes it particularly easy to understand the notion of selective attention towards visual inputs, this selectivity is not restricted only to visual aspects. Most of the other senses of

Recurrent Models of Visual attention

on important parts of an image. The idea is to use a (relatively simple) neural network in which only the resolution of specific portions of the image centered at a particular location is high. This location can change with time, as one learns more about the relevant portions of the image to explore over the course of time. Selecting a particular location in a given time-stamp is referred to as a *glimpse*. A recurrent neural network is used as the controller to identify the precise location in each time-stamp; this choice is based on the feedback from the glimpse in the previous time-stamp. The work in [338] shows that using a simple neural



Glimpse network: The glimpse network contains the glimpse sensor and encodes both the glimpse location l_{t-1} and the glimpse representation $\rho(\overline{X}_t, l_{t-1})$ into hidden spaces using linear layers. Subsequently, the two are combined into a single hidden representation using another linear layer. The resulting output g_t is the input into the tth time-stamp of the hidden layer in the recurrent neural network. The glimpse network

Recurrent neural network: The recurrent neural network is the main network that is creating the action-driven outputs in each time-stamp (for earning rewards). The recurrent neural network includes the glimpse network, and therefore it includes the glimpse sensor as well (since the glimpse sensor is a part of the glimpse network). This output action of the network at time-stamp t is denoted by a_t, and rewards are associated with the action. In the simplest case, the reward might be the class

are associated with the action. In the simplest case, the reward might be the class
label of the object or a numerical digit in the *Google Streetview* example. It also outputs a location l_t in the image for the next time-stamp, on which the glimpse network should focus. The output $\pi(a_t)$ is implemented as a probability of action a_t. This probability is implemented with the softmax function, as is common in policy

Reinforcement Learning

This approach is couched within the framework of reinforcement learning, which allows it to be used for any type of visual reinforcement learning task (e.g., robot selecting actions to achieve a particular goal) instead of image recognition or classification. Nevertheless, supervised learning is a simple special case of this approach.

The actions a_t correspond to choosing the choosing the class label with the use of a softmax prediction. The reward r_t in the tth time-stamp might be 1 if the classification is correct after t time-stamps of that roll out, and 0, otherwise. The overall reward R_t at the tth time-stamp is given by the sum of all discounted rewards over future time stamps. However, this action can vary with the application at hand. For example, in an image captioning application, the action might correspond to choosing the next word of the caption.

Section 9.5.2 of Chapter 9. The gradient of the expected reward at time-stamp t is given by the following:

$$\nabla E[R_t] = R_t \nabla \log(\pi(a_t)) \tag{10.1}$$

Backpropagation is performed in the neural network using this gradient and policy rollouts. In practice, one will have multiple rollouts, each of which contains multiple actions. Therefore, one will have to add the gradients with respect to all these actions (or a mini-batch of these actions) in order to obtain the final direction of ascent. As is common in policy gradient methods, a baseline is subtracted from the rewards to reduce variance. Since a class label is output at each time-stamp, the accuracy will improve as more glimpses are used. The approach performs quite well using between six and eight glimpses on various types of data.

Image Captioning

is discussed in Section 7.7.1 of Chapter 7. In this approach, a single feature representation $\overline{v}$ of the entire image is input to the *first time-stamp* of a recurrent neural network. When a feature representation of the entire image is input, it is only provided as input at the first time-stamp when the caption begins to be generated. However, when attention is used, we want to focus on the portion of image that corresponds to the word being generated. Therefore, it makes sense to provide different attention-centric inputs at different time-stamps. For example, consider an image with the following caption:

"*Bird flying during sunset.*"

The attention should be on the location in the image corresponding to the wings of the bird while generating the word "*flying,*" and the attention should be on the setting sun, while generating the word "*sunset.*" In such a case, each time-stamp of the recurrent neural

network should receive a representation of the image in which the attention is on a specific location. Furthermore, as discussed in the previous section, the values of these locations are also generated by the recurrent network in the previous time-stamp.

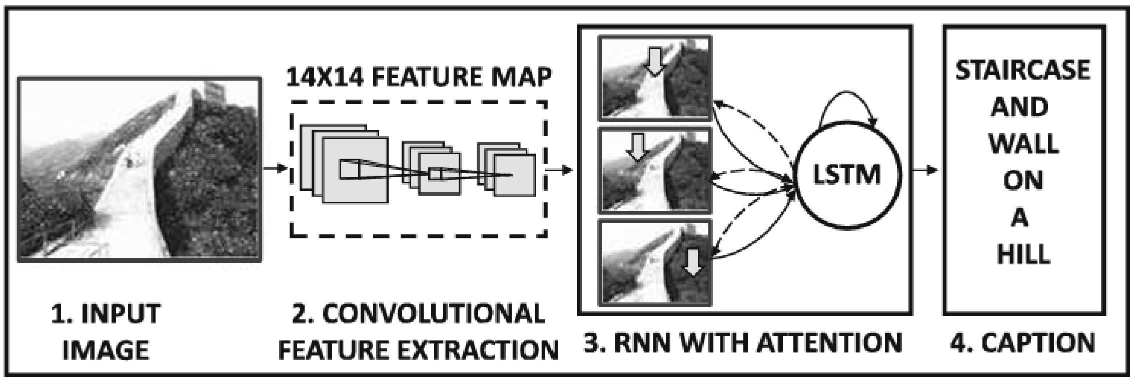


Fig: Visual attention in Image captioning

handle the higher complexity of the problem. First, the glimpse network does use a more sophisticated convolutional architecture to create a 14×14 feature map. This architecture is illustrated in Figure 10.3. Instead of using a glimpse sensor to produce the modified version of the image in each time-stamp, the work in [540] starts with L different preprocessed variants on the image. These preprocessed variants are centered at different locations in the image, and therefore the attention mechanism is restricted to selecting from one of these locations. Then, instead of producing a location l_t in the $(t-1)$th time-stamp, it produces a probability vector $\overline{\alpha}_t$ of length L indicating the relevance of each of the L locations for which representations were preprocessed in the convolutional neural network. In hard attention models, one of the L locations is sampled by using the probability vector $\overline{\alpha}_t$, and the preprocessed representation of that location is provided as input into the hidden state

for which representations were preprocessed in the convolutional neural network. In hard attention models, one of the L locations is sampled by using the probability vector $\overline{\alpha}_t$, and the preprocessed representation of that location is provided as input into the hidden state
h_t of the recurrent network at the next time-stamp. In other words, the glimpse network in the classification application is replaced with this sampling mechanism. In soft attention

in the classification application is replaced with this sampling mechanism. In soft attention
models, the representation models of all L locations are averaged by using the probability vector $\overline{\alpha}_t$ as weighting. This averaged representation is provided as input to the hidden state at time-stamp t. For soft attention models, straightforward backpropagation is used for training, whereas for hard attention models, the REINFORCE algorithm (cf. Section 9.5.2

Generative Adversarial Networks (GANs)

Two types of learning models used for creating adversarial Nwtworks are:

1. *Discriminative models:* Discriminative models directly estimate the conditional probability $P(y|\overline{X})$ of the label y, given the feature values in $\overline{X}$. An example of a discriminative model is logistic regression.

2. *Generative models:* Generative models estimate the joint probability $P(\overline{X}, y)$, which is a generative probability of a data instance. Note that the joint probability can be used to estimate the conditional probability of y given $\overline{X}$ by using the Bayes rule as follows:
$$P(y|\overline{X}) = \frac{P(\overline{X}, y)}{P(\overline{X})} = \frac{P(\overline{X}, y)}{\sum_z P(\overline{X}, z)} \tag{10.18}$$
An example of a generative model is the naïve Bayes classifier.

Discriminative models can only be used in supervised settings, whereas generative models are used in both supervised and unsupervised settings. For example, in a multiclass setting,

Generative adversarial networks work with two neural network models simultaneously. The first is a generative model that produces synthetic examples of objects that are similar to a real repository of examples. Furthermore, the goal is to create synthetic objects that are so realistic that it is impossible for a trained observer to distinguish whether a particular object belongs to the original data set, or whether it was generated synthetically. For example, if

real and fake examples of car images. The second network is a discriminative network that has been trained on a data set which is labeled with the fact of whether the images are synthetic or fake. The discriminative model takes in inputs of either real examples from the base data or synthetic objects created by the generator network, and tries to discern as to whether the objects are real or fake. In a sense, one can view the generative network

as to whether the objects are real or fake. In a sense, one can view the generative network
as a "counterfeiter" trying to produce fake notes, and the discriminative network as the "police" who is trying to catch the counterfeiter producing fake notes. Therefore, the two networks are adversaries, and training makes both adversaries better, until an equilibrium is reached between them. As we will see later, this adversarial approach to training boils

When the discriminative network is correctly able to flag a synthetic object as fake, the fact is used by the generative network to modify its weights, so that the discriminative network will have a harder time classifying samples generated from it. After modifying the weights of the generator network, new samples are generated from it, and the process is repeated. Over time, the generative network gets better and better at producing counterfeits. Eventually, it becomes impossible for the discriminator to distinguish between real and synthetically generated objects. In fact, it can be formally shown that the *Nash equilibrium* of this minimax game is a (generator) parameter setting in which the distribution of points created by the generator is the same as that of the data samples. For the approach to work well, it is important for the discriminator to be a high-capacity model, and also have access to a lot of data.

The generated objects are often useful for creating large amounts of synthetic data for machine learning algorithms, and may play a useful role in data augmentation. Further-

Training GANs

The training process of a generative adversarial network proceeds by alternately updating the parameters of the generator and the discriminator. Both the generator and discriminator are neural networks. The discriminator is a neural network with d-dimensional inputs and a single output in $(0,1)$, which indicates the probability whether or not the d-dimensional input example is real. A value of 1 indicates that the example is real, and a value of 0 indicates that the example is synthetic. Let the output of the discriminator for input $\overline{X}$ be denoted by $D(\overline{X})$.

The generator takes as input noise samples from a p-dimensional probability distribution, and uses it to generate d-dimensional examples of the data. One can view the

The goal for the discriminator is to correctly classify the real examples to a label of 1, and the synthetically generated examples to a label of 0. On the other hand, the goal for the generator is generate examples so that they fool the discriminator (i.e., encourage the discriminator to label such examples as 1). Let R_m be m randomly sampled examples from the real data set, and S_m be m synthetic samples that are generated by using the generator.

input to create the data samples $S_m = \{G(\overline{Z}_1)\ldots G(\overline{Z}_m)\}$. Therefore, the *maximization* objective function J_D for the discriminator is as follows:

$$\text{Maximize}_D\, J_D = \underbrace{\sum_{\overline{X}\in R_m} \log\left[D(\overline{X})\right]}_{m \text{ samples of real examples}} + \underbrace{\sum_{\overline{X}\in S_m} \log\left[1 - D(\overline{X})\right]}_{m \text{ samples of synthetic examples}}$$

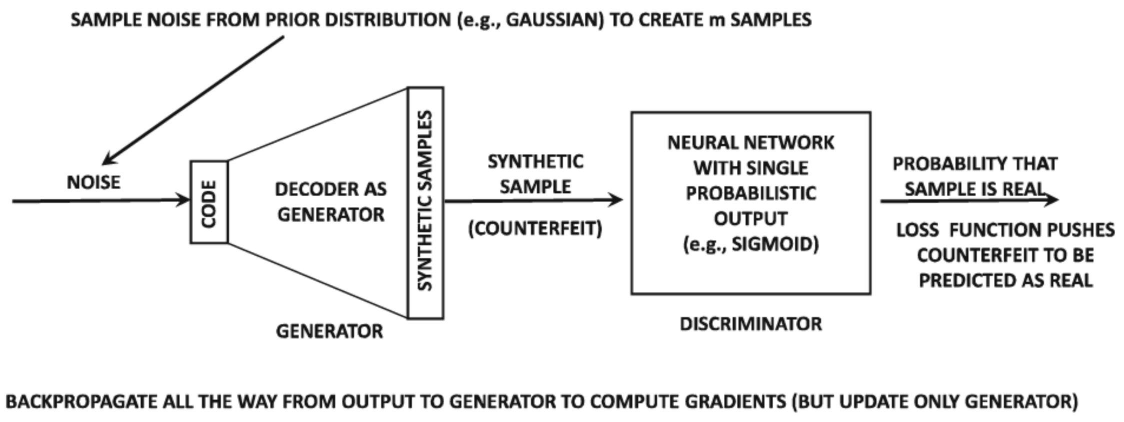
Next we define the objective function of the generator, whose goal is to fool the discriminator. For the generator, we do not care about the real examples, because the generator only cares about the sample it generates. The generator creates m synthetic samples, S_m, and the goal is to ensure that the discriminator recognizes these examples as genuine ones. Therefore, the generator objective function, J_G, *minimizes* the likelihood that these samples are flagged as synthetic, which results in the following optimization problem:

$$\begin{aligned}\text{Minimize}_G\, J_G &= \underbrace{\sum_{\overline{X}\in S_m} \log\left[1 - D(\overline{X})\right]}_{m \text{ samples of synthetic examples}}\\ &= \sum_{\overline{Z}\in N_m} \log\left[1 - D(G(\overline{Z}))\right]\end{aligned}$$

The overall optimization problem is therefore formulated as a minimax game over J_D. Note that maximizing J_G over different choices of the parameters in the generator G is the same as maximizing J_D because $J_D - J_G$ does not include any of the parameters of the generator G. Therefore, one can write the overall optimization problem (over both generator and discriminator) as follows:

$$\text{Minimize}_G\, \text{Maximize}_D\, J_D \tag{10.19}$$

The result of such an optimization is a *saddle point* of the optimization problem. Examples


SAMPLE NOISE FROM PRIOR DISTRIBUTION (e.g., GAUSSIAN) TO CREATE m SAMPLES
NOISE
CODE
DECODER AS GENERATOR
SYNTHETIC SAMPLES
GENERATOR
SYNTHETIC SAMPLE
(COUNTERFEIT)
NEURAL NETWORK WITH SINGLE PROBABILISTIC OUTPUT (e.g., SIGMOID)
DISCRIMINATOR
PROBABILITY THAT SAMPLE IS REAL
LOSS FUNCTION PUSHES COUNTERFEIT TO BE PREDICTED AS REAL
BACKPROPAGATE ALL THE WAY FROM OUTPUT TO GENERATOR TO COMPUTE GRADIENTS (BUT UPDATE ONLY GENERATOR)