# Transformer Networks

Instructor: John Thickstun

Discussion Board: Available on Ed

Zoom Link: Available on Canvas

Instructor Contact: thickstn@cs.washington.edu

Course Webpage: https://courses.cs.washington.edu/courses/cse599i/20au/

# Autoregressive Modeling

- Factor the joint distribution into conditionals:

$$p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t | x_{<t}).$$

- Learn the conditional distributions $p(x_t | x_{<t})$.

- With an estimator conditioned on finite history:

$$\hat{p}(\mathbf{x}) = \prod_{t=1}^{T} \hat{p}(x_t | x_{<t}) = \hat{p}(x_1)\hat{p}(x_2|x_1)\hat{p}(x_3|x_2, x_1) \cdots \prod_{t=p+1}^{T} \hat{p}(x_t | x_{t-p}, \ldots, x_{t-1}).$$

# Learning the Marginals

- Learn an estimator conditioned on finite history $p$:

$$\hat{p}(\mathbf{x}) = \prod_{t=1}^{T} \hat{p}(x_t | x_{<t}) = \hat{p}(x_1)\hat{p}(x_2|x_1)\hat{p}(x_3|x_2, x_1) \cdots \prod_{t=p+1}^{T} \hat{p}(x_t | x_{t-p}, \ldots, x_{t-1}).$$

- Option A: pad the sequence with $p$ “start” tokens.

- Option B: learn the marginals directly with NADE.

# NADE Marginal Estimation

- Neural Autoregressive Distribution Estimation (NADE):

$$\mathbf{h}_t = \text{sigmoid}\left(\mathbf{c}_t + \sum_{s<t} \mathbf{V}_s x_s\right), \qquad \mathbf{V} \in \mathbb{R}^{p \times k \times d}, \mathbf{c} \in \mathbb{R}^{p \times k},$$

$$f_t(\mathbf{x}_{<t}) = \mathbf{b}_t + \sum_{s \leq t} \mathbf{W}_s \mathbf{h}_s, \qquad \mathbf{W} \in \mathbb{R}^{p \times d \times k}, \mathbf{b} \in \mathbb{R}^{p \times d}.$$

- Learn $p_t(x_t | x_1, \ldots, x_{t-1}) = \text{Categorical}(\text{softmax}(f_t(x_1, \ldots, x_{t-1})))$.

- For each $t = 1, \ldots, p$. IMPORTANT: this helps regularize the optimization!

# AR Models in the Wild

- OpenAI GPT-2/GPT-3 Models are AR models:

  ➡ Parameterized by Transformer Networks (today's lecture).

- DeepMind's WaveNet models are AR models:

  ➡ Parameterized by ConvNets (next week).

- VQ-VAE models are a mashup of AR models and VAE's (next week):

  ➡ Introducing VAE-based neural compression helps AR models scale.

# Transformer Networks

- A **transformer block** is a family of functions $b : \mathbb{R}^{p \times d} \to \mathbb{R}^{p \times d}$.
- A **transformer** is a stack of blocks: $f = b^{(L)} \circ \cdots \circ b^{(1)} : \mathbb{R}^{p \times d} \to \mathbb{R}^{p \times d}$.
- We'll use a transformer to parameterize a NADE.

$$p_t(x_t | x_1, \ldots, x_{t-1}) = \text{Categorical}(\text{softmax}(f_t(x_1, \ldots, x_{t-1}))).$$

- Prefix function is defined by masking: $f_t(x_1, \ldots, x_{t-1}) = f(\mathbf{m}_t \odot \mathbf{x})$.

# Transformer Blocks

| Transformation | Weights |
| --- | --- |
| $Q^{(h)}(\mathbf{x}_i) = W_{h,q}^T \mathbf{x}_i,\ K^{(h)}(\mathbf{x}_i) = W_{h,k}^T \mathbf{x}_i,\ V^{(h)}(\mathbf{x}_i) = W_{h,v}^T \mathbf{x}_i,$ | $W_{h,q}, W_{h,k}, W_{h,v} \in \mathbb{R}^{d \times k},$ |
| $\alpha_{i,j}^{(h)} = \text{softmax}_j \left( \frac{\langle Q^{(h)}(\mathbf{x}_i), K^{(h)}(\mathbf{x}_j) \rangle}{\sqrt{k}} \right),$ | |
| $\mathbf{u}_i' = \sum_{h=1}^{H} W_{c,h}^T \sum_{j=1}^{p} \alpha_{i,j} V^{(h)}(\mathbf{x}_j),$ | $W_{c,h} \in \mathbb{R}^{k \times d},$ |
| $\mathbf{u}_i = \text{LayerNorm}(\mathbf{x}_i + \mathbf{u}_i'; \gamma_1, \beta_1),$ | $\gamma_1, \beta_1 \in \mathbb{R},$ |
| $\mathbf{z}_i' = W_2^T \text{ReLU}(W_1^T \mathbf{u}_i),$ | $W_1 \in \mathbb{R}^{d \times m}, W_2 \in \mathbb{R}^{m \times d},$ |
| $\mathbf{z}_i = \text{LayerNorm}(\mathbf{u}_i + \mathbf{z}_i'; \gamma_2, \beta_2),$ | $\gamma_2, \beta_2 \in \mathbb{R}.$ |

# Multi-Headed Attention

| Transformation | Weights |
| --- | --- |
| $Q^{(h)}(\mathbf{x}_i) = W_{h,q}^T \mathbf{x}_i,\ K^{(h)}(\mathbf{x}_i) = W_{h,k}^T \mathbf{x}_i,\ V^{(h)}(\mathbf{x}_i) = W_{h,v}^T \mathbf{x}_i,$ | $W_{h,q}, W_{h,k}, W_{h,v} \in \mathbb{R}^{d \times k},$ |
| $\alpha_{i,j}^{(h)} = \text{softmax}_j \left( \frac{\langle Q^{(h)}(\mathbf{x}_i), K^{(h)}(\mathbf{x}_j) \rangle}{\sqrt{k}} \right),$ | |
| $\mathbf{u}_i' = \sum_{h=1}^{H} W_{c,h}^T \sum_{j=1}^{p} \alpha_{i,j} V^{(h)}(\mathbf{x}_j),$ | $W_{c,h} \in \mathbb{R}^{k \times d},$ |
| $\mathbf{u}_i = \text{LayerNorm}(\mathbf{x}_i + \mathbf{u}_i'; \gamma_1, \beta_1),$ | $\gamma_1, \beta_1 \in \mathbb{R},$ |
| $\mathbf{z}_i' = W_2^T \text{ReLU}(W_1^T \mathbf{u}_i),$ | $W_1 \in \mathbb{R}^{d \times m}, W_2 \in \mathbb{R}^{m \times d},$ |
| $\mathbf{z}_i = \text{LayerNorm}(\mathbf{u}_i + \mathbf{z}_i'; \gamma_2, \beta_2),$ | $\gamma_2, \beta_2 \in \mathbb{R}.$ |

# Multi-Headed Attention

| Transformation | Weights |
| --- | --- |
| $Q^{(h)}(\mathbf{x}_i) = W_{h,q}^T \mathbf{x}_i,\ K^{(h)}(\mathbf{x}_i) = W_{h,k}^T \mathbf{x}_i,\ V^{(h)}(\mathbf{x}_i) = W_{h,v}^T \mathbf{x}_i,$ | $W_{h,q}, W_{h,k}, W_{h,v} \in \mathbb{R}^{d \times k},$ |
| $\alpha_{i,j}^{(h)} = \text{softmax}_j \left( \frac{\langle Q^{(h)}(\mathbf{x}_i), K^{(h)}(\mathbf{x}_j) \rangle}{\sqrt{k}} \right),$ | |
| $\mathbf{u}_i' = \sum_{h=1}^{H} W_{c,h}^T \sum_{j=1}^{p} \alpha_{i,j} V^{(h)}(\mathbf{x}_j),$ | $W_{c,h} \in \mathbb{R}^{k \times d},$ |

- Could think of this like a learned, differentiable lookup table.

- "Queries" $Q$, "Keys" $K$, and "Values" $V$. "Attention" values $\alpha^{(h)} \in \mathbb{R}^{p \times p}$.

# Multi-Headed Attention

**Transformation**

$$Q^{(h)}(\mathbf{x}_i) = W_{h,q}^T \mathbf{x}_i,\ K^{(h)}(\mathbf{x}_i) = W_{h,k}^T \mathbf{x}_i,\ V^{(h)}(\mathbf{x}_i) = W_{h,v}^T \mathbf{x}_i,$$

$$\alpha_{i,j}^{(h)} = \text{softmax}_j \left( \frac{\langle Q^{(h)}(\mathbf{x}_i), K^{(h)}(\mathbf{x}_j) \rangle}{\sqrt{k}} \right),$$

$$\mathbf{u}_i' = \sum_{h=1}^{H} W_{c,h}^T \sum_{j=1}^{p} \alpha_{i,j} V^{(h)}(\mathbf{x}_j),$$

**Weights**

$$W_{h,q}, W_{h,k}, W_{h,v} \in \mathbb{R}^{d \times k},$$

$$W_{c,h} \in \mathbb{R}^{k \times d},$$

- Could think of this like a kernel method, especially if $Q = K$.

- Do we even need separate $Q, K$ transformations? (Course project… ?)

# Multi-Headed Attention

| Transformation | Weights |
| --- | --- |
| $Q^{(h)}(\mathbf{x}_i) = W_{h,q}^T \mathbf{x}_i,\ K^{(h)}(\mathbf{x}_i) = W_{h,k}^T \mathbf{x}_i,\ V^{(h)}(\mathbf{x}_i) = W_{h,v}^T \mathbf{x}_i,$ | $W_{h,q}, W_{h,k}, W_{h,v} \in \mathbb{R}^{d \times k},$ |
| $\alpha_{i,j}^{(h)} = \text{softmax}_j \left( \frac{\langle Q^{(h)}(\mathbf{x}_i), K^{(h)}(\mathbf{x}_j) \rangle}{\sqrt{k}} \right),$ | |
| $\mathbf{u}_i' = \sum_{h=1}^{H} W_{c,h}^T \sum_{j=1}^{p} \alpha_{i,j} V^{(h)}(\mathbf{x}_j),$ | $W_{c,h} \in \mathbb{R}^{k \times d},$ |

- The weights aren't dependent on sequence length!

- Interaction terms between positions don't have explicit weights.

# Position-wise Feedforward

| Transformation | Weights |
|---|---|
| $Q^{(h)}(\mathbf{x}_i) = W_{h,q}^T \mathbf{x}_i,\ K^{(h)}(\mathbf{x}_i) = W_{h,k}^T \mathbf{x}_i,\ V^{(h)}(\mathbf{x}_i) = W_{h,v}^T \mathbf{x}_i,$ | $W_{h,q}, W_{h,k}, W_{h,v} \in \mathbb{R}^{d \times k},$ |
| $\alpha_{i,j}^{(h)} = \text{softmax}_j \left( \frac{\langle Q^{(h)}(\mathbf{x}_i), K^{(h)}(\mathbf{x}_j) \rangle}{\sqrt{k}} \right),$ | |
| $\mathbf{u}_i' = \sum_{h=1}^{H} W_{c,h}^T \sum_{j=1}^{p} \alpha_{i,j} V^{(h)}(\mathbf{x}_j),$ | $W_{c,h} \in \mathbb{R}^{k \times d},$ |
| $\mathbf{u}_i = \text{LayerNorm}(\mathbf{x}_i + \mathbf{u}_i'; \gamma_1, \beta_1),$ | $\gamma_1, \beta_1 \in \mathbb{R},$ |
| $\mathbf{z}_i' = W_2^T \text{ReLU}(W_1^T \mathbf{u}_i),$ | $W_1 \in \mathbb{R}^{d \times m}, W_2 \in \mathbb{R}^{m \times d},$ |
| $\mathbf{z}_i = \text{LayerNorm}(\mathbf{u}_i + \mathbf{z}_i'; \gamma_2, \beta_2),$ | $\gamma_2, \beta_2 \in \mathbb{R}.$ |

# Position-wise Feedforward

**Transformation**

$$\mathbf{u}_i = \mathrm{LayerNorm}(\mathbf{x}_i + \mathbf{u}'_i; \gamma_1, \beta_1),$$
$$\mathbf{z}'_i = W_2^T \mathrm{ReLU}(W_1^T \mathbf{u}_i),$$
$$\mathbf{z}_i = \mathrm{LayerNorm}(\mathbf{u}_i + \mathbf{z}'_i; \gamma_2, \beta_2),$$

**Weights**

$$\gamma_1, \beta_1 \in \mathbb{R},$$
$$W_1 \in \mathbb{R}^{d \times m}, W_2 \in \mathbb{R}^{m \times d},$$
$$\gamma_2, \beta_2 \in \mathbb{R}.$$

- Forget the normalizing transformations for a moment.
- Two-layer, fully-connected ReLU network.
- Applied per-position (no interaction between terms in the sequence).

# Transformer Blocks

| Transformation | Weights |
| --- | --- |
| $Q^{(h)}(\mathbf{x}_i) = W_{h,q}^T \mathbf{x}_i,\ K^{(h)}(\mathbf{x}_i) = W_{h,k}^T \mathbf{x}_i,\ V^{(h)}(\mathbf{x}_i) = W_{h,v}^T \mathbf{x}_i,$ | $W_{h,q}, W_{h,k}, W_{h,v} \in \mathbb{R}^{d \times k},$ |
| $\alpha_{i,j}^{(h)} = \text{softmax}_j \left( \frac{\langle Q^{(h)}(\mathbf{x}_i), K^{(h)}(\mathbf{x}_j) \rangle}{\sqrt{k}} \right),$ | |
| $\mathbf{u}_i' = \sum_{h=1}^{H} W_{c,h}^T \sum_{j=1}^{p} \alpha_{i,j} V^{(h)}(\mathbf{x}_j),$ | $W_{c,h} \in \mathbb{R}^{k \times d},$ |
| $\mathbf{u}_i = \text{LayerNorm}(\mathbf{x}_i + \mathbf{u}_i'; \gamma_1, \beta_1),$ | $\gamma_1, \beta_1 \in \mathbb{R},$ |
| $\mathbf{z}_i' = W_2^T \text{ReLU}(W_1^T \mathbf{u}_i),$ | $W_1 \in \mathbb{R}^{d \times m}, W_2 \in \mathbb{R}^{m \times d},$ |
| $\mathbf{z}_i = \text{LayerNorm}(\mathbf{u}_i + \mathbf{z}_i'; \gamma_2, \beta_2),$ | $\gamma_2, \beta_2 \in \mathbb{R}.$ |

# 5-Minute Break

# Layer Normalization

| Transformation | Weights |
| --- | --- |
| $\mathrm{LayerNorm}(\mathbf{z};\gamma,\beta) = \gamma\frac{(\mathbf{z}-\mu_{\mathbf{z}})}{\sigma_{\mathbf{z}}} + \beta,$ | $\gamma, \beta \in \mathbb{R}^k.$ |
| $\mu_{\mathbf{z}} = \frac{1}{k}\sum_{i=1}^{k}\mathbf{z}_i, \quad \sigma_{\mathbf{z}} = \sqrt{\frac{1}{k}\sum_{i=1}^{k}(\mathbf{z}_i - \mu_{\mathbf{z}})^2}.$ | |

- Normalize activations of a layer to have learned mean $\beta$ and variance $\gamma^2$.

- Applied per-sequence (not across batches).

# Positional Encoding

| Transformation | Weights |
| --- | --- |
| $\mathbf{z}_k = W_z^T \mathrm{ReLU}(W_x^T \mathbf{x}_k + W_e^T \mathbf{e}_k) \in \mathbb{R}^d,$ | $W_x \in \mathbb{R}^{d \times m}, W_e \in \mathbb{R}^{p \times m}, W_z \in \mathbb{R}^{m \times d}.$ |

- Transformer doesn't account for sequence position.
- Need to encode sequence position as data.
- Learn positional embeddings, or joint token-position embeddings.

# Dropout

| Transformation | Weights |
| --- | --- |
| $Q^{(h)}(\mathbf{x}_i) = W_{h,q}^T \mathbf{x}_i,\ K^{(h)}(\mathbf{x}_i) = W_{h,k}^T \mathbf{x}_i,\ V^{(h)}(\mathbf{x}_i) = W_{h,v}^T \mathbf{x}_i,$ | $W_{h,q}, W_{h,k}, W_{h,v} \in \mathbb{R}^{d \times k},$ |
| $\alpha_{i,j}^{(h)} = \text{Dropout}\left(\text{softmax}_j\left(\frac{\langle Q^{(h)}(\mathbf{x}_i), K^{(h)}(\mathbf{x}_j)\rangle}{\sqrt{k}}\right)\right),$ | |
| $\mathbf{u}_i' = \sum_{h=1}^{H} W_{c,h}^T \sum_{j=1}^{p} \alpha_{i,j} V^{(h)}(\mathbf{x}_j),$ | $W_{c,h} \in \mathbb{R}^{k \times d},$ |
| $\mathbf{u}_i = \text{LayerNorm}(\mathbf{x}_i + \text{Dropout}(\mathbf{u}_i'); \gamma_1, \beta_1),$ | $\gamma_1, \beta_1 \in \mathbb{R},$ |
| $\mathbf{z}_i' = W_2^T \text{Dropout}\left(\text{ReLU}(W_1^T \mathbf{u}_i)\right),$ | $W_1 \in \mathbb{R}^{d \times m}, W_2 \in \mathbb{R}^{m \times d},$ |
| $\mathbf{z}_i = \text{LayerNorm}(\mathbf{u}_i + \text{Dropout}(\mathbf{z}_i'); \gamma_2, \beta_2),$ | $\gamma_2, \beta_2 \in \mathbb{R}.$ |

# Dataset Augmentation

- Input dropout: randomly mask word in the history (uncertain history).
- Embedding dropout: randomly mask dimensions in the input embeddings.
- Makes a big difference! At increased computational cost….
- Wikitext-2 without augmentation: ~100ppl, with augmentation: ~60ppl

https://twitter.com/Tim_Dettmers/status/1247998807494684672

# Hyper-Parameter Selection

- Width of the transformer block $d$.
- Width of the internal position-wise feedforward network $m$.
- Width of each transformer head $k$.
- Number of transformer heads $H$.
- Depth of the stack of transformer blocks $L$.
- Context length $p$.
- All the usual hyper-parameters: dropout, l2, learning rates, initialization…

# Comparison to RNN/LSTM

- RNN/LSTM maintain a running state:

$$\begin{aligned}
\mathbf{h}_0 &= \mathbf{W}_{\text{init}}, & \mathbf{W}_{\text{init}} &\in \mathbb{R}^k, \\
\mathbf{h}_t &= \text{sigmoid}(\mathbf{c} + \mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t), & \mathbf{W}_h &\in \mathbb{R}^{k \times k}, \mathbf{W}_x \in \mathbb{R}^{k \times d}, \mathbf{c} \in \mathbb{R}^k, \\
f_t(\mathbf{x}_{<t}) &= \mathbf{b} + \mathbf{W}_o \mathbf{h}_{t-1}, & \mathbf{W}_o &\in \mathbb{R}^{d \times k}, \mathbf{b} \in \mathbb{R}^d.
\end{aligned}$$

- Relatively fast to sample from RNN/LSTM.
- Relatively slow to train RNN/LSTM: more serial computation.
- Easier to learn long-term dependencies in transformer networks?