WORKSHOP ON RECOMMENDER SYSTEMS

Day 1: Introduction to Machine Learning and Deep Learning

Jonathan Guymont

August 20, 2019

Objectives[1]

- Understanding what machine learning is.
- Understanding some machine learning algorithms.
- Being familiar with gradient based learning.
- Being familiar with important machine learning concept: overfitting, bias-variance trade-off, regularization.
- Understanding how machine learning algorithms are developed in practice.

[1] Some slides are taken from the McGill COMP-652 and COMP-551 courses and have been created by Doina Precup and Joelle Pineau.

What is Machine Learning?

What is Machine Learning?

- A. Samuel, who coined the term *machine learning*:

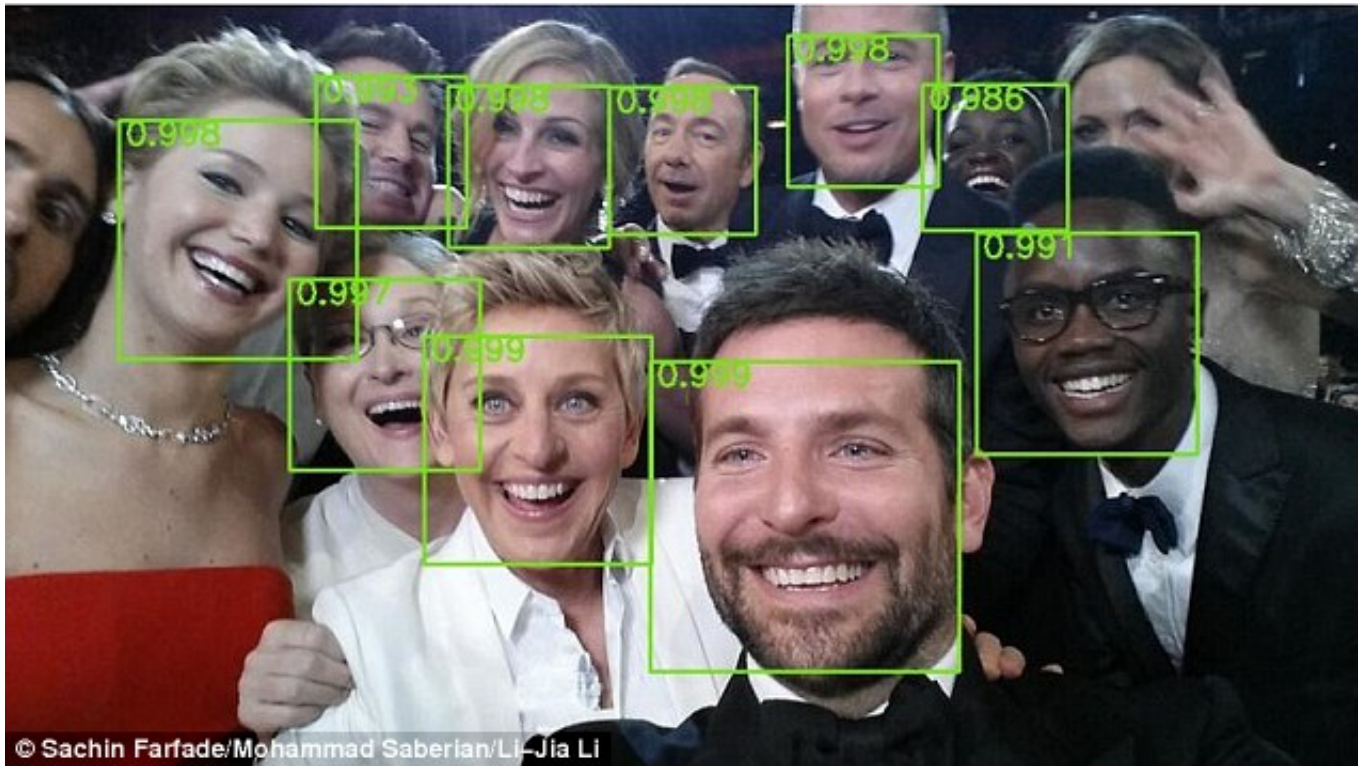
  "Field of study that gives computers the ability to learn without being explicitly programmed."

- The machine learning approach allow computers to learn from experience.
- Gathering knowledge from experience avoids the need for human operators to formally specify all the knowledge that the computer needs.
- What is not learning
  - Hard-coding knowledge about the world (e.g. accounting software).
  - Solving problems analytically (e.g. shortest path with Dijkstra's algorithm).

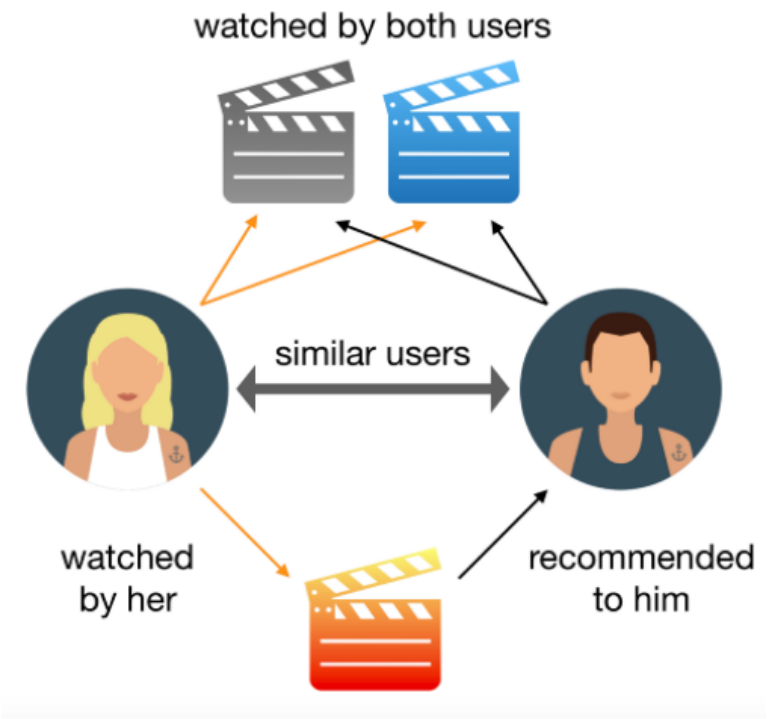
Current Examples of Machine Learning Applications

- Translation (e.g. Google translate)
- Computer vision systems (e.g. autonomous vehicles)
- Voice recognition (eg. Siri, Google assistant)
- Smart product recommendations (e.g. Netflix, Amazon)
- Decision making (e.g. autonomous vehicle)

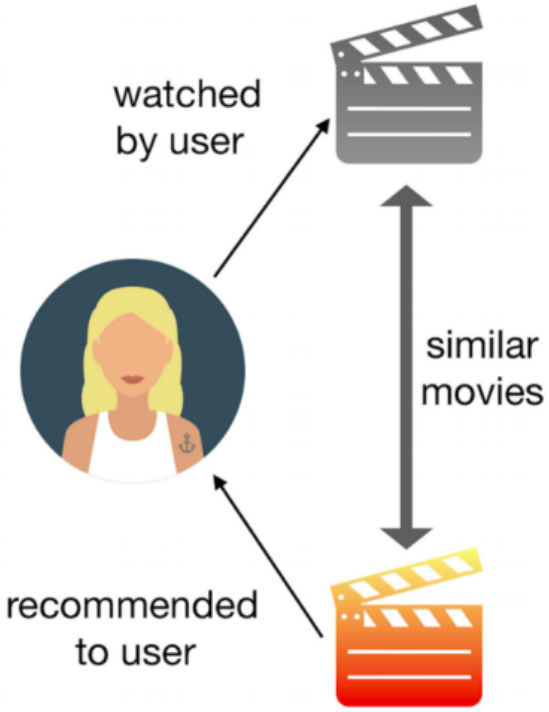
Example: Computer Vision, Face Recognition



Example: Collaborative and Content Based Filters



Collaborative-based filter.



Content-based filter.

Example: Language Modeling

- Learning a language model:

Text corpus ⟶ Statistical language model

$$P(W_n|W_{n-1}) = \frac{P(W_{n-1}, W_n)}{P(W_{n-1})}$$

Example: Language Modeling

"Two pizzas sitting on top of a stove top oven"

"A group of young people playing a game of frisbee"

Example: Language Modeling

"A refrigerator filled with lots of food and drinks"

"A yellow school bus parked in a parking lot"

Supervised Learning Problems

- Supervised learning: For every input x there is a label y
  - **Classification.** In this type of task, the computer program is asked to specify which of k categories some input belongs to.
    - The target y represents a category or class.
  - **Regression.** In this type of task, the computer program is asked to predict a numerical value given some input, i.e. y is a real value.
    - To solve this task, the learning algorithm is asked to output a function $f : \mathbb{R}^d \mapsto \mathbb{R}$.
    - An example of a regression task is the prediction of the expected claim amount that an insured person will make.

Data

Supervised Example (or sample)

In the context of supervised learning, examples are pairs of input-target:

$$(x, y)$$

Text classification examples:

("IPhone 16 will revolutionize the mobile industry","positive")

("Apple has difficulty to innovate since Steve Jobs","negative")

Supervised Example (or sample)

In the context of supervised learning, examples are pairs of input-target:

$$(x, y)$$

Image classification examples:

$$\left(x = \text{  }, y = \text{"bear"} \right) \qquad \left(x = \text{  }, y = \text{"cat"} \right)$$

Supervised Example (or sample)

In the context of supervised learning, examples are pairs of input-target:

$$(x, y)$$

Regression:

$$\big(x = (\#\text{bedrooms}, \#\text{bathrooms}, \text{property\_type}, \text{size})^\top, y = \text{price}\big)$$

$$\big(x = (4, 1, \text{"condominium"}, 50000)^\top, y = 100000\big)$$

Dataset

A dataset is a set of examples:

$$D = \{(x^{(1)}, y^{(1)}), ..., (x^{(n)}, y^{(n)})\}$$

- For example, a dataset for sentiment analysis could look like this

	reviews	sentiment
0	[crust, is, not, good]	0
1	[not, tasty, and, the, texture, was, just, nasty]	0
2	[stopped, by, during, the, late, may, bank, ho...	1
3	[the, selection, on, the, menu, was, great, an...	1
4	[now, i, am, getting, angry, and, i, want, my,...	0

Dataset

- A dataset is a set of examples:

$$D = \{(x^{(1)}, y^{(1)}), ..., (x^{(n)}, y^{(n)})\}$$

- House pricing example

# bedrooms	# bathrooms	property_type	size	price
4	1	condominium	50000	100000
1	1	space station	20000	250000000
1	0	doghouse	10	400
14	5	condominium	500000	10000000

Table 1: Example of dataset use to train a house pricing model

Design Matrix Dataset

- A common way of describing a dataset is with a **design matrix**. A design matrix is a matrix containing a different example in each row. Each column of the matrix corresponds to a different feature.

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots x_{1d} \\ x_{21} & x_{22} & \cdots x_{2d} \\ \vdots & \vdots & \ddots \vdots \\ x_{n1} & x_{n2} & \cdots x_{nd} \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Preprocessing

- Often the data needs to be transformed before it can be fed to an algorithm (e.g. image, text, categorical feature,...)
- Image are represented with the value of their pixels
- Words and categorical features are represented by a unique index

$$[\text{"crust"}, \text{"is"}, \text{"not"}, \text{"good"}] \rightarrow (34, 67, 87, 65)$$

$$\text{"condominium"} \rightarrow 0 \quad \text{"space station"} \rightarrow 1 \quad \text{"doghouse"} \rightarrow 2$$

- Usually, the data is then processed to facilitate optimization and/or increase capacity
  - Z-normalization and min-max scaling for real valued inputs
  - One-hot encoding and embedding for categorical features

Preprocessing

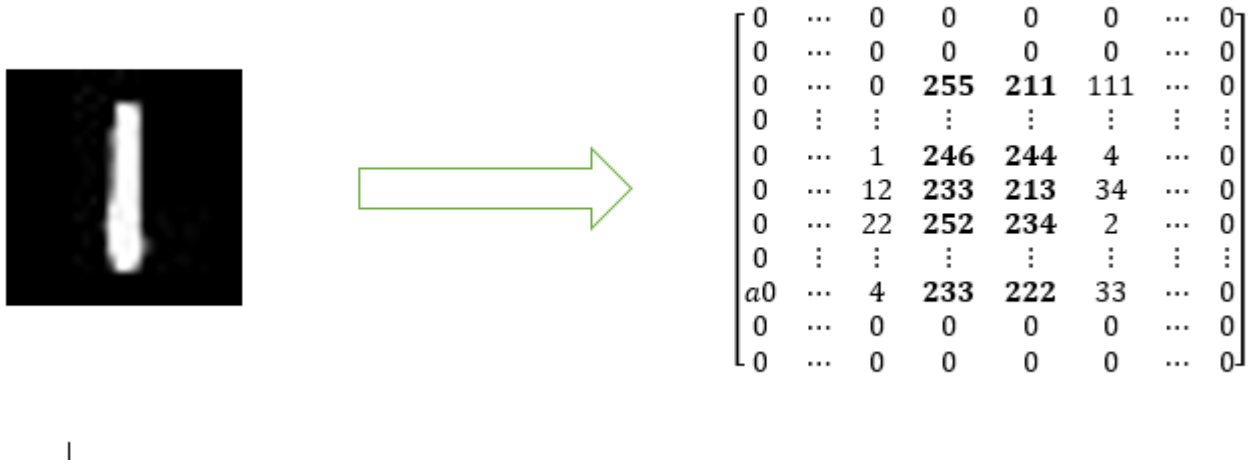


Figure 1: Preprocessing of an image: images need to be transformed into something machines can understand, a matrix. The dimension of the matrix correspond to the dimension of the image in pixels. Each element of the matrix is equal to the value of the pixel at the same position.

Questions?

Linear Regression

Linear Regression

- A linear regression model can be express as

$$f_{\mathbf{w}}(\mathbf{x}) = b + w_1 x_1 + \cdots w_d x_d \tag{1}$$

$$= b + \sum_{i=1}^{d} w_i x_i \tag{2}$$

$$= b + \mathbf{w}^T \mathbf{x} \tag{3}$$

- $\mathbf{x} = (x_1, ..., x_d)^\top$ are the input features
- f is the predictor function
- w_i are called *parameters* or *weights* (or coefficients in classic statistics)
- b is called the *bias* (or intercept in classic statistics)
- $\mathbf{w}$ and $\mathbf{x}$ are vectors of size d.

Linear Regression Example: Data

Example: predict the score streamers are going to give to the new Lion King movie based on there rating of two other Disney movies.

	Lion King (original)	Aladin (Remake)	Lion King (Remake)
1	0.70	0.41	0.70
2	0.86	0.50	0.85
3	0.45	0.58	0.58
4	0.48	0.71	0.64
5	0.58	0.82	0.73
6	0.78	0.88	0.88
7	0.69	0.50	0.71
8	0.94	0.57	0.89
9	0.67	0.81	0.79
10	0.78	0.95	0.91
11	0.61	0.64	0.72
12	0.64	0.92	0.80

Table 2: Disney movie score

Linear Regression Example: Model

Example: predict the score streamers are going to give to the new Lion King movie based on there rating of two other Disney movies.

	inputs $\mathbf{x}=(x_1,x_2)^\top$		target y
	Lion King (original)	Aladin (Remake)	Lion King (Remake)
1	0.70	0.41	0.70
2	0.86	0.50	0.85
3	0.45	0.58	0.58
⋮	⋮	⋮	⋮

Table 3: Disney movie scores

In this example, we want a model with the form

$$f(\mathbf{x}) = b + w_1 x_1 + w_2 x_2 = b + \mathbf{w}^\top \mathbf{x}$$

How should we pick $\mathbf{w}$ *and* b*?*

Linear Regression Example: Model

- The weights $\mathbf{w}$ are initialize randomly. Usually, from a centered distribution with low variance is used, e.g $w_i \sim \mathcal{N}(0, 0.5)$.
- The bias b is usually initialize at 0
- Most programming language support random sampling from common distribution

Suppose our randomly initialize model is as follow:

$$f(\mathbf{x}) = -0.3614x_1 + 1.4332x_2$$

and let's look at the prediction

Linear Regression Example: Untrained Results

	inputs $\mathbf{x}=(x_1,x_2)^\top$		target y		
	Lion King (original)	Aladin (Remake)	Lion King (Remake)	$f(\mathbf{x})$	$\lvert f(\mathbf{x}) - y \rvert$
1	0.70	0.41	0.70	0.33	0.37
2	0.86	0.50	0.85	0.40	0.45
3	0.45	0.58	0.58	0.41	0.17
4	0.48	0.71	0.64	0.49	0.15
5	0.58	0.82	0.73	0.57	0.16
6	0.78	0.88	0.88	0.63	0.25
7	0.69	0.50	0.71	0.38	0.33
8	0.94	0.57	0.89	0.45	0.44
9	0.67	0.81	0.79	0.58	0.21
10	0.78	0.95	0.91	0.68	0.23
11	0.61	0.64	0.72	0.46	0.26
12	0.64	0.92	0.80	0.64	0.16

Linear Regression: Loss Function / Objective

- Our objective is too have a model that predicts values that are as closed as possible to the ground truth.
- The convention in statistics is to minimize a $loss$.
- In order to train our model, we need a $loss$ function that measures the cost of an error made by the model so it can improve himself.
- Intuitively, any function that sum the absolute errors could work, e.g.
  - **Mean Absolute Error (MAE).**

$$L(b, \mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} |y^{(i)} - f(\mathbf{x}^{(i)})|$$

  - **Mean Squared Error (MSE).**

$$L(b, \mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} (f(\mathbf{x}^{(i)}) - y^{(i)})^2$$

Linear Regression: Loss Function / Objective

- We will use the MSE loss and justify this choice later

$$L(b, \mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} (f(\mathbf{x}^{(i)}) - y^{(i)})^2$$

- Minimizing the loss w.r.t b and $\mathbf{w}$ is an optimization problem. It is also what is called *training* in machine learning.
- The most popular method to optimize machine learning models, especially deep learning models, is gradient descent.

Questions?

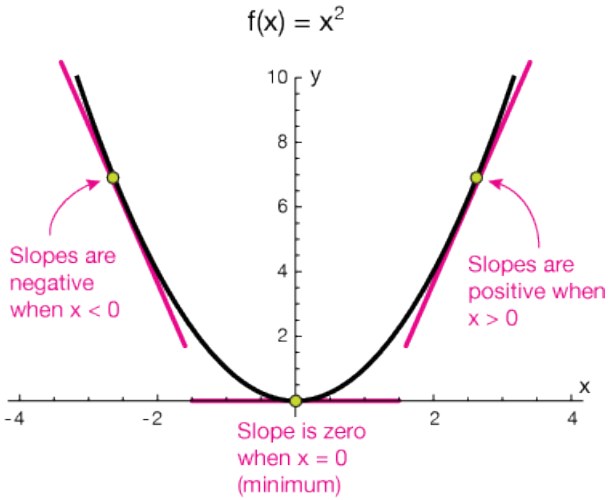
Gradient Based Learning

Derivative

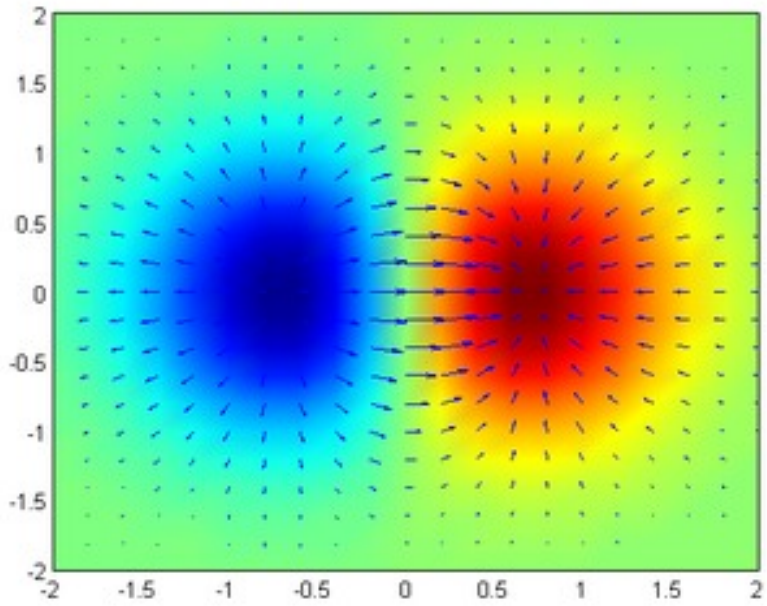
- Measures the sensitivity of a function f to a change of its input (at a point x)
- Let $f : \mathbb{R} \to \mathbb{R}$, at a point x, if we add a small ϵ to x, how much will $f(x)$ move?

$$f^{'}(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

- Geometrically: this is the slope of the curve at each point.



Gradient



- Multivartiate generalization of the derivative.
- Points in the direction of the greatest increase of the function.
- Consider a function $f(u_1, u_2, \ldots, u_n) : \mathbb{R}^n \mapsto \mathbb{R}$ (for us, this will usually be a loss function)

- The *partial derivative* w.r.t. u_i is denoted:

$$\frac{\partial}{\partial u_i} f(u_1, u_2, \ldots, u_n) : \mathbb{R}^n \mapsto \mathbb{R}$$

  The partial derivative is the derivative along the u_i axis, keeping all other variables fixed.

- The *gradient* $\nabla f(u_1, u_2, \ldots, u_n) : \mathbb{R}^n \mapsto \mathbb{R}^n$ is a function which outputs a vector containing the partial derivatives. That is

$$\nabla f = \left\langle \frac{\partial}{\partial u_1} f, \frac{\partial}{\partial u_2} f, \ldots, \frac{\partial}{\partial u_n} f \right\rangle$$

Optimization Using the Gradient Descent Method

- Notation: $\theta \equiv$ set of parameters (e.g. $\theta = \{b, \mathbf{w}\}$)
- Initialize all parameters θ randomly (or using an appropriate heuristic)
- Until convergence ($\nabla_\theta L(\theta) \approx 0$), repeat:

$$\theta \leftarrow \theta - \alpha \nabla L(\theta)$$

- α is a positive real number called the learning rate or step size.

Gradient Descent

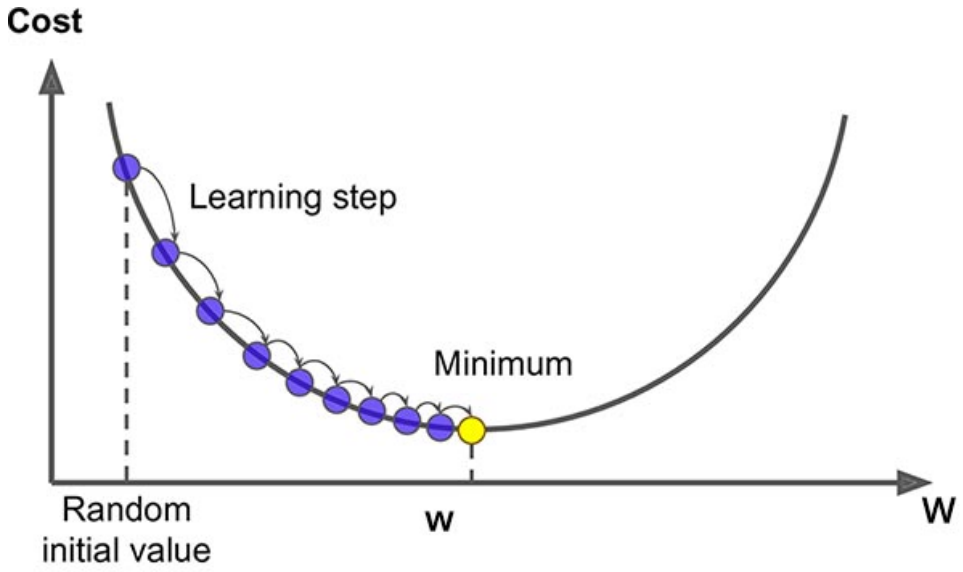


Figure 2: Illustration of gradient descent

Local and Global Minimum

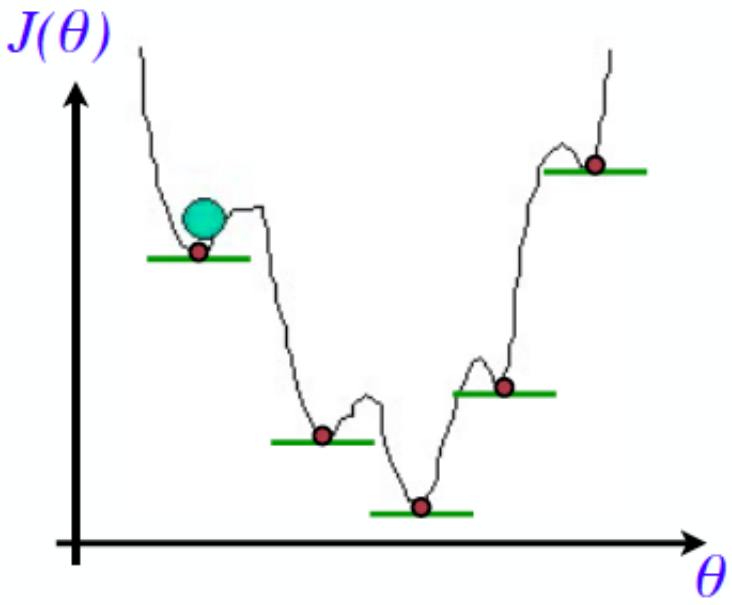
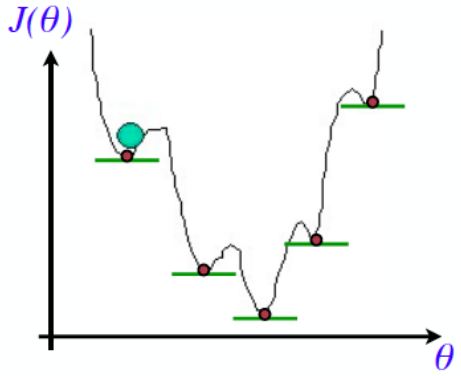
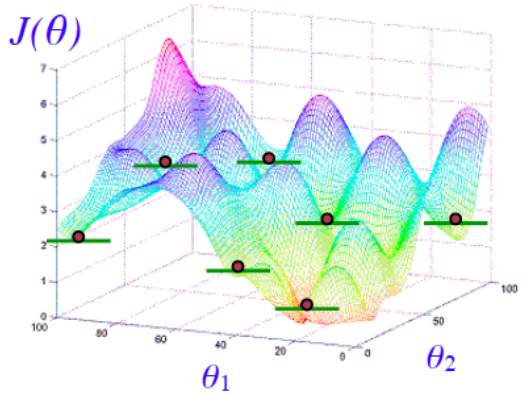


Figure 3: Illustration of local and global minimum. *Source: Andrew Ng machine learning course*

The Real Risk Minimizer in Linear Regression

- For some classes of function, it is possible to find the global minimum by setting the gradient to zero and solving for the parameters
- In the case of linear regression, setting gradient equal to zero $\Rightarrow \mathbf{w}^* = (\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\mathbf{y}$





- This is not true for neural network. When optimizing the parameters of neural network, we converge to a local minimum (rather than a global one) which depends on the starting point.

Questions?

Back to the Disney Example

Linear Regression Example: Gradient

- In our earlier example, we had the following loss function

$$L(b, \mathbf{w}) = L(b, w_1, w_2) = \frac{1}{n} \sum_{i=1}^{n} (f(\mathbf{x}^{(i)}) - y^{(i)})^2$$

- We need $\frac{\partial}{\partial b} L(b, w_1, w_2)$, $\frac{\partial}{\partial w_1} L(b, w_1, w_2)$, and $\frac{\partial}{\partial w_2} L(b, w_1, w_2)$

Linear Regression Example: Derivative of the loss w.r.t b

$$
\begin{aligned}
\frac{\partial}{\partial b} L(b, w_1, w_2) =& \frac{\partial}{\partial b} \frac{1}{n} \sum_{i=1}^{n} (f(\mathbf{x}^{(i)}) - y^{(i)})^2 \\
=& \frac{2}{n} \sum_{i=1}^{n} (f(\mathbf{x}^{(i)}) - y^{(i)}) \frac{\partial}{\partial b} (f(\mathbf{x}^{(i)}) - y^{(i)}) \\
=& \frac{2}{n} \sum_{i=1}^{n} (f(\mathbf{x}^{(i)}) - y^{(i)}) \frac{\partial}{\partial b} (b + \mathbf{w}^\top \boldsymbol{x}^{(i)} - y^{(i)}) \\
=& \frac{2}{n} \sum_{i=1}^{n} (f(\mathbf{x}^{(i)}) - y^{(i)})
\end{aligned}
$$

Linear Regression Example: Derivative of the loss w.r.t w_1 and w_2

$$
\begin{aligned}
\frac{\partial}{\partial w_1} L(b, w_1, w_2) =& \frac{\partial}{\partial w_1} \frac{1}{n} \sum_{i=1}^{n} (f(\mathbf{x}^{(i)}) - y^{(i)})^2 \\
=& \frac{2}{n} \sum_{i=1}^{n} (f(\mathbf{x}^{(i)}) - y^{(i)}) \frac{\partial}{\partial b} (f(\mathbf{x}^{(i)}) - y^{(i)}) \\
=& \frac{2}{n} \sum_{i=1}^{n} (f(\mathbf{x}^{(i)}) - y^{(i)}) \frac{\partial}{\partial b} (b + w_1 x_1^{(i)} + w_2 x_2^{(i)} - y^{(i)}) \\
=& \frac{2}{n} \sum_{i=1}^{n} (f(\mathbf{x}^{(i)}) - y^{(i)}) x_1^{(i)} \\
\frac{\partial}{\partial w_2} L(b, w_1, w_2) =& \frac{2}{n} \sum_{i=1}^{n} (f(\mathbf{x}^{(i)}) - y^{(i)}) x_2^{(i)}
\end{aligned}
$$

Linear Regression Example: Model

	inputs $\mathbf{x}=(x_1,x_2)^\top$		target y
	Lion King (original)	Aladin (Remake)	Lion King (Remake)
1	0.70	0.41	0.70
2	0.86	0.50	0.85
3	0.45	0.58	0.58
⋮	⋮	⋮	⋮

Table 4: Disney movie scores

In this example, we want a model with the form

$$f(\mathbf{x}) = b + w_1 x_1 + w_2 x_2 = b + \mathbf{w}^\top \mathbf{x}$$

How should we pick $\mathbf{w}$ *and* b*?*

We choose $\mathbf{w}$ and b such that the MSE is mimimized. We achive this using gradient descent.

Linear Regression Example: Training

	inputs $\mathbf{x}=(x_1,x_2)^\top$		target y
	Lion King (original)	Aladin (Remake)	Lion King (Remake)
1	0.70	0.41	0.70
2	0.86	0.50	0.85
3	0.45	0.58	0.58
⋮	⋮	⋮	⋮

Table 5: Disney movie scores

- Split the data in 2:

$$D^{(\text{train})} = \left\{ \left(\mathbf{x}^{(i)}, y^{(i)} \right) : i \in \{1, 2, ..., 9\}) \right\}$$
$$D^{(\text{test})} = \left\{ \left(\mathbf{x}^{(i)}, y^{(i)} \right) : i \in \{10, 11, 12\} \right\}$$

Linear Regression Example: Training

- TOL $= 0.0001$
- $\alpha = 0.02$
- MAX_ITER $= 500$
- $i = 0$
- `while` $\max\left\{\left|\frac{\partial}{\partial b}L(b, \mathbf{w})\right|, \left|\frac{\partial}{\partial w_1}L(b, \mathbf{w})\right|, \left|\frac{\partial}{\partial w_2}L(b, \mathbf{w})\right|\right\} >$ TOL `do:`

  $b \leftarrow b - \alpha\frac{\partial}{\partial b}L(b, \mathbf{w})$

  $w_1 \leftarrow w_1 - \alpha\frac{\partial}{\partial w_1}L(b, \mathbf{w})$

  $w_2 \leftarrow w_2 - \alpha\frac{\partial}{\partial w_2}L(b, \mathbf{w})$

  `if` $i >$ MAX_ITER `do:`

  `break`

  Where

$$L(b, \mathbf{w}) = \frac{1}{n}\sum_{(\mathbf{x},y)\in D^{(\text{train})}}\left(f_{b,\mathbf{w}}(\mathbf{x}) - y\right)^2$$

Linear Regression Example: Final Results

	inputs $\mathbf{x}=(x_1,x_2)^\top$		target y		
	Lion King (original)	Aladin (Remake)	Lion King (Remake)	$f(\mathbf{x})$	$\lvert f(\mathbf{x}) - y \rvert$
1	0.70	0.41	0.70	0.70	0.00
2	0.86	0.50	0.85	0.83	0.02
3	0.45	0.58	0.58	0.58	0.00
4	0.48	0.71	0.64	0.64	0.00
5	0.58	0.82	0.73	0.73	0.00
6	0.78	0.88	0.88	0.88	0.00
7	0.69	0.50	0.71	0.72	0.01
8	0.94	0.57	0.89	0.90	0.01
9	0.67	0.81	0.79	0.79	0.00
10	0.78	0.95	0.91	0.90	0.01
11	0.61	0.64	0.72	0.70	0.02
12	0.64	0.92	0.80	0.80	0.00

Questions?

Linear Classifiers

What is a Linear Classifier?

- A learning algorithm for classification that makes it possible to learn a decision function $f(\boldsymbol{x})$ for a classification problem.
- If its decision function can be expressed in one of the following simple forms:
  - Binary Classifier: $y \in \{-1, +1\}$

$$f_\theta(\boldsymbol{x}) = \text{sign}(g(\boldsymbol{x}))$$

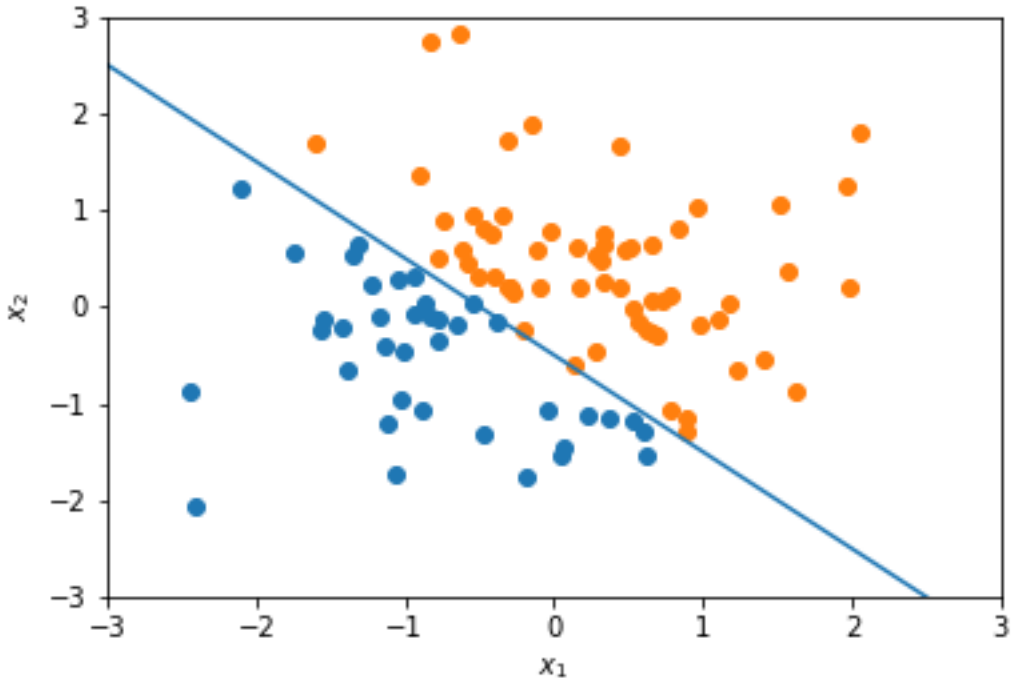
    where $g(x) = \mathbf{w}^\top \mathbf{x} + b \quad \theta = \{\mathbf{w}, \mathbf{b}\}$
  - Multi-class Classifier: $y \in \{1, 2, ..., m\}$

$$f_\theta(\boldsymbol{x}) = \arg\max(g(\boldsymbol{x})) = \arg\max(g_1(\boldsymbol{x}), g_2(\boldsymbol{x}), ..., g_m(\boldsymbol{x}))$$

    where g_i is the ith classifier.

Geometry of a linear discriminant function

- Decision regions: $R_+ = \{\mathbf{x}\colon g(\mathbf{x}) > 0\}$ and $R_- = \{\mathbf{x}\colon g(\mathbf{x}) < 0\}$
- Decision boundary: $H = \{\mathbf{x}\colon g(\mathbf{x}) = 0\}$
- Linear discriminant: $g(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$



Logistic Regression

- Can be use for binary classification task: $y \in \{0, 1\}$.
- Logistic regression algorithm

$$f_\theta(\mathbf{x}) = f_{b,\mathbf{w}}(\mathbf{x}) = \text{sigmoid}(\mathbf{w}^\top \mathbf{x} + b)$$

where

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

- The prediction $f_\theta(\mathbf{x})$ lies in the interval $(0, 1)$ and can be interpreted as $p(y = 1|\mathbf{x})$.
- Loss function for binary classification: **binary cross-entropy**

$$L(b, \mathbf{w}) = -\sum_{(\mathbf{x},y)\in D} \left[y \ln f_{b,\mathbf{w}}(\mathbf{x}) + (1 - y) \ln(1 - f_{b,\mathbf{w}}(\mathbf{x}))\right]$$

Logistic Regression Example

- Predict if the users of a streaming service are going to like a movie or not.

Logistic Regression Example: Data

- Predict if the users of a streaming service are going to like a movie or not.

	Stranger Things	Daredevil	Black Mirror
1	0	1	1
2	0	0	0
3	1	1	0
4	1	0	1
5	0	1	1
6	1	1	0
7	1	0	1
8	0	0	0
9	1	0	1
10	0	1	1
11	0	0	0
12	0	1	1

Table 6: Netflix shows score

Logistic Regression Example: Model

- Predict if the users of a streaming service are going to like a movie or not.

	inputs $\mathbf{x}=(x_1,x_2)^\top$		target y
	Stranger Things	Daredevil	Black Mirror
1	0	1	1
2	0	0	0
3	1	1	0
⋮	⋮	⋮	⋮

In this example, we want a model with the form

$$f(\mathbf{x}) = \text{sigmoid}\big(b + w_1 x_1 + w_2 x_2\big) = \text{sigmoid}\big(b + \mathbf{w}^\top \mathbf{x}\big)$$

Logistic Regression Example: Training

- Predict if the users of a streaming service are going to like a movie or not.

	inputs $\mathbf{x}=(x_1,x_2)^\top$		target y
	Stranger Things	Daredevil	Black Mirror
1	0	1	1
2	0	0	0
3	1	1	0
⋮	⋮	⋮	⋮

- Split the data in 2:

$$D^{(\text{train})} = \left\{ \left(\mathbf{x}^{(i)}, y^{(i)} \right) : i \in \{1, 2, ..., 9\}) \right\}$$
$$D^{(\text{test})} = \left\{ \left(\mathbf{x}^{(i)}, y^{(i)} \right) : i \in \{10, 11, 12\} \right\}$$

Logistic Regression Example: Training

- TOL $= 0.0001$
- $\alpha = 0.02$
- MAX_ITER $= 500$
- $i = 0$
- `while` $\max\left\{\left|\frac{\partial}{\partial b}L(b, \mathbf{w})\right|, \left|\frac{\partial}{\partial w_1}L(b, \mathbf{w})\right|, \left|\frac{\partial}{\partial w_2}L(b, \mathbf{w})\right|\right\} >$ TOL `do:`

  $b \leftarrow b - \alpha\frac{\partial}{\partial b}L(b, \mathbf{w})$

  $w_1 \leftarrow w_1 - \alpha\frac{\partial}{\partial w_1}L(b, \mathbf{w})$

  $w_2 \leftarrow w_2 - \alpha\frac{\partial}{\partial w_2}L(b, \mathbf{w})$

  `if` $i >$ MAX_ITER `do:`

    `break`
- Where $L(b, \mathbf{w}) = -\sum_{(\mathbf{x},y)\in D^{(\text{train})}} \left[y \ln f(\mathbf{x}) + (1 - y) \ln(1 - f(\mathbf{x}))\right]$

Logistic Regression Example: Results

	inputs $\mathbf{x}=(x_1,x_2)^\top$		target y	
	Stranger Things	Daredevil	Black Mirror	$f(\mathbf{x})$
1	0	1	1	0
2	0	0	0	0
3	1	1	0	1
4	1	0	1	1
5	0	1	1	0
6	1	1	0	1
7	1	0	1	1
8	0	0	0	0
10	0	1	1	0
11	0	0	0	0
12	0	1	1	0

Table 7: Predictions of the Black Mirror ratings using a logistic regression

Logistic Regression Example: Results

- Training accuracy: 56%
- Test accuracy: 33%
- What happened?

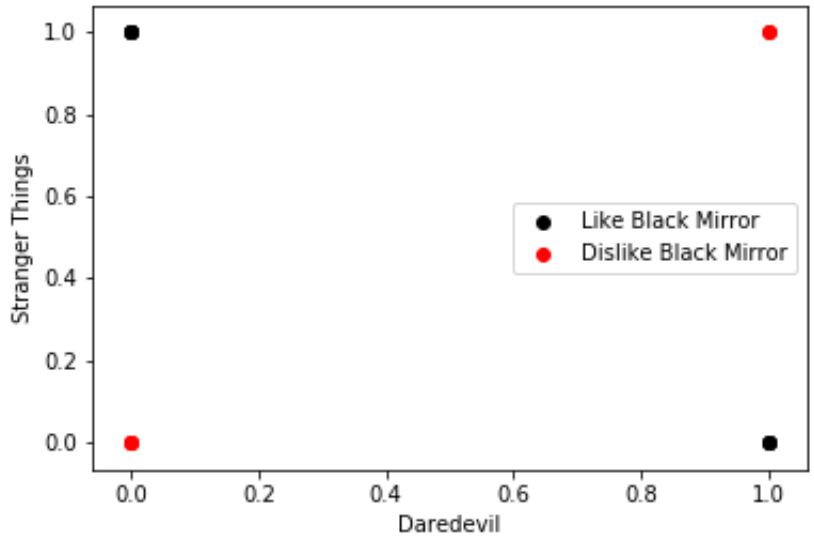
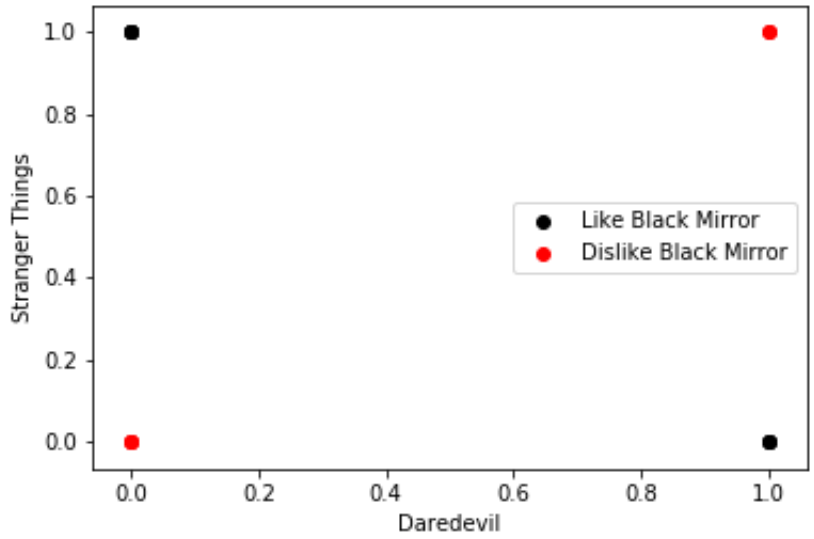


Figure 4: Data space for the Neflix example.

Logistic Regression Example: Results

- The data is not linearly separable, i.e. no straight line can separate the red and the black points.
- No linear classifier could work in this example...



Questions?

Beyond Linear Model

Multi-layer Perceptron (MLP)

- Sometimes called feedforward neural network or just neural network
- The multi-layer perceptron is a machine learning algorithm that can be use for classification and regression
- Because of it's hidden layers, it is possible to learn non-linear relation with a MLP

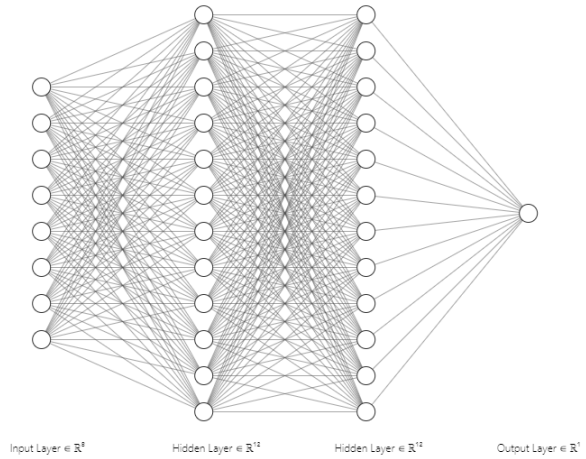
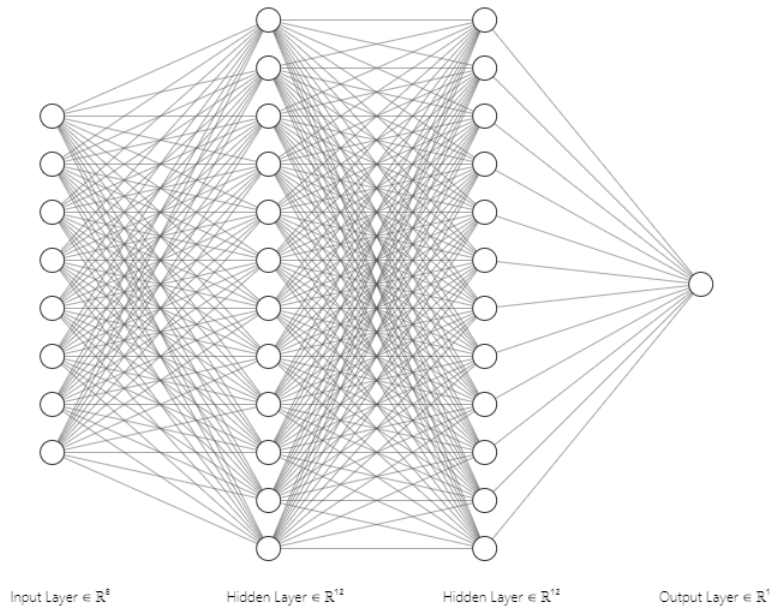


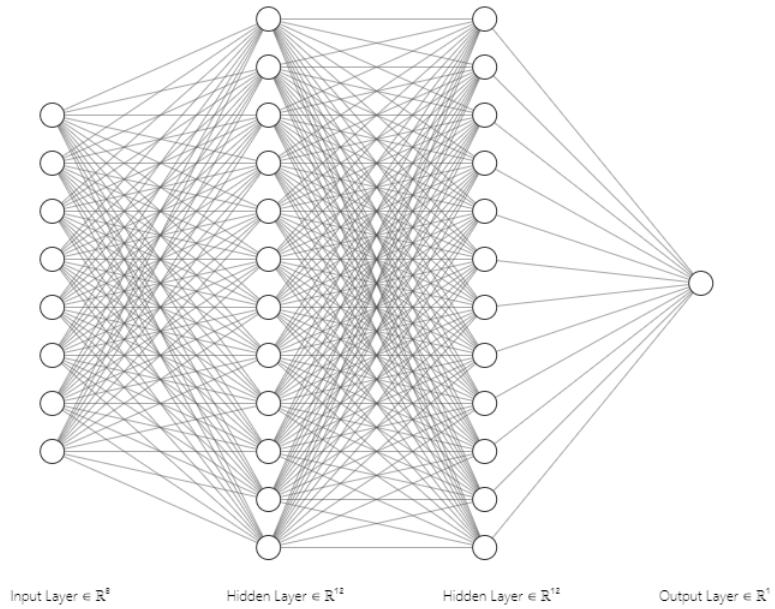
Figure 5: Representation of a MLP with 2 hidden layers.

Multi-layer Perceptron (MLP)



- The first and the last layer are the input and the output layer, respectively
- The 2 layers in the middle are called **hidden layers**
- An element of a hidden layer is called a **hidden unit** or neuron
- The number of hidden layers and their size are parameters that must be decide prior to the optimization. Such parameters are called **hyperparameters**.

Multi-layer Perceptron (MLP)



- $\mathbf{h}_1 = g_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$
- $\mathbf{h}_2 = g_2(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)$
- $o = g_3(\mathbf{W}_3\mathbf{h}_2 + b_3)$
- The functions g_1, g_2, and g_3 are called activation function. They allow the model to learn more complex relation between the input and the target and facilitate optimization.
- The set of parameters that needs to be optimize is $\theta = \{\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3, \mathbf{b}_1, \mathbf{b}_2, b_3\}$.

Activation Functions

- Activation functions allow the model to learn more complex relation between the input and the target.
- They are applied element wise on the hidden units. For example, let $\mathbf{a}_i$ be a L-dimention vector, and g be an activation function:

$$\mathbf{a}_i = (a_{11}, ..., a_{1L})^\top$$

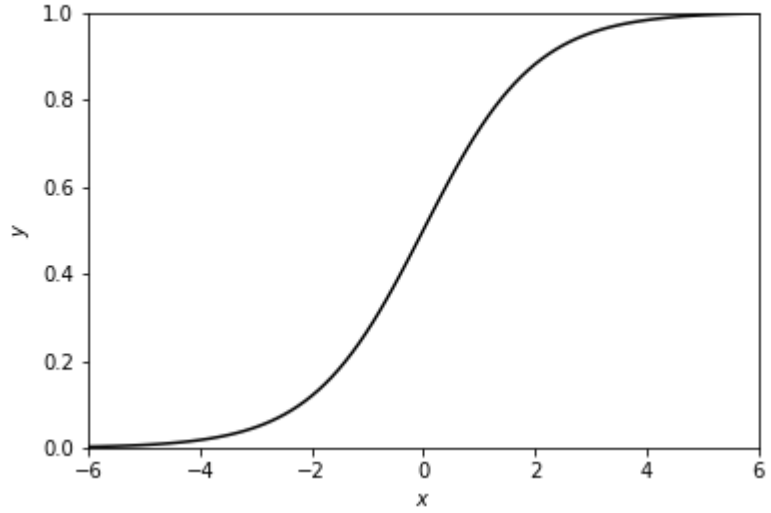
then

$$g(\mathbf{a}_i) = (g(a_{11}), ..., g(a_{1L}))^\top$$

- The simplest activation function is the identity function: $\text{Identity}(\mathbf{a}) = \mathbf{a}$

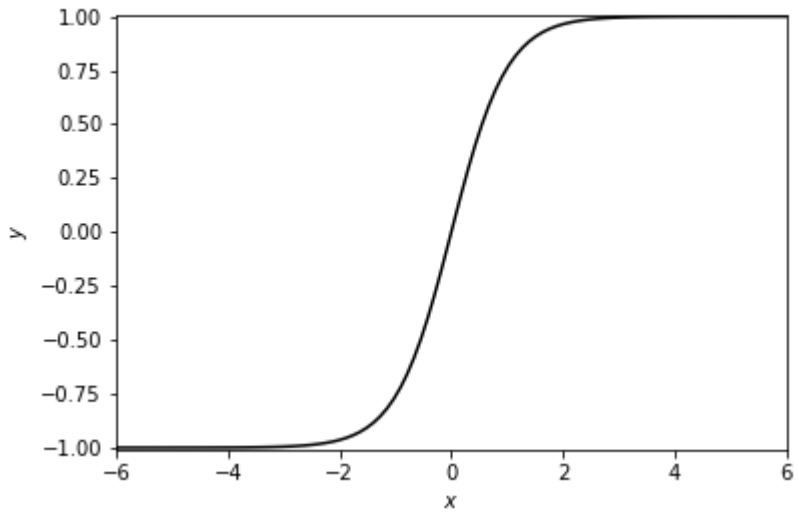
Activation Functions: Sigmoid

- $\text{sigmoid}(a) = \frac{1}{1+e^{-a}}$
- Squashes hidden units between 0 and 1.
- Not a good choice for middle layers. Why?
- Commonly used for the last layer activation function



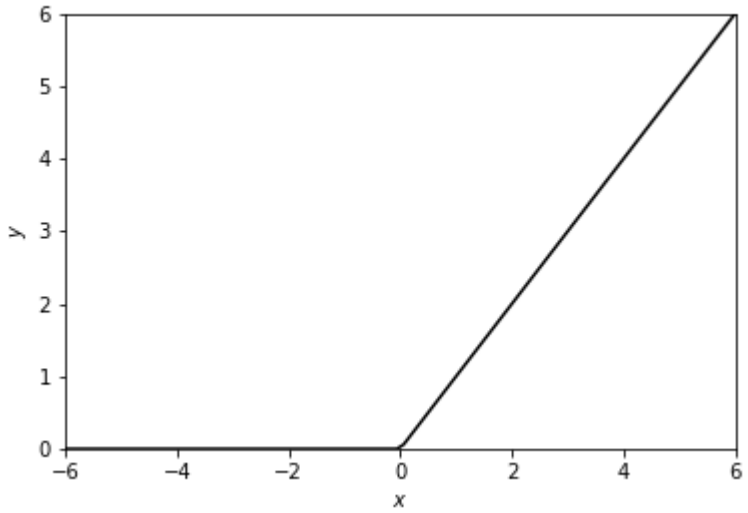
Activation Functions: Hyperbolic Tangent

- $\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} = \frac{e^{2a} - 1}{e^{2a} + 1}$
- Squashes hidden units between -1 and 1.
- Preferred over sigmoid



Activation Functions: Rectified Linear Unit (relu)

- $\text{relu}(a) = \max(0, a)$
- Popular for the middle layers activation function! why?



Activation Functions: Softmax

$$\text{softmax}(a) = \left(\frac{\exp(a_1)}{\exp \sum_j a_j}, ..., \frac{\exp(a_n)}{\exp \sum_j a_j} \right) \qquad a = (a_1, a_2, \cdots, a_n)$$

- Values between 0 and 1
- Used for the output layer of multi-class classification models
- Sum to one
- Predicted class= highest probability

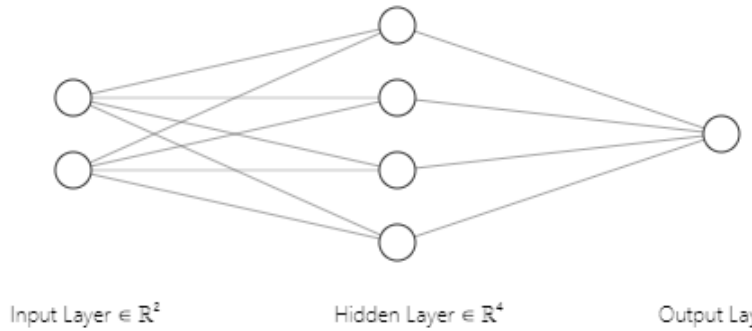
Choosing Activation Functions

- Neural networks are easier to optimize when the distribution of the hidden units are normalish (average of 0 and kind of symetric)
- The most used activation function for middle layers is relu, although it does not produce hidden units that are centered at 0.
- In deep neural network, a normalization methods called **batch normalization** is usually applied after relu to speed up learning.
- $\mathrm{softmax}$ and $\mathrm{sigmoid}$ are usually used on the output layer.

MLP Example 1: Model

- Let's try to solve the Netflix example using a MLP

	inputs $\mathbf{x}=(x_1,x_2)^\top$		target y
	Stranger Things	Daredevil	Black Mirror
1	0	1	1
2	0	0	0
⋮	⋮	⋮	⋮



$$\mathbf{h} = \text{relu}\left(\mathbf{W}\mathbf{x} + \mathbf{b}_1\right)$$

$$f(\mathbf{x}) = \text{sigmoid}\left(\mathbf{w}^\top \mathbf{h} + b_2\right)$$

MLP Example 1: Training

- $\text{TOL} = 0.0001$
- $\alpha = 0.02$
- $\text{MAX\_ITER} = 500$
- $i = 0$
- `while` $\max\left\{\left|\frac{\partial}{\partial\omega}L(\theta)\right| : \omega \in \theta\right\} > \text{TOL}$ `do:`

  $\mathbf{b}_1 \leftarrow \mathbf{b}_1 - \alpha\frac{\partial}{\partial\mathbf{b}_1}L(\theta)$

  $b_2 \leftarrow b_2 - \alpha\frac{\partial}{\partial b_2}L(\theta)$

  $\mathbf{W} \leftarrow \mathbf{W} - \alpha\frac{\partial}{\partial\mathbf{W}}L(\theta)$

  $\mathbf{w} \leftarrow \mathbf{w} - \alpha\frac{\partial}{\partial\mathbf{w}}L(\theta)$

  `if` $i > \text{MAX\_ITER}$ `do:`

    `break`
- Where

$$L(\theta) = -\sum_{(\mathbf{x},y)\in D^{(\text{train})}} [y \ln f_\theta(\mathbf{x}) + (1-y)\ln(1-f_\theta(\mathbf{x}))]$$

Questions?

MLP Example 1: Results

	inputs $\mathbf{x}=(x_1,x_2)^\top$		target y	
	Stranger Things	Daredevil	Black Mirror	$f(\mathbf{x})$
1	0	1	1	1
2	0	0	0	0
3	1	1	0	0
4	1	0	1	1
5	0	1	1	1
6	1	1	0	0
7	1	0	1	1
8	0	0	0	0
9	1	0	1	1
10	0	1	1	1
11	0	0	0	0
12	0	1	1	1

MLP Example 2: Data

- Predict the score on movie C using the scores on movie A and B.

	inputs $\mathbf{x}=(x_1,x_2)^\top$		target y
	Movie A	Movie B	Movie C
1	0.70	0.41	0.21
2	0.86	0.50	0.47
3	0.45	0.58	0.27
4	0.48	0.71	0.37
5	0.58	0.82	0.49
6	0.78	0.88	0.59
7	0.69	0.50	0.38
8	0.94	0.57	0.61
9	0.67	0.81	0.52
10	0.78	0.95	0.72
11	0.61	0.64	0.51
12	0.64	0.92	0.64

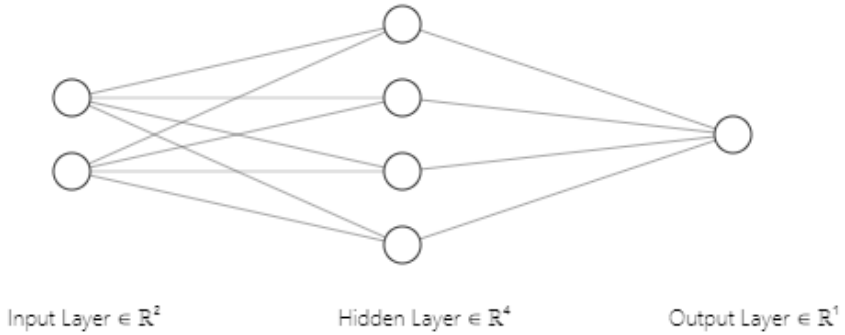
MLP Example 2: Baseline

	inputs $\mathbf{x}=(x_1,x_2)^\top$		target y
	Movie A	Movie B	Movie C
1	0.70	0.41	0.21
2	0.86	0.50	0.47
⋮	⋮	⋮	⋮

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$

MLP Example 2: Model

	inputs $\mathbf{x}=(x_1,x_2)^\top$		target y
	Movie A	Movie B	Movie C
1	0.70	0.41	0.21
2	0.86	0.50	0.47
⋮	⋮	⋮	⋮



$$\mathbf{h} = \text{relu}\left(\mathbf{W}\mathbf{x} + \mathbf{b}_1\right)$$

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{h} + b_2$$

MLP Example 2: Results

	Absolute error	
	Linear regression	Multi-layer perceptron
1	0.17	0.09
2	0.03	0.01
3	0.02	0.02
4	0.03	0.02
5	0.06	0.00
6	0.03	0.04
7	0.02	0.03
8	0.05	0.06
9	0.04	0.01
10	0.14	0.05
11	0.11	0.12
12	0.15	0.06

MLP Example 2: Results

	Linear regression	Multi-layer perceptron
Training MAE	0.050	0.029
Test MAE	0.135	0.077

- We can see that the performance on the training set is significantly better then the performance on the test set.
- This situation is know as **overfitting** in machine learning.

Overfitting, Capacity, and Regularization

Parameters v.s. Hyperparameters

- **Parameters** are optimized on the training set(learned by the algorithm).
  - Example: Coefficients b, w_1, and w_2 in a linear regression:

$$f(x) = b + w_1 x_1 + w_2 x_2$$

- **Hyper-parameters** are fixed by the user generally before training.
  - Example: learning rate, number of hidden layers in a neural network

Capacity

- **Model Capacity** controls the model ability to fit wide variety of functions.
  - Example: neural network have more capacity then linear models
- **Control capacity**
  - The choice of the function set and the hyper-parameters of the algorithms make it possible to control the capacity of the model.
  - Example: reducing the number of hidden layers in a neural network reduces it's capacity
- More capacity means ability to learn more complex relations in the data
- More capacity also means more risk of overfitting

Overfitting: "Memorizing" Training Data rather than "Learning"

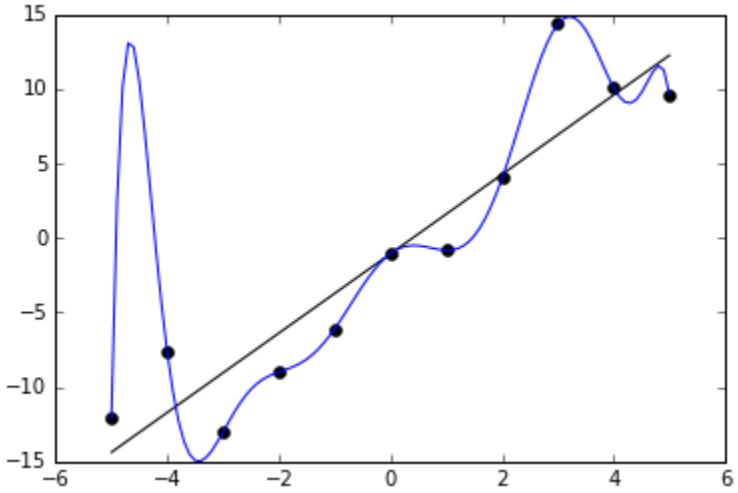


Figure 6: Noisy (roughly linear) data is fitted to a linear function and a polynomial function. Although the polynomial function is a perfect fit, the linear function can be expected to generalize better: if the two functions were used to extrapolate beyond the fitted data, the linear function should make better predictions. Source: By Ghiles - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=47471056

Overfitting: Overtraining

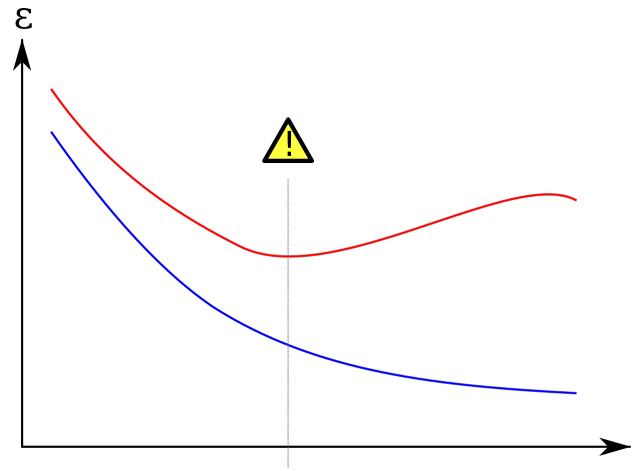


Figure 7: Training error is shown in blue, validation error in red, both as a function of the number of training iterations. If the validation error increases (positive slope) while the training error steadily decreases (negative slope) then a situation of overfitting may have occurred. The best predictive and fitted model would be where the validation error has its global minimum. Source: By Gringer - Own work, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=2959742

Regularization

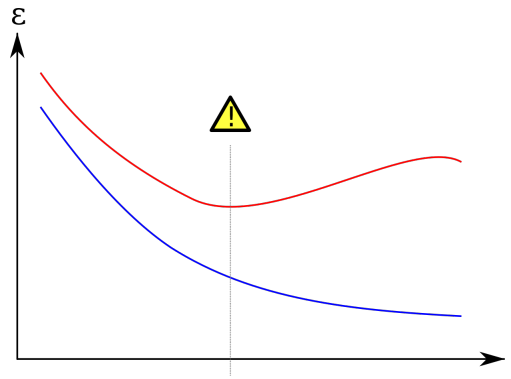
- A central problem in machine learning is how to make an algorithm that will perform well not just on the training data, but also on new inputs.
- **Regularization:** strategies used in machine learning that are explicitly designed to reduce the test error, possibly at the expense of increased training error.

Early Stopping

- Split the data in 3 subset instead of 2:

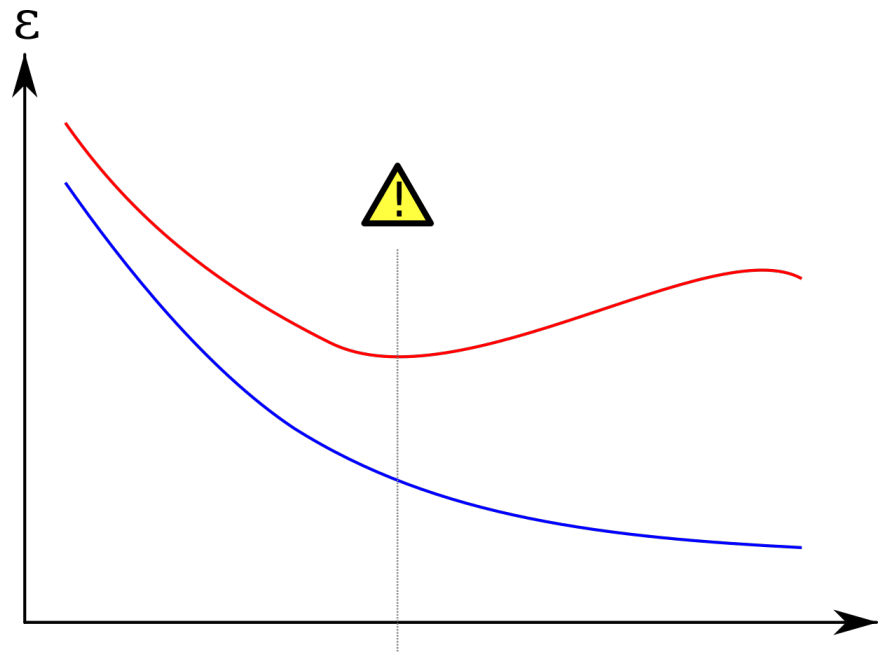
$$
\begin{aligned}
D^{(\text{train})} &= \left\{ \left(\mathbf{x}^{(i)}, y^{(i)} \right) : i \in \text{train\_indices} \right) \right\} \\
D^{(\text{valid})} &= \left\{ \left(\mathbf{x}^{(i)}, y^{(i)} \right) : i \in \text{valid\_indices} \right) \right\} \\
D^{(\text{test})} &= \left\{ \left(\mathbf{x}^{(i)}, y^{(i)} \right) : i \in \text{test\_indices} \right) \right\}
\end{aligned}
$$

- Use the model model with the validation set error



Early Stopping

- In practice, save the parameters each time the model improve. Load the parameters when training is completed.



L_1 and L_2 Regularization

- Regularized empirical loss

$$L_\lambda(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(f_\theta(x^{(i)}), y^{(i)}) + \lambda\Omega(\theta)$$

- L_2 regularization (or weight decay) adds a squared penalty on the weights:

$$\Omega(\theta) = \Omega(\boldsymbol{w}, b) = ||\boldsymbol{w}||_2^2 = \sum_{j=1}^{d} w_j^2$$

- L_1 regularization

$$\Omega(\theta) = \Omega(\boldsymbol{w}, b) = ||\boldsymbol{w}||_1 = \sum_{j=1}^{d} |w_j|$$

L_1 and L_2 Regularization

- Linear regression with L_2 penalty is sometimes called **Ridge regression**
- Linear regression with L_1 penalty is sometimes called **Lasso regression**.

Regularization

What is the best regularizer?

MORE DATA

Questions?

Back to the MLP Regression Example

MLP Example 2: New Data Split

	inputs $\mathbf{x}=(x_1,x_2)^\top$		target y
	Movie A	Movie B	Movie C
1	0.70	0.41	0.21
2	0.86	0.50	0.47
3	0.45	0.58	0.27
4	0.48	0.71	0.37
5	0.58	0.82	0.49
6	0.78	0.88	0.59
7	0.69	0.50	0.38
8	0.94	0.57	0.61
9	0.67	0.81	0.52
10	0.78	0.95	0.72
11	0.61	0.64	0.51
12	0.64	0.92	0.64

MLP Example 2: New Data Split

- Split the data in 3 subsets instead of 2:

$$
\begin{aligned}
D^{(\text{train})} &= \left\{ \left(\mathbf{x}^{(i)}, y^{(i)} \right) : i \in \{1, 2, 3, 4\}) \right\} \\
D^{(\text{valid})} &= \left\{ \left(\mathbf{x}^{(i)}, y^{(i)} \right) : i \in \{5, 6, 7, 8\}) \right\} \\
D^{(\text{test})} &= \left\{ \left(\mathbf{x}^{(i)}, y^{(i)} \right) : i \in \{9, 10, 11, 12\} \right\}
\end{aligned}
$$

MLP Example 2: Learning Curve

- Remember the gradient descent algorithm:

  While $\left|\nabla_\theta L(\theta)\right| > \text{Tol}$:

$$\theta \leftarrow \theta - \alpha \nabla L(\theta)$$

- At each iteration, we can compute the training and the validation loss:

$$\text{loss}_{\text{train}} = \sum_{(\mathbf{x},y)\in D^{(\text{train})}} \big(f(\mathbf{x}) - y\big)^2$$

$$\text{loss}_{\text{valid}} = \sum_{(\mathbf{x},y)\in D^{(\text{valid})}} \big(f(\mathbf{x}) - y\big)^2$$

MLP Example 2: Learning Curve

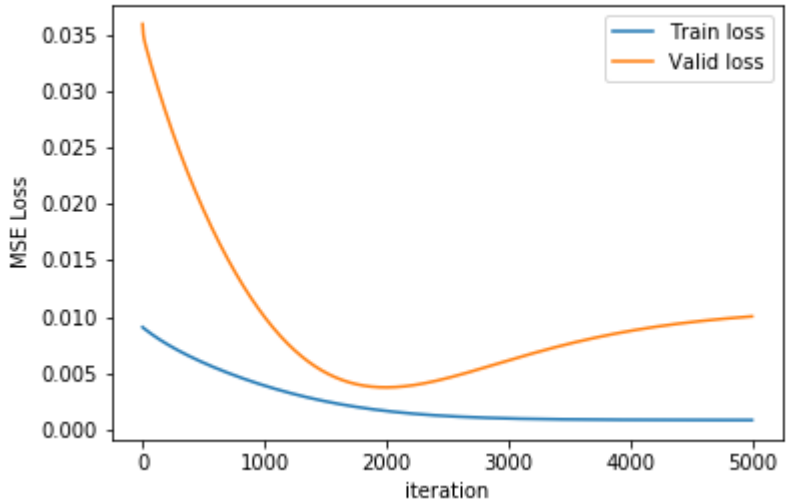


Figure 8: MLP learning curve

- We can see that after about 2000 iterations, the validation loss starts increasing while the training loss continue to decrease. This is a typical overfitting situation.

MLP Example 2: Training with Early Stopping

- $\alpha = 0.02$
- $\text{MAX\_ITER} = 500$
- $i = 0$
- $\text{loss}^*_{\text{valid}} = \infty$
- `for` $i = 1$ `to` MAX_ITER `do:`

  `for` $\omega \in \theta$ `do:`

  $\omega \leftarrow \omega - \alpha \frac{\partial}{\partial \omega} L(\theta)$

  $\text{loss}_{\text{valid}} = \sum_{(\mathbf{x},y) \in D^{(\text{valid})}} \left(f(\mathbf{x}) - y \right)^2$

  `if` $\text{loss}_{\text{valid}} < \text{loss}^*_{\text{valid}}$ `do:`

  $\text{loss}^*_{\text{valid}} = \text{loss}_{\text{valid}}$

  $\theta^* = \theta$

MLP example 2: Final Results

	Linear regression	MLP	MLP (Early Stopping)
Training MAE	0.050	0.027	0.03
Validation MAE	N/A	0.083	0.049
Test MAE	0.135	0.088	0.058

- Note that using early stopping with linear model usually lead to a loss in performance.

MLP: Conclusion

- Universal approximation theorem (Hornik, 1991): "A single hidden layer neural network (MLP) with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units"
- Neural network can capture non-linear relation between the input and the target; linear model only capture linear relation.
- Should we always use neural network? Absolutely not!
  - Always compare with simpler model.
  - Often, more data is require to capture non-linear dependencies.
- The more noise in your data, the more risk of overfitting. When working with noisy data, you should collect has much data has possible.

How much data we need to use neural networks?

Questions?

The Bias-Variance Trade-off

Bias-Variance Decomposition of Squared Error

- For a model f and a test sample (x, y)

$$E\big[(y - f(x))^2\big] = \mathrm{Bias}\big[f(x)\big]^2 + \mathrm{Var}\big[f(x)\big] + \sigma_\epsilon^2$$

where

$$\mathrm{Bias}\big[f(x)\big] = E\big[f(x)\big] - y$$

and

$$\mathrm{Var}\big[f(x)\big] = E\big[f(x)^2\big] - E\big[f(x)\big]^2$$

Bias-Variance Trade-off[2]

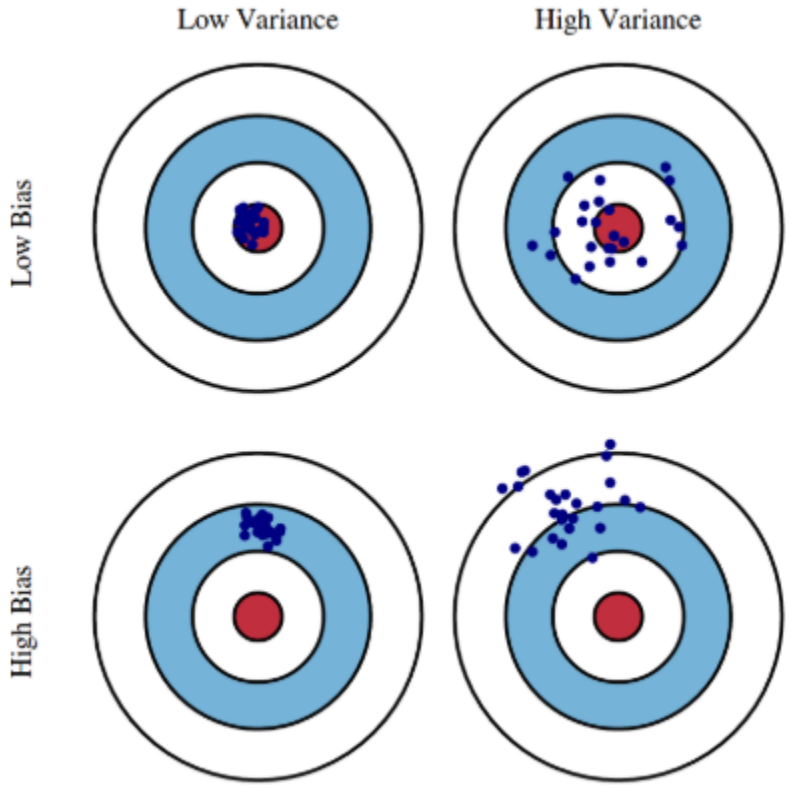


Figure 9: Graphical illustration of bias and variance.

[2]Source: http://scott.fortmann-roe.com/docs/BiasVariance.html

Detecting High Variance Situation

- Let's say you have 10000 observations (x, y) to develop a supervised algorithm (e.g. classification)
- Keep 2000 observations to test your final models. Let's called this subset $D^{(\text{test})}$.
- Randomly split the remaining 8000 in two sets $D_1^{(\text{train})}$, $D_1^{(\text{valid})}$
  - Train a model and obtain a first estimator f_1
- Randomly split the remaining 8000 in two new sets $D_2^{(\text{train})}$, $D_2^{(\text{valid})}$
  - Train a second model and obtain a second estimator f_2
- If the prediction of f_1 on $D^{(\text{test})}$ are significantly different then the prediction of f_2, the variance is high.

Dealing with High Variance Situation

- Is it possible to collect more data?
- Do you really need a neural network?
- Try reducing the capacity (e.g. reduce the size of the hidden layer)
- Try regularization methods (early stopping, L_1 and L_2 regularization,...)

High Bias Situation

- Training error and test error are high
- Example
  - Using a linear classifier on a non-linearly separable data (the Netflix example)
  - Using linear regression when there is almost no linear dependence between x and y

Hyperparameter Search

Hyperparameter Search

- Using a training set only

  Training set $D^{(\text{train})}$
  Set of hyperparameter Λ
  `for` $\lambda \in \Lambda$ `do:`
    Optimize θ on $D^{(\text{train})}$
    `train_loss` $= L_\lambda(D^{(\text{train})}; \theta)$
    Save results (`train_loss`, λ)

- Final model: f_{λ^*} where λ^* is the set of hyperparameters that yielded the best training error.

Hyperparameter Search

- Using a training set and a validation set

  Training set $D^{(\text{train})}$
  Validation set $D^{(\text{valid})}$
  Set of hyperparameter Λ
  `for` $\lambda \in \Lambda$ `do:`
    Optimize θ on $D^{(\text{train})}$
    `valid_loss` $= L_\lambda(D^{(\text{valid})}; \theta)$
    Save results (`valid_loss`, λ)

- Final model: f_{λ^*} where λ^* is the set of hyperparameters that yielded the best validation error.
- WARNING: Never use the test set to select the hyperparameters, else it is no longer a test set.

Questions?

Clustering

Clustering

- **Clustering** is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters)

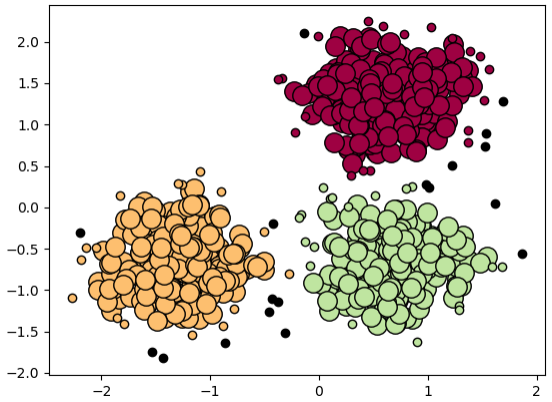


Figure 10: Illustration of the clustering of a dataset into 3 clusters. Source: https://scikit-learn.org/stable/modules/clustering.html#k-means

k-Means Clustering

- Given a set of observations $(x_1, ..., x_n)$, where each observation is a d dimensional real vector, k-means clustering aims to partition the n observations into k sets $\mathbf{S} = \{S_1, ..., S_k\}$ so as to minimize the within-cluster sum of squares. Formally, the objective is to find:

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} ||\mathbf{x} - \mu_i||^2$$

  where μ_i is the mean of points in the cluster S_i.

k-Means Clustering: Lloyd's Algorithm

1) Initialize the centroids μ_i by randomly choosing k observations
2) Assign every data point to its nearest cluster
3) Update the centroids based on the (possibly) new cluster

$$\mu_i = \frac{1}{n} \sum_{\mathbf{x} \in S_i} \mathbf{x}$$

- Repeat 2-3 until convergence, i.e. until all the clusters no longer change
- Interactive example: *link*

Questions?

Key Takeaways

- You should always test simple models (e.g. linear regression, logistic regression) at least to have a baseline.
- If you use large neural networks, you should split your data in 3 and visualize the learning curve / use early stopping.
- If you do a hyperparameter search, don't use the test set to select the hyperparameters, else it is no longer a test set.
- A good understanding of bias and variance is critical for developing reliable machine learning algorithm.