Lecture 13: Neural Networks
Administrative

- HW3 due Wednesday 3/10
Where we are:

1. Use **Linear Models** for image classification problems

2. Use **Loss Functions** to express preferences over different choices of weights

3. Use **Stochastic Gradient Descent** to minimize our loss functions and train the model

4. Add **Regularization** to control overfitting

\[ s = f(x; W) = Wx \]

1. Softmax

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

2. SVM

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

\[
L = \frac{1}{N} \sum_{i=1}^{N} L_i + R(W)
\]

\[
v = 0 \\
for t in range(num_steps):
    dw = compute_gradient(w) \\
v = rho * v + dw \\
w -= learning_rate * v\]
Problem: Linear Classifiers not enough

Geometric Viewpoint

Visual Viewpoint
One template per class:
Can’t recognize different modes of a class
One solution: Feature Transforms

Original space

\[ r = \sqrt{x^2 + y^2} \]
\[ \theta = \tan^{-1}\left(\frac{y}{x}\right) \]

Feature transform
One solution: Feature Transforms

Original space

Feature transform

r = (x^2 + y^2)^{1/2}
\theta = \tan^{-1}(y/x)

Feature space
One solution: Feature Transforms

Original space

\[ r = (x^2 + y^2)^{1/2} \]
\[ \theta = \tan^{-1}(y/x) \]

Feature transform

Feature space

Linear classifier in feature space
One solution: Feature Transforms

Original space

Feature space

$$r = (x^2 + y^2)^{1/2}$$
$$\theta = \tan^{-1}(y/x)$$

Feature transform

Nonlinear classifier in original space!

Linear classifier in feature space
Image Features: Color Histogram

Ignores texture, spatial positions

Frog image is in the public domain
Image Features: Histogram of Oriented Gradients (HoG)

1. Compute edge direction / strength at each pixel
2. Divide image into 8x8 regions
3. Within each region compute a histogram of edge directions weighted by edge strength

Lowe, "Object recognition from local scale-invariant features", ICCV 1999
Dalal and Triggs, "Histograms of oriented gradients for human detection," CVPR 2005
Image Features: Histogram of Oriented Gradients (HoG)

1. Compute edge direction / strength at each pixel
2. Divide image into 8x8 regions
3. Within each region compute a histogram of edge directions weighted by edge strength

Example: 320x240 image gets divided into 40x30 bins; 8 directions per bin; feature vector has 30*40*9 = 10,800 numbers
Image Features: Histogram of Oriented Gradients (HoG)

1. Compute edge direction / strength at each pixel
2. Divide image into 8x8 regions
3. Within each region compute a histogram of edge directions weighted by edge strength

Example: 320x240 image gets divided into 40x30 bins; 8 directions per bin; feature vector has 30*40*9 = 10,800 numbers

Lowe, "Object recognition from local scale-invariant features", ICCV 1999
Dalal and Triggs, "Histograms of oriented gradients for human detection," CVPR 2005
Image Features: Histogram of Oriented Gradients (HoG)

Captures texture and position, robust to small image changes

1. Compute edge direction / strength at each pixel
2. Divide image into 8x8 regions
3. Within each region compute a histogram of edge directions weighted by edge strength

Example: 320x240 image gets divided into 40x30 bins; 8 directions per bin; feature vector has 30*40*9 = 10,800 numbers

Image Features: Bag of Words

Learn a feature transform from data!

Step 1: Build codebook

Extract random patches
Image Features: Bag of Words

Learn a feature transform from data!

**Step 1: Build codebook**

Extract random patches

Cluster patches to form “codebook” of “visual words”
Image Features: Bag of Words

Learn a feature transform from data!

Step 1: Build codebook
- Extract random patches
- Cluster patches to form “codebook” of “visual words”

Step 2: Encode images

Fei-Fei and Perona, “A bayesian hierarchical model for learning natural scene categories”, CVPR 2005
Image Features

Common trick: Combine multiple feature transforms
Winner of 2011 ImageNet Challenge

Low-level feature extraction ≈ 10k patches per image
  • SIFT: 128-dim
  • color: 96-dim \{ \text{reduced to 64-dim with PCA} \}

FV extraction and compression:
  • N=1,024 Gaussians, R=4 regions \( \rightarrow \) 520K dim x 2
  • compression: G=8, b=1 bit per dimension

One-vs-all SVM learning with SGD

Late fusion of SIFT and color systems

Image Features vs Neural Networks

Feature Extraction

10 numbers giving scores for classes

training

Image Features vs Neural Networks

Feature Extraction

Deep Neural Network

10 numbers giving scores for classes

training

Neural Networks

Input image: \( x \in \mathbb{R}^D \)
Category scores: \( s \in \mathbb{R}^C \)

Linear Classifier:
\[
\begin{align*}
    s &= Wx \\
    W &\in \mathbb{R}^{C \times D}
\end{align*}
\]

In practice we add a learnable bias
\(+b\) after each matrix multiply
Neural Networks

Input image: $x \in \mathbb{R}^D$

Category scores: $s \in \mathbb{R}^C$

Linear Classifier:

$$s = Wx$$
$$W \in \mathbb{R}^{C \times D}$$

2-layer Neural Net:

$$s = W_2 \max(0, W_1 x)$$
$$W_1 \in \mathbb{R}^{H \times D}$$
$$W_2 \in \mathbb{R}^{C \times H}$$

In practice we add a learnable bias +b after each matrix multiply
Neural Networks

Input image: \( x \in \mathbb{R}^D \)
Category scores: \( s \in \mathbb{R}^C \)

Linear Classifier:
\[
s = Wx \quad W \in \mathbb{R}^{C \times D}
\]

2-layer Neural Net:
\[
s = W_2 \max(0, W_1 x) \quad W_1 \in \mathbb{R}^{H \times D} \quad W_2 \in \mathbb{R}^{C \times H}
\]

3-layer Neural Net:
\[
s = W_3 \max(0, W_2 \max(0, W_1 x))
\]
Neural Networks

Two-Layer Neural Network: \( s = W_2 \max(0, W_1 x) \)

\[
x \in \mathbb{R}^D, \; W_1 \in \mathbb{R}^{H \times D}, \; W_2 \in \mathbb{R}^{C \times H}
\]
Neural Networks

Two-Layer Neural Network: \( s = W_2 \max(0, W_1 x) \)

Element \((i, j)\) of \(W_1\) gives the effect on \(h_i\) from \(x_j\)

Element \((i, j)\) of \(W_2\) gives the effect on \(s_i\) from \(h_j\)

\[ x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H} \]

Input: 3072

Hidden layer: 100

Output: 10
Neural Networks

Two-Layer Neural Network: \( s = W_2 \max(0, W_1 x) \)

Element (i, j) of \( W_1 \) gives the effect on \( h_i \) from \( x_j \)

Element (i, j) of \( W_2 \) gives the effect on \( s_i \) from \( h_j \)

All elements of \( x \) affect all elements of \( h \)

“Fully-Connected” neural network
Also “Multi-Layer Perceptron” (MLP)
Neural Networks

Linear classifier:  $s = Wx$
One template per class

Two-Layer Neural Network:
$s = W_2 \max(0, W_1 x)$

$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$
Neural Networks

Neural Network:
First layer is a bank of templates
Second layer recombines templates

Two-Layer Neural Network:
\[ s = W_2 \max(0, W_1 x) \]

Input: 3072
Hidden layer: 100
Output: 10

\[ x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H} \]
Neural Networks

Different templates can cover different modes of a class!

Two-Layer Neural Network:
\[ s = W_2 \max(0, W_1 x) \]

Input: 3072
Hidden layer: 100
Output: 10

\[ x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H} \]
Neural Networks

Many templates not interpretable: “Distributed representation”

Two-Layer Neural Network:
\[ s = W_2 \max(0, W_1 x) \]

Input: 3072

Hidden layer: 100

Output: 10

\( x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H} \)
Deep Neural Networks

Depth = number of layers

Input: 3072

Output: 10

\[ s = W_6 \max(0, W_5 \max(0, W_4 \max(0, W_3 \max(0, W_2 \max(0, W_1 x)))))) \]
Activation Functions

2-layer Neural Network

The function $ReLU(z) = \max(0, z)$ is called “Rectified Linear Unit”

$s = W_2 \max(0, W_1 x)$

This is called the activation function of the neural network.
Activation Functions

2-layer Neural Network

The function $ReLU(z) = \max(0, z)$ is called “Rectified Linear Unit”

$$s = W_2 \max(0, W_1 x)$$

This is called the activation function of the neural network

Q: What happens if we build a neural network with no activation function?

$$s = W_2 W_1 x$$
Activation Functions

2-layer Neural Network

The function $ReLU(z) = \max(0, z)$ is called “Rectified Linear Unit”

$s = W_2 \max(0, W_1 x)$

This is called the activation function of the neural network

Q: What happens if we build a neural network with no activation function?

$s = W_2 W_1 x$

A: We get a linear classifier!

$W_3 = W_2 W_1 \in \mathbb{R}^{C \times D}$

$s = W_3 x$
Activation Functions

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

**Leaky ReLU**
\[ \max(0.1x, x) \]

**tanh**
\[ \tanh(x) \]

**Maxout**
\[ \max(w_1^T x + b_1, w_2^T x + b_2) \]

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

**ELU**
\[ \begin{cases} 
  x & x \geq 0 \\ 
  \alpha(e^x - 1) & x < 0 
\end{cases} \]
Activation Functions

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

**tanh**
\[ \tanh(x) \]

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

**Leaky ReLU**
\[ \max(0.1x, x) \]

**Maxout**
\[ \max(w_1^T x + b_1, w_2^T x + b_2) \]

**ELU**
\[ \begin{cases} 
  x & x \geq 0 \\
  \alpha(e^x - 1) & x < 0 
\end{cases} \]
Neural Net in <20 lines!

```python
import numpy as np
from numpy.random import randn

N, Din, H, Dout = 64, 1000, 100, 10
x, y = randn(N, Din), randn(N, Dout)
w1, w2 = randn(Din, H), randn(H, Dout)
for t in range(10000):
    h = 1.0 / (1.0 + np.exp(-x.dot(w1)))
    y_pred = h.dot(w2)
    loss = np.square(y_pred - y).sum()
    dy_pred = 2.0 * (y_pred - y)
    dw2 = h.T.dot(dy_pred)
    dh = dy_pred.dot(w2.T)
    dw1 = x.T.dot(dh * h * (1 - h))
    w1 -= 1e-4 * dw1
    w2 -= 1e-4 * dw2
```
Neural Net in <20 lines!

Initialize weights and data

```python
import numpy as np
from numpy.random import randn

N, Din, H, Dout = 64, 1000, 100, 10
x, y = randn(N, Din), randn(N, Dout)

w1, w2 = randn(Din, H), randn(H, Dout)

for t in range(10000):
    h = 1.0 / (1.0 + np.exp(-x.dot(w1)))
    y_pred = h.dot(w2)
    loss = np.square(y_pred - y).sum()
    dy_pred = 2.0 * (y_pred - y)
    dw2 = h.T.dot(dy_pred)
    dh = dy_pred.dot(w2.T)
    dw1 = x.T.dot(dh * h * (1 - h))
    w1 -= 1e-4 * dw1
    w2 -= 1e-4 * dw2
```
Neural Net in <20 lines!

Initialize weights and data

```
import numpy as np
from numpy.random import randn

N, Din, H, Dout = 64, 1000, 100, 10
x, y = randn(N, Din), randn(N, Dout)
w1, w2 = randn(Din, H), randn(H, Dout)
```

Compute loss (sigmoid activation, L2 loss)

```
for t in range(10000):
    h = 1.0 / (1.0 + np.exp(-x.dot(w1)))
y_pred = h.dot(w2)
loss = np.square(y_pred - y).sum()
dy_pred = 2.0 * (y_pred - y)
dw2 = h.T.dot(dy_pred)
dh = dy_pred.dot(w2.T)
dw1 = x.T.dot(dh * h * (1 - h))
w1 -= 1e-4 * dw1
w2 -= 1e-4 * dw2
```
Neural Net in <20 lines!

Initialize weights and data

Compute loss (sigmoid activation, L2 loss)

Compute gradients

```
import numpy as np
from numpy.random import randn

N, Din, H, Dout = 64, 1000, 100, 10
x, y = randn(N, Din), randn(N, Dout)
w1, w2 = randn(Din, H), randn(H, Dout)

for t in range(10000):
    h = 1.0 / (1.0 + np.exp(-x.dot(w1)))
    y_pred = h.dot(w2)
    loss = np.square(y_pred - y).sum()
    dy_pred = 2.0 * (y_pred - y)
    dw2 = h.T.dot(dy_pred)
    dh = dy_pred.dot(w2.T)
    dw1 = x.T.dot(dh * h * (1 - h))
    w1 -= 1e-4 * dw1
    w2 -= 1e-4 * dw2
```
Neural Net in <20 lines!

Initialize weights and data

```
import numpy as np
from numpy.random import randn

N, Din, H, Dout = 64, 1000, 100, 10
x, y = randn(N, Din), randn(N, Dout)
w1, w2 = randn(Din, H), randn(H, Dout)
```

Compute loss (sigmoid activation, L2 loss)

```
for t in range(10000):

    h = 1.0 / (1.0 + np.exp(-x.dot(w1)))
    y_pred = h.dot(w2)
    loss = np.square(y_pred - y).sum()
    dy_pred = 2.0 * (y_pred - y)
    dw2 = h.T.dot(dy_pred)
    dh = dy_pred.dot(w2.T)
    dw1 = x.T.dot(dh * h * (1 - h))

    w1 -= 1e-4 * dw1
    w2 -= 1e-4 * dw2
```
“Neural” Networks
Our brains are made of Neurons

- **Cell body**
- **Dendrites**
- **Axon**
Our brains are made of Neurons

- **Cell body**
- **Dendrite**
- **Axon**
- **Presynaptic terminal**
- **Synapse**
Our brains are made of Neurons

- **Cell body**
- **Dendrite**
- **Axon**
- **Presynaptic terminal**
- **Synapse**

**Impulses**
- Carried toward cell body
- Carried away from cell body
Our brains are made of Neurons

- **Cell body**
- **Dendrite**
- **Axon**
- **Presynaptic terminal**

Impulses carried toward cell body

Impulses carried away from cell body

Firing rate is a nonlinear function of inputs
Neurons and Artificial Neurons

**Biological Neuron**
- Dendrites
- Cell body
- Axon
- Presynaptic terminal

**Artificial Neuron**
- Input layer
- Hidden layer 1
- Hidden layer 2
- Output layer

The mathematical representation of an artificial neuron is:

\[ f \left( \sum_{i} w_i x_i + b \right) \]

Where:
- \( x_i \) are the inputs
- \( w_i \) are the weights
- \( b \) is the bias
- \( f \) is the activation function

*Neuron image by Felipe Perucho is licensed under CC BY 3.0*
Biological Neurons: Complex connectivity patterns

Neurons in a neural network: Organized into regular layers for computational efficiency
Be very careful with brain analogies!

**Biological Neurons:**
- Many different types
- Can perform complex non-linear computations
- Synapses are not a single weight but a complex non-linear dynamical system
- Can have feedback, time-dependent
- Probably don’t learn via gradient descent

[Dendritic Computation. London and Hausser]
Space Warping

Consider a linear transform: \( h = Wx \)
Where \( x, h \) are both 2-dimensional
Space Warping

Consider a linear transform: $h = Wx$

Where $x, h$ are both 2-dimensional
Space Warping

Consider a linear transform: \( h = Wx \)
Where \( x, h \) are both 2-dimensional
Space Warping

Consider a linear transform: $h = Wx$
Where $x$, $h$ are both 2-dimensional

Points not linearly separable in original space
Space Warping

Consider a linear transform: $h = Wx$
Where $x$, $h$ are both 2-dimensional

Points not linearly separable in original space

Points not linearly separable in feature space

Feature transform: $h = Wx$
Consider a neural net hidden layer:
\[ h = \text{ReLU}(Wx) = \max(0, Wx) \]
Where \( x, h \) are both 2-dimensional.
Consider a neural net hidden layer:
\[ h = \text{ReLU}(Wx) = \max(0, Wx) \]
Where \( x, h \) are both 2-dimensional.
Consider a neural net hidden layer:

\[ h = \text{ReLU}(Wx) = \max(0, Wx) \]

Where \( x, h \) are both 2-dimensional.
Space Warping

Consider a neural net hidden layer:
\[ h = \text{ReLU}(Wx) = \max(0, Wx) \]
Where \( x, h \) are both 2-dimensional.

Feature transform:
\[ h = \text{ReLU}(Wx) \]
- B is “collapsed” onto +\( h_2 \) axis
- D “collapsed” onto +\( h_1 \) axis
Space Warping

Consider a neural net hidden layer:
\[ h = \text{ReLU}(Wx) = \max(0, Wx) \]
Where \( x, h \) are both 2-dimensional

Feature transform:
\[ h = \text{ReLU}(Wx) \]

- A is “collapsed” onto +\( h_2 \) axis
- B is “collapsed” onto +\( h_1 \) axis
- C “collapsed” onto origin
- D “collapsed” onto +\( h_1 \) axis
Consider a neural net hidden layer:
\[ h = \text{ReLU}(Wx) = \max(0, Wx) \]
Where \( x, h \) are both 2-dimensional.

Feature transform:
\[ h = Wx \]

Points not linearly separable in original space.
Consider a neural net hidden layer:
\[ h = \text{ReLU}(Wx) = \max(0, Wx) \]
Where \( x, h \) are both 2-dimensional

Points not linearly separable in original space
Space Warping

Consider a neural net hidden layer: 
\[ h = \text{ReLU}(Wx) = \max(0, Wx) \]
Where \( x, h \) are both 2-dimensional

Points not linearly separable in original space

Points are linearly separable in features space!
Space Warping

Linear classifier in feature space gives nonlinear classifier in original space.

Consider a neural net hidden layer:
\[ h = \text{ReLU}(Wx) = \max(0, Wx) \]
Where \( x, h \) are both 2-dimensional.

Feature transform:
\[ h = \text{ReLU}(Wx) \]

Points are linearly separable in features space!
Points not linearly separable in original space!
Neural Networks Web Demo

(Web demo with ConvNetJS: http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html)
Next Time: How to compute gradients? Backpropagation