Back-Propagation and Networks for Recognition

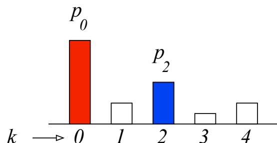
COMPSCI 527 — Computer Vision

Outline

1. Loss and Risk
2. Back-Propagation
3. Convolutional Neural Networks
4. AlexNet
5. The State of the Art of Image Classification

The 0-1 Loss is Useless for Training

- Example: $K = 5$ classes, scores $\mathbf{p} = h(\mathbf{x}_n; \mathbf{w})$ as in figure
- True label $y_n = 2$, predicted label $\hat{y}_n = 0$ because $p_0 > p_{y_n} = p_2$. Therefore, the 0-1 loss is 1



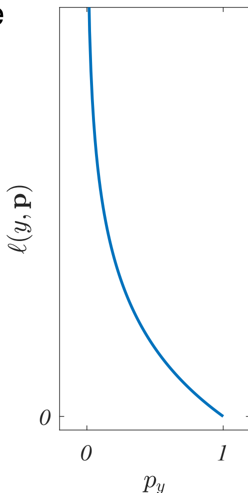
- Changing $\mathbf{w}$ by an inifinitesimal amount may *reduce* but not close the gap between p_0 and p_2: loss stays 1
- That is, $\nabla \ell_n(\mathbf{w}) = \frac{\partial \ell_{0\text{-}1}}{\partial \mathbf{w}} = 0$
- Gradient provides no information towards reducing the gap!

(Can still use 0-1 loss for *validation* or *evaluation*)

The Cross-Entropy Loss

- We compute the loss on the score vector **p**, not on the prediction $\hat{y}_n$
- Use *cross-entropy loss* on the score **p** as a proxy loss

  $\ell(y, \mathbf{p}) = -\log p_y$
- Unbounded loss for total misclassification
- Differentiable, nonzero derivative everywhere
- Meshes well with the soft-max (the layer that produces **p**)

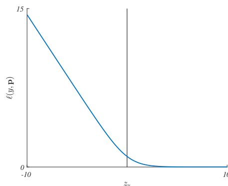


Example, Continued

- Last layer before soft-max has activations $\mathbf{z} \in \mathbb{R}^K$
- Soft-max has output $\mathbf{p} = \sigma(\mathbf{z})$ with $p_k = \frac{e^{z_k}}{\sum_{j=0}^{4} e^{z_j}} \in \mathbb{R}^5$
- $p_k > 0$ for all k and $\sum_{k=0}^{4} p_k = 1$
- Ideally, if the correct class is $y = 2$, we would like output $\mathbf{p}$ to equal $\mathbf{q} = [0, 0, 1, 0, 0]$, the *one-hot encoding* of y
- That is, $q_y = q_2 = 1$ and all other q_j are zero
- $\ell(y, \mathbf{p}) = -\log p_y = -\log p_2$
- When $\mathbf{p}$ approaches $\mathbf{q}$ we have $p_y \to 1$ and $\ell(y, \mathbf{p}) \to 0$
- When $\mathbf{p}$ is far from $\mathbf{q}$ we have $p_y \to 0$ and $\ell(y, \mathbf{p}) \to \infty$

Example, Continued

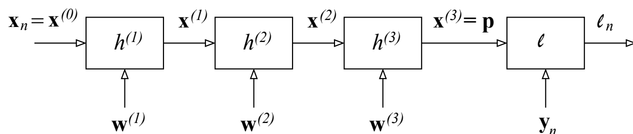
- *Cross-entropy loss meshes well with soft-max*



- $\ell(y, \mathbf{p}) = -\log p_y = -\log \frac{e^{z_y}}{\sum_{j=0}^{4} e^{z_j}} = \log(\sum_{j=0}^{4} e^{z_j}) - z_y$
- When $z_y \gg z_{y'}$ for all $y' \neq y$ we have $\log(\sum_{j=0}^{4} e^{z_j}) \approx \log e^{z_y} = z_y$ so that $\ell(y, \mathbf{p}) \to 0$
- When $z_y \ll z_{y'}$ for some $y' \neq y$ we have $\log(\sum_{j=0}^{4} e^{z_j}) \approx c$ (c effectively independent of z_y) so that $\ell(y, \mathbf{p}) \to c - z_y \to \infty$ linearly as $z_y \to -\infty$ (Actual plot depends on all values in $\mathbf{z}$)
- This is a "soft hinge loss" in $\mathbf{z}$ (not in $\mathbf{p}$)

Back-Propagation

- We need $\nabla L_B(\mathbf{w})$ over some mini-batch B and therefore $\nabla \ell_n(\mathbf{w}) = \frac{\partial \ell_n}{\partial \mathbf{w}} = \left(\frac{\partial \ell_n}{\partial \mathbf{w}_1}, \ldots, \frac{\partial \ell_n}{\partial \mathbf{w}_J}\right)^T$ for a network with J layers



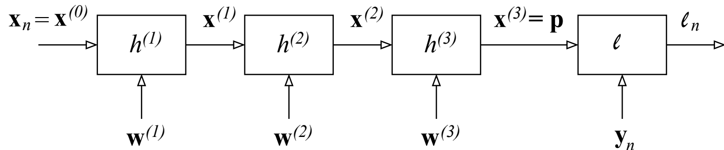
- Computations from $\mathbf{x}_n$ to ℓ_n form a chain: use the chain rule!
- Derivatives of ℓ_n w.r.t. layer j or before go through $\mathbf{x}^{(j)}$

$$\frac{\partial \ell_n}{\partial \mathbf{w}^{(j)}} = \frac{\partial \ell_n}{\partial \mathbf{x}^{(j)}} \frac{\partial \mathbf{x}^{(j)}}{\partial \mathbf{w}^{(j)}}$$

$$\frac{\partial \ell_n}{\partial \mathbf{x}^{(j-1)}} = \frac{\partial \ell_n}{\partial \mathbf{x}^{(j)}} \frac{\partial \mathbf{x}^{(j)}}{\partial \mathbf{x}^{(j-1)}} \qquad \text{(recursion!)}$$

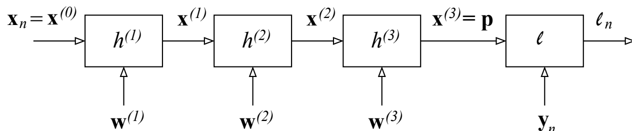
- Start: $\frac{\partial \ell_n}{\partial \mathbf{x}^{(J)}} = \frac{\partial \ell}{\partial \mathbf{p}}$

Local Jacobians



- Local computations at layer j: $\frac{\partial \mathbf{x}^{(j)}}{\partial \mathbf{w}^{(j)}}$ and $\frac{\partial \mathbf{x}^{(j)}}{\partial \mathbf{x}^{(j-1)}}$
- Partial derivatives of $h^{(j)}$ with respect to layer weights and input to the layer
- Local Jacobian matrices, can compute by knowing what the layer does
- The start of the process can be computed from knowing the loss function, $\frac{\partial \ell_n}{\partial \mathbf{x}^{(J)}} = \frac{\partial \ell}{\partial \mathbf{p}}$
- Another local Jacobian
- The rest is going recursively from output to input, one layer at a time, accumulating $\frac{\partial \ell_n}{\partial \mathbf{w}^{(j)}}$ into a vector $\frac{\partial \ell_n}{\partial \mathbf{w}}$

Back-Propagation Spelled Out for $J = 3$



$$\frac{\partial \ell_n}{\partial \mathbf{x}^{(3)}} = \frac{\partial \ell}{\partial \mathbf{p}}$$

$$\frac{\partial \ell_n}{\partial \mathbf{w}^{(3)}} = \frac{\partial \ell_n}{\partial \mathbf{x}^{(3)}} \frac{\partial \mathbf{x}^{(3)}}{\partial \mathbf{w}^{(3)}}$$

$$\frac{\partial \ell_n}{\partial \mathbf{x}^{(2)}} = \frac{\partial \ell_n}{\partial \mathbf{x}^{(3)}} \frac{\partial \mathbf{x}^{(3)}}{\partial \mathbf{x}^{(2)}}$$

$$\frac{\partial \ell_n}{\partial \mathbf{w}^{(2)}} = \frac{\partial \ell_n}{\partial \mathbf{x}^{(2)}} \frac{\partial \mathbf{x}^{(2)}}{\partial \mathbf{w}^{(2)}}$$

$$\frac{\partial \ell_n}{\partial \mathbf{x}^{(1)}} = \frac{\partial \ell_n}{\partial \mathbf{x}^{(2)}} \frac{\partial \mathbf{x}^{(2)}}{\partial \mathbf{x}^{(1)}}$$

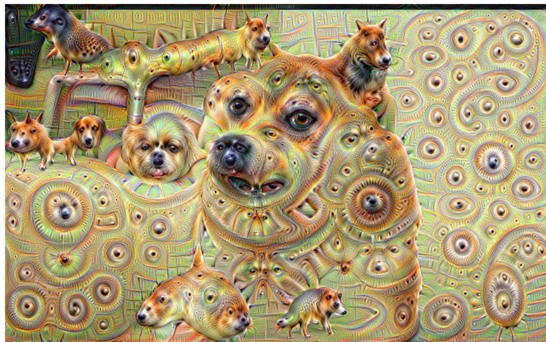
$$\frac{\partial \ell_n}{\partial \mathbf{w}^{(1)}} = \frac{\partial \ell_n}{\partial \mathbf{x}^{(1)}} \frac{\partial \mathbf{x}^{(1)}}{\partial \mathbf{w}^{(1)}}$$

$$\left(\frac{\partial \ell_n}{\partial \mathbf{x}^{(0)}} = \frac{\partial \ell_n}{\partial \mathbf{x}^{(1)}} \frac{\partial \mathbf{x}^{(1)}}{\partial \mathbf{x}^{(0)}} \right)$$

$$\frac{\partial \ell_n}{\partial \mathbf{w}} = \begin{bmatrix} \frac{\partial \ell_n}{\partial \mathbf{w}^{(1)}} \\ \frac{\partial \ell_n}{\partial \mathbf{w}^{(2)}} \\ \frac{\partial \ell_n}{\partial \mathbf{w}^{(3)}} \end{bmatrix}$$

(Jacobians in blue are local)

A Google Deep Dream Image



- Train a network to recognize animals (yields $\mathbf{w}$)
- Set $\mathbf{x}_0 =$ random noise image, $y =$ dog
- Minimize $\ell(y, h(\mathbf{x}))$ with respect to $\mathbf{x}$ rather than minimizing $L_T(\mathbf{w})$ with respect to $\mathbf{w}$

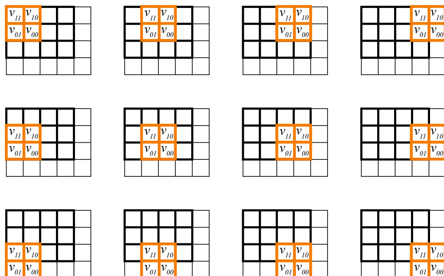
Convolutional Layers

- A layer with input $\mathbf{x} \in \mathbb{R}^d$ and output $\mathbf{y} \in \mathbb{R}^e$ has e neurons, each with d gains and one bias
- Total of $(d+1)e$ weights to be trained in a single layer
- For images, d, e are in the order of hundreds of thousands or even millions
- Too many parameters
- *Convolutional layers* are layers restricted in a special way
- Many fewer parameters to train
- Also some justification in terms of heuristic principles (see notes)

A Convolutional Layer

- Convolution + bias: $\mathbf{a} = \mathbf{x} * \mathbf{v} + b$
- Example: 3×4 input image $\mathbf{x}$, 2×2 kernel $\mathbf{v} = \begin{bmatrix} v_{00} & v_{01} \\ v_{10} & v_{11} \end{bmatrix}$

["Same" style convolution]



- Do you want to see this as one convolution with $\mathbf{v}$ (plus bias) or as 12 neurons with the same weights?

"Local" Neurons

- Neurons are now "local"
- Just means that many coefficients are zero:

0	0	0	0
0	0	v_{00}	v_{01}
0	0	v_{10}	v_{11}

- If a neuron is viewed as being connected to all input pixels, then the 12 neurons share their *nonzero* weights
- So a convolutional layer is the same as a fully-connected layer where each neuron has many weights clamped to zero, and the remaining weights are *shared* across neurons

There is Still a Gain Matrix

0 1 2 3 4
0 | | | | | 0
1 | | | v_{11} | v_{10} | 0
2 | | | v_{01} | v_{00} | 0
3 | 0 | 0 | 0 | 0 | 0

v_{00} v_{01}
v_{10} v_{11}

0 1 2 3
0
1 a_{12}
2

x **v** **a**

- Neuron number 6 (starting at 0):

$$a_{12} = v_{11}x_{12} + v_{10}x_{13} + v_{01}x_{22} + v_{00}x_{23} + b$$

- Activation number six $a_{12} = V[6,:]\,\mathbf{x}$ where

$$\begin{array}{rl} \mathbf{x} & = (x_{00}, x_{01}, x_{02}, x_{03}, x_{10}, x_{11}, x_{12}, x_{13}, x_{20}, x_{21}, x_{22}, x_{23})^T \\ V[6,:] & = (0, 0, 0, 0, 0, 0, v_{11}, v_{10}, 0, 0, v_{01}, v_{00}) \end{array}$$

Gain Matrix for a Convolutional Layer

$$\mathbf{a} = \mathbf{x} * \mathbf{v} + b \quad \text{or} \quad \mathbf{a}_{\text{flat}} = V\mathbf{x}_{\text{flat}} + b$$

$$\begin{bmatrix} a_{00} \\ a_{01} \\ a_{02} \\ a_{03} \\ a_{10} \\ a_{11} \\ a_{12} \\ a_{13} \\ a_{20} \\ a_{21} \\ a_{22} \\ a_{23} \end{bmatrix} = \begin{bmatrix}
v_{11} & v_{10} & 0 & 0 & v_{01} & v_{00} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & v_{11} & v_{10} & 0 & 0 & v_{01} & v_{00} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & v_{11} & v_{10} & 0 & 0 & v_{01} & v_{00} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & v_{11} & v_{10} & 0 & 0 & v_{01} & v_{00} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & v_{11} & v_{10} & 0 & 0 & v_{01} & v_{00} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & v_{11} & v_{10} & 0 & 0 & v_{01} & v_{00} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & v_{11} & v_{10} & 0 & 0 & v_{01} & v_{00} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & v_{11} & v_{10} & 0 & 0 & v_{01} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & v_{11} & v_{10} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & v_{11} & v_{10} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & v_{11} & v_{10} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & v_{11}
\end{bmatrix} \begin{bmatrix} x_{00} \\ x_{01} \\ x_{02} \\ x_{03} \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{20} \\ x_{21} \\ x_{22} \\ x_{23} \end{bmatrix} + b$$

- A “regular” layer with many zeros and shared weights
  [Boundary neurons have fewer nonzero weights]
- Zeros cannot be changed during training
- One scalar bias instead of 12

Stride

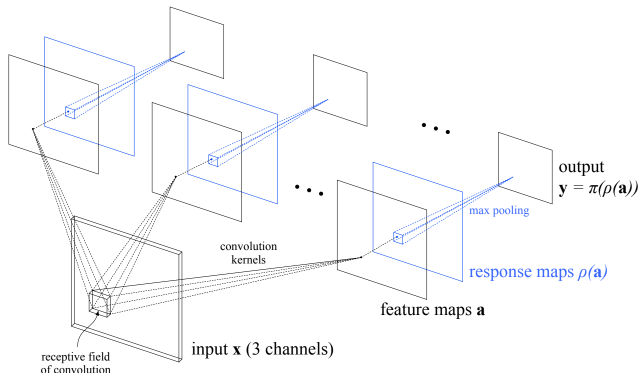
- Activation a_{ij} is often similar to $a_{i,j+1}$ and $a_{i+1,j}$
- Images often vary slowly over space
- Activations are redundant
- Reduce the redundancy by computing convolutions with a *stride* s_m greater than one
- Only compute every s_m output values in dimension m
- Output size shrinks from $d_1 \times d_2$ to about $d_1/s_1 \times d_2/s_2$
- Typically $s_m = s$ (same stride in all dimensions)
- Layers get smaller and smaller because of stride
- Multiscale image analysis, efficiency

Max Pooling

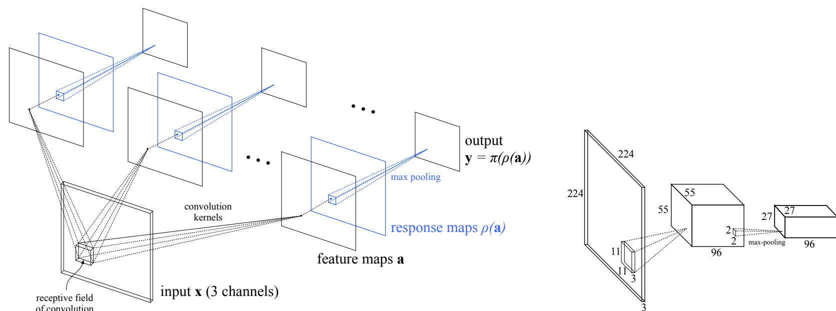
- Another way to reduce output resolution is *max pooling*
- This is a layer of its own, separate from convolution
- Consider $k \times k$ windows with stride s
- Often $s = k$ (adjacent, non-overlapping windows)
- For each window, output the maximum value
- Output is about $d_1/s \times d_2/s$
- Returns highest response in window, rather than the response in a fixed position
- More expensive than strided convolution because the entire convolution needs to be computed before the max is found in each window
- No longer very popular

The Input Layer of AlexNet

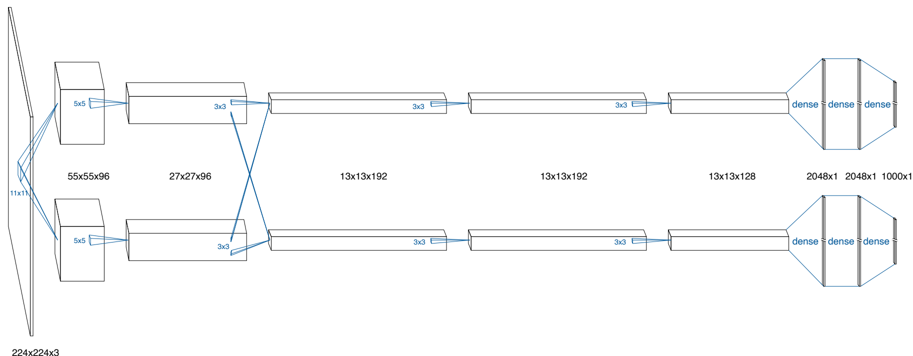
- AlexNet *circa* 2012, classifies color images into one of 1000 categories
- Trained on ImageNet, a large database with millions of labeled images



A more Compact Drawing



AlexNet



AlexNet Numbers

- Input is $224 \times 224 \times 3$ (color image)
- First layer outputs 96 feature maps of size 55×55
- A fully-connected first layer would have about $224 \times 224 \times 3 \times 55 \times 55 \times 96 \approx 4.4 \times 10^{10}$ gains
- With convolutional kernels of size 11×11, there are only $96 \times (3 \times 11^2 + 1) = 34{,}944$ weights
- That's a big deal! Locality and reuse
- Most of the complexity is in the last few, fully-connected layers, which still have millions of parameters
- More recent neural networks have much lighter final layers, but many more layers
- There are also *fully convolutional* neural networks

The State of the Art of Image Classification

- ImageNet Large Scale Visual Recognition Challenge (ILSVRC)
- Based on ImageNet,1.4 million images, 1000 categories (Fei-Fei Li, Stanford)
- Three different competitions:
  - *Classification*:
    - One label per image, 1.2M images available for training, 50k for validation, 100k withheld for testing
    - Zero-one loss for evaluation, 5 guesses allowed
  - *Localization*: Classification, plus bounding box on one instance
    Correct if $\geq$ 50% overlap with true box
  - *Detection*: Same as localization, but find every instance in the image. Measure the fraction of mistakes (false positives, false negatives)

Image classification

Steel drum

Ground truth

Steel drum Folding chair Loudspeaker	Scale T-shirt Steel drum Drumstick Mud turtle	Scale T-shirt Giant panda Drumstick Mud turtle
Accuracy: 1	Accuracy: 1	Accuracy: 0

Single-object localization

Steel drum

Ground truth

Accuracy: 1

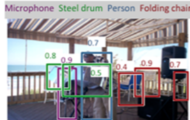
Accuracy: 0

Accuracy: 0

Object detection

Ground truth

Microphone Steel drum Person Folding chair



AP: 1.0 1.0 1.0 1.0

Microphone Steel drum Person Folding chair

AP: 0.0 0.5 1.0 0.3

Microphone Steel drum Person Folding chair

AP: 1.0 0.7 0.5 0.9

[Image from Russakovsky *et al.*, ImageNet Large Scale Visual Recognition Challenge, *Int'l. J. Comp. Vision* 115:211-252, 2015]

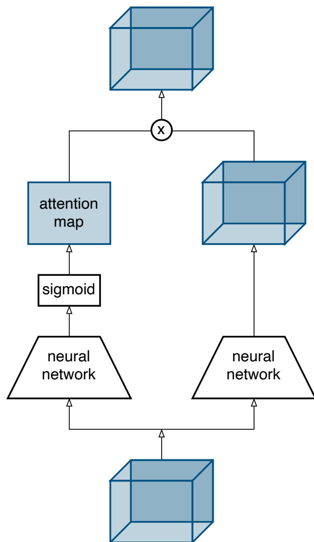
Difficulties of ILSVRC

- Images are “natural.” Arbitrary backgrounds, different sizes, viewpoints, lighting. Partially visible objects
- 1,000 categories, subtle distinctions. Example: Siberian husky and Eskimo dog
- Variations of appearance within one category can be significant (how many lamps can you think of?)
- What is the label of one image? For instance, a picture of a group of people examining a fishing rod was labeled as “reel.”

Errors on Image Classification

- Answer included in top 5:
  - 2010: 28.2 percent error rate
  - 2017: 2.3 percent (ensemble of several deep networks)
- 2021: Now we do *single top answer*, $\approx$ 10 percent error rate
- 2021 SotA has 2.4 billion weights (CoAtNet, Dai *et al.*, Google Brain). Attention mechanisms are gaining interest
  - Attention: Inputs to a layer are weighted by a learnable *attention mask* that emphasizes relevant parts of the image
  - Just add the mask, then SGD will tune it to do the right thing
- Improvement results from both architectural insights (residuals, attention, positional encoding, ...) and persistent engineering
- A book on "tricks of the trade in deep learning!"
- Problem solved? Only on ImageNet!
- "Meta-overfitting"

Attention



- The right network is the usual network (or part of it)
- The left network is similar but outputs a single channel with values in $(0, 1)$
- The product multiplies each (scalar) pixel in the attention map with each (vector) pixel in the activation map
- The rest is the same: Train the whole system by SGD