Deep Learning Fundamentals: Synthesis

1. Activation Functions

Activation functions are crucial components added between layers of a neural network to introduce **non-linearity**. Without them, a neural network, regardless of its depth, would merely possess the representational capacity of a linear classifier (since a composition of linear operations is still a linear operation).

Common Activation Functions

1. ReLU (Rectified Linear Unit) Currently the default recommendation for modern neural networks.

$$g(z) = \max\{0, z\}$$

- **Pros:** The derivative is computationally "free" (either 0 or 1), accelerating training. It avoids the vanishing gradient problem for positive inputs.
- **Cons:** The "Dying ReLU" problem. If weights adjust such that the input to a neuron is always negative, it outputs 0 perpetually. The gradient becomes 0, and learning stops for that neuron.

2. Leaky ReLU Designed to solve the "Dying ReLU" problem by allowing a small, non-zero gradient when the unit is not active.

$$g(z) = \max\{\alpha z, z\}$$

- Here, α is a small constant (e.g., 0.01).

3. Sigmoid Squishes numbers to range $[0, 1]$.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- **Cons:** Saturates at the tails (large positive or negative values), killing gradients and stalling learning. Expensive exponential calculation.

4. Tanh (Hyperbolic Tangent) Squishes numbers to range $[-1, 1]$.

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

- **Cons:** Like Sigmoid, it suffers from saturation regions that kill gradients.

2. Output Units

The choice of the final layer (output unit) depends directly on the task:

1. **Regression (Single Integer/Continuous Value):**
   - **Architecture:** No output activation function is used.
   - **Mechanism:** Perform a linear transformation on the final feature vector and accept the raw result.
2. **Binary Classification (Probability of One Label):**
   - **Architecture:** Sigmoid activation.
   - **Mechanism:** The linear output is "squished" by the sigmoid function to provide a valid probability between 0 and 1.
3. **Multi-class Classification (Probabilities for N Labels):**
   - **Architecture:** Softmax.
   - **Mechanism:** First, a linear layer outputs unnormalized probabilities (logits). Then, the Softmax operation normalizes them so they sum to 1.

$$\text{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

3. Loss Functions

The loss function quantifies how "wrong" the model is. Modern Deep Learning heavily relies on **Cross-Entropy Loss**, often combined with regularization terms to prevent overfitting (since loss theoretically can approach 0 but never reach it).

Cross Entropy

The concept is derived from Information Theory and "surprise." We want to minimize the surprise of seeing the true label given our predictions.

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\text{data}}} \log p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta})$$

- $\mathbb{E}$ represents the expected value (average) over the dataset.
- We negate the expression because we want to **minimize** cost, and the log of a probability (0 to 1) is always negative.
- **Mini-batching:** In practice, calculating the expected value over the entire dataset is computationally infeasible. Gradients are calculated with respect to θ on small **mini-batches** of data.

4. Gradient-Based Learning & Optimization

Convexity vs. Non-Convexity

- **Linear Models:** Typically have **convex** loss functions. Deterministic algorithms guarantee reaching the global minimum from any starting point.
- **Neural Networks:** Formed by stacking linear layers with non-linearities, resulting in **non-convex** loss functions.
  - **Implication:** There is no guarantee of reaching a global minimum. We calculate gradients and travel in that direction, but the process is fundamentally stochastic.
  - **Initialization:** Highly susceptible to initial parameters; weights should be initialized as small random values to break symmetry.

Optimization Algorithms

Since standard Stochastic Gradient Descent (SGD) can be slow or get stuck in local minima/saddle points, advanced optimizers are used to smooth the learning curve.

1. Momentum

- **Concept:** Instead of updating weights based solely on the current gradient, we take a moving average of past gradients (velocity v).
- **Analogy:** Think of a ball rolling down a hill. It gains momentum, allowing it to plow through small bumps (local minima) and continue in the general direction of the downhill slope, damping oscillations.
- **Equation:**

$$v_t = \beta v_{t-1} + (1 - \beta)\nabla_\theta J(\theta)$$

$$\theta_{t+1} = \theta_t - \alpha v_t$$

(Where β is the momentum term, usually 0.9)

2. RMSProp (Root Mean Square Propagation)

- **Concept:** Adapts the learning rate for each parameter individually. It divides the gradient by a running average of its recent magnitude (squared gradients).
- **Benefit:** Helpful when the error surface has different curvatures in different directions (e.g., steep in one dimension, flat in another).
- **Equation:**

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1 - \beta)(g_t)^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

(Where η is learning rate and ϵ prevents division by zero)

3. Adam (Adaptive Moment Estimation)

- **Concept:** The current standard default for most DL tasks. It combines the best of both worlds:
  - It uses **Momentum** (first moment m_t).
  - It uses **RMSProp** ideas (second moment v_t).
- **Equation:** *First Moment (Momentum):* $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$ *Second Moment (RMSProp):* $v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$ *Bias Correction:* $\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$ *Update:*

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

- **Result:** Fast convergence and generally robust performance across various architectures.