Linear Classification

Score function: $s = f(x; W) = Wx + b$

Softmax classifier: $P(y_i|x_i) = \frac{e^{s_{y_i}}}{\sum_j e^{s_j}}$

Cross-entropy loss: $L_i = -\log(P(y_i|x_i))$

- Min loss: 0 (when $P(y_i|x_i) = 1$)
- Max loss: ∞ (when $P(y_i|x_i) \approx 0$)
- Random initialization: $\log(C)$ for C classes

SVM loss: $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$

- Δ is margin parameter (typically $\Delta = 1$)
- Wants correct class score higher than incorrect class scores by at least Δ
- Geometric interpretation: Linear hyperplanes separating classes

Key concepts:

- Any FC network can be expressed as a CNN (with 1×1 filters) and vice versa
- Loss gradients flow from softmax loss to weight matrix proportional to input
- Linear classifiers can't solve XOR problems (need nonlinearities)

Regularization

Full loss: $L = \frac{1}{N}\sum_{i=1}^{N} L_i + \lambda R(W)$

Types:

- L2: $R(W) = \sum_k \sum_l W_{k,l}^2$ (prefers diffuse weights)
- L1: $R(W) = \sum_k \sum_l |W_{k,l}|$ (promotes sparsity)
- Elastic Net: $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$
- Dropout: Randomly zero outputs during training (scale by p at test)
- Batch Norm: Normalize activations across batch dimension

Key concepts:

- Early stopping: Regularization effect by halting training when validation error increases
- Dropout forces redundant representations, acts like ensemble averaging
- Regularizing bias terms is generally avoided (mainly regularize weights)
- Data augmentation: Adding transformed training examples

Optimization Algorithms

SGD: $w_{t+1} = w_t - \alpha \nabla L(w_t)$

SGD+Momentum:

$$v_{t+1} = \rho v_t + \nabla L(w_t)$$
$$w_{t+1} = w_t - \alpha v_{t+1}$$

RMSProp:

$$\text{grad\_squared} = \beta \cdot \text{grad\_squared} + (1-\beta) \cdot g^2$$
$$w_{t+1} = w_t - \frac{\alpha \cdot g}{\sqrt{\text{grad\_squared}} + \epsilon}$$

Adam:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \quad \text{(momentum)}$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \quad \text{(RMSProp)}$$
$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \quad \text{(bias correction)}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \quad \text{(bias correction)}$$
$$w_{t+1} = w_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Learning Rate Decay:

- Step decay: $\alpha_t = \alpha_0 \cdot \gamma^{\lfloor t/\text{epoch} \rfloor}$
- Cosine decay: $\alpha_t = \alpha_{\min} + \frac{1}{2}(\alpha_0 - \alpha_{\min})(1 + \cos(\frac{t}{T}\pi))$

Key concepts:

- SGD issues: poor conditioning, getting stuck in local minima/saddle points, noisy
- Momentum helps overcome oscillations and escape poor local minima
- RMSProp addresses AdaGrad's decaying learning rate issue
- Adam combines momentum (first moment) and RMSProp (second moment)
- AdamW separates weight decay from gradient update for better regularization

Neural Networks

Multi-layer Perceptron: $f = W_2 \cdot \max(0, W_1 \cdot x + b_1) + b_2$

Dimensions: $x \in \mathbb{R}^D$, $W_1 \in \mathbb{R}^{H \times D}$, $b_1 \in \mathbb{R}^H$, $W_2 \in \mathbb{R}^{C \times H}$, $b_2 \in \mathbb{R}^C$

Activation Functions:

- ReLU: $f(x) = \max(0, x)$ (most common)
- Leaky ReLU: $f(x) = \max(\alpha x, x)$, α small (e.g., 0.01)
- ELU: $f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}$
- GELU: $f(x) = x \cdot \Phi(x)$ (used in transformers)
- Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$ (vanishing gradient)
- Tanh: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ (zero-centered)

Key concepts:

- ReLU has zero gradient when inputs are negative (can cause "dying ReLU")
- Leaky ReLU can't cause numerically zero gradients (always has non-zero slope)
- Tanh has zero-centered outputs (advantage over sigmoid)
- Without nonlinear activations, deep networks reduce to linear models
- Deeper networks can represent more complex functions with fewer parameters

Backpropagation

Chain rule: $\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$

Gradient flow: Local gradients × upstream gradients = downstream gradients

Key steps:

1. Forward pass: compute outputs and cache intermediates
2. Backward pass: compute gradients using chain rule
3. Update parameters using gradients

Vector derivatives:

- Same shape as original variables
- Apply chain rule using matrix calculus

Special derivatives:

- Sigmoid: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$
- Tanh: $\frac{d\tanh(x)}{dx} = 1 - \tanh^2(x)$
- ReLU: $\frac{d\text{ReLU}(x)}{dx} = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{cases}$

Key Concepts:

- Computational graphs organize calculations as directed acyclic graphs
- Backprop can handle arbitrary complex computational graphs

Convolutional Neural Networks

Conv Layer Summary:

- **Input**: $C_{in} \times H \times W$
- **Hyperparameters**:
  - Kernel size: $K_H \times K_W$
  - Number filters: C_{out}
  - Padding: $P = (K-1)/2$ ("same" padding)
  - Stride: S
- **Weight matrix**: $C_{out} \times C_{in} \times K_H \times K_W$
- **Bias vector**: C_{out}
- **Output size**: $C_{out} \times H' \times W'$ where:

$$H', W' = \frac{H - K_{\{H,W\}} + 2P}{S} + 1$$

Advantages:

- Parameter sharing: Same filter applied across image
- Sparse connectivity: Each output depends on small local region
- Translation equivariance: Shifting input shifts output

Pooling layers:

- Max pooling: Take maximum value in window
- Average pooling: Average values in window
- Reduces spatial dimensions, increases receptive field

Receptive field: Region of input that affects output

- For stacked K×K filters, RF grows by (K-1) per layer
- With L layers, RF is $1 + L * (K-1)$

Key concepts:

- Multiple 3×3 filters better than single large filter: fewer parameters, more nonlinearities
- 1×1 convolutions: used for dimension reduction across channels
- Dilated convolutions: expand receptive field without increasing parameters
- Depth-wise separable convs: separate spatial and cross-channel operations

Normalization Techniques

Batch Normalization:

$$\mu_c = \frac{1}{N}\sum_{n=1}^{N} x_{n,c,h,w}$$
$$\sigma_c = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(x_{n,c,h,w} - \mu_c)^2}$$
$$\hat{x}_{n,c,h,w} = \frac{x_{n,c,h,w} - \mu_c}{\sigma_c}$$
$$y_{n,c,h,w} = \gamma_c \cdot \hat{x}_{n,c,h,w} + \beta_c$$

Normalizes across batch dimension for each channel

Layer Normalization:

$$\mu_n = \frac{1}{C}\sum_{c=1}^{C} x_{n,c,h,w}$$
$$\sigma_n = \sqrt{\frac{1}{C}\sum_{c=1}^{C}(x_{n,c,h,w} - \mu_n)^2}$$
$$\hat{x}_{n,c,h,w} = \frac{x_{n,c,h,w} - \mu_n}{\sigma_n}$$
$$y_{n,c,h,w} = \gamma_c \cdot \hat{x}_{n,c,h,w} + \beta_c$$

Normalizes across channel dimension for each sample

Instance Normalization:

$$\mu_{n,c} = \frac{1}{H \times W}\sum_{h,w} x_{n,c,h,w}$$
$$\sigma_{n,c} = \sqrt{\frac{1}{H \times W}\sum_{h,w}(x_{n,c,h,w} - \mu_{n,c})^2}$$
$$\hat{x}_{n,c,h,w} = \frac{x_{n,c,h,w} - \mu_{n,c}}{\sigma_{n,c}}$$
$$y_{n,c,h,w} = \gamma_c \cdot \hat{x}_{n,c,h,w} + \beta_c$$

Normalizes each channel independently for each sample

Key concepts:

- BatchNorm: Used in CNNs, must track running stats for inference
- LayerNorm: Used in transformers, no dependence on batch statistics
- Instance Norm: Used in style transfer, normalizes each feature map
- Group Norm: Compromise between Layer and Instance Norm
- Normalization adds regularization effect due to noise in batch stats

CNN Architectures

VGG:

- Multiple 3×3 convs followed by max-pooling
- Multiple 3×3 filters have same receptive field as larger filter with fewer parameters
- Uniform design: doubles channels after each pooling

ResNet:

- Skip connections: output $= F(x) + x$
- Allow deeper networks by learning residual mapping
- Solves vanishing gradient problem in deep nets
- Typical block: Conv → BN → ReLU → Conv → BN → Add → ReLU

Bottleneck layer:

- 1×1 conv to reduce channels, 3×3 conv, 1×1 conv to expand
- Reduces computation by decreasing channels in 3×3 conv

Key concepts:

- Network design trade-offs: depth vs. width vs. resolution
- Skip connections help gradient flow and enable deeper networks
- Deeper networks generally need more regularization
- ResNet skip connections: $O_l = I_l + F(I_l)$ not $O_l = I_l * F(I_l)$

Weight Initialization

Xavier/Glorot: $W \sim \mathcal{N}(0, \sqrt{\frac{2}{n_{in}+n_{out}}})$

- For tanh/sigmoid activations
- Maintains variance across linear layers

Kaiming initialization: $W \sim \mathcal{N}(0, \sqrt{\frac{2}{D_{in}}})$ for ReLU

- For ReLU activations (accounts for half being zeroed)
- For CNN: $D_{in} = \text{kernel\_size}^2 \times \text{input\_channels}$

Key concepts:

- Poor initialization can cause vanishing/exploding gradients
- Initialization in deep nets is crucial for trainability
- Even with good normalization, bad initialization slows training
- Initialization should match the activation function

Training Techniques

Data Preprocessing:

- Zero-centering: $\tilde{x} = x - \mu$
- Normalization: $\tilde{x} = \frac{x-\mu}{\sigma}$

Data Augmentation:

- Horizontal flips, random crops, color jitter
- Rotations, scaling, shearing (with limits)
- CutMix, Mixup: interpolate between images

Transfer Learning:

- Small dataset: Freeze pretrained model, retrain final layers
- Medium dataset: Freeze early layers, fine-tune later layers
- Large dataset: Initialize with pretrained weights, fine-tune all layers

Diagnostics:

- Underfitting: Low train/val accuracy, small gap
- Overfitting: High train accuracy, low val accuracy, large gap
- Not training enough: Low train/val accuracy with gap

Hyperparameter selection:

- Random search usually better than grid search
- Check initial loss, overfit small sample first
- Find LR that makes loss decrease within 100 iterations

Loss Functions

Cross-entropy: $L = -\sum_i y_i \log(\hat{y}_i)$

- For classification problems
- Measures dissimilarity between two probability distributions

KL Divergence: $D_{KL}(p||q) = \sum_i p_i \log \frac{p_i}{q_i} = \sum_i p_i(\log p_i - \log q_i)$

- Rewrite as subtraction for numerical stability
- Not symmetric: $D_{KL}(p||q) \neq D_{KL}(q||p)$

Smooth L1/Huber Loss:

$$L_\delta(x,y) = \begin{cases} \frac{1}{2}(x-y)^2 & \text{if } |x-y| < \delta \\ \delta(|x-y| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$$

- Combines MSE (near zero) and L1 (for outliers)
- Differentiable everywhere, robust to outliers

Triplet margin loss: $L(a,p,n) = \max\{d(a,p) - d(a,n) + \text{margin}, 0\}$

- Used in contrastive learning
- Pushes anchor (a) closer to positive (p) than negative (n)
- Margin controls separation between positive and negative pairs

Recurrent Neural Networks

Vanilla RNN:

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$
$$y_t = W_{hy}h_t + b_y$$

LSTM:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \odot \tanh(C_t)$$

RNN Applications:

- Language modeling: Predict next token in sequence
- Image captioning: CNN feature extractor + RNN decoder
- Sequence-to-sequence: Translation, summarization

Training RNNs:

- Backpropagation through time (BPTT)
- Truncated BPTT for long sequences
- Gradient clipping to prevent explosion

Key concepts:

- RNNs can process variable-length sequences
- Vanishing/exploding gradients limit learning long-term dependencies
- LSTM addresses vanishing gradients via cell state pathway
- RNNs sequential processing limits parallelization

Attention Mechanism

Inputs:

- Input sequence: $X \in \mathbb{R}^{N \times D_{in}}$
- Query matrix: $W_Q \in \mathbb{R}^{D_{in} \times D_{out}}$
- Key matrix: $W_K \in \mathbb{R}^{D_{in} \times D_{out}}$
- Value matrix: $W_V \in \mathbb{R}^{D_{in} \times D_{out}}$

Computation:

$$\text{Queries: } Q = XW_Q \in \mathbb{R}^{N \times D_{out}}$$
$$\text{Keys: } K = XW_K \in \mathbb{R}^{N \times D_{out}}$$
$$\text{Values: } V = XW_V \in \mathbb{R}^{N \times D_{out}}$$
$$\text{Similarities: } E = \frac{QK^T}{\sqrt{D_{out}}} \in \mathbb{R}^{N \times N}$$
$$\text{Weights: } A = \text{softmax}(E, \text{dim}=1) \in \mathbb{R}^{N \times N}$$
$$\text{Output: } Y = AV \in \mathbb{R}^{N \times D_{out}}$$

Multi-Head Attention:

Inputs:

- Input vectors: $X \in \mathbb{R}^{N \times D}$
- For each head $h \in \{1, ..., H\}$:
  - Key matrix: $W_K^h \in \mathbb{R}^{D \times D_H}$
  - Value matrix: $W_V^h \in \mathbb{R}^{D \times D_H}$
  - Query matrix: $W_Q^h \in \mathbb{R}^{D \times D_H}$
- Output matrix: $W_O \in \mathbb{R}^{HD_H \times D}$

Computation for each head h:

$$\text{Queries: } Q^h = XW_Q^h \in \mathbb{R}^{N \times D_H}$$
$$\text{Keys: } K^h = XW_K^h \in \mathbb{R}^{N \times D_H}$$
$$\text{Values: } V^h = XW_V^h \in \mathbb{R}^{N \times D_H}$$
$$\text{Similarities: } E^h = \frac{Q^h(K^h)^T}{\sqrt{D_H}} \in \mathbb{R}^{N \times N}$$
$$\text{Attention weights: } A^h = \text{softmax}(E^h, \text{dim}=1) \in \mathbb{R}^{N \times N}$$
$$\text{Head output: } Y^h = A^hV^h \in \mathbb{R}^{N \times D_H}$$

Combining heads:

$$\text{Concatenated output: } Y = [Y^1; Y^2; ...; Y^H] \in \mathbb{R}^{N \times HD_H}$$
$$\text{Final output: } O = YW_O \in \mathbb{R}^{N \times D}$$

Types of Attention:

- Self-attention: Q, K, V from same sequence
- Cross-attention: Q from one sequence, K, V from another
- Masked attention: Future positions masked (decoder)

Key concepts:

- Time complexity: $O(n^2 d)$ for sequence length n and dimension d
- Memory complexity: $O(n^2)$ for attention weights
- Attention weights computed from Q and K (not V)
- Scaling factor $\sqrt{d_k}$ prevents vanishing gradients with large dimensions
- Self-attention is permutation equivariant without positional encoding

Transformers

Transformer block:

1. Layer normalization
2. Multi-head self-attention
3. Residual connection
4. Layer normalization
5. Feed-forward network (MLP)
6. Residual connection

Parameters in transformer block:

- Self-attention: $4d^2$ (Q, K, V projections + output)
- Feed-forward: $2df$ (where f is FF dimension, typically $4d$)

Vision Transformer (ViT):

- Split image into patches (16×16)
- Linear projection + position embeddings
- Standard transformer encoder architecture

CLS token or pooling for classification

Key concepts:

- Transformers use LayerNorm, NOT BatchNorm
- Pre-norm vs. post-norm: affects training stability
- Transformers parallelize better than RNNs for sequences
- Positional encodings enable model to learn position information

Semantic Segmentation

Task: Classify each pixel in an image

Architectures:

- Fully Convolutional Networks (FCN)
- U-Net: Encoder-decoder with skip connections
- DeepLab: Atrous convolutions for dense predictions

Upsampling techniques:

- Unpooling: Reverse pooling operation
- Transposed convolution: Learnable upsampling
- Bilinear interpolation + 1×1 convs: Smoother results

Key concepts:

- Semantic segmentation: One label per pixel, no instance separation
- Downsampling followed by upsampling preserves context while maintaining resolution
- Skip connections help preserve spatial detail
- Dilated/atrous convolutions expand receptive field without losing resolution

Object Detection

Key architectures:

- R-CNN family: Region proposals + classification
- YOLO: Single-pass detection with grid cells
- DETR: Transformers with object queries

Region Proposal Network:

- Generate candidate boxes
- Binary classification (object vs. background)
- Bounding box regression

Evaluation metrics:

- IoU (Intersection over Union): $\frac{\text{area of intersection}}{\text{area of union}}$
- Precision: $\frac{\text{TP}}{\text{TP + FP}}$, Recall: $\frac{\text{TP}}{\text{TP + FN}}$
- AP: Area under PR curve for each class
- mAP: Mean AP across all classes

Key concepts:

- Two-stage detectors (R-CNN family): region proposal + classification
- One-stage detectors (YOLO, SSD): directly predict boxes from grid cells
- NMS (Non-Maximum Suppression): Remove duplicate detections
- Anchor boxes: Pre-defined box shapes to match during training

Instance Segmentation

Mask R-CNN:

- Extends Faster R-CNN with mask branch
- RoIAlign for accurate feature extraction
- Parallel heads for classification, box regression, mask prediction

Key concepts:

- RoIAlign: Keeps spatial information intact (avoids quantization)
- Instance segmentation separates individual instances of same class
- Panoptic segmentation: Combines semantic and instance segmentation

Video Understanding

Architectures:

- Single-frame CNN + temporal pooling
- Early fusion: Treat time as channels
- 3D CNN: 3D convolutions (C3D, I3D)
- CNN + RNN: CNN features fed to RNN
- Transformer: Space-time attention

3D convolution:

$$\text{Output}: F \times T' \times H' \times W'$$
$$\text{Filter size}: C \times k_t \times k \times k$$

Two-stream networks:

- Spatial stream: RGB frames
- Temporal stream: Optical flow
- Late fusion of predictions

Key concepts:

- 3D CNN receptive fields span space and time dimensions
- Early fusion builds temporal receptive field all at once
- Slow fusion gradually builds temporal receptive field
- 3D CNNs have temporal-shift invariance (early fusion doesn't)

Neural Network Visualization

Saliency maps:

- Compute gradient of class score w.r.t input pixels
- Highlights regions important for classification

Class Activation Mapping (CAM):

$$M_c(x,y) = \sum_k w_k^c \cdot f_k(x,y)$$

Grad-CAM:

- Generalizes CAM to any CNN architecture
- Global-average-pools gradients for importance weights
- Weighted combination of feature maps

Key concepts:

- Visualizations help debug network decisions
- CNN filters often detect edges, textures, patterns, and semantic concepts
- Attention maps in transformers provide built-in visualization

Evaluation Metrics

Classification:

- Accuracy: $\frac{\text{correct predictions}}{\text{total predictions}}$
- Precision: $\frac{\text{TP}}{\text{TP + FP}}$
- Recall: $\frac{\text{TP}}{\text{TP + FN}}$
- F1 Score: $\frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$

Segmentation:

- Pixel accuracy: $\frac{\text{correctly classified pixels}}{\text{total pixels}}$
- Mean IoU: Average IoU across all classes
- Dice coefficient: $\frac{2 \times \text{intersection}}{\text{sum of areas}}$

Point Cloud Processing:

- Translation equivariance: Output shifts when input shifts
- Rotation equivariance: Output rotates when input rotates
- Convs on grid structure not rotation-equivariant
- Continuous point convs use weight functions of relative positions