Feature Learning with Neural Networks
(aka Deep Learning)

CS194: Intro to Computer Vision and Comp. Photo
Angjoo Kanazawa, UC Berkeley, Fall 2021
Recap: Filter Bank Response

We can form a feature vector from the list of responses at each pixel.

[r1, r2, ..., r38]
The story so far

- Feature bank responses are biologically motivated image representations
- This is useful for recognition tasks:
  - Ex: Cluster the features into “visual words” to represent an image to do classification
  - Convolution with a linear kernel followed by simple non-linearities is a good model for early visual cortex (Retina, LGN, V1), but how to go beyond this?

How can we learn these filters??
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

[ slide courtesy Yann LeCun ]
Next Step

• To learn good features, we first need a task!
Quick Background on Statistical Learning Framework
Common Tasks

Image Classification

"Cat"
Common Tasks

Object Detection
Common Tasks

Keypoint Detection (project 5!)
Detection, semantic segmentation, instance segmentation

image classification

object detection

semantic segmentation

instance segmentation

Slide courtesy of Lana Lezebnik
Image classification

Slides courtesy of Lana Lezebnik
The statistical learning framework

- Apply a prediction function to a feature representation of the image to get the desired output:

\[
f(\text{apple}) = \text{“apple”}
\]
\[
f(\text{tomato}) = \text{“tomato”}
\]
\[
f(\text{cow}) = \text{“cow”}
\]
The statistical learning framework

\[ y = f(x) \]

- **Training set:** given a *training set* of labeled examples \( \{(x_1, y_1), \ldots, (x_N, y_N)\} \), estimate the prediction function \( f \) by minimizing the prediction error on the training set.
- **Test set:** apply \( f \) to a never before seen *test example* \( x \) and output the predicted value \( y = f(x) \).
- **Validation set:** same as test, held-out to tune hyper-parameters. Never use the “test” set for tuning! That’s cheating!

Slide modified from Lana Lezebnik
Steps

Training

- Training Images
- Training Labels
- Image Features
- Training
- Learned model

Testing

- Test Image
- Image Features
- Prediction

Slide credit: D. Hoiem
“Classic” recognition pipeline

- Hand-crafted feature representation
- Off-the-shelf trainable classifier
Classifiers: Nearest neighbor

\[ f(x) = \text{label of the training example nearest to } x \]

All we need is a distance or similarity function for our inputs
No training required!
Linear classifiers: Binary classification

Find a linear function to separate the classes:

$$f(x) = \text{sgn}(w \cdot x + b)$$
Linear classifiers: Perceptron

Input

Weights

Output: \( \text{sgn}(w \cdot x + b) \)

Can incorporate bias as component of the weight vector by always including a feature with value set to 1
Loose inspiration: Human neurons
Find a *linear function* to separate the classes:

\[ f(x) = \text{sgn}(w \cdot x + b) \]
Objective (Loss) functions

- Find weights $w$ to minimize the prediction loss between true and estimated labels of training examples:

$$E(w) = \sum_i l(x_i, y_i; w)$$

- Possible losses (for binary problems):
  - **Quadratic loss**: $l(x_i, y_i; w) = (f_w(x_i) - y_i)^2$
  - Log Likelihood loss: $l(x_i, y_i; w) = -\log P_w(y_i | x_i)$
Perceptron training algorithm

• Initialize weights \( w \) randomly
• Cycle through training examples in multiple passes (epochs)
• For each training example \( x \) with label \( y \):
  • Classify with current weights:
    \[
y' = \text{sgn}(w \cdot x)\]
  • If classified incorrectly, update weights:
    \[
w \leftarrow w + \alpha(y - y')x\]
    \( (\alpha \text{ is a positive learning rate that decays over time}) \)
Perceptron update rule: Binary Classification

\[ y' = \text{sgn}(w \cdot x) \]

\[ w \leftarrow w + \alpha(y - y')x \]

- The raw response of the classifier changes to
  \[ w \cdot x + \alpha(y - y')\|x\|^2 \]

- If \( y = 1 \) and \( y' = -1 \), the response is initially negative and will be increased
- If \( y = -1 \) and \( y' = 1 \), the response is initially positive and will be decreased
Linaer Classifiers: Beyond Binary Classification

Example Setup: 3 classes

Model – one weight per class:

\[ w_0^T x \] big if cat
\[ w_1^T x \] big if dog
\[ w_2^T x \] big if hippo

Stack together: \( W_{3 \times F} \) where \( x \) is in \( \mathbb{R}^F \)
### 3-Classes

#### Weight Matrix
- **Cat weight vector**
  - 0.2, 0.5, 0.1, 2.0, 1.1
- **Dog weight vector**
  - 1.5, 1.3, 2.1, 0.0, 3.2
- **Hippo weight vector**
  - 0.0, 0.3, 0.2, 0.3, 1.2

#### Prediction
- **$Wx_i$**
- **Cat score** = -96.8
- **Dog score** = 437.9
- **Hippo score** = 61.95

#### Weight Matrix Explanation
- **$W$** is a weight matrix that contains scoring functions, one per class.
- **$x_i$** is the prediction vector, where the $j$th component is the "score" for the $j$th class.

**Diagram by:** Karpathy, Fei-Fei
Training 3-Classes

Weight matrix a collection of scoring functions, one per class:

\[
W = \begin{bmatrix}
0.2 & 1.5 & 0.0 \\
0.5 & 1.3 & 0.3 \\
0.1 & 2.1 & 0.2 \\
2.0 & 0.0 & 0.3 \\
1.1 & 3.2 & 1.2 \\
\end{bmatrix}
\]

Prediction is vector where jth component is “score” for jth class:

\[
s_i = Wx_i
\]

Gain the correct class to have probability = 1

Cat weight vector
- 0.5
- 0.1
- 2.0
- 0.3
- 1.1

Dog weight vector
- 1.5
- 1.3
- 2.1
- 0.0
- 3.2

Hippo weight vector
- 0.0
- 0.3
- 0.2
- 0.3
- 1.2

\[
L_i = -\log(p_{y_i})
\]

Cat score
- 96.8

Dog score
437.9

Hippo score
61.95

Cat probability
0.9999

Dog probability
0.0000

Hippo probability
0.0000

\[
p_i = \frac{\exp(s_i)}{\sum_j \exp(s_j)}
\]

Diagram by: Karpathy, Fei-Fei
Visualizing linear classifiers

Linear Classifiers: Visual Intuition

Decision rule is \( \mathbf{w}^T \mathbf{x} \). If \( \mathbf{w}_i \) is big, then big values of \( x_i \) are indicative of the class.

Deer or Plane?
Decision rule is $w^T x$. If $w_i$ is big, then big values of $x_i$ are indicative of the class.

**Ship or Dog?**
Are linear classifiers enough??

One template per class:
Can’t recognize different modes of a class
Why?? Linear Classifiers not enough

**Geometric Viewpoint**

**Visual Viewpoint**

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

\[
\begin{aligned}
  r &= (x^2 + y^2)^{1/2} \\
  \theta &= \tan^{-1}(y/x)
\end{aligned}
\]
One solution: **Feature Transforms**

Original space

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

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

Feature transform

Feature space
One solution: **Feature Transforms**

\[
\begin{align*}
r &= (x^2 + y^2)^{1/2} \\
\theta &= \tan^{-1}(y/x)
\end{align*}
\]
One solution: **Feature Transforms**

Neural Networks (MLPs) learn the transforms itself from data

Original space

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

Feature space

- $r$
- $\theta$

Nonlinear classifier in original space!

Linear classifier in feature space

Feature transform
Recognition pipeline: With Perceptrons

Image Pixels → Feature representation → Perceptron → Class label
Recognition pipeline: With Multiple Perceptrons

Image Pixels $\rightarrow$ Feature representation $\rightarrow$ Perceptron $\rightarrow$ Perceptron $\rightarrow$ Class label

Multi-Layer Perceptrons (MLPs), Fully-connected networks

Can replace feature computation with layers of perceptrons! = Deep Network
“Deep” recognition pipeline

- Learn a feature hierarchy from pixels to classifier
- Each layer extracts features from the output of previous layer
- Train all layers jointly

Slide courtesy of Lana Lezebnik
Loose inspirations: Biological Neurons

[Diagram of a neuron with labels: Cell body, Axon, Dendrite]
Loose inspirations: Biological Neurons

- Synapse
- Cell body
- Dendrite
- Axon
- Presynaptic terminal
Loose inspirations: Biological Neurons

Cell body
Axon
Dendrite
Presynaptic terminal

Impulses carried toward cell body
Impulses carried away from cell body
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]
Multi-layer perceptrons

• To make nonlinear classifiers out of perceptrons, build a multi-layer neural network!
  • This requires each perceptron to have a nonlinearity
Remember this?

Image Classification

"Cat"
Remember this?

Image Classification

"Cat"
Multi-layer perceptrons / Fully-Connected Layer

![Diagram of a multi-layer perceptron with an input layer, two hidden layers, and an output layer. The input is labeled x, and the output is labeled "Cat". The layers are connected with weights W1, W2, and W3.]
Multi-layer perceptrons

- To make nonlinear classifiers out of perceptrons, build a multi-layer neural network!
  - This requires each perceptron to have a nonlinearity
  - To be trainable, the nonlinearity should be *differentiable*

\[
Sigmoid: \quad g(t) = \frac{1}{1 + e^{-t}}
\]

\[
Rectified linear unit (ReLU): \quad g(t) = \max(0, t)
\]

Slide courtesy of Lana Lezebnik
Why do we need non-linearities?

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

**3-layer Neural Net:**
$$s = W_3 \max(0, W_2 \max(0, W_1 x))$$

**Q:** with no activation function?
$$s = W_2 W_1 x$$

---

Slide modified from Johnson & Fouhey
Training of multi-layer networks

- Find network weights to minimize the prediction loss between true and estimated labels of training examples:

\[ E(w) = \sum_{i} l(x_i, y_i; w) \]

- Update weights by gradient descent:

\[ w \leftarrow w - \alpha \frac{\partial E}{\partial w} \]
Training of multi-layer networks

- Find network weights to minimize the prediction loss between true and estimated labels of training examples:

\[ E(w) = \sum_{i} l(x_i, y_i; w) \]

- Update weights by **gradient descent**: \( w \leftarrow w - \alpha \frac{\partial E}{\partial w} \)

- **Back-propagation**: gradients are computed in the direction from output to input layers and combined using chain rule

- **Stochastic gradient descent**: compute the weight update w.r.t. one training example (or a small batch of examples) at a time, cycle through training examples in random order in multiple epochs

Slide courtesy of Lana Lezebnik
**Back prop in 2 slides**

Where \( x^l = \sigma(W^lx^{l-1}) \)

What do we need for training \( W \)s?

\[
\frac{\partial L}{\partial W^3} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial W^3}
\]

\[
\frac{\partial L}{\partial W^2} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial x^2} \frac{\partial x^2}{\partial W^2}
\]

Chain rule!!
Back prop intuition

1. Every single layer is like a team of neurons (x), that a manager (W) is listening to decide outputs (y).

2. Then you get a feedback from higher ups of what you should've done.

3. You adjust your confidence in each of your neurons,

4. Then pass on what they should've said.
The point is

Each layer needs to adjust how it listened to each neuron (W) and compute gradients wrt its input and pass it down.

\[
\frac{\partial L}{\partial W^l} = \frac{\partial L}{\partial x^{l+1}} \frac{\partial x^{l+1}}{\partial W^l}
\]

Updates: Higher up orders multiplied by how much the manager listened to each neuron

\[
\frac{\partial L}{\partial x^l} = \frac{\partial L}{\partial x^{l+1}} \frac{\partial x^{l+1}}{\partial x^l}
\]

Pass down: Higher up orders, weighted by how much the manager listened to each neuron
What we have so far: Fully Connected Layers (MLPs)

Example: 200x200 image
40K hidden units

- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..

2B parameters!!!
Locally Connected Layer

Example: 200x200 image

40K hidden units

Filter size: 10x10

4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).
Locally Connected Layer

**STATIONARITY?** Statistics is similar at different locations

Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters
Convolutional Layer

Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels
Convolutional Neural Networks
Neural Nets: a particularly useful Black Box

image $X$ \hspace{1cm} \text{Convolutional Neural Network} \hspace{1cm} \text{“Penguin”} \hspace{1cm} \text{label } Y
Classic Object Recognition

Feature extractors

Edges
Texture
Colors

Segments
Parts

Classifier

“Penguin”

Slide by Philip Isola
Classic Object Recognition

image X

Feature extractors

Edges

Texture

Colors

Segments

Parts

Classifier

“Penguin”

label Y

Slide by Philip Isola
Learning Features

image $X$  

Feature extractors

- Edges
- Texture
- Colors

label $Y$
Neural Network: algorithm + feature + data!

Slide by Philip Isola
Vanilla (fully-connected) Neural Networks
Fully Connected Layer

Example: 200x200 image
40K hidden units

2B parameters!!!

- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..
Locally Connected Layer

Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).
Locally Connected Layer

**STATIONARITY?** Statistics is similar at different locations

Example: 200x200 image
- 40K hidden units
- Filter size: 10x10
- 4M parameters
Convolutions with learned kernels

Share the same parameters across different locations (assuming input is stationary):
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Learn multiple filters.

E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters
Convolution Layer

32x32x3 image

32 height

32 width

3 depth
**Convolution Layer**

32x32x3 image

5x5x3 filter

**Convolve** the filter with the image i.e. “slide over the image spatially, computing dot products”
Convolution Layer

A 32x32x3 image is convolved with a 5x5x3 filter. Filters always extend the full depth of the input volume. The process is referred to as convolving the filter with the image, i.e., "slide over the image spatially, computing dot products."
Convolution Layer

- 32x32x3 image
- 5x5x3 filter \( w \)

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

\[ w^T x + b \]
Convolution Layer

32x32x3 image
5x5x3 filter

32
32
32

convolve (slide) over all spatial locations

activation map

1
28
28
Convolution Layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

consider a second, green filter

activation maps
For example, if we had 6 5x5 filters, we’ll get 6 separate activation maps:

We stack these up to get a “new image” of size 28x28x6!