Recap: Transformers

\[
\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2
\]

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

Query \( Q = EW^Q \) \quad Values \( V = EW^V \) \quad Keys \( K = EW^K \)

Position encoding: \( E = E + \text{enc}(\text{index}) \)
**Transformer Language Modeling**

What do Transformers produce?

- **Coding of each word** — can pass this to another layer to make a prediction (like predicting the next word for language modeling).
- Like RNNs, Transformers can be viewed as a transformation of a sequence of vectors into a sequence of context-dependent vectors.

**Training Transformer LMs**

- Input is a sequence of words, output is those words shifted by one.
- Allows us to train on predictions across several timesteps simultaneously (similar to batching but this is NOT what we refer to as batching).

\[
P(w|\text{context}) = \text{softmax}(Wh_i)
\]

- \(W\) is a \((\text{vocab size}) \times \text{(hidden size)}\) matrix; linear layer in PyTorch (rows are word embeddings).
Training Transformer LMs

- Loss function: \( \text{loss} = - \log P(w^* | \text{context}) \)
- Total loss: sum of negative log likelihoods at each position

\[
\text{loss}_\text{fcn} = \text{nn.NLLLoss()}
\]
\[
\text{loss} += \text{loss}_\text{fcn} (\text{log_probs, ex.output_tensor})
\]

- Batching is a little tricky with NLLLoss: need to collapse \([\text{batch}, \text{seq} \text{len}, \text{num classes}]\) to \([\text{batch} \times \text{seq} \text{len}, \text{num classes}]\). You do not need to batch.

Batched LM Training

- Multiple sequences and multiple timesteps per sequence

A Small Problem with Transformer LMs

- This Transformer LM as we’ve described it will easily achieve perfect accuracy. Why?

- With standard self-attention: “I” attends to “saw” and the model is “cheating”. How do we ensure that this doesn’t happen?

Attention Masking

- We want to prohibit

- We want to mask out everything in red (an upper triangular matrix)
Implementing in PyTorch

- `nn.TransformerEncoder` can be built out of `nn.TransformerEncoderLayer`, can accept an input and a mask for language modeling:

```python
# Inside the module; need to fill in size parameters
layers = nn.TransformerEncoderLayer [...]  
transformer_encoder = nn.TransformerEncoder(encoder_layers, num_layers=[]) [...]  
# Inside forward(): puts negative infinities in the red part
mask = torch.triu(torch.ones(len, len) * float('-inf'), diagonal=1)  
output = transformer_encoder(input, mask=mask)
```

- You cannot use these for Part 1, only for Part 2

LM Evaluation

- Accuracy doesn’t make sense — predicting the next word is generally impossible so accuracy values would be very low

- Evaluate LMs on the likelihood of held-out data (averaged to normalize for length)

\[
\frac{1}{n} \sum_{i=1}^{n} \log P(w_i|w_1, \ldots, w_{i-1})
\]

- Perplexity: \(\exp(\text{average negative log likelihood})\). Lower is better

  - Suppose we have probs 1/4, 1/3, 1/4, 1/3 for 4 predictions
  - Avg NLL (base e) = 1.242  Perplexity = 3.464 \(\leq\) geometric mean of denominators

Scaling Laws

![Graphs showing scaling laws for different metrics: Test Loss, Dataset Size, Parameters.](image)

Figure 1  Language modeling performance improves smoothly as we increase the model size, dataset size, and amount of compute used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

- Transformers scale really well!  Kaplan et al. (2020)

Takeaways

- Transformers are going to be the foundation for the much of the rest of this class and are a ubiquitous architecture nowadays

- Many details to get right, many ways to tweak and extend them, but core idea is the multi-head self attention and their ability to contextualize items in sequences
What is pre-training?

- “Pre-train” a model on a large dataset for task X, then “fine-tune” it on a dataset for task Y
- Key idea: X is somewhat related to Y, so a model that can do X will have some good neural representations for Y as well
- ImageNet pre-training is huge in computer vision: learn generic visual features for recognizing objects
- GloVe can be seen as pre-training: learn vectors with the skip-gram objective on large data (task X), then fine-tune them as part of a neural network for sentiment/any other task (task Y)

GloVe is insufficient

- GloVe uses a lot of data but in a weak way
- Having a single embedding for each word is wrong
  - they swing the bats  they see the bats
- Identifying discrete word senses is hard, doesn’t scale. Hard to identify how many senses each word has
- How can we make our word embeddings more context-dependent?

ELMo

- CNN over each word => RNN
  
  [Diagram showing Char CNNs for "they", "swing", "the", "bats" processed through 4096-dim LSTMs and 2048 CNN filters projected down to 512-dim]

  Representation of visited (plus vectors from another LM running backwards)

  Peters et al. (2018)
ELMo

- Use the embeddings as a drop-in replacement for GloVe
- Huge gains across many high-profile tasks: NER, question answering, semantic role labeling (similar to parsing), etc.
- But what if the pre-training isn’t only the embeddings?

BERT

- AI2 made ELMo in spring 2018, GPT (transformer-based ELMo) was released in summer 2018, BERT came out October 2018
- Four major changes compared to ELMo:
  - Transformers instead of LSTMs
  - Bidirectional model with “Masked LM” objective instead of standard LM
  - Fine-tune instead of freeze at test time
  - Operates over word pieces (byte pair encoding)

- ELMo is a unidirectional model (as is GPT): we can concatenate two unidirectional models, but is this the right thing to do?
- ELMo reprs look at each direction in isolation; BERT looks at them jointly

A stunning ballet dancer, Copeland is one of the best performers to see live.
**BERT**

- How to learn a “deeply bidirectional” model? What happens if we just replace an LSTM with a transformer?

  ![Diagram of BERT architecture](https://example.com/bert_diagram)

  - John visited Madagascar yesterday

**ELMo (Language Modeling)**

- You could do this with a “one-sided” transformer, but this “two-sided” model can cheat

  ![Diagram of ELMo](https://example.com/elm_diagram)

**Masked Language Modeling**

- How to prevent cheating? Next word prediction fundamentally doesn’t work for bidirectional models, instead do masked language modeling

  ![Diagram of Masked Language Modeling](https://example.com/masked_lm_diagram)

- BERT formula: take a chunk of text, mask out 15% of the tokens, and try to predict them

**Next “Sentence” Prediction**

- Input: [CLS] Text chunk 1 [SEP] Text chunk 2
- 50% of the time, take the true next chunk of text, 50% of the time take a random other chunk. Predict whether the next chunk is the “true” next
- BERT objective: masked LM + next sentence prediction

  ![Diagram of Next Sentence Prediction](https://example.com/next_sentence_diagram)

**BERT Architecture**

- BERT Base: 12 layers, 768-dim per wordpiece token, 12 heads. Total params = 110M

  ![Diagram of BERT Architecture](https://example.com/bert_architecture_diagram)

- BERT Large: 24 layers, 1024-dim per wordpiece token, 16 heads. Total params = 340M

- Positional embeddings and segment embeddings, 30k word pieces
- This is the model that gets pre-trained on a large corpus

**Devlin et al. (2019)**
What can BERT do?

‣ Artificial [CLS] token is used as the vector to do classification from
‣ Sentence pair tasks (entailment): feed both sentences into BERT
‣ BERT can also do tagging by predicting tags at each word piece

Devlin et al. (2019)

Natural Language Inference

Premise Hypothesis
A boy plays in the snow \textit{entails} A boy is outside
A man inspects the uniform of a figure \textit{contradicts} The man is sleeping
An older and younger man smiling \textit{neutral} Two men are smiling and laughing at cats playing

‣ Long history of this task: “Recognizing Textual Entailment” challenge in 2006 (Dagan, Glickman, Magnini)
‣ Early datasets: small (hundreds of pairs), very ambitious (lots of world knowledge, temporal reasoning, etc.)

What can BERT do?

Entails (first sentence implies second is true)

Transformer

[CLS] A boy plays in the snow [SEP] A boy is outside

‣ How does BERT model this sentence pair stuff?
‣ Transformers can capture interactions between the two sentences, even though the NSP objective doesn’t really cause this to happen

SQuAD

Q: What was Marie Curie the first female recipient of?

Passage: One of the most famous people born in Warsaw was Marie Skłodowska-Curie, who achieved international recognition for her research on radioactivity and was the first female recipient of the Nobel Prize. Famous musicians include Władysław Szpilman and Frédéric Chopin. Though Chopin was born in the village of Żelazowa Wola, about 60 km (37 mi) from Warsaw, he moved to the city with his family when he was seven months old. Casimir Pulaski, a Polish general and hero of the American Revolutionary War, was born here in 1745.

Answer = Nobel Prize

‣ Assume we know a passage that contains the answer. More recent work has shown how to retrieve these effectively (will discuss when we get to QA)
SQuAD

Q: What was Marie Curie the first female recipient of?

Passage: One of the most famous people born in Warsaw was Marie Skłodowska-Curie, who achieved international recognition for her research on radioactivity and was the first female recipient of the Nobel Prize. ...

- Predict answer as a pair of (start, end) indices given question q and passage p; compute a score for each word and softmax those

\[
P(\text{start} \mid q, p) = \begin{bmatrix} 0.01 & 0.010 & 0.10 & 0.85 & 0.01 \\ \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\ \text{recipient of the Nobel Prize} \end{bmatrix}
\]

\[
P(\text{end} \mid q, p) = \text{same computation but different params}
\]

QA with BERT

- Predict answer as a pair of (start, end) indices given question q and passage p; compute a score for each word and softmax those

\[
P(\text{start} \mid q, p) = \begin{bmatrix} 0.01 & 0.010 & 0.10 & 0.85 & 0.01 \\ \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\ \text{recipient of the Nobel Prize} \end{bmatrix}
\]

\[
P(\text{end} \mid q, p) = \text{same computation but different params}
\]

What can BERT NOT do?

- BERT **cannot** generate text (at least not in an obvious way)
- Can fill in MASK tokens, but can’t generate left-to-right (well, you could put MASK at the end repeatedly, but this is slow)
- Masked language models are intended to be used primarily for “analysis” tasks

Fine-tuning BERT

- Fine-tune for 1-3 epochs, batch size 2-32, learning rate 2e-5 - 5e-5
- Large changes to weights up here (particularly in last layer to route the right information to [CLS])
- Smaller changes to weights lower down in the transformer
- Small LR and short fine-tuning schedule mean weights don’t change much
- Often requires tricky learning rate schedules (“triangular” learning rates with warmup periods)
BERT Results

Results

- Huge improvements over prior work (even compared to ELMo)
- Effective at “sentence pair” tasks: textual entailment (does sentence A imply sentence B), paraphrase detection

What does BERT learn?

- Heads on transformers learn interesting and diverse things: content heads (attend based on content), positional heads (based on position), etc.

Evaluation: GLUE

<table>
<thead>
<tr>
<th>Corpus</th>
<th>Train</th>
<th>Test</th>
<th>Task</th>
<th>Metrics</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoLA</td>
<td>8.5k</td>
<td>1k</td>
<td>acceptability sentiment</td>
<td>Matthews corr. acc.</td>
<td>misc.</td>
</tr>
<tr>
<td>SST-2</td>
<td>67k</td>
<td>1.8k</td>
<td></td>
<td></td>
<td>movie reviews</td>
</tr>
<tr>
<td>MRPC</td>
<td>3.7k</td>
<td>1.7k</td>
<td>paraphrase</td>
<td>acc./F1</td>
<td>news</td>
</tr>
<tr>
<td>STS-B</td>
<td>7k</td>
<td>1.4k</td>
<td>sentence similarity</td>
<td>Pearson/Spearman corr.</td>
<td>misc.</td>
</tr>
<tr>
<td>QQP</td>
<td>364k</td>
<td>391k</td>
<td>paraphrase</td>
<td>acc./F1</td>
<td>social QA questions</td>
</tr>
<tr>
<td>MNLI</td>
<td>303k</td>
<td>20k</td>
<td>NLI</td>
<td>matched acc./mismatched acc.</td>
<td>misc.</td>
</tr>
<tr>
<td>QNLI</td>
<td>105k</td>
<td>5.4k</td>
<td>QA/NLI</td>
<td>acc.</td>
<td>Wikipedia</td>
</tr>
<tr>
<td>RTE</td>
<td>2.5k</td>
<td>3k</td>
<td>NLI</td>
<td>acc.</td>
<td>news, Wikipedia</td>
</tr>
<tr>
<td>WNLI</td>
<td>654k</td>
<td>146</td>
<td>coreference/NLI</td>
<td>acc.</td>
<td>fiction books</td>
</tr>
</tbody>
</table>

Devlin et al. (2018)

Clark et al. (2019)
What does BERT learn?

- Direct objects attend to their verbs: 86.9% accuracy at the det relation.

- Noun modifiers (e.g., determiners) attend to their nouns: 86.8% accuracy at the det relation.

- Coreferent mentions attend to their antecedents: 65.1% accuracy at linking the head of a coreferent mention to the head of an antecedent.

Still way worse than what supervised systems can do, but interesting that this is learned organically.

Clark et al. (2019)

Takeaways

- Pre-trained models and BERT are very powerful for a range of NLP tasks

- They build on our Transformer language modeling ideas, with modification (e.g., bidirectional nature of BERT)

- These models have enabled big advances in NLI and QA specifically

Handling Rare Words

- Words are a difficult unit to work with. Why?
  - When you have 100,000+ words, the final matrix multiply and softmax start to dominate the computation, many params, still some words you haven’t seen, doesn’t take advantage of morphology, ...

- Character-level models were explored extensively in 2016-2018 but simply don’t work well — becomes very expensive to represent sequences

Subword Tokenization
Subword Tokenization

- Subword tokenization: wide range of schemes that use tokens that are between characters and words in terms of granularity
- These “word pieces” may be full words or parts of words
  _the _eco tax _port i co _in _Po nt - de - Bu is ...
- _ indicates the word piece starting a word (can think of it as the space character).

Sennrich et al. (2016)

Byte Pair Encoding (BPE)

- Start with every individual byte (basically character) as its own symbol
  ```python
  for i in range(num_merges):
      pairs = get_stats(vocab)
      best = max(pairs, key=pairs.get)
  s = vocab = merge_vocab(best, vocab)
  ```
- Count bigram character cooccurrences
- Merge the most frequent pair of adjacent characters
- Doing 8k merges => vocabulary of around 8000 word pieces. Includes many whole words
- Most SOTA NMT systems use this on both source + target

Sennrich et al. (2016)
What’s in the token vocab?

Matthew Watkins
@SoC_trilogy

I’ve just found out that several of the anomalous GPT tokens ("TheNitromeFan", "SolidGoldMagikarp", "davidji", "Smartstocks", "RandomRedditorWithNo") are handles of people who are (competitively? collaboratively?) counting to infinity on a Reddit forum. I kid you not.

<table>
<thead>
<tr>
<th>Rank</th>
<th>User</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>/u/Whoa123</td>
<td>163477</td>
</tr>
<tr>
<td>2</td>
<td>/u/Smartstocks</td>
<td>113829</td>
</tr>
<tr>
<td>3</td>
<td>/u/atomicimploker</td>
<td>103178</td>
</tr>
<tr>
<td>4</td>
<td>/u/TheNitromeFan</td>
<td>85581</td>
</tr>
<tr>
<td>5</td>
<td>/u/SolidGoldMagikarp</td>
<td>65753</td>
</tr>
<tr>
<td>6</td>
<td>/u/RandomRedditorWithNo</td>
<td>63434</td>
</tr>
<tr>
<td>7</td>
<td>/u/rideride</td>
<td>59024</td>
</tr>
<tr>
<td>8</td>
<td>/u/Moorell</td>
<td>57785</td>
</tr>
<tr>
<td>9</td>
<td>/u/Removedpixel</td>
<td>55080</td>
</tr>
<tr>
<td>10</td>
<td>/u/Adinida</td>
<td>48415</td>
</tr>
<tr>
<td>11</td>
<td>/u/scharzoid</td>
<td>47132</td>
</tr>
</tbody>
</table>

Tokenization Today

- **All pre-trained** models use some kind of subword tokenization with a tuned vocabulary; usually between 50k and 250k pieces (larger number of pieces for multilingual models)

- As a result, classical word embeddings like GloVe are **not used**. All subword representations are randomly initialized and learned in the Transformer models