Feature Learning with Neural Networks
(aka Deep Learning)
Images of the Russian Empire:
Colorizing the Prokodin-Gorskii photo collection

Janise Liang
Proj 2 highlights: results next class

Andrew Zhang
Recap: Filter Bank Response

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

\[ [r1, r2, \ldots, r38] \]
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??
How to learn?

- To learn useful features, we first need a task!
Quick Background on the Statistical Learning Framework
Common Tasks

Image Classification

"Cat"
Common Tasks

Object Detection

“Cat”
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
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!
Steps

Training
- Training Images

Testing
- Test Image

Image Features

Training Labels

Training

Image Features

Learned model

Prediction

Learned model
“Classic” recognition pipeline

- Hand-crafted feature representation
- Off-the-shelf trainable classifier

Slide modified from Lana Lezebnik
Non-Linear 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!
Find a linear function to separate the classes:

\[
f(x) = \text{sgn}(w \cdot x + b)\]
Linear classifiers: Binary Classification aka Perceptron

Input

Weights

$X_1$ $W_1$

$X_2$ $W_2$

$X_3$ $W_3$

\ldots

$X_D$ $W_D$

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:

\[ f(x) = \text{sgn}(\mathbf{w} \cdot \mathbf{x} + b) \]
Perceptron training algorithm

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

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

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

• The raw response of the classifier changes to

\[ \mathbf{w} \cdot \mathbf{x} + \alpha (y - y') \|\mathbf{x}\|^2 \]

• If \( y = 1 \) and \( y' = -1 \), the response is initially \textit{negative} and will be \textit{increased}

• If \( y = -1 \) and \( y' = 1 \), the response is initially \textit{positive} and will be \textit{decreased}
Other Linear Classifiers: Logistic Regression

Soft version of sign with the sigmoid fn $\sigma$

$$y' = \sigma(w \cdot x + b)$$

To classify, threshold: $f(x) = y' > 0.5$
Logistic Regression: Probabilistic Interpretation

\[ \sigma(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-(wx + b)}} \]

Ground-truth label = \{0, 1\}

\[ P(Y = 1 \mid x) = \sigma(w \cdot x + b) \]

what is \( P(Y = 0 \mid x) \)?

\[ = 1 - P(Y = 1 \mid x) \]
How do we find, or learn, the W?

We need an objective!! aka Loss function

“How far am I from the true labels aka ground truth”
Objective (Loss) functions

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

\[
\text{minimize}_w L(w) = \sum_i l(x_i, y_i; w)
\]

• Possible losses (for binary problems):
  • **Quadratic loss:** \(l(x_i, y_i; w) = (f_w(x_i) - y_i)^2\)
  • Log Likelihood loss: \(l(x_i, y_i; w) = -\log P_w(y_i \mid x_i)\)
    \[= -\sum_k \mathbb{1}(y_i = k) \log P_w(k \mid x_i)\]
    \(k = \{0, 1\}\) for binary case, can be multi-class \(k=\{0, \ldots K\}\)
Linear Classifiers: Beyond Binary Classification

Example Setup: 3 classes

Model – one weight per class:

$$w_0^T x \quad \text{big if cat}$$
$$w_1^T x \quad \text{big if dog}$$
$$w_2^T x \quad \text{big if hippo}$$

Stack together: $$W_{3x_F}$$ where $$x$$ is in $$\mathbb{R}^F$$
3-Classes

Weight matrix $W$ a collection of scoring functions, one per class

$W \mathbf{x}_i$ Prediction is vector where $j$th component is “score” for $j$th class.

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

Loss

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

Weight matrix \( W \) is a collection of scoring functions, one per class.

Prediction is vector where the \( j \)th component is "score" for the \( j \)th class.

\[ s_i = W x_i \]

Want the correct class to have probability = 1.

Diagram by: Karpathy, Fei-Fei
Visualizing linear classifiers

Linear Classifiers: Visual Intuition

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

Deer or Plane?
Linear Classifiers: Visual Intuition

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

Original space

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

Feature transform
One solution: Feature Transforms

Original space

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

Feature space

Feature transform
One solution: **Feature Transforms**

Original space

Feature transform

Feature space

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

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

Linear classifier in feature space
One solution: **Feature Transforms**

Neural Networks (MLPs) learn the transforms itself from data

![Diagram showing feature transforms](image)

Original space:
- Nonlinear classifier in original space!

Feature space:
- Linear classifier in feature space

Mathematical transformations:
- \( r = (x^2 + y^2)^{1/2} \)
- \( \theta = \tan^{-1}(y/x) \)
Recognition pipeline: With Perceptrons

Image Pixels $\rightarrow$ Feature representation $\rightarrow$ Perceptron $\rightarrow$ Class label
Recognition pipeline: With Multiple Perceptrons

Image Pixels ➔ Feature representation ➔ Perceptron ➔ Perceptron ➔ 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
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

\[ \begin{align*}
W^3 & \quad y \\
W^2 & \quad x^2 \\
W^1 & \quad x^1 \\
x & \quad x
\end{align*} \]
Non-linearity

- 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*

One fully-connected layer:

\[ z = W x + b \]
\[ y = \sigma(z) \]

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

**Rectified linear unit (ReLU):**
\[ \sigma(z) = \max(0, z) \]

Most popular. Use subderivatives at 0
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 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
Recall: Chain Rule

\[ h(x) = f(g(x)) \]

\[ h'(x) = f'(g(x))g'(x) \]

What's the derivative?

Let

\[ z = g(x) \]
\[ y = f(x) \]

\[ \frac{dy}{dx} = \frac{dy}{dz} \times \frac{dz}{dx} \]
Back prop in 2 slides

What do we need for training $W$s?

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

Chain rule!!

$$\frac{\partial L}{\partial W^3} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial W^3}$$

$$\frac{\partial L}{\partial W^2} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial x^2} \frac{\partial x^2}{\partial W^2}$$
Intuitive Backprop Analogy

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

2. Then the boss gives you a feedback on what your output \(y\) should’ve been.

3. You adjust how you should’ve listened to each neurons.

4. Each neuron passes on the frustration \((L)\) weighted by how much they were listened to.

5. Repeat down the chain of report.
Backprop as workplace analogy

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 how much to listen to each: Higher up loss multiplied by how loud each neuron was

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

Pass down: Higher up orders, weighted by how much each neuron was listened to
Case study with MNIST

Training set: 60k images, with their ground truth label =\{0, 1, \ldots, 9\}

Test set: 10k

Each image is 28x28 black and white.

Fig. 4. Size-normalized examples from the MNIST database.
One layer NN

input:

\[ x = 28 \times 28 \times 1 \]

represents how much this weight or "manager" listens to this pixel \((i,j)\)

\[ W = 10 \times 28 \times 28 \]

\[ y = 10 \times 1 \]

take index with the maximum votes as the predicted digit
Displaying the Weights

Per-pixel intensity represents the magnitude of the weight and the color (black or white) represents the sign.

Slide by Geoff Hinton
Learn the weights

Show the network an image and **increment** the weights from active pixels to the correct class.

Then **decrement** the weights from active pixels to whatever class the network guesses.
The image
The image
The learned weights

The image

Slide by Geoff Hinton
Why the simple learning algorithm is insufficient

• This setup (single learned layer) is equivalent to having a rigid template for each shape.
  – The winner is the template that has the biggest overlap with the ink.

• The ways in which hand-written digits vary are much too complicated to be captured by simple template matches of whole shapes.

Slide by Geoff Hinton
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

CS194: Computer Vision and Comp. Photo
Angjoo Kanazawa & Alexei Efros, UC Berkeley, Fall 2022
Neural Nets: a particularly useful Black Box

Convolutional Neural Network

image X \rightarrow \text{Convolutional Neural Network} \rightarrow \text{"Penguin"}

label Y
Classic Object Recognition

Feature extractors:
- Edges
- Texture
- Colors

Classifier:
- Segments
- Parts

Label Y: “Penguin”
Classic Object Recognition

Image $X$ to label $Y$.

Feature extractors:
- Edges
- Texture
- Colors

Classifier:
- Segments
- Parts

"Penguin"
Learning Features

Feature extractors

- Edges
- Texture
- Colors

image X

label Y

Learned
Neural Network: algorithm + feature + data!

image X

label Y

“Penguin”
Vanilla (fully-connected) Neural Networks
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 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
Convolutional Layer
Learn multiple filters.

E.g.: 200x200 image

100 Filters

Filter size: 10x10

10K parameters
before:

```
input layer
```

```
hidden layer
```

```
output layer
```

now:

```
input layer
```

```
hidden layer
```

```
output layer
```
Convolution Layer

32x32x3 image

height

width

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

32x32x3 image

5x5x3 filter

Filters always extend the full depth of the input volume

Convolve 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. $5\times 5\times 3 = 75$-dimensional dot product + bias)

$$w^T x + b$$
Convolution Layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map
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!