Attention Neural Networks

Amanda

September 28, 2021

Contents

Background

Deep learning, as a subfield of machine learning, has attracted a few waves of research interests in the history. The current wave started in 2006, and is mainly due to increased dataset sizes and computation resources that led to significantly improved model performance [GBC16]. The models in deep learning, known as neural networks, have achieved state-of-the-art performance in various tasks such as object detection and speech recognition [LBH15]. Researchers have designed different neural network architectures to tackle different problems [LWL$^+$17], and many neural networks that performed well in natural language processing (NLP) [BCB14] [VSP$^+$17] and computer vision [PWLK18] [WPLK18] [WJQ$^+$17] employed attention mechanisms. For simplicity, we will use "attention networks" to refer to neural networks with attention mechanisms. This essay will introduce the notion of attention in neural networks, and provide an overview of some attention network structures. Since attention networks also have many variants, we will discuss specific instances in the context of different applications.

Inspiration from Biology

The idea of attention in machine learning was inspired by the concept of selective attention in biology and psychology. Our brains pay selective attention to only a small fraction of all the visual information that the eyes perceive, and we are only able to discern and focus on a conversation in a noisy environment because we can selectively attend to specific auditory information [FRC10]. However, the attention mechanisms do not fully resemble the structures of human perceptual systems and neural systems, as the biological structures are too complex, and mechanisms that try to resemble them as much as possible may take too much computation resources and may even have less reliable performance [FRC10].

Implicit Attention in Neural Networks

A neural network in supervised learning can be viewed as a parametric function that maps an input to an output. By construction, the network (possibly without attention mechanisms) implicitly pays more attention to some input dimensions than others, and the network Jacobian reveals which input dimensions have more influence over each output dimension [Gra]. For a differentiable neural network mapping an M-dimensional input x to an N-dimensional output y, the Jacobian is an $N \times M$ matrix J, with $J_{ij} = \partial y_i / \partial x_j$.

For example, a convolutional neural network (CNN) that classifies M-dimensional images might map input pixels to the probability that the image shows a cat and the probability that it shows a dog (so $N = 2$ in this case), and the CNN might implicitly focus on the pixels that constitute the animal's ears more than those that constitute the background.

Benefits of Explicit Attention

Although neural networks can learn implicit attention, there is little control over what to pay attention to. For example, the basic recurrent neural network for machine translation takes a sequence of input words and outputs the prediction of the next word, but due to the vanishing gradient problem, it can hardly learn the relevance of earlier inputs when the sequence is long, even if the early inputs are more relevant to the output [BSF94]. Hence, researchers have used attention mechanisms to explicitly build attention into the networks. Such networks are more likely to reap the benefits of selective attention, ignoring irrelevant information and spending more computation power analysing relevant inputs [Gra].

Moreover, the parameters and outputs of the attention mechanisms provide straightforward insight about which parts of the input are the most relevant to the output, making it easier for humans to interpret what the model focuses on [VSP+17].

Neural Networks

This section briefly introduces neural network architectures that the attention mechanisms in the next two sections will build upon. This section is written with reference to [GBC16], which gives more comprehensive descriptions of the architectures.

Feedforward Neural Networks

Feedforward neural networks, also known as the multilayer perceptions (MLPs), have the most fundamental deep neural network architectures. A neuron is the basic unit of an MLP. It takes some input vector $\mathbf{x}$ and performs a linear transformation on them to produce an output known as its pre-activation $\mathbf{w}^T\mathbf{x} + b$, where $\mathbf{w}$ and b are model parameters. A layer of an MLP consists of one or more neurons, and there can be a non-linear transformation on the pre-activations of its neurons to produce outputs known as the activations. This non-linear transformation may be a function across all pre-activations of this layer like the *softmax* function, or it may independently transform the pre-activation of each neuron. The first layer is the input layer that passes the input $\mathbf{x}$ into the network, the final layer is the output layer, and any other layers in the middle are hidden layers. Figure 1 shows a simple MLP. In the model training, we aim to find model parameters that minimise some loss function, such as the negative log likelihood. Information only flows forward in an MLP. Other types of neural networks also consist of neurons and have similar training methods, but the connections between neurons may be different.

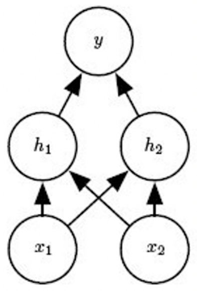

Figure 1: A multi-layer perceptron with 3 layers [GBC16]

Recurrent Neural Networks (RNNs)

RNNs are usually used in tasks where the inputs are sequential data, such as words in a sentence, or signals through time. The inputs might be a sequence of real values, or it might be a sequence of vectors. Figure 2 shows a simple RNN on the left with the output of the hidden layer at time step $t-1$ also serving as the input to the hidden layer at time step t, and it shows the unfolded RNN on the right.

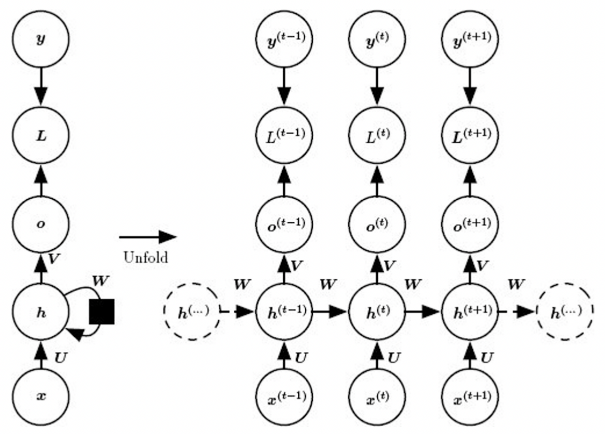

Figure 2: A recurrent neural network (left) with input layer x, hidden layer h and output layer o. U, V and W are model parameters. L is the loss function that computes loss given the predicted value o and the true value y. The black square represents a delay by one time step. The RNN is unfolded to give the network on the right [GBC16]

The RNNs introduced above has the hidden state (which is the output of the hidden layer) at time step t representing some compressed information about the inputs at time steps at and before t, so the output of the network at time step t depends on those inputs only. Bidirectional RNNs, however, has unrolling in both directions as shown in Figure 3, such that the output at time step t depends on all inputs, particularly inputs whose time steps are around t. Bidirectional RNNs are better for tasks like speech recognition, where the output at a time step can depend on the context around the time step.

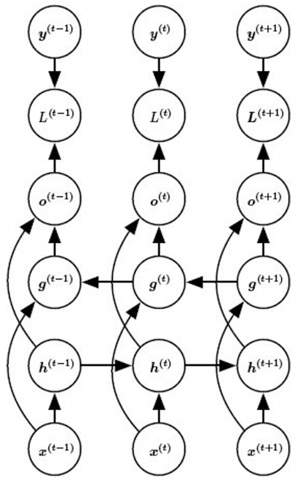

Figure 3: A bidirectional recurrent neural network [GBC16]

With some modifications, we have RNNs that map a sequence of inputs to a single "context vector" that summarises the inputs, and RNNs that map a single input vector to a sequence of outputs. These can constitute the encoder-decoder model as shown in Figure 4, commonly used in machine translation.

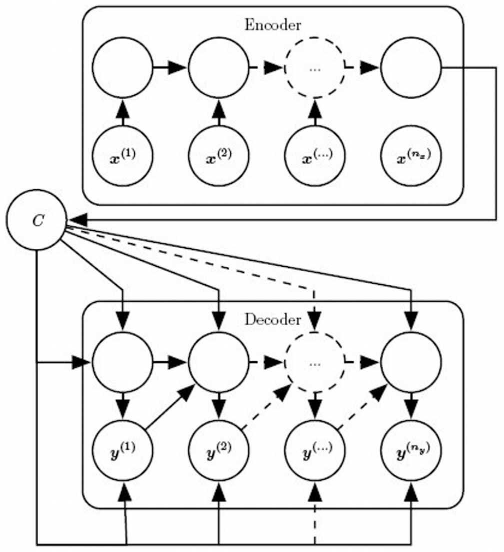

Figure 4: A simple encoder-decoder architecture consisting of RNNs [GBC16]

Convolutional Neural Networks (CNNs)

CNNs are typically used for tasks whose inputs have a grid structure. For example, if the input is an image with width w and height h in c channels, the input may be represented by a grid of $w \times h \times c$ values. CNNs use the convolution operation between some network layers, and most CNNs also use another operation called pooling. Convolution applies a kernel across an input grid to extract the presence of some pattern that the kernel detects at different parts of the input (see Figure 5), such as sharp vertical edges in an image. The output, sometimes called the "feature map", also takes the shape of a grid. The kernel is a matrix whose components are model parameters to be learnt. Pooling reduces the dimensions of the input grid, dividing the input grid into patches and summarising each patch into a single value, often by taking the maximum or the average. Figure 6 shows a common structure found in CNN architectures.

Attention Networks in Natural Language Processing (NLP)

In this section, we discuss two attention mechanisms used for NLP, with neural machine translation as the concrete task. Neural machine translation is the task of translating an input sequence in one language to an equivalent output sequence in another language using a neural network [BCB14]. We will first introduce an attention mechanism built upon RNNs, and then one that gets rid of the recurrent structure.

The First Attention Mechanism in Neural Machine Translation

[BCB14] first proposed to use an attention mechanism in neural machine translation, making an improvement from the basic encoder-decoder models. Figure 7 shows the architecture.

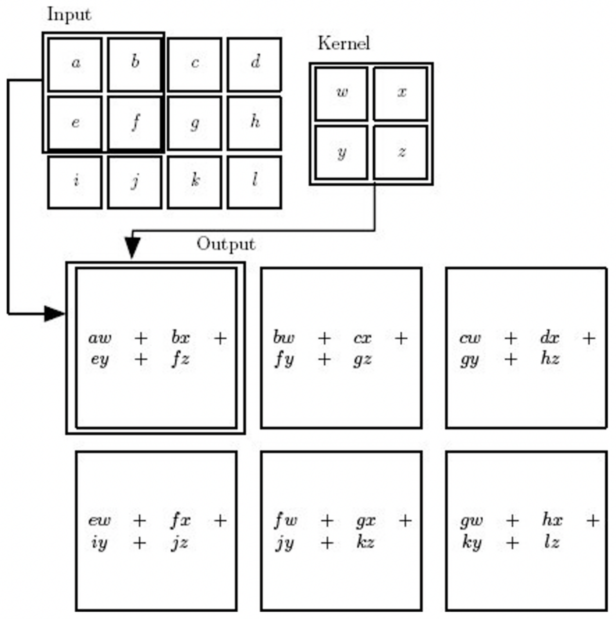

Figure 5: The convolution operation [GBC16]

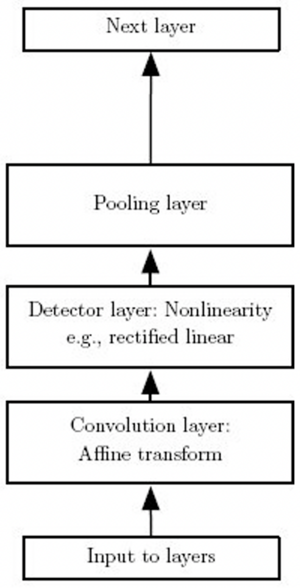
Next layer
Pooling layer
Detector layer: Nonlinearity e.g., rectified linear
Convolution layer: Affine transform
Input to layers

Figure 6: A common structure in convolutional neural networks [GBC16]

The encoder is a bidirectional RNN. Each of the $T = T_x$ input words is represented by a vector $\mathbf{x}_j$ where $j \in \{1, ..., T_x\}$, and the outputs are a sequence of vectors $\mathbf{y}_i$ where $i \in \{1, ..., T_y\}$. Let $\mathbf{h}_j$ be the concatenation of the two hidden states from the encoder at position j. The new model does not attempt to compress all information about the inputs

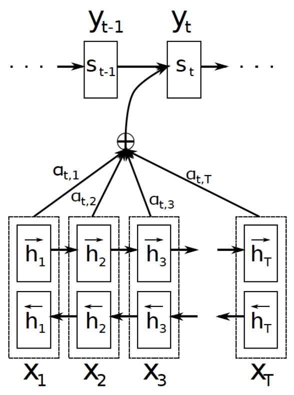

Figure 7: The attention mechanism in [BCB14]

into a single context vector like the original encoder-decoder does. Each hidden state $\mathbf{h}_j$ from the encoder now serves as a representation of input information around the position j, and for each $i \in \{1, ..., T_y\}$, the decoder uses a context vector $\mathbf{c}_i$ for the output word $\mathbf{y}_i$ by

$$\mathbf{c}_i = \sum_{j=1}^{T_x} \alpha_{ij}\mathbf{h}_j \tag{1}$$

$\mathbf{c}_i$ is therefore a weighted sum of the hidden states at all input positions. α_{ij} is defined below.

The decoder has a sequence of hidden states $\mathbf{s}_i$ where

$$\mathbf{s}_i = f(\mathbf{s}_{i-1}, \mathbf{y}_{i-1}, \mathbf{c}_i) \tag{2}$$

for some function f to be learnt. Intuitively, $\mathbf{s}_i$ remembers the previous output sequence and considers the current context vector to produce the current output vector. α_{ij} is then computed by

$$\alpha_{ij} = \frac{\exp(r_{ij})}{\sum_{k=1}^{T_x} \exp(r_{ik})} \tag{3}$$

where

$$r_{ij} = a(\mathbf{s}_{i-1}, \mathbf{h}_j) \tag{4}$$

The function a in Equation 4 evaluates the relevance between the output at position i and the inputs around position j, and a is incorporated into the neural network as a feedforward module, jointly trained with the other modules by backpropagation. As a result, the weighted sum $\mathbf{c}_i$ pays more attention to the inputs that are more relevant to the output at i.

The model is compared against the more conventional encoder-decoder model in [CvMG$^+$14] twice, first with sentences that have up to 30 words, and then up to 50 words, and the new model outperformed the conventional one in both cases. In particular, [BCB14] conjectured that the conventional encoder-decoder would underperform given longer sentences since it needs to encode the entire input into a fixed-length context vector. The experimental results showed that the new model is significantly better at translating longer sentences, corroborating the conjecture.

The Transformers

[VSP$^+$17] introduced an architecture called ”the Transformer” for sequence modelling tasks. It replaces RNNs with multi-head self-attention mechanisms. Compared to RNNs, self-attention mechanisms have a lower computational complexity as long as the sequence length is smaller than the number of dimensions in the representation of a symbol. Moreover, multi-head self-attention allows much more parallelised computation, and it learns long-range dependencies between words more easily.

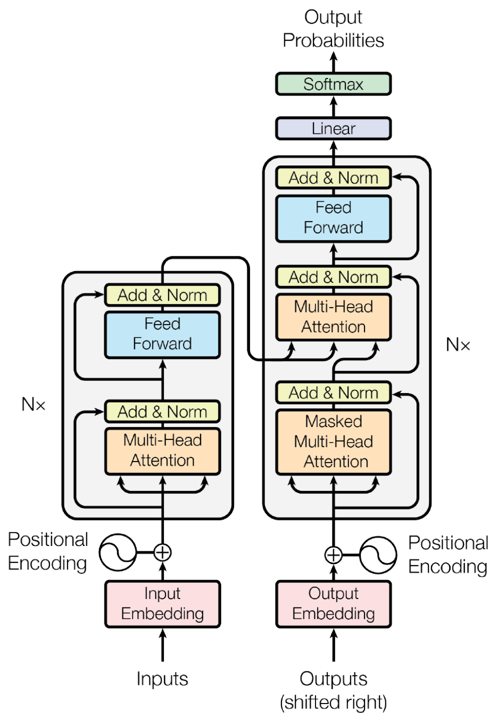

Figure 8: The Transformer architecture. The three inputs of a multi-head attention module are used as key, value and query from left to right in that order [VSP$^+$17]

The Transformer architecture is shown in Figure 8, and it also contains an encoder and a decoder. The encoder maps a sequence of pre-processed input representations $(\mathbf{x}_1, ..., \mathbf{x}_n)$ to intermediate representations $(\mathbf{z}_1, ..., \mathbf{z}_n)$. The decoder takes the intermediate representations together with previously generated outputs to generate the next output symbol, thereby producing the output sequence $(\mathbf{y}_1, ...\mathbf{y}_m)$ one at a time. The encoder consists of a stack of N layers, each having a multi-head self-attention sub-layer followed by a feedforward sub-layer, and the final encoder layer produces $(\mathbf{z}_1, ..., \mathbf{z}_n)$. Similarly, the decoder contains a stack of N layers, but each of them contains an additional multi-head attention sub-layer that takes the output from the encoder. Each of the N layers in the decoder shares the same $(\mathbf{z}_1, ..., \mathbf{z}_n)$ from the encoder, while passing the output of this layer as the other input to the next decoder layer. There are residual connections that skip the attention and feedforward modules for faster learning, as they counteract vanishing gradients in backpropagation.

All the layers mentioned above and the embedding layers have d_{model} dimensions so that the number of dimensions is compatible between layers. That is, every $\mathbf{x}_i$ and $\mathbf{z}_i$ have d_{model} dimensions. Note that although the model takes the entire sequence of inputs at once, a layer works on each input identically (in the context of all inputs) and produces a corresponding output, so we do not need to worry about the sequence lengths. The rest of this subsection explains how that is done.

The input embedding layer pre-processes each input symbol into a meaningful d_{model}-dimensional vector $\mathbf{x}_i$ for later use. However, identical symbols give rise to identical embeddings, so these embeddings do not encode the positions of symbols in the sequence. The Transformer adds information about the relative positions of symbols via positional encodings so that the model can attend to this property. A symbol's positional encoding also has d_{model} dimensions and can be added to the symbol's embedding directly.

The self-attention mechanism computes a new representation of its input sequence with regard to the relations between symbols at different positions. [VSP+17] uses the scaled dot-product attention function for the self-attention mechanisms as well as the attention mechanism across the encoder and the decoder.

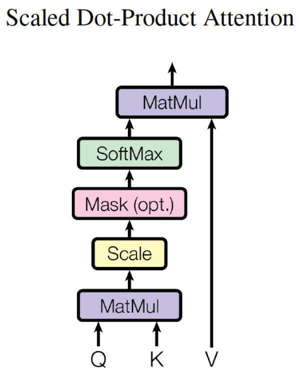

Figure 9: The scaled dot-product attention function. The mask is applied in the self-attention for previous outputs in the decoder, so as to ensure that the resulting representation of the output at position i only depends on outputs at positions less than i [VSP+17]

Figure 9 shows the attention function. It takes a "query" vector, a "key" vector and a "value" vector for each input. The query and the key both have d_k dimensions, the value d_v dimensions. For each input, the mechanism takes the dot product between its query and the key of every input symbol, scaling the results by $1/\sqrt{d_k}$ to avoid exploding values due to high dimensionality, and applies $softmax$ on the resulting values to get weights. The output, which is the new representation of the input, is then the weighted sum of the value vectors of all the input symbols. We can collate all queries as rows in the matrix Q, keys in the matrix K, and values in V, and express the operation on all inputs by

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{5}$$

The outputs are the rows of the resulting matrix.

The multi-head attention (Figure 10) performs h copies (or heads) of the attention function, but rather than d_{model}-dimensional keys, queries and values, it projects the Q, K and V matrices to different subspaces via linear transformations. The linear transformations are expressed by matrices with appropriate dimensions, and the matrices are model parameters to be learnt. [VSP$^+$17] uses transformations such that $d_k = d_v = d_{model}/h$, and the output from all the heads are concatenated and passed through another linear transformation to give outputs of d_{model} dimensions.

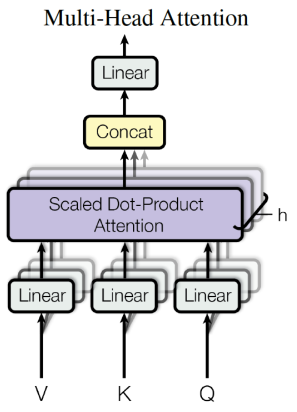

Figure 10: Multi-head attention with h heads [VSP$^+$17]

The Transformer attained state-of-the-art performance in translation tasks, and it is faster to train than the common encoder-decoder architectures that use RNNs. It even outperformed all previously reported models in one of the translations tasks used as an experiment.

Attention Networks in Computer Vision

Attention mechanisms for CNNs, which are commonly used in computer vision, often attend to spatial dimensions and channel dimensions. This section introduces an architecture for each of these two types of mechanisms.

Spatial Attention

[JSZ$^+$15] presented an architecture called "spatial transformers" (Figure 11), which generalises a few previous differentiable spatial attention mechanisms. A spatial transformer consists of three parts, namely the localisation net, the grid generator and the sampler. It transforms each channel of the input feature map U identically. The entire module is differentiable due to the use of a differentiable sampler, so it allows end-to-end training by backpropagation.

The localisation network takes a feature map, $U \in \mathbb{R}^{H \times W \times C}$, where H is the height, W the width, and C the number of channels. It then outputs $\theta = f_{loc}(U)$ as a parameter for the grid generator. That is, f_{loc} is implemented as the localisation network, which can take many forms, such as a convolutional network. The parameterised grid generator T_θ takes the parameter θ and produces a mapping from a location $G_i = (x_i^t, y_i^t)$ in the desired output

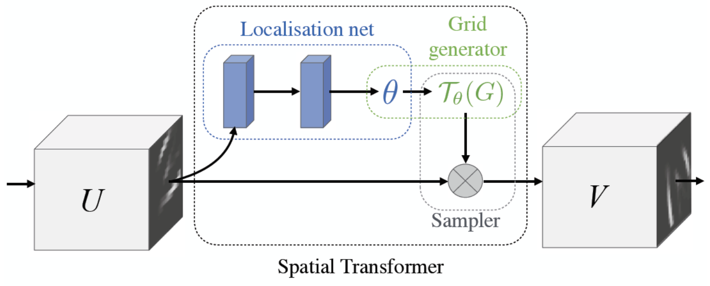

Figure 11: The spatial transformer architecture [JSZ+15]

grid G to a location in the input feature map, (x_i^s, y_i^s), and (x_i^s, y_i^s) serves as the centre of the sampling location for generating (x_i^t, y_i^t) in the next step. The output feature map of the module is $V \in \mathbb{R}^{H' \times W' \times C}$. In each channel $c \in C$, to generate the value at the spatial location (x_i^t, y_i^t) in V, differentiable image sampling applies a kernel centred at (x_i^s, y_i^s) to U, where $(x_i^s, y_i^s) = T_\theta(x_i^t, y_i^t)$. The transformation T_θ can take many forms too, allowing operations like cropping, translation, rotation and skew. By restricting its form, we can also enforce what it attends to. The mechanism only attends to the part of the input that is used. Moreover, by explicitly learning transformations-invariant representations, it ignores the noise in the input due to those transformations, e.g., translation in the case of number recognition.

The mechanism is fast and can fit into any place that has a feature map in a CNN. Stacking many such modules together allows the network to learn more abstract transformations, while applying many copies in parallel enables the network to attend to different objects.

Channel Attention

[HSS18] proposed an architectural unit named the “Squeeze-and-Excitation” (SE) block which recalibrates the attention paid to each channel based on the interdependence between channels.

Figure 12 shows the architecture of an SE block. Let $\mathbf{U} \in \mathbb{R}^{H \times W \times C}$ be output of some previous transformation applied on $\mathbf{X} \in \mathbb{R}^{H' \times W' \times C'}$. For example, the transformation may be a sequence of convolution operations, and $\mathbf{U}$ may be a feature map. An SE block takes $\mathbf{U}$ as the input, performs a squeeze operation and an excitation operation on it to produce weights, and then re-weighs $\mathbf{U}$ in the channel dimensions using the weights.

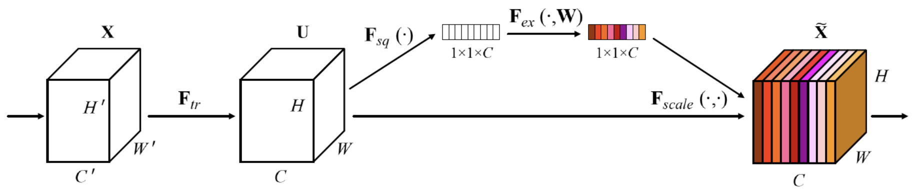

Figure 12: A Squeeze-and-Excitation block [HSS18]

The squeeze operation, $\mathbf{F}_{sq}$, aims to aggregate information within a channel. [HSS18] maps $\mathbf{U}$ to $\mathbf{z} \in \mathbb{R}^C$, with the c-th component of $\mathbf{z}$ simply computed as the average of the $H \times W$ values in channel c of $\mathbf{U}$. The excitation operation, $\mathbf{F}_{ex}$, is implemented as a fully-connected feedforward layer with the ReLU function as the non-linearity, and then another fully-connected feedforward layer followed by the sigmoid function. It computes $\mathbf{s} = \mathbf{F}_{ex}(\mathbf{z}) = \sigma(\mathbf{W}_2 r(\mathbf{W}_1 \mathbf{z}))$, where σ is the sigmoid function, r the ReLU function, and $\mathbf{W}_1$ and $\mathbf{W}_2$ are matrices with appropriate dimensions so that $\mathbf{s} \in \mathbb{R}^C$. The aim is to let the model learn which channel features are more important than others given the input feature map. Finally, the $\mathbf{F}_{scale}$ function scales all $H \times W$ values in channel c of $\mathbf{U}$ by s_c, and the resulting recalibrated matrix is the output.

Conclusion

Attention mechanisms in neural networks have brought about significant improvement in prediction accuracy, and attention in neural networks is still an active research area. There are many attention mechanisms other than the ones mentioned in this essay. For example, mixed attention combines spatial and channel attention mechanisms, hard attention in computer vision completely ignores parts of the input stochastically, and the introspective attention mechanism attends to a neural network's internal state and does both selective reading and selective writing to the state [Gra]. The mechanisms provide yet another perspective from which we can build deep neural networks and understand the outputs. With greater expressive power and more flexibility, the models are likely to learn faster and perform better.

References

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[BSF94] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[CvMG$^+$14] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[FRC10] Simone Frintrop, Erich Rome, and Henrik I Christensen. Computational visual attention systems and their cognitive foundations: A survey. *ACM Transactions on Applied Perception (TAP)*, 7(1):1–39, 2010.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. Adaptive computation and machine learning. Cambridge, Massachusetts, 2016.

[Gra] Alex Graves. Deep learning 7. attention and memory in deep learning. https://www.youtube.com/watch?v=Q57rzaHHO0k&list=PLBnEs0gxis_i0bM1xo_BTEAAeZSYS9FN0&index=4.

[HSS18] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[JSZ$^+$15] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. *Advances in neural information processing systems*, 28:2017–2025, 2015.

[LBH15] Y. LeCun, Y. Bengio, and G Hinton. Deep learning. *Nature*, 521:436–444, 2015.

[LWL$^+$17] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E Alsaadi. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26, 2017.

[PWLK18] Jongchan Park, Sanghyun Woo, Joon-Young Lee, and In So Kweon. Bam: Bottleneck attention module. *arXiv preprint arXiv:1807.06514*, 2018.

[VSP$^+$17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[WJQ$^+$17] Fei Wang, Mengqing Jiang, Chen Qian, Shuo Yang, Cheng Li, Honggang Zhang, Xiaogang Wang, and Xiaoou Tang. Residual attention network for image classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2017.

[WPLK18] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.