Lecture 14: Convolutional Neural Networks
Recap
Support vector machines

- Want line that maximizes the margin.

For support vectors, \( x_i \cdot w + b = \pm 1 \)

Distance between point and line:

For support vectors:

\[
\frac{w^T x + b}{\|w\|} = \pm 1 \quad M = \left| \frac{1}{\|w\|} - \frac{-1}{\|w\|} \right| = \frac{2}{\|w\|}
\]

Therefore, the margin is \( \frac{2}{\|w\|} \).
Finding the maximum margin line

- Solution: \( \mathbf{w} = \sum_{i} \alpha_i y_i \mathbf{x}_i \)

- Most of the \( \alpha_i \) will turn out to be 0.
- The non-zero \( \alpha_i \) correspond to support vectors.
Finding the maximum margin line

• Solution: \( \mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i \)

\[ b = y_i - \mathbf{w} \cdot \mathbf{x}_i \quad \text{(for any support vector)} \]

\[ \mathbf{w} \cdot \mathbf{x} + b = \sum_i \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b \]

• Classification function:

\[ f(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b) = \text{sign}(\sum_i \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b) \]

If \( f(x) < 0 \), classify as negative.
If \( f(x) > 0 \), classify as positive.

Note: only dot products with data points appear here
Non-linear SVMs: feature spaces

- General idea: the original input space can be mapped to some higher-dimensional feature space where the training set is separable:

\[ \Phi: x \rightarrow \varphi(x) \]
Nonlinear SVMs

• *The kernel trick*: instead of explicitly computing the lifting transformation $\phi(x)$, define a *kernel function* $K$ such that

$$K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$$

• This gives a nonlinear decision boundary in the original feature space:

$$\sum_i \alpha_i y_i K(x_i, x) + b$$

• Simply replace all dot products in original space with the kernel function, also expressible as a dot product:

$$f(x) = \text{sign}(w \cdot x + b) = \text{sign}(\sum_i \alpha_i y_i x_i \cdot x + b) = \text{sign}(\sum_i \alpha_i y_i K(x_i, x) + b)$$
Window-based object detection: recap

Training:
1. Obtain training data
2. Define features
3. Define classifier

Given new image:
1. Slide window
2. Score by classifier
SVMs for recognition

1. Define your representation for each example.

2. Select a kernel function.

3. Compute pairwise kernel values between labeled examples.

4. Use this “kernel matrix” to solve for SVM support vectors and weights.

5. To classify a new example: compute kernel values between new input and support vectors, apply weights, check sign of output.

Kristen Grauman
Person detection with HoGs and linear SVMs

- Map each grid cell in the input window to a histogram counting the gradients per orientation.
- Train a linear SVM using training set of pedestrian or non-pedestrian windows.

Dalal and Triggs, CVPR 2005
Person detection with HoGs and linear SVMs

Dalal and Triggs, CVPR 2005
Current object detectors

Use other cues to propose a few candidate bounding boxes, then extract rich features to classify.

R-CNN: Regions with CNN features

1. Input image
2. Extract region proposals (~2k)
3. Compute CNN features
4. Classify regions

Girschick et al., Rich feature hierarchies for accurate object detection
Learning Features
Traditional Image Categorization: Training phase

- Training Images
- Training
  - Image Features
  - Classifier Training
  - Trained Classifier
- Training Labels

Slide credit: Jia-Bin Huang
Traditional Image Categorization: Testing phase

Training Images

Training

Image Features

Classifier Training

Trained Classifier

Test Image

Testing

Image Features

Trained Classifier

Prediction Outdoor

Slide credit: Jia-Bin Huang
Features have been key

- **SIFT** [Lowe IJCV 04]
- **HOG** [Dalal and Triggs CVPR 05]
- **SPM** [Lazebnik et al. CVPR 06]
- and many others:
  - SURF, MSER, LBP, GLOH, .....
Learning a Hierarchy of Feature Extractors

• Hierarchical and expressive feature representations
• Trained end-to-end, rather than hand-crafted for each task
• Remarkable in transferring knowledge across tasks
Neural Networks
Biological neuron and Perceptrons

A biological neuron

An artificial neuron (Perceptron)

Input

\[ x_1, x_2, x_3, \ldots, x_D \]

Weights

\[ w_1, w_2, w_3, \ldots, w_D \]

Output:

\[ \text{sgn}(w \cdot x + b) \]

Slide credit: Jia-Bin Huang
Simple, Complex and Hypercomplex cells

David H. Hubel and Torsten Wiesel

Suggested a **hierarchy of feature detectors** in the visual cortex, with higher level features responding to patterns of activation in lower level cells, and propagating activation upwards to still higher level cells.

**Slide credit:** Jia-Bin Huang
Hubel-Wiesel Architecture and Multi-layer Neural Network

Hubel and Weisel’s architecture

Multi-layer Neural Network

Slide credit: Jia-Bin Huang
Neuron: Linear Perceptron

- Inputs are feature values
- Each feature has a weight
- Sum is the activation

\[
activation_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)
\]

- If the activation is:
  - Positive, output +1
  - Negative, output -1
Two-layer perceptron network
Two-layer perceptron network

Slide credit: Pieter Abeel and Dan Klein
Two-layer perceptron network

Slide credit: Pieter Abeel and Dan Klein
Two-layer perceptron network

Slide credit: Pieter Abeel and Dan Klein
Neural networks

Linear score function:

2-layer Neural Network

\[ f = Wx \]
\[ f = W_2 \max(0, W_1 x) \]

Non-linearity
Activation functions

**Sigmoid**

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

**tanh** \( \tanh(x) \)

**ReLU** \( \max(0, x) \)

**Leaky ReLU**

\[ \max(0.1x, x) \]

**ELU**

\[ f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases} \]
Neural networks

Linear score function:

2-layer Neural Network

\[ f = W x \]
\[ f = W_2 \max(0, W_1 x) \]

3-layer Neural Network

\[ f = W_3 \max(0, W_2 \max(0, W_1 x)) \]
Multi-layer neural network
Learning $w$

- **Training examples**
  $$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$$

- **Objective: a misclassification loss**
  $$\min_w \sum_{i=1}^{m} \left(y^{(i)} - h_w(f(x^{(i)}))\right)^2$$

- **Procedure:**
  - Gradient descent or hill climbing
Hill climbing

- **Simple, general idea:**
  - Start wherever
  - Repeat: move to the best neighboring state
  - If no neighbors better than current, quit
  - Neighbors = small perturbations of $w$

- **What’s bad?**
  - Optimal?

Slide credit: Pieter Abeel and Dan Klein
Two-layer perceptron network

Slide credit: Pieter Abeel and Dan Klein
Two-layer perceptron network

Slide credit: Pieter Abeel and Dan Klein
Two-layer neural network

\[ h_w(f(x)) \]

\[ z \rightarrow \frac{1}{1 + e^{-z}} \]
Neural network properties

- Theorem (Universal function approximators): A two-layer network with a sufficient number of neurons can approximate any continuous function to any desired accuracy.

- Practical considerations:
  - Can be seen as learning the features
  - Large number of neurons
    - Danger for overfitting
  - Hill-climbing can get stuck in local optima

Approximation by Superpositions of Sigmoidal Function, 1989

Slide credit: Pieter Abeel and Dan Klein
Convolutional Neural Networks
Convolutional Neural Networks

• CNN = a multi-layer neural network with
  – **Local** connectivity:
    • Neurons in a layer are only connected to a small region of the layer before it
  – **Share** weight parameters across spatial positions:
    • Learning shift-invariant filter kernels

Image credit: A. Karpathy
Significant recent impact on the field

Big labeled datasets -> Deep learning -> GPU technology

ImageNet top-5 error (%)

Slide credit: Dinesh Jayaraman
What is a Convolution?

- Weighted moving sum

slide credit: S. Lazebnik
Spatial filtering is convolution

Original image:

| 1 1 1 0 0 |
| 0 1 1 1 0 |
| 0 0 1 1 1 |
| 0 0 1 1 0 |
| 0 1 1 0 0 |

Convolutional filter 1:

| 1 0 1 |
| 0 1 0 |
| 1 0 1 |

Convolving the image:

| 1 \times 1 | 1 \times 0 | 1 \times 1 | 0 | 0 |
| 0 \times 0 | 1 \times 1 | 1 \times 0 | 1 | 0 |
| 0 \times 1 | 0 \times 0 | 1 \times 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |

Result:

| 4 |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |

Inner product:

\[ I(x, y) \ast h = \sum_{i=-a}^{a} \sum_{j=-b}^{b} I(x - i, y - j) \cdot h(i, j) \]
Spatial filtering is convolution

Original image

Convolving the image

Result

Convolutional filter 1

\[
I(x, y) * h = \sum_{i=-a}^{a} \sum_{j=-b}^{b} I(x-i, y-j) \cdot h(i, j)
\]
Spatial filtering is convolution

Original image

Convolving the image

Result

Convolutional filter 1

\[ I(x, y) * h = \sum_{i=-a}^{a} \sum_{j=-b}^{b} I(x-i, y-j) \cdot h(i, j) \]
From fully connected to convolutional networks

image

Fully connected layer
From fully connected to convolutional networks
From fully connected to convolutional networks
From fully connected to convolutional networks

- Image
- Convolutional layer
- Feature map
- Learned weights
From fully connected to convolutional networks
Dimensions of convolution

\[ h_{out} = \frac{h_{in} - h_f}{s} + 1 \]
\[ w_{out} = \frac{w_{in} - w_f}{s} + 1 \]
\[ d_{out} = n_f \]
\[ d_f = d_{in} \]
Dimensions of convolution

- Our images get smaller and smaller
- Not too deep architectures
- Details are lost

Image:

<table>
<thead>
<tr>
<th>1 _x1</th>
<th>1 _x0</th>
<th>1 _x1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 _x0</td>
<td>1 _x1</td>
<td>1 _x0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0 _x1</td>
<td>0 _x0</td>
<td>1 _x1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0 _x0</td>
<td>0 _x0</td>
<td>1 _x1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0 _x1</td>
<td>0 _x0</td>
<td>1 _x1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0 _x0</td>
<td>0 _x0</td>
<td>1 _x1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

After conv 1:

<table>
<thead>
<tr>
<th>4</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

After conv 2:

| 18 |
**Dimensions of convolution**

- For $s = 1$, surround the image with $(h_f-1)/2$ and $(w_f-1)/2$ layers of 0.

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 2 & 1 & 0 & 0 & 0 \\
1 & 0 & 2 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 3 & 0 & 0 & 0 & 0 \\
\end{array}
\]

$\ast$
Convolutional Neural Networks

- **Question:** Spatial structure?
  - **Answer:** Convolutional filters

- **Question:** Huge input dimensionalities?
  - **Answer:** Parameters are shared between filters

- **Question:** Local variances?
  - **Answer:** Pooling

[Slides credit: Efstratios Gavves]
Spatial properties

- Images are 2-D
  - k-D if you also count the extra channels
  - RGB, hyperspectral, etc.

- What does a 2-D input really mean?
  - Neighboring variables are locally correlated
Example filter when $k = 2$

Sobel filter
Learnable filters

- Several, handcrafted filters in computer vision
  - Canny, Sobel, Gaussian blur, smoothing, low-level segmentation, morphological filters, Gabor filters

- Are they optimal for recognition?
- Can we learn them from our data?
- Are they going resemble the handcrafted filters?
2D spatial filters

- If images are 2-D, parameters should also be organized in 2-D
  - That way they can learn the local correlations between input variables
  - That way they can “exploit” the spatial nature of images
k-D spatial filters

- Similarly, if images are k-D, parameters should also be k-D.
Number of weights

How many weights for this neuron?

\[ 7 \cdot 7 \cdot 3 = 147 \]
Number of weights

How many weights for these 5 neurons?

$$5 \cdot 7 \cdot 7 \cdot 3 = 735$$
Local connectivity

- The weight connections are surface-wise local!
  - Local connectivity

- The weights connections are depth-wise global

- For standard neurons no local connectivity
  - Everything is connected to everything
Filters at one spatial location

$735$ weights
Filters over the whole image

Assume the image is 30x30x3.
1 filter every pixel (stride = 1)
How many parameters in total?
Filters over the whole image

Assume the image is 30x30x3. 1 filter every pixel (stride = 1)
How many parameters in total?

24 filters along the x axis
24 filters along the y axis
Depth of 5
× 7 * 7 * 3 parameters per filter

423K parameters in total
Too many parameters

- Clearly, too many parameters
- With a only $30 \times 30$ pixels image and a single hidden layer of depth 5 we would need 85K parameters
  - With a $256 \times 256$ image we would need $46 \cdot 10^6$ parameters
- **Problem 1**: Fitting a model with that many parameters is not easy
- **Problem 2**: Finding the data for such a model is not easy
- **Problem 3**: Are all these weights necessary?
Weight sharing

Insight: Images have similar features at various spatial locations!

- So, if we are anyways going to compute the same filters, why not share?
  - Sharing is caring

Assume the image is 30x30x3.
1 column of filters common across the image.
How many parameters in total?
Weight sharing

Insight: Images have similar features at various spatial locations!

- So, if we are anyways going to compute the same filters, why not share?
  - Sharing is caring

Assume the image is 30x30x3.
1 column of filters common across the image.
How many parameters in total?

Depth of 5
\[ \times 7 \times 7 \times 3 \text{ parameters per filter} \]
735 parameters in total
When weight sharing is not good

- When images are registered and each pixel has a particular significance
  - E.g. after face alignment specific pixels hold specific types of inputs, like eyes, nose, etc.
- In these cases maybe better every spatial filter to have different parameters
  - Network learns particular weights for particular image locations [Taigman2014]
Convolutional layer is differentiable

- Activation function
  \[ a_{rc} = \sum_{i=-a}^{a} \sum_{j=-b}^{b} x_{r-i,c-j} \cdot \theta_{ij} \]

- Essentially a dot product, similar to linear layer
  \[ a_{rc} \sim x_{\text{region}}^T \cdot \theta \]

- Gradient w.r.t. the parameters
  \[ \frac{\partial a_{rc}}{\partial \theta_{ij}} = \sum_{r=0}^{N-2a} \sum_{c=0}^{N-2b} x_{r-i,c-j} \]
Pooling operations

- Aggregate multiple values into a single value

Single depth slice

\[
\begin{array}{cccc}
1 & 1 & 2 & 4 \\
5 & 6 & 7 & 8 \\
3 & 2 & 1 & 0 \\
1 & 2 & 3 & 4 \\
\end{array}
\]

max pool with 2x2 filters and stride 2

\[
\begin{array}{cc}
6 & 8 \\
3 & 4 \\
\end{array}
\]
Pooling operations

- Aggregate multiple values into a single value
- Invariance to small transformations
  - Keep only most important information for next layer
- Reduces the size of the next layer
  - Fewer parameters, faster computations
- Observe larger receptive field in next layer
  - Hierarchically extract more abstract features

![Pooling example](image)
Max Pooling

- Run a sliding window of size \([h_f, w_f]\).
- At each location keep the maximum value.
- Activation function: \(i_{\text{max}}, j_{\text{max}} = \arg \max_{i,j \in \Omega(r,c)} x_{ij} \rightarrow a_{rc} = x[i_{\text{max}}, j_{\text{max}}]\)
- Gradient w.r.t. input \(\frac{\partial a_{rc}}{\partial x_{ij}} = \begin{cases} 1, & \text{if } i = i_{\text{max}}, j = j_{\text{max}} \\ 0, & \text{otherwise} \end{cases}\)
- The preferred choice of pooling.
Average Pooling

- Run a sliding window of size \([h_f, w_f]\).
- At each location keep the maximum value.
- Activation function: \(a_{rc} = \frac{1}{r \cdot c} \sum_{i,j \in \Omega(r,c)} x_{ij}\).
- Gradient w.r.t. input: \(\frac{\partial a_{rc}}{\partial x_{ij}} = \frac{1}{r \cdot c}\).
1 x 1 convolutions

1 x 1 convolution layers also possible, equivalent to a dot product.

1x1 CONV with 32 filters

(each filter has size 1x1x64, and performs a 64-dimensional dot product)
Key operations in a CNN

- Input Image
- Convolution (Learned)
- Non-linearity
- Spatial pooling
- Feature maps

Source: R. Fergus, Y. LeCun

Slide: Lazebnik
Convolution as a feature extractor

example 5x5 filters
(32 total)

We call the layer convolutional because it is related to convolution of two signals:

\[ f[x,y] \ast g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1,y-n_2] \]

elementwise multiplication and sum of a filter and the signal (image)
Key operations in a CNN

- Input Image
- Convolution (Learned)
- Non-linearity
- Spatial pooling
- Feature maps

Rectified Linear Unit (ReLU)

Source: R. Fergus, Y. LeCun
Key operations in a CNN

- Feature maps
- Spatial pooling
- Non-linearity
- Convolution (Learned)
- Input Image

Source: R. Fergus, Y. LeCun

Slide: Lazebnik
Types of Neural Networks

Neural Network

Convolutional Neural Network
Convolutional Neural Networks
Optimization in CNNs
A 3-layer network for digit recognition

- Each grayscale image is of size 28x28.
- 60,000 training images and 10,000 test images
- 10 possible labels (0,1,2,3,4,5,6,7,8,9)

MNIST dataset

Example outputs:
6 -> [0000001000]'
The network tries to approximate the function $y(x)$ and its output is $a$.

We use a quadratic cost function, or MSE, or “L2-loss”.

$$C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2$$
Gradient descent

\[ C(w, b) \equiv \frac{1}{2n} \sum_{x} \|y(x) - a\|^2 \]

parameters to compute

\# of input samples

\[ \Delta C \approx \frac{\partial C}{\partial v_1} \Delta v_1 + \frac{\partial C}{\partial v_2} \Delta v_2 \]

Small changes in parameters to leads to small changes in output

\[ \nabla C \equiv \left( \frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2} \right)^T \]

Gradient vector!

\[ \Delta v = -\eta \nabla C \]

Change the parameter using learning rate (positive) and gradient vector!

\[ v \rightarrow v' = v - \eta \nabla C \]

Update rule!
Stochastic gradient descent

Cost function is a sum over all the training samples:

$$C(w, b) = \sum_x C_x(w, b), \quad \text{where } C_x(w, b) = \frac{1}{2} \|y(x) - a\|^2$$

Gradient from entire training set:

$$\nabla C = \frac{1}{n} \sum_x \nabla C_x$$

Usually, $n$ is very large.

Update rules for each parameter:

$$w_k \rightarrow w'_k = w_k - \eta \frac{\partial C}{\partial w_k}$$

$$b_l \rightarrow b'_l = b_l - \eta \frac{\partial C}{\partial b_l}$$
Stochastic gradient descent

Gradient from entire training set:

$$\nabla C = \frac{1}{n} \sum_x \nabla C_x$$

• For large training data, gradient computation takes a long time
  • Leads to “slow learning”

• Instead, consider a mini-batch with $m$ samples
• If sample size is large enough, properties approximate the dataset

$$\frac{\sum_{j=1}^{m} \nabla C_{X_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C.$$
Stochastic gradient descent

What if the loss function has a local minima or saddle point?

Zero gradient, gradient descent gets stuck
Stochastic gradient descent

What if loss changes quickly in one direction and slowly in another?
What does gradient descent do?
Very slow progress along shallow dimension, jitter along steep direction.

Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large.
Stochastic gradient descent

Our gradients come from minibatches so they can be noisy!

\[ L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(x_i, y_i, W) \]

\[ \nabla_W L(W) = \frac{1}{N} \sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) \]
Stochastic gradient descent

Momentum update:

Velocity

actual step

Gradient

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

Build up velocity as a running mean of gradients.
Layer to layer relationship

\[ a_j^l = \sigma(z_j^l) \]
\[ z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l \]
\[ a_j^l = \sigma \left( \sum_k w_{jk}^l a_k^{l-1} + b_j^l \right) \]

- \( b_j^l \) is the bias term in the jth neuron in the lth layer.
- \( a_j^l \) is the activation in the jth neuron in the lth layer.
- \( z_j^l \) is the weighted input to the jth neuron in the lth layer.
Cost and gradient computation

\[ C = \frac{1}{2n} \sum_x \| y(x) - a^L(x) \|^2 \]

- The goal of the backpropagation algorithm is to compute the gradients \( \frac{\partial C}{\partial w} \) and \( \frac{\partial C}{\partial b} \) of the cost function \( C \) with respect to each and every weight and bias parameters. Note that backpropagation is only used to compute the gradients.

\[ C = \frac{1}{2n} \sum_x \| y(x) - a^L(x) \|^2 \]

- Stochastic gradient descent is the training algorithm.
Chain rule of differentiation

- In order to differentiate a function $z = f(g(x))$ w.r.t $x$, we can do the following:

Let $y = g(x)$, $z = f(y)$,  $\frac{dz}{dx} = \frac{dz}{dy} \times \frac{dy}{dx}$

Let $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$, $g$ maps from $\mathbb{R}^m$ to $\mathbb{R}^n$, and $f$ maps from $\mathbb{R}^n$ to $\mathbb{R}$. If $y = g(x)$ and $z = f(y)$, then

$$\frac{\partial z}{\partial x_i} = \sum_k \frac{\partial z}{\partial y_k} \frac{\partial y_k}{\partial x_i}$$

This is all you need to know to get the gradients in a neural network!
Backpropagation:

• In order to differentiate a function $z = f(g(x))$ w.r.t $x$, we can do the following:

Let $y = g(x)$, $z = f(y)$, $\frac{dz}{dx} = \frac{dz}{dy} \times \frac{dy}{dx}$

Let $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$, $g$ maps from $\mathbb{R}^m$ to $\mathbb{R}^n$, and $f$ maps from $\mathbb{R}^n$ to $\mathbb{R}$. If $y = g(x)$ and $z = f(y)$, then

$$\frac{\partial z}{\partial x_i} = \sum_k \frac{\partial z}{\partial y_k} \frac{\partial y_k}{\partial x_i}$$

This is all you need to know to get the gradients in a neural network!
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

[Slides credit: Fei-Fei Li]
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, \ y = 5, \ z = -4 \)

\[
q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1
\]

\[
f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q
\]

Want: \( \frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} \)
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[
\begin{align*}
q &= x + y \\
\frac{\partial q}{\partial x} &= 1, \quad \frac{\partial q}{\partial y} = 1
\end{align*}
\]

\[
\begin{align*}
f &= qz \\
\frac{\partial f}{\partial q} &= z, \quad \frac{\partial f}{\partial z} = q
\end{align*}
\]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, \ y = 5, \ z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \ \frac{\partial f}{\partial y}, \ \frac{\partial f}{\partial z} \)
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

\[ q = x + y \]
\[ \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \]
\[ \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \[ \frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} \]
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)

Chain rule:

\[ \frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y} \]
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, \ y = 5, \ z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \ \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \ \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \ \frac{\partial f}{\partial y}, \ \frac{\partial f}{\partial z} \)

Chain rule:

\[ \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \ \frac{\partial q}{\partial x} \]
Backpropagation example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want:

\[ \frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} \]

Chain rule:

\[ \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x} \]

Add gate: gradient distributor

Mul gate: gradient switcher
Backpropagation example
Backpropagation example

\[ \frac{\partial z}{\partial x} \]

\[ \frac{\partial z}{\partial y} \]

\[ \frac{\partial L}{\partial z} \]

"local gradient"
Backpropagation example

\[
\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial x}
\]
Backpropagation example

\[
\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x}
\]

\[
\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial y}
\]

“local gradient”

\[
\frac{\partial z}{\partial x}
\]

\[
\frac{\partial z}{\partial y}
\]

\[
\frac{\partial L}{\partial z}
\]

generates
Backpropagation example

Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]
Patterns in backpropagation

**add** gate: gradient distributor

**max** gate: gradient router

**mul** gate: gradient switcher
Convolutional layer is differentiable

- Activation function
  \[ a_{rc} = \sum_{i=-a}^{a} \sum_{j=-b}^{b} x_{r-i,c-j} \cdot \theta_{ij} \]

- Essentially a dot product, similar to linear layer
  \[ a_{rc} \sim x_{region}^T \cdot \theta \]

- Gradient w.r.t. the parameters
  \[ \frac{\partial a_{rc}}{\partial \theta_{ij}} = \sum_{r=0}^{N-2a} \sum_{c=0}^{N-2b} x_{r-i,c-j} \]
CNN Architectures
Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network’s input is 150,528-dimensional, and the number of neurons in the network’s remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.
Architectural details of AlexNet

- Similar framework to LeCun 1998 but:
  - Bigger model (7 hidden layers, 650k units, 60M parameters)
  - More data (10^6 images instead of 10^3 images)
  - GPU implementation (50 times speedup over CPU)

![Diagram of AlexNet architecture]

18.2% error in ImageNet
Removing layer 7

1.1% drop in performance, 16 million less parameters
Removing layers 6 and 7

5.7% drop in performance, 50 million less parameters
Removing layers 3 and 4

3.0% drop in performance, 1 million less parameters. Why?
Removing layers 3, 4, 6 and 7

33.5% drop in performance. Conclusion? **Depth!**