Probabilistic Graphical Models

Deep Sequence Models

Zhiting Hu
Lecture 14, March 2, 2020

Reading: see class homepage
Overview: Deep Learning & Generative Models

- 2/19 Lecture 11 Statistical and Algorithmic Foundations of Deep Learning
- 2/24 Lecture 12 Deep generative models (part 1)
- 2/26 Lecture 13 Deep generative models (part 2)
- 3/2 Lecture 14 Deep sequence models
- 3/4 Lecture 15 A unified view of deep generative models
Overview: Deep Learning & Generative Models

- 2/19 Lecture 11 Statistical and Algorithmic Foundations of Deep Learning
- 2/24 Lecture 12 Deep generative models (part 1)
- 2/26 Lecture 13 Deep generative models (part 2)
- 3/2 Lecture 14 Deep sequence models
- 3/4 Lecture 15 A unified view of deep generative models
Outline

- Convolutional Networks (ConvNets)
- Recurrent Networks (RNNs)
  - Long-range dependency, vanishing gradients
  - LSTM
  - RNNs in different forms
- Attention Mechanisms
  - (Query, Key, Value)
  - Attention on Text and Images
- Transformers: Multi-head Attention
  - Transformer
  - BERT
Outline

- Convolutional Networks (ConvNets)
- Recurrent Networks (RNNs)
  - Long-range dependency, vanishing gradients
  - LSTM
  - RNNs in different forms
- Attention Mechanisms
  - (Query, Key, Value)
  - Attention on Text and Images
- Transformers: Multi-head Attention
  - Transformer
  - BERT
Convolutional Networks (ConvNets)

- Biologically-inspired variants of MLPs [LeCun et al. NIPS 1989]
  - Receptive field [Hubel & Wiesel 1962; Fukushima 1982]
    - Visual cortex contains a complex arrangement of cells
    - These cells are sensitive to small sub-regions of the visual field
  - The sub-regions are tiled to cover the entire visual field

Exploit the strong spatially local correlation present in natural images
Convolutional Networks (ConvNets)

- Sparse connectivity
- Shared weights
- Increasingly “global” receptive fields
  - simple cells detect local features
  - complex cells “pool” the outputs of simple cells within a retinotopic neighborhood.

Feature maps $m + 1$

Feature maps $m$

Feature maps $m - 1$
Convolutional Networks (ConvNets)

- Hierarchical Representation Learning [Zeiler & Fergus 2013]
Evolution of ConvNets

AlexNet, 8 layers
- 11x11 conv, 96, /4, pool/2
- 5x5 conv, 256, pool/2
- 3x3 conv, 384
- 3x3 conv, 384
- 3x3 conv, 256, pool/2
- fc, 4096
- fc, 4096
- fc, 1000
2012

VGG, 19 layers
- 3x3 conv, 64
- 3x3 conv, 54, pool/2
- 3x3 conv, 128
- 3x3 conv, 128, pool/2
- 3x3 conv, 256
- 3x3 conv, 256
- 3x3 conv, 256
- 3x3 conv, 256, pool/2
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512, pool/2
- fc, 4096
- fc, 4096
- fc, 1000
2015

GoogleNet, 22 layers
- 3x3 conv, 64
- 3x3 conv, 54, pool/2
- 3x3 conv, 128
- 3x3 conv, 128, pool/2
- 3x3 conv, 256
- 3x3 conv, 256
- 3x3 conv, 256
- 3x3 conv, 256, pool/2
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512, pool/2
- fc, 4096
- fc, 4096
- fc, 1000
2014

ResNet, 152 layers
- 3x3 conv, 64
- 3x3 conv, 54, pool/2
- 3x3 conv, 128
- 3x3 conv, 128, pool/2
- 3x3 conv, 256
- 3x3 conv, 256
- 3x3 conv, 256
- 3x3 conv, 256, pool/2
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512, pool/2
- fc, 4096
- fc, 4096
- fc, 1000
2015

Figure courtesy: Kaiming He
Outline

● Convolutional Networks (ConvNets)

● Recurrent Networks (RNNs)
  ○ Long-range dependency, vanishing
  ○ LSTM
  ○ RNNs in different forms

● Attention Mechanisms
  ○ (Query, Key, Value)
  ○ Attention on Text and Images

● Transformers: Multi-head Attention
  ○ Transformer
  ○ BERT
ConvNets vs. Recurrent Networks (RNNs)

- Spatial Modeling vs. Sequential Modeling
- Fixed vs. variable number of computation steps.

The output depends ONLY on the current input

The hidden layers and the output additionally depend on previous states of the hidden layers
RNNs in Various Forms

One to One

Image classification

One to Many

Image captioning

Many to One

Sentence sentiment analysis / Video recognition

Many to Many

Machine Translation

(Sequence-to-sequence)

Named Entity Recognition

(Sequence tagging)
Vanishing / Exploding Gradients in RNNs

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

Bengio et al., 1994 “Learning long-term dependencies with gradient descent is difficult”
Pascanu et al., 2013 “On the difficulty of training recurrent neural networks”
Vanishing / Exploding Gradients in RNNs

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

Bengio et al., 1994 “Learning long-term dependencies with gradient descent is difficult”
Pascanu et al., 2013 “On the difficulty of training recurrent neural networks”
Vanishing / Exploding Gradients in RNNs

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

Bengio et al., 1994 “Learning long-term dependencies with gradient descent is difficult”
Pascanu et al., 2013 “On the difficulty of training recurrent neural networks”
Long-term Dependency Problem

I live in **France** and I know __________

Example courtesy: Manik Soni
I live in France and I know __French__
I live in France and I know ___French___

I live in France, a beautiful country, and I know ___French___

Example courtesy: Manik Soni
LSTMs are designed to explicitly alleviate the long-term dependency problem [Hochreiter & Schmidhuber (1997)]
Long Short Term Memory (LSTM)

- Gate functions make decisions of reading, writing, and resetting information

- Forget gate: whether to erase cell (reset)
- Input gate: whether to write to cell (write)
- Output gate: how much to reveal cell (read)
Long Short Term Memory (LSTM)

- **Forget gate**: decides what must be removed from $h_{t-1}$

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

- **Forget gate:** decides what must be removed from $h_{t-1}$

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

- **Input gate:** decides what new information to store in the cell

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \text{tanh}(W_c \cdot [h_{t-1}, x_t] + b_c)$$
Long Short Term Memory (LSTM)

- Update cell state:

\[ C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \]

forgetting unneeded things

scaling the new candidate values by how much we decided to update each state value.
Long Short Term Memory (LSTM)

- **Update cell state:**

  \[ C_t = f_t \ast C_{t-1} + i_t \ast \tilde{C}_t \]

  - **forgetting unneeded things:**
  - scaling the new candidate values by how much we decided to update each state value.

- **Output gate:** decides what to output from our cell state

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

  \[ h_t = o_t \ast \tanh(C_t) \]

  - sigmoid decides what parts of the cell state we’re going to output
Backpropagation in LSTM

- No multiplication with matrix $W$ during backprop
- Multiplied by different values of forget gate $\rightarrow$ less prone to vanishing/exploding gradient
RNNs in Various Forms

One to One

Image classification

One to Many

Image captioning

Many to One

Sentence sentiment analysis / Video recognition

Many to Many

Machine Translation (Sequence-to-sequence)

Many to Many

Named Entity Recognition (Sequence tagging)
**RNNs in Various Forms**

- **Bi-directional RNN**
  - Hidden state is the concatenation of both forward and backward hidden states.
  - Allows the hidden state to capture both past and future information.

[Speech Recognition with Deep Recurrent Neural Networks, Alex Graves]
**RNNs in Various Forms**

- **Bi-directional RNN**
  - Hidden state is the concatenation of both forward and backward hidden states.
  - Allows the hidden state to capture both past and future information.

- **Tree-structured RNN**
  - Hidden states condition on both an input vector and the hidden states of arbitrarily many child units.
  - Standard LSTM = a special case of tree-LSTM where each internal node has exactly one child.

---

[Speech Recognition with Deep Recurrent Neural Networks, Alex Graves]
RNNs in Various Forms

- RNN for 2-D sequences

[Pixel Recurrent Neural Networks, van den Oord. et al. 2016]
RNNs in Various Forms

- RNN for Graph Structures
  - Used in, e.g., image segmentation

[Semantic Object Parsing with Graph LSTM. Liang et al. 2016]
Outline

- Convolutional Networks (ConvNets)
- Recurrent Networks (RNNs)
  - Long-range dependency, vanishing
  - LSTM
  - RNNs in different forms
- Attention Mechanisms
  - (Query, Key, Value)
  - Attention on Text and Images
- Transformers: Multi-head Attention
  - Transformer
  - BERT
Attention: Examples

- Chooses which features to pay attention to

A woman is throwing a frisbee in a park.  
A dog is standing on a hardwood floor.  
A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.  
A group of people sitting on a boat in the water.  
A giraffe standing in a forest with trees in the background.

Image captioning [Show, attend and tell. Xu et al. 15]
Attention: Examples

- Chooses which features to pay attention to

Machine Translation

Figure courtesy: Olah & Carter, 2016
Why Attention?
Why Attention?

- Long-range dependencies
  - Dealing with gradient vanishing problem
Why Attention?

- Long-range dependencies
  - Dealing with gradient vanishing problem
- Fine-grained representation instead of a single global representation
  - Attending to smaller parts of data: patches in images, words in sentences

Figure courtesy: Lilian Weng
Why Attention?

- Long-range dependencies
  - Dealing with gradient vanishing problem
- Fine-grained representation instead of a single global representation
  - Attending to smaller parts of data: patches in images, words in sentences
- Improved Interpretability

Figure courtesy: Olah & Carter, 2016
Attention Computation

- Encode each token in the input sentence into vectors
- When decoding, perform a linear combination of these vectors, weighted by “attention weights”
  - \( a = \text{softmax}(\text{alignment_scores}) \)

Figure courtesy: MARTA R. COSTA-JUSSÀ
Attention Computation (cont’d)

- Combine together value by taking the weighted sum
Attention Computation (cont’d)

- Combine together value by taking the weighted sum

- **Query**: decoder state
- **Key**: all encoder states
- **Value**: all encoder states
Attention Variants

- Popular attention mechanisms with different alignment score functions

Alignment score = f(Query, Keys)

- **Query**: decoder state $s_t$
- **Key**: all encoder states $h_i$
- **Value**: all encoder states $h_i$

<table>
<thead>
<tr>
<th>Name</th>
<th>Alignment score function</th>
<th>Citation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content-base</td>
<td>$\text{score}(s_t, h_i) = \text{cosine}[s_t, h_i]$</td>
<td>Graves2014</td>
</tr>
<tr>
<td>attention</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Additive(*)</td>
<td>$\text{score}(s_t, h_i) = v_a^T \tanh(W_a[s_t; h_i])$</td>
<td>Bahdanau2015</td>
</tr>
<tr>
<td>Location-Base</td>
<td>$\alpha_{t,i} = \text{softmax}(W_a s_t)$</td>
<td>Luong2015</td>
</tr>
<tr>
<td></td>
<td>Note: This simplifies the softmax alignment to only depend on the target position.</td>
<td></td>
</tr>
<tr>
<td>General</td>
<td>$\text{score}(s_t, h_i) = s_t^T W_a h_i$</td>
<td>Luong2015</td>
</tr>
<tr>
<td></td>
<td>where $W_a$ is a trainable weight matrix in the attention layer.</td>
<td></td>
</tr>
<tr>
<td>Dot-Product</td>
<td>$\text{score}(s_t, h_i) = s_t^T h_i$</td>
<td>Luong2015</td>
</tr>
<tr>
<td>Scaled Dot-Product</td>
<td>$\text{score}(s_t, h_i) = \frac{s_t^T h_i}{\sqrt{n}}$</td>
<td>Vaswani2017</td>
</tr>
</tbody>
</table>

Note: very similar to the dot-product attention except for a scaling factor; where $n$ is the dimension of the source hidden state.

 Courtesy: Lilian Weng
Attention on Images – Image Captioning

1. Input Image
2. Convolutional Feature Extraction
3. RNN with attention over the image
4. Word by word generation

- **Query**: decoder state
- **Key**: visual feature maps
- **Value**: visual feature maps

[Show, attend and tell. Xu et al. 15]
Attention on Images – Image Captioning

Hard attention vs Soft attention

A bird flying over a body of water.

Hard attention:

$\hat{z}_t = \begin{cases} \bullet, \bullet, \bullet, \bullet \\ \end{cases}$

$L_z = \sum_{z \in \{\bullet, \bullet, \bullet, \bullet\}} \log p(y | z)$

Soft attention:

$\hat{z}_t = \phi((a_t, \{a_i\}))$

$L_s = \sum_{s} p(s | a) \log p(y | s, a)$

A variational lower bound of maximum likelihood

$\hat{z}_t = \langle p_1, p_2, p_3, p_4, p_5, p_6 \rangle$

Computes the expected attention

14x14x512 = 196 x 512 (L x D) annotations
Attention on Images – Image Captioning

Hard attention vs Soft attention

A bird flying over a body of water
Attention on Images – Image Paragraph Generation

- Generate a long paragraph to describe an image
  - Long-term visual and language reasoning
  - Contentful descriptions -- ground sentences on visual features

This picture is taken for three baseball players on a field. The man on the left is wearing a blue baseball cap. The man has a red shirt and white pants. The man in the middle is in a wheelchair and holding a baseball bat. Two men are bending down behind a fence. There are words band on the fence.

A tennis player is attempting to hit the tennis ball with his left foot hand. He is holding a tennis racket. He is wearing a white shirt and white shorts. He has his right arm extended up. There is a crowd of people watching the game. A man is sitting on the chair.

A couple of zebra are standing next to each other on dirt ground near rocks. There are trees behind the zebras. There is a large log on the ground in front of the zebra. There is a large rock formation to the left of the zebra. There is a small hill near a small pond and a wooden log. There are green leaves on the tree.
Attention on Images – Image Paragraph Generation

Attention on Images – Image Paragraph Generation

Semantic region detection & captioning

Local Phrases

- people playing baseball
- a man wearing white shirt and pants
- man holding a baseball bat
- person wearing a helmet in the field
- a man bending over
Attention on Images – Image Paragraph Generation

![Diagram showing the process of image paragraph generation.

Semantic Regions
Attentive Reasoning
Generator
Sentence
Sentence
Sentence
Sentence Discriminator
Topic-Transition Discriminator
Paragraph description Corpus

Semantic region detection & captioning
Attention on both visual regions and text phrases

The diagram illustrates the process of generating a paragraph from an image. Semantic regions are detected and captioned, then the generator produces sentences that are assessed by both a sentence discriminator and a topic-transition discriminator. The adversarial training framework is used to optimize the model.

The objective of the adversarial framework is written as:

\[ L = \sum_{t,i} \left( -\log D_t(s_{t,i}) \right) + \lambda \left( -\log G_t(s_{t,i}) \right) \]

where

- \( \lambda \) is the balancing parameter fixed to 0.1
- \( G_t(s_{t,i}) \) is the reconstruction loss for generator optimization, which is defined as:

\[ L_s = \sum_{t} \sum_{i} \left( -\log D_t(s_{t,i}) \right) \]

and

\[ L_c = \sum_{t} \sum_{i} \left( -\log G_t(s_{t,i}) \right) \]

The discrete nature of text samples hinders gradient back-propagation from the discriminators to the generator, which is resolved by the implementation of SeqGAN.

The scheme of the paragraph generator and the multi-level discriminators attempts to confuse the discriminators by generating a critic between real and generated samples, while the generator recurrently produces each sentence by reasoning about local semantic regions and preceding paragraph state.

Figure 2. Our RTT-GAN alternatively optimizes a structured paragraph generator and two discriminators following an adversarial training scheme.
Attention on Images – Image Paragraph Generation

Semantic Regions

Semantic region detection & captioning

Attentive Reasoning

Attentive on both visual regions and text phrases

Hierarchical text generation

Generator

Sentence

Sentence

Sentence

Sentence

Sentence Discriminator

Topic-Transition Discriminator

Paragraph description Corpus

Figure 2. Our RTT-GAN alternatively optimizes a structured paragraph generator and two discriminators following an adversarial training.
Attention on Images – Image Paragraph Generation

Attention on both visual regions and text phrases

Semantic region detection & captioning

Semantic Regions

Attentive Reasoning

Generator

Sentence

Sentence

Sentence

Hierarchical text generation

Sentence Discriminator

Topic-Transition Discriminator

Paragraph description Corpus

Multi-level adversarial learning

Attention on Images – Image Paragraph Generation

Semantic Regions

Attentive Reasoning

Generator

Sentence

Sentence

Sentence

Hierarchical text generation

Sentence Discriminator

Topic-Transition Discriminator

Paragraph description Corpus

Multi-level adversarial learning

Attention on both visual regions and text phrases

Semantic region detection & captioning
Paragraph: A group of people are riding bikes. There are two people riding bikes parked on the sidewalk. He is wearing a black shirt and jeans. A woman is wearing a short sleeve yellow shirt and shorts. There are many other people on the red and black bikes. A woman wearing a shirt is riding a bicycle.
Outline

- Convolutional Networks (ConvNets)
- Recurrent Networks (RNNs)
  - Long-range dependency, vanishing
  - LSTM
  - RNNs in different forms
- Attention Mechanisms
  - (Query, Key, Value)
  - Attention on Text and Images
- Transformers: Multi-head Attention
  - Transformer
  - BERT
Transformers – Multi-head (Self-)Attention

- State-of-the-art Results by Transformers
  - [Vaswani et al., 2017] Attention Is All You Need
    - Machine Translation
  - [Devlin et al., 2018] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
    - Pre-trained Text Representation
  - [Radford et al., 2019] Language Models are Unsupervised Multitask Learners
    - Language Models
3.2.1 Scaled Dot-Product Attention

We call our particular attention “Scaled Dot-Product Attention” (Figure 2). The input consists of queries and keys of dimension $d_k$, and values of dimension $d_v$. We compute the dot products of the query with all keys, divide each by $p_{d_k}$, and apply a softmax function to obtain the weights on the values.

In practice, we compute the attention function on a set of queries simultaneously, packed together into a matrix $Q$. The keys and values are also packed together into matrices $K$ and $V$. We compute the matrix of outputs as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{p_{d_k}}\right)V$$

The two most commonly used attention functions are additive attention $^2$, and dot-product (multiplicative) attention. Dot-product attention is identical to our algorithm, except for the scaling factor of $\frac{1}{p_{d_k}}$. Additive attention computes the compatibility function using a feed-forward network with a single hidden layer. While the two are similar in theoretical complexity, dot-product attention is much faster and more space-efficient in practice, since it can be implemented using highly optimized matrix multiplication code.

While for small values of $d_k$ the two mechanisms perform similarly, additive attention outperforms dot product attention without scaling for larger values of $d_k$. We suspect that for large values of $d_k$, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients $^4$. To counteract this effect, we scale the dot products by $\frac{1}{p_{d_k}}$.

3.2.2 Multi-Head Attention

Instead of performing a single attention function with $d_{model}$-dimensional keys, values and queries, we found it beneficial to linearly project the queries, keys and values $h$ times with different, learned linear projections to $d_k$, $d_k$ and $d_v$ dimensions, respectively. On each of these projected versions of queries, keys and values we then perform the attention function in parallel, yielding $d_v$-dimensional output values. These are concatenated and once again projected, resulting in the final values, as depicted in Figure 2.

4 To illustrate why the dot products get large, assume that the components of $q$ and $k$ are independent random variables with mean 0 and variance 1. Then their dot product, $q \cdot k = \sum_{i=1}^{d_k} q_i k_i$, has mean 0 and variance $d_k$. 
We call our particular attention "Scaled Dot-Product Attention" (Figure 2). The input consists of variables with mean $\mu_0$. Then their dot product, $Q \cdot K$, has mean $\mu_1$. The two mechanisms perform similarly, additive attention outperforms multiplicative attention. Dot-product attention is identical to our algorithm, except for the scaling factor.

To illustrate why the dot products get large, assume that the components of $k$ are extremely small. The dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients.

Instead of performing a single attention function with dot product attention without scaling for larger values of $p$, we found it beneficial to linearly project the queries, keys and values. The two most commonly used attention functions are additive attention and dot-product (multiplicative) attention. Dot-product attention is identical to our algorithm, except for the scaling factor.
We call our particular attention "Scaled Dot-Product Attention" (Figure 2). The input consists of values. Variables with mean output values. These are concatenated and once again projected, resulting in the final values, as queries, keys and values we then perform the attention function in parallel, yielding extremely small gradients.

The dot product attention without scaling for larger values of matrix multiplication code.

A single hidden layer. While the two are similar in theoretical complexity, dot-product attention is multiplicative attention. Dot-product attention is identical to our algorithm, except for the scaling factor.

In practice, we compute the attention function on a set of queries simultaneously, packed together query with all keys, divide each by queries and keys of dimension $d$. To counteract this effect, we scale the dot products by the matrix of outputs as:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{Q K}{\sqrt{d}}) V$$

We suspect that for large values of $d$, the dot products grow large in magnitude, pushing the softmax function into regions where it has predictions for position $i$, the dot products grow large in magnitude, pushing the softmax function into regions where it has predictions for position $i$. To illustrate why the dot products get large, assume that the components of $Q$, $K$, and $V$ are independent random vectors with mean 0 and variance 1. Additive attention computes the compatibility function using a feed-forward network with a single hidden layer. While the two are similar in theoretical complexity, additive attention is multiplicative attention.

Multi-Head Attention consists of several attention layers running in parallel.
Multi-head Attention in Encoders and Decoders

Encoder:
The encoder is composed of a stack of $N = 6$ identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. We employ a residual connection around each of the two sub-layers, followed by layer normalization. That is, the output of each sub-layer is $\text{LayerNorm}(x + \text{Sublayer}(x))$, where $\text{Sublayer}(x)$ is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension $d_{\text{model}} = 512$.

Decoder:
The decoder is also composed of a stack of $N = 6$ identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, we employ residual connections around each of the sub-layers, followed by layer normalization. We also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This masking, combined with the fact that the output embeddings are offset by one position, ensures that the predictions for position $i$ can depend only on the known outputs at positions less than $i$.

3.2 Attention
An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.
Multi-head Attention in Encoders and Decoders

Transformer

encoder self attention
1. Multi-head Attention
2. Query=Key=Value

decoder self attention
1. Masked Multi-head Attention
2. Query=Key=Value

encoder-decoder attention
1. Multi-head Attention
2. Encoder Self attention=Key=Value
3. Decoder Self attention=Query

Figure 1: The Transformer - model architecture.
BERT: Pre-trained Text Representation Model

Image source: Vaswani, et al., 2017
**BERT: Pre-trained Text Representation Model**

- Conventional word embedding:
  - Word2vec, Glove
  - A pre-trained **matrix**, each row is an embedding vector of a word

<table>
<thead>
<tr>
<th>Word</th>
<th>Embedding Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>fox</td>
<td>-0.348680 -0.077720 0.177750 -0.094953 -0.452890 0.237790 0.209440 0.037886 0.035064 0.899010</td>
</tr>
<tr>
<td>ham</td>
<td>-0.773320 -0.282540 0.580760 0.841480 0.258540 0.585210 -0.021890 -0.463680 0.139070 0.658720</td>
</tr>
<tr>
<td>brown</td>
<td>-0.374120 -0.076264 0.109260 0.186620 0.029943 0.182700 -0.631980 0.133060 -0.128980 0.603430</td>
</tr>
<tr>
<td>beautiful</td>
<td>0.171200 0.534390 -0.348540 -0.097234 0.101800 -0.170860 0.296560 -0.041816 -0.516550 2.117200</td>
</tr>
<tr>
<td>jumps</td>
<td>-0.334850 0.215990 -0.350440 -0.260020 0.411070 0.154010 -0.386110 0.206380 0.386700 1.460500</td>
</tr>
<tr>
<td>eggs</td>
<td>-0.417810 -0.035192 -0.126150 -0.215930 -0.669740 0.513250 -0.797090 -0.068611 0.634660 1.256300</td>
</tr>
<tr>
<td>beans</td>
<td>-0.423290 -0.264500 0.200870 0.082187 0.066944 1.027600 -0.989140 -0.259950 0.145960 0.766450</td>
</tr>
<tr>
<td>sky</td>
<td>0.312550 -0.303080 0.019587 -0.354940 0.100180 -0.141530 -0.514270 0.886110 -0.530540 1.556600</td>
</tr>
<tr>
<td>bacon</td>
<td>-0.430730 -0.016025 0.484620 0.101390 -0.299200 0.761820 -0.353130 -0.325290 0.156730 0.873210</td>
</tr>
<tr>
<td>breakfast</td>
<td>0.073378 0.227670 0.208420 -0.456790 -0.078219 0.601960 -0.024494 -0.467980 0.054627 2.283700</td>
</tr>
<tr>
<td>toast</td>
<td>0.130740 -0.193730 0.253270 0.090102 -0.272580 -0.030571 0.096945 -0.115060 0.484000 0.848300</td>
</tr>
<tr>
<td>today</td>
<td>-0.156570 0.594890 -0.031445 -0.077586 0.278630 -0.502910 -0.066350 -0.081890 -0.047986 2.803600</td>
</tr>
<tr>
<td>blue</td>
<td>0.129450 0.036518 0.032298 -0.060034 0.399840 -0.103020 -0.507880 0.076630 -0.422920 0.815730</td>
</tr>
<tr>
<td>green</td>
<td>-0.072368 0.233200 0.137260 -0.156630 0.248440 0.349870 -0.241700 -0.091426 -0.530150 1.341300</td>
</tr>
<tr>
<td>kings</td>
<td>0.259230 -0.854690 0.360010 -0.642000 0.568530 -0.321420 0.173250 0.133030 -0.089720 1.526800</td>
</tr>
<tr>
<td>dog</td>
<td>-0.057120 0.052685 0.003026 -0.048517 0.007043 0.041856 -0.024704 -0.039783 0.000614 0.380416</td>
</tr>
<tr>
<td>sausages</td>
<td>-0.174290 -0.064890 -0.046976 0.287420 -0.128150 0.647630 0.056315 -0.240440 -0.259094 0.502220</td>
</tr>
<tr>
<td>lazy</td>
<td>-0.353320 -0.299710 -0.176230 -0.321940 -0.385640 0.586110 0.411160 -0.416080 0.073093 1.486500</td>
</tr>
<tr>
<td>love</td>
<td>0.139490 0.534530 -0.252470 -0.125650 0.048748 0.152440 0.199060 -0.065970 0.128830 2.059900</td>
</tr>
<tr>
<td>quick</td>
<td>-0.445630 0.191510 -0.249210 0.465900 0.161950 0.212780 -0.046480 0.021170 0.417660 1.686900</td>
</tr>
</tbody>
</table>

Image source: Vaswani, et al., 2017
**BERT: Pre-trained Text Representation Model**

- Conventional word embedding:
  - Word2vec, Glove
  - A pre-trained matrix, each row is an embedding vector of a word

**English Wikipedia Corpus**

The Amtrak Commuter continued through July 4, 2003. This final annual commuter took place less than a week before the June 26 Stonewall riots, in which the patrons of the Stonewall Inn, a gay bar in Greenwich Village, fought against police who raided the bar. Rockwell received several telephone calls threatening him and the other New York participants, but he was able to arrange for police protection for the chemists as all the way to Philadelphia. About 40 people participated, including the deputy mayor of Philadelphia and his wife. The case came to an end when on the train, but two remainders from the New York contingent broke from the single life group line and held hands. What remains is to be seen open. Rockwell vigorously defended him to unloading members of the press.

Following the 2008 Presidential referendum, there was a sense, particularly among the younger and more radical participants, that the time for silent polishing had passed. Dissent and disobedience had begun to take new and more emphatic forms (in society). The conference passed a resolution drafted by itself, its partner Fred Sargent, Ehren and Linn. Those to move the demonstration from July 4 in Philadelphia to the last weekend in June in New York City, as well as proposing to “other organizations throughout the country... suggesting that they hold passive demonstrations on that day” to commemorate the Stonewall riot.
BERT: Pre-trained Text Representation Model

- BERT: A model to extract *contextualized* word embedding
BERT: Pre-trained Text Representation Model

- BERT: A model to extract *contextualized* word embedding
BERT: Pre-trained Text Representation Model

- BERT: A model to extract *contextualized* word embedding
BERT: Pre-trained Text Representation Model

- Use BERT for sentence classification

Classifier
(Feed-forward neural network + softmax)
BERT Results

• Huge improvements over SOTA on 12 NLP task

<table>
<thead>
<tr>
<th>System</th>
<th>MNLI-(m/mm)</th>
<th>QQP</th>
<th>QNLI</th>
<th>SST-2</th>
<th>CoLA</th>
<th>STS-B</th>
<th>MRPC</th>
<th>RTE</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>392k</td>
<td>363k</td>
<td>108k</td>
<td>67k</td>
<td>8.5k</td>
<td>5.7k</td>
<td>3.5k</td>
<td>2.5k</td>
<td></td>
</tr>
<tr>
<td>Pre-OpenAI SOTA</td>
<td>80.6/80.1</td>
<td>66.1</td>
<td>82.3</td>
<td>93.2</td>
<td>35.0</td>
<td>81.0</td>
<td>86.0</td>
<td>61.7</td>
<td>74.0</td>
</tr>
<tr>
<td>BiLSTM+ELMo+Attn</td>
<td>76.4/76.1</td>
<td>64.8</td>
<td>79.9</td>
<td>90.4</td>
<td>36.0</td>
<td>73.3</td>
<td>84.9</td>
<td>56.8</td>
<td>71.0</td>
</tr>
<tr>
<td>OpenAI GPT</td>
<td>82.1/81.4</td>
<td>70.3</td>
<td>88.1</td>
<td>91.3</td>
<td>45.4</td>
<td>80.0</td>
<td>82.3</td>
<td>56.0</td>
<td>75.2</td>
</tr>
<tr>
<td>BERT_{BASE}</td>
<td>84.6/83.4</td>
<td>71.2</td>
<td>90.1</td>
<td>93.5</td>
<td>52.1</td>
<td>85.8</td>
<td>88.9</td>
<td>66.4</td>
<td>79.6</td>
</tr>
<tr>
<td>BERT_{LARGE}</td>
<td><strong>86.7/85.9</strong></td>
<td><strong>72.1</strong></td>
<td><strong>91.1</strong></td>
<td><strong>94.9</strong></td>
<td><strong>60.5</strong></td>
<td><strong>86.5</strong></td>
<td><strong>89.3</strong></td>
<td><strong>70.1</strong></td>
<td><strong>81.9</strong></td>
</tr>
</tbody>
</table>

Table 1: GLUE Test results, scored by the GLUE evaluation server. The number below each task denotes the number of training examples. The “Average” column is slightly different than the official GLUE score, since we exclude the problematic WNLI set. OpenAI GPT = (L=12, H=768, A=12); BERT_{BASE} = (L=12, H=768, A=12); BERT_{LARGE} = (L=24, H=1024, A=16). BERT and OpenAI GPT are single-model, single task. All results obtained from https://gluebenchmark.com/leaderboard and https://blog.openai.com/language-unsupervised/.
BERT: Pre-training Procedure

- **Model architecture:**
  - A big Transformer Encoder (240M free parameters)

- **Dataset:**
  - Wikipedia (2.5B words) + a collection of free ebooks (800M words)
BERT: Pre-training Procedure

- Model architecture:
  - A big Transformer Encoder (240M free parameters)

- Dataset:
  - Wikipedia (2.5B words) + a collection of free ebooks (800M words)

- Training procedure
  - masked language model (masked LM)
    - Masks some percent of words from the input and has to reconstruct those words from context
**BERT: Pre-training Procedure**

- **Masked LM**

  Use the output of the masked word's position to predict the masked word.

  Possible classes:
  - All English words
  - Improvisation
  - Zyzzyva

  FFNN + Softmax

  **Input**

  Randomly mask 15% of tokens.
BERT: Pre-training Procedure

- Model architecture:
  - A big Transformer Encoder (240M free parameters)

- Dataset:
  - Wikipedia (2.5B words) + a collection of free ebooks (800M words)

- Training procedure
  - masked language model (masked LM)
    - Masks some percent of words from the input and has to reconstruct those words from context
  - Two-sentence task
    - To understand relationships between sentences
    - Concatenate two sentences A and B and predict whether B actually comes after A in the original text
BERT: Pre-training Procedure

- Two sentence task

Predict likelihood that sentence B belongs after sentence A

Input:

[CLS] the man [MASK] to the store [SEP] penguin [MASK] are flightless birds [SEP]

Sentence A

Sentence B

Tokenized Input

[CLS] the man [MASK] to the store [SEP] penguin [MASK] are flightless birds [SEP]

Input

1 2 3 4 5 6 7 8 ... 512

• FFNN + Softmax

1% isNext

99% NotNext
BERT: Pre-training Procedure

- BERT is trained on 4 TPU pods (=256 TPU chips) in 4 days
  - TPU: a matrix multiplication engine

- = 64 V100 GPUs, Infiniband network, 5.3 days

- = a standard 4 GPU desktop with RTX 2080Ti, 99 days

source: Tim Dettmers
Word Embedding on Texar

Texar

- A general-purpose text generation toolkit on TensorFlow

Texar stack

<table>
<thead>
<tr>
<th>Applications</th>
<th>Library APIs</th>
<th>Model templates + Config files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td>Evaluation</td>
<td>Prediction</td>
</tr>
</tbody>
</table>

Texar stack

<table>
<thead>
<tr>
<th>Models</th>
<th>Data</th>
<th>Trainer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architectures</td>
<td>Losses</td>
<td>Executive</td>
</tr>
<tr>
<td>Encoder</td>
<td>(Seq) MaxLikelihood</td>
<td>MonoText</td>
</tr>
<tr>
<td>Decoder</td>
<td>Adversarial</td>
<td>Dialog</td>
</tr>
<tr>
<td>Embedder</td>
<td></td>
<td>Seq/Episodic RL Agent</td>
</tr>
<tr>
<td>Classifier</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QNet</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connector</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Policy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rewards</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RL-related</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Regularize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multi-field/type Parallel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ir decay / grad clip / ...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Word Embedding on Texar

- Word2vec, Glove

```python
import texar as tx

# Load data and pre-trained word embedding matrix
data = tx.data.MonoTextData(hparams=config.data)
iterator = tx.data.DataIterator(data)
data_batch = iterator.get_next()

# Create and initialize word embedder
embedder = tx.modules.WordEmbedder(
    init_value=data.embedding_init_value, hparams=config.emb)

# Embed text into vectors
data_embed = embedder(data_batch)

# Downstream tasks
classifier = tx.modules.Conv1DClassifier(hparams=config.clas)
logits, pred = classifier(input=data_embed)
```
Word Embedding on Texar

- Word2vec, Glove
- BERT

```python
import texar as tx

data = tx.data.MonotextData(hparams=config.data)
iterator = tx.data.DataIterator(data)
data_batch = iterator.get_next()

# Create and initialize word embedder
embedder = tx.modules.WordEmbedder(
    init_value=data.embedding_init_value, hparams=config)

# Create BERT embedder
embedder = tx.modules.TransformerEncoder(hparams=bert_config)

# Initialize BERT embedder
text.init_bert_checkpoint("./bert.ckpt")

# Embed text into vectors
data_embed = embedder(data_batch)

# Downstream tasks
classifier = tx.modules.Conv1DClassifier(hparams=config.classifier)
logits, pred = classifier(input=data_embed)
```
# Read data
```python
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()
```

# Encode
```python
embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
encoder = TransformerEncoder(hparams=encoder_hparams)
enc_outputs = encoder(embedder(batch['source_text_ids']),
                     batch['source_length'])
```

# Decode
```python
decoder = AttentionRNNDecoder(memory=enc_outputs,
                              hparams=decoder_hparams)
outputs, length, _ = decoder(inputs=embedder(batch['target_text_ids']),
                            seq_length=batch['target_length']-1)
```

# Loss
```python
loss = sequence_sparse_softmax_cross_entropy(
      labels=batch['target_text_ids'][1:],
      logits=outputs.logits,
      seq_length=length)
```
Seq2seq Attention on Texar

```python
# Read data
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()

# Encode
embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
encoder = TransformerEncoder(hparams=encoder_hparams)
enc_outputs = encoder(embedder(batch['source_text_ids']),
batch['source_length'])

# Decode
decoder = AttentionRNNDecoder(memory=enc_outputs,
hparams=decoder_hparams)
outputs, length, _ = decoder(inputs=embedder(batch['target_text_ids']),
seq_length=batch['target_length']-1)

# Loss
loss = sequence_sparse_softmax_cross_entropy(
  labels=batch['target_text_ids'][:,1:],
  logits=outputs.logits, seq_length=length)
```

Decoder hyperparameters:

```
  decoder_hparams = {
    'rnn_cell': {
      'type': 'LSTMCell'
    },
    'num_layers': 2,
    'attention': {
      'type': 'LuongAttention',
      'kwargs': {
        'num_units': 256,
      }
    }
  }
```
Takeaways

- Convolutional Networks (ConvNets)
- Recurrent Networks (RNNs)
  - LSTM designed for long-range dependency, vanishing gradients
  - RNNs not only for sequence data, but also 2D sequences, Trees, graphs
- Attention Mechanisms
  - Three core elements: (Query, Key, Value)
  - Many variants based on alignment score functions
  - Attention on Text and Images
- Transformers: Multi-head Attention
  - Transformer: encoder-decoder
  - BERT: pre-trained text representation
  - GPT-2: pre-trained language model