Deep Learning Tutorial

Courtesy of Hung-yi Lee
Machine learning is a field of computer science that gives computers the ability to **learn without being explicitly programmed**.

Methods that can learn from and make predictions on data.
Types of Learning

**Supervised**: Learning with a *labeled training* set
Example: email *classification* with already labeled emails

**Unsupervised**: Discover *patterns* in *unlabeled* data
Example: *cluster* similar documents based on text

**Reinforcement learning**: learn to *act* based on *feedback/reward*
Example: learn to play Go, reward: *win or lose*

Classifiers:
- Classification
- Anomaly Detection
- Sequence labeling

Regression

Clustering

ML vs. Deep Learning

Most machine learning methods work well because of **human-designed representations** and **input features**. ML becomes just **optimizing weights** to best make a final prediction.

### Machine Learning in Practice

- **Describing your data with features a computer can understand**
  - Domain specific, requires Ph.D. level talent

- **Learning algorithm**
  - Optimizing the weights on features

### Table: Features vs. NER

<table>
<thead>
<tr>
<th>Feature</th>
<th>NER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Word</td>
<td>✓</td>
</tr>
<tr>
<td>Previous Word</td>
<td>✓</td>
</tr>
<tr>
<td>Next Word</td>
<td>✓</td>
</tr>
<tr>
<td>Current Word Character n-gram</td>
<td>all</td>
</tr>
<tr>
<td>Current POS Tag</td>
<td>✓</td>
</tr>
<tr>
<td>Surrounding POS Tag Sequence</td>
<td>✓</td>
</tr>
<tr>
<td>Current Word Shape</td>
<td>✓</td>
</tr>
<tr>
<td>Surrounding Word Shape Sequence</td>
<td>✓</td>
</tr>
<tr>
<td>Presence of Word in Left Window</td>
<td>size 4</td>
</tr>
<tr>
<td>Presence of Word in Right Window</td>
<td>size 4</td>
</tr>
</tbody>
</table>
What is Deep Learning (DL)?

A machine learning subfield of learning representations of data. Exceptional effective at learning patterns.
Deep learning algorithms attempt to learn (multiple levels of) representation by using a hierarchy of multiple layers.
If you provide the system tons of information, it begins to understand it and respond in useful ways.

[Diagram of Machine Learning and Deep Learning processes]

Traditional and deep learning

(a) Traditional vision pipeline

(b) Classic machine learning pipeline

(c) Deep learning pipeline
Why is DL useful?

- Manually designed features are often over-specified, incomplete and take a long time to design and validate.
- Learned Features are easy to adapt, fast to learn.
- Deep learning provides a very flexible, (almost?) universal, learnable framework for representing world, visual and linguistic information.
- Can learn both unsupervised and supervised.
- Effective end-to-end joint system learning.
- Utilize large amounts of training data.

In ~2010 DL started outperforming other ML techniques first in speech and vision, then NLP.
Image Classification: A core task in Computer Vision

(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

\[\text{cat}\]
The Problem: Semantic Gap

What the computer sees

An image is just a big grid of numbers between [0, 255]:

e.g. 800 x 600 x 3
(3 channels RGB)
Challenges: Viewpoint variation

All pixels change when the camera moves!
Challenges: Illumination
Challenges: Deformation
**Challenges:** Occlusion
Challenges: Background Clutter
Challenges: Intraclass variation
Linear Classification
Recall CIFAR10

- airplane
- automobile
- bird
- cat
- deer
- dog
- frog
- horse
- ship
- truck

50,000 training images
each image is 32x32x3

10,000 test images.
**Parametric Approach**

**Image**

Array of $32 \times 32 \times 3$ numbers (3072 numbers total)

$f(x, W)$

10 numbers giving class scores

$W$

parameters or weights
Parametric Approach: Linear Classifier

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

Image

Array of 32x32x3 numbers (3072 numbers total)

**W**

parameters or weights

10 numbers giving class scores
Parametric Approach: Linear Classifier

Image

Array of $32 \times 32 \times 3$ numbers (3072 numbers total)

$\begin{bmatrix} W \end{bmatrix}$

Parameters or weights

$W$

$3072 \times 1$

$10 \times 1$

$10 \times 3072$

$f(x, W) = \begin{bmatrix} W \end{bmatrix} x$

10 numbers giving class scores

Lecture 2
Parametric Approach: Linear Classifier

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

Array of 32x32x3 numbers (3072 numbers total)

Image

W

parameters or weights

10 numbers giving class scores
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Input image

Stretch pixels into column

W

b

Cat score

Dog score

Ship score
Example for 2x2 image, 3 classes (cat/dog/ship)

Input image
(2, 2)

Stretch pixels into column

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

\[
\begin{array}{c|c|c}
56 & 231 & \\
24 & 2 & \\
\end{array}
\]

\[
\begin{array}{c|c|c}
56 & 231 & 24 & 2 \\
\end{array}
\]

(4, )
Example for 2x2 image, 3 classes
\( \text{(cat/dog/ship)} \)

\[
\begin{align*}
\text{Input image} & \quad (2, 2) \\
\begin{array}{c|c|c|c|c}
\hline
56 & 0.2 & -0.5 & 0.1 & 2.0 \\
\hline
231 & 1.5 & 1.3 & 2.1 & 0.0 \\
\hline
24 & 0 & 0.25 & 0.2 & -0.3 \\
\hline
2 & & & & \\
\end{array}
\end{align*}
\]

\[
W \quad (3, 4)
\]

\[
\begin{array}{c|c|c|c|c}
\hline
56 & 1.1 & & & \\
\hline
231 & 3.2 & & & \\
\hline
24 & -1.2 & & & \\
\hline
2 & & & & \\
\end{array}
\]

\[
b \quad (3,)
\]

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

EECS 498-007 Lecture 2 - 24
Linear Classifier: Algebraic Viewpoint

Input image (2, 2)

Stretch pixels into column

\[
W = \begin{pmatrix}
0.2 & -0.5 & 0.1 & 2.0 \\
1.5 & 1.3 & 2.1 & 0.0 \\
0 & 0.25 & 0.2 & -0.3 \\
\end{pmatrix}
\]

\[
b = \begin{pmatrix}
1.1 \\
231 \\
24 \\
2 \\
\end{pmatrix}
\]

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

EECS 498-007 Lecture 2 - 25
Linear Classifier: Bias Trick

Add extra one to data vector; bias is absorbed into last column of weight matrix

Stretch pixels into column

Input image (2, 2)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>231</td>
</tr>
<tr>
<td>24</td>
<td>2</td>
</tr>
</tbody>
</table>

W (3, 5)

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>-0.5</td>
<td>0.1</td>
<td>2.0</td>
<td>1.1</td>
</tr>
<tr>
<td>1.5</td>
<td>1.3</td>
<td>2.1</td>
<td>0.0</td>
<td>3.2</td>
</tr>
<tr>
<td>0</td>
<td>0.25</td>
<td>0.2</td>
<td>-0.3</td>
<td>-1.2</td>
</tr>
</tbody>
</table>

= (5,)

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
</tr>
<tr>
<td>231</td>
</tr>
<tr>
<td>24</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

= (3,)

-96.8

437.9

61.95
Linear Classifier: Predictions are Linear!

\[ f(x, W) = Wx \quad \text{(ignore bias)} \]

\[ f(cx, W) = W(cx) = c \times f(x, W) \]
Linear Classifier: Predictions are Linear!

\[ f(x, W) = Wx \quad \text{(ignore bias)} \]

\[ f(cx, W) = W(cx) = c \times f(x, W) \]

Image

Scores

0.5 * Image

0.5 * Scores

\[
\begin{array}{c|c|c}
\text{Image} & \text{Scores} & \text{0.5 * Image} \\
\hline
\text{Image} & -96.8 & -48.4 \\
& 437.8 & 218.9 \\
& 62.0 & 31.0 \\
\end{array}
\]
Interpreting a Linear Classifier

Algebraic Viewpoint

\[ f(x,W) = Wx + b \]
Interpreting a Linear Classifier

**Algebraic Viewpoint**

\[ f(x, W) = Wx + b \]
Interpreting an Linear Classifier

<table>
<thead>
<tr>
<th>airplane</th>
<th>automobile</th>
<th>bird</th>
<th>cat</th>
<th>deer</th>
<th>dog</th>
<th>frog</th>
<th>horse</th>
<th>ship</th>
<th>truck</th>
</tr>
</thead>
</table>

\[
\begin{align*}
W & = \begin{bmatrix} 0.2 & -0.5 \\ 0.1 & 2.0 \\ 1.5 & 1.3 \\ 2.1 & 0.0 \\ 0 & 0.25 \\ 0.2 & -0.3 \end{bmatrix} \\
b & = \begin{bmatrix} 1.1 \\ -96.8 \\ 3.2 \\ 437.9 \\ -12 \end{bmatrix}
\end{align*}
\]
Interpreting an Linear Classifier:
Visual Viewpoint

airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck

W

<table>
<thead>
<tr>
<th></th>
<th>0.2</th>
<th>-0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.1</td>
<td>2.0</td>
</tr>
<tr>
<td>b</td>
<td>1.1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1.5</th>
<th>1.3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.1</td>
<td>0.0</td>
</tr>
<tr>
<td>b</td>
<td>3.2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0.25</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.2</td>
<td>-0.3</td>
</tr>
<tr>
<td>b</td>
<td>-1.2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>437.9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>61.95</td>
</tr>
</tbody>
</table>

plane car bird cat deer dog frog horse ship truck
Interpreting an Linear Classifier: Visual Viewpoint

Linear classifier has one “template” per category
Interpreting an Linear Classifier: Visual Viewpoint

Linear classifier has one “template” per category
A single template cannot capture multiple modes of the data

e.g. horse template has 2 heads!
Interpreting a Linear Classifier: Geometric Viewpoint

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

Array of 32x32x3 numbers (3072 numbers total)
Interpreting a Linear Classifier: Geometric Viewpoint

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

Array of $32 \times 32 \times 3$ numbers (3072 numbers total)

Car score increases this way

Pixel (11, 11, 0)

Pixel (15, 8, 0)

Car Score = 0
Interpreting a Linear Classifier: **Geometric Viewpoint**

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

Array of \(32 \times 32 \times 3\) numbers (3072 numbers total)

Plot created using [Wolfram Cloud](https://www.wolframcloud.com)

Cat image by [Nikita](https://creativecommons.org/licenses/by/2.0) is licensed under [CC BY 2.0](https://creativecommons.org/licenses/by/2.0)
Hard Cases for a Linear Classifier

**Class 1:**
First and third quadrants

**Class 2:**
Second and fourth quadrants

**Class 1:**
$1 \leq L_2\text{ norm} \leq 2$

**Class 2:**
Everything else

**Class 1:**
Three modes

**Class 2:**
Everything else
Linear Classifier: Three Viewpoints

Algebraic Viewpoint

$$f(x, W) = Wx$$

Visual Viewpoint

One template per class

Geometric Viewpoint

Hyperplanes cutting up space
So Far: Defined a linear **score** function

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

Given a \( W \), we can compute class scores for an image \( x \).

But how can we actually choose a good \( W \)?

<table>
<thead>
<tr>
<th>Category</th>
<th>Score 1</th>
<th>Score 2</th>
<th>Score 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>airplane</td>
<td>-3.45</td>
<td>-0.51</td>
<td>3.42</td>
</tr>
<tr>
<td>automobile</td>
<td>-8.87</td>
<td>6.04</td>
<td>4.64</td>
</tr>
<tr>
<td>bird</td>
<td>0.09</td>
<td>5.31</td>
<td>2.65</td>
</tr>
<tr>
<td>cat</td>
<td>2.9</td>
<td>-4.22</td>
<td>5.1</td>
</tr>
<tr>
<td>deer</td>
<td>4.48</td>
<td>-4.19</td>
<td>2.64</td>
</tr>
<tr>
<td>dog</td>
<td>8.02</td>
<td>3.58</td>
<td>5.55</td>
</tr>
<tr>
<td>frog</td>
<td>3.78</td>
<td>4.49</td>
<td>-4.34</td>
</tr>
<tr>
<td>horse</td>
<td>1.06</td>
<td>-4.37</td>
<td>-1.5</td>
</tr>
<tr>
<td>ship</td>
<td>-0.36</td>
<td>-2.09</td>
<td>-4.79</td>
</tr>
<tr>
<td>truck</td>
<td>-0.72</td>
<td>-2.93</td>
<td>6.14</td>
</tr>
</tbody>
</table>
Choosing a good $W$

$$f(x, W) = Wx + b$$

TODO:

1. Use a **loss function** to quantify how good a value of $W$ is

2. Find a $W$ that minimizes the loss function (optimization)
Loss Function

A loss function tells how good our current classifier is

Low loss = good classifier
High loss = bad classifier

(Also called: objective function; cost function)
Loss Function

A loss function tells how good our current classifier is

Low loss = good classifier
High loss = bad classifier

(Also called: objective function; cost function)

Negative loss function sometimes called reward function, profit function, utility function, fitness function, etc
Loss Function

A **loss function** tells how good our current classifier is

- Low loss = good classifier
- High loss = bad classifier

(Also called: **objective function**; **cost function**)

Negative loss function sometimes called **reward function**, **profit function**, **utility function**, **fitness function**, etc

Given a dataset of examples

\[
\{(x_i, y_i)\}_{i=1}^{N}
\]

Where \( x_i \) is image and \( y_i \) is (integer) label
Loss Function

A **loss function** tells how good our current classifier is

Low loss = good classifier
High loss = bad classifier

(Also called: **objective function**; **cost function**)

Negative loss function sometimes called **reward function**, **profit function**, **utility function**, **fitness function**, etc

Given a dataset of examples

\[
\{(x_i, y_i)\}_{i=1}^{N}
\]

Where \(x_i\) is image and \(y_i\) is (integer) label

Loss for a single example is

\[
L_i(f(x_i, W), y_i)
\]
Loss Function

A **loss function** tells how good our current classifier is

Low loss = good classifier
High loss = bad classifier

(Also called: **objective function**; **cost function**)

Negative loss function sometimes called **reward function**, **profit function**, **utility function**, **fitness function**, etc

Given a dataset of examples

$$\{ (x_i, y_i) \}_{i=1}^N$$

Where $x_i$ is image and $y_i$ is (integer) label

Loss for a single example is

$$L_i(f(x_i, W), y_i)$$

Loss for the dataset is average of per-example losses:

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$
Multiclass SVM Loss

"The score of the correct class should be higher than all the other scores"
Multiclass SVM Loss

"The score of the correct class should be higher than all the other scores"

Score for correct class

Loss

Highest score among other classes
Multiclass SVM Loss

"The score of the correct class should be higher than all the other scores"

Loss

“Hinge Loss”

Score for correct class

Highest score among other classes

“Margin”
Multiclass SVM Loss

"The score of the correct class should be higher than all the other scores"

Given an example \((x_i, y_i)\) (\(x_i\) is image, \(y_i\) is label)

Let \(s = f(x_i, W)\) be scores

Then the SVM loss has the form:

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

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

**Data loss**: Model predictions should match training data
Regularization: Beyond Training Error

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

**Data loss**: Model predictions should match training data

**Regularization**: Prevent the model from doing too well on training data
Regularization: Beyond Training Error

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

- **Data loss**: Model predictions should match training data
- **Regularization**: Prevent the model from doing **too** well on training data

\[ \lambda = \text{regularization strength (hyperparameter)} \]
Regularization: Beyond Training Error

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

\( \lambda \) = regularization strength (hyperparameter)

**Data loss**: Model predictions should match training data

**Regularization**: Prevent the model from doing too well on training data

**Simple examples**
- **L2 regularization**: 
  \[ R(W) = \sum_k \sum_l W_{k,l}^2 \]
- **L1 regularization**: 
  \[ R(W) = \sum_k \sum_l |W_{k,l}| \]
- **Elastic net (L1 + L2)**: 
  \[ R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}| \]

**More complex**: 
- Dropout
- Batch normalization
- Cutout, Mixup, Stochastic depth, etc...
Regularization: Beyond Training Error

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

- **Data loss**: Model predictions should match training data
- **Regularization**: Prevent the model from doing *too* well on training data

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

**Purpose of Regularization:**
- Express preferences in among models beyond "minimize training error"
- Avoid **overfitting**: Prefer simple models that generalize better
- Improve optimization by adding curvature
Regularization: Prefer Simpler Models
Regularization: Prefer Simpler Models

The model $f_1$ fits the training data perfectly
The model $f_2$ has training error, but is simpler
Regularization: Prefer Simpler Models

Regularization pushes against fitting the data too well so we don’t fit noise in the data.

\[ f_1 \text{ is not a linear model; could be polynomial regression, etc} \]

\[ x \]
\[ \begin{align*} y \\ f_1 \\ f_2 \end{align*} \]
Regularization: Prefer Simpler Models

Regularization is important! You should (usually) use it.

Regularization pushes against fitting the data too well so we don’t fit noise in the data.

\[ f_1 \]

\[ f_2 \]

F1 is not a linear model; could be polynomial regression, etc.
Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

<table>
<thead>
<tr>
<th>Animal</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
</tr>
</tbody>
</table>
Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as probabilities

\[ s = f(x_i; W) \]

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

Softmax function

cat \hspace{1cm} 3.2

car \hspace{1cm} 5.1

frog \hspace{1cm} -1.7
Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

$$s = f(x_i; W)$$

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

**Softmax function**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
</tr>
</tbody>
</table>

Unnormalized log-probabilities / logits
Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

\[ s = f(x_i; W) \]

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

Softmax function

<table>
<thead>
<tr>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2</td>
<td>5.1</td>
<td>-1.7</td>
</tr>
</tbody>
</table>

Unnormalized log-probabilities / logits

<table>
<thead>
<tr>
<th>0</th>
<th>24.5</th>
<th>164.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.18</td>
<td>0.018</td>
</tr>
</tbody>
</table>

Unnormalized probabilities

Probabilities must be >= 0
Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

\[
s = f(x_i; W)
\]

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

Unnormalized log-probabilities / logits

Unnormalized probabilities

Probabilities

Probabilities must be $\geq 0$

Probabilities must sum to 1

**Softmax function**
**Cross-Entropy Loss (Multinomial Logistic Regression)**

Want to interpret raw classifier scores as **probabilities**

\[ s = f(x_i; W) \]

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

Softmax function

Unnormalized log-probabilities / logits

Unnormalized probabilities

Probabilities

Probabilities must be \( \geq 0 \)

Probabilities must sum to 1

\[ L_i = -\log P(Y = y_i | X = x_i) \]

Cat: 3.2

Car: 5.1

Frog: -1.7

\[ \begin{array}{c|c|c|c|c}
\text{cat} & 3.2 & 24.5 & 0.13 & \text{L}_i = -\log(0.13) = 2.04 \\
\text{car} & 5.1 & 164. & 0.87 & \\
\text{frog} & -1.7 & 0.18 & 0.00 & 
\end{array} \]

EECS 498-007 Lecture 2 - 82
Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as probabilities

\[ s = f(x_i; W) \]

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

Softmax function

<table>
<thead>
<tr>
<th>Cat</th>
<th>Car</th>
<th>Frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2</td>
<td>5.1</td>
<td>-1.7</td>
</tr>
</tbody>
</table>

Unnormalized log-probabilities / logits

Unnormalized probabilities

Probabilities must be >= 0
Probabilities must sum to 1

\[ L_i = -\log P(Y = y_i | X = x_i) \]

Maximum Likelihood Estimation
Choose weights to maximize the likelihood of the observed data

\[ L_i = -\log(0.13) = 2.04 \]
Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as probabilities

\[ s = f(x_i; W) \]

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

Softmax function

\[ L_i = -\log P(Y = y_i | X = x_i) \]

Probabilities must be \( \geq 0 \)
Probabilities must sum to 1

<table>
<thead>
<tr>
<th>cat</th>
<th>3.2</th>
<th>24.5</th>
<th>0.13</th>
<th>1.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>car</td>
<td>5.1</td>
<td>164.</td>
<td>0.87</td>
<td>0.00</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
<td>0.18</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Unnormalized log-probabilities / logits
Unnormalized probabilities
Correct probs

EECS 498-007 Lecture 2 - 84
Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as probabilities

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

Probabilities must be \( \geq 0 \)
Probabilities must sum to 1

\[ L_i = -\log P(Y = y_i | X = x_i) \]

Want to compare unnormalized log-probabilities

Unnormalized log-probabilities / logits
Unnormalized probabilities
Correct probs

Compare

\[ D_{KL}(P \parallel Q) = \sum_y P(y) \log \frac{P(y)}{Q(y)} \]
**Cross-Entropy Loss (Multinomial Logistic Regression)**

Want to interpret raw classifier scores as **probabilities**

\[ s = f(x_i; W) \]

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

Softmax function

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>raw</td>
<td>3.2</td>
<td>5.1</td>
<td>-1.7</td>
</tr>
<tr>
<td>exp</td>
<td>24.5</td>
<td>164.</td>
<td>0</td>
</tr>
<tr>
<td>exp</td>
<td>0</td>
<td>0</td>
<td>0.18</td>
</tr>
</tbody>
</table>

Probabilities must be >= 0

Probabilities must sum to 1

\[ L_i = - \log P(Y = y_i | X = x_i) \]

Cross Entropy

\[ H(P, Q) = H(p) + D_{KL}(P \parallel Q) \]

Correct probs

EECS 498-007 Lecture 2 - 86
Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

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

Maximize probability of correct class

\[ L_i = -\log P(Y = y_i | X = x_i) \]

Putting it all together:

\[ L_i = -\log(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}) \]
Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

\[ s = f(x_i; W) \]

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

Maximize probability of correct class

\[ L_i = -\log P(Y = y_i | X = x_i) \]

Putting it all together:

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

**Q:** What is the min / max possible loss \( L_i \)?

<table>
<thead>
<tr>
<th>cat</th>
<th>3.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>car</td>
<td>5.1</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
</tr>
</tbody>
</table>
Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as probabilities

\[ s = f(x_i; W) \]

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

Maximize probability of correct class

\[ L_i = -\log P(Y = y_i | X = x_i) \]

Putting it all together:

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

**Q:** What is the min / max possible loss \( L_i \)?

**A:** Min 0, max +infinity

### Examples

- cat: 3.2
- car: 5.1
- frog: -1.7
Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as probabilities

\[ s = f(x_i; W) \]

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

Maximize probability of correct class

\[ L_i = -\log P(Y = y_i | X = x_i) \]

Putting it all together:

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

**Q:** If all scores are small random values, what is the loss?

---

cat \hspace{1em} 3.2

car \hspace{1em} 5.1

greif \hspace{1em} -1.7
Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

\[ s = f(x_i; W) \]

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

Softmax function

Maximize probability of correct class

\[ L_i = -\log P(Y = y_i | X = x_i) \]

Putting it all together:

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

**Q:** If all scores are small random values, what is the loss?

**A:** \(-\log(1/C)\)

\[ \log(10) \approx 2.3 \]

cat | 3.2
---|---
car | 5.1
frog | -1.7
Recap: Three ways to think about linear classifiers

**Algebraic Viewpoint**

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

**Visual Viewpoint**

One template per class

**Geometric Viewpoint**

Hyperplanes cutting up space
Recap: Loss Functions quantify preferences

- We have some dataset of $(x, y)$
- We have a **score function**:
  $$s = f(x; W) = Wx$$
- We have a **loss function**:
  $$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{sj}}\right)$$  \textbf{Softmax}
  $$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{yi} + 1)$$  \textbf{SVM}
  $$L = \frac{1}{N} \sum_{i=1}^{N} L_i + R(W)$$  \textbf{Full loss}

Linear classifier
Recap: Loss Functions quantify preferences

- We have some dataset of \((x, y)\)
- We have a **score function**: 
  \[ s = f(x; W) = Wx \]
- We have a **loss function**:

**Softmax**

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

**SVM**

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

**Full loss**

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

Q: How do we find the best \(W\)?
Problem: Linear Classifiers aren’t that powerful

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

---

EECS 498-007 Lecture 2 - 96
One solution: Feature Transforms

Original space

Feature space

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

Original space

Feature space

Feature transform

Linear classifier in feature space

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

\[ \theta = \tan^{-1}(y/x) \]
One solution: Feature Transforms

Original space

Feature transform

Feature space

Linear classifier in feature space

Nonlinear classifier in original space!

Original space:

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

Feature space:
Deep learning attracts lots of attention.

- Google Trends
How the Human Brain learns

- In the human brain, a typical neuron collects signals from others through a host of fine structures called dendrites.
- The neuron sends out spikes of electrical activity through a long, thin stand known as an axon, which splits into thousands of branches.
- At the end of each branch, a structure called a synapse converts the activity from the axon into electrical effects that inhibit or excite activity in the connected neurons.
Our brains are made of Neurons

- Cell body
- Axon
- Dendrite
- Synapse
- Impulses carried toward cell body
- Impulses carried away from cell body
- Presynaptic terminal

Firing rate is a nonlinear function of inputs
A Neuron Model

• When a neuron receives excitatory input that is sufficiently large compared with its inhibitory input, it sends a spike of electrical activity down its axon. Learning occurs by changing the effectiveness of the synapses so that the influence of one neuron on another changes.

• We conduct these neural networks by first trying to deduce the essential features of neurons and their interconnections.

• We then typically program a computer to simulate these features.
A Simple Neuron

- An artificial neuron is a device with many inputs and one output.
- The neuron has two modes of operation;
  - the training mode and
  - the using mode.
A Simple Neuron (Cont.)

• In the training mode, the neuron can be trained to fire (or not), for particular input patterns.

• In the using mode, when a taught input pattern is detected at the input, its associated output becomes the current output. If the input pattern does not belong in the taught list of input patterns, the firing rule is used to determine whether to fire or not.

• The firing rule is an important concept in neural networks and accounts for their high flexibility. A firing rule determines how one calculates whether a neuron should fire for any input pattern. It relates to all the input patterns, not only the ones on which the node was trained on previously.
Part I: Introduction of Deep Learning

What people already knew in 1980s
Example Application

• Handwriting Digit Recognition
Handwriting Digit Recognition

**Input**

- $16 \times 16 = 256$
- Ink → 1
- No ink → 0

**Output**

- $x_1$
- $x_2$
- $\ldots$
- $x_{256}$

- Each dimension represents the confidence of a digit.

- The image is “2”
- Ink → 1
- No ink → 0

- 0.1 → is 1
- 0.7 → is 2
- 0.2 → is 0

Each dimension represents the confidence of a digit.
Example Application

• Handwriting Digit Recognition

In deep learning, the function $f$ is represented by neural network

$f: R^{256} \rightarrow R^{10}$
Element of Neural Network

**Neuron**  \( f : \mathbb{R}^K \rightarrow \mathbb{R} \)

\[
z = a_1 w_1 + a_2 w_2 + \cdots + a_K w_K + b
\]
Neural Network

Deep means many hidden layers
Example of Neural Network

Sigmoid Function

\[ \sigma(z) = \frac{1}{1 + e^{-z}} \]
Example of Neural Network

The diagram depicts a neural network with three layers. Each node (square or circle) represents a neuron, and the arrows indicate the connections between neurons. The numbers on the arrows represent the weights associated with those connections. The network takes an input (1), processes it through the layers, and outputs a result (2). The weights are as follows:

- Layer 1 to Layer 2: 0.98, 0.12
- Layer 2 to Layer 3: 0.86, 0.11
- Layer 3 to Output: 0.62, 0.83
Example of Neural Network

\[ f : \mathbb{R}^2 \rightarrow \mathbb{R}^2 \]

\[ f \left( \begin{bmatrix} 1 \\ -1 \end{bmatrix} \right) = \begin{bmatrix} 0.62 \\ 0.83 \end{bmatrix} \quad f \left( \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} 0.51 \\ 0.85 \end{bmatrix} \]

Different parameters define different function
Matrix Operation

\[
\sigma( \begin{bmatrix} 1 & -2 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} ) = \begin{bmatrix} 0.98 \\ 0.12 \end{bmatrix}
\]
Neural Network

\[
\sigma(W^1 x + b^1) \sigma(W^2 a^1 + b^2) \sigma(W^L a^{L-1} + b^L)
\]
Neural Network

\[
y = f(x) = \sigma(W^L + b^L) + \sigma(W^2 + b^2) + \cdots + \sigma(W^1 + b^1) + x
\]

Using parallel computing techniques to speed up matrix operation.
Softmax

- Softmax layer as the output layer

**Ordinary Layer**

\[ y_1 = \sigma(z_1) \]

\[ y_2 = \sigma(z_2) \]

\[ y_3 = \sigma(z_3) \]

In general, the output of network can be any value.

May not be easy to interpret
Softmax

- Softmax layer as the output layer

**Softmax Layer**

\[
\begin{align*}
\text{Probability:} \\
\quad &1 > y_i > 0 \\
\quad &\sum_i y_i = 1
\end{align*}
\]
How to set network parameters

\[ \theta = \{W^1, b^1, W^2, b^2, \ldots, W^L, b^L\} \]

Set the network parameters \( \theta \) such that ......

Input: \( y_2 \) has the maximum value

How to let the neural network achieve this

Input: \( 1 \times 16 = 256 \)

Ink \( \rightarrow 1 \)

No ink \( \rightarrow 0 \)
Training Data

• Preparing training data: images and their labels

Using the training data to find the network parameters.
Cost can be Euclidean distance or cross entropy of the network output and target.
Total Cost

For all training data ...

\[
\sum \left( \begin{array}{c}
\mathbf{y}_1 \\
\mathbf{y}_2 \\
\mathbf{y}_3 \\
\vdots \\
\mathbf{y}_R
\end{array} \right) = \begin{array}{c}
\mathbf{\hat{y}}_1 \\
\mathbf{\hat{y}}_2 \\
\mathbf{\hat{y}}_3 \\
\vdots \\
\mathbf{\hat{y}}_R
\end{array}
\]

Total Cost:

\[
C(\theta) = \sum_{r=1}^{R} L^r(\theta)
\]

How bad the network parameters \( \theta \) is on this task

Find the network parameters \( \theta^* \) that minimize this value
Gradient Descent

Assume there are only two parameters $w_1$ and $w_2$ in a network.

\[ \theta = \{w_1, w_2\} \]

Randomly pick a starting point $\theta^0$.

Compute the negative gradient at $\theta^0$

\[ -\nabla C(\theta^0) \]

Times the learning rate $\eta$

\[ -\eta \nabla C(\theta^0) \]
Gradient Descent

Eventually, we would reach a minima ..... 

Randomly pick a starting point $\theta^0$

Compute the negative gradient at $\theta^0$

$-\nabla C(\theta^0)$

Times the learning rate $\eta$

$-\eta \nabla C(\theta^0)$
Local Minima

• Gradient descent never guarantee global minima

Different initial point $\theta^0$

Reach different minima, so different results

Who is Afraid of Non-Convex Loss Functions?
http://videolectures.net/eml07_lecun_wia/
Besides local minima ......

Very slow at the plateau

Stuck at saddle point

Stuck at local minima

\[ \nabla C(\theta) \approx 0 \]

\[ \nabla C(\theta) = 0 \]

\[ \nabla C(\theta) = 0 \]
Mini-batch

Randomly initialize $\theta^0$

Pick the 1st batch
$C = C^1 + C^{31} + \cdots$
$\theta^1 \leftarrow \theta^0 - \eta \nabla C(\theta^0)$

Pick the 2nd batch
$C = C^2 + C^{16} + \cdots$
$\theta^2 \leftarrow \theta^1 - \eta \nabla C(\theta^1)$

Until all mini-batches have been picked

one epoch

Repeat the above process
Neural Networks

(Before) Linear score function:

\[ f = Wx \]

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

(Before) Linear score function:

$$f = Wx$$

(Now) 2-layer Neural Network

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

$$W_2 \in \mathbb{R}^{C \times H} \quad W_1 \in \mathbb{R}^{H \times D} \quad x \in \mathbb{R}^D$$

(In practice we will usually add a learnable bias at each layer as well)
Neural Networks

(Before) Linear score function:

$$f = Wx$$

(Now) 2-layer Neural Network

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

or 3-layer Neural Network

$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

$$W_3 \in \mathbb{R}^{C \times H_2}, \quad W_2 \in \mathbb{R}^{H_2 \times H_1}, \quad W_1 \in \mathbb{R}^{H_1 \times D}, \quad x \in \mathbb{R}^D$$

(In practice we will usually add a learnable bias at each layer as well)
Neural Networks

(Before) Linear score function:

\[ f = Wx \]

(Now) 2-layer Neural Network

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

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

(Before) Linear score function:

\[ f = Wx \]

(Now) 2-layer Neural Network

\[ f = W_2 \max(0, W_1x) \]

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 \(h_j\)

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

(Before) Linear score function:
\[ f = Wx \]

(Now) 2-layer Neural Network
\[ f = W_2 \max(0, W_1x) \]

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

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

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

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

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

Linear classifier: One template per class

(Before) Linear score function:

(Now) 2-layer Neural Network

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

Neural net: first layer is bank of templates; Second layer recombines templates

(Before) Linear score function:

(Now) 2-layer Neural Network

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

Can use different templates to cover multiple modes of a

(Before) Linear score function:

(Now) 2-layer Neural Network

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

“Distributed representation”: Most templates not interpretable!

(Before) Linear score function:

(Now) 2-layer Neural Network

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

Input: 3072
Output: 10

Depth = number of layers

Width: Size of each layer

\[ s = W_6 \max(0, 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”

This is called the **activation function** of the neural network

$$f = W_2 \max(0, W_1 x)$$
Activation Functions

2-layer Neural Network

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

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”

This function $f = W_2 \max(0, W_1 x)$ is called the activation function of the neural network.

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

A: We end up with a linear classifier!

$$s = W_2 W_1 x$$

$$W_3 = W_2 W_1 \in \mathbb{R}^{C \times H} \quad s = W_3 x$$
Activation Functions

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

\text{tanh}
\[ \tanh(x) \]

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

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

Maxout
\[ \max(w_1^Tx + b_1, w_2^Tx + b_2) \]

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

ReLU is a good default choice for most problems.
Neural Net in <20 lines!

```python
1 import numpy as np
2 from numpy.random import randn
3
4 N, Din, H, Dout = 64, 1000, 100, 10
5 x, y = randn(N, Din), randn(N, Dout)
6 w1, w2 = randn(Din, H), randn(H, Dout)
7
8 for t in range(10000):
9     h = 1.0 / (1.0 + np.exp(-x.dot(w1)))
10    y_pred = h.dot(w2)
11    loss = np.square(y_pred - y).sum()
12    dy_pred = 2.0 * (y_pred - y)
13    dw2 = h.T.dot(dy_pred)
14    dh = dy_pred.dot(w2.T)
15    dw1 = x.T.dot(dh * h * (1 - h))
16    w1 -= 1e-4 * dw1
17    w2 -= 1e-4 * dw2
```

Initialize weights and data

Compute loss (sigmoid activation, L2 loss)

Compute gradients

Stochastic Gradient Descent (SGD) step
Our brains are made of Neurons

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

Impulses carried toward cell body
Impulses carried away from cell body

Firing rate is a nonlinear function of inputs
**Biological Neuron**
- **dendrite**
- **cell body**
- **axon**
- **presynaptic terminal**

**Artificial Neuron**
- **input layer**
- **hidden layer 1**
- **hidden layer 2**
- **output layer**

The diagram illustrates the components of a biological neuron and an artificial neuron. The biological neuron consists of a dendrite, cell body, axon, and presynaptic terminal. The artificial neuron model includes layers for input, hidden, and output, with connections similar to the biological neuron's synaptic structure.

*Neuron image by Felipe Perucho is licensed under CC-BY 3.0*
Setting the number of layers and their sizes

More hidden units = more capacity
Summary

Feature transform + Linear classifier allows nonlinear decision boundaries

Original space

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

Feature space

Nonlinear classifier in original space!

Neural Networks as learnable feature transforms

Feature Extraction

10 numbers giving scores for classes

training

10 numbers giving scores for classes

training


Reproduced with permission.
Summary

From linear classifiers to fully-connected networks

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

Input: 3072

Hidden layer: 100

Output: 10

Linear classifier: One template per class

Neural networks: Many reusable templates
Backpropagation

• A network can have millions of parameters.
  • Backpropagation is the way to compute the gradients efficiently (not today)
  • Ref:

• Many toolkits can compute the gradients automatically

Ref:
Size of Training Data

• Rule of thumb:
  • the number of training examples should be at least five to ten times the number of weights of the network.

• Other rule:

\[ N > \frac{|W|}{(1 - a)} \]

- \(|W|\) = number of weights
- \(a\) = expected accuracy on test set
Training: Backprop algorithm

• The Backprop algorithm searches for weight values that minimize the total error of the network over the set of training examples (training set).

• Backprop consists of the repeated application of the following two passes:
  • **Forward pass**: in this step the network is activated on one example and the error of (each neuron of) the output layer is computed.
  • **Backward pass**: in this step the network error is used for updating the weights. Starting at the output layer, the error is propagated backwards through the network, layer by layer. This is done by recursively computing the local gradient of each neuron.
Back Propagation

- Back-propagation training algorithm

Network activation
Forward Step

Error propagation
Backward Step

- Backprop adjusts the weights of the NN in order to minimize the network total mean squared error.
Problem: How to compute gradients?

\[ s = f(x; W_1, W_2) = W_2 \max(0, W_1 x) \quad \text{Nonlinear score function} \]

\[ L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \quad \text{SVM Loss on predictions} \]

\[ R(W) = \sum_k W_k^2 \quad \text{Regularization} \]

\[ L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda R(W_1) + \lambda R(W_2) \quad \text{Total loss: data loss + regularization} \]

If we can compute \( \frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_2} \) then we can learn \( W_1 \) and \( W_2 \)
(Bad) Idea: Derive on paper $\nabla_W L$

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

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

$= \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1)$

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

$= \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1) + \lambda \sum_k W_k^2$

$\nabla_W L = \nabla_W \left( \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1) + \lambda \sum_k W_k^2 \right)$

**Problem:** Very tedious: Lots of matrix calculus, need lots of paper

**Problem:** What if we want to change loss? E.g. use softmax instead of SVM? Need to re-derive from scratch. Not modular!

**Problem:** Not feasible for very complex models!
Better Idea: Computational Graphs

\[ f = Wx \]

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

\[ R(W) \]
Backpropagation: Simple Example

\[ f(x, y, z) = (x + y)z \]
Backpropagation: Simple Example

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

e.g. \( x = -2, y = 5, z = -4 \)
Backpropagation: Simple Example

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

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

1. **Forward pass**: Compute outputs

\[ q = x + y \quad f = qz \]
Backpropagation: Simple Example

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

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

1. **Forward pass**: Compute outputs

\[ q = x + y \quad f = qz \]

2. **Backward pass**: Compute derivatives

Want:

\[ \frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} \]
Backpropagation: Simple Example

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

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

1. **Forward pass**: Compute outputs

\[ q = x + y \quad f = qz \]

2. **Backward pass**: Compute derivatives

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

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

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

1. **Forward pass**: Compute outputs

\[ q = x + y \quad f = qz \]

2. **Backward pass**: Compute derivatives

Want:

\[ \frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} \]
Backpropagation: Simple Example

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

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

1. **Forward pass**: Compute outputs

\[ q = x + y \quad f = qz \]

2. **Backward pass**: Compute derivatives

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

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

\[ \text{e.g. } x = -2, y = 5, z = -4 \]

1. **Forward pass**: Compute outputs

\[ q = x + y \]

\[ f = qz \]

2. **Backward pass**: Compute derivatives

Want:

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

\[ \frac{\partial f}{\partial z} = q \]
Backpropagation: Simple Example

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

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

1. **Forward pass**: Compute outputs

\[ q = x + y \quad f = qz \]

2. **Backward pass**: Compute derivatives

Want:

\[ \frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} \]
Backpropagation: Simple Example

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

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

1. **Forward pass**: Compute outputs
   \[ q = x + y \quad f = qz \]

2. **Backward pass**: Compute derivatives
   Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation: Simple Example

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

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

1. **Forward pass**: Compute outputs

\[ q = x + y \quad f = qz \]

2. **Backward pass**: Compute derivatives

Want:

\[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \]
Backpropagation: Simple Example

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

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

1. **Forward pass:** Compute outputs

\[ q = x + y \quad f = qz \]

2. **Backward pass:** Compute derivatives

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

**Chain Rule**

\[ \frac{\partial f}{\partial y} = \frac{\partial q}{\partial y} \frac{\partial f}{\partial q} \]
Backpropagation: Simple Example

\[ f(x, y, z) = (x + y)z \]
e.g. \( x = -2, y = 5, z = -4 \)

1. **Forward pass**: Compute outputs
\[ q = x + y \quad f = qz \]

2. **Backward pass**: Compute derivatives
Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)

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

\[ \frac{\partial q}{\partial y} = 1 \]

**Chain Rule**

\[ \text{Downstream Gradient} \quad \text{Local Gradient} \quad \text{Upstream Gradient} \]
Backpropagation: Simple Example

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

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

1. **Forward pass**: Compute outputs

\[ q = x + y \quad f = qz \]

2. **Backward pass**: Compute derivatives

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

**Chain Rule**

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

\[ \frac{\partial q}{\partial y} = 1 \]
Backpropagation: Simple Example

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

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

1. **Forward pass**: Compute outputs

   \[ q = x + y \quad f = qz \]

2. **Backward pass**: Compute derivatives

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

   \[ \frac{\partial f}{\partial x} = \frac{\partial q}{\partial x} \frac{\partial f}{\partial q} \quad \frac{\partial q}{\partial x} = 1 \]
Backpropagation: Simple Example

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

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

1. **Forward pass**: Compute outputs

\[ q = x + y \quad f = qz \]

2. **Backward pass**: Compute derivatives

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

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

\( \frac{\partial q}{\partial x} = 1 \)

Chain Rule

Downstream Gradient

Local Gradient

Upstream Gradient

EECS 498-007 Lecture 2 - 175
Part II: Why Deep?
Universality Theorem

Any continuous function $f$

$$f : R^N \rightarrow R^M$$

Can be realized by a network with one hidden layer

(given **enough** hidden neurons)

Why “Deep” neural network not “Fat” neural network?

Reference for the reason:
Fat + Short v.s. Thin + Tall

The same number of parameters

Which one is better?

Shallow

Deep
Both shallow (a) and deep (b) networks are universal, that is they can approximate arbitrarily well any continuous function of d variables on a compact domain.

We show that the approximation of functions with a compositional structure – such as $f(x_1, \cdots, x_d) = h_1(h_2 \cdots (h_j (h_{i_1}(x_1, x_2), h_{i_2}(x_3, x_4)), \cdots))$ – can be achieved with the same degree of accuracy by deep and shallow networks but that the number of parameters, the VC-dimension and the fat-shattering dimension are much smaller for the deep networks than for the shallow network with equivalent approximation accuracy.

It is intuitive that a hierarchical network matching the structure of a compositional function should be “better” at approximating it than a generic shallow network but universality of shallow networks makes the statement less than obvious. Our result makes clear that the intuition is indeed correct and provides quantitative bounds.

Why are compositional functions important? We argue that the basic properties of scalability and shift invariance in many natural signals such as images and text require compositional algorithms that can be well approximated by Deep Convolutional Networks. Of course, there are many situations that do not require shift invariant, scalable algorithms. For the many functions that are not compositional we do not expect any advantage of deep convolutional networks.
Recipe for Learning

Don’t forget!

Modify the Network
Better optimization Strategy

Preventing Overfitting

Does it do well on the training data? Yes No
Does it do well on the test data? Yes No

overfitting

Done!

Recipe for Learning

Neural networks re-visited
Neural networks: without the brain stuff

(Before) Linear score function: \( f = Wx \)
Neural networks: without the brain stuff

(Before) Linear score function: \( f = Wx \)

(Now) 2-layer Neural Network

\[
  f = W_2 \max(0, W_1 x)
\]
Neural networks: without the brain stuff

(Before) Linear score function:

$$f = Wx$$

(Now) 2-layer Neural Network

$$f = W_2 \max(0, W_1 x)$$
Neural networks: without the brain stuff

(Before) Linear score function:

\[ f = Wx \]

(Now) 2-layer Neural Network

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

- **Before**: Linear score function: \( f = Wx \)
- **Now**: 2-layer Neural Network
  - \( W_1 \) with dimensions 3072 \( \times \) 100
  - \( W_2 \) with dimensions 100 \( \times \) 10

Diagram with input images of plane, car, bird, cat, deer, dog, frog, horse, ship, and truck.
Neural networks: without the brain stuff

(Before) Linear score function:
\[ f = Wx \]

(Now) 2-layer Neural Network
\[ f = W_2 \max(0, W_1x) \]

or 3-layer Neural Network
\[ f = W_3 \max(0, W_2 \max(0, W_1x)) \]
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 networks: Architectures

“2-layer Neural Net”, or “1-hidden-layer Neural Net”

“3-layer Neural Net”, or “2-hidden-layer Neural Net”

“Fully-connected” layers
Next: Convolutional Neural Networks

Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1
Gradient-based learning applied to document recognition
[LeCun, Bottou, Bengio, Haffner 1998]

A bit of history:
A bit of history:
ImageNet Classification with Deep Convolutional Neural Networks
[Krizhevsky, Sutskever, Hinton, 2012]

“AlexNet”
Fast-forward to today: ConvNets are everywhere

self-driving cars

NVIDIA Tesla line
(these are the GPUs on rye01.stanford.edu)

Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.
Convolutional Neural Networks

(First without the brain stuff)
**Problem:** So far our classifiers don’t respect the spatial structure of images!
$$f(x, W) = Wx$$

Problem: So far our classifiers don’t respect the spatial structure of images!

Solution: Define new computational nodes that operate on images!

Input: 3072

Hidden layer: 100

Output: 10

$$f = W_2 \max(0, W_1 x)$$
Components of a Fully-Connected Network

Fully-Connected Layers

Activation Function
Components of a Convolutional Network

- **Fully-Connected Layers**
- **Activation Function**
- **Convolution Layers**
- **Pooling Layers**
- **Normalization**

Mathematically, the normalization can be expressed as:

\[
\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma^2_j} + \varepsilon}
\]
Components of a Convolutional Network

- Fully-Connected Layers
- Convolution Layers
- Activation Function
- Pooling Layers
- Normalization

\[ \hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}} \]
Fully-Connected Layer

32x32x3 image -> stretch to 3072 x 1

Input 3072

$Wx$

10 x 3072 weights

Output 10

1
Fully-Connected Layer

32x32x3 image -> stretch to 3072 x 1

Input

1
3072

$Wx$
10 x 3072 weights

Output

1
10

1 number: the result of taking a dot product between a row of $W$ and the input (a 3072-dimensional dot product)
Convolution Layer

$3 \times 32 \times 32$ image: preserve spatial structure
Convolution Layer

3x32x32 image

3x5x5 filter

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

Filters (almost) 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

3x32x32 image

3x5x5 filter

1 number:
the result of taking a dot product between the filter
and a small 3x5x5 chunk of the image
(i.e. 3*5*5 = 75-dimensional dot product + bias)

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

3x32x32 image

3x5x5 filter

32

3

convolve (slide) over all spatial locations

1x28x28 activation map

1

28

32
Convolution Layer

3x32x32 image

3x5x5 filter

Consider repeating with a second (green) filter:

Convolve (slide) over all spatial locations

two 1x28x28 activation map

32

32

3

EECS 498-007 Lecture 2 - 207
Convolution Layer

3x32x32 image

Consider 6 filters, each 3x5x5

6x3x5x5 filters

Convolucion Layer

Stack activations to get a 6x28x28 output image!

6 activation maps, each 1x28x28
Convolution Layer

3x32x32 image

Also 6-dim bias vector:

6 activation maps, each 1x28x28

Stack activations to get a 6x28x28 output image!
Convolution Layer

3x32x32 image

28x28 grid, at each point a 6-dim vector

Also 6-dim bias vector:

Stack activations to get a 6x28x28 output image!
Convolution Layer

2x3x32x32 Batch of images

Also 6-dim bias vector:

6x3x5x5 filters

2x6x28x28 Batch of outputs

EECS 498-007 Lecture 2 - 211
Convolution Layer

Batch of images

$N \times C_{in} \times H \times W$

$C_{in}$

$W$

$H$

$C_{out} \times C_{in} \times K_w \times K_h$ filters

Convolution Layer

Also $C_{out}$-dim bias vector:

Batch of outputs

$N \times C_{out} \times H' \times W'$
Stacking Convolutions

Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

Second hidden layer:
N x 10 x 26 x 26

Conv

W_1: 6x3x5x5
b_1: 5

Conv

W_2: 10x6x3x3
b_2: 10

Conv

W_3: 12x10x3x3
b_3: 12

....
Stacking Convolutions

Q: What happens if we stack two convolution layers?
A: We get another convolution!

Q: How to fix this?

(Recall $y = W_2 W_1 x$ is a linear classifier)
Stacking Convolutions

Input: $N \times 3 \times 32 \times 32$

First hidden layer:

$W_1: 6 \times 3 \times 5 \times 5$

$N \times 6 \times 28 \times 28$

$W_2: 10 \times 6 \times 3 \times 3$

$N \times 10 \times 26 \times 26$

$W_3: 12 \times 10 \times 3 \times 3$

Q: What happens if we stack two convolution layers?
A: We get another convolution!

Q: How to fix this?
A: Non-linearity

(Recall $y = W_2 W_1 x$ is a linear classifier)
What do convolutional filters learn?

Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

W_1: 6x3x5x5
b_1: 6

W_2: 10x6x3x3
b_2: 10

Second hidden layer:
N x 10 x 26 x 26

W_3: 12x10x3x3
b_3: 12

ReLU

Conv

Conv

Conv

ReLU
Visual illustration of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].
Preview

VGG-16 Conv1_1 → Low-level features → Mid-level features → High-level features → Linearily separable classifier

Retinal ganglion cell receptive fields → LGN and V1 simple cells

Visual stimulus

Complex cells: Response to light orientation and movement
Hypercomplex cells: response to movement with an end point

No response
Response (end point)
What do convolutional filters learn?

Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

Linear classifier: One template per class

 Conv ReLU

$W_1: 6 \times 3 \times 5 \times 5$

$b_1: 6$
What do convolutional filters learn?

Input: \(N \times 3 \times 32 \times 32\)

First hidden layer: \(N \times 6 \times 28 \times 28\)

Conv ReLU

\(W_1: 6 \times 3 \times 5 \times 5\)

\(b_1: 6\)

MLP: Bank of whole-image templates
What do convolutional filters learn?

Input: \( N \times 3 \times 32 \times 32 \)

First hidden layer: \( N \times 6 \times 28 \times 28 \)

First-layer conv filters: local image templates
(Often learns oriented edges, opposing colors)

\( W_1: 6\times3\times5\times5 \)
\( b_1: 6 \)

AlexNet: 64 filters, each 3x11x11
A closer look at spatial dimensions

Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

Conv ReLU

$W_1: 6 \times 3 \times 5 \times 5$

$b_1: 6$
A closer look at spatial dimensions

Input: 7x7
Filter: 3x3
A closer look at spatial dimensions

Input: 7x7
Filter: 3x3
A closer look at spatial dimensions

Input: 7x7
Filter: 3x3
A closer look at spatial dimensions

Input: 7x7
Filter: 3x3
A closer look at spatial dimensions

Input: 7x7
Filter: 3x3
Output: 5x5
A closer look at spatial dimensions

Input: 7x7
Filter: 3x3
Output: 5x5

In general:
Input: W
Filter: K
Output: W – K + 1

Problem: Feature maps “shrink” with each layer!
A closer look at spatial dimensions

Input: 7x7
Filter: 3x3
Output: 5x5

In general:
Input: W
Filter: K
Output: W – K + 1

Problem:
Feature maps “shrink” with each layer!

Solution: padding
Add zeros around the input
A closer look at spatial dimensions

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

Input: 7x7  
Filter: 3x3  
Output: 5x5

In general:  
Input: $W$  
Filter: $K$  
Padding: $P$  
Output: $W - K + 1 + 2P$

Very common:  
Set $P = \frac{(K - 1)}{2}$ to make output have same size as input!
Receptive Fields

For convolution with kernel size $K$, each element in the output depends on a $K \times K$ receptive field in the input.
Receptive Fields

Each successive convolution adds $K - 1$ to the receptive field size.

With $L$ layers the receptive field size is $1 + L \times (K - 1)$.

Be careful – “receptive field in the input” vs “receptive field in the previous layer.”

Hopefully clear from context!
Receptive Fields

Each successive convolution adds $K - 1$ to the receptive field size.
With $L$ layers the receptive field size is $1 + L \times (K - 1)$.

Problem: For large images we need many layers for each output to “see” the whole image.
Receptive Fields

Each successive convolution adds $K - 1$ to the receptive field size.
With $L$ layers the receptive field size is $1 + L \times (K - 1)$.

Problem: For large images we need many layers for each output to “see” the whole image.

Solution: Downsample inside the network.
Strided Convolution

Input: 7x7
Filter: 3x3
Stride: 2
Strided Convolution

Input: 7x7
Filter: 3x3
Stride: 2
Strided Convolution

Input: 7x7
Filter: 3x3
Stride: 2
Output: 3x3
Strided Convolution

Input: 7x7
Filter: 3x3
Stride: 2
Output: 3x3

In general:
Input: W
Filter: K
Padding: P
Stride: S
Output: \((W - K + 2P) / S + 1\)
Convolution Example

Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

Output volume size: ?
Convolution Example

Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
10 x 32 x 32
Convolution Example

Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

Output volume size: 10 x 32 x 32
Number of learnable parameters: ?
Convolution Example

Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

Output volume size: 10 x 32 x 32
Number of learnable parameters: 760
Parameters per filter: $3 \times 5 \times 5 + 1$ (for bias) = 76
10 filters, so total is $10 \times 76 = 760$
Convolution Example

Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

Output volume size: 10 x 32 x 32
Number of learnable parameters: 760
Number of multiply-add operations: ?
Convolution Example

Input volume: \(3 \times 32 \times 32\)
10 \(5\times5\) filters with stride 1, pad 2

Output volume size: \(10 \times 32 \times 32\)
Number of learnable parameters: 760
Number of multiply-add operations: \(768,000\)
\(10 \times 32 \times 32 = 10,240\) outputs; each output is the inner product of two \(3\times5\times5\) tensors (75 elems); total = \(75 \times 10240 = 768K\)
Example: 1x1 Convolution

1x1 CONV with 32 filters
(each filter has size 1x1x64, and performs a 64-dimensional dot product)
Example: 1x1 Convolution

Stacking 1x1 conv layers gives MLP operating on each input position

Convolution Summary

Input: \( C_{\text{in}} \times H \times W \)

Hyperparameters:
- Kernel size: \( K_H \times K_W \)
- Number filters: \( C_{\text{out}} \)
- Padding: \( P \)
- Stride: \( S \)

Weight matrix: \( C_{\text{out}} \times C_{\text{in}} \times K_H \times K_W \)
giving \( C_{\text{out}} \) filters of size \( C_{\text{in}} \times K_H \times K_W \)

Bias vector: \( C_{\text{out}} \)

Output size: \( C_{\text{out}} \times H' \times W' \) where:
- \( H' = (H - K + 2P) / S + 1 \)
- \( W' = (W - K + 2P) / S + 1 \)
Convolution Summary

Input: $C_{in} \times H \times W$

Hyperparameters:
- Kernel size: $K_H \times K_W$
- Number filters: $C_{out}$
- Padding: $P$
- Stride: $S$

Weight matrix: $C_{out} \times C_{in} \times K_H \times K_W$
giving $C_{out}$ filters of size $C_{in} \times K_H \times K_W$

Bias vector: $C_{out}$

Output size: $C_{out} \times H' \times W'$ where:
- $H' = (H - K + 2P) / S + 1$
- $W' = (W - K + 2P) / S + 1$

Common settings:
$K_H = K_W$ (Small square filters)
$P = (K - 1) / 2$ ("Same" padding)
$C_{in}, C_{out} = 32, 64, 128, 256$ (powers of 2)
$K = 3, P = 1, S = 1$ (3x3 conv)
$K = 5, P = 2, S = 1$ (5x5 conv)
$K = 1, P = 0, S = 1$ (1x1 conv)
$K = 3, P = 1, S = 2$ (Downsample by 2)
Components of a Convolutional Network

- Convolution Layers
- Pooling Layers
- Fully-Connected Layers
- Activation Function
- Normalization

\[ \hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}} \]
Pooling Layers: Another way to downsample

Hyperparameters:
- Kernel Size
- Stride
- Pooling function
Max Pooling

Single depth slice

Max pooling with 2x2 kernel size and stride 2

Introduces **invariance** to small spatial shifts
No learnable parameters!
Pooling Summary

**Input**: C x H x W

**Hyperparameters**:
- Kernel size: K
- Stride: S
- Pooling function (max, avg)

**Output**: C x H’ x W’ where
- H’ = (H – K) / S + 1
- W’ = (W – K) / S + 1

**Learnable parameters**: None!

*Common settings:*
- max, K = 2, S = 2
- max, K = 3, S = 2 (AlexNet)
Components of a Convolutional Network

- Fully-Connected Layers
- Activation Function
- Convolution Layers
- Pooling Layers
- Normalization

Convolution Layers:

Pooling Layers:

\[
\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma^2_j + \varepsilon}}
\]
Convolutional Networks

Classic architecture: [Conv, ReLU, Pool] x N, flatten, [FC, ReLU] x N, FC

Example: LeNet-5

Example: LeNet-5

<table>
<thead>
<tr>
<th>Layer</th>
<th>Output Size</th>
<th>Weight Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>1 x 28 x 28</td>
<td></td>
</tr>
</tbody>
</table>

Example: LeNet-5*

<table>
<thead>
<tr>
<th>Layer</th>
<th>Output Size</th>
<th>Weight Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>1 x 28 x 28</td>
<td></td>
</tr>
<tr>
<td>Conv ((C_{out}=20^*, K=5, P=2, S=1))</td>
<td>20 x 28 x 28</td>
<td>20 x 1 x 5 x 5</td>
</tr>
<tr>
<td>ReLU**</td>
<td>20 x 28 x 28</td>
<td></td>
</tr>
</tbody>
</table>

* Original paper: \(C_{out} = 6\)
** Original paper: sigmoid

## Example: LeNet-5

<table>
<thead>
<tr>
<th>Layer</th>
<th>Output Size</th>
<th>Weight Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>1 x 28 x 28</td>
<td></td>
</tr>
<tr>
<td>Conv ($C_{out}=20$, $K=5$, $P=2$, $S=1$)</td>
<td>20 x 28 x 28</td>
<td>20 x 1 x 5 x 5</td>
</tr>
<tr>
<td>ReLU</td>
<td>20 x 28 x 28</td>
<td></td>
</tr>
<tr>
<td>MaxPool($K=2$, $S=2$)*</td>
<td>20 x 14 x 14</td>
<td></td>
</tr>
</tbody>
</table>

* 2x2 strided convolution

---


---

EECS 498-007 Lecture 2 - 265
Example: LeNet-5

<table>
<thead>
<tr>
<th>Layer</th>
<th>Output Size</th>
<th>Weight Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>1 x 28 x 28</td>
<td></td>
</tr>
<tr>
<td>Conv (C_{out}=20, K=5, P=2, S=1)</td>
<td>20 x 28 x 28</td>
<td>20 x 1 x 5 x 5</td>
</tr>
<tr>
<td>ReLU</td>
<td>20 x 28 x 28</td>
<td></td>
</tr>
<tr>
<td>MaxPool(K=2, S=2)</td>
<td>20 x 14 x 14</td>
<td></td>
</tr>
<tr>
<td>Conv (C_{out}=50*,K=5, P=2, S=1)</td>
<td>50 x 14 x 14</td>
<td>50 x 20 x 5 x 5</td>
</tr>
<tr>
<td>ReLU**</td>
<td>50 x 14 x 14</td>
<td></td>
</tr>
</tbody>
</table>

* Original paper: C_{out} = 16, grouped convolutions
** Original paper: sigmoid

Example: LeNet-5

<table>
<thead>
<tr>
<th>Layer</th>
<th>Output Size</th>
<th>Weight Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>1 x 28 x 28</td>
<td></td>
</tr>
<tr>
<td>Conv ($C_{out}=20$, $K=5$, $P=2$, $S=1$)</td>
<td>20 x 28 x 28</td>
<td>20 x 1 x 5 x 5</td>
</tr>
<tr>
<td>ReLU</td>
<td>20 x 28 x 28</td>
<td></td>
</tr>
<tr>
<td>MaxPool($K=2$, $S=2$)</td>
<td>20 x 14 x 14</td>
<td></td>
</tr>
<tr>
<td>Conv ($C_{out}=50$, $K=5$, $P=2$, $S=1$)</td>
<td>50 x 14 x 14</td>
<td>50 x 20 x 5 x 5</td>
</tr>
<tr>
<td>ReLU</td>
<td>50 x 14 x 14</td>
<td></td>
</tr>
<tr>
<td>MaxPool($K=2$, $S=2$)*</td>
<td>50 x 7 x 7</td>
<td></td>
</tr>
</tbody>
</table>

*C 2x2 strided convolution

Example: LeNet-5

<table>
<thead>
<tr>
<th>Layer</th>
<th>Output Size</th>
<th>Weight Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>1 x 28 x 28</td>
<td></td>
</tr>
<tr>
<td>Conv ( C_{out}=20, K=5, P=2, S=1 )</td>
<td>20 x 28 x 28</td>
<td>20 x 1 x 5 x 5</td>
</tr>
<tr>
<td>ReLU</td>
<td>20 x 28 x 28</td>
<td></td>
</tr>
<tr>
<td>MaxPool(K=2, S=2)</td>
<td>20 x 14 x 14</td>
<td></td>
</tr>
<tr>
<td>Conv ( C_{out}=50, K=5, P=2, S=1 )</td>
<td>50 x 14 x 14</td>
<td>50 x 20 x 5 x 5</td>
</tr>
<tr>
<td>ReLU</td>
<td>50 x 14 x 14</td>
<td></td>
</tr>
<tr>
<td>MaxPool(K=2, S=2)</td>
<td>50 x 7 x 7</td>
<td></td>
</tr>
<tr>
<td>Flatten</td>
<td>2450</td>
<td></td>
</tr>
</tbody>
</table>

## Example: LeNet-5

<table>
<thead>
<tr>
<th>Layer</th>
<th>Output Size</th>
<th>Weight Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>1 x 28 x 28</td>
<td></td>
</tr>
<tr>
<td>Conv ($C_{out}=20$, $K=5$, $P=2$, $S=1$)</td>
<td>20 x 28 x 28</td>
<td>20 x 1 x 5 x 5</td>
</tr>
<tr>
<td>ReLU</td>
<td>20 x 28 x 28</td>
<td></td>
</tr>
<tr>
<td>MaxPool($K=2$, $S=2$)</td>
<td>20 x 14 x 14</td>
<td></td>
</tr>
<tr>
<td>Conv ($C_{out}=50$, $K=5$, $P=2$, $S=1$)</td>
<td>50 x 14 x 14</td>
<td>50 x 20 x 5 x 5</td>
</tr>
<tr>
<td>ReLU</td>
<td>50 x 14 x 14</td>
<td></td>
</tr>
<tr>
<td>MaxPool($K=2$, $S=2$)</td>
<td>50 x 7 x 7</td>
<td></td>
</tr>
<tr>
<td>Flatten</td>
<td>2450</td>
<td></td>
</tr>
<tr>
<td>Linear (2450 -&gt; 500)</td>
<td>500</td>
<td>2450 x 500</td>
</tr>
<tr>
<td>ReLU*</td>
<td>500</td>
<td></td>
</tr>
</tbody>
</table>

* *Original paper has different 1x1 convolutions, sigmoid non-linearities*

Example: LeNet-5*

<table>
<thead>
<tr>
<th>Layer</th>
<th>Output Size</th>
<th>Weight Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>1 x 28 x 28</td>
<td></td>
</tr>
<tr>
<td>Conv ((C_{\text{out}}=20, K=5, P=2, S=1))</td>
<td>20 x 28 x 28</td>
<td>20 x 1 x 5 x 5</td>
</tr>
<tr>
<td>ReLU</td>
<td>20 x 28 x 28</td>
<td></td>
</tr>
<tr>
<td>MaxPool((K=2, S=2))</td>
<td>20 x 14 x 14</td>
<td></td>
</tr>
<tr>
<td>Conv ((C_{\text{out}}=50, K=5, P=2, S=1))</td>
<td>50 x 14 x 14</td>
<td>50 x 20 x 5 x 5</td>
</tr>
<tr>
<td>ReLU</td>
<td>50 x 14 x 14</td>
<td></td>
</tr>
<tr>
<td>MaxPool((K=2, S=2))</td>
<td>50 x 7 x 7</td>
<td></td>
</tr>
<tr>
<td>Flatten</td>
<td>2450</td>
<td></td>
</tr>
<tr>
<td>Linear ((2450 -&gt; 500))</td>
<td>500</td>
<td>2450 x 500</td>
</tr>
<tr>
<td>ReLU</td>
<td>500</td>
<td></td>
</tr>
<tr>
<td>Linear ((500 -&gt; 10))*</td>
<td>10</td>
<td>500 x 10</td>
</tr>
</tbody>
</table>

* Original paper uses RBF (radial basis function) kernels instead of a softmax

**Example: LeNet-5**

<table>
<thead>
<tr>
<th>Layer</th>
<th>Output Size</th>
<th>Weight Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>1 x 28 x 28</td>
<td></td>
</tr>
<tr>
<td>Conv ($C_{out}=20$, $K=5$, $P=2$, $S=1$)</td>
<td>20 x 28 x 28</td>
<td>20 x 1 x 5 x 5</td>
</tr>
<tr>
<td>ReLU</td>
<td>20 x 28 x 28</td>
<td></td>
</tr>
<tr>
<td>MaxPool($K=2$, $S=2$)</td>
<td>20 x 14 x 14</td>
<td></td>
</tr>
<tr>
<td>Conv ($C_{out}=50$, $K=5$, $P=2$, $S=1$)</td>
<td>50 x 14 x 14</td>
<td>50 x 20 x 5 x 5</td>
</tr>
<tr>
<td>ReLU</td>
<td>50 x 14 x 14</td>
<td></td>
</tr>
<tr>
<td>MaxPool($K=2$, $S=2$)</td>
<td>50 x 7 x 7</td>
<td></td>
</tr>
<tr>
<td>Flatten</td>
<td>2450</td>
<td></td>
</tr>
<tr>
<td>Linear (2450 -&gt; 500)</td>
<td>500</td>
<td>2450 x 500</td>
</tr>
<tr>
<td>ReLU</td>
<td>500</td>
<td></td>
</tr>
<tr>
<td>Linear (500 -&gt; 10)</td>
<td>10</td>
<td>500 x 10</td>
</tr>
</tbody>
</table>

As we go through the network:

Spatial size **decreases** (using pooling or strided conv)

Number of channels **increases** (total “volume” is preserved!)

Problem: Deep Networks very hard to train!
Components of a Convolutional Network

- Fully-Connected Layers
- Convolution Layers
- Pooling Layers
- Activation Function
- Normalization

**Equation for Normalization:**

\[ \hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma^2_j + \varepsilon}} \]
Components of a Convolutional Network

Convolution Layers

Pooling Layers

Fully-Connected Layers

Activation Function

Normalization

\[ \hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma^2_j + \varepsilon}} \]
Components of a Convolutional Network

- **Convolution Layers**
  - Most computationally expensive!

- **Pooling Layers**
  - Illustration showing downsampling from 224x224x64 to 112x112x64

- **Fully-Connected Layers**

- **Activation Function**
  - Graph showing an activation function

- **Normalization**
  - Formula: $\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$
Summary: Components of a Convolutional Network

Convolution Layers

Pooling Layers

Fully-Connected Layers

Activation Function

Normalization

\[
\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}
\]

EECS 498-007 Lecture 2 - 276
Summary: Components of a Convolutional Network

Problem: What is the right way to combine all these components?
Convolutional neural networks++

- Training and optimization
- More regularization (dropout, ...)
- Convolutional neural networks
- Pooling
- Batch normalization
- CNN architectures