

# UC Berkeley · CSW182 | [Deep Learning]

## Designing, Visualizing and Understanding Deep Neural Networks (2021)

### CSW182 (2021) · 课程资料包 @ShowMeAI



视频

中英双语字幕



课件

一键打包下载



笔记

官方笔记翻译



代码

作业项目解析



视频 · B 站 [ 扫码或点击链接 ]

<https://www.bilibili.com/video/BV1Ff4y1n7ar>



课件 & 代码 · 博客 [ 扫码或点击链接 ]

<http://blog.showmeai.tech/berkeley-csw182>

Berkeley 循环神经网络 可视化 梯度策略  
Q-Learning 风格迁移 模仿学习 元学习  
计算机视觉 机器学习基础 生成模型 卷积网络

Awesome AI Courses Notes Cheatsheets 是 [ShowMeAI](#) 资料库的分支系列, 覆盖最具知名度的 **TOP50+** 门 AI 课程, 旨在为读者和学习者提供一整套高品质中文学习笔记和速查表。

**点击**课程名称, 跳转至课程**资料包**页面, **一键下载**课程全部资料!

机器学习	深度学习	自然语言处理	计算机视觉
Stanford · CS229	Stanford · CS230	Stanford · CS224n	Stanford · CS231n
# Awesome AI Courses Notes Cheatsheets · 持续更新中			
知识图谱	图机器学习	深度强化学习	自动驾驶
Stanford · CS520	Stanford · CS224W	UCBerkeley · CS285	MIT · 6.S094



微信公众号

资料下载方式 2: 扫码点击**底部菜单栏**  
称为 **AI 内容创作者?** 回复 [ 添砖加瓦 ]

This discussion covers Attention Mechanisms and Transformers.

## 1 Attention Mechanisms

For many NLP and visual tasks we train our deep models on, features appear on the input text/visual data often contributes unevenly to the output task. For example, in a translation task, not the entirety of the input sentence will be useful (and may even be confusing) for the model to generate a certain output word, or not the entirety of the image contributes to a certain sentence generated in the caption.

While some RNN architectures we previously covered possess the capability to maintain a memory of the previous inputs/outputs, to compute output and to modify the memory accordingly, these memory states need to encompass information of many previous states, which can be difficult especially when performing tasks with long-term dependencies.

Attention mechanisms were developed to improve the network's capability of orienting perception onto parts of the data, and to allow random access to the memory of processing previous inputs. In the context of RNNs, attention mechanisms allow networks to not only utilize the current hidden state, but also the hidden states of the network computed in previous time steps as shown in Figure 5.

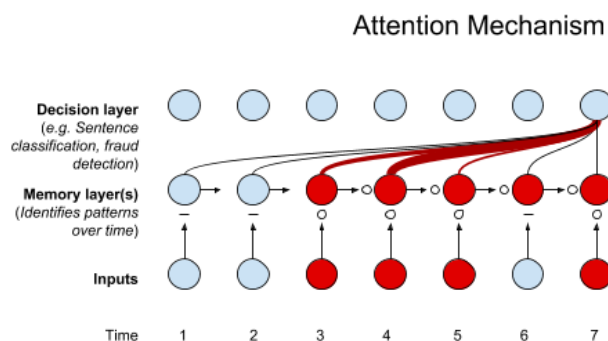


Figure 1: Attention Mechanism Illustrated

### 1.1 Luong Attention

The Luong attention is one of the commonly used attention mechanism in Neural Machine Translation, which is trained on the task of translating text from source to target language. At each time-step of decoding, the Luong Attention computes a set of alignment (attention) weights  $a_l$  based on alignment scores and use this to augment the computation of output probabilities in the translation task.

As shown in Figure 2, we have our encoding states (RNN hidden states when passed a source language into the model first)  $h_t$ . At decoding time, we have the original hidden states  $h_l$  initially computed with the previous output token and hidden states.

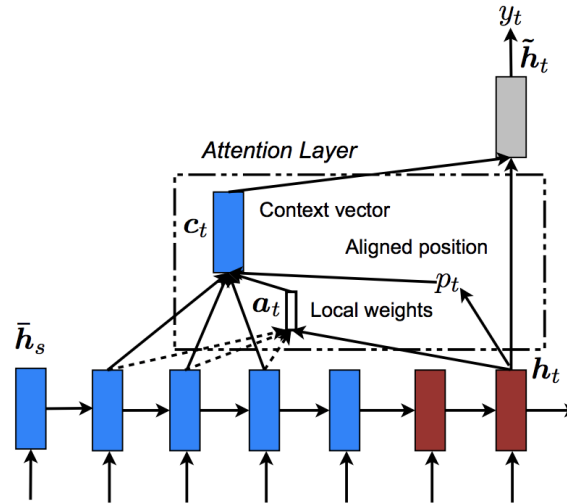


Figure 2: Luong Attention

With  $h_l$  and each  $h_t$  we compute an alignment score <sup>1</sup>, then take a Softmax to obtain attention weights from the alignment scores.

With these attention weights, we compute a context vector  $c_t$ , which is a weighted sum of  $h_s$ . As we get  $c_t$ , we compute a transformation  $\tilde{h}_t = \tanh(W_f[h_t; c_t])$  applied on the concatenation of the context vector and the original hidden state. This is then used to compute the final output.

The motivation for this kind of machine translation system was to treat the encoder (the blue part) as like a memory which we can access later during the decoding process (colored red). Most of the early neural machine translation systems ran the encoder with the input sentence to get a hidden state, and then that hidden state was the input to the decoder which needed to generate the resulting sentence. With this model, we are able to use the other hidden state outputs from the encoder, not just the last one.

## 1.2 Self-Attention in Transformer Networks

Self attention is an attention mechanism introduced in the Transformer architecture which undergoes similar procedures as the Luong attention. The first step of the attention is to compute  $Q, K, V$  using different transformations from the original input embedding as shown in Figure 3.

<sup>1</sup>the original paper proposed multiple alignment scores, the general one they propose is  $h_l^\top W_a h_t$ , another simple 'location' based attention is just computed using the decoding hidden state at time  $t$ :  $W_a h_l$

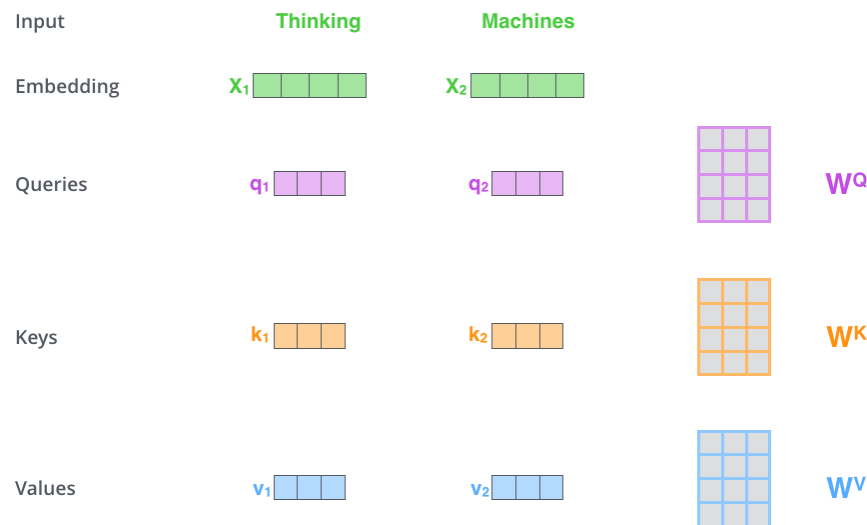


Figure 3: Computing K, Q, V from input embeddings in a Transformer Network

Then, using Q and K, we can compute a dot product as the 'score' of K for Q as shown in Figure 4. Intuitively, Q is the querying term that you would like to find. Its relations for each corresponding K and V pairs (key-value) pairs, can be computed using the key. Note that this dot product is computed across various time steps by matrix multiplication. So we get a score for each K for each Q. We then use a Softmax function to get our attention weights.

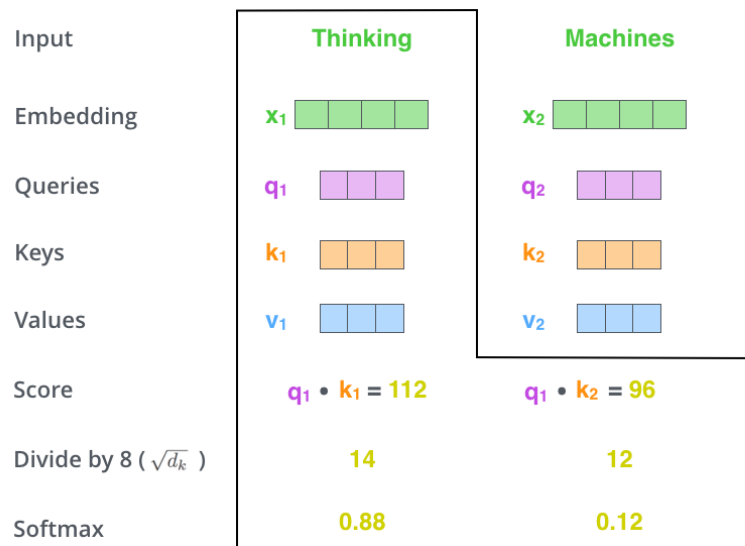


Figure 4: Computing Attention Scores from K, Q, V

Finally, using these weights, we can compute our weighted sum by multiplying the weights with the values. Comparing to the Luong attention, query is analogous to the original  $h_l$ , key and query are analogous to the original  $h_t$ .

#### Problem: Attention in RNNs

Explain how we incorporate self-attention into an RNN model at a high-level.

#### Problem: Generalized Attention in Matrix Form

Consider a form of attention that matches query  $q$  to keys  $k_1, \dots, k_t$  in order to attend over associated values  $v_1, \dots, v_t$ .

If we have multiple queries  $q_1, \dots, q_l$ , how can we write this version of attention in matrix notation?

#### Problem: Justifying Scaled Self-Attention

In practice, Transformers use a Scaled Self-Attention. Suppose  $q, k \in \mathbb{R}^d$  are two random vectors with  $q, k \sim \mathcal{N}(\mu, \sigma^2 I)$ , where  $\mu \in \mathbb{R}^d$  and  $\sigma \in \mathbb{R}^+$

1. Define  $\mathbb{E}[q^\top k]$  in terms of  $\mu, \sigma, d$
2. Define  $\text{Var}(q^\top k)$  in terms of  $\mu, \sigma, d$
3. Let  $s$  be the scaling factor on the dot product. We would like  $\mathbb{E}[q^\top k/s]$  to scale linearly with  $d$ . What should  $s$  be in terms of  $\mu, \sigma, d$
4. Briefly explain what would happen to the variance of dot product if  $s = 1$ .

### 1.3 Tutorials on Attention Networks and Eager Execution on Tensorflow

Here is a tutorial that we recommend you run through in RNNs with `tf.eager` and `keras`: [https://colab.research.google.com/github/tensorflow/tensorflow/blob/master/tensorflow/contrib/eager/python/examples/nmt\\_with\\_attention/nmt\\_with\\_attention.ipynb](https://colab.research.google.com/github/tensorflow/tensorflow/blob/master/tensorflow/contrib/eager/python/examples/nmt_with_attention/nmt_with_attention.ipynb)

## 2 Transformers

At a high-level, transformers consist of the Transformer Encoder and Transformer Decoders.

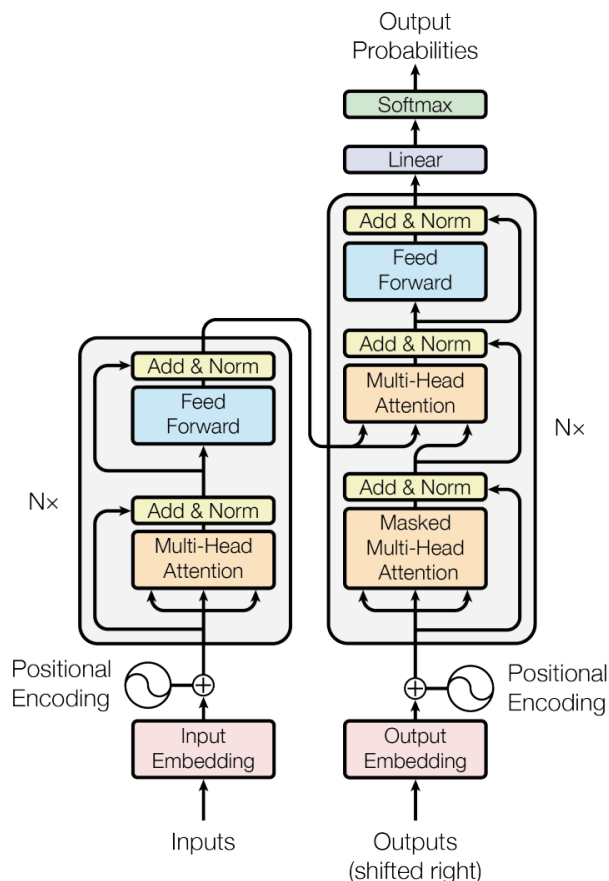


Figure 5: Overview of Transformer architecture

Both operate similarly, except the Transformer Decoder takes  $x_{target}$  as input, but Transformer Encoder takes in  $x_{source}$  as input. In addition, there are several differences in cross-attention and self-attention operations. In particular, transformers are novel in that they add,

- Positional Encoding: Addresses lack of sequence information
- Multi-headed Attention: Allows querying multiple positions at each layer
- Non-linearities
- Masked Decoding: Prevent attention lookups into the future

### 2.1 Notations

To ensure a level of clarity, we will let  $B$  be the batch size,  $L_{source}$  represent the source sequence length,  $L_{target}$  be the target sequence length,  $D$  represent the model hidden dimension and  $H$  represent the number of attention heads.

In particular, transformers receive two sequences as input. The first is  $x_{source} \in \mathbb{Z}^{B \times L_{source}}$  and the second is  $x_{target} \in \mathbb{Z}^{B \times L_{target}}$ . These are integer tensors, and each integer represents a word or token.

## 2.2 Transformer Encoders

**Input & Positional Embedding** The source tensor is embedded into the model hidden dimension, and produces a tensor  $X_{source} \in \mathbb{R}^{B \times L_{source} \times D}$ . We then add a positional encoding that differs for each sequence position in order to enable the model to differentiate the positions in the sequence. In general, we need this information since position of words in a sentence carries information.

**Encoder Attention** The Encoder Attention is self-attention. Specifically, in Transformer networks, we use the Scaled QKV Attention (not covered explicitly in lecture). In other words, we would like to build a representation of a single sequence such that every position in the sequence has information about every other position in the sequence. In particular, to enable this, we will use the Query-Key-Values (QKV) Attention. Our queries, keys and values will be tensors in  $X_{source} \in \mathbb{R}^{B \times L_{source} \times D}$  and weight matrices will be  $W_Q, W_K, W_V \in \mathbb{R}^{D \times D}$ . Ultimately, we will retrieve,

$$\begin{aligned}Q &= X_{source} W_Q \\K &= X_{source} W_K \\V &= X_{source} W_V\end{aligned}$$

Using  $Q, K, V$ , we will compute the attention scores (tensor in  $\mathbb{R}^{B \times L_{source} \times L_{source}}$ ). For each element in the batch, each entry  $i, j$  in the matrix would be  $\frac{q_i^\top k_j}{\sqrt{D}}$  for scaled dot product attention. Alternatively, we can compute,  $\frac{QK^\top}{\sqrt{D}}$ . To produce weights over each position in the sequence, we want each score to sum to one over the keys  $K$ . To accomplish this, we take a softmax update over the last dimension of the attention scores. Then, to produce the attention update, we multiply these attention weights by our values  $V$ ,

$$C_{update} = \text{softmax} \left( \frac{QK^\top}{\sqrt{D}} \right) V$$

where  $C_{update} \in \mathbb{R}^{B \times L_{source} \times D}$

One of the key changes in Transformers is the *multi-headed attention* mechanism. To turn it into multi-headed attention, we can take any such update matrices and reshape and permute the matrix from shape  $B \times L_{source} \times D$  to  $B \times H \times L_{source} \times \frac{D}{H}$ .

We finally consider padding. In general, we operate on a batch of  $B$  sequences, but these sequences may not be the same length. We pad each sequence to  $L_{source}$ . To prevent our model from paying attention to padded positions, we add  $-\infty$  to attention scores prior to the Softmax of any position that should be ignored.

**Feedforward Layer** The feedforward layer applies linear transformation to each position, apply a nonlinear activation, then applies a second linear transformation.

## 2.3 Transformer Decoder

**Masked Decoder Self-Attention** Masked decoder self-attention is the same as encoder self-attention, but with different masking. In particular, we would like every position to pay attention to all previous positions, but not future positions. To achieve this, we set attention score to  $\frac{q_i^\top k_j}{\sqrt{D}}$  if  $i \leq j$  and  $-\infty$  otherwise.

**Encoder-Decoder Attention** Encoder-Decoder attention operated similarly as well, except that we have two sequences: (1) generate queries and (2) generate keys-values. Hence, we let  $Q = X_{target} W_Q, K = X_{source} W_K, V = X_{source} W_V$ , where  $X_{source}$  is the output of the transformer encoder on the source sequences.

### Problem: Machine Translation

1. What is the reason for positional encoding? How is it typically implemented?
2. What is the advantage of multi-head attention? Give some examples of structures that can be found using multi-head attention
3. For input sequences of length  $M$  and output sequences of length  $N$ , what are the complexities of (1) Encoder Self-Attention (2) Decoder-Encoder Attention (3) Decoder Self-Attention. Further let  $k$  be the hidden dimension of the network
4. Do activation of the encoder depend on decoder activation? How much additional computation is needed to translate a source sequence into a different target language, in terms of  $M$  and  $N$ ?

## 2.4 Why Transformers

In general, transformers are good for long-range connections, are easy to parallelize and transformers can be made much deeper than RNNs. On the other hand, attention computations are complex to implement and computations take  $\mathcal{O}(n^2)$  time.

However, in practice, it turns out the benefits vastly outweigh the downsides, and transformers work better than RNNs and LSTMs in many cases.



# UC Berkeley · CSW182 | [Deep Learning]

## Designing, Visualizing and Understanding Deep Neural Networks (2021)

### CSW182 (2021) · 课程资料包 @ShowMeAI



视频

中英双语字幕



课件

一键打包下载



笔记

官方笔记翻译



代码

作业项目解析



视频 · B 站 [ 扫码或点击链接 ]

<https://www.bilibili.com/video/BV1Ff4y1n7ar>



课件 & 代码 · 博客 [ 扫码或点击链接 ]

<http://blog.showmeai.tech/berkeley-csw182>

Berkeley 循环神经网络 可视化 梯度策略  
Q-Learning 风格迁移 模仿学习 元学习  
计算机视觉 机器学习基础 生成模型 卷积网络

Awesome AI Courses Notes Cheatsheets 是 [ShowMeAI](#) 资料库的分支系列, 覆盖最具知名度的 **TOP50+** 门 AI 课程, 旨在为读者和学习者提供一整套高品质中文学习笔记和速查表。

点击课程名称, 跳转至课程**资料包**页面, **一键下载**课程全部资料!

机器学习	深度学习	自然语言处理	计算机视觉
Stanford · CS229	Stanford · CS230	Stanford · CS224n	Stanford · CS231n
# Awesome AI Courses Notes Cheatsheets · 持续更新中			
知识图谱	图机器学习	深度强化学习	自动驾驶
Stanford · CS520	Stanford · CS224W	UCBerkeley · CS285	MIT · 6.S094



微信公众号

资料下载方式 2: 扫码点击**底部菜单栏**  
称为 **AI 内容创作者?** 回复 [ 添砖加瓦 ]