Neural Networks Class 3

Optimization & Training Dynamics

Learning Objectives:

- Master gradient descent variants and their mathematical properties
- Understand training challenges: vanishing/exploding gradients
- Learn regularization techniques and practical considerations

Gradient Descent Family - Batch Gradient Descent

Mathematical Formulation:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{1}{m} \sum_{i=1}^{m} \nabla_{\mathbf{W}} L(\mathbf{W}, \mathbf{x}_i, y_i)$$

Complete Dataset Update:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}} J(\mathbf{W})$$

Where: $J(\mathbf{W}) = \frac{1}{m} \sum_{i=1}^{m} L(\mathbf{W}, \mathbf{x}_i, y_i)$

Properties:

- **Convergence:** Guaranteed for convex functions
- **Computation:** Expensive for large datasets
- **Memory:** Requires full dataset in memory

Gradient Descent Family - Stochastic GD (SGD)

Mathematical Formulation:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}} L(\mathbf{W}, \mathbf{x}_i, y_i)$$

Key Differences from Batch GD:

- Updates after **each example**
- **Noisy gradients** - high variance
- **Faster iterations** but more steps needed

Convergence Properties:

- **Non-convex:** Can escape local minima due to noise
- **Learning rate decay:** $\eta_t = \frac{\eta_0}{1+\alpha t}$ often needed
- **Oscillatory behavior** around optimum

Gradient Descent Family - Mini-batch GD

Mathematical Formulation:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{1}{B} \sum_{i \in \mathcal{B}} \nabla_{\mathbf{W}} L(\mathbf{W}, \mathbf{x}_i, y_i)$$

Where $\mathcal{B}$ is a mini-batch of size B

Batch Size Trade-offs:

- **Small batches** ($B = 32$): Fast, noisy, good generalization
- **Large batches** ($B = 512$): Stable, parallel, may overfit
- **Sweet spot:** Usually $B \in [32, 256]$

Practical Advantages:

- **Vectorization:** Efficient GPU computation

Momentum Methods - Classical Momentum

Mathematical Formulation:

$$\mathbf{v}_t = \gamma \mathbf{v}_{t-1} + \eta \nabla_{\mathbf{W}} L(\mathbf{W})$$
$$\mathbf{W}_t = \mathbf{W}_{t-1} - \mathbf{v}_t$$

Physical Analogy:

- $\mathbf{v}_t$: Velocity vector
- γ: Friction coefficient (typically 0.9)
- $\eta \nabla L$: Force from gradient

Benefits:

- **Accelerates** in consistent gradient directions
- **Dampens oscillations** in high-curvature directions, **Escapes** small local minima

Momentum Methods - Nesterov Accelerated Gradient

Mathematical Formulation:

$$\mathbf{v}_t = \gamma \mathbf{v}_{t-1} + \eta \nabla_{\mathbf{W}} L(\mathbf{W}_{t-1} - \gamma \mathbf{v}_{t-1})$$
$$\mathbf{W}_t = \mathbf{W}_{t-1} - \mathbf{v}_t$$

Key Insight: "Look ahead" before computing gradient

- Evaluate gradient at $\mathbf{W}_{t-1} - \gamma \mathbf{v}_{t-1}$
- More responsive to changes in gradient direction

Convergence Rate:

- **Convex functions:** $O(1/t^2)$ vs $O(1/t)$ for SGD
- **Better practical performance** in many cases

Adaptive Learning Rates - AdaGrad

Mathematical Formulation:

$$\mathbf{G}_t = \mathbf{G}_{t-1} + \nabla_{\mathbf{W}} L(\mathbf{W})^2$$

$$\mathbf{W}_t = \mathbf{W}_{t-1} - \frac{\eta}{\sqrt{\mathbf{G}_t + \epsilon}} \nabla_{\mathbf{W}} L(\mathbf{W})$$

Key Features:

- **Per-parameter** learning rates
- **Larger updates** for infrequent features
- **Smaller updates** for frequent features

Problem: Learning rate decay too aggressive

- $\mathbf{G}_t$ monotonically increases, Eventually stops learning

Adaptive Learning Rates - Adam Optimizer

Mathematical Formulation:

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1)\nabla_{\mathbf{W}} L(\mathbf{W})$$

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2)(\nabla_{\mathbf{W}} L(\mathbf{W}))^2$$

Bias Correction:

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1 - \beta_1^t}, \quad \hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1 - \beta_2^t}$$

Parameter Update:

$$\mathbf{W}_t = \mathbf{W}_{t-1} - \frac{\eta}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon} \hat{\mathbf{m}}_t$$

Default Values: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$

Adam Optimizer Intuition - Combining Best of Both

Momentum Component ($\mathbf{m}_t$):

- Exponential moving average of gradients
- Provides direction and acceleration
- Similar to classical momentum

Adaptive Component ($\mathbf{v}_t$):

- Exponential moving average of squared gradients
- Provides per-parameter scaling
- Similar to AdaGrad but with decay

Bias Correction:

- Corrects initialization bias ($\mathbf{m}_0 = \mathbf{v}_0 = 0$), Important in early training steps

Training Challenges - Vanishing Gradient Problem

Mathematical Analysis:

Consider L-layer network with weights $\mathbf{W}^{(l)}$ and activations $\sigma(\cdot)$

Gradient at layer l:

$$\frac{\partial L}{\partial \mathbf{W}^{(l)}} = \frac{\partial L}{\partial \mathbf{A}^{(L)}} \prod_{k=l+1}^{L} \mathbf{W}^{(k)} \sigma'(\mathbf{Z}^{(k-1)})$$

For Sigmoid: $\sigma'(z) \leq 0.25$

Product of many small terms: $\prod_{k=l+1}^{L} |\mathbf{W}^{(k)}| \cdot 0.25 \to 0$

Consequence: Early layers learn very slowly

Training Challenges - Exploding Gradient Problem

Mathematical Condition:

If $|\mathbf{W}^{(k)}| \cdot |\sigma'(\mathbf{Z}^{(k)})| > 1$ for most layers:

$$\prod_{k=l+1}^{L} |\mathbf{W}^{(k)}| \cdot |\sigma'(\mathbf{Z}^{(k)})| \to \infty$$

Consequences:

- **Unstable training:** Loss oscillates wildly
- **Numerical overflow:** Gradients become NaN
- **Poor convergence:** Cannot find good solution

Solution Preview: Gradient clipping, better initialization, skip connections

Overfitting vs Underfitting - Bias-Variance Trade-off

Underfitting (High Bias):

- Model too simple for data complexity
- **Training error:** High
- **Validation error:** High
- **Gap:** Small

Overfitting (High Variance):

- Model too complex, memorizes training data
- **Training error:** Low
- **Validation error:** High
- **Gap:** Large

Regularization Techniques

L1 and L2 Regularization

L2 Regularization (Ridge): - Shrinks weights, smooth solutions

$$J_{\text{reg}}(\mathbf{W}) = J(\mathbf{W}) + \frac{\lambda}{2} \sum_{l=1}^{L} ||\mathbf{W}^{(l)}||_F^2$$

Gradient Modification:

$$\nabla_{\mathbf{W}} J_{\text{reg}} = \nabla_{\mathbf{W}} J + \lambda \mathbf{W}$$

L1 Regularization (Lasso): - Sparse weights, feature selection

$$J_{\text{reg}}(\mathbf{W}) = J(\mathbf{W}) + \lambda \sum_{l=1}^{L} ||\mathbf{W}^{(l)}||_1$$

Regularization Techniques - Dropout

Training Phase:

$$\mathbf{r}^{(l)} \sim \text{Bernoulli}(p)$$

$$\tilde{\mathbf{A}}^{(l)} = \mathbf{r}^{(l)} \odot \mathbf{A}^{(l)}$$

$$\mathbf{A}^{(l+1)} = \sigma(\mathbf{W}^{(l+1)}\tilde{\mathbf{A}}^{(l)} + \mathbf{b}^{(l+1)})$$

Testing Phase:

$$\mathbf{A}^{(l+1)} = \sigma(\mathbf{W}^{(l+1)}(p \cdot \mathbf{A}^{(l)}) + \mathbf{b}^{(l+1)})$$

Intuition:

- Randomly "drop" neurons during training
- Forces network to not rely on specific neurons
- Creates ensemble effect

Weight Initialization Strategies - Xavier/Glorot

Problem: Poor initialization leads to vanishing/exploding gradients

Xavier Initialization:

For layer with n_{in} inputs and n_{out} outputs:

$$\mathbf{W} \sim \mathcal{N}\left(0, \frac{2}{n_{\text{in}} + n_{\text{out}}}\right)$$

Or Uniform:

$$\mathbf{W} \sim \mathcal{U}\left(-\sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}}}}, \sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}}}}\right)$$

Goal: Maintain activation variance across layers

Weight Initialization Strategies - He Initialization

For ReLU Networks:

$$\mathbf{W} \sim \mathcal{N}\left(0, \frac{2}{n_{\text{in}}}\right)$$

Mathematical Justification:

- ReLU kills half the neurons ($\mathbb{E}[\text{ReLU}(x)] = \frac{1}{2}\mathbb{E}[x]$ for $x \sim \mathcal{N}(0, \sigma^2)$)
- Need larger variance to compensate
- Maintains signal propagation through deep networks

Rule of Thumb:

- **Sigmoid/Tanh:** Xavier initialization
- **ReLU/Leaky ReLU:** He initialization

Learning Rate Scheduling - Adaptive Learning Rate

Step Decay:

$$\eta_t = \eta_0 \cdot \gamma^{\lfloor t/s \rfloor}$$

Exponential Decay:

$$\eta_t = \eta_0 \cdot e^{-\lambda t}$$

Cosine Annealing:

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min})(1 + \cos(\frac{t\pi}{T}))$$

Learning Rate Warmup:

$$\eta_t = \eta_{\text{base}} \cdot \frac{t}{t_{\text{warmup}}} \quad \text{for } t < t_{\text{warmup}}$$

Batch Normalization - Mathematical Formulation

For mini-batch $\mathcal{B} = \{x_1, x_2, \ldots, x_m\}$:

Statistics:

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i, \sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$

Normalization:

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

Scale and Shift:

$$y_i = \gamma \hat{x}_i + \beta$$

Where γ and β are learnable parameters

Batch Normalization Benefits - Why It Works

Internal Covariate Shift Reduction:

- Stabilizes distribution of layer inputs
- Reduces dependence on initialization, Allows higher learning rates

Regularization Effect:

- Noise from mini-batch statistics
- Reduces overfitting (similar to dropout)

Gradient Flow:

- Prevents vanishing gradients, Enables training of very deep networks

Mathematical Insight:

BN ensures $\mathbb{E}[\hat{x}] = 0$ and $\mathrm{Var}[\hat{x}] = 1$ for each layer

Practical Training Tips - Monitoring and Debugging

Loss Curves:

- **Training loss decreasing:** Model is learning
- **Validation loss increasing:** Overfitting
- **Both losses plateauing:** Need more capacity or different architecture

Gradient Monitoring:

- **Gradient norm:** Should not be too small (vanishing) or too large (exploding)
- **Parameter updates:** Should be ~1% of parameter magnitude

Activation Statistics:

- **Mean activations:** Should not be all 0 or saturated
- **Activation variance:** Should remain stable across layers

Summary: Class 3 Key Concepts

Optimization Algorithms:

- **SGD variants:** Batch, mini-batch, momentum, Nesterov
- **Adaptive methods:** AdaGrad, Adam with mathematical formulations

Training Challenges:

- **Vanishing gradients:** $\prod \mathbf{W}\sigma' \to 0$
- **Exploding gradients:** $\prod \mathbf{W}\sigma' \to \infty$
- **Overfitting:** High variance, regularization needed

Practical Solutions:

- **Regularization:** L1/L2, dropout mathematical formulations
- **Initialization:** Xavier for sigmoid/tanh, He for ReLU