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 non-linearities)

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)
- Normalization adds regularization effect due to noise in batch stats

Key concepts:
- Early stopping: end training when val error increases
- Dropout forces redundant representations, acts like ensemble that shares parameters
- 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 issues: poor conditioning, getting stuck in local minima/saddle points, noisy

SGD+Momentum:

$$v_{t+1} = \rho v_t + \nabla L(w_t) \qquad \text{(typically } \rho = 0.9 \text{ or } 0.99\text{)}$$

$$w_{t+1} = w_t - \alpha v_{t+1}$$

- Momentum overcomes oscillations and escape poor local minima, continues moving in prev direction

RMSProp:

$$\text{grad\_squared} = \beta \cdot \text{grad\_squared} + (1-\beta) \cdot (\nabla L(w_t))^2$$

$$w_{t+1} = w_t - \frac{\alpha \cdot \nabla L(w_t)}{\sqrt{\text{grad\_squared}} + \epsilon}$$

- RMSProp adds per-parameter learning rate, addresses AdaGrad's decaying learning rate issue
- Progress on steep dir is damped, flat dir is accelerated

Adam:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla L(w_t) \qquad \text{(momentum)}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla L(w_t))^2 \qquad \text{(RMSProp)}$$

$$\hat{m}_t = m_t/(1-\beta_1^t) \qquad \text{(bias correction)}$$

$$\hat{v}_t = v_t/(1-\beta_2^t) \qquad \text{(bias correction)}$$

$$w_{t+1} = w_t - \alpha \hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon)$$

- Adam combines momentum and RMSProp
- AdamW separates weight decay from gradient update for better regularization

Second Order:
- Decrease LR over time (step, cosine, linear, etc.)
- Linear warmup: increase LR from 0 over first few steps, prevent exploding loss

Key concepts:
- Second-order methods: $\theta^* = \theta_0 - \alpha H^{-1}\nabla L(\theta_0)$
- Better updates, $O(N^2)$ mem and $O(N^3)$ time to invert

Neural Networks

MLP: $f = W_2 \max(0, W_1 x + b_1) + b_2$
$x \in \mathbb{R}^D$, $W_1 \in \mathbb{R}^{H\times D}$, $b_1 \in \mathbb{R}^H$, $W_2 \in \mathbb{R}^{C\times H}$, $b_2 \in \mathbb{R}^C$

Activation Functions:
- ReLU: $f(x) = \max(0, x)$
- Leaky ReLU: $f(x) = \max(\alpha x, x)$ with small α
- ELU: $f(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{if } x < 0 \end{cases}$
- GELU: $f(x) = x \cdot \Phi(x)$
- Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$
- Tanh: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

Key concepts:
- ReLU has zero grad with negative inputs (dying ReLU)
- Leaky ReLU (and variants) always has non-zero slope
- Tanh has zero-centered outputs (sigmoid has $\mu = 0.5$)
- Without nonlinear activations, deep networks reduce to linear models
- Deeper networks can represent more complex functions with fewer parameters (more non-linearities)

Backpropagation

Gradient flow: Upstream × Local = Downstream

Vector derivatives:
- $d_x f(x)$ has same shape as x
- Apply chain rule using matrix calculus
- Matmul: $\frac{\partial}{\partial X}(XW) = W^T$ and $\frac{\partial}{\partial W}(XW) = X^T$
- Each element $X_{n,d}$ affects the whole row Y_n
- Backprop: X: [N, D], W: [D, M], Y: [N, M]

$$\frac{\partial L}{\partial X} = \left(\frac{\partial L}{\partial Y}\right) W^T \in [N, D] \qquad \frac{\partial L}{\partial W} = X^T \left(\frac{\partial L}{\partial Y}\right) \in [D, M]$$

Special derivatives:
- Sigmoid: $d_x \sigma(x) = \sigma(x)(1-\sigma(x))$
- Tanh: $d_x \tanh(x) = 1 - \tanh^2(x)$
- ReLU: $d_x \text{ReLU}(x) = \mathbb{1}(x > 0)$
- Max: $d_x \max(x, y) = \mathbb{1}(x > y)$
- Softmax: $\frac{dp_i}{ds_j} = p_i(\mathbb{1}(i=j) - p_j)$ where $p = \frac{e^s}{\sum_k e^{s_k}}$
- Cross-entropy: $d_{s_i}\left(-\sum_j y_j \log(p_j)\right) = p_i - y_i$
- Huber Loss: $d_x L_\delta(x, y) = \begin{cases} x - y & \text{if } |x-y| < \delta \\ \delta \cdot \text{sign}(x-y) & \text{otherwise} \end{cases}$
- L1 Loss: $d_x|x-y| = \text{sign}(x-y)$

Key Backpropagation Concepts:
- Vanishing gradients: gradients become too small in deep networks (esp. with sigmoid/tanh)
- Exploding gradients: gradients become too large (common in RNNs)
- Gradient clipping: Cap gradient magnitude to prevent explosion

Convolutional Neural Networks

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

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

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:
- Given input C×H×W, downsample each 1×H×W plane
- 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 K×K filters, RF grows by $(K-1)$ per layer
- With L layers and $S = 1$, RF is $1 + L * (K-1)$
- In general, $R_0 = 1$ and $R_l = R_{l-1} + (K-1) \times S$

Key concepts:
- Multiple 3×3 filters better than single large filter: fewer parameters, more nonlinearities
- 1×1 conv: same H/W, dim reduction across channels
- Dilated convolutions: expand receptive field without increasing parameters

CNN Architectures

VGG:
- Multiple 3×3 convs followed by max-pooling
- Stack of three 3×3 convs has same receptive field as 7×7 conv, but deeper with less params ($3 \times 9C^2$ vs $49C^2$)
- Uniform design: doubles channels after each pooling

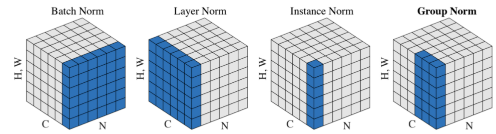
ResNet:
- Skip connections: output = $F(x) + x$
- Allow deeper networks by learning residual mapping
- Solves vanishing gradient problem in deep nets
- Conv → BN → ReLU → Conv → BN → Add → ReLU

Equivariance and Invariance

Definitions:
- **Equivariant:** $f(Tx) = Tf(x)$ (output transforms in the same way as input)
- **Invariant:** $f(Tx) = f(x)$ (output does not change under transformation)

Key Types:
- **Translation equivariant:** Shifting the input causes the output to shift in the same way.
  - *Example:* **Convolution:** If you shift an image, the output activation map shifts the same amount.
  - *Non-example:* Fully connected (MLP) layers are not translation equivariant.
- **Translation invariant:** Shifting the input does not change the output.
  - *Example:* **Global pooling:** $\max(x)$ or $\sum x$ over all positions.
- **Rotation equivariant:** Rotating the input causes the output to rotate in the same way.
  - *Non-example:* Standard CNNs with fixed kernels are not rotation equivariant (rotating the image does not rotate the output activation map).
- **Rotation invariant:** Rotating the input does not change the output.
  - *Example:* $\|x\|_2$ (L2 vector norm)
- **Permutation equivariant (Self-attention):**
  - *Definition:* Reordering (permuting) the input sequence reorders the output in the same way.
  - *Example:* **Self-attention without positional encoding:** If you swap two tokens in the input, the outputs for those tokens are swapped.
  - *Non-example:* Self-attention with positional encoding is not permutation equivariant.

Normalization Techniques



Batch Normalization:

μ_c = mean of feature values across batch for channel c

σ_c = standard deviation across batch for channel c

$y = \gamma_c \cdot (x - \mu_c)/\sigma_c + \beta_c$

- Normalizes across batch dimension for each channel
- Used in CNNs, must track running stats for inference

Layer Normalization:

μ_n = mean across all channels for sample n

σ_n = standard deviation across all channels for sample n

$y = \gamma_c \cdot (x - \mu_n)/\sigma_n + \beta_c$

- Normalizes across channel dimension for each sample
- Used in transformers, no dependence on batch statistics, good for sequence models

Group Normalization:
- Group channels into groups, normalize each group independently for each sample
- Used in CNNs, good for parallelization

Weight Initialization

Kaiming initialization: $W \sim \mathcal{N}(0, \sqrt{\frac{2}{D_{in}}})$ for ReLU
- For ReLU activations (accounts for half being zeroed)
- For CNN: $D_{in} = C_{\text{in}} \times K_H \times K_W$

Key concepts:
- Too small or too large initializations can cause vanishing/exploding gradients
- Initialization in deep nets is crucial for trainability
- Normalization mitigates bad initialization (not solve)
- Initialization should match the activation function

Training Techniques

Data Normalization: subtract per-channel mean, divide by per-channel std, better convergence and generalization

Data Augmentation:
- Increases dataset size/diversity without new data
- Improves robustness to image variations
- Common techniques: flips, crops, color jitter, rotations

Transfer Learning: use pre-trained models
- Small dataset + similar: retrain final layer
- Small dataset + different: another model or more data
- Large dataset + similar: finetune all model layers
- Large dataset + different: either finetune all layers or train from scratch

Diagnostics:
- Underfitting: Low train/val accuracy, small or no gap
- Overfitting: High train, 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 quickly
- Split data into train/val/test; tune on validation set
- K-fold cross-validation useful for small datasets

Loss Functions

Cross-entropy: $L = -\sum_i y_i \log(\hat{y}_i)$
- For classification problems, measures how well predictions match true labels

KL: $D_{KL}(p||q) = \sum_i p_i \log\frac{p_i}{q_i} = \sum_i p_i(\log p_i - \log q_i)$
- $D_{KL}(p||q) = \text{CrossEntropy}(p, q) - H(p)$
- In one-hot classification, KL is same as CE because H(one hot true labels) is zero
- Measures dissimilarity between probability dist.
- 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: L(a,p,n) = $\max\{d(a,p) - d(a,n) + \Delta, 0\}$
- Used in contrastive learning, pushes anchor (a) closer to positive (p) than negative (n)
- Margin controls separation between positive and negative pairs
- Small margin → harder to separate, larger margin → too much separation, difficult to learn

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:

- Solves vanilla RNN's vanishing gradient problem
- Cell state (C_t) maintains long-term memory
- Three gates control information flow:
  - Forget gate: decides what to discard from cell state
  - Input gate: decides what new information to store
  - Output gate: controls what parts of cell state affect output
- Gradient can flow unchanged through cell state
- More complex but better at capturing long sequences

RNN Applications:

- Language modeling: Predict next token in sequence
- Captioning: CNN feature extractor + RNN decoder
- Sequence-to-sequence: encoder-decoder for translation

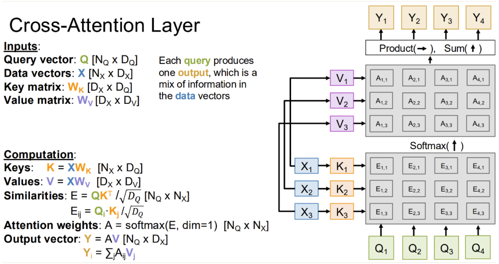
Training RNNs:

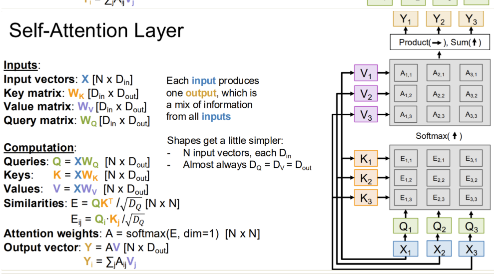
- Backpropagation through time (BPTT)
- Truncated BPTT for long sequences
- Gradient clipping to prevent explosion

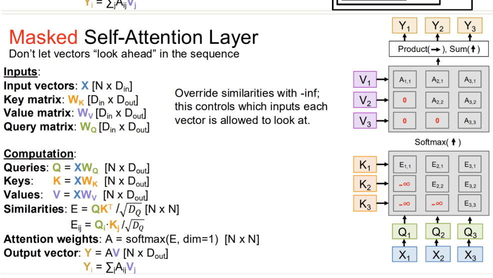
Key concepts:

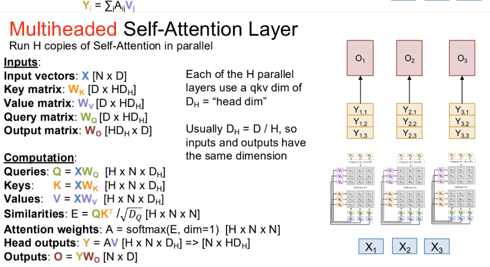
- RNNs can process variable-length sequences
- Vanishing gradients limit long-term learning
- RNNs sequential processing limits parallelization

Attention







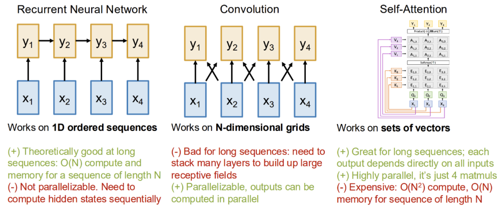


Types of Attention:

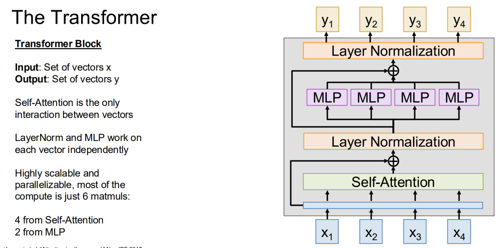
- Self-attention: Q, K, V from same sequence
- Cross-attention: Q from one, KV from another

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



Transformers



Language Modeling:

- Learn an embedding and projection matrix to convert tokens to vectors and background
- Use masked attention to prevent future peeking
- Train to predict next token with softmax + CE loss

Vision Transformer (ViT):

- Split image into patches (16×16)
- Linear projection + position embeddings (same as 16×16 conv with stride 16, $C_{\text{in}} = 3$ and $C_{\text{out}} = D$)
- Standard transformer encoder architecture
- CLS token or pooling for classification

More Details:

- Positional encodings to learn position information
- Pre-norm transformer: Normalization inside residual block allows learning identity function
- RMSNorm: Alternative normalization layer used in modern transformers
- SwiGLU MLP: Improved MLP architecture for transformer blocks
- Mixture of Experts (MoE): Uses E different MLPs but only activates $A < E$ per token, increasing parameters with modest compute cost

Semantic Segmentation

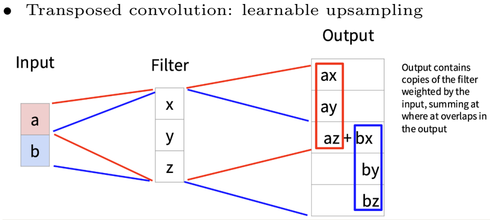
Task: Classify each pixel in an image

Architectures:

- Fully Convolutional Networks (FCN)
- U-Net: Downsample with conv (increases receptive field, loses spatial information) then upsample with transposed conv (high resolution mapping), includes skip connections

Upsampling techniques:

- Unpooling: reverse pooling operation
- Transposed convolution: learnable upsampling



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

Object Detection

R-CNN (multi-pass detection):

- **R-CNN**: Extract region proposals → CNN classifies each region independently as object or background
- **Fast R-CNN**: CNN first, then extract regions from feature maps → more efficient
- **Faster R-CNN**: Region Proposal Network (RPN) to generate proposals from feature maps
- **RPN**: Predicts objectness score and box coordinates for anchor boxes at each location
  - K bounding boxes of different size/aspect ratio at each location
  - For each anchor: binary classification (object/not) and regress box to ground truth
  - Outputs region proposals, picks top ones

YOLO (single-pass detection):

- Divides image into grid of cells → each cell predicts: prob of object, potential bounding boxes, class scores
- Each bounding box includes: (x, y, w, h, confidence)
- Single forward pass → much faster than R-CNN family

DETR (transformer-based):

- CNN backbone extracts image features → transformer encoder processes them
- Transformer decoder with object queries attends to encoded image
- Directly outputs fixed set of bounding boxes and class predictions

Instance Segmentation

Mask R-CNN:

- Extends Faster R-CNN with mask branch for segmentation, operates on each ROI and predicts binary mask
- CNN backbone → RPN → RoIAlign → mask prediction
- RoIAlign preserves spatial precision (outputs classification scores and box coordinates)

Neural Network Visualization

Saliency maps:

- Compute gradient of class score w.r.t input pixels
- Highlights regions important for classification
- Simple technique to visualize what the network looks at

Class Activation Mapping (CAM):

- Extract feature maps from the final convolutional layer
- Weight these maps using the classification layer weights
- Requires specific network architecture with global average pooling

Grad-CAM:

- Works with any CNN architecture (more flexible)
- Computes importance of feature map for target class
- Combines feature maps weighted by their importance
- Creates heatmap highlighting discriminative regions

Key concepts:

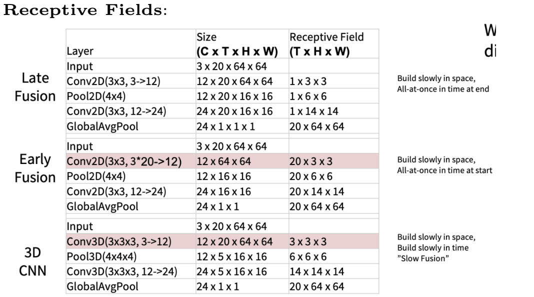
- Visualization reveal network's attention regions
- Early layers detect low-level features (edges, textures)
- Deeper layers detect high-level concepts (objects, parts)
- Helps identify dataset bias and explain model decisions
- Can verify if model focuses on relevant image regions

Video Understanding

Architectures for Video Classification:

- Single-frame CNN: Process each frame, average preds
- Late fusion: Process frames independently with CNN, combine features with MLP or pooling
- Early fusion: Treat time as channels (reshape to $3T \times H \times W$), apply 2D CNN to get class scores
- 3D CNN: 3D convolutions across space-time dimensions
- CNN + RNN: Extract CNN features from frames, feed sequence to RNN for long-term temporal modeling
- Recurrent Convolutional Network: Replace RNN matrix multiplications with convolutions
- Transformer: Space-time self-attention on video tokens

Receptive Fields:



	Layer	Size (C x T x H x W)	Receptive Field (T x H x W)	
Late Fusion	Input	3 x 20 x 64 x 64		
	Conv2D(3x3, 3->12)	12 x 20 x 64 x 64	1 x 3 x 3	Build slowly in space, All-at-once in time at end
	Pool2D(4x4)	12 x 20 x 16 x 16	1 x 6 x 6	
	Conv2D(3x3, 12->24)	24 x 20 x 16 x 16	1 x 14 x 14	
	GlobalAvgPool	24 x 1 x 1 x 1	20 x 64 x 64	
Early Fusion	Input	3 x 20 x 64 x 64		
	Conv2D(3x3, 3*20->12)	12 x 64 x 64	20 x 3 x 3	Build slowly in space, All-at-once in time at start
	Pool2D(4x4)	12 x 16 x 16	20 x 6 x 6	
	Conv2D(3x3, 12->24)	24 x 16 x 16	20 x 14 x 14	
	GlobalAvgPool	24 x 1 x 1	20 x 64 x 64	
3D CNN	Input	3 x 20 x 64 x 64		
	Conv3D(3x3x3, 3->12)	12 x 20 x 64 x 64	3 x 3 x 3	Build slowly in space, Build slowly in time "Slow Fusion"
	Pool3D(4x4x4)	12 x 5 x 16 x 16	6 x 6 x 6	
	Conv3D(3x3x3, 12->24)	24 x 5 x 16 x 16	14 x 14 x 14	
	GlobalAvgPool	24 x 1 x 1	20 x 64 x 64	

RCN Layer:

$$H_t^l = \tanh(W_{xh}^l * X_t^l + W_{hh}^l * H_{t-1}^l)$$

$$X_t^{l+1} = H_t^l$$

- Replaces matmuls in RNNs with convolutions
- Each layer is a convolution, each column is a time step
- Captures both spatial and temporal patterns simultaneously, though its slow

Key concepts:

- Early Fusion has no temporal shift invariance, needs to learn filters for same motion at different times.
- 3D CNN has temporal shift invariance, each filter slides over time.
- I3D: Inflated 2D CNN, expands 2D filters to 3D by repeating weights temporally
- Long-term temporal structure: CNN+RNN or space-time self-attention
- Video understanding benefits from multi-modal inputs (RGB, optical flow, audio)