Part 1: Sequence Modeling and Recurrent Neural Networks

1. Vanilla Recurrent Neural Networks (RNN)

Structure:
RNNs process sequence data ($x_1, x_2, \dots, x_t$) by maintaining a hidden state h_t that acts as a summary of the past.

- **Update Rule:** $h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$
- **Output:** $Y_t = W_{hy}h_t + b_y$
- **Loss:** The total loss is the sum of losses at each time step: $L = \sum_t L_t$.

Backpropagation Through Time (BPTT):
To train the RNN, we compute gradients relative to parameters (e.g., W). The gradient flows back through time:

$$\frac{\partial L}{\partial W} = \sum_t \frac{\partial L_t}{\partial W}$$

Applying the chain rule exposes a product of Jacobian matrices:

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}}$$

Gradient Problems:

- **Vanishing Gradient:** If the singular values of the weight matrix are < 1, or due to the derivative of $\tanh$ (which is ≤ 1), the gradient shrinks exponentially as it propagates back. This makes learning long-term dependencies impossible ($||\frac{\partial h_t}{\partial h_{t-n}}|| \ll 1$).
- **Exploding Gradient:** If singular values are > 1, gradients grow exponentially.
  - *Solution:* **Gradient Clipping** (scaling down the gradient vector if its norm exceeds a threshold).

2. Long Short-Term Memory (LSTM)

Designed to solve the vanishing gradient problem by introducing a **Cell State (C_t)** separate from the hidden state.

Gates:

- **Forget Gate (f_t):** Decides what to discard from C_{t-1}.
- **Input Gate (i_t):** Decides which values to update.
- **Gate Gate (g_t):** Candidate values for the state.
- **Output Gate (o_t):** Decides what to output based on the cell state.

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

Cell Update & Gradient Flow:

$$C_t = f_t \odot C_{t-1} + i_t \odot g_t$$

$$h_t = o_t \odot \tanh(C_t)$$

Key Insight (The Gradient Superhighway):
During backpropagation, the gradient for the cell state relates to:

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

The additive nature of the update (unlike the matrix multiplication in vanilla RNNs) creates an "uninterrupted gradient flow," allowing gradients to propagate effectively over long sequences. C_t acts as long-term memory, while h_t acts as short-term memory.

Part 2: Attention and Transformers

1. Sequence-to-Sequence & Attention

Language Modeling:
Goal: Predict the next word probability $P(x_t|x_{t-1}, \dots, x_1)$.

- **Training:** Cross-Entropy Loss between predicted distribution and one-hot target.
- **Inference:**
    - *Greedy Decoding:* Pick the highest probability word (can lead to suboptimal sentences).
    - *Stochastic:* Sample based on probability distribution.

Attention Mechanism:
In Encoder-Decoder models (e.g., Image Captioning or Translation), the fixed-size context vector is a bottleneck. Attention allows the decoder to focus on different parts of the input sequence (or image features) at every time step.

Definition:

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

- **Query (Q):** What I am looking for (e.g., current decoder state).
- **Key (K):** What features I have (e.g., encoder states).
- **Value (V):** The actual content to retrieve (usually same as K).
- **Scaling ($\sqrt{d_k}$):** Prevents the dot product from becoming too large, which would push the softmax into regions with extremely small gradients.

2. The Transformer Architecture

Replaces recurrence entirely with Self-Attention.

Key Components:

1. **Positional Encoding:** Since there is no recurrence, absolute position information is injected into the embeddings using sine and cosine functions of different frequencies.

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

2. **Multi-Head Attention:** Projects queries, keys, and values h times with different linear projections. Allows the model to attend to information from different representation subspaces jointly.
3. **Masked Self-Attention (Decoder):** Masks future positions (setting attention scores to $-\infty$) so the model cannot "cheat" by seeing words it hasn't generated yet.
4. **Feed-Forward Networks (FFN):** applied position-wise.
5. **Add & Norm:** Residual connections followed by Layer Normalization.

3. Vision Transformer (ViT)

Applying the Transformer encoder directly to images.

Process:

1. **Patching:** Split image into fixed-size patches (e.g., 16×16).
2. **Flattening & Projection:** Flatten patches and map to D dimensions via linear projection.
3. **Position Embedding:** Add learnable position vectors.
4. **Class Token:** Prepend a learnable [CLS] token. Its state at the output of the Transformer serves as the image representation for classification.

Inductive Bias Comparison:

- **CNN:** High inductive bias (Translation Equivariance, Locality). Efficient with less data.
- **ViT:** Low inductive bias (Global connectivity). Requires massive datasets to learn spatial relationships but eventually outperforms CNNs due to flexibility.

Swin Transformer: Introduces "Shifted Windows" to calculate self-attention locally within windows, improving efficiency from quadratic $O(N^2)$ to linear $O(N)$ w.r.t image size, while maintaining global connections across layers.

Part 3: Generative Models

1. Overview

Goal: Density Estimation. Given data X, we want to learn a model P_θ such that $P_\theta(x) \approx P_{data}(x)$.
Objective: Minimize KL Divergence, which is equivalent to Maximizing Log-Likelihood (MLE).

$$\min_\theta D_{KL}(P_{data}||P_\theta) \iff \max_\theta \mathbb{E}_{x\sim data}[\log P_\theta(x)]$$

2. Auto-Regressive Models

Decompose the joint distribution using the chain rule of probability:

$$P_\theta(x) = \prod_{i=1}^{n} P_\theta(x_i|x_1, \ldots, x_{i-1})$$

- **PixelRNN/PixelCNN:** Models images pixel-by-pixel.
- **Masked Convolution:** In PixelCNN, convolution filters are masked (center pixel and future pixels zeroed out) to ensure the prediction for pixel x_i depends only on $x_{<i}$, preserving the autoregressive property.

3. Latent Variable Models (VAE)

Concept:
Assume data x is generated from a latent variable z.

- $z \sim P(z)$ (Prior, usually $\mathcal{N}(0, I)$)
- $x \sim P_\theta(x|z)$ (Likelihood / Decoder)

The Problem:
To train via MLE, we need the marginal $P_\theta(x) = \int P_\theta(x|z)P(z)dz$. This integral is intractable. The posterior $P(z|x)$ is also intractable.

Solution: Variational Auto-Encoder (VAE):
Introduce an approximate posterior (Encoder) $q_\phi(z|x)$ to approximate $P(z|x)$.

Derivation of ELBO (Evidence Lower Bound):
We wish to maximize $\log P(x)$.

$$\log P(x) = \log \int P(x|z)P(z)dz$$

Through Jensen's Inequality or direct derivation:

$$\log P_\theta(x) = \mathcal{L}_{ELBO}(\theta, \phi; x) + D_{KL}(q_\phi(z|x)||P(z|x))$$

Since $D_{KL} \geq 0$:

$$\log P_\theta(x) \geq \mathbb{E}_{q_\phi(z|x)}[\log P_\theta(x|z)] - D_{KL}(q_\phi(z|x)||P(z))$$

Loss Function Components:

1. **Reconstruction Loss:** $\mathbb{E}_{q_\phi}[\log P_\theta(x|z)]$ (Maximize likelihood of data given latent).
2. **Regularization:** $D_{KL}(q_\phi(z|x)||P(z))$ (Force the latent distribution to be close to the standard normal prior).

Reparameterization Trick:
To compute gradients through the sampling process $z \sim q_\phi(z|x) = \mathcal{N}(\mu, \sigma^2 I)$, we rewrite z:

$$z = \mu_\phi(x) + \sigma_\phi(x) \odot \epsilon, \quad \text{where } \epsilon \sim \mathcal{N}(0, I)$$

This moves the stochasticity to ϵ, allowing backpropagation through μ and σ.

Proof: Analytical KL Divergence for Gaussians (Trace Trick):
We need to calculate $D_{KL}(N(\mu_1, \Sigma_1)||N(\mu_2, \Sigma_2))$.
Using the definition $D_{KL}(p||q) = \mathbb{E}_p[\log p - \log q]$:

The term $\mathbb{E}_p[(x - \mu_2)^T \Sigma_2^{-1} (x - \mu_2)]$ is crucial.
Using the trace trick $x^T A x = \text{tr}(x^T A x) = \text{tr}(A x x^T)$:

$$\mathbb{E}[(x - \mu_2)^T \Sigma_2^{-1} (x - \mu_2)] = \text{tr}\left(\Sigma_2^{-1} \mathbb{E}[(x - \mu_2)(x - \mu_2)^T]\right)$$

Expanding $(x - \mu_2) = (x - \mu_1) + (\mu_1 - \mu_2)$, and noting that cross terms vanish under expectation $\mathbb{E}_{x\sim p}$:

$$\mathbb{E}[(x - \mu_2)(x - \mu_2)^T] = \Sigma_1 + (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T$$

Substituting back, the KL divergence becomes:

$$D_{KL} = \frac{1}{2}\left(\text{tr}(\Sigma_2^{-1}\Sigma_1) + (\mu_2 - \mu_1)^T \Sigma_2^{-1} (\mu_2 - \mu_1) - k + \ln \frac{|\Sigma_2|}{|\Sigma_1|}\right)$$

Where k is the dimensionality of the distribution. This analytic solution allows efficient computation of the regularization loss.