CS 1674: Intro to Computer Vision

Neural Networks

Prof. Adriana Kovashka
University of Pittsburgh
November 16, 2016
Announcements

• Please watch the videos I sent you, if you haven’t yet (that’s your reading)
• We won’t be ready to release HW10P until tomorrow night, so we’re pushing HW9P’s deadline until then (11:59pm, Thursday)
• HW10W, HW10P will be due after Thanksgiving
Plan for next few CNN lectures

Why convolutional neural networks?

Neural network basics
- Architecture
- Biological inspiration
- Loss functions
- Optimization
- Training with backprop

CNNs
- Special operations
- Common architectures

Understanding CNNs
- Visualization
- Synthesis / style transfer
- Breaking CNNs

Practical matters
- Tips and tricks for training
- Transfer learning
- Software packages
Why CNNs?

Obtained state of the art performance on many problems…

Most papers in CVPR 2016 use deep learning

Razavian et al., CVPR 2014 Workshops
ImageNet Challenge 2012

[Deng et al. CVPR 2009]

- ~14 million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon Turk
- Challenge: 1.2 million training images, 1000 classes

AlexNet: Similar framework to LeCun’98 but:
- Bigger model (7 hidden layers, 650,000 units, 60,000,000 params)
- More data ($10^6$ vs. $10^3$ images)
- GPU implementation (50x speedup over CPU)
  - Trained on two GPUs for a week
- Better regularization for training (DropOut)

ImageNet Challenge 2012

Krizhevsky et al. -- **16.4% error** (top-5)
Next best (non-convnet) – **26.2% error**
CNN features for detection

R-CNN: Regions with CNN features

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

Object detection system overview. Our system (1) takes an input image, (2) extracts around 2000 bottom-up region proposals, (3) computes features for each proposal using a large convolutional neural network (CNN), and then (4) classifies each region using class-specific linear SVMs. **R-CNN achieves a mean average precision (mAP) of 53.7% on PASCAL VOC 2010.** For comparison, Uijlings et al. (2013) report 35.1% mAP using the same region proposals, but with a spatial pyramid and bag-of-visual-words approach. **The popular deformable part models perform at 33.4%.**


Lana Lazebnik
Object Detection

Mean Average Precision (mAP)

- DPM (2011): 33.7, 29.6
- Regionlets (2013): 41.7, 39.7
- R-CNN (2014, AlexNet): 54.2, 50.2
- R-CNN + bbox reg (AlexNet): 58.5, 53.7
- R-CNN (VGG-16): 66, 62.9

- VOC 2007
- VOC 2010
Object Detection

<table>
<thead>
<tr>
<th>Faster!</th>
<th>R-CNN</th>
<th>Fast R-CNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Time:</td>
<td>84 hours</td>
<td>9.5 hours</td>
</tr>
<tr>
<td>(Speedup)</td>
<td>1x</td>
<td>8.8x</td>
</tr>
<tr>
<td>Test time per image</td>
<td>47 seconds</td>
<td>0.32 seconds</td>
</tr>
<tr>
<td>(Speedup)</td>
<td>1x</td>
<td>146x</td>
</tr>
<tr>
<td>mAP (VOC 2007)</td>
<td>66.0</td>
<td>66.9</td>
</tr>
</tbody>
</table>

Using VGG-16 CNN on Pascal VOC 2007 dataset
Beyond classification

Detection
Segmentation
Regression
Pose estimation
Synthesis

and many more…
What are CNNs?

- Convolutional neural networks are a type of *neural network*
- The neural network includes layers that perform special operations
- Used in vision, but to a lesser extent also in NLP, biomedical, etc.
- Often they are *deep*
Deep neural network

Figure from http://neuralnetworksanddeeplearning.com/chap5.html
Traditional Recognition Approach

• Features are key to recent progress in recognition, but research shows they’re flawed…
• Where next? Better classifiers? Or keep building more features?

Adapted from Lana Lazebnik
What about learning the features?

- Learn a *feature hierarchy* all the way from pixels to classifier
- Each layer extracts features from the output of previous layer
- Train all layers jointly
“Shallow” vs. “deep” architectures

Traditional recognition: “Shallow” architecture

Image/Video Pixels → Hand-designed feature extraction → Trainable classifier → Object Class

Deep learning: “Deep” architecture

Image/Video Pixels → Layer 1 → ... → Layer N → Simple classifier → Object Class

Lana Lazebnik
Neural network definition

Figure 5.1 Network diagram for the two-layer neural network corresponding to (5.7). The input, hidden, and output variables are represented by nodes, and the weight parameters are represented by links between the nodes, in which the bias parameters are denoted by links coming from additional input and hidden variables $x_0$ and $z_0$. Arrows denote the direction of information flow through the network during forward propagation.

- **Activations:**
  \[ a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \]

- **Nonlinear activation function $h$ (e.g. sigmoid, tanh):**
  \[ z_j = h(a_j) \]
  \[ \tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} \]

Recall SVM:
\[ w^T x + b \]
Neural network definition

• Layer 2

\[ a_j = \sum_{i=1}^{D} w^{(1)}_{ji} x_i + w^{(1)}_{j0} \]

• Layer 3 (final)

\[ a_k = \sum_{j=1}^{M} w^{(2)}_{kj} z_j + w^{(2)}_{k0} \]

• Outputs

(binary) \[ y_k = \sigma(a_k) = \frac{1}{1 + \exp(-a_k)} \]

(multiclass) \[ y_k = \frac{\exp(a_k)}{\sum_j \exp(a_j)} \]

• Finally:

(binary) \[ y_k(x, w) = \sigma \left( \sum_{j=1}^{M} w^{(2)}_{kj} h \left( \sum_{i=1}^{D} w^{(1)}_{ji} x_i + w^{(1)}_{j0} \right) + w^{(2)}_{k0} \right) \]
Activation functions

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

\textbf{tanh} \hspace{1cm} \textbf{tanh}(x)

\textbf{ReLU} \hspace{1cm} \text{max}(0, x)

\textbf{Leaky ReLU} \hspace{1cm} \text{max}(0.1x, x)

\textbf{Maxout} \hspace{1cm} \text{max}(w_1^T x + b_1, w_2^T x + b_2)

\textbf{ELU} \hspace{1cm} f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases}
A multi-layer neural network

- Nonlinear classifier
- Can approximate any continuous function to arbitrary accuracy given sufficiently many hidden units
Inspiration: Neuron cells

- Neurons
  - accept information from multiple inputs,
  - transmit information to other neurons.
- Multiply inputs by weights along edges
- Apply some function to the set of inputs at each node
- If output of function over threshold, neuron “fires”
A neuron (perceptron)

Input

Weights

$\begin{align*}
  x_1 & \rightarrow w_1 \\
  x_2 & \rightarrow w_2 \\
  x_3 & \rightarrow w_3 \\
  \vdots & \\
  x_d & \rightarrow w_d
\end{align*}$

Output: $\sigma(w \cdot x + b)$

Sigmoid function:

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$
Types of neurons

Linear Neuron

Logistic Neuron

Perceptron

Θ denotes the same as w!
Multilayer networks

- Cascade neurons together
- Output from one layer is the input to the next
- Each layer has its own sets of weights
Feed-forward networks

• Predictions are fed forward through the network to classify
Feed-forward networks

- Predictions are fed forward through the network to classify
Feed-forward networks

- Predictions are fed forward through the network to classify
Feed-forward networks

- Predictions are fed forward through the network to classify
Feed-forward networks

- Predictions are fed forward through the network to classify
Feed-forward networks

- Predictions are fed forward through the network to classify
Deep neural networks

• Lots of hidden layers
• Depth = power (usually)

Figure from http://neuralnetworksanddeeplearning.com/chap5.html
How do we train them?

- It involves computing gradients
- Backpropagation (backprop): Propagates error from output layer to intermediate layers
- Goal is to iteratively find such a set of weights that allow the activations to match the desired output
- Trained with stochastic gradient descent
- We want to minimize a loss function
### Classification goal

Example dataset: **CIFAR-10**
- 10 labels
- **50,000** training images
- each image is **32x32x3**
- **10,000** test images.

<table>
<thead>
<tr>
<th>Class</th>
<th>Images</th>
</tr>
</thead>
<tbody>
<tr>
<td>airplane</td>
<td><img src="image1.png" alt="Images of airplanes" /></td>
</tr>
<tr>
<td>automobile</td>
<td><img src="image2.png" alt="Images of automobiles" /></td>
</tr>
<tr>
<td>bird</td>
<td><img src="image3.png" alt="Images of birds" /></td>
</tr>
<tr>
<td>cat</td>
<td><img src="image4.png" alt="Images of cats" /></td>
</tr>
<tr>
<td>deer</td>
<td><img src="image5.png" alt="Images of deer" /></td>
</tr>
<tr>
<td>dog</td>
<td><img src="image6.png" alt="Images of dogs" /></td>
</tr>
<tr>
<td>frog</td>
<td><img src="image7.png" alt="Images of frogs" /></td>
</tr>
<tr>
<td>horse</td>
<td><img src="image8.png" alt="Images of horses" /></td>
</tr>
<tr>
<td>ship</td>
<td><img src="image9.png" alt="Images of ships" /></td>
</tr>
<tr>
<td>truck</td>
<td><img src="image10.png" alt="Images of trucks" /></td>
</tr>
</tbody>
</table>
Classification scores

\[ f(x, W) = Wx \]

[32x32x3]
array of numbers 0...1
(3072 numbers total)

10 numbers, indicating class scores
Linear classifier

- SVM: Decision = \( \text{sign}(w^Tx) = \text{sign}(w_1x_1 + w_2x_2) \)

- What should the weights be?
Linear classifier

- We want to train a “spam or not” classifier
- Features are how many times each word occurs
- Let’s say you only have 5 words in your vocabulary: “cash”, “bank”, “homework”, “beer”, “book”
- Say I want a high score for “spam”, and low score for “not spam”
- What would the weights be on each?
Linear classifier

\[ f(x, W) = Wx + b \]

[32x32x3] array of numbers 0...1

Parameters, or “weights”

10 numbers, indicating class scores
# Linear classifier

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

![Input Image](image)

- **Input Image**: Image of a cat

- **Weights (W)**:
  - \(0.2\) \(-0.5\) \(0.1\) \(2.0\)
  - \(1.5\) \(1.3\) \(2.1\) \(0.0\)
  - \(0\) \(0.25\) \(0.2\) \(-0.3\)

- **Bias (b)**:
  - \(56\)
  - \(231\)
  - \(24\)
  - \(2\)
  - \(1.1\)
  - \(3.2\)
  - \(-1.2\)

- **scores**:
  - **Cat Score**: \(-96.8\)
  - **Dog Score**: \(437.9\)
  - **Ship Score**: \(61.95\)

\[ f(x_i; W, b) = \sum_{j} W_{ij} x_j + b_i \]

Andrej Karpathy
Linear classifier

Going forward: Loss function/Optimization

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>airplane</td>
<td>-3.45</td>
<td>-0.51</td>
</tr>
<tr>
<td>automobile</td>
<td>-8.87</td>
<td>6.04</td>
</tr>
<tr>
<td>bird</td>
<td>0.09</td>
<td>5.31</td>
</tr>
<tr>
<td>cat</td>
<td>2.9</td>
<td>-4.22</td>
</tr>
<tr>
<td>deer</td>
<td>4.48</td>
<td>-4.19</td>
</tr>
<tr>
<td>dog</td>
<td>8.02</td>
<td>3.58</td>
</tr>
<tr>
<td>frog</td>
<td>3.78</td>
<td>4.49</td>
</tr>
<tr>
<td>horse</td>
<td>1.06</td>
<td>-4.37</td>
</tr>
<tr>
<td>ship</td>
<td>-0.36</td>
<td>-2.09</td>
</tr>
<tr>
<td>truck</td>
<td>-0.72</td>
<td>-2.93</td>
</tr>
</tbody>
</table>

TODO:

1. Define a **loss function** that quantifies our unhappiness with the scores across the training data.
2. Come up with a way of efficiently finding the parameters that minimize the loss function. (**optimization**)
Suppose: 3 training examples, 3 classes. With some $W$ the scores $f(x, W) = Wx$ are:

<table>
<thead>
<tr>
<th></th>
<th>3.2</th>
<th>1.3</th>
<th>2.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td>car</td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>
### Linear classifier: SVM loss

Suppose: 3 training examples, 3 classes.  
With some $W$ the scores $f(x, W) = Wx$ are:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td></td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

### Multiclass SVM loss:

Given an example $(x_i, y_i)$ where $x_i$ is the image and $y_i$ is the (integer) label, and using the shorthand for the scores vector: $s = f(x_i, W)$, the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$
Linear classifier: SVM loss

Suppose: 3 training examples, 3 classes. With some $W$ the scores $f(x, W) = Wx$ are:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

Losses: 2.9

Multiclass SVM loss:

Given an example $(x_i, y_i)$ where $x_i$ is the image and $y_i$ is the (integer) label, and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$L_i = \sum_{j\neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$= \max(0, 5.1 - 3.2 + 1) + \max(0, -1.7 - 3.2 + 1)$$

$$= \max(0, 2.9) + \max(0, -3.9)$$

$$= 2.9 + 0$$

$$= 2.9$$
**Linear classifier: SVM loss**

Suppose: 3 training examples, 3 classes. With some $W$ the scores $f(x, W) = Wx$ are:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>score</td>
<td>3.2</td>
<td>5.1</td>
<td>-1.7</td>
</tr>
<tr>
<td>score</td>
<td>1.3</td>
<td>4.9</td>
<td>2.0</td>
</tr>
<tr>
<td>score</td>
<td>2.2</td>
<td>2.5</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

**Multiclass SVM loss:**

Given an example $(x_i, y_i)$ where $x_i$ is the image and $y_i$ is the (integer) label, and using the shorthand for the scores vector: $s = f(x_i, W)$

The SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$= \max(0, 1.3 - 4.9 + 1) + \max(0, 2.0 - 4.9 + 1)$$

$$= \max(0, -2.6) + \max(0, -1.9)$$

$$= 0 + 0$$

$$= 0$$
Suppose: 3 training examples, 3 classes. With some $W$ the scores $f(x, W) = Wx$ are:

<table>
<thead>
<tr>
<th></th>
<th>Score for cat</th>
<th>Score for car</th>
<th>Score for frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

**Losses:**

- 2.9
- 0
- 10.9

**Multiclass SVM loss:**

Given an example $(x_i, y_i)$ where $x_i$ is the image and $y_i$ is the (integer) label, and using the shorthand for the scores vector: $s = f(x_i, W)$

The SVM loss has the form:

$$ L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) $$

$$ = \max(0, 2.2 - (-3.1) + 1) + \max(0, 2.5 - (-3.1) + 1) $$

$$ = \max(0, 5.3) + \max(0, 5.6) $$

$$ = 5.3 + 5.6 $$

$$ = 10.9 $$
Linear classifier: SVM loss

Suppose: 3 training examples, 3 classes. With some $W$ the scores $f(x, W) = Wx$ are:

<table>
<thead>
<tr>
<th>Class</th>
<th>Score1</th>
<th>Score2</th>
<th>Score3</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

Losses: 2.9 0 10.9

Multiclass SVM loss:

Given an example $(x_i, y_i)$ where $x_i$ is the image and $y_i$ is the (integer) label, and using the shorthand for the scores vector: $s = f(x_i, W)$, the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

and the full training loss is the mean over all examples in the training data:

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i$$

$L = (2.9 + 0 + 10.9)/3 = 4.6$
Linear classifier: SVM loss

\[ f(x, W) = Wx \]

\[ L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) \]
Linear classifier: SVM loss

Weight Regularization

\[ L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \lambda R(W) \]

\( \lambda = \text{regularization strength (hyperparameter)} \)

In common use:

**L2 regularization**

\[ R(W) = \sum_k \sum_l W_{k,l}^2 \]

**L1 regularization**

\[ R(W) = \sum_k \sum_l |W_{k,l}| \]

**Dropout** (will see later)

Adapted from Andrej Karpathy
Another loss: Softmax

scores = unnormalized log probabilities of the classes.

\[ P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \]

where \( s = f(x_i; W) \)

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

\[ L_i = - \log P(Y = y_i|X = x_i) \]
Another loss: Softmax

\[ L_i = - \log \left( \frac{e^{s_{y_i}}}{\sum_j e^{s_j}} \right) \]

<table>
<thead>
<tr>
<th>Cat</th>
<th>3.2</th>
<th>24.5</th>
<th>0.13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Car</td>
<td>5.1</td>
<td>164.0</td>
<td>0.87</td>
</tr>
<tr>
<td>Frog</td>
<td>-1.7</td>
<td>0.18</td>
<td>0.00</td>
</tr>
</tbody>
</table>

unnormalized log probabilities
unnormalized probabilities
probabilities
How to minimize the loss function?
How to minimize the loss function?

In 1-dimension, the derivative of a function:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}$$

In multiple dimensions, the **gradient** is the vector of (partial derivatives).
current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,...]

loss 1.25347

gradient dW:

[?,
?,
?,
?,
?,
?,
?,
?,
?,
?,
?,
?,
?,
?,
?,
?,...]
<table>
<thead>
<tr>
<th>current $W$:</th>
<th>$W + h$ (first dim):</th>
<th>gradient $dW$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]</td>
<td>[0.34 + 0.0001, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]</td>
<td>[?, ?, ?, ?, ?, ?, ?, ?, ?,...]</td>
</tr>
<tr>
<td>loss 1.25347</td>
<td>loss 1.25322</td>
<td></td>
</tr>
<tr>
<td>current $W$:</td>
<td>$W + h$ (first dim):</td>
<td>gradient $dW$:</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>[0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]</td>
<td>[0.34 + 0.0001, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]</td>
<td>[-2.5, ?, ?, (1.25322 - 1.25347)/0.0001 = -2.5]</td>
</tr>
<tr>
<td>loss 1.25347</td>
<td>loss 1.25322</td>
<td></td>
</tr>
</tbody>
</table>

\[
\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}
\]

?,

?,

?,...
<table>
<thead>
<tr>
<th>current W:</th>
<th>$W + h$ (second dim):</th>
<th>gradient $dW$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]</td>
<td>[0.34, -1.11 + 0.0001, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]</td>
<td>[-2.5, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,...]</td>
</tr>
</tbody>
</table>
| loss 1.25347 | loss 1.25353 | }
<table>
<thead>
<tr>
<th>current W:</th>
<th>( W + h ) (second dim):</th>
<th>gradient ( dW ):</th>
</tr>
</thead>
<tbody>
<tr>
<td>([0.34, \quad -1.11, \quad 0.78, \quad 0.12, \quad 0.55, \quad 2.81, \quad -3.1, \quad -1.5, \quad 0.33, \ldots])</td>
<td>([0.34, \quad -1.11 + 0.0001, \quad 0.78, \quad 0.12, \quad 0.55, \quad 2.81, \quad -3.1, \quad -1.5, \quad 0.33, \ldots])</td>
<td>([-2.5, \quad 0.6, \quad ?, \quad ?, \quad ?, \quad ?, \quad (1.25353 - 1.25347)/0.0001 = 0.6])</td>
</tr>
<tr>
<td>loss 1.25347</td>
<td>loss 1.25353</td>
<td></td>
</tr>
</tbody>
</table>

\[
\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}
\]
<table>
<thead>
<tr>
<th>current W:</th>
<th>W + h (third dim):</th>
<th>gradient dW:</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0.34,</td>
<td>[0.34,</td>
<td>[-2.5,</td>
</tr>
<tr>
<td>-1.11,</td>
<td>-1.11,</td>
<td>0.6,</td>
</tr>
<tr>
<td>0.78,</td>
<td>0.78 + 0.0001,</td>
<td>?,</td>
</tr>
<tr>
<td>0.12,</td>
<td>0.12,</td>
<td>?,</td>
</tr>
<tr>
<td>0.55,</td>
<td>0.55,</td>
<td>?,</td>
</tr>
<tr>
<td>2.81,</td>
<td>2.81,</td>
<td>?,</td>
</tr>
<tr>
<td>-3.1,</td>
<td>-3.1,</td>
<td>?,</td>
</tr>
<tr>
<td>-1.5,</td>
<td>-1.5,</td>
<td>?,</td>
</tr>
<tr>
<td>0.33,...]</td>
<td>0.33,...]</td>
<td>0.33,...]</td>
</tr>
<tr>
<td>loss 1.25347</td>
<td>loss 1.25347</td>
<td></td>
</tr>
</tbody>
</table>
This is silly. The loss is just a function of $W$:

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \sum_k W_k^2$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$s = f(x; W) = Wx$$

want $\nabla_W L$
This is silly. The loss is just a function of $W$:

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \sum_k W_k^2$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$s = f(x; W) = Wx$$

want $\nabla_W L$

Calculus

$$\nabla_W L = ...$$
current $W$: 

\[
\begin{bmatrix}
0.34, \\
-1.11, \\
0.78, \\
0.12, \\
0.55, \\
2.81, \\
-3.1, \\
-1.5, \\
0.33, \\
\end{bmatrix}
\]

loss 1.25347

gradient $dW$: 

\[
\begin{bmatrix}
-2.5, \\
0.6, \\
0, \\
0.2, \\
0.7, \\
-0.5, \\
1.1, \\
1.3, \\
-2.1, \\
\end{bmatrix}
\]

\[dW = \ldots \]

(some function data and $W$)
negative gradient direction

original $W$

$W_{1}$

$W_{2}$
Gradient descent in multi-layer nets

- We’ll update weights
- Move in direction opposite to gradient:

\[ w^{(\tau+1)} = w^{(\tau)} - \eta \nabla L(w^{(\tau)}) \]
How to minimize the loss function?

• Use gradient descent
• Iteratively *subtract* the gradient with respect to the model parameters ($w$)
• I.e. we’re moving in a direction opposite to the gradient of the loss
• I.e. we’re moving towards *smaller* loss
Learning rate selection

The effects of step size (or “learning rate”)

- Very high learning rate
- Low learning rate
- High learning rate
- Good learning rate

Andrzej Karpathy
Mini-batch gradient descent

• In classic gradient descent, we compute the gradient from the loss for all training examples
• Could also only use some of the data for each gradient update, then cycle through all training samples
• Allows faster training (e.g. on GPUs), parallelization
Gradient descent in multi-layer nets

• We’ll update weights
• Move in direction opposite to gradient:

\[ w^{(\tau+1)} = w^{(\tau)} - \eta \nabla L(w^{(\tau)}) \]

• How to update the weights at all layers?
• Answer: backpropagation of error from higher layers to lower layers
Backpropagation general algorithm

- (Using sigmoid activations, two-layer net)
- Initialize all weights to small random values
- Until convergence (error stops decreasing) repeat:
  - For each \((\mathbf{x}, \; t=\text{class}(\mathbf{x}))\) in training set:
    - Calculate network outputs: \(Y_k\)
    - Compute errors (gradients wrt activations) for each unit:
      \[
      \delta_k := y_k (1-y_k) (t_k - y_k) \quad \text{for output units}
      \]
      \[
      \delta_j := y_k (1-y_k) \sum_k w_{kj} \delta_k \quad \text{for hidden units}
      \]
    - Update weights:
      \[
      w_{kj} \leftarrow w_{kj} - \eta \delta_k z_j \quad \text{for output units}
      \]
      \[
      w_{ji} \leftarrow w_{ji} - \eta \delta_j x_i \quad \text{for hidden units}
      \]

Adapted from Rebecca Hwa and Ray Mooney
Backpropagation

Entire lecture on backpropagation:
Karpathy Lecture 4 (second video I sent you)
Comments on training algorithm

• Not guaranteed to converge to zero training error, may converge to local optima or oscillate indefinitely.
• However, in practice, does converge to low error for many large networks on real data.
• Thousands of epochs (epoch = network sees all training data once) may be required, hours or days to train.
• To avoid local-minima problems, run several trials starting with different random weights (*random restarts*), and take results of trial with lowest training set error.
• May be hard to set learning rate and to select number of hidden units and layers.
• Neural networks had fallen out of fashion in 90s, early 2000s; back with a new name and significantly improved performance (deep networks trained with dropout and lots of data).
Over-training prevention

- Running too many epochs can result in over-fitting.

- Keep a hold-out validation set and test accuracy on it after every epoch. Stop training when additional epochs actually increase validation error.

Adapted from Ray Mooney
Determining best number of hidden units

- Too few hidden units prevents the network from adequately fitting the data.
- Too many hidden units can result in over-fitting.

Use internal cross-validation to empirically determine an optimal number of hidden units.
Effect of number of neurons

more neurons = more capacity
Effect of regularization

Do not use size of neural network as a regularizer. Use stronger regularization instead:

\[ \lambda = 0.001 \quad \lambda = 0.01 \quad \lambda = 0.1 \]

(you can play with this demo over at ConvNetJS: http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html)
Hidden unit interpretation

• Trained hidden units can be seen as newly constructed features that make the target concept linearly separable in the transformed space.
• On many real domains, hidden units can be interpreted as representing meaningful features such as vowel detectors or edge detectors, etc.
• However, the hidden layer can also become a distributed representation of the input in which each individual unit is not easily interpretable as a meaningful feature.