Raw Data

Neural Language Models and Transformers
Neural Language Models

• LMs so far: count-based estimates of probabilities
  
  - Counts are brittle and generalize poorly, so we added smoothing

• The quantity that we are focused on estimating (e.g., for tri-gram model):

\[
p(\bar{x}) = \prod_{i=1}^{n} p(x_i | x_{i-1}, x_{i-2}), \text{ where } x_0, x_{-1} = *, x_i \in \mathcal{V} \cup \{\text{STOP}\}
\]

• Can we use neural networks for this task? What would it give us? What are the costs?
Neural Language Models
A Very Simple Approach

• Instead of having count-based distributions, parameterize them

\[ p(x_i | x_{i-1}, x_{i-2}; \theta) \]

• How would we model this with a neural network?
  - Hint: so far, only learned about MLPs
Neural Language Models
A Very Simple Approach

• A simple MLP-ish model

\[ p(x_i = w | x_{i-1}, x_{i-2}; \theta) = \text{softmax}(y)_w \]

\[ y = b + Wx + U \tanh(d + Hx) \]

\[ x = [\phi(x_{i-1}); \phi(x_{i-2})] \]

where \( \phi \) is an embedding function, and \( \theta = (b, d, W, U, H, C, \phi) \)

• The parameters \( \theta \) are estimated by maximizing the log probability of the data

• During inference, you compute the neural network every time you need a value from the probability distribution

[Bengio et al. 2003]
A simple MLP-ish model

\[ p(x_i = w | x_{i-1}, x_{i-2}; \theta) = \text{softmax}(y)_w \]

\[ y = b + Wx + U \tanh(d + Hx) \]

\[ x = [\phi(x_{i-1}); \phi(x_{i-2})] \]

where \( \phi \) is an embedding function, and \( \theta = (b, d, W, U, H, C, \phi) \)

• What does it give us? Think smoothing …
Neural Language Models
A Very Simple Approach

• A simple MLP-ish model

\[ p(x_i = w | x_{i-1}, x_{i-2}; \theta) = \text{softmax}(y)_w \]

\[ y = b + Wx + U \text{tanh}(d + Hx) \]

\[ x = [\phi(x_{i-1}); \phi(x_{i-2})] \]

where \( \phi \) is an embedding function, and \( \theta = (b, d, W, U, H, C, \phi) \)

• What does it give us? Think smoothing …

\[ \text{softmax}(y)_w = \frac{\exp(y_w)}{\sum_{y \in y} \exp(y)} \]

- What does the softmax do the smoothing problem?

• What are the costs?
Neural Language Models

- The MLP approach can help with smoothing at some costs
- But essentially makes the same modeling choices
  - Assuming a finite horizon — the Markov assumption
  - We adopted this assumption because of sparsity (i.e., smoothing) challenges
- Can neural networks allow us to revisit these assumptions?
Neural Language Models
Revisiting the Markov Assumption

• The Markov assumption was critical for generalization
• But: it’s terrible for natural language!
  - “I ate a strawberry with some cream”
  - “I ate a strawberry that was picked in the field by the best farmer in the world with some cream”
• Dependencies can bridge arbitrarily long linear distances
  - We saw that already with word2vec
• It get even worse beyond the single sentence
Neural Language Models

An MLP with No Markov Assumption

• Without the Markov assumption, the model is

\[ p(\bar{x}) = \prod_{i=1}^{n} p(x_i | x_1, \ldots, x_{i-1}) \]

• We need to model the parameterized distribution

\[ p(x_{i+1} | x_1, \ldots, x_i; \theta) \]

  - Note: shifted the index here, because it will make things nicer later on — just a notation change

• How can we do this with the tools we already know?
Neural Language Models

An MLP with No Markov Assumption

- We need to model the parameterized distribution

\[ p(x_{i+1} \mid x_1, \ldots, x_i; \theta) \]

- We can just treat the context as a bag of words
  - Then it doesn’t matter how long it is
  - A very simple example (two layer MLP)

\[
\begin{align*}
    h &= \tanh(W' \frac{1}{i} \sum_{j=1}^{i} \phi(x_j) + b') \\
    p(x_{i+1} \mid x_1, \ldots, x_i) &= \text{softmax}(W''h + b'')
\end{align*}
\]
Neural Language Models
An MLP with No Markov Assumption

• We can just treat the context as a bag-of-words, for example:

\[ h = \tanh \left( \frac{1}{i} \sum_{j=1}^{i} \phi(x_j) + b' \right) \]

\[ p(x_{i+1} \mid x_1, \ldots, x_i) = \text{softmax}(W''h + b'') \]

• Why is this a terrible idea?
Neural Language Models
An MLP with No Markov Assumption

• We can just treat the context as a bag-of-words, for example:

\[ h = \tanh\left( \sum_{j=1}^{i} W'_{ij} \phi(x_j) + b' \right) \]

\[ p(x_{i+1} | x_1, \ldots, x_i) = \text{softmax}(W''h + b'') \]

• Why is this a terrible idea?
  - Order matters a lot in language 😞
  - But it worked so well for text categorization … 😞
  - What may work for tasks that just require focusing on salient words (e.g., topic categorization), is not sufficient for language models (i.e., next-word prediction)
Neural Language Models
Bag of Words

- BOW can handle arbitrary length 😊
- But losses any notion of order 😞
- Furthermore, dependencies are complex 😢
  - Not following linear order
  - Importance follow complex patterns
    - “I ate a strawberry that was picked in the field by the best farmer in the world with some cream”
    - “I ate a strawberry that was picked in the field by the best farmer in the world with clippers”
  - The model needs to focus on different parts in the context to predict different words
Bag of Words
A Uniform Distribution Over Past Words

- We can view BOW as a attending to all previous tokens equally
- So can rewrite our simple example MLP using a uniform distribution

\[ p(j) = \frac{1}{i}, \quad j = 1, \ldots, i \]
\[ h = \tanh(W' \sum_{j=1}^{i} p(j)\phi(x_j) + b') \]
\[ p(x_{i+1} | x_1, \ldots, x_i) = \text{softmax}(W''h + b'') \]
- What if we want to attend to past tokens in an adaptive way?
Bag of Words
A Uniform Distribution Over Past Words

- We can view BOW as a **attending** to all previous tokens equally
- So can rewrite our simple example MLP using a uniform distribution

\[
p(j) = \frac{1}{i}, \quad j = 1, \ldots, i
\]

\[
h = \tanh(W' \sum_{j=1}^{i} p(j) \phi(x_j) + b')
\]

\[
p(x_{i+1} \mid x_1, \ldots, x_i) = \text{softmax}(W''h + b'')
\]

- What if we want to attend to past tokens in an adaptive way?
  - We need a way to do weighted processing of context to represent that different words depend on context differently
  - This weighted processing must reflect ordering
Attention

- An architecture that functions similar to a soft query-key-value dictionary lookup

- Given a query \( q \in \mathbb{R}^{d_k} \) and a key-value dictionary \( \{ (k^{(i)}, v^{(i)}) \}_{i=1}^{N} \) where \( k^{(i)} \in \mathbb{R}^{d_k}, v^{(i)} \in \mathbb{R}^{d_v} \)

  1. Compute a probability distribution over dictionary entries

      \[
      a_i = q \cdot k^{(i)}, \quad p(i) = \text{softmax}(a)
      \]

  2. Output \( z \in \mathbb{R}^{d_v} \) is weighted average of values:

      \[
      z = \sum_{i=1}^{N} p(i)v^{(i)}
      \]
Self-attention

• Attention where the query, keys, and values come from the same input

• Given a set of vectors \( \{x^{(1)}, \ldots, x^{(N)}\} \) and a query position \( j \in 1, \ldots, N \) we want to create a weighted sum of all vectors

1. Compute query, keys, and values vectors via linear transformation

\[
q = W_q x^{(j)} \quad k^{(i)} = W_k x^{(i)} \quad v^{(i)} = W_v x^{(i)}
\]

2. Compute a probability distribution over dictionary entries

\[
a_i = q \cdot k^{(i)} , \quad p(i) = \text{softmax}(a)
\]

3. Output \( z \in \mathbb{R}^{d_v} \) is weighted average of values: \( z = \sum_{i=1}^{N} p(i)v^{(i)} \)
Self-attention

More Important Details

• Computing attention using loops is crazy slow → it is critical to do everything with a few matrix multiplications by packing all keys and values in matrices $K$ and $V$

• We usually compute for multiple queries $Q$, resulting in multiple outputs $Z$

• Finally, it is common to divide by $\sqrt{d_k}$ because the dot-product is likely to get large in relation the key dimensionality

$$\text{SelfAttn}(Q, K, V) = Z = \text{softmax}(QK/\sqrt{d_k})V$$
LM with Self-attention
From BOW to Self-attention

• Reminder, this is the simple BOW LM we showed earlier

\[ p(j) = \frac{1}{i}, \; j = 1, \ldots, i \]

\[ h = \tanh(W' \sum_{j=1}^{i} p(j) \phi(x_j) + b') \]

\[ p(x_{i+1} | x_1, \ldots, x_i) = \text{softmax}(W''h + b'') \]

• We can easily plug in self-attention to create a weighted processing of the context

• The query is computed from the most recent token

• Keys and values are computed from entire context (i.e., all previous tokens)

• Did we solve the issues with BOW?

  - ✅ Words can’t depend on context differently

  - ❌ Attention is order invariant

\[ q = W_q \phi(x_i) \]

\[ K = W_k[\phi(x_1) \cdots \phi(x_i)] \]

\[ V = W_v[\phi(x_1) \cdots \phi(x_i)] \]

\[ z = \text{SelfAttn}(q, K, V) \]

\[ h = W'' \tanh(W'z + b') + b'' \]

\[ p(x_{i+1} | x_1, \ldots, x_i) = \text{softmax}(h) \]
Marking Positions
Self-attention with Positional Embeddings

- Idea: let’s mark positions
- Learning will figure out what how to use them
- Simple version: learnable embeddings $\phi_p(i)$
- More advanced: fixed embeddings, where values determined by sine waves, with different frequency and offset of each dimensions

Either way, add them to token embeddings

$x_j = \phi(x_j) + \phi_p(j), j = 1, \ldots, i$
$q = W_q x_i$
$K = W_k [x_1 \ldots x_i]$
$V = W_v [x_1 \ldots x_i]$
$z = \text{SelfAttn}(q, K, V)$
$h = W'' \text{tanh}(W' z + b') + b''$
$p(x_{i+1} | x_1, \ldots, x_i) = \text{softmax}(h)$
• Did we solve the issues with BOW?

- ✔ Words can’t depend on context \textbf{differently}

- ✔ Attention is \textbf{order} invariant

• Let’s make it more expressive!

\[ x_j = \phi(x_j) + \phi_p(j), j = 1, \ldots, i \]
\[ q = W_q x_i \]
\[ K = W_k[x_1 \cdots x_i] \]
\[ V = W_v[x_1 \cdots x_i] \]
\[ z = \text{SelfAttn}(q, K, V) \]
\[ h = W''\text{tanh}(W'z + b') + b'' \]
\[ p(x_{i+1} | x_1, \ldots, x_i) = \text{softmax}(h) \]
Self-attention LM

Multiple Attention Heads

• Words need to attend to different elements in context
  \[ x_j = \phi(x_j) + \phi_p(j), \ j = 1, \ldots, i \]

• But attention just does weighted average
  \[ q^{(l)} = W_q^{(l)} x_i \]
  \[ K^{(l)} = W_k^{(l)} [x_1 \cdots x_i] \]
  \[ V^{(l)} = W_v^{(l)} [x_1 \cdots x_i] \]

• So: add more attention heads

Let \( L \) be the number of attention heads

\[ z = [\text{SelfAttn}(q^{(1)}, K^{(1)}, V^{(1)}); \ldots; \text{SelfAttn}(q^{(L)}, K^{(L)}, V^{(L)})] \]

\[ h = W'' \text{tanh}(W'z + b') + b'' \]

\[ p(x_{i+1} | x_1, \ldots, x_i) = \text{softmax}(h) \]
Self-attention LM
Add Neural Network Tricks

- Switch activation to GELU (Gaussian Error Linear Unit)

- Residual connection: shown to help with training very deep networks

- LayerNorm (LN): shown to improve performance
  - Post-norm (original and here)
    \[ b = \text{Module}(\text{LN}(a)) + a \]
  - Pre-norm (modern)
    \[ b = \text{LN}(\text{Module}(a) + a) \]

\[
\begin{align*}
  x_j &= \phi(x_j) + \phi_p(j), j = 1,\ldots, i \\
  q^{(l)} &= W_q^{(l)}x_i \\
  K^{(l)} &= W_k^{(l)}[x_1\cdots x_i] \\
  V^{(l)} &= W_v^{(l)}[x_1\cdots x_i] \\
  z &= \text{LN}([\text{SelfAttn}(q^{(1)}, K^{(1)}, V^{(1)}); \ldots; \\
                  \text{SelfAttn}(q^{(L)}, K^{(L)}, V^{(L)})] + x_i) \\
  h &= \text{LN}(W''\text{GELU}(W'z + b') + b'' + z) \\
  p(x_{i+1} | x_1, \ldots, x_i) &= \text{softmax}(h)
\end{align*}
\]
Self-attention LM

Abstract and Stack It

- Abstract the whole computation as a **Transformer** block
- And stack it

\[
\text{TransformerBlock}^k(u_1, \ldots, u_i)
\]

- \(q^{(l)} = W_q^{(l)}u_i\)
- \(K^{(l)} = W_k^{(l)}[u_1 \cdots u_i]\)
- \(V^{(l)} = W_v^{(l)}[u_1 \cdots u_i]\)
- \(z = \text{LN}([\text{SelfAttn}(q^{(1)}, K^{(1)}, V^{(1)}); \ldots; \text{SelfAttn}(q^{(L)}, K^{(L)}, V^{(L)})] + u_i)\)
- \(h_i^k = \text{LN}(W''\text{GELU}(W'z + b') + b'' + z)\)

\[
x_i = \phi(x_i) + \phi_p(i)
\]
\[
h_i^1 = \text{TransformerBlock}^1(x_1, \ldots, x_i)
\]
\[
h_i^2 = \text{TransformerBlock}^2(h_i^1, \ldots, h_i^1)
\]
\[
\vdots
\]
\[
h_i^K = \text{TransformerBlock}^K(h_i^{K-1}, \ldots, h_i^{K-1})
\]
\[
p(x_{i+1} | x_1, \ldots, x_i) = \text{softmax}(W^{\gamma}h_i^K)
\]
Transformers

- A variable length architecture
  - Was not the first architecture to do that
  - But we are not following the chronological order of events

- Key concept: **self-attention**

- Quickly became maybe the most dominant architecture
  - Try to think why

---

**Attention Is All You Need**

<table>
<thead>
<tr>
<th>Ashish Vaswani*</th>
<th>Noam Shazeer*</th>
<th>Niki Parmar*</th>
<th>Jakob Uszkoreit*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Brain</td>
<td>Google Brain</td>
<td>Google Research</td>
<td>Google Research</td>
</tr>
<tr>
<td><a href="mailto:avaswani@google.com">avaswani@google.com</a></td>
<td><a href="mailto:noam@google.com">noam@google.com</a></td>
<td><a href="mailto:nikip@google.com">nikip@google.com</a></td>
<td><a href="mailto:usz@google.com">usz@google.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LLion Jones*</th>
<th>Aidan N. Gomez*</th>
<th>Łukasz Kaiser*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Research</td>
<td>University of Toronto</td>
<td>Google Brain</td>
</tr>
<tr>
<td><a href="mailto:llion@google.com">llion@google.com</a></td>
<td><a href="mailto:aidan@cs.toronto.edu">aidan@cs.toronto.edu</a></td>
<td><a href="mailto:lukaszkaiser@google.com">lukaszkaiser@google.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Illia Polosukhin*</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:illia.polosukhin@gmail.com">illia.polosukhin@gmail.com</a></td>
</tr>
</tbody>
</table>

[Vaswani et al. 2017]
The Transformer
Decoder-only Variant

TransformerBlock\(^k\)(\(u_1, \ldots, u_i\))

\[ q^{(l)} = W^{(l)}_q u_i \]
\[ K^{(l)} = W^{(l)}_k [u_1 \cdots u_i] \]
\[ V^{(l)} = W^{(l)}_v [u_1 \cdots u_i] \]
\[ z = \text{LN}([\text{SelfAttn}(q^{(1)}, K^{(1)}, V^{(1)}); \cdots; \text{SelfAttn}(q^{(L)}, K^{(L)}, V^{(L)})] + u_i) \]
\[ h_i^k = \text{LN}(W\text{"GELU}(W'z + b') + b'' + z) \]

Self-attention reminder

\[ \text{SelfAttn}(Q, K, V) = \text{softmax}(QK/\sqrt{d_k})V \]

\[ x_i = \phi(x_i) + \phi_p(i) \]
\[ h_i^1 = \text{TransformerBlock}^1(x_1, \ldots, x_i) \]
\[ h_i^2 = \text{TransformerBlock}^2(h_i^1, \ldots, h_i^1) \]
\[ \ldots \]
\[ h_i^k = \text{TransformerBlock}^k(h_i^{k-1}, \ldots, h_i^{k-1}) \]
\[ \ldots \]
\[ h_i^K = \text{TransformerBlock}^K(h_i^{K-1}, \ldots, h_i^{K-1}) \]

\[ p(x_{i+1} | x_1, \ldots, x_i) = \text{softmax}(W^T h_i^K) \]

During learning, compute the whole sequence at ones by **masking** items you shouldn’t attend to in softmax — easy by setting softmax to \(-\infty\)

[Vaswani et al. 2017]
For each time step:

- Input: previous word (and everything computed before)

- Output: probability distribution over the vocabulary

Transformer
Shifted Outputs as Inputs

\[
\begin{align*}
\text{I} & \quad \text{love} \\
\text{I} & \quad \text{Lucy} \\
\text{I} & \quad <s> \\
\text{I} & \quad \text{love} \\
\text{I} & \quad \text{Lucy} \\
\text{I} & \quad </s>
\end{align*}
\]
Transformer

Language Model Training

• Training loss is the per-token negative log likelihood:

\[ \mathcal{L} = - \log p(x_i | x_1, \ldots, x_{i-1}) \]

• During training: we know all tokens
  - So masked self-attention
  - To account for ordering

• Transformers are very sensitive to learning rate schedule → linear warm up + cosine decay
Transformer

Issues

• Time and memory complexity
  - Time: attention is quadratic $O(n^2)$ in sequence length $n$
  - Memory: Need to keep almost all past activation for self-attention

• Positional embeddings mean you can only handle positions up to the length you observed in training

• A lot of existing and ongoing work on both issues
Transformer
Technical Complexities

• Some complexities you will encounter:
  - Masking self-attention
  - Batching
  - Learning rate sensitivity
Transformers

A Success Story

- Transformers were designed with hardware in mind
  - Especially TPUs, but also GPUs
- Exceptionally designed for scale as far as hardware
- Turns out, also scale well for learning
- Unparalleled success in NLP, vision, speech, RL, science, and other areas
Transformers
Natural Language

Decoder-only

Encoder-only

Encoder-decoder

GPT

BERT

T5

Das ist gut.
A storm in Attala caused 6 victims.
This is not toxic.

Translate EN-DE: This is good.
Summarize: state authorities dispatched.
Is this toxic: You look beautiful today!
Transformers
Computer Vision

• ViT: cut image to patches
• Project each patch to a vector
• Treat them as token embeddings

[Dosovitskiy et al. 2020]
Transformers

Speech

• Same as computer vision

• But: spectrograms instead of images

• The Whisper model
Transformers
Reinforcement Learning (RL)

- Decision Transformers
- Inputs are action states and target values
- Value is (in a nutshell) how much reward you want to get
- Outputs are actions
Transformers

Robotics

- Take observations and commands, all tokenized
- Output continuous joint control actions

[Brohan et al. 2023]
Transformers

Everything Everywhere All at Once

• Whatever you can tokenize, the Transformer will take

• What more: you can feed them all to the same model

[Image from: https://deepmind.google/discover/blog/rt-2-new-model-translates-vision-and-language-into-action/]
Acknowledgements

• Some content was adapted from slides by Lucas Beyer

• We thank Greg Durrett, Ana Marasović, and Christian von der Weth for very helpful discussions.