The Big Picture

- Natural language (& speech) are central to human intelligence
- **Natural Language Processing (NLP)** attempts to capture some of this intelligence algorithmically and is of huge practical importance
  - machine translation, chatbots, automatic fact checking, ...
- NLP has seen several transformative shifts in the last few years
Goals of this Tutorial

- Build basic NLP literacy by looking at language models
- Get up to date with recent developments
  - BERT, GPT, Self-Supervised Learning (SSL)
- Know where to look if you’re starting an NLP project
- Will focus more on building intuition than math
Language Models
Language Models

Language models (LMs) assign **probabilities** to sequences of words

- **Speech recognition:** $P(\text{I will be back soonish}) > P(\text{I will be bassoon dish})$

- **Spell checkers:** $P(\text{There are two midterms}) > P(\text{Their are two midterms})$

- **Machine translation:**

  他 向 记者 介绍了 主要 内容

  $P(\text{He to reporters introduced main content}) <$

  $P(\text{He briefed reporters on the main contents of the statement}) <$
Anatomy of a Language Model

The probability of a sequence of \(n\) words is the product of the conditional probability of each word and its history (chain rule):

\[
P(w_1:n) = \prod_{k=1}^{n} P(w_k | w < k)
\]

How do we compute the conditional probability? In practice, we take the log sum:
Language Models

- We can use **recurrent** neural networks (RNNs)! E.g. LSTMs, GRUs
- Work well for variable length inputs, like sentences
Language Models

Recurrent neural networks have some shortcomings:

- Not parallelizable within training examples
- Difficult to optimize due to vanishing gradients
- Difficulty modelling long range dependencies
Language Models

We’d like an architectural primitive that:

● Is parallelizable within training examples
  ○ Take advantage of accelerators like GPUs/TPUs
● Directly facilities interactions between tokens
  ○ To better model long range dependencies
● Attention to the rescue?
Attention
Many flavours of attention have been proposed.

We will focus on the most common, (scaled) **dot-product** attention.

Scaled dot-product attention is the backbone of **transformers**.

Like any attention mechanism, we need to make two decisions:

- How to compute similarity? → **dot-product**
- How do we normalize the similarity score? → **softmax**
Scaled dot-product attention takes three matrices as input:

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

A softmax normalizes similarities → [0, 1]

Similarity is simply the dot product between Q and K

The output is simply a scaling of V

Attention maps a query, Q and a set of key-value (K, V) pairs to an output. 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.

Queries, Keys and What?

These will change depending on how the attention mechanism is used

- In self-attention, $Q == K == V$
- Updating the representation of each token based on the other tokens in the sequence

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QQ^T}{\sqrt{d_k}} \right) V$$
Queries, Keys and What?

These will change depending on how the attention mechanism is used

- In **cross**-attention, \( K = V \) and come from the encoder. \( Q \) comes from the decoder.
- The decoder “focuses” on certain tokens in the encoders output

\[
\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right)V
\]
Residuals: Projections

- Q, K & V are *projections* of embedded tokens
- If this is a multi-layered network (e.g. a transformer), they are outputs of the previous layer

https://jalammar.github.io/illustrated-transformer/
Residuals: The Beast with Many Heads

Usually, we use **multi-head** scaled dot-product attention

https://jalammar.github.io/illustrated-transformer/
Transformers (covered in lecture)

https://jalammar.github.io/illustrated-transformer/
The Payoff

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. $n$ is the sequence length, $d$ is the representation dimension, $k$ is the kernel size of convolutions and $r$ the size of the neighborhood in restricted self-attention.

<table>
<thead>
<tr>
<th>Layer Type</th>
<th>Complexity per Layer</th>
<th>Sequential Operations</th>
<th>Maximum Path Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-Attention</td>
<td>$O(n^2 \cdot d)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Recurrent</td>
<td>$O(n \cdot d^2)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Convolutional</td>
<td>$O(k \cdot n \cdot d^2)$</td>
<td>$O(1)$</td>
<td>$O(\log_k(n))$</td>
</tr>
<tr>
<td>Self-Attention (restricted)</td>
<td>$O(r \cdot n \cdot d)$</td>
<td>$O(1)$</td>
<td>$O(n/r)$</td>
</tr>
</tbody>
</table>

Before we move on...

- LMs assign **probabilities to sequences** and are the “workhorse” of NLP
- Typically implemented with RNNs; being replaced with **Transformers**
- **Multi**-head **scaled** dot-product attention the backbone of Transformers
  - Allows us to learn long range dependencies and parallelize computation within training examples
- **How do we train Transformers as language models?**
Pretrained Language Models & Self-Supervised Learning
Wishlist

- We want to train transformers as LMs
  - Learn general properties of language that can be transferred to downstream tasks
- Ideally, we could train LMs using unlabeled text
  - Leverage unsupervised or Self-Supervised Learning (SSL)
- (At least) two paradigms have emerged
  - Generative Pretrained Transformer (GPT)
    - Next-token prediction, decoder only transformer
  - Bidirectional Encoder Representations from Transformers (BERT)
    - Masked language modelling, encoder only transformer
Generative Pretrained Transformer (GPT)

- GPT is a decoder only transformer pretrained on huge amounts of text
- The latest version, GPT-3 is trained on 45TB of unlabelled text
- The (pre)training objective is simply to predict the next token
- For this, we will need to slightly tweak the self-attention...

Masked Self-Attention

*Future* timesteps are **masked** to prevent decoder from “peaking”
Next Token Prediction

https://jalammar.github.io/illustrated-gpt2/
Once pretrained, GPT can be used for any “text in, text out” task.
Generative Pretrained Transformer (GPT)

Once pretrained, GPT can be used for any “text in, text out” task

**Grammar correction**

Corrects sentences into standard English.

**Prompt**

Correct this to standard English:

She no went to the market.

**Sample response**

She didn't go to the market.

**English to other languages**

Translates English text into French, Spanish and Japanese.

**Prompt**

Translate this into 1. French, 2. Spanish and 3. Japanese:

What rooms do you have available?

1.

**Sample response**

Quels sont les chambres disponibles?
2. ¿Cuáles son las habitaciones disponibles?
3. 何室がありますか?
Bidirectional Encoder Representations from Transformers (BERT)

- GPT is a **unidirectional** LM, incorporating context from previous tokens
- This is likely sub-optimal for many token- or sentence-level tasks
- BERT proposes a **bidirectional** LM based on a transformer encoder
- BERT is pretrained with two self-supervised objectives:
  - Masked Language Modelling (**MLM**)
  - Next Sentence Prediction (**NSP**)

Masked Language Modelling (MLM)

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

Possible classes:
- All English words
- Improvisation
- Aardvark
- Zyzzyva

Randomly mask 15% of tokens

Input

https://jalammar.github.io/illustrated-bert/
Next Sentence Prediction (NSP)

Predict likelihood that sentence B belongs after sentence A.

Tokenized Input

Input

1% IsNext
99% NotNext

FFNN + Softmax

https://jalammar.github.io/illustrated-bert/
Fine-tuning BERT

- BERT has learned rich representations which encode **syntax & semantics**
  - We can take advantage of this for "downstream" tasks via using **fine-tuning**
- Add & initialize a new layer on top of BERTs outputs
  - Use **supervised learning** to tune all parameters
- Because BERT is pretrained, fine-tuning is typically cheap
  - 3-4 epochs on 100s or 1000s of labelled examples
  - Typically takes a few hours to fine-tune on GPU(s)
- Many, if not most, SOTA methods in NLP incorporate BERT-like models
Fine-tuning BERT

Input Features

85% Spam
15% Not Spam

Classifier
(Feed-forward neural network + softmax)

Help Prince Mayuko Transfer
Huge Inheritance

https://jalammar.github.io/illustrated-bert/
Resources

- For pretrained models and datasets, try [HuggingFace](https://huggingface.co)
- For NLP specific machine learning library, try [AllenNLP](https://allenai.org/)
- For a great free textbook, try [Speech and Language Processing](https://nlp.stanford.edu/tao-book/)
- For a great MOOC, try [Sequence Models](https://www.coursera.org/learn/sequence-models) (free with UofT Coursera)
- For great blog posts illustrating these concepts, try [https://jalammar.github.io/](https://jalammar.github.io/)
Thank you for your attention! (get it?)