10-708: Probabilistic Graphical Models, Spring 2020

14: Deep Sequence Models

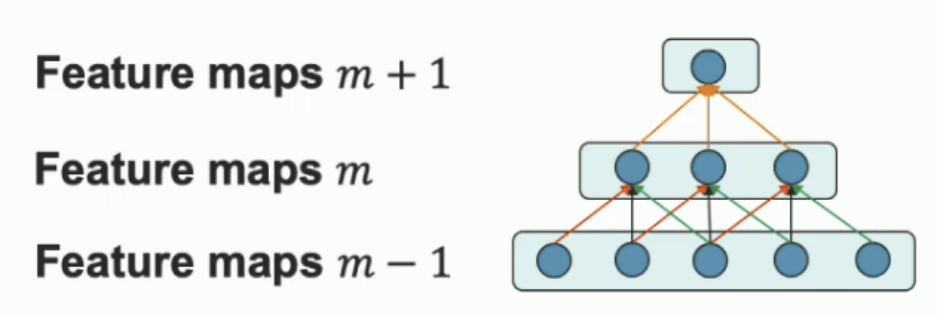
Lecturer: Zhiting Hu *Scribe: Karmesh Yadav, Vivek Pandit, Chirag Pabbaraju, Laavanye Bahl*

This lecture is about four fundamental and popular model architectures.

1 CNN

CNNs are biologically-inspired variants of MLPs that exploit the strong spatial local correlations present in images. The biological concept of the receptive field states that the visual cortex contains a complex arrangement of cells that are sensitive to small sub-regions that are tiled to cover the visual field.

There are three properties of convolutional neural networks:

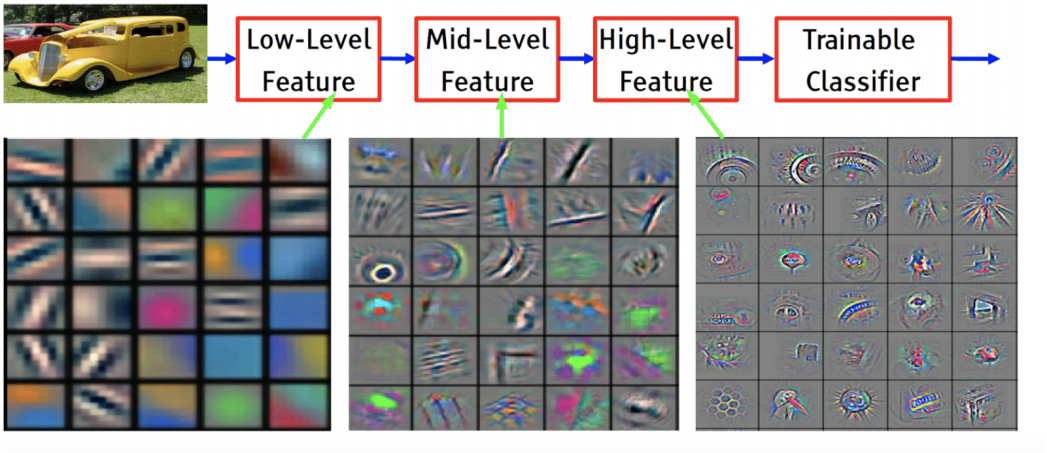


- **Sparse connectivity**
  Each neuron is connected to a subset of neurons from the preceding layer.

- **Shared weights**
  Weights are shared in the form of filters which helps in traversing the image.

- **Increasingly 'global' receptive fields**
  Using the biological analogue, simple cells detect local features while complex ones pool the outputs of simpler cells.

- **Heirarchical Representation Learning**
  Stacking multiple layers can result in lower layers learning low-level features like edges and corners, while upper layers learn high-level representations like textures.



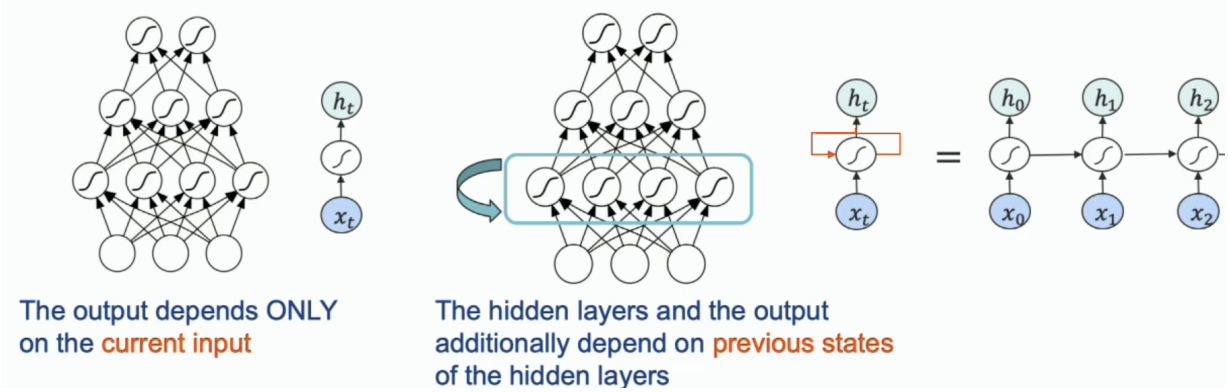
Evolution of ConvNets

1. AlexNet, 8 layers, 2012 [1]
2. VGG, 19 layers, 2015 [9]
3. GoogLeNet, 22 layers, 2014 [18]
4. ResNet, 152 layers, 2015 [6]

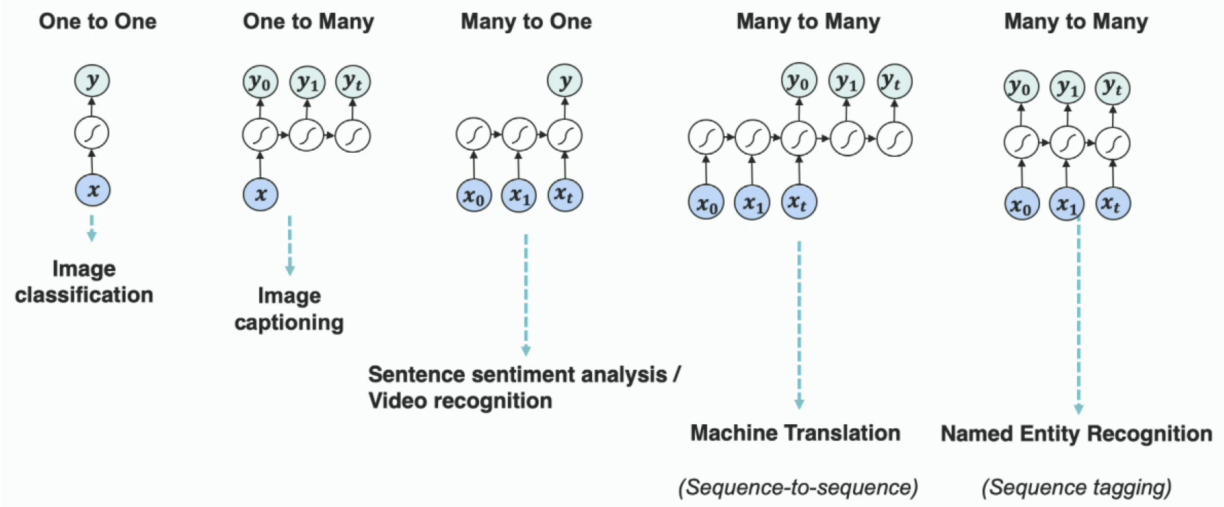
2 RNN

Following are the differences between CNNs and RNNs:

- CNNs are used for spatial modelling. The temporal (or sequential) analogue to the CNN is the RNN.
- RNNs can have a variable number of computation steps unlike CNNs.
- Unlike MLPs and CNNs, RNN outputs depend not only on the current input, but also on the previous states of hidden layers as shown in figure below.

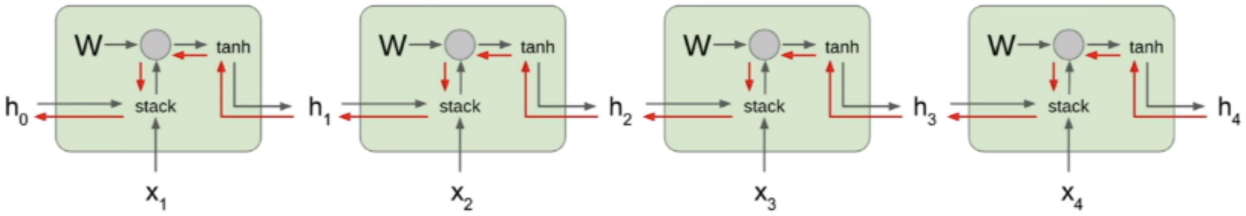


2.1 Various forms of RNNs with applications



2.2 Vanishing/Exploding Gradients in RNNs [3, 14]

Following is an illustration of the flow of computation in an RNN:

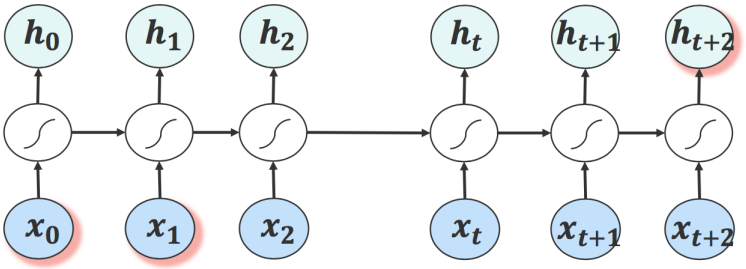


where

$$h_t = \tanh(W^{hh} h_{t-1} + W^{hx} x_t)$$

As we can see, computing the gradient of any loss function w.r.t. say h_0 will involve repeated multiplication of the matrix W along with repeated tanh functions. Thus, if the largest singular value of W is greater than 1, the chained multiplication of W will lead to an exploding gradient w.r.t. h_0. Similarly, if the largest singular value of W is less than 1, this will lead to a vanishing gradient w.r.t. h_0. To remedy the problem of exploding gradients, we can apply gradient clipping i.e. if the norm of the gradient is larger than some predetermined threshold, we can rescale the gradient to have norm equal to the threshold. However, we do not have such suitable ad-hoc fixes for the vanishing gradient problem.

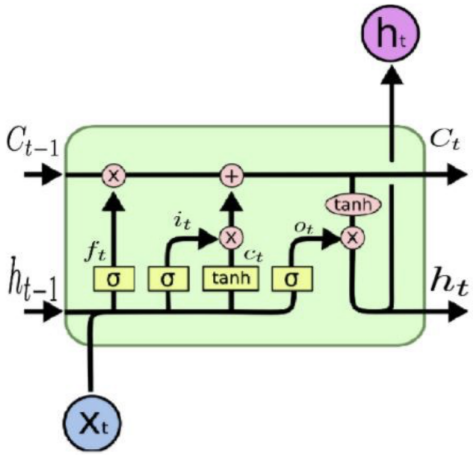
2.3 Long term dependency problem



This optimization issue leads to problems in modelling long-term dependencies in sequences. For example, suppose we are fed the sequence "I live in France and I know -" and our sequence model is tasked to predict the next word. Here, since the word "France" was just a few words past, the gradient information should not have been killed during optimization and the model should have learned to output "French". However, if we instead feed the sequence "I live in France, a beautiful country, and I know -", the gradient information from a state that is further behind would likely have been killed and the model would have difficulty in outputting the desired word.

3 Long Short Term Memory (LSTM) [7]

To alleviate the problems of modelling long-term dependencies due to vanishing/exploding gradients, the LSTM model was introduced in 1997. This variant of RNNs involves gating mechanisms to prevent loss of gradient information over time. Specifically an LSTM cell structure consists of forget gates, input gates and output gates. These gates control the operations of reading, writing and resetting information into the maintained cell state.



Here, x_t denotes the input, h_t denotes the hidden state and c_t denotes the cell state at time step t and σ denotes the sigmoid activation. Also, the pink circles with $+$ and $\times$ denote coordinate-wise addition and multiplication of vectors.

- Forget Gate: This gate decides what must be removed from the existing cell state. Specifically, the forget gate performs the following computation, w.r.t. the cell parameters W_f and b_f. Here, $[h_{t-1}, x_t]$ denotes the concatenation of vectors h_{t-1} and x_t.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

- Input Gate: This gate determines the new information to be written into the cell state. Specifically, the input gate performs the following computation

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

- Update Cell State: Then, we perform the following computation, which updates the outputs of the forget gate and the input gate into the cell state.

$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t$$

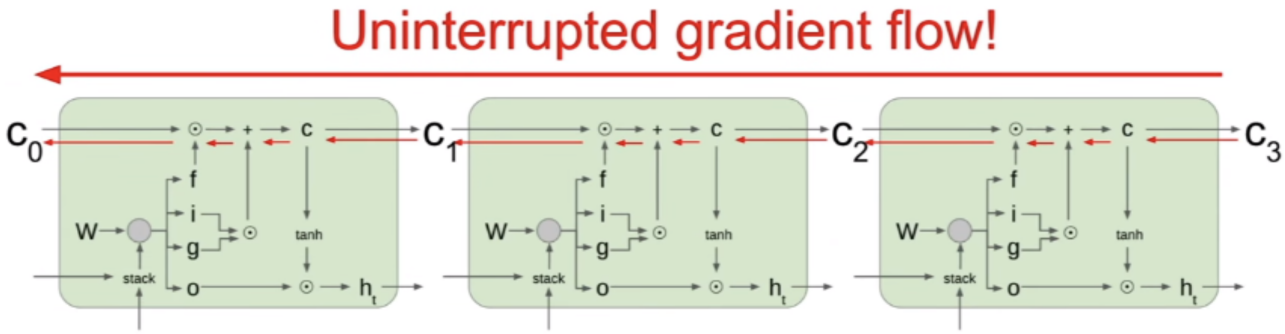
  Specifically, the first product removes unnecessary information from the cell state and the second product introduces new information into the cell state scaled appropriately by the values in $\tilde{C}_t$.

- Output Gate: Finally, we perform the following computation to decide what to output (h_t) from the current cell state. Specifically, we perform the following computation

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \times \tanh(C_t)$$

  Here, the sigmoid controls the parts of the cell state that the cell outputs.

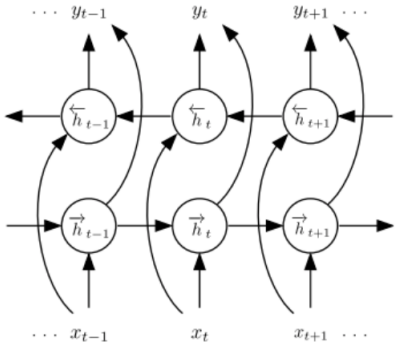
3.1 Backpropagation in LSTM



With this gating mechanism, it can be seen that while backpropagating gradients, repeated multiplication of a matrix W is avoided. Further, different values of the forget gate activations at each time step redress the problem of exploding/vanishing gradients. A detailed working out of this can be found at [2].

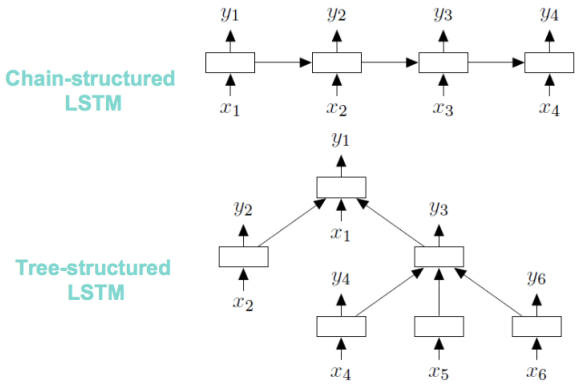
4 Other RNN variants

4.1 Bi-directional RNNs [5]



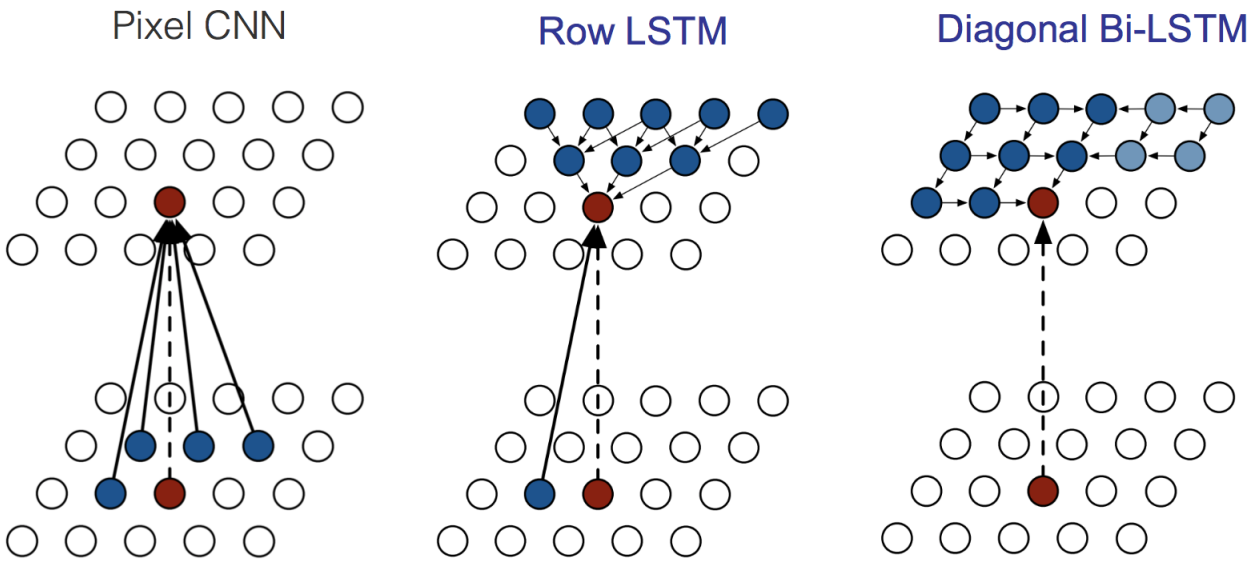
- In bi-directional RNNs, the hidden state is the concatenation of both forward and backward hidden states
- This allows the hidden state to capture both past and future information

4.2 Tree-structured RNNs [19]



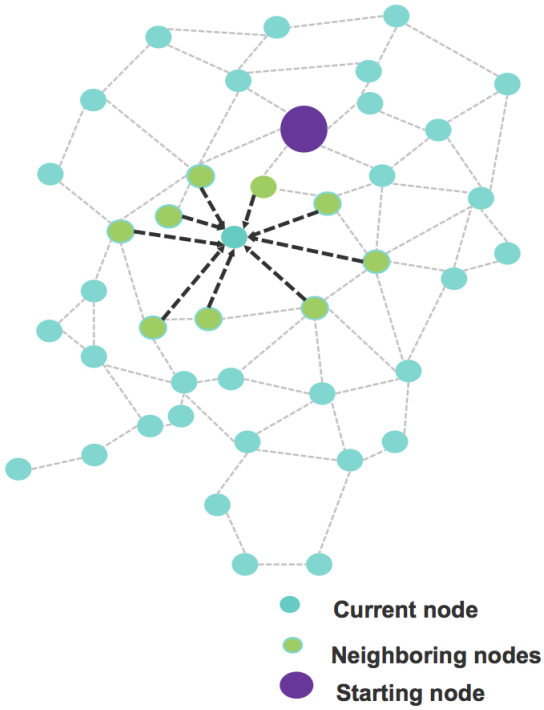
- In tree-structured RNNs, the hidden states condition on both an input vector and the hidden states of arbitrarily many child units
- Thus, standard LSTMs are in fact special cases of tree-LSTMs where each internal node has exactly one child.

4.3 RNNs for 2-D sequences [13]



- Instead of using CNNs for extracting features in 2-D data, we can use LSTMs over the rows/diagonals of the data as shown above to extract quality features

4.4 RNNs for Graph structures [11]



- We can also have an LSTM operate on graph-structured data e.g. in image segmentation applications, we can model the pixels to be nodes in a graph where the features extracted at each node also depend on the features extracted for neighbouring nodes in the graph

5 Attention

Attention mechanisms are techniques used to choose which feature to focus on, in the input data. They have improve performance in tasks such as machine translation, image captioning and speech recognition, to name a few.

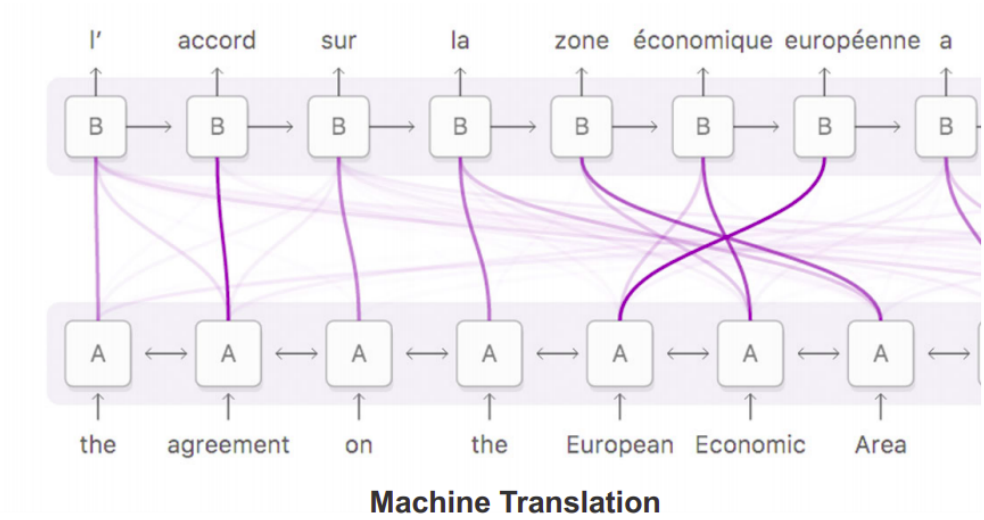


Figure 1: Example of attention used in machine translation

They are used primarily to handle for long-range dependencies and hence dealing with vanishing gradients, a problem typically seen in RNNs. By allowing for fine-grained localized representations of portions of data, like patches in images or words in sentences, attention improves model interpretability.

5.1 Attention Computation

Attention can be computed for a machine translation task using the following procedure:

- Encode each token in the input sentence into a key vector. This corresponds to what is available in the input
- When decoding, compare the query vector with the encoder states, and generate alignment scores corresponding to each key vector. Query vector corresponds to what is required to be generated, depending on what tokens have already been generated.
- Compute the attention weights by normalizing the alignment scores.
- Treat the encoder states as value vectors and compute a weighted sum, using the attention weights.
- Use this weighted sum in the decoder to generate the next token.

5.2 Attention Variants

There are a number of different alignment score functions used to generate alignment scores. Some of these are shown in the table below:

Soft and hard attention are variants of attention that respectively use deterministic and stochastic methods in computing the weights for each token. The computation described above is for soft attention. Instead of

Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

Figure 2: Popular alignment score functions

using the attention weights to compute a weighted average, hard attention uses these as probabilities and samples from the corresponding features using this distribution. A comparison for attention used on images can be illustrated below. Notice how soft attention can be diffuse, and assign nonzero weights to significant weights to large portions of the image at times, while hard attention focuses on a particular equally-sized part of the image in each case. Soft attention is presently the more popular variant, primarily because it allows for simpler backpropagation in the network.

5.3 Applications in Computer Vision

Attention are widely used in computer vision tasks such as Image captioning, Image paragraph generation, etc. [10] introduce a technique using attention over both image and text. The entire pipeline can be seen in Fig. 3, and proceeds as follows:

- The image is first segmented into semantic regions, and each region is captioned with local phrases.
- Attention is applied on both visual regions and the text phrases and the resulting features fed to a generator which generates sentences using hierarchical text generation.
- These sentences are then fed to a sentence discriminator and a recurrent topic-transition discriminator for assessing sentence plausibility and topic coherence respectively
- A paragraph description corpus is adopted to provide linguistic knowledge about paragraph generation, which depicts the true data distribution of the discriminators

Another application is Image captioning. [21] introduce a technique, described as follows:

- The entire image is given as input to a Convolutional Neural Network (CNN).
- Multiple feature maps are used to extract visual features from the images.
- RNNs are used to attend to the images. Both hard and soft attention is used. Hard attention samples regions directly from the image. Soft attention computes the expected attention over the entire image.
- LSTM use the attention weights to generate captions word by word.

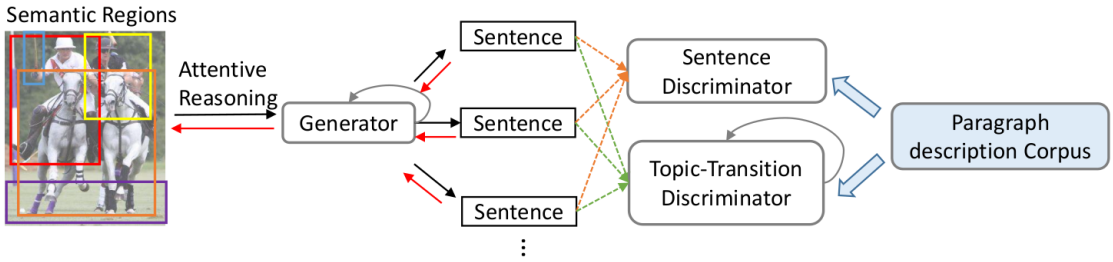


Figure 3: Pipeline for Image Paragraph Generation

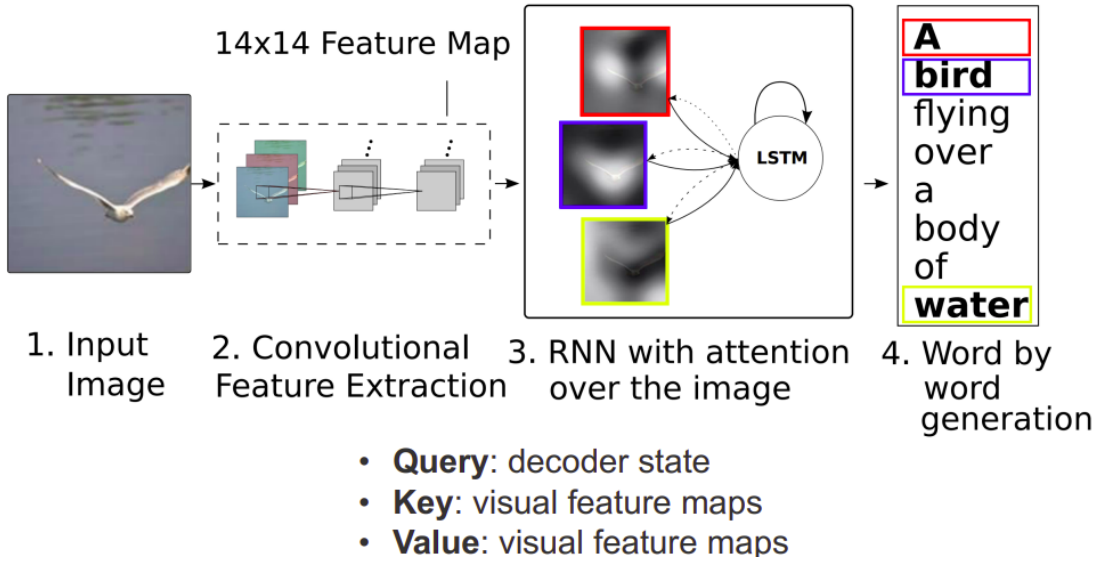


Figure 4: Pipeline for Image captioning

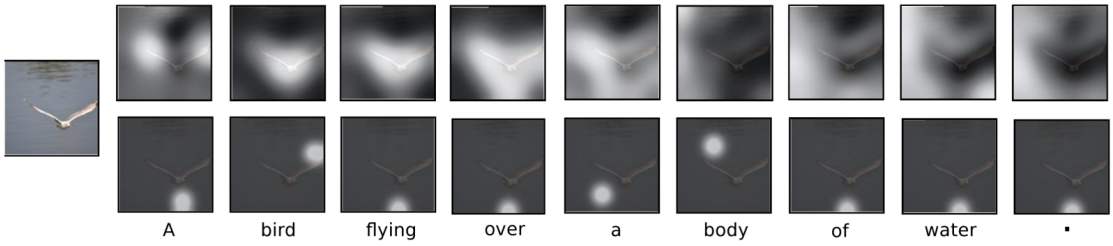


Figure 5: Above: Soft Attention, Below: Hard Attention in Image Captioning

6 Transformers: Multi-Headed Attention [20]

Recently, [20] introduced a novel, non-recurrent neural network architecture composed purely of self-attention called the Transformer. The Transformer has attained state-of-the-art results in many sequence-to-sequence natural language processing tasks, such as machine translation. Since the Transformer architecture lacks recurrent networks, it can be more amenable to learning long-range dependencies over sequences while also

improving training and inference speed.

As shown below, the Transformer employs multi-headed self-attention, in which multiple attention layers run in parallel. Intuitively, this can enable different heads to focus on different parts of the sequence.

Formally, multiple heads of Queries Q, Keys K of dimension d_k, and Values V can be packed together into separate matrices to allow for attention to be computed efficiently using the scaled dot-product variant, which they suggest prevents diminished gradients during training:

$$\text{Attention } (Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V \tag{1}$$

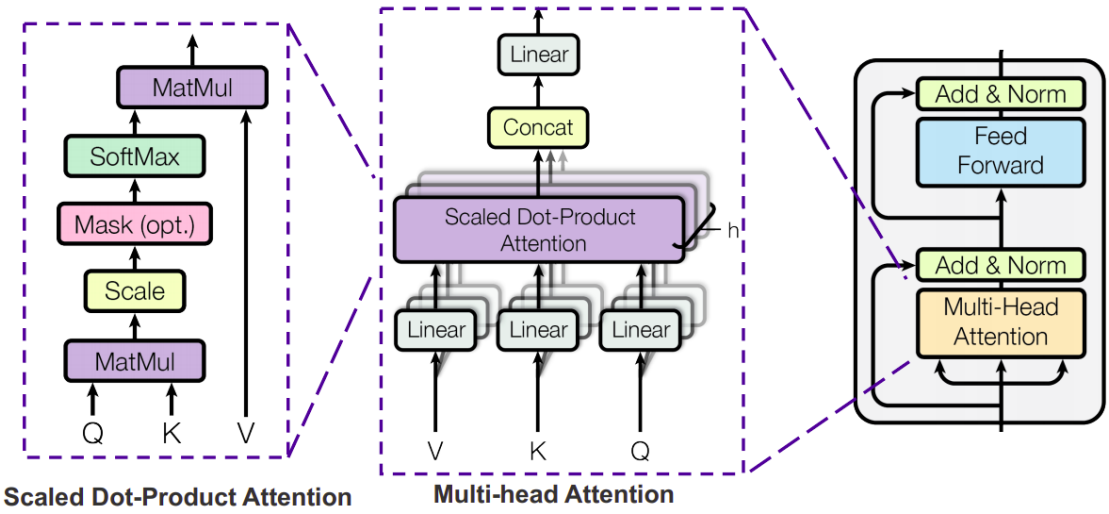


Figure 6: Multi headed attention

Multi-headed attention can then jointly attend to information from multiple different representations at different positions by:

$$\begin{aligned} \text{MultiHead } (Q, K, V) &= \text{ Concat } (\text{head}, , \ldots, \text{ head }_h) W^O \\ \text{where } head_i &= \text{Attention } \left(QW_i^Q, KW_i^K, VW_i^V\right) \end{aligned} \tag{2}$$

6.1 Unsupervised Language Model Pre-training

Recently, language representation via unsupervised language model pre-training has revolutionized the field of natural language processing. Parameters learned during the training of large language models using self-supervision have been shown to be extremely effective when transferred to other NLP prediction tasks. [16] [15] [4] [17]. Since language modeling requires the resolving of long-term dependencies, hierarchical relations, and sentiment, it can be seen as an ideal source task for transfer learning in NLP. [8]

ELMo [15] introduce deep contextualized word representations, which are learned functions of the internal states of a deep bidirectional LSTM language model trained to predict both the next word in a sentence given its history and the previous word in a sentence given its future words. These contextualized representations can then be frozen and used as embeddings for other downstream tasks like question answering, textual

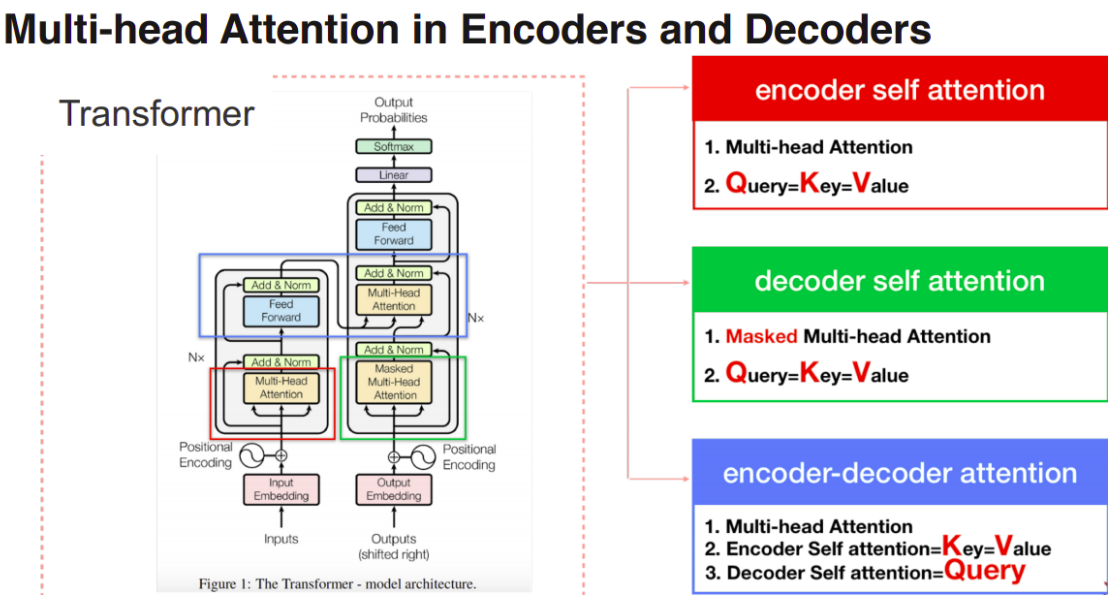


Figure 7: Pipe Line for Multi headed attention in encoders and decoders

entailment, and sentiment analysis. Instead of just transferring word embeddings for a new task, Howard and Ruder's (2018) ULMFiT aims to transfer the language model itself for new tasks. Particularly, the authors train an AWD-LSTM [12] language model on 103 million words of Wikipedia data, fine-tune on a smaller amount of task-specific data using different learning rates for different layers of the model, and add a final classifier on the end of the network for the target task. Thus, while ELMo requires task-specific architectures when transferring to new tasks, ULMFiT simply adds a classifier on top of the language model to obtain state-of-the-art results on six benchmarks. OpenAI then adapted this method (dubbed Generative Pre-training or GPT) to work with the popular Transformer [20] architecture, an auxiliary language modeling loss during fine-tuning to obtain even better results, and adaptation to more difficult tasks such as machine translation. [16] Just this year, OpenAI followed up their work with GPT2, which is the highest performing language model to date. They use a similar but much higher capacity model to GPT as they find that capacity improves performance log-linearly. Due to its high, human-like language generation performance, they controversially decided to not release their largest model: a 1.5B parameter Transformer trained on 8 million documents of web text.

7 BERT: Pre-trained Text Representation Model

In BERT (Bidirectional Encoder Representations from Transformers), Devlin et al. (2018) [4] used a bidirectional Transformer architecture to obtain improved contextualized word embeddings in an extension to OpenAI GPT [16]. Before BERT, there were some conventional word embedding methods like Word2Vec and Glove, which had a pretrained matrix with each row denoting the embedding vector vector of a particular word. The matrix was pretrained with a large corpus of text using the above mentioned methods.

The paper introduces two new objectives to adapt the traditional task of predicting the next word in language modeling to benefit from bidirectionality. After encoding each word in a given sentence into a contextualized representation, they have the model both predict a random masked word from the original sentence and perform a binary classification on two sentences to identify whether or not one sentence follows the other.

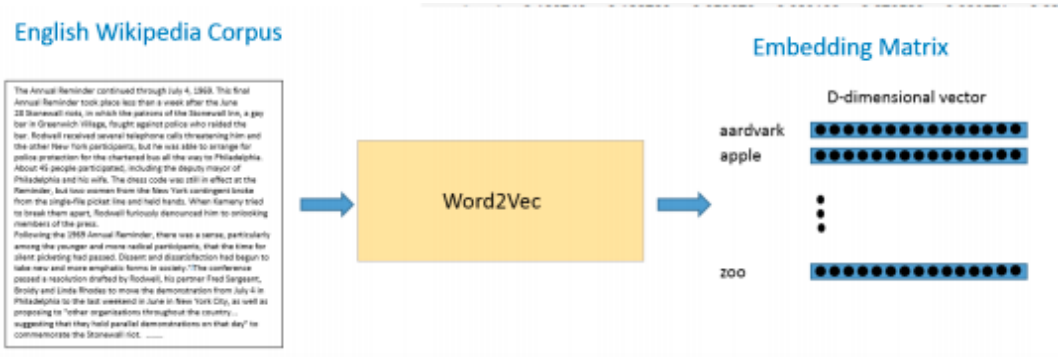


Figure 8: Word2Vector

Although the masked language model objective requires more pre-training steps since each prediction is no longer sequential, they find that performance increases over the traditional objective are immediate. They find that the next sentence classification objective is particularly beneficial to tasks like natural language inference and question answering since they require multi-sentence reasoning.

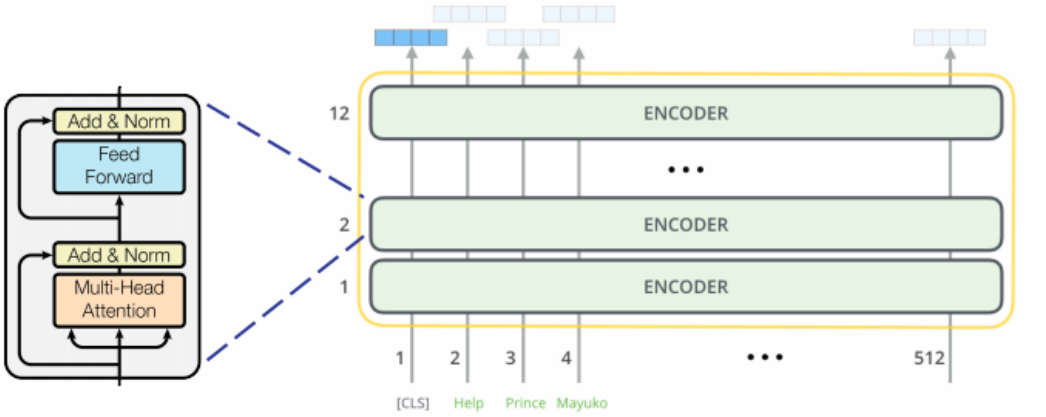


Figure 9: The BERT encoder

In the paper, the authors use both Wikipedia data (2.5B words) and free online eBook data (800M words) for training a Transformer encoder with hundreds of millions of parameters. An ablation study on model size empirically shows that extreme model sizes lead to large improvements on even very small scale tasks, provided that the model has been sufficiently pre-trained. Although training takes many more steps to converge than a traditional language model objective, BERT, with only single output layer modifications, performs at the state-of-the-art for eleven NLP tasks including sentiment, question answering, and natural language inference. The authors report that they were able to train BERT in just 4 days on 4 Google TPU pods (256 TPU Chips). In comparison to this, it will take 5.3 days to train it on a cluster having 64 V100 GPU and 99 days on a standard 4 GPU desktop with an RTX 2080Ti.

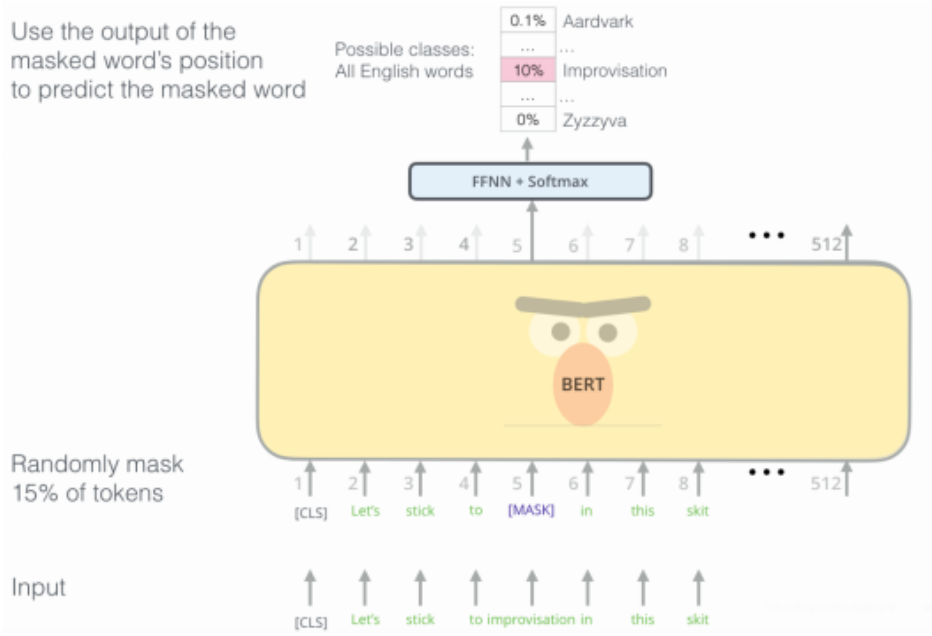


Figure 10: The masked language model objective

References

[1] G.E. Hinton. A. Krizhevsky, I. Sutskever. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems 25, pp. 1097–1105*, 2012.

[2] Nir Arbel. How lstm networks solve the problem of vanishing gradients. `https://medium.com/datadriveninvestor/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577`, Feb 2020.

[3] 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.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[5] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[8] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*, 2018.

[9] A. Zisserman K. Simonyan. Very deep convolutional networks for large-scale image recognition. 2014.

[10] Xiaodan Liang, Zhiting Hu, Hao Zhang, Chuang Gan, and Eric P Xing. Recurrent topic-transition gan for visual paragraph generation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3362–3371, 2017.

[11] Xiaodan Liang, Xiaohui Shen, Jiashi Feng, Liang Lin, and Shuicheng Yan. Semantic object parsing with graph lstm. In *European Conference on Computer Vision*, pages 125–143. Springer, 2016.

[12] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017.

[13] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.

[14] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.

[15] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[16] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf*, 2018.

[17] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

[18] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[19] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

[20] 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.

[21] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.