Week 9: Sequence Modeling, Attention, and Transformers

1. Recurrent Neural Networks (RNN)

1.1 Vanilla RNN

- **Structure:** Processes sequential data by maintaining a hidden state h_t that acts as a memory of previous inputs.
- **Forward Pass:**
  - $h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b)$
  - $y_t = W_{hy}h_t + b_y$
  - Loss: $L_t = \frac{1}{2}(y_t - \hat{y}_t)^2$ (example using MSE). Total Loss $L = \sum L_t$.
- **Backpropagation Through Time (BPTT):**
  - Gradients are calculated by unrolling the network through time.
  - Gradient of loss with respect to weights W:

$$\frac{\partial L}{\partial W} = \sum_t \frac{\partial L_t}{\partial W} = \sum_t \sum_{k=1}^{t} \frac{\partial L_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

  - **The Chain Rule Problem:** The term $\frac{\partial h_t}{\partial h_k}$ involves a product of Jacobians $\prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}}$.
- **Issues:**
  - **Vanishing Gradient:** If singular values of $W_{hh} < 1$, gradients approach 0 exponentially as the sequence length increases. Long-term dependencies are lost.
  - **Exploding Gradient:** If singular values > 1, gradients grow exponentially.
  - **Solution (Exploding):** Gradient Clipping (if $\|g\| > \text{threshold}$, scale it down).

1.2 Long Short-Term Memory (LSTM)

Designed to solve the vanishing gradient problem by maintaining a separate cell state C_t with additive updates.

- **Gates:**
  - **Input gate (i):** Controls how much new information flows into the cell.
  - **Forget gate (f):** Controls how much past information is retained.

- **Output gate (o):** Controls the output based on the cell state.
- **Gate formulation:**

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

- **State Updates:**
  - **Cell State:** $C_t = f_t \odot C_{t-1} + i_t \odot g_t$
  - **Hidden State:** $h_t = o_t \odot \tanh(C_t)$
- **Gradient Flow:** The additive nature of the cell state update ($+$) creates a "gradient superhighway," allowing uninterrupted gradient flow during backpropagation:

$$\frac{\partial C_t}{\partial C_{t-1}} = f_t + \dots$$

(This additive term prevents the gradient from collapsing to zero).

1.3 Sequence-to-Sequence & Image Captioning

- **Language Modeling:** Predict next word likelihood: $P(x_t|x_{t-1}, \dots, x_1)$.
- **Inference:**
  - **Greedy Decoding:** Select argmax at each step.
  - **Beam Search:** Keep top-k sequences at each step to avoid suboptimal greedy choices.
- **Image Captioning:** Uses a Pretrained CNN (e.g., ImageNet) as an Encoder and an RNN as a Decoder.
  - Input: Feature vector from CNN FC layer (or spatial features).
  - Training: Minimizing Cross-Entropy Loss between generated word and ground truth.

2. Attention Mechanisms

2.1 Visual Attention

Instead of compressing an entire image into a single vector, the model focuses on specific spatial locations.

- **Spatial Features:** Extracted from a convolutional layer (e.g., $14 \times 14 \times 512$).
- **Attention Score:** Calculated using the decoder's hidden state h_{t-1} and image features.
  - Score: $e = f_{att}(X, h_{t-1})$

    - Weights: $\alpha = \text{softmax}(e)$
    - **Context Vector (z_t):** Weighted sum of features: $z_t = \sum \alpha_{ij} x_{ij}$.
- This allows the model to align specific words (e.g., "bird") with specific image regions.

2.2 Types of Attention

- **Spatial Attention:** Attention over $H \times W$ grid.
- **Channel Attention (Squeeze-and-Excitation):** Reweights feature channels based on global importance.
    - Squeeze: Global Average Pooling.
    - Excitation: Fully Connected layers to predict channel-wise weights.

3. The Transformer

3.1 Self-Attention

Process inputs X into Query (Q), Key (K), and Value (V) matrices via linear projections.

- **Scaled Dot-Product Attention:**

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

    - Division by $\sqrt{d_k}$ prevents the dot product from growing too large, which would push softmax into regions with small gradients.
- **Multi-Head Attention:** Perform attention h times in parallel with different learnable projections, allowing the model to attend to different representation subspaces.

3.2 Transformer Architecture

- **Positional Encoding:** Since attention is permutation invariant, sine and cosine functions are added to embeddings to inject order information.

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

- **Encoder Block:**
    i. Multi-Head Attention

ii. Add & Norm (Residual connection + Layer Normalization)
iii. Feed-Forward Network (MLP)
iv. Add & Norm

- **Decoder:** Similar to encoder but includes "Masked" Self-Attention (prevents attending to future tokens during training).

3.3 Linearization of Attention

The complexity of standard attention is $O(N^2)$ (quadratic w.r.t sequence length).

- **Kernel Trick:** Approximating softmax allows calculating $K^T V$ first, reducing complexity to $O(N)$ (linear).

$$\text{softmax}(QK^T)V \approx \phi(Q)(\phi(K)^T V)$$

4. Vision Transformer (ViT) & Variants

4.1 ViT Architecture

Adapts Transformer for images by treating image patches as "words".

1. **Patching:** Divide image into fixed-size patches (e.g., 16×16).
2. **Linear Projection:** Flatten patches and map to embedding dimension D.
3. **Class Token:** Prepend a learnable [CLS] token (used for final classification).
4. **Position Embedding:** Add learnable embeddings to retain spatial structure.
5. **Transformer Encoder:** Standard blocks with Layer Norm (pre-norm configuration often used).
6. **MLP Head:** Classification based on the output of the [CLS] token.

4.2 ViT vs. CNN

- **Inductive Bias:** CNNs have high inductive bias (Locality, Translation Equivariance). ViTs have low inductive bias and learn relationships globally from data (requires more data).
- **Global vs. Local:** ViT allows global receptive field from the first layer. CNNs grow receptive fields gradually.

4.3 Swin Transformer

Introduces **Locality** back into Transformers for efficiency and better vision performance.

- **Windowed Attention:** Compute self-attention only within local windows (reduces complexity).

- **Shifted Windows:** Shifts window partitioning in successive layers to allow cross-window connections.
- **Patch Merging:** Hierarchically reduces resolution (like pooling in CNNs) to create multi-scale feature maps.

Week 10: Generative Models

1. Introduction to Generative Models

- **Goal:** Learn a probability distribution $P_{model}(x)$ that approximates the true data distribution $P_{data}(x)$.
- **Objective:** Minimize the **KL Divergence** between P_{data} and P_θ (model parameters).

$$\min_\theta D_{KL}(P_{data} \| P_\theta) = \min_\theta E_{x \sim P_{data}} \left[\log \frac{P_{data}(x)}{P_\theta(x)} \right]$$

  - This is equivalent to **Maximum Likelihood Estimation (MLE)**:

$$\max_\theta E_{x \sim P_{data}} [\log P_\theta(x)]$$

2. Autoregressive Models

Models that generate data one element at a time, conditioned on previous elements.

- **Formula:** $P(x) = \prod_{i=1}^{n} P(x_i | x_1, \ldots, x_{i-1})$.
- **PixelRNN:**
  - Uses LSTM layers (Row LSTM or Diagonal BiLSTM) to predict pixels sequentially.
  - Generation is slow (sequential).
- **PixelCNN:**
  - Uses **Masked Convolutions** to enforce the autoregressive property (a pixel only sees previous pixels).
  - Training is parallelizable, but generation is still sequential.

3. Latent Variable Models & Autoencoders

3.1 Autoencoders (AE)

- **Encoder:** Maps input x to a lower-dimensional latent vector z.
- **Decoder:** Reconstructs x from z.
- **Limitations:** The latent space is not regularized, making it poor for generation (sampling random z often yields noise).

3.2 Variational Autoencoders (VAE)

A generative model that learns the distribution of latent variables.

- **Problem:** We want to maximize log-likelihood $\log P_\theta(x) = \log \int P_\theta(x|z)P(z)dz$. The integral is intractable.
- **Solution:** Variational Inference. Introduce an approximate posterior $q_\phi(z|x)$.

3.3 Derivation of the ELBO (Evidence Lower Bound)

The objective is to maximize the lower bound of the log-likelihood.

$$\log P_\theta(x) = \log \int P_\theta(x|z)P(z)\frac{q_\phi(z|x)}{q_\phi(z|x)}dz$$

$$= \log E_{z\sim q_\phi}\left[\frac{P_\theta(x|z)P(z)}{q_\phi(z|x)}\right]$$

By Jensen's Inequality ($\log E[y] \geq E[\log y]$):

$$\log P_\theta(x) \geq E_{z\sim q_\phi}\left[\log P_\theta(x|z) + \log P(z) - \log q_\phi(z|x)\right]$$

$$\text{ELBO} = E_{z\sim q_\phi}[\log P_\theta(x|z)] - D_{KL}(q_\phi(z|x)\|P(z))$$

- **Term 1 (Reconstruction Loss):** Encourages the decoder to reconstruct x accurately.
- **Term 2 (Regularization):** Forces the latent distribution $q_\phi(z|x)$ to be close to the prior $P(z)$ (usually $N(0, I)$).

3.4 The Reparameterization Trick

To perform backpropagation through the sampling process $z \sim N(\mu_\phi, \sigma_\phi^2)$, we cannot simply sample.

- **Trick:** Express z as a deterministic transformation of a noise variable ϵ.

$$\epsilon \sim N(0, I)$$

$$z = \mu_\phi + \sigma_\phi \odot \epsilon$$

- This makes z differentiable with respect to μ_ϕ and σ_ϕ.

3.5 Analytic KL Divergence

Since both the prior $P(z)$ and approximate posterior $q_\phi(z|x)$ are Gaussian, the KL term can be computed analytically.
Given $q(z|x) = N(\mu, \sigma^2)$ and $P(z) = N(0, I)$:

$$D_{KL}(N(\mu, \sigma^2) \| N(0, I)) = \frac{1}{2} \sum_{j=1}^{J} (1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2)$$

- **Derivation involves:**
  i. Writing out the definition of KL integral.
  ii. Using the **Trace Trick** for expectations of quadratic forms: $E[x^T A x] = \mathrm{Tr}(A\Sigma) + \mu^T A \mu$.
  iii. Simplifying terms based on properties of Gaussian distributions.