CMP784
DEEP LEARNING

Lecture #2 – Machine Learning Overview

Aykut Erdem // Hacettepe University // Spring 2018
Previously on CMP784

• what is deep learning
• a brief history of deep learning
• compositionality
• end-to-end learning
• distributed representations
Lecture overview

• what is learning?
• types of machine learning problems
• image classification
• linear regression
• generalization
• cross-validation
• maximum likelihood estimation

• Disclaimer: Much of the material and slides for this lecture were borrowed from
  — Bernhard Schölkopf’s MLSS 2017 lecture,
  — Tommi Jaakkola’s 6.867 class,
  — Fei-Fei Li, Andrej Karpathy and Justin Johnson’s CS231n class
Good news, everyone!

• Practical 1 will be out next week
  • Multi-layer perceptrons
  • Word embeddings
  • Due March 15, 2018
  • Install TensorFlow to your machine to get ready!
More good news, everyone!

• Paper presentations will start in two weeks!
• Quizzes will also start at March 8th, 2018
  – 1 or 2 short answer questions about the paper (10 mins long)
  – You are also responsible for the corresponding ML subreddit discussions (if any)
    reddit.com/r/MachineLearning
More on paper presentations

• Choose your paper
  – 9 papers
  – 48 students

• Choose your role
  – **Presenters** (2 students)
  – **Code walker** (1 student)
  – **Demonstrator(s)** (1 or 2 students)
Presenters (2 students) // Paper presentations

• Discuss your slides with me 3-4 days prior to your presentation
• Submit your final slides by the night before the class
• Each presentation should be roughly 20 mins long
• You are allowed to reuse the material already exist on the web
Code walker (1 student) // Paper presentations

- Walk through the implementation of the paper
- It should be roughly 5-10 mins long
- Try your best to find workarounds or fixes which are not discussed in the paper
Demos (1-2 students) // Paper presentations

- Conduct experiments demonstrating how the method works in practice
- It should be roughly 5-10 mins long
- Try your best to fool the models
What is learning?
Two definitions of learning

• “Learning is the acquisition of knowledge about the world.”
  Kupfermann (1985)

• “Learning is an adaptive change in behavior caused by experience.”
  Shepherd (1988)
Empirical Inference

• Drawing conclusions from empirical data (observations, measurements)

• Example 1: scientific inference
Empirical Inference

• Drawing conclusions from empirical data (observations, measurements)

• Example 1: scientific inference

\[ y = a \times x \]
Empirical Inference

• Drawing conclusions from empirical data (observations, measurements)

• Example 1: scientific inference

\[ y = \sum_{i} a_i k(x, x_i) + b \]
Empirical Inference

• Drawing conclusions from empirical data (observations, measurements)

• Example 1: scientific inference

\[ y = \sum_i a_i k(x, x_i) + b \]
Empirical Inference

• Example 2: perception
The choice of representation may determine whether the learning task is very easy or very difficult!
Generalization

• observe $1, 2, 4, 7, \ldots$
• What’s next?

- 1, 2, 4, 7, 11, 16, $\ldots$: $a_{n+1} = a_n + n$ ("lazy caterer’s sequence")
- 1, 2, 4, 7, 12, 20, $\ldots$: $a_{n+2} = a_{n+1} + a_n + 1$
- 1, 2, 4, 7, 13, 24, $\ldots$: "Tribonacci"-sequence
- 1, 2, 4, 7, 14, 28, $\ldots$: divisors of 28
- 1, 2, 4, 7, 1, 1, 5, $\ldots$: decimal expansions of $\pi = 3.14159\ldots$ and $e = 2.718\ldots$ interleaved (thanks to O. Bousquet)

• don’t need $e$: 1247 appears at position 16992 in $\pi$
• The On-Line Encyclopedia of Integer Sequences: > 600 hits…
Generalization, II

• Question: which continuation is correct ("generalizes")?

• Answer? There’s no way to tell ("induction problem")

• Question of statistical learning theory: how to come up with a law that generalizes ("demarcation problem")
Types of ML problems
Types of machine learning problems

Based on the information available:

• Supervised learning
• Unsupervised learning
• Semi-supervised learning
• Reinforcement learning
Supervised learning

• **Input:** \( \{(x, y)\} \)

• **Task:** Predict target \( y \) from input \( x \)
  – **Classification:** Discrete output
  – **Regression:** Real-valued output

![Image classification](cat.png)

![Face detection](face.png)
Unsupervised learning

• **Input:** \( \{ x \} \)

• **Task:** Reveal structure in the observed data
  – **Clustering:** Partition data into groups
  – **Feature extraction:** Learning meaningful features automatically
  – **Dimensionality reduction:** Learning a lower-dimensional representation of input
Semi-supervised learning

• **Input:**
  - Few labeled examples $\{(x, y)\}$
  - Many unlabeled examples $\{x\}$

• **Task:** Predict target $y$ from input $x$
  - **Classification:** Discrete output
  - **Regression:** Real-valued output

Try to improve predictions based on examples by making use of the additional “unlabeled” examples

Reinforcement learning

• Input: Interaction with an environment; the agent receives a numerical reward signal

• Task: A way of behaving that is very rewarding in the long run

• Goal is to estimate and maximize the long-term cumulative reward

Adapted from Doina Precup
Types of machine learning problems

“If intelligence was a cake, unsupervised learning would be the cake, supervised learning would be the icing on the cake, and reinforcement learning would be the cherry on the cake. We know how to make the icing and the cherry, but we don't know how to make the cake.”

– Yann LeCun
NIPS 2016 Keynote

- **“Pure” Reinforcement Learning (cherry)**
  - The machine predicts a scalar reward given once in a while.
  - **A few bits for some samples**

- **Supervised Learning (icing)**
  - The machine predicts a category or a few numbers for each input.
  - Predicting human-supplied data
  - **10→10,000 bits per sample**

- **Unsupervised/Predictive Learning (cake)**
  - The machine predicts any part of its input for any observed part.
  - Predicts future frames in videos
  - **Millions of bits per sample**

(Yes, I know, this picture is slightly offensive to RL folks. But I’ll make it up)
Image classification

— non-parametric vs. parametric models
— nearest neighbor classifier
— hyperparameter
— cross-validation
Image Classification: a core task in Computer Vision

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

---
cat
The problem: semantic gap

Images are represented as 3D arrays of numbers, with integers between [0, 255].

e.g.
300 x 100 x 3

(3 for 3 color channels RGB)
Challenges: Viewpoint Variation
Challenges: Illumination
Challenges: Deformation
Challenges: Occlusion
Challenges: Background clutter
Challenges: Intraclass variation
An image classifier

```
def predict(image):
    # ????
    return class_label
```

Unlike e.g. sorting a list of numbers,

**no obvious way** to hard-code the algorithm for recognizing a cat, or other classes.
Data-driven approach:
1. Collect a dataset of images and labels
2. Use Machine Learning to train an image classifier
3. Evaluate the classifier on a withheld set of test images

```python
def train(train_images, train_labels):
    # build a model for images -> labels...
    return model

def predict(model, test_images):
    # predict test_labels using the model...
    return test_labels
```

Example training set
First classifier: Nearest Neighbor Classifier

def train(train_images, train_labels):
    # build a model for images -> labels...
    return model

def predict(model, test_images):
    # predict test_labels using the model...
    return test_labels

Remember all training images and their labels

Predict the label of the most similar training image
Example dataset: **CIFAR-10**

10 labels

50,000 training images, each image is tiny: 32x32

10,000 test images.
Example dataset: CIFAR-10
10 labels
50,000 training images
10,000 test images.

For every test image (first column), examples of nearest neighbors in rows.
How do we compare the images? What is the **distance metric**?

**L1 distance**: \( d_1(I_1, I_2) = \sum_p |I^p_1 - I^p_2| \)
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N ""
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for ""
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros((num_test, self.ytr.dtype))

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred

Nearest Neighbor classifier
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred

Nearest Neighbor classifier

remember the training data
for every test image:
- find nearest train image with L1 distance
- predict the label of nearest training image
Nearest Neighbor classifier

Q: how does the classification speed depend on the size of the training data?
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros((num_test, ), dtype = self.ytr.dtype)

        # loop over all test rows
        for i in range(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred

Q: how does the classification speed depend on the size of the training data? **linearly :(**

This is **backwards**:  
- test time performance is usually much more important in practice.  
- CNNs flip this: expensive training, cheap test evaluation
Aside: Approximate Nearest Neighbor

find approximate nearest neighbors quickly
The choice of distance is a **hyperparameter**

common choices:

<table>
<thead>
<tr>
<th>L1 (Manhattan) distance</th>
<th>L2 (Euclidean) distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>(d_1(I_1, I_2) = \sum_p</td>
<td>I_1^p - I_2^p</td>
</tr>
</tbody>
</table>
k-Nearest Neighbor
find the k nearest images, have them vote on the label

Example dataset: **CIFAR-10**

- 10 labels
- **50,000** training images
- **10,000** test images.

For every test image (first column), examples of nearest neighbors in rows...
Q: what is the accuracy of the nearest neighbor classifier on the training data, when using the Euclidean distance?
Q2: what is the accuracy of the $k$-nearest neighbor classifier on the training data?
What is the best **distance** to use?
What is the best value of **k** to use?

i.e. how do we set the **hyperparameters**?
What is the best distance to use?
What is the best value of k to use?

i.e. how do we set the hyperparameters?

Very problem-dependent.
Must try them all out and see what works best.
Try out what hyperparameters work best on test set.
Trying out what hyperparameters work best on test set:
Very bad idea. The test set is a proxy for the generalization performance!
Use only VERY SPARINGLY, at the end.
Validation data

use to tune hyperparameters
Cross-validation cycle through the choice of which fold is the validation fold, average results.
Example of 5-fold cross-validation for the value of $k$.

Each point: single outcome.

The line goes through the mean, bars indicated standard deviation

(Seems that $k \sim 7$ works the best for this data)
k-Nearest Neighbor on images never used.

- terrible performance at test time
- distance metrics on level of whole images can be very unintuitive

(all 3 images have same L2 distance to the one on the left)
The learning problem
— linear classification
— hypothesis class, estimation algorithm
— loss and estimation criterion
— sampling, empirical and expected losses
The Learning Problem

- **Steps**
  - entertain a (biased) set of possibilities (hypothesis class)
  - adjust predictions based on available examples (estimation)
  - rethink the set of possibilities (model selection)

- **Principles of learning are “universal”**
  - society (e.g., scientific community)
  - animal (e.g., human)
  - machine
Hypothesis class

• Representation: examples are binary vectors of length $d = 64$

\[ \mathbf{x} = [111 \ldots 0001]^T = \]

and labels $y \in \{-1, 1\}$ ("no", "yes")

• The mapping from examples to labels is a "linear classifier"

\[ \hat{y} = \text{sign} (\theta \cdot \mathbf{x}) = \text{sign} (\theta_1 x_1 + \ldots + \theta_d x_d) \]

where $\theta$ is a vector of parameters we have to learn from examples.
Linear classifier/experts

- We can understand the simple linear classifier

\[ \hat{y} = \text{sign} ( \theta \cdot x ) = \text{sign} ( \theta_1 x_1 + \ldots + \theta_d x_d ) \]

as a way of combining expert opinion (in this case simple binary features)
Estimation

\[ \mathbf{x} \]
\[
\begin{array}{c}
0111110011100100000010000000100111110111011111001110111110001 \\
000111110000001100000011100000110011111011111011111100111110000011 \\
111111000000110000011000111100110000011110000111110001101111111
\end{array}
\]
\[ y \]
\[
\begin{array}{c}
+1 \\
+1 \\
-1
\end{array}
\]

... ...

• How do we adjust the parameters \( \theta \) based on the labeled examples?

\[ \hat{y} = \text{sign} \left( \theta \cdot \mathbf{x} \right) \]

For example, we can simply refine/update the parameters whenever we make a mistake (perceptron algorithm):

\[ \theta_i \leftarrow \theta_i + y x_i, \quad i = 1, \ldots, d \quad \text{if prediction was wrong} \]
Evaluation

• Does the simple mistake driven algorithm work?

(average classification error as a function of the number of examples and labels seen so far)
Illustration of Convergence

- Convergence of the perceptron learning algorithm
Linear classifier: image classification

\[ f(x, W) \]

Image parameters

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

10 numbers, indicating class scores
Linear classifier: image classification

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

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

10 numbers, indicating class scores
Linear classifier: image classification

$[32 \times 32 \times 3]$ array of numbers 0...1

parameters, or “weights”

10 numbers, indicating class scores

$f(x, W) = Wx$

$10 \times 1$  

$10 \times 3072$  

$3072 \times 1$
Linear classifier: image classification

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

- \([32 \times 32 \times 3]\) array of numbers 0...1
- \(10 \times 1\) parameters, or “weights”
- \(10 \times 3072\)
- \(3072 \times 1\)
- \((+b)\) \(10 \times 1\)
- 10 numbers, indicating class scores
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

<table>
<thead>
<tr>
<th>W</th>
<th>b</th>
<th>f(x_i; W, b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2 -0.5 0.1 2.0</td>
<td>1.1</td>
<td>-96.8</td>
</tr>
<tr>
<td>1.5 1.3 2.1 0.0</td>
<td>3.2</td>
<td>437.9</td>
</tr>
<tr>
<td>0 0.25 0.2 -0.3</td>
<td>-1.2</td>
<td>61.95</td>
</tr>
</tbody>
</table>

Input image:

stretch pixels into single column

\[ f(x_i; W, b) = W x_i + b \]
Q: what does the linear classifier do, in English?

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

Example trained weights of a linear classifier trained on CIFAR-10:

$$f(x_i, W, b) = Wx_i + b$$
Interpreting a Linear Classifier

\[ f(x_i, W, b) = Wx_i + b \]

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

- The simple linear classifier cannot solve all the problems (e.g., XOR)

- Can we rethink the approach to do even better?

- We can, for example, add "polynomial experts"

\[
\hat{y} = \text{sign} \left( \theta_1 x_1 + \ldots + \theta_d x_d + \theta_{12} x_1 x_2 + \ldots \right)
\]
Model selection (cont’d)

linear

2nd order polynomial

4th order polynomial

8th order polynomial

Tommi Jaakkola, MIT CSAIL
Review: The learning problem

Image Classification

• **Hypothesis class:** we consider some **restricted** set \( \mathcal{F} \) of mappings
  \( f : X \rightarrow L \) from images to labels

• **Estimation:** on the basis of a training set of examples and labels,
  \( \{(x_1, y_1), \ldots, (x_n, y_n)\} \), we find an estimate \( \hat{f} \in \mathcal{F} \)

• **Evaluation:** we measure how well \( \hat{f} \) **generalizes** to yet unseen examples,
  i.e., whether \( \hat{f}(x_{new}) \) agrees with \( y_{new} \)
Hypothesis and estimation

• We used a simple linear classifier, a parameterized mapping \( f(x; \theta) \) from images \( \mathcal{X} \) to labels \( \mathcal{L} \), to solve a binary image classification problem (2’s vs 3’s):

\[
\hat{y} = f(x; \theta) = \text{sign}(\theta \cdot x)
\]

where \( x \) is a pixel image and \( \hat{y} \in \{-1, 1\} \).

• The parameters \( \theta \) were adjusted on the basis of the training examples and labels according to a simple mistake driven update rule (written here in a vector form)

\[
\theta \leftarrow \theta + y_i x_i, \quad \text{whenever} \quad y_i \neq \text{sign}(\theta \cdot x_i)
\]

• The update rule attempts to minimize the number of errors that the classifier makes on the training examples.
Estimation criterion

• We can formulate the binary classification problem more explicitly by defining a zero-one loss:

\[
\text{Loss}(y, \hat{y}) = \begin{cases} 
0, & y = \hat{y} \\
1, & y \neq \hat{y}
\end{cases}
\]

so that

\[
\frac{1}{n} \sum_{i=1}^{n} \text{Loss}(y_i, \hat{y}_i) = \frac{1}{n} \sum_{i=1}^{n} \text{Loss}(y_i, f(x_i; \theta))
\]

gives the fraction of prediction errors on the training set.

• This is a function of the parameters \( \theta \) and we can try to minimize it directly.
Suppose: 3 training examples, 3 classes. With some $W$ the scores $f(x, W) = WX$ are:

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

Multiclass SVM loss:

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

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$
Suppose: 3 training examples, 3 classes. With some $W$ the scores $f(x, W) = Wx$ are:

<table>
<thead>
<tr>
<th>Animal</th>
<th>Score 1</th>
<th>Score 2</th>
<th>Score 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

**Losses:** 2.9

### Multiclass SVM loss:

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

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

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

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

$$= 2.9 + 0$$

$$= 2.9$$
Suppose: 3 training examples, 3 classes. With some W the scores \( f(x, W) = Wx \) are:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>frog</th>
<th>car</th>
</tr>
</thead>
<tbody>
<tr>
<td>scores</td>
<td>3.2</td>
<td>-1.7</td>
<td>5.1</td>
</tr>
<tr>
<td>loss</td>
<td>2.2</td>
<td>-3.1</td>
<td>2.5</td>
</tr>
<tr>
<td>Losses</td>
<td>2.9</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

### Multiclass SVM loss:

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

the SVM loss has the form:

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

\[
= \max(0, 1.3 - 4.9 + 1) + \max(0, 2.0 - 4.9 + 1)
= \max(0, -2.6) + \max(0, -1.9)
= 0 + 0
= 0
\]
Suppose: 3 training examples, 3 classes.
With some $W$ the scores $f(x, W) = Wx$ are:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scores</td>
<td>3.2</td>
<td>5.1</td>
<td>-1.7</td>
</tr>
<tr>
<td></td>
<td>1.3</td>
<td>4.9</td>
<td>2.0</td>
</tr>
<tr>
<td>Losses</td>
<td>2.2</td>
<td>2.5</td>
<td>-3.1</td>
</tr>
<tr>
<td>2.9</td>
<td>0</td>
<td>10.9</td>
<td></td>
</tr>
</tbody>
</table>

**Multiclass SVM loss:**

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

the SVM loss has the form:

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

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

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

$$= 5.3 + 5.6$$

$$= 10.9$$
Suppose: 3 training examples, 3 classes.
With some $W$ the scores $f(x, W) = WX$ are:

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

Losses: 2.9 0 10.9

**Multiclass SVM loss:**

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

the SVM loss has the form:

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

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

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

$L = (2.9 + 0 + 10.9)/3 = 4.6$
Suppose: 3 training examples, 3 classes. With some $W$ the scores $f(x, W) = Wx$ are:

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

| Losses: | 2.9 | 0 | 10.9 |

**Multiclass SVM loss:**

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

the SVM loss has the form:

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

Q: what is the min/max possible loss?
Suppose: 3 training examples, 3 classes. With some $W$ the scores $f(x, W) = Wx$ are:

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

| Losses: | 2.9 | 0   | 10.9 |

---

**Multiclass SVM loss:**

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

the SVM loss has the form:

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

**Q:** usually at initialization $W$ are small numbers, so all $s \approx 0$. What is the loss?
\[ f(x, W) = Wx \]

\[ L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) \]
There is a bug with the loss:

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

\[
L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)
\]
There is a bug with the loss:

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

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

e.g. suppose that we found a \( W \) such that \( L = 0 \). Is this \( W \) unique?
Suppose: 3 training examples, 3 classes. With some $W$ the scores $f(x, W) = Wx$ are:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>frog</th>
<th>car</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.2</td>
<td>5.1</td>
<td>1.3</td>
</tr>
<tr>
<td></td>
<td>2.2</td>
<td>-1.7</td>
<td>4.9</td>
</tr>
<tr>
<td>Losses:</td>
<td>2.9</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

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

**Before:**

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

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

$= 0 + 0$

$= 0$

**With W twice as large:**

$= \max(0, 2.6 - 9.8 + 1) + \max(0, 4.0 - 9.8 + 1)$

$= \max(0, -6.2) + \max(0, -4.8)$

$= 0 + 0$

$= 0$
Weight Regularization

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

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

In common use:

**L2 regularization**

L1 regularization

Elastic net (L1 + L2)

Max norm regularization

Dropout (will see later)

Batch normalization (will see later)
L2 regularization: motivation

\[ x = [1, 1, 1, 1] \]

\[ w_1 = [1, 0, 0, 0] \]

\[ w_2 = [0.25, 0.25, 0.25, 0.25] \]

\[ w_1^T x = w_2^T x = 1 \]
Estimation criterion (revisited)

• We have reduced the estimation problem to a minimization problem

\[
\text{find } \theta \text{ that minimizes } \frac{1}{n} \sum_{i=1}^{n} \text{Loss}(y_i, f(x_i; \theta))
\]

—valid for any parameterized class of mappings from examples to predictions
—valid when the predictions are discrete labels, real valued, or other provided that the loss is defined appropriately
—may be ill-posed (under-constrained) as stated

• But why is it sensible to minimize the empirical loss in the first place since we are only interested in the performance on new examples?
Training and test performance: sampling

• We assume that each training and test example-label pair, \((x, y)\) is drawn independently at random from the same but unknown population of examples and labels.

• We can represent this population as a joint probability distribution \(P(x, y)\) so that each training/test example is a sample from this distribution \((x_i, y_i) \sim P\)
Training and test performance: sampling

- We assume that each training and test example-label pair, \((x, y)\) is drawn independently at random from the same but unknown population of examples and labels.

- We can represent this population as a joint probability distribution \(P(x, y)\) so that each training/test example is a sample from this distribution \((x_i, y_i) \sim P\)

\[
\text{Empirical (training) loss} = \frac{1}{n} \sum_{i=1}^{n} \text{Loss}(y_i, f(x_i; \theta))
\]

\[
\text{Expected (test) loss} = E_{(x, y) \sim P} \{ \text{Loss}(y, f(x; \theta)) \}
\]

- The training loss based on a few sampled examples and labels serves as a proxy for the test performance measured over the whole population.
Regression, example
Linear regression
— Estimation, errors, analysis
Regression

• The goal is to make quantitative (real valued) predictions on the basis of a (vector of) features or attributes

• Example: predicting vehicle fuel efficiency (mpg) from 8 attributes

<table>
<thead>
<tr>
<th>y</th>
<th>cyls</th>
<th>disp</th>
<th>hp</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>18.0</td>
<td>8</td>
<td>307.0</td>
<td>130.00</td>
<td>3504</td>
</tr>
<tr>
<td>26.0</td>
<td>4</td>
<td>97.00</td>
<td>46.00</td>
<td>1835</td>
</tr>
<tr>
<td>33.5</td>
<td>4</td>
<td>98.00</td>
<td>83.00</td>
<td>2075</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

• We need to
  — specify the class of functions (e.g., linear)
  — select how to measure prediction loss
  — solve the resulting minimization problem
Linear regression

- We begin by considering linear regression (easy to extend to more complex predictions later on)

\[
f : \mathcal{R} \to \mathcal{R} \quad f(x; w) = w_0 + w_1 x
\]

\[
f : \mathcal{R}^d \to \mathcal{R} \quad f(x; w) = w_0 + w_1 x_1 + \ldots + w_d x_d
\]

where \( w \) are parameters we need to set.
Linear regression: squared loss

We can measure the prediction loss in terms of squared error, so that the empirical loss on \( n \) training samples becomes mean squared error

\[
J_n(w) = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i; w))^2
\]

- We can measure the prediction loss in terms of squared error, 
  \( \text{Loss}(y, \hat{y}) = (y - \hat{y})^2 \), so that the empirical loss on \( n \) training samples becomes mean squared error

\[
f : \mathcal{R} \to \mathcal{R} \quad f(x; w) = w_0 + w_1 x
\]

\[
f : \mathcal{R}^d \to \mathcal{R} \quad f(x; w) = w_0 + w_1 x_1 + \ldots w_d x_d
\]
Linear regression: estimation

• We have to minimize the **empirical** squared loss

\[
J_n(w) = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i; w))^2
\]

\[
= \frac{1}{n} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i)^2 \quad (1\text{-dim})
\]

• By setting the derivatives with respect to \(w_1\) and \(w_0\) to zero, we get necessary conditions for the "optimal" parameter values

\[
\frac{\partial}{\partial w_1} J_n(w) = 0
\]

\[
\frac{\partial}{\partial w_0} J_n(w) = 0
\]
Optimality conditions: derivation

\[
\frac{\partial}{\partial w_1} J_n(w) = \frac{\partial}{\partial w_1} \frac{1}{n} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i)^2
\]
Optimality conditions: derivation

\[
\frac{\partial}{\partial w_1} J_n(w) = \frac{\partial}{\partial w_1} \frac{1}{n} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i)^2
\]

\[
= \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial w_1} (y_i - w_0 - w_1 x_i)^2
\]
Optimality conditions: derivation

\[
\frac{\partial}{\partial w_1} J_n(w) = \frac{\partial}{\partial w_1} \frac{1}{n} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i)^2
\]

\[
= \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial w_1} (y_i - w_0 - w_1 x_i)^2
\]

\[
= \frac{2}{n} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i) \frac{\partial}{\partial w_1} (y_i - w_0 - w_1 x_i)
\]
Optimality conditions: derivation

\[
\frac{\partial}{\partial w_1} J_n(w) = \frac{\partial}{\partial w_1} \frac{1}{n} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i)^2
\]

\[
= \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial w_1} (y_i - w_0 - w_1 x_i)^2
\]

\[
= \frac{2}{n} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i) \frac{\partial}{\partial w_1} (y_i - w_0 - w_1 x_i)
\]

\[
= \frac{2}{n} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i)(-x_i) = 0
\]
Optimality conditions: derivation

\[
\frac{\partial}{\partial w_1} J_n(w) = \frac{\partial}{\partial w_1} \frac{1}{n} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i)^2
\]

\[
= \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial w_1} (y_i - w_0 - w_1 x_i)^2
\]

\[
= \frac{2}{n} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i) \frac{\partial}{\partial w_1} (y_i - w_0 - w_1 x_i)
\]

\[
= \frac{2}{n} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i)(-x_i) = 0
\]

\[
\frac{\partial}{\partial w_0} J_n(w) = \frac{2}{n} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i)(-1) = 0
\]
Linear regression: matrix notation

• We can express the solution a bit more generally by resorting to a matrix notation so that

\[ y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}, \quad X = \begin{bmatrix} 1 & x_1 \\ \vdots & \vdots & \vdots \\ 1 & x_n \end{bmatrix}, \quad w = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \]

so that

\[
\frac{1}{n} \sum_{t=1}^{n} (y_t - w_0 - w_1 x_t)^2
= \frac{1}{n} \left\| \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} - \begin{bmatrix} 1 & x_1 \\ \vdots & \vdots & \vdots \\ 1 & x_n \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \right\|^2
= \frac{1}{n} \left\| y - Xw \right\|^2
\]
Linear regression: solution

• By setting the derivatives of $\frac{1}{n}||y - Xw||^2$ to zero, we get the same optimality conditions as before, now expressed in a matrix form

$$\frac{\partial}{\partial w} \frac{1}{n}||y - Xw||^2 = \frac{\partial}{\partial w} \frac{1}{n}(y - Xw)^T(y - Xw)$$
Linear regression: solution

• By setting the derivatives of $\frac{1}{n} \| y - Xw \|^2$ to zero, we get the same optimality conditions as before, now expressed in a matrix form

$$\frac{\partial}{\partial w} \frac{1}{n} ||y - Xw||^2 = \frac{\partial}{\partial w} \frac{1}{n} (y - Xw)^T(y - Xw)$$

$$= \frac{2}{n} X^T(y - Xw)$$
Linear regression: solution

• By setting the derivatives of $\frac{1}{n}\|y - Xw\|^2$ to zero, we get the same optimality conditions as before, now expressed in a matrix form

$$\frac{\partial}{\partial w} \frac{1}{n}\|y - Xw\|^2 = \frac{\partial}{\partial w} \frac{1}{n}(y - Xw)^T(y - Xw)$$

$$= \frac{2}{n}X^T(y - Xw)$$

$$= \frac{2}{n}(X^Ty - X^TXw) = 0$$

which gives

$$\hat{w} = (X^TX)^{-1}X^Ty$$

• The solution is a linear function of the outputs $y$
Alternative: Gradient Descent Algorithm

- One straightforward method: gradient descent
  - initialize $\theta$ (e.g., randomly)
  - repeatedly update $\theta$ based on the gradient

$$
\Delta = -\frac{1}{T} \sum_t \nabla_{\theta} l(f(x^{(t)}; \theta), y^{(t)}) - \lambda \nabla_{\theta} \Omega(\theta)
$$

$$
\theta \leftarrow \theta + \alpha \Delta
$$

- $\alpha$ is the learning rate
Effect of learning rate $\lambda$

- Large $\lambda \Rightarrow$ Fast convergence but larger residual error
  Also possible oscillations
- Small $\lambda \Rightarrow$ Slow convergence but small residual error
Local and Global Optima

![Graph showing local and global optima]

- **Global Maximum**
- **Local Maximum**
- **Local Minimum**
- **Global Minimum**
Stochastic Gradient Descent

- Two ways to generalize this for all examples in training set:

  1. **Batch updates**: sum or average updates across every example \( n \), then change the parameter values

  2. **Stochastic/online updates**: update the parameters for each training case in turn, according to its own gradients

\[
\Delta = -\nabla_\theta l(f(x^{(t)}; \theta), y^{(t)}) - \lambda \nabla_\theta \Omega(\theta)
\]

\[
\theta \leftarrow \theta + \alpha \Delta
\]
Linear regression: generalization

• As the number of training examples increases our solution gets “better”

We’d like to understand the error a bit better
Linear regression: types of errors

• **Structural error** measures the error introduced by the limited function class (infinite training data):

\[
\min_{w_1, w_0} E_{(x,y) \sim P} (y - w_0 - w_1 x)^2 = E_{(x,y) \sim P} (y - \hat{w}_0 - \hat{w}_1 x)^2
\]

where \((\hat{w}_0, \hat{w}_1)\) are the optimal linear regression parameters.

• **Approximation error** measures how close we can get to the optimal linear predictions with limited training data:

\[
E_{(x,y) \sim P} (w_0^* + w_1^* x - \hat{w}_0 - \hat{w}_1 x)^2
\]

where \((\hat{w}_0, \hat{w}_1)\) are the parameter estimates based on a small training set (therefore themselves random variables).
Linear regression: error decomposition

• The expected error of our linear regression function decomposes into the sum of structural and approximation errors

\[
E_{(x,y)} \sim P \left( y - \hat{w}_0 - \hat{w}_1 x \right)^2 = \\
E_{(x,y)} \sim P \left( y - w^*_0 - w^*_1 x \right)^2 + \\
E_{(x,y)} \sim P \left( w^*_0 + w^*_1 x - \hat{w}_0 - \hat{w}_1 x \right)^2
\]
Bias-Variance Tradeoff

- **Variance** of trained model: does it vary a lot if the training set changes
- **Bias** of trained model: is the average model close to the true solution?
- Generalization error can be seen as the sum of bias and the variance
Parametric vs. non-parametric models

• Parametric model: its capacity is fixed and does not increase with the amount of training data
  – examples: linear classifier, neural network with fixed number of hidden units, etc.

• Non-parametric model: the capacity increases with the amount of training data
  – examples: k nearest neighbors classifier, neural network with adaptable hidden layer size, etc.
Beyond linear regression models
— additive regression models, examples
— generalization and cross-validation
— population minimizer
Linear regression

• Linear regression functions,

\[ f : \mathcal{R} \rightarrow \mathcal{R} \quad f(x; \mathbf{w}) = w_0 + w_1 x, \text{ or} \]
\[ f : \mathcal{R}^d \rightarrow \mathcal{R} \quad f(\mathbf{x}; \mathbf{w}) = w_0 + w_1 x_1 + \ldots + w_d x_d \]

combined with the squared loss, are convenient because they are linear in the parameters.

— we get closed form estimates of the parameters

\[ \hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \]

where, for example, \( \mathbf{y} = [y_1, \ldots, y_n]^T \).

— the resulting prediction errors \( \epsilon_i = y_i - f(\mathbf{x}_i; \hat{\mathbf{w}}) \) are uncorrelated with any linear function of the inputs \( \mathbf{x} \).

— we can easily extend these to non-linear functions of the inputs while still keeping them linear in the parameters
Beyond linear regression

- Example extension: $m^{th}$ order polynomial regression where $f: \mathcal{R} \rightarrow \mathcal{R}$ is given by

$$f(x; \mathbf{w}) = w_0 + w_1 x + \ldots + w_{m-1} x^{m-1} + w_m x^m$$

—linear in the parameters, non-linear in the inputs
—solution as before

$$\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

where

$$\hat{\mathbf{w}} = \begin{bmatrix} \hat{w}_0 \\ \hat{w}_1 \\ \vdots \\ \hat{w}_m \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^m \\ 1 & x_2 & x_2^2 & \ldots & x_2^m \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^m \end{bmatrix}$$
Polynomial regression

degree = 1

degree = 3

degree = 5

degree = 7
Underfitting and Overfitting

Appropriate capacity

Overfitting

So far we have described only one way of changing a model's capacity: by changing the number of input features it has, and simultaneously adding new parameters associated with those features. There are in fact many ways of changing a model's capacity. Capacity is not determined only by the choice of model. The model specifies which family of functions the learning algorithm can choose from when varying the parameters in order to reduce a training objective. This is called the representational capacity of the model. In many cases, finding the best function within this family is a very difficult optimization problem. In practice, the learning algorithm does not actually find the best function, but merely one that significantly reduces the training error. These additional limitations, such as
Generalization and Capacity

Figure 5.3: Typical relationship between capacity and error. Training and test error behave differently. At the left end of the graph, training error and generalization error are both high. This is the underfitting regime. As we increase capacity, training error decreases, but the gap between training and generalization error increases. Eventually, the size of this gap outweighs the decrease in training error, and we enter the overfitting regime, where capacity is too large, above the optimal capacity.

The concept of non-parametric models. So far, we have seen only parametric models, such as linear regression. Parametric models learn a function described by a parameter vector whose size is finite and fixed before any data is observed. Non-parametric models have no such limitation.

Sometimes, non-parametric models are just theoretical abstractions (such as an algorithm that searches over all possible probability distributions) that cannot be implemented in practice. However, we can also design practical non-parametric models by making their complexity a function of the training set size. One example of such an algorithm is nearest neighbor regression. Unlike linear regression, which has a fixed-length vector of weights, the nearest neighbor regression model simply stores the $X$ and $y$ from the training set. When asked to classify a test point $x$, the model looks up the nearest entry in the training set and returns the associated regression target. In other words, $\hat{y} = y_i$ where $i = \arg\min ||X_i - x||_2$.

The algorithm can also be generalized to distance metrics other than the $L^2$ norm, such as learned distance metrics (Goldberger et al., 2005). If the algorithm is allowed to break ties by averaging the $y_i$ values for all $X_i$ that are tied for nearest, then this algorithm is able to achieve the minimum possible training error (which might be greater than zero, if two identical inputs are associated with different outputs) on any regression dataset.

Finally, we can also create a non-parametric learning algorithm by wrapping a
Bias and Variance

Bias and Variance

The MSE measures the overall expected deviation—in a squared error sense—between the estimator and the true value of the parameter. As is clear from equation 5.54, evaluating the MSE incorporates both the bias and the variance. Desirable estimators are those with small MSE and these are estimators that manage to keep both their bias and variance somewhat in check.

![Bias and Variance Diagram](Image credit: Ian Goodfellow)

The relationship between bias and variance is tightly linked to the machine learning concepts of capacity, underfitting and overfitting. In the case where generalization error is measured by the MSE (where bias and variance are meaningful components of generalization error), increasing capacity tends to increase variance and decrease bias. This is illustrated in figure 5.6, where we see again the U-shaped curve of generalization error as a function of capacity.

5.4.5 Consistency

So far we have discussed the properties of various estimators for a training set of fixed size. Usually, we are also concerned with the behavior of an estimator as the amount of training data grows. In particular, we usually wish that, as the number of data points \( m \) in our dataset increases, our point estimates converge to the true

sufficiently simpler models are more likely to generalize
Complexity and overfitting

• With limited training examples our polynomial regression model may achieve zero training error but nevertheless has a large test (generalization) error.

\[
\begin{align*}
\text{train} & : \frac{1}{n} \sum_{t=1}^{n} (y_t - f(x_t; \hat{w}))^2 \approx 0 \\
\text{test} & : E_{(x,y) \sim P} (y - f(x; \hat{w}))^2 \gg 0
\end{align*}
\]

• We suffer from overfitting when the training error no longer bears any relation to the generalization error.
Avoiding overfitting: cross-validation

• **Cross-validation** allows us to estimate the generalization error based on training examples alone.

Leave-one-out cross-validation treats each training example in turn as a test example:

\[
CV = \frac{1}{n} \sum_{i=1}^{n} \left( y_i - f(x_i; \hat{w}^{-i}) \right)^2
\]

where \( \hat{w}^{-i} \) are the least squares estimates of the parameters without the \( i^{th} \) training example.
Polynomial regression: example (cont’d)

degree = 1, CV = 0.6    degree = 3, CV = 1.5

degree = 5, CV = 6.0    degree = 7, CV = 15.6
Additive models

- More generally, predictions can be based on a linear combination of a set of basis functions (or features) \( \{\phi_1(x), \ldots, \phi_m(x)\} \), where each \( \phi_i(x) : \mathcal{R}^d \rightarrow \mathcal{R} \), and

\[
f(x; \mathbf{w}) = w_0 + w_1 \phi_1(x) + \ldots + w_m \phi_m(x)
\]

- Examples

If \( \phi_i(x) = x^i \), \( i = 1, \ldots, m \), then

\[
f(x; \mathbf{w}) = w_0 + w_1 x + \ldots + w_{m-1} x^{m-1} + w_m x^m
\]
Additive models

• More generally, predictions can be based on a linear combination of a set of basis functions (or features) \( \{ \phi_1(x), \ldots , \phi_m(x) \} \), where each \( \phi_i(x) : \mathcal{R}^d \rightarrow \mathcal{R} \), and

\[
    f(x; w) = w_0 + w_1 \phi_1(x) + \ldots + w_m \phi_m(x)
\]

• Examples

  If \( \phi_i(x) = x^i, i = 1, \ldots , m \), then

  \[
  f(x; w) = w_0 + w_1 x + \ldots + w_{m-1} x^{m-1} + w_m x^m
  \]

  If \( m = d \), \( \phi_i(x) = x_i, i = 1, \ldots , d \), then

  \[
  f(x; w) = w_0 + w_1 x_1 + \ldots + w_d x_d
  \]
Additive models (cont’d)

• The basis functions can capture various (e.g., qualitative) properties of the inputs.

• For example: we can try to rate companies based on text descriptions

\[ x = \text{text document (collection of words)} \]

\[ \phi_i(x) = \begin{cases} 1 & \text{if word } i \text{ appears in the document} \\ 0 & \text{otherwise} \end{cases} \]

\[ f(x; w) = w_0 + \sum_{i \in \text{words}} w_i \phi_i(x) \]
Additive models (cont’d)

- We can view the additive models graphically in terms of simple “units” and “weights”

\[ f(x; w) = w_0 + \sum_{i=1}^{m} w_i \phi_i(x) \]

- In neural networks the basis functions themselves have adjustable parameters (cf. prototypes)
Take-home messages

Training error high?
  Yes → Bigger model
  No → Train-Dev error high?
  Yes → More data
  No → Dev error high?
  Yes → Make training data more similar to test data.
  No → Test error high?
  Yes → More dev set data
  No → Done!

(Bias)

(Variances)

(Train-test data mismatch)

(Overfit dev set)
Statistical regression models
— model formulation, motivation
— maximum likelihood estimation
Statistical view of linear regression

• In a statistical regression model we model both the function and noise

\[
\text{Observed output} = \text{function} + \text{noise}
\]

\[
y = f(x; w) + \epsilon
\]

where, e.g., \( \epsilon \sim \mathcal{N}(0, \sigma^2) \).

• Whatever we cannot capture with our chosen family of functions will be interpreted as noise
Statistical view of linear regression

• $f(x; w)$ is trying to capture the mean of the observations $y$ given the input $x$:

\[ E\{ y \mid x \} = E\{ f(x; w) + \epsilon \mid x \} = f(x; w) \]

where $E\{ y \mid x \}$ is the conditional expectation of $y$ given $x$, evaluated according to the model (not according to the underlying distribution $P$)
Statistical view of linear regression

- According to our statistical model

\[ y = f(x; w) + \epsilon, \quad \epsilon \sim N(0, \sigma^2) \]

the outputs \( y \) given \( x \) are normally distributed with mean \( f(x; w) \) and variance \( \sigma^2 \):

\[ p(y|x, w, \sigma^2) = \frac{1}{\sqrt{2\pi \sigma^2}} \exp\left\{ -\frac{1}{2\sigma^2}(y - f(x; w))^2 \right\} \]

(we model the uncertainty in the predictions, not just the mean)

- Loss function? Estimation?
Maximum likelihood estimation

• Given observations \( D_n = \{(x_1, y_1), \ldots, (x_n, y_n)\} \) we find the parameters \( w \) that maximize the (conditional) likelihood of the outputs

\[
L(D_n; w, \sigma^2) = \prod_{i=1}^{n} p(y_i|x_i, w, \sigma^2)
\]

• Example: linear function

\[
p(y|x, w, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ -\frac{1}{2\sigma^2}(y - w_0 - w_1 x)^2 \right\}
\]

(why is this a bad fit according to the likelihood criterion?)
Maximum likelihood estimation (cont’d)

Likelihood of the observed outputs:

\[ L(D; \mathbf{w}, \sigma^2) = \prod_{i=1}^{n} P(y_i | \mathbf{x}_i, \mathbf{w}, \sigma^2) \]

• It is often easier (but equivalent) to try to maximize the log-likelihood:

\[
l(D; \mathbf{w}, \sigma^2) = \log L(D; \mathbf{w}, \sigma^2) = \sum_{i=1}^{n} \log P(y_i | \mathbf{x}_i, \mathbf{w}, \sigma^2) \]

\[
= \sum_{i=1}^{n} \left( -\frac{1}{2\sigma^2} (y_i - f(\mathbf{x}_i; \mathbf{w}))^2 - \log \sqrt{2\pi\sigma^2} \right) \\
= \left( -\frac{1}{2\sigma^2} \right) \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i; \mathbf{w}))^2 + \ldots
\]
Maximum likelihood estimation (cont’d)

• Maximizing log-likelihood is equivalent to minimizing empirical loss when the loss is defined according to

\[
\text{Loss}(y_i, f(x_i; w)) = -\log P(y_i|x_i, w, \sigma^2)
\]

Loss defined as the negative log-probability is known as the log-loss.
Maximum likelihood estimation (cont’d)

• The log-likelihood of observations

\[
\log L(D; w, \sigma^2) = \sum_{i=1}^{n} \log P(y_i | x_i, w, \sigma^2)
\]

is a generic fitting criterion and can be used to estimate the noise variance \( \sigma^2 \) as well.

• Let \( \hat{w} \) be the maximum likelihood (here least squares) setting of the parameters. What is the maximum likelihood estimate of \( \sigma^2 \), obtained by solving

\[
\frac{\partial}{\partial \sigma^2} \log L(D; w, \sigma^2) = 0 \quad ?
\]
Maximum likelihood estimation (cont’d)

• The log-likelihood of observations

\[ \log L(D; \mathbf{w}, \sigma^2) = \sum_{i=1}^{n} \log P(y_i | x_i, \mathbf{w}, \sigma^2) \]

is a generic fitting criterion and can be used to estimate the noise variance \( \sigma^2 \) as well.

• Let \( \hat{\mathbf{w}} \) be the maximum likelihood (here least squares) setting of the parameters. The maximum likelihood estimate of the noise variance \( \sigma^2 \) is

\[ \hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i; \hat{\mathbf{w}}))^2 \]

i.e., the mean squared prediction error.
Polynomial regression

• Consider again a simple $m^{th}$ degree polynomial regression model

$$y = w_0 + w_1 x + \ldots + w_m x^m + \epsilon, \quad \epsilon \sim N(0, \sigma^2)$$

where $\sigma^2$ is assumed fixed (known).

• In this model the outputs $\{y_1, \ldots, y_n\}$ corresponding to any inputs $\{x_1, \ldots, x_n\}$ are generated according to

$$y = Xw + e,$$

where

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}, \quad X = \begin{bmatrix} 1 & x_1 & \ldots & x_1^m \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \ldots & x_n^m \end{bmatrix}, \quad e = \begin{bmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

and $\epsilon_i \sim N(0, \sigma^2)$, $i = 1, \ldots, n$. 

Tommi Jaakkola, MIT CSAIL
ML estimator, uncertainty

• We are interested in studying how the choice of inputs \{x_1,\ldots,x_n\} or, equivalently, \(X\), affects the accuracy of our regression model.

• Our model for the outputs \{y_1,\ldots,y_n\} given \(X\) is

\[
y = Xw + e, \quad e \sim N(0, \sigma^2 I)
\]

• We assume also that the training outputs are actually generated by a model in this class with some fixed but unknown parameters \(w^*\) (same \(\sigma^2\))

\[
y = Xw^* + e, \quad e \sim N(0, \sigma^2 I)
\]

• We can now ask, for a given \(X\), how accurately we are able to recover the "true" parameters \(w^*\).
ML estimator, uncertainty

• The ML estimator $\hat{w}$ viewed here as a function of the outputs $y$ for a fixed $X$, is given by

$$\hat{w} = (X^TX)^{-1}X^Ty$$

• We need to understand how $\hat{w}$ varies in relation to $w^*$ when the outputs are generated according to

$$y = Xw^* + e, \quad e \sim N(0, \sigma^2I)$$

• In the absence of noise $e$, the ML estimator would recover $w^*$ exactly (with only minor constraints on $X$)

$$\hat{w} = (X^TX)^{-1}X^T(Xw^*)$$

$$= (X^TX)^{-1}(X^TX)w^*$$

$$= w^*$$
ML estimator, uncertainty

• In the presence of noise we can still use the fact that $y = Xw^* + e$ to simplify the parameter estimates

$$\hat{w} = (X^TX)^{-1}X^Ty$$

$$= (X^TX)^{-1}X^T(Xw^* + e)$$

$$= (X^TX)^{-1}(X^TX)w^* + (X^TX)^{-1}X^Te$$

$$= w^* + (X^TX)^{-1}X^Te$$

• So the ML estimate is the correct parameter vector plus an estimate based purely on noise
Next Lecture:

Basic Concepts of Artificial Neural Networks