Recap

- Linear regression
  - Univariate: single feature $x \in R$
  - Multivariate: multiple features $\boldsymbol{x} \in R^m$
  - Linear transformation: $\tilde{y} = \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}$
  - Loss: measure the difference between the prediction and ground truth
  - Training is to optimize (i.e., minimize) the loss w.r.t parameters (**w**)
- Gradient descent algorithm
  - Minimize the target loss iteratively; for each iteration,
  - Compute the gradient of the average loss (over all training examples) w.r.t **w**
  - Update **w** in the **opposite** of the gradient direction

$$\boldsymbol{w} = \boldsymbol{w} - \alpha \frac{\partial J}{\partial \boldsymbol{w}}$$

Slide credit: Wang Wei

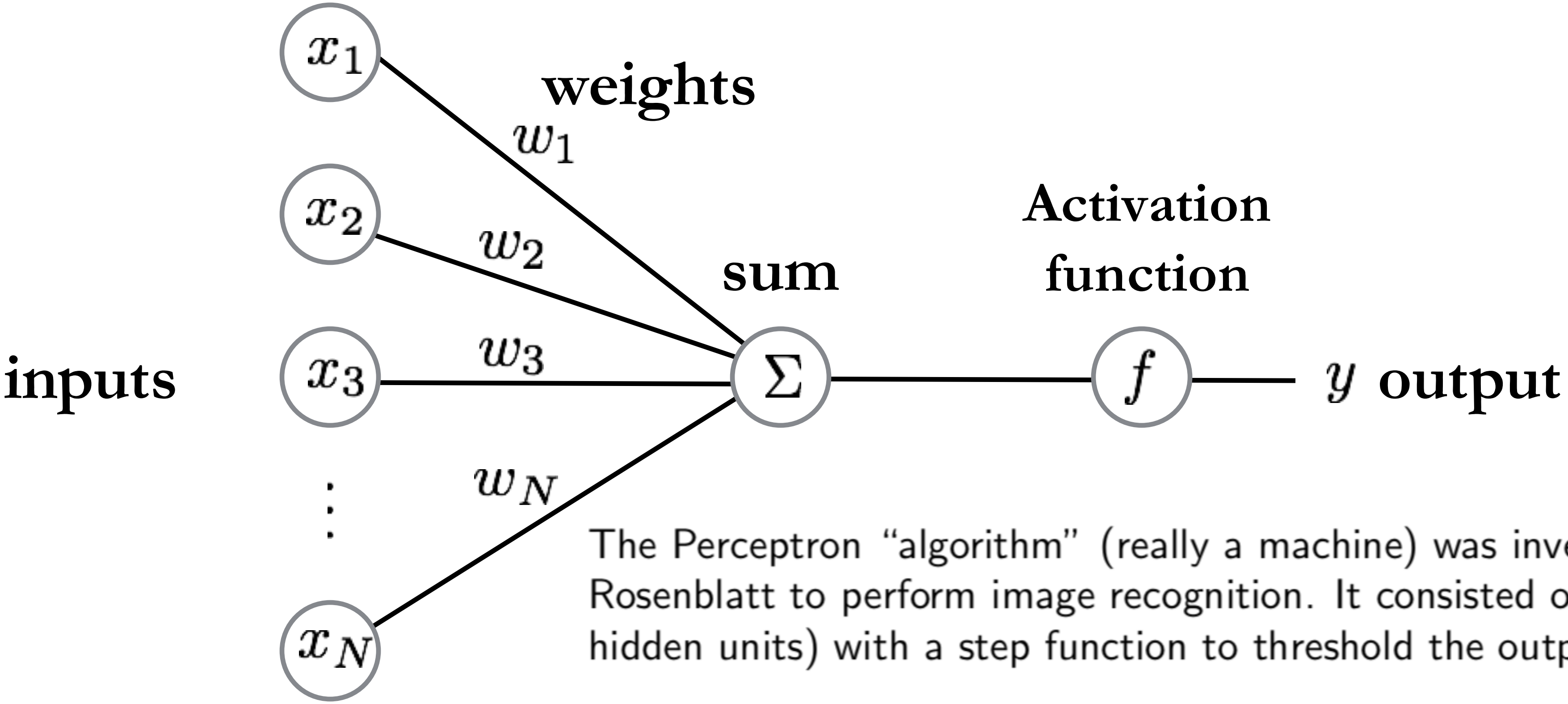
Today's Lecture

- Perceptrons
  - Training perceptrons with back-propagation
  - Multi-layer perceptrons
- Regression
  - Polynomial regression
    - Overfitting and underfitting
    - Dataset splitting for hyper-parameter tuning
- Classification
  - Logistic regression
    - binary cross-entropy
  - Multinomial regression (Softmax regression)

Perceptrons

single perceptrons, multi-layer perceptrons

The Perceptron



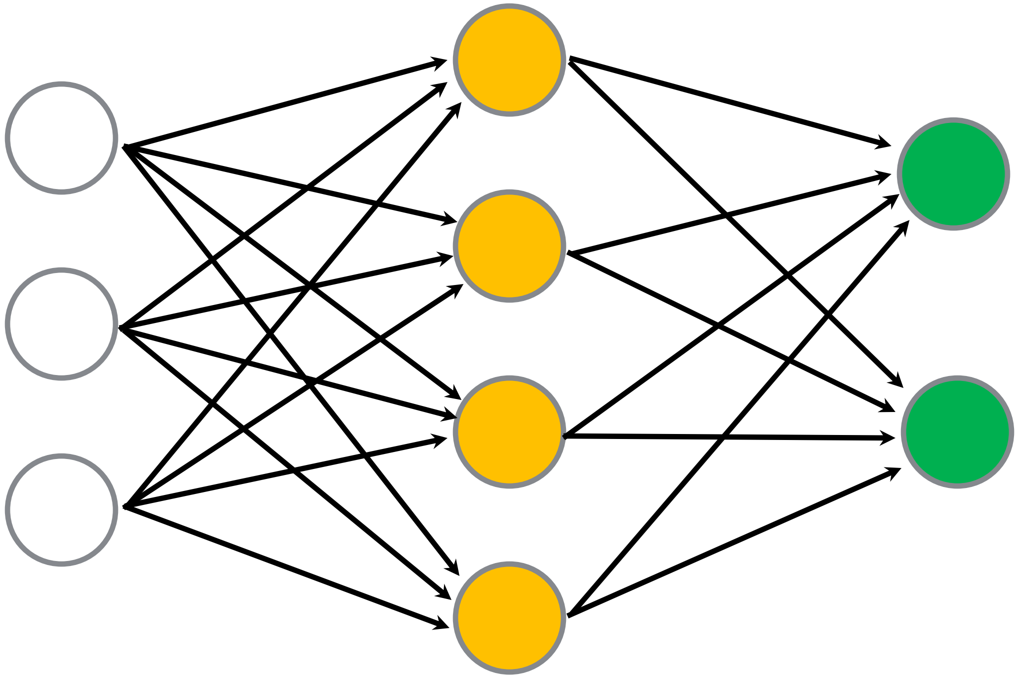
The Perceptron "algorithm" (really a machine) was invented in 1957 by Frank Rosenblatt to perform image recognition. It consisted of a single layer (*i.e.* no hidden units) with a step function to threshold the output:

$$y(\boldsymbol{x}, \boldsymbol{w}, \theta) = \begin{cases} 1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} + b > \theta \\ 0 & \text{otherwise} \end{cases}$$

Activation function used in original perceptron. More on this in later lectures.

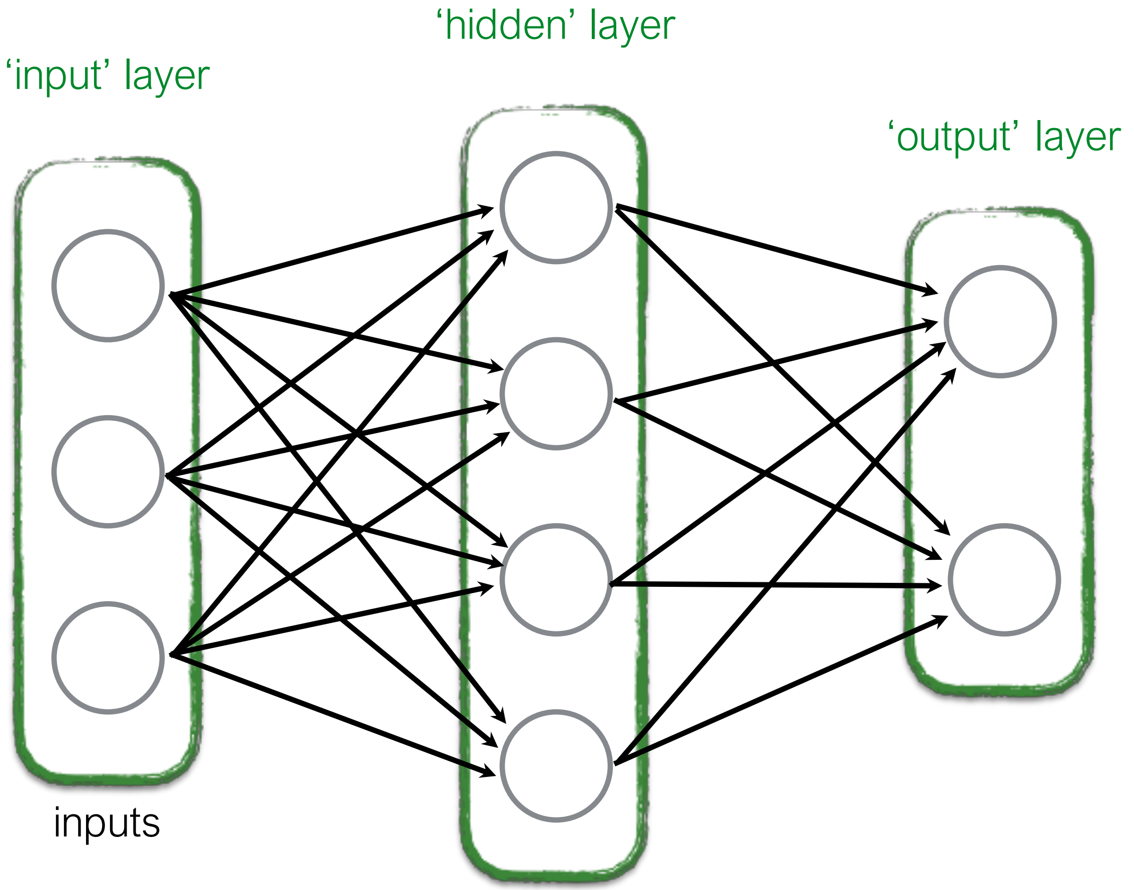
A Neural Network is simply

connecting a bunch of perceptrons together …



Slide Credit: K. Kitani

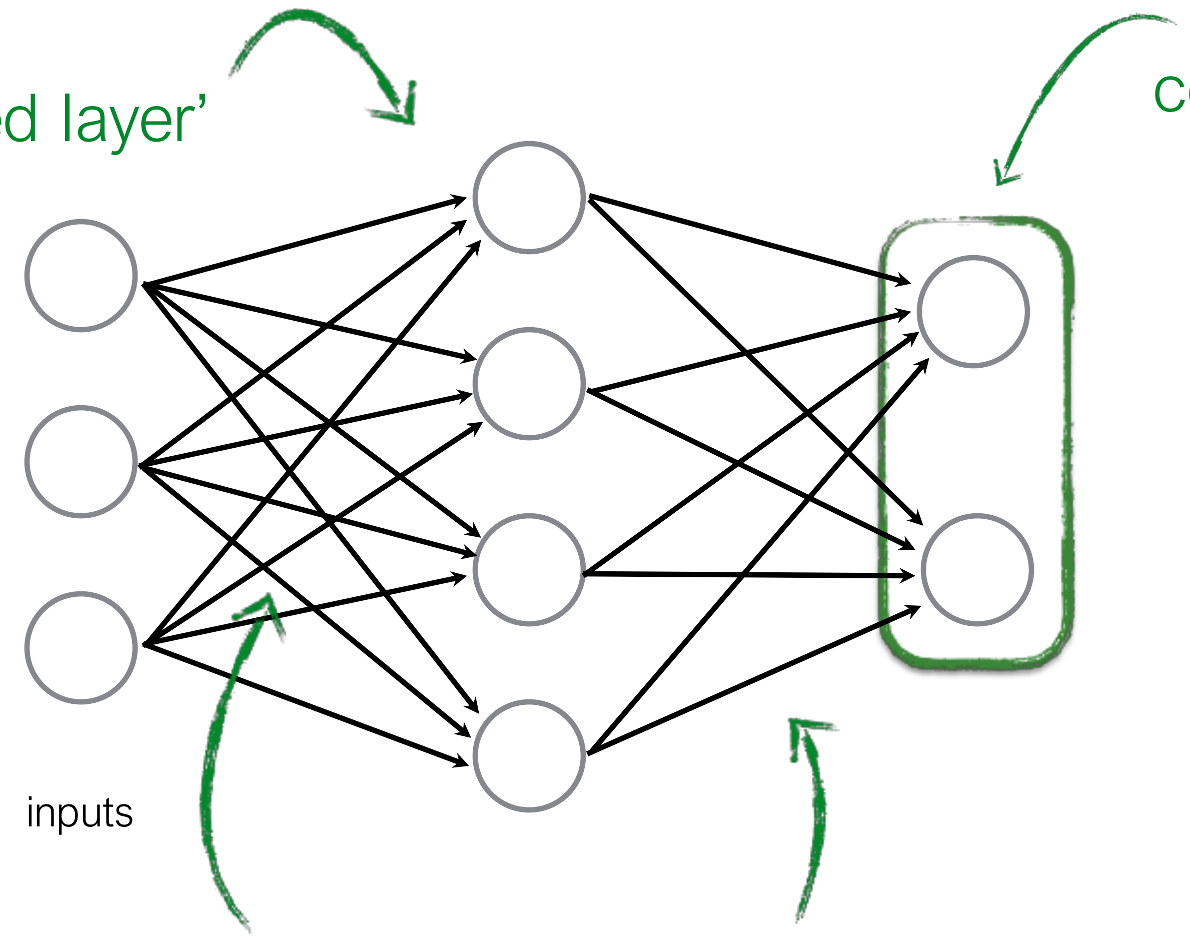
Neural Network Terminology



...also called a **Multi-layer Perceptron** (MLP)

Slide Credit: K. Kitani

Neural Network Terminology



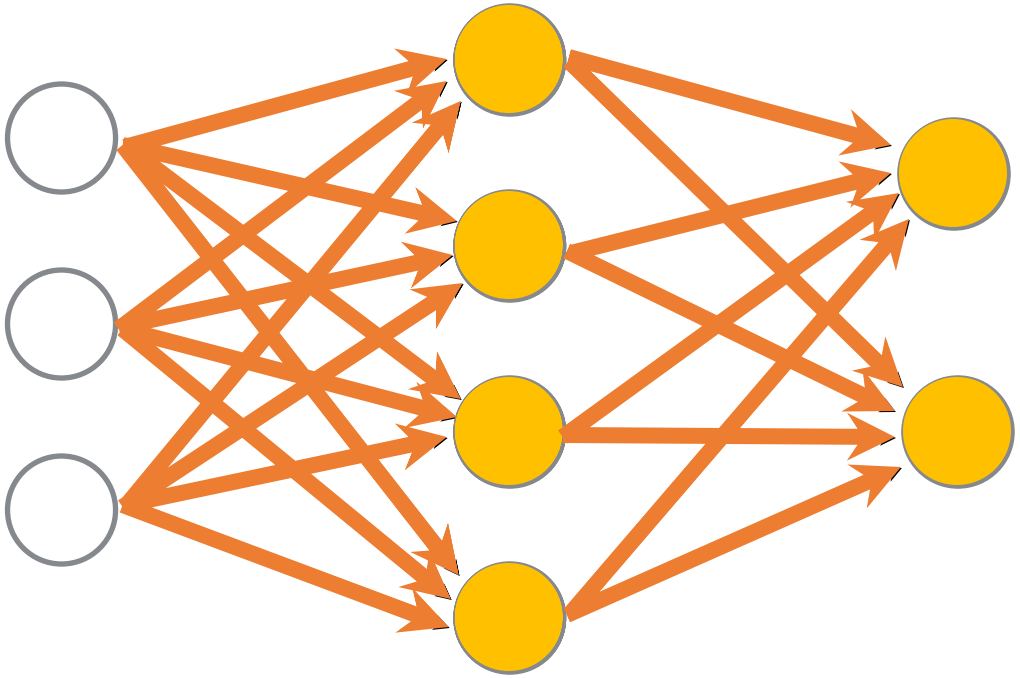
Slide Credit: K. Kitani

How many neurons (perceptrons)?

$4 + 2 = 6$

How many weights (edges)?

$(3 \times 4) + (4 \times 2) = 20$



How many learnable parameters total?

$20 + (4 + 2) = 26$

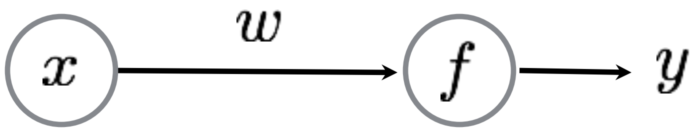
6 bias terms
1 per perceptrons

Slide Credit: K. Kitani

Training Perceptrons

Partial derivatives, gradient descent, back-propagation.

world’s smallest perceptron!



$$y = wx$$

(a.k.a. line equation, linear regression)

Slide Credit: K. Kitani

Learning a Perceptron

Given a set of samples and a Perceptron

$$\{x_i, y_i\}$$

$$y = f_{\text{PER}}(x; w)$$

what is this activation function?

linear function! $f(x) = wx$

Estimate the parameter of the Perceptron

$$w$$

Slide Credit: K. Kitani

An Incremental Learning Strategy

(gradient descent)

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

and a perceptron

$$\hat{y} = wx$$

Modify weight w such that $\hat{y}$ gets **'closer'** to y

perceptron parameter

perceptron output

what does this mean?

true label

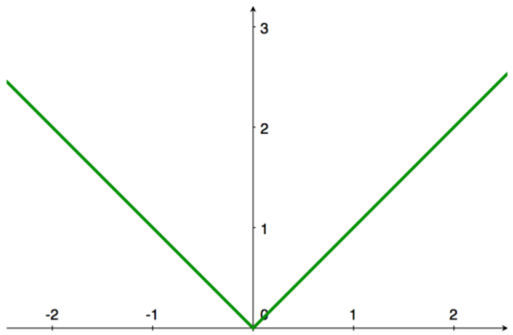
Slide Credit: K. Kitani

Loss Function
defines what it means to be **close** to the true solution

YOU get to chose the loss function!
(some are better than others depending on what you want to do)

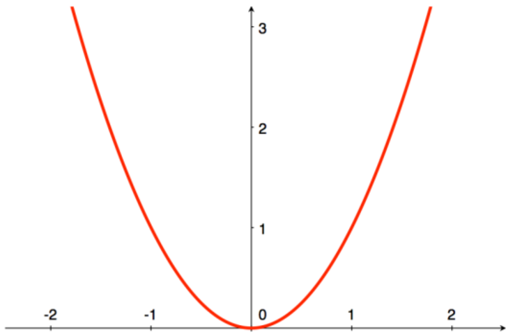
L1 Loss

$$\ell(\hat{y}, y) = |\hat{y} - y|$$



L2 Loss

$$\ell(\hat{y}, y) = (\hat{y} - y)^2$$

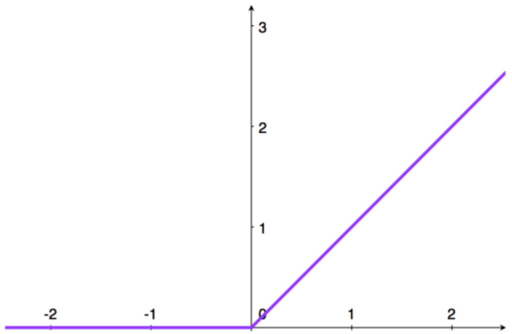


Zero-One Loss

$$\ell(\hat{y}, y) = \mathbf{1}[\hat{y} \neq y]$$

Hinge Loss

$$\ell(\hat{y}, y) = \max(0, 1 - y \cdot \hat{y})$$



Slide Credit: K. Kitani

Code to train your perceptron

for $n = 1 \ldots N$:

$$w = w + \underline{(y_n - \hat{y})x_n}$$

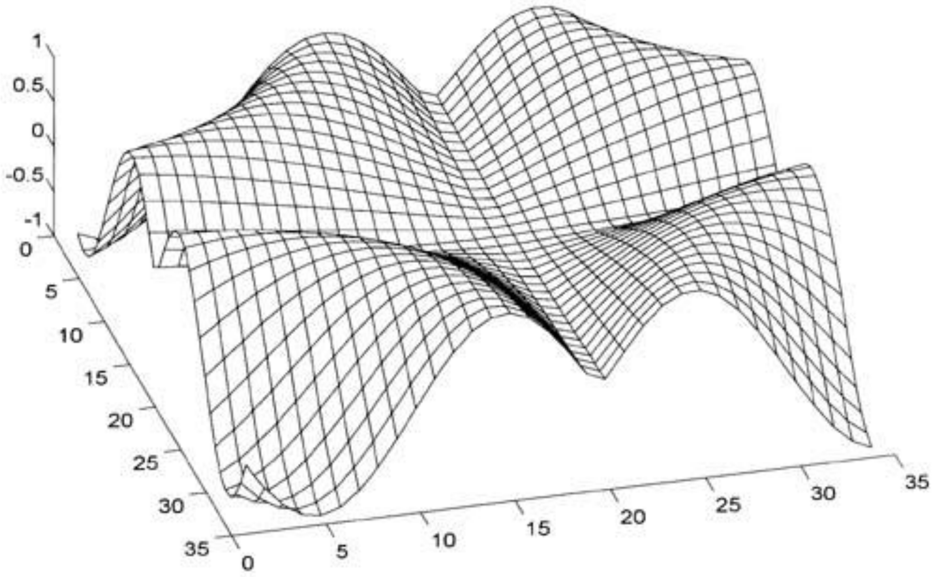
Just 2 lines!?
How can this be?

Slide Credit: K. Kitani

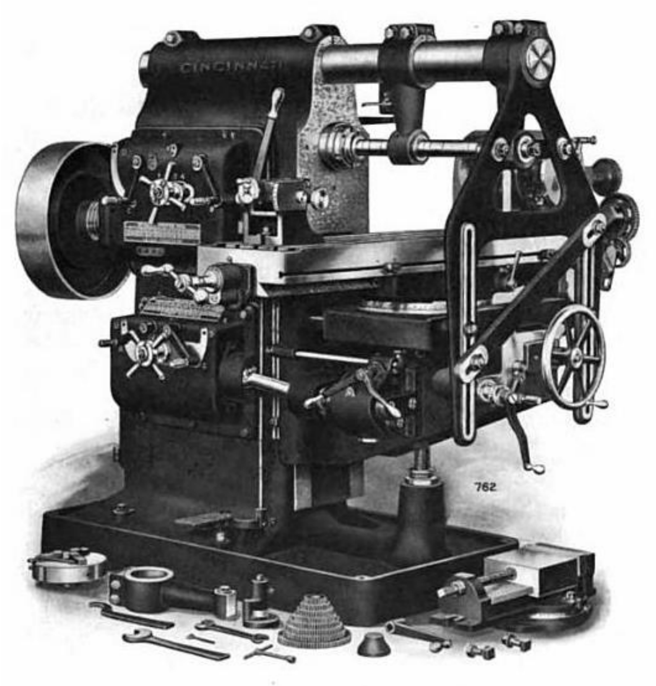
(Partial) Derivatives

tell us how much one variable affects another

Two ways to think about them:



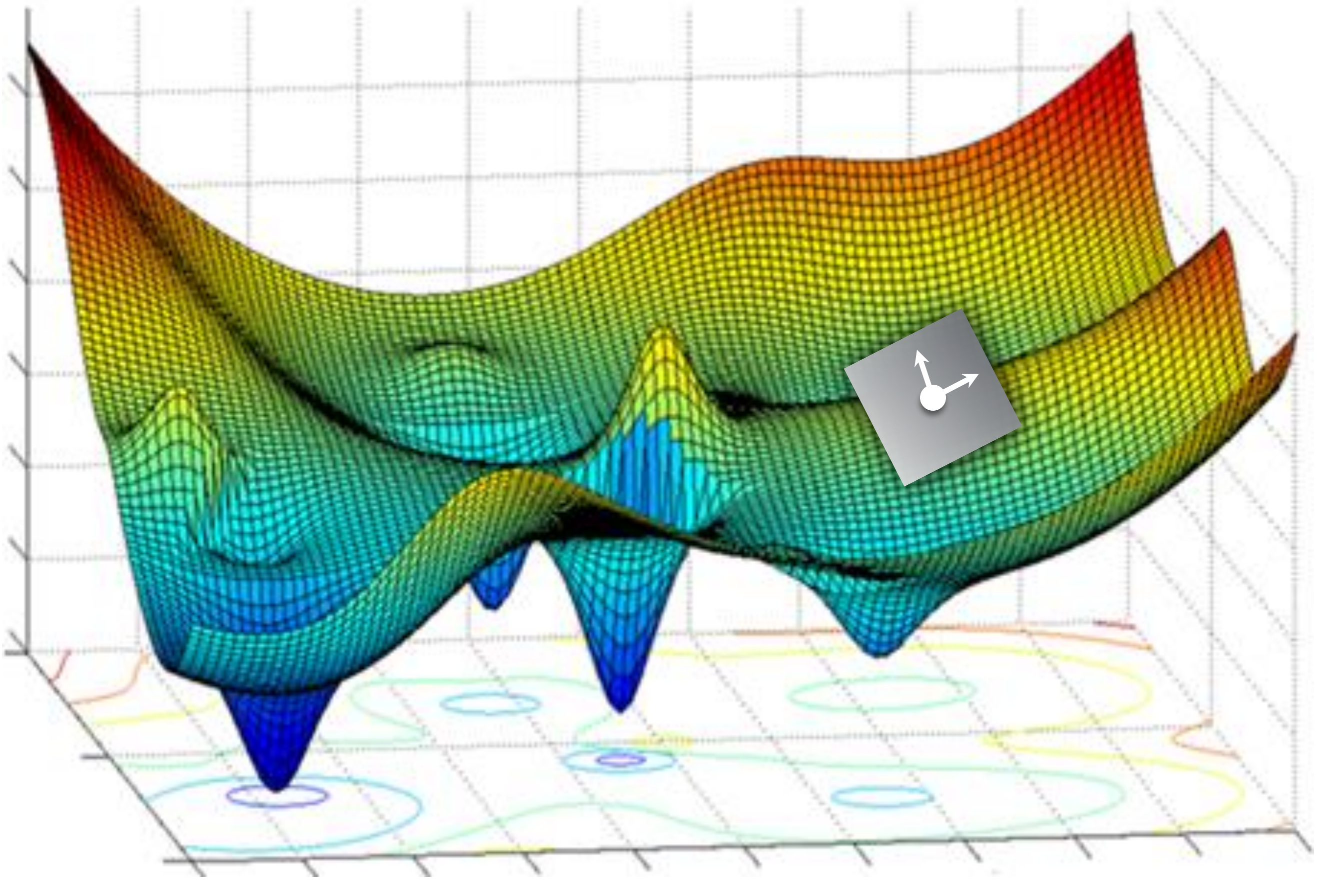
Slope of a function



Knobs on a machine

Slide Credit: K. Kitani

1. Slope of a function:



$$\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} = \left[\frac{\partial f(\boldsymbol{x})}{\partial x}, \frac{\partial f(\boldsymbol{x})}{\partial y}\right]$$

The slope around a point

Slide Credit: K. Kitani

2. Knobs on a machine:

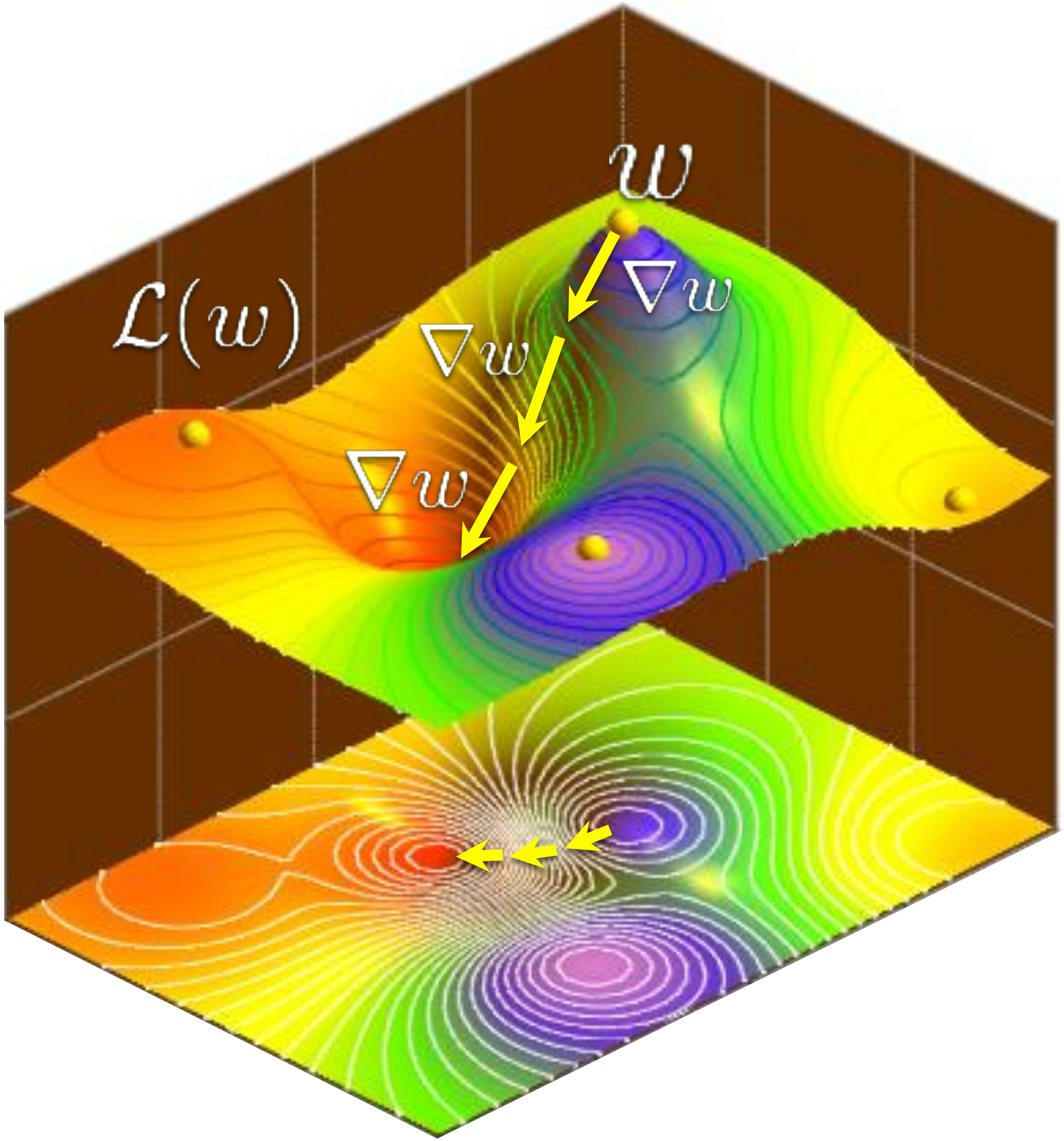
input $\boldsymbol{x}$ ➡

➡ output $f(\boldsymbol{x};\boldsymbol{w})$

describes how each 'knob' affects the output $\frac{\partial f(x)}{\partial w_1}$ $\frac{\partial f(x)}{\partial w_2}$ $\frac{\partial f(x)}{\partial w_3}$

small change in parameter Δw_1 ➡ output will change by $\frac{\partial f(x)}{\partial w_1}\Delta w_1$

Slide Credit: K. Kitani



Gradient Descent:

given a fixed-point on a function, move in the direction opposite of the gradient.

update rule:

$$\boxed{w = w - \nabla w}$$

Slide Credit: K. Kitani

Training the world's smallest perceptron

for $n = 1 \ldots N$:

$$w = w + \underline{(y_n - \hat{y})x_n}$$

This is just gradient descent, that means…

this should be the gradient of the loss function

Slide Credit: K. Kitani

Understanding Derivatives

$$\frac{d\mathcal{L}}{dw}$$ …is the rate at which **this** will change…

$$\mathcal{L} = \frac{1}{2}(y - \hat{y})^2$$

(the loss function)

… per unit change of **this**

$$y = wx$$

(the weight parameter)

Slide Credit: K. Kitani

Compute the derivative

$$\frac{d\mathcal{L}}{dw} = \frac{d}{dw}\left\{\frac{1}{2}(y-\hat{y})^2\right\}$$

$$= -(y-\hat{y})\frac{dwx}{dw}$$

$$= -(y-\hat{y})x = \nabla w$$ shorthand

That means the weight update for **gradient descent** is:

$$w = w - \nabla w$$ move in direction of negative gradient

$$= w + (y-\hat{y})x$$

Slide Credit: K. Kitani

Gradient Descent (world's smallest perceptron)

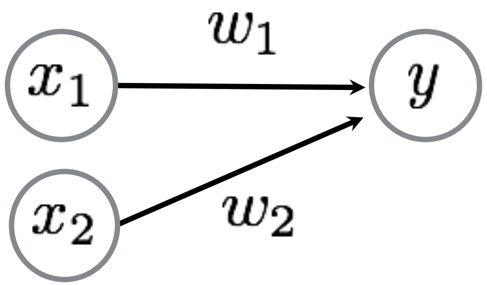
For each sample $\{x_i, y_i\}$

1. Predict
   a. Forward pass $\hat{y} = wx_i$
   b. Compute Loss $\mathcal{L}_i = \frac{1}{2}(y_i - \hat{y})^2$
2. Update
   a. Back Propagation $\frac{d\mathcal{L}_i}{dw} = -(y_i - \hat{y})x_i = \nabla w$
   b. Gradient update $w = w - \nabla w$

Note that in this formulation, we are making a parameter update based on the gradient derived from every single training sample; later on we will look at how to do updates per batch of data samples based on the average gradient.

Slide Credit: K. Kitani

world's (second) smallest perceptron!



function of **two** parameters!

Slide Credit: K. Kitani

Gradient Descent (world's second smallest perceptron)

For each sample $\{x_i, y_i\}$

1. Predict
   a. Forward pass
   b. Compute Loss

we just need to compute partial derivatives for this network

2. Update
   a. Back Propagation
   b. Gradient update

Slide Credit: K. Kitani

Computing Derivatives

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial}{\partial w_1}\left\{\frac{1}{2}(y-\hat{y})^2\right\}$$
$$= -(y-\hat{y})\frac{\partial \hat{y}}{\partial w_1}$$
$$= -(y-\hat{y})\frac{\partial \sum_i w_i x_i}{\partial w_1}$$
$$= -(y-\hat{y})\frac{\partial w_1 x_1}{\partial w_1}$$
$$\boxed{= -(y-\hat{y})x_1 = \nabla w_1}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial}{\partial w_2}\left\{\frac{1}{2}(y-\hat{y})^2\right\}$$
$$= -(y-\hat{y})\frac{\partial \hat{y}}{\partial w_2}$$
$$= -(y-\hat{y})\frac{\partial \sum_i w_i x_i}{\partial w_1}$$
should be w2
$$= -(y-\hat{y})\frac{\partial w_2 x_2}{\partial w_2}$$
$$\boxed{= -(y-\hat{y})x_2 = \nabla w_2}$$

Gradient Update

$$w_1 = w_1 - \eta \nabla w_1$$
$$\boxed{= w_1 + \eta(y-\hat{y})x_1}$$

$$w_2 = w_2 - \eta \nabla w_2$$
$$\boxed{= w_2 + \eta(y-\hat{y})x_2}$$

Slide Credit: K. Kitani

Gradient Descent

For each sample $\{x_i, y_i\}$

1. Predict

   a. Forward pass $\hat{y} = w_1 x_{1i} + w_2 x_{2i}$

   b. Compute Loss $\mathcal{L}_i = \frac{1}{2}(y_i - \hat{y})^2$

   (side computation to track loss.
   not needed for backprop)

2. Update

   a. Back Propagation

   $$\nabla w_{1i} = -(y_i - \hat{y})x_{1i}$$
   $$\nabla w_{2i} = -(y_i - \hat{y})x_{2i}$$

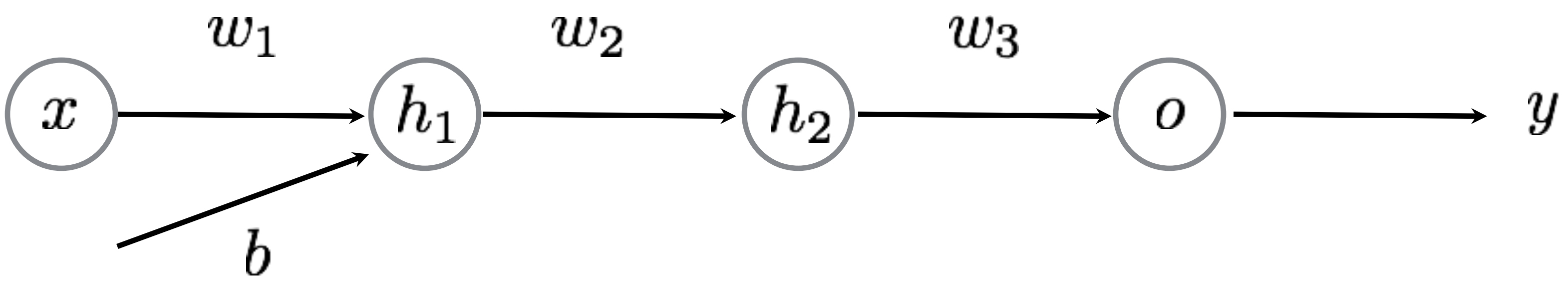
   b. Gradient update

   $$w_{1i} = w_{1i} + \eta(y - \hat{y})x_{1i}$$
   $$w_{2i} = w_{2i} + \eta(y - \hat{y})x_{2i}$$

(adjustable step size)

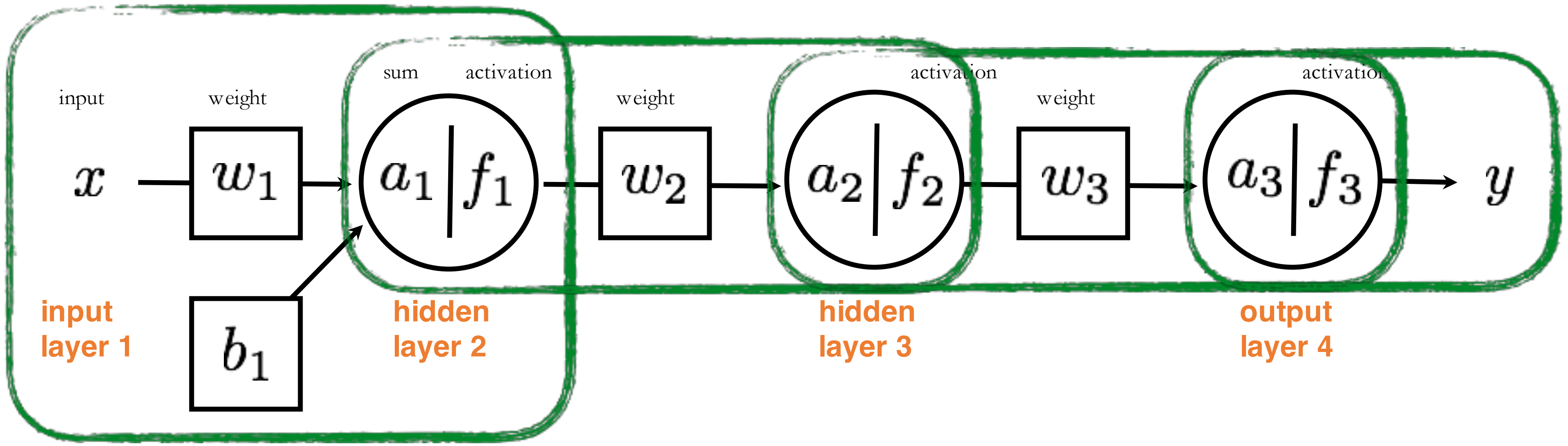
Slide Credit: K. Kitani

MLP: multi-layer perceptron



function of **FOUR** parameters and **FOUR** layers!

Slide Credit: K. Kitani



$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$

$$a_3 = w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1))$$

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

Slide Credit: K. Kitani

Entire network can be written out as one long equation

known

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

activation function
sometimes has parameters

unknown

We need to train the network:

What is known? What is unknown?

Slide Credit: K. Kitani

Learning an MLP

Given a set of samples and an MLP

$$\{x_i, y_i\}$$

$$y = f_{\text{MLP}}(x; \theta)$$

Estimate the parameters of the MLP

$$\theta = \{f, w, b\}$$

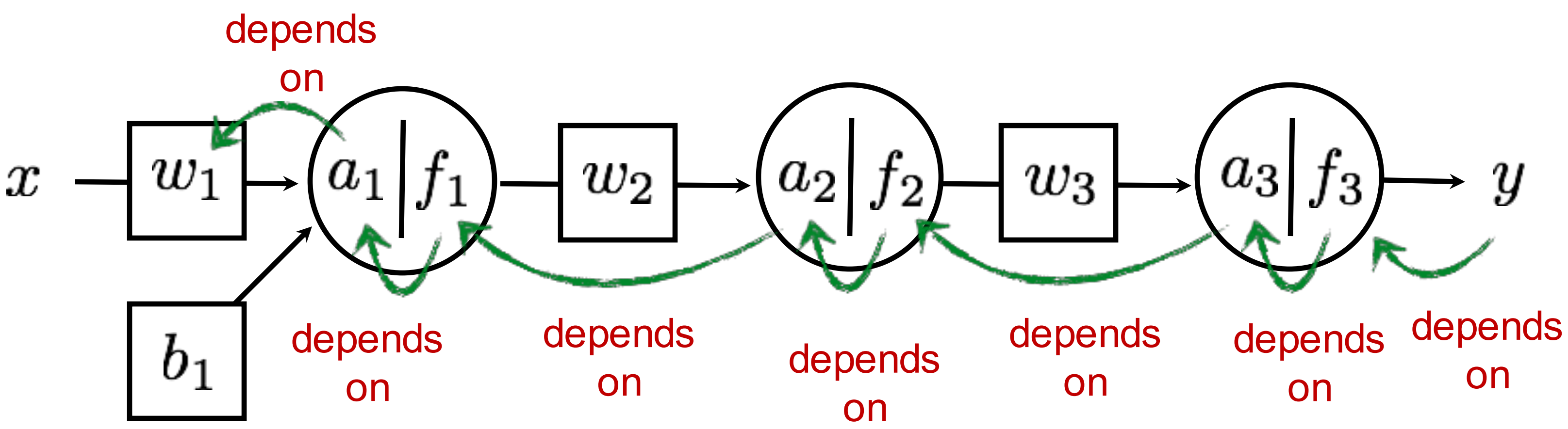
Slide Credit: K. Kitani

Gradient Descent for a multilayer perceptron

For each **random** sample $\{x_i, y_i\}$

1. Predict
   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$
   b. Compute Loss
2. Update
   a. Back Propagation $\frac{\partial \mathcal{L}}{\partial \theta}$ vector of parameter partial derivatives
   b. Gradient update $\theta \leftarrow \theta - \eta \nabla \theta$ vector of parameter update equations

Slide Credit: K. Kitani



The term back-propagation comes from the application of the chain rule, in which the gradients or partial derivatives from down-stream are used upstream.

$$\frac{\partial \mathcal{L}}{\partial w_3} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}}\frac{\partial a_3}{\partial w_3}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \boxed{\boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}}\frac{\partial a_2}{\partial w_2}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \boxed{\boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}}\frac{\partial a_1}{\partial w_1}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}}\frac{\partial a_1}{\partial b}$$

Slide Credit: K. Kitani

Gradient Descent for a multilayer perceptron

For each data sample $\{x_i, y_i\}$

1. Predict

   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$ $\theta = [w_1, w_2, w_3, b]$

   b. Compute Loss $\mathcal{L}_i$

2. Update

   a. Back Propagation

$$\left.\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_3} &= \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3} \\
\frac{\partial \mathcal{L}}{\partial w_2} &= \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial w_2} \\
\frac{\partial \mathcal{L}}{\partial w_1} &= \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1} \\
\frac{\partial \mathcal{L}}{\partial b} &= \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial b}
\end{aligned}\right\} \frac{\partial \mathcal{L}}{\partial \theta}$$

vector of parameter partial derivatives

   b. Gradient update

$$\left.\begin{aligned}
w_3 &= w_3 - \eta \nabla w_3 \\
w_2 &= w_2 - \eta \nabla w_2 \\
w_1 &= w_1 - \eta \nabla w_1 \\
b &= b - \eta \nabla b
\end{aligned}\right\} \theta \leftarrow \theta + \eta \frac{\partial \mathcal{L}}{\partial \theta}$$

vector of parameter update equations

Slide Credit: K. Kitani

Regression

Polynomial regression, over/underfitting, dataset-splitting

Polynomial regression

- f(x) is a polynomial function
  - $f(x) = xw1 + x^2w2 + x^3\ w3 \ldots + x^M\ wM + b$
  - M, the order of the polynomial, is unknown
  - As shown by the curve
- We have a training dataset generated from
  - $y = f(x) + \varepsilon$ random noise term
  - As shown by the circles
- Train a polynomial regression model over the training data
  - Find M: 0, 1, 2, …?
  - Tune $w_1, w_2, \ldots w_M, b$: $f(x) = \boldsymbol{w}^T\boldsymbol{x},$ where $\boldsymbol{x} = (x, x^2, \ldots x^M, 1)^T$, $\boldsymbol{w}$ =?

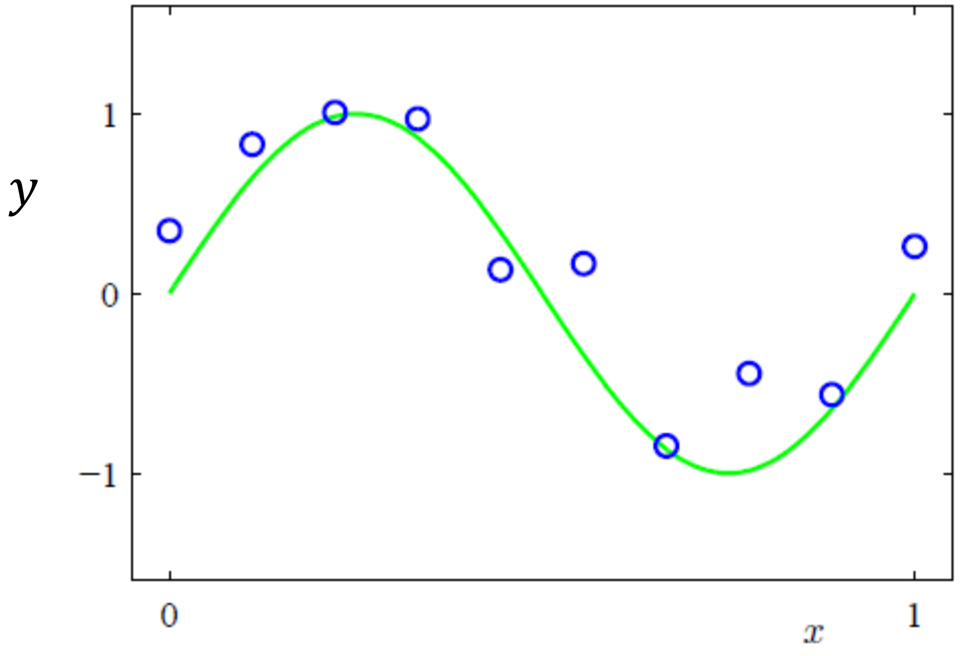


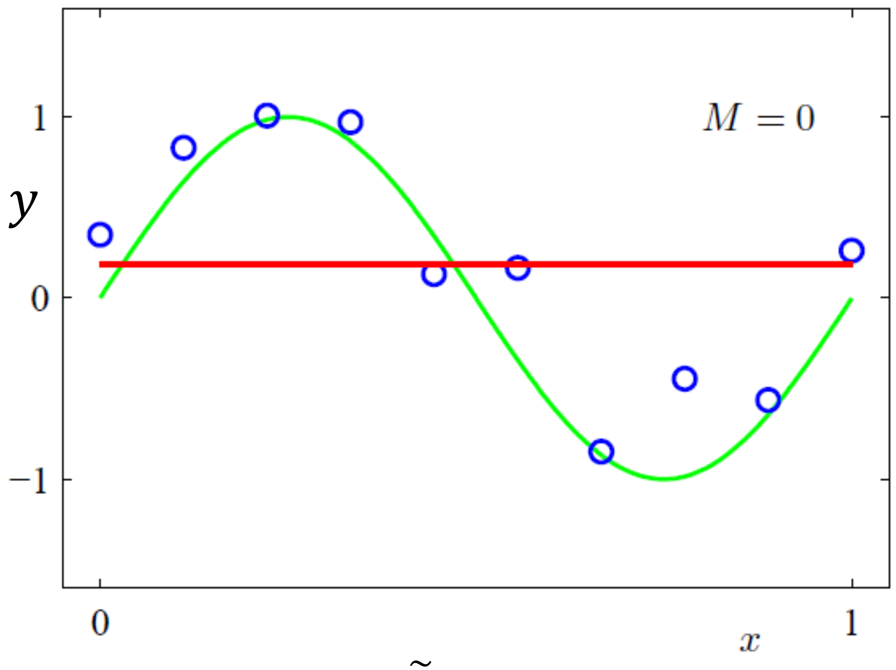
Image source: Pattern Recognition and Machine Learning, Christopher Bishop.

Slide credit: Wang Wei

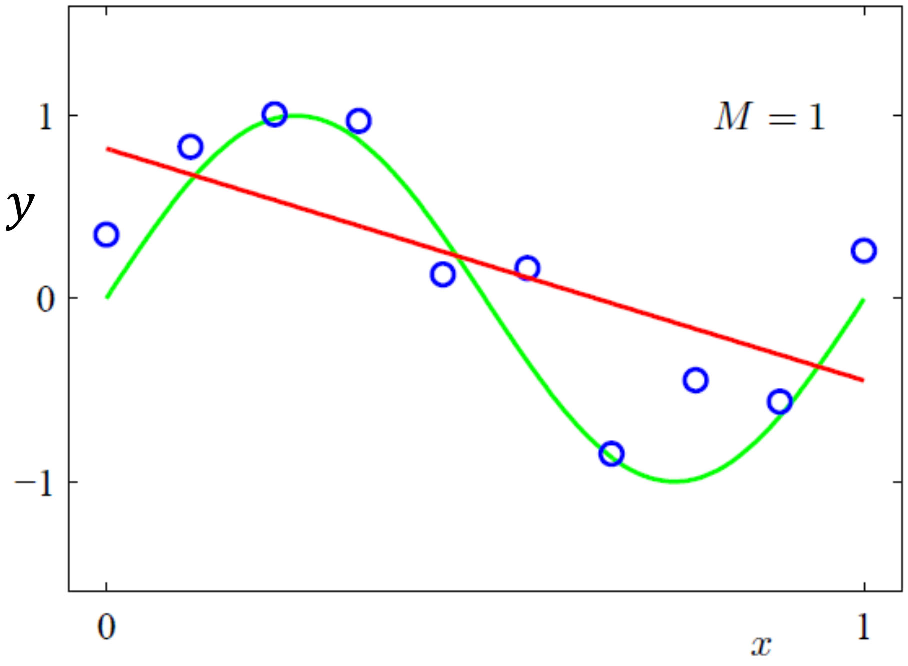
Underfitting

A slightly tautological definition of bias (error):
An algorithm's tendency to consistently learn wrong things by not taking into account sufficient information found in the data

- Low model capacity / complexity → model too simple to fit training data
- High bias



$\tilde{y} = b$



$\tilde{y} = xw + b$

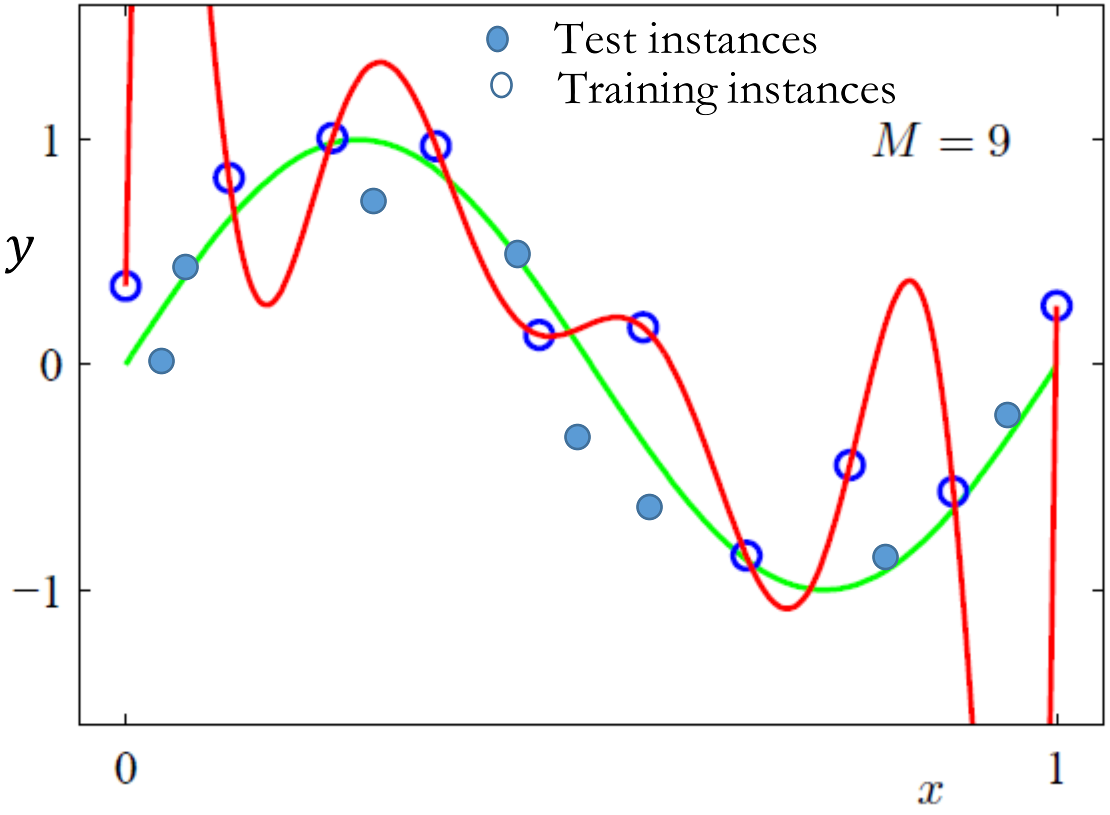
Image source: Pattern Recognition and Machine Learning, Christopher Bishop.

Slide credit: Wang Wei

Overfitting

- High capacity/complexity → fits well to seen data, i.e. training data
- High variance across datasets
- Cannot generalize well onto new data, so performs poorly on unseen data, i.e. test

Variance (errors):
Amount that the estimates (in our case the parameters **w**) will change if we had different of training data (but still drawn from the same source).



$$\tilde{y} = xw1 + x^2w2 + x^3\, w3 + \cdots + x^9\, w9 + b$$

Image source: Pattern Recognition and Machine Learning, Christopher Bishop.

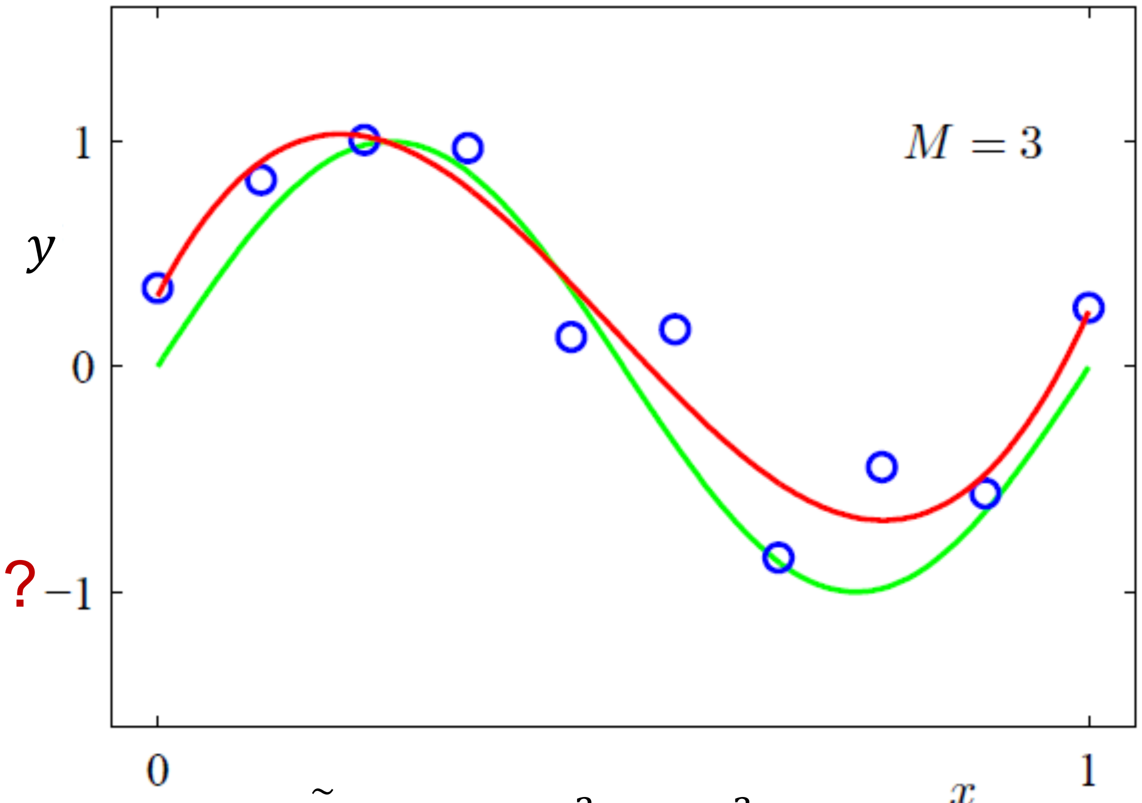
Slide credit: Wang Wei

A Good Model

- Uses the “right” capacity / complexity to model the data and can generalize to unseen data
- Strikes a balance between under- and over-fitting, as well as bias and variance

Q: Can’t we have low bias AND low variance?

No. opposing phenomenon derived from same factor: model capacity. Detailed explanations [A][B]

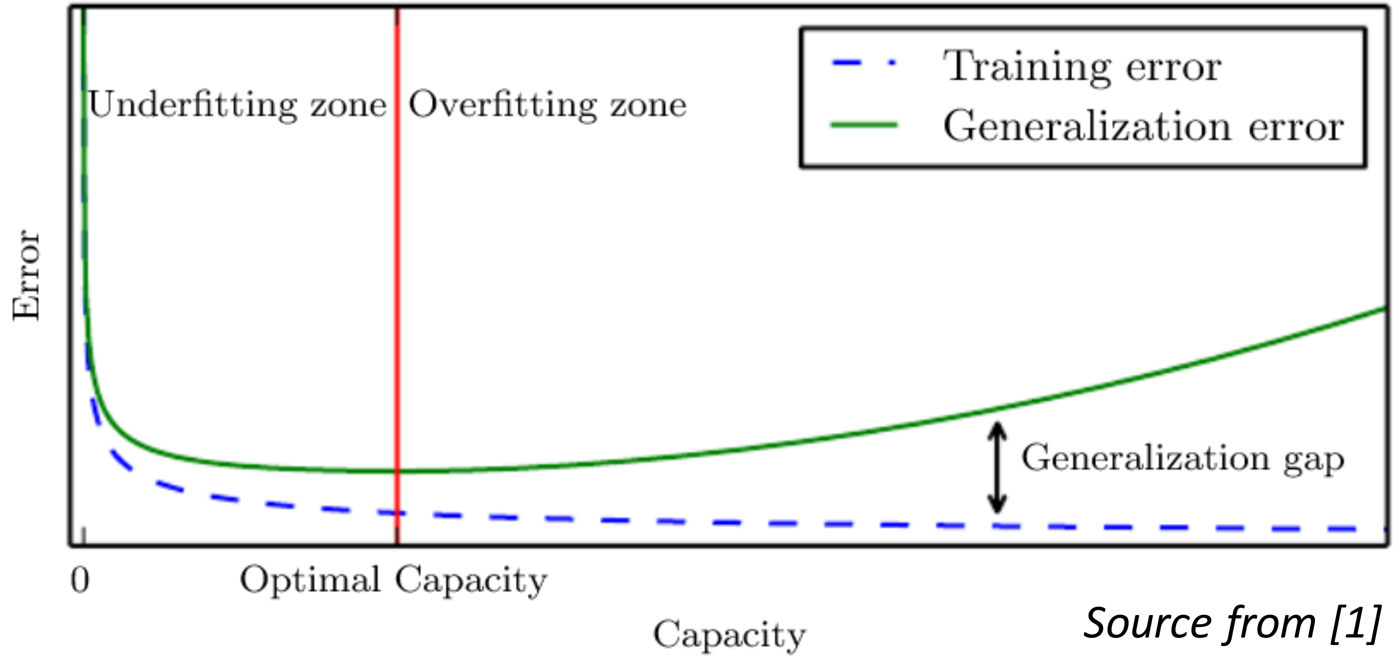


$$\tilde{y} = xw1 + x^2w2 + x^3\,w3 + b$$

Image source: Pattern Recognition and Machine Learning, Christopher Bishop.

Slide credit: Wang Wei

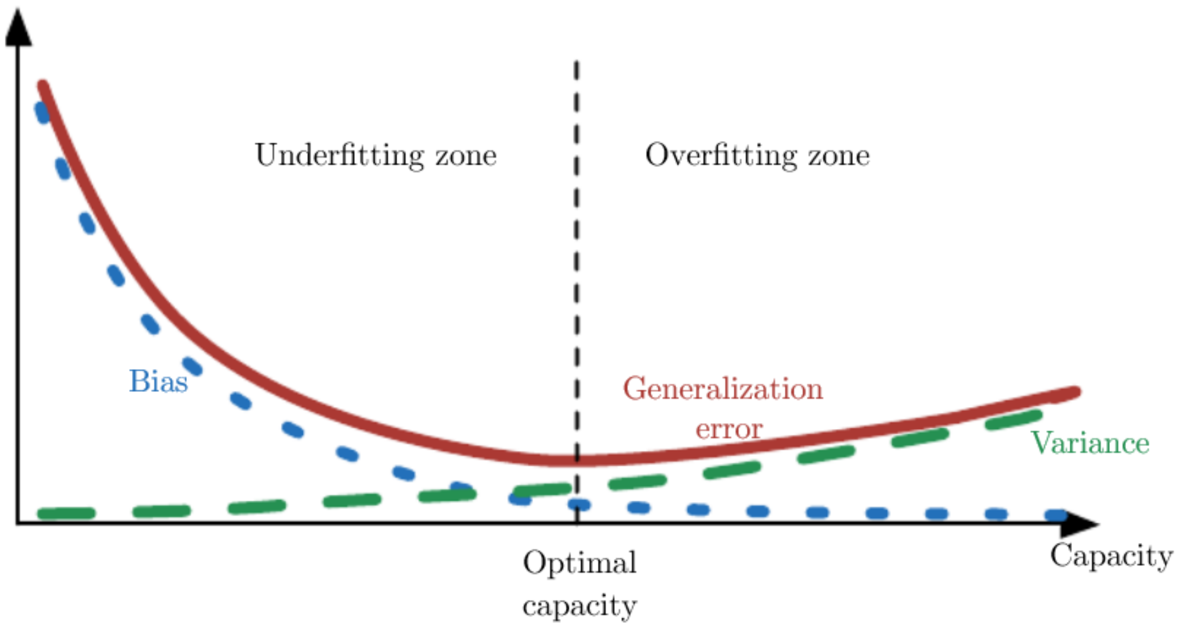
Underfitting and overfitting



Source from [1]

Slide credit: Wang Wei

All in one picture



Source from [1]

First, train a bigger model to reduce the bias (to avoid underfitting)
Second, regularize the model to reduce the variance (to avoid overfitting)

Slide credit: Wang Wei

Hyper-parameter/model tuning

$\tilde{y} = w_1 x + b$ $\tilde{y} = w_1 x + w_2 x^2 + b$ $\tilde{y} = w_1 x + w_2 x^2 + \cdots + w_9 x^9 + b$

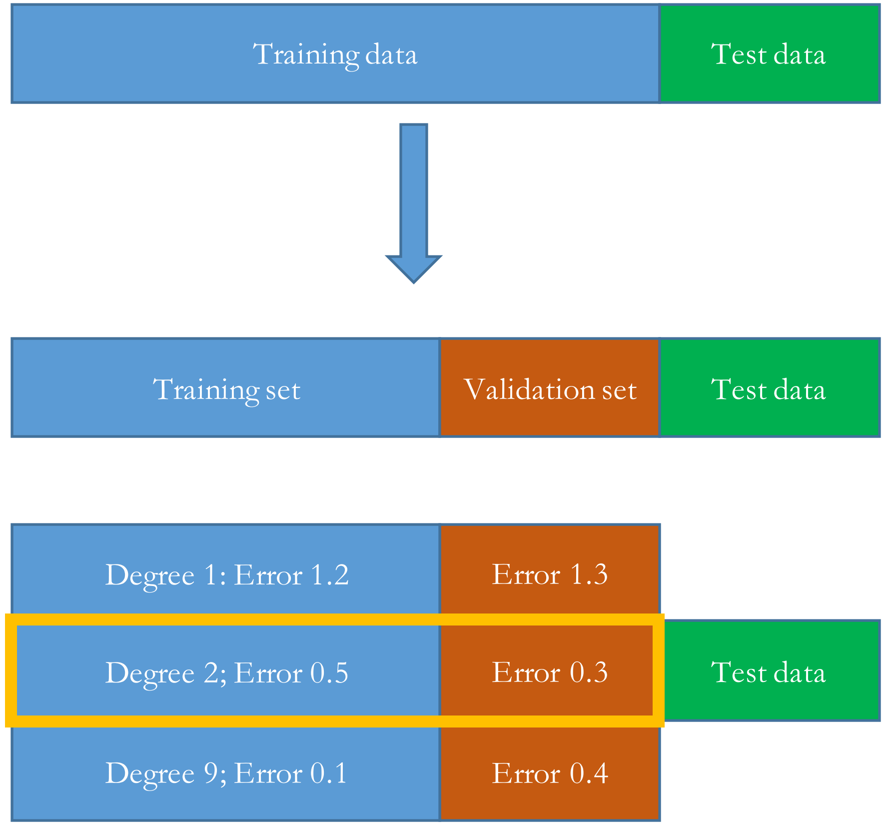
- Which degree/model to use to avoid underfitting or overfitting?
  - Degree is a hyper-parameter or configuration knob
  - Tuning the degree is called hyper-parameter tuning or model selection
  - For complex models, there could be many such hyper-parameters
    - Learning rate of the gradient descent algorithm, α
    - Number of layers for a neural network
- Training VS Testing
  - Never train or tune the model over test data
  - Test data is purely for reporting the final (unbiased) performance

Students can't see the exam questions and answers before the exam.

Hyper-parameter & Model Tuning

- Split the training data
  - Training set for model training
  - Validation set for model selection/tuning
- How to split?
  - K-fold cross-validation if few training data
  - fixed ratio partitioning, e.g., 80:20, 90:10 or 95:5

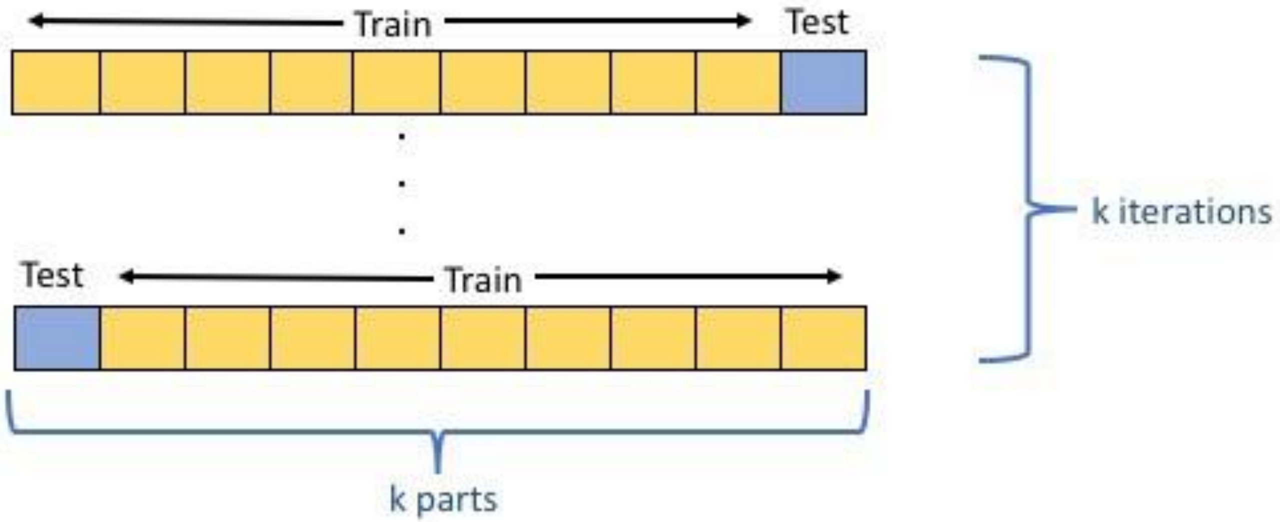
- $\tilde{y} = w_1 x + b$
- $\tilde{y} = w_1 x + w_2 x^2 + b$
- $\tilde{y} = w_1 x + w_2 x^2 + \cdots + w_9 x^9 + b$



Slide credit: Wang Wei

K Folds Cross Validation Method

1. Divide the sample data into k parts.
2. Use k-1 of the parts for training, and 1 for testing.
3. Repeat the procedure k times, rotating the test set.
4. Determine an expected performance metric (mean square error, misclassification error rate, confidence interval, or other appropriate metric) based on the results across the iterations



The hyper-parameters that can achieve the best averaged accuracy

Splitting the Data

- Case 1: Real Applications
  - Split all your data into training and validation

Hope is that this validation data is representative of the test data encountered when application is deployed.

- Case 2: Challenges, e.g. Kaggle competitions
  - Split the training data into training and validation
  - Test performance determined by organizers on private test data
    - Test your submitted model OR
    - Evaluate submitted test results for which labels are kept private
- Case 3: Research
  - Split all data into training and test (e.g., 5%, 10% or 20%)
  - Split the training data into training set and validation set (see prev. slide)

Slide credit: Wang Wei

Classification

Regression vs. classification, cross-entropy loss

Multi-class, muli-label classification

Multi-class, single-label classification

Regression VS Classification

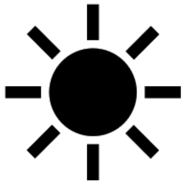
Q: What is the difference?

Quantity vs. Label:
regression maps to a continuous domain, while classification maps to a finite set.

- Regression: What's the temperature of tomorrow?
- Classification (Binary): Is it sunny tomorrow?
- Classification (Multi-class): Is it sunny, cloudy, or rainy?

Q: How many mm of rain makes it a "rainy" day? How many hours of sunshine makes it a "sunny" day?

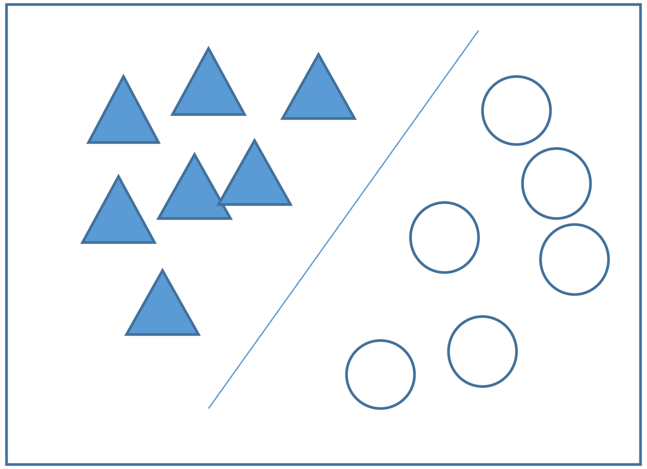
Often, classification labels must be derived from continuous values (measured or regressed).



today	1
tomorrow	0

	Sunny	Rainy	Cloudy
Monday	1	0	0
Tuesday	0	1	0
Wednesday	0	0	1

From regression to classification



- Thresholding (Perceptron)
- $\tilde{y} = \begin{cases} 1, & if\, \boldsymbol{w}^T \boldsymbol{x} > c \\ 0, & else \end{cases}$
- How to set the threshold c?

Learn it as a part of learning weights.

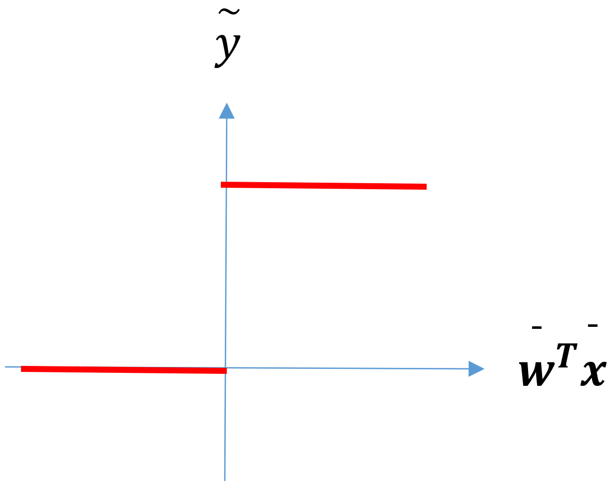
$$\boldsymbol{w}^T \boldsymbol{x} > c$$

$$x_1 w_1 + x_2 w_2 \ldots + x_m w_m + b > c$$

$$x_1 w_1 + x_2 w_2 \ldots + x_m w_m \underbrace{+ (b - c)}_{\text{Merge c as part of the offset / b parameter}} > 0$$

$$\bar{\boldsymbol{w}}^T \bar{\boldsymbol{x}} > 0$$

$$\tilde{y} = \begin{cases} 1, & if\, \bar{\boldsymbol{w}}^T \bar{\boldsymbol{x}} > 0 \\ 0, & else \end{cases}$$



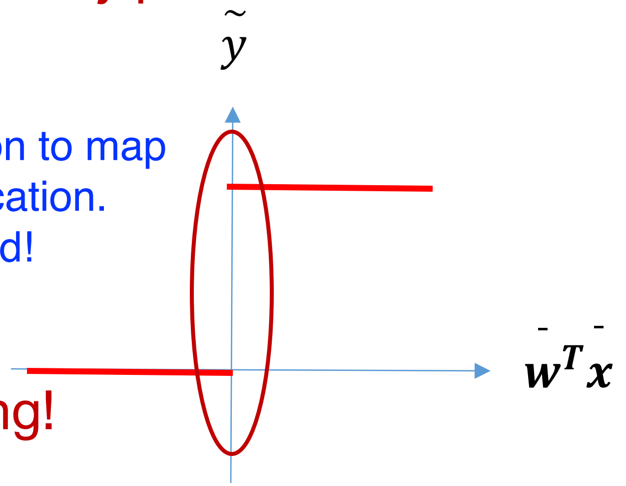
Modified from Wang Wei

Logistic regression

Only piecewise differentiable

$$\tilde{y} = \begin{cases} 1, & if\, \boldsymbol{w^T x} > 0 \\ 0, & else \end{cases}$$

We want to find a function to map
regression to classification.
But this one is bad!



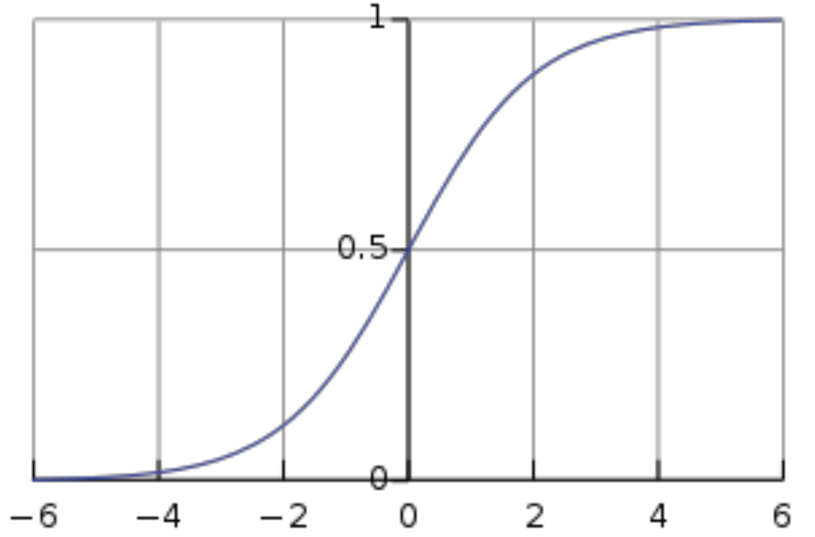
What is a better function? let us try Logistic

Gradient of a constant is always 0, so cannot learn anything!

Logistic function: $p = \sigma(\boldsymbol{w^T x}) = \frac{1}{1+e^{-\boldsymbol{w^T x}}}$

- Range is within [0, 1]
- Possible interpretation: probability of the label being 1
- Logistic function sometimes also referred to as a sigmoid function

Why Logistic is good?



Modified from Wang Wei

Gradient Vanishing

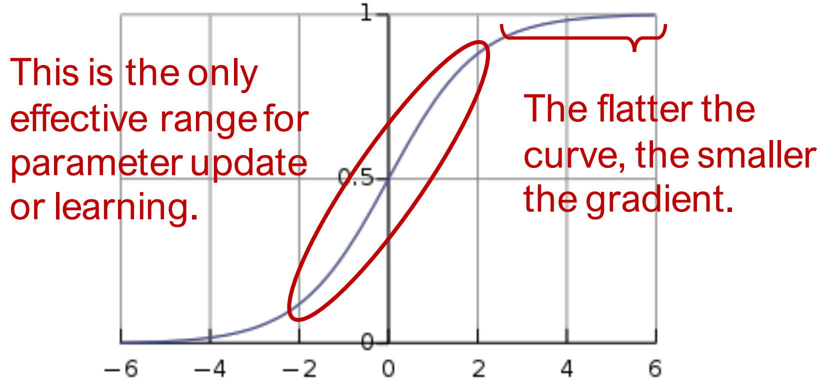
How should we learn the weights of the logistic function? Start with a simple L2 loss.

- $L(\boldsymbol{x}, y) = \frac{1}{2}||\sigma(\boldsymbol{w}^T\boldsymbol{x}) - y||^2, \frac{\partial L}{\partial \boldsymbol{w}}$?
  - denote $z = \boldsymbol{w}^T\boldsymbol{x}, L = \frac{1}{2}(\sigma(z) - y)^2$

Is Logistic function good enough?

- $\frac{\partial L}{\partial \boldsymbol{w}} = (\sigma(z) - y) * \sigma(z)(1 - \sigma(z))\ \mathbf{x}$
- If $\sigma \approx 0\ or\ 1, \frac{\partial \sigma}{\partial z} \approx 0 \rightarrow \frac{\partial L}{\partial \boldsymbol{w}} \approx \mathbf{0}$
- Impact: training gets stuck since $\boldsymbol{w} = \boldsymbol{w} - \alpha * \frac{\partial L}{\partial \boldsymbol{w}}$

gradient vanishing



Training stops because of zero or very tiny gradients

Modified from Wang Wei

Cross-entropy loss

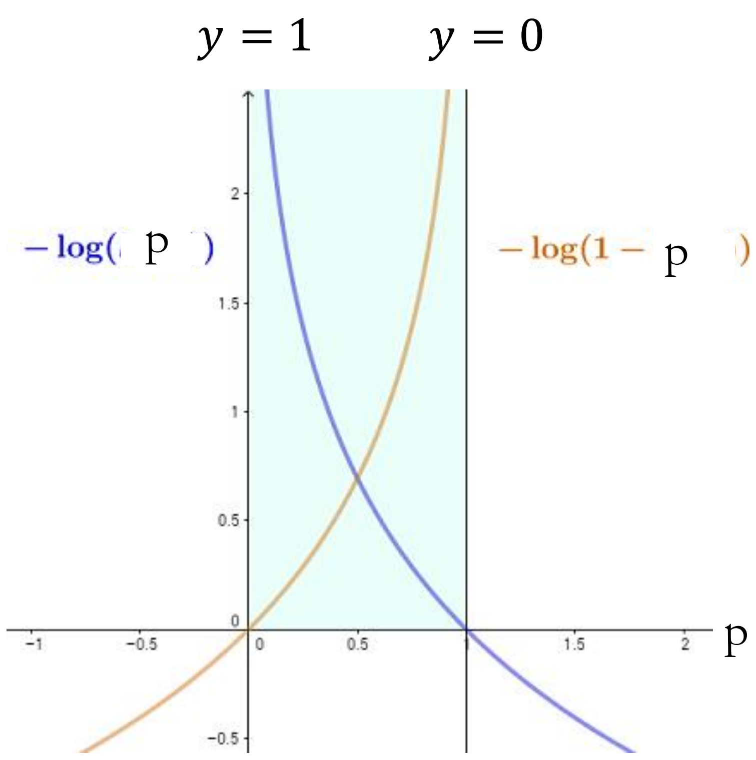
- Denote $p = \sigma(z), z = \boldsymbol{w}^T \boldsymbol{x}$
- Instead of using an L2 loss, we propose the following:
- $L_{ce}(x, y) = \underbrace{-y \log p}_{} - \underbrace{(1-y)\log(1-p)}_{}$

This term goes to 0 if ground truth label is 0

This term goes to 0 if ground truth label is 1

$$= \begin{cases} -y\log p\,, & if\ y = 1 \\ -(1-y)\log(1-p)\,, & if\ y = 0 \end{cases}$$

$$= \begin{cases} -\log p\,, & if\ y = 1 \\ -\log(1-p)\,, & if\ y = 0 \end{cases}$$



What is the intuition behind this loss? Does it actually help us learn the right weights?

Modified from Wang Wei

Cross-entropy loss explanation

Consider the probability of a classifier being correct.

$$P(correct|\boldsymbol{x}) = \begin{cases} P(\tilde{y} = 1|\boldsymbol{x}), & if\ y = 1 \\ P(\tilde{y} = 0|\boldsymbol{x}), & if\ y = 0 \end{cases}$$

(depends on the ground truth label y)

collapse cases into a single function

$$= P(\tilde{y} = 1|\boldsymbol{x})^{y} P(\tilde{y} = 0|\boldsymbol{x})^{1-y}$$

We want to maximize this, i.e. to maximize the probability of our classifier being correct!

Log-likelihood of our classifier being correct:

$$\log P(correct|\boldsymbol{x}) = y \log P(\tilde{y} = 1|\boldsymbol{x}) + (1-y) \log P(\tilde{y} = 0|\boldsymbol{x})$$

Note that so far, this is general and that we have not made any assumptions about the classifier itself, i.e. the specific form of $P(\tilde{y}|\boldsymbol{x})$

Objective equivalent to minimizing the negative log-likelihood

$$\min -\log P(correct|\boldsymbol{x}) = \min -y \log P(\tilde{y} = 1|\boldsymbol{x}) - (1-y) \log P(\tilde{y} = 0|\boldsymbol{x})$$

Modified from Wang Wei

Cross-entropy loss explanation

$$P(\tilde{y} = 1 | \boldsymbol{x}) = p = \sigma(z)$$

$$P(\tilde{y} = 0 | \boldsymbol{x}) = 1 - p$$

Because range of logistic is between 0-1, we adopt p as the probability of the of x having a label.

Problem is binary, equate probability of having label 0 as the complement.

Minimizing negative log likelihood:

$$\min -\log P(correct | \boldsymbol{x}) = \min -y \log P(\tilde{y} = 1 | \boldsymbol{x}) - (1 - y) \log P(\tilde{y} = 0 | \boldsymbol{x})$$

$$= \underbrace{\min -y \log p - (1 - y) \log (1 - p)}_{L_{ce}(\boldsymbol{x}, y)}$$

Substituting the logistic function for $P(\tilde{y}|\boldsymbol{x})$

That's how we get the cross-entropy loss.

The cross-entropy loss minimizes the classifier's log-likelihood.

Cross-entropy loss gradient

$$\frac{\partial L_{ce}}{\partial \boldsymbol{w}} = \frac{\partial L_{ce}}{\partial z}\frac{\partial z}{\partial \boldsymbol{w}} = \frac{\partial L_{ce}}{\partial p}\frac{\partial p}{\partial z}\frac{\partial z}{\partial \boldsymbol{w}} = \left(-\frac{y}{p} + \frac{1-y}{1-p}\right) * p * (1-p)\boldsymbol{x}$$

$$= (p-y)\boldsymbol{x}$$

Modified from Wang Wei

Multi-class classification

- Single-label

	Sunny	Rainy	Cloudy
Monday	1	0	0
Tuesday	0	1	0
Wednesday	0	0	1

One hot vector

- Multi-label: can belong to more than one class

Monday	1	0	1
Tuesday	0	1	1

Sunny & Cloudy
Rainy & Cloudy

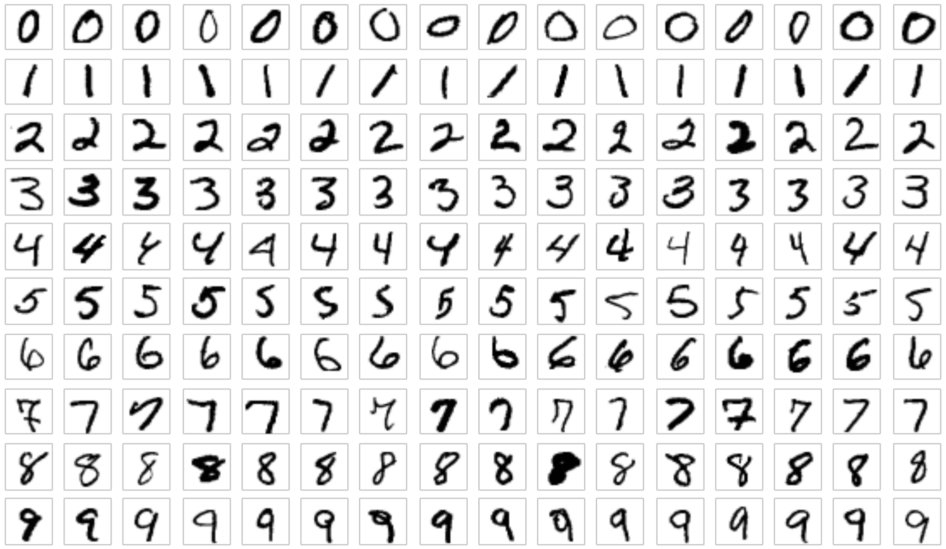
Modified from Wang Wei

Applications

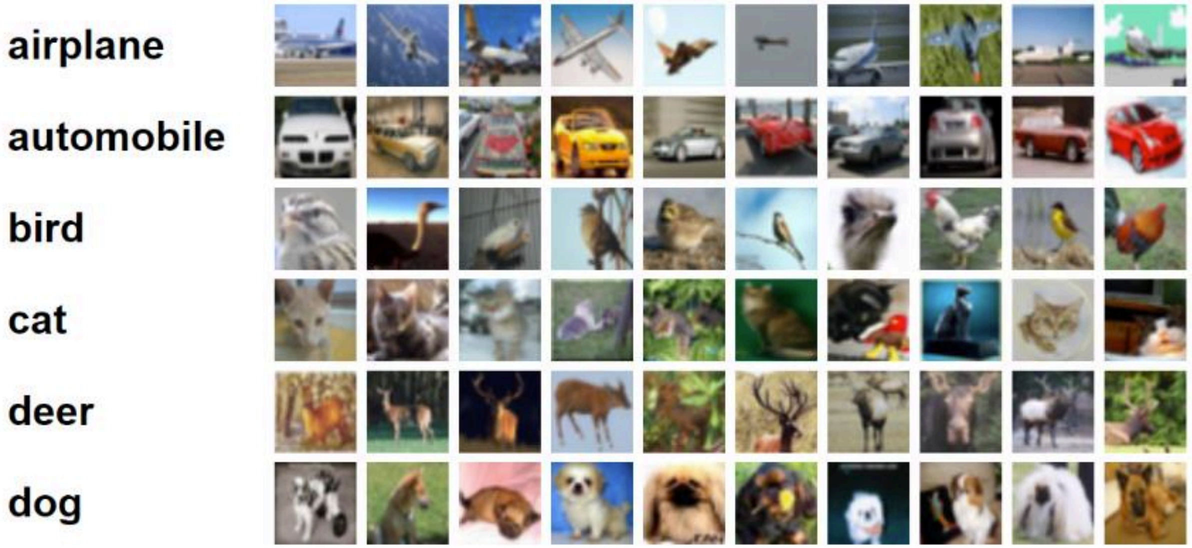
- Multi-class image classification
  - Classify each image into one of the class
  - [MNIST dataset]
    - {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
    - What is **x?** i.e., how to represent an image
  - [Cifar10 dataset]
    - {Dog, Cat, Horse, Ship, Truck, Frog, Deer, Bird, Automobile, Airplane}
- Multi-class document classification
  - 20 Newsgroups, {hardware, autos, space, etc}

Slide credit: Wang Wei

Applications



MNIST
handwritten digits recognition

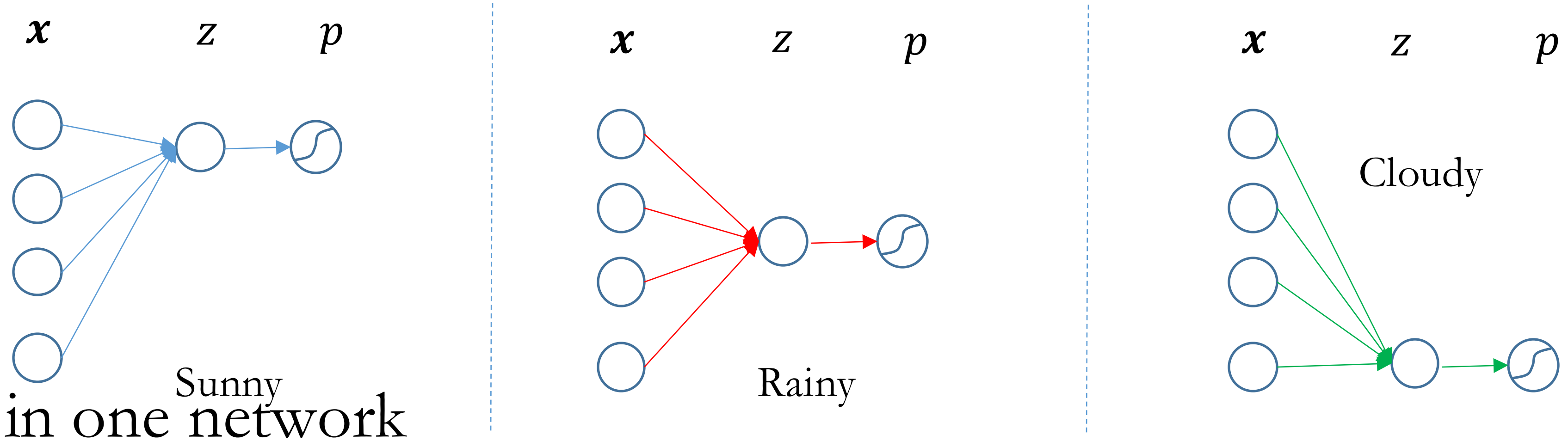


CIFAR
Object Recognition in Images

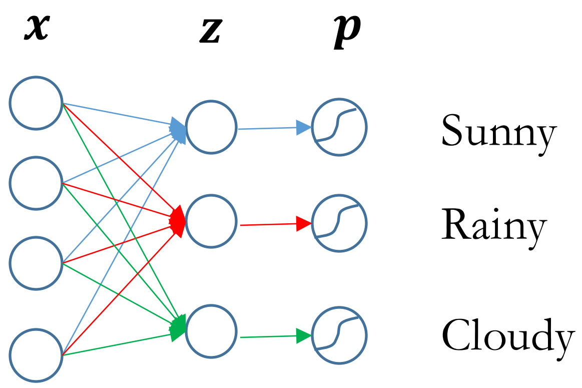
Modified from Wang Wei

Multi-class multi-label classification

- Binary classification for each label

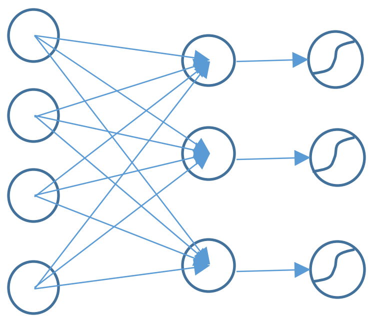


- All in one network



Slide credit: Wang Wei

Multi-class multi-label classification



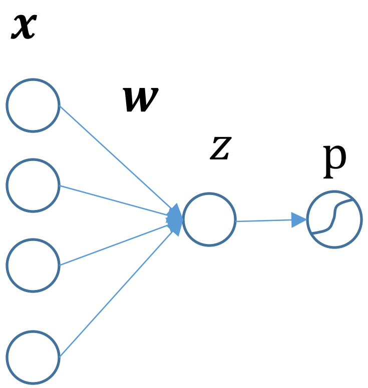
- For binary classification
  - $z = \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} + b, \quad p = \sigma(z), \quad L_{ce} = -y \log p - (1-y)\log(1-p)$
  - $\boldsymbol{x} \in R^n, \boldsymbol{w} \in R^n, b \in R, \; p \in R$
- For multi-class, the i-th class
  - We assume that the classes are independently conditioned on the input
  - $z_i = \boldsymbol{W}_i\boldsymbol{x} + b_i, \quad p_i = \sigma(z_i), \quad L_{ce} = -y_i \log p_i - (1-y_i)\log(1-p_i)$
  - $x \in R^n, \; \boldsymbol{W}_i \in R^n, \; b_i \in R^1, \; p_i \in R^1$
- For multi-class, multi-label (vectorized form)
  - $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}, \quad \boldsymbol{p} = \sigma(\boldsymbol{z}), \quad L_{ce} = \boldsymbol{1}^{\boldsymbol{T}}(-\boldsymbol{y}\log \boldsymbol{p} - (\boldsymbol{1}-\boldsymbol{y})\log(\boldsymbol{1}-\boldsymbol{p}))$
  - $x \in R^n, \; W \in R^{k\times n}, \; \boldsymbol{b} \in R^k, \; \boldsymbol{p} \in R^k, \; \boldsymbol{y} \in \{0,1\}^k$

Probability of belonging to class i is independent of belonging to class j, once conditioned on the input evidence

Slide credit: Wang Wei

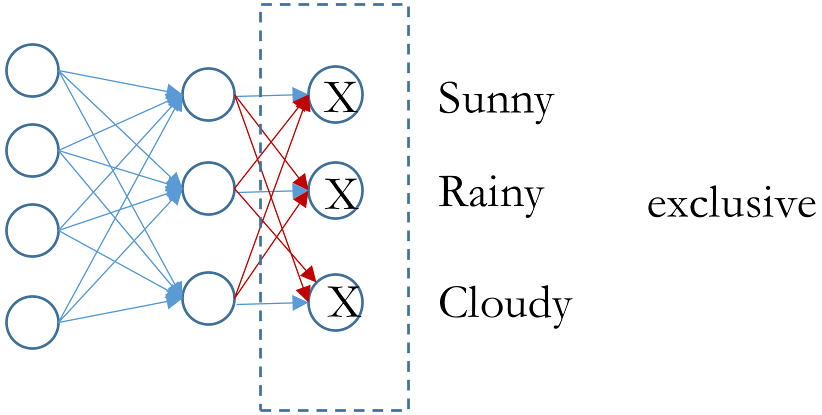
Multi-class single-label classification

- Choose one label from multiple classes
  - Exclusive
    - If $p(sunny)$ is large, then $p(rainy) + p(cloudy)$ small



- How to enforce this constraint?
  - $p(sunny) + p(rainy) + p(cloudy) = 1$
  - $\sum_{i=1} p_i = 1$

Our previous model had connections only from each z_i to p_i; now we add the red arrows to connect all z_i to p_i



Slide credit: Wang Wei

Multi-class single-label classification

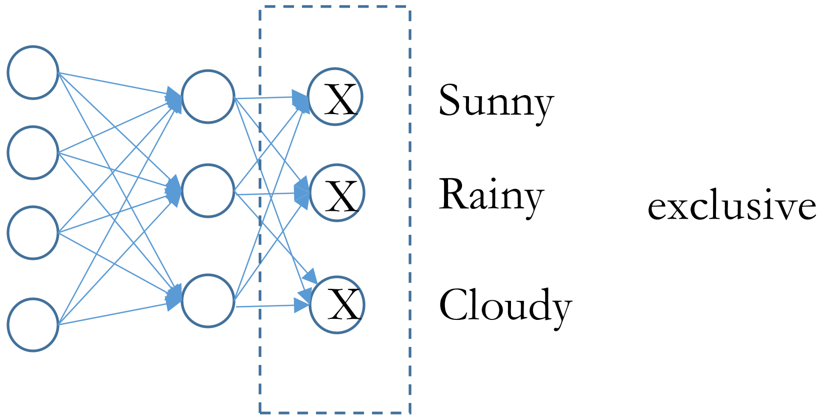
- Softmax regression or multinomial logistic regression
  - $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x}, \boldsymbol{W} \in R^{k\times n}$,
  - $p_i = \frac{e^{z_i}}{\sum_j e^{z_j}} = \mathrm{softmax}(z_i)$

We choose this to make sure the sum is 1

  - Then we have $\sum_i p_i = 1$
  - z_i is called a logit
  - $t = \underset{i}{\mathrm{argmax}}\, p_i\,;\ \tilde{y_i} = 1\ if\ i = t;\ else\ 0;$

How can we make prediction? The winner takes it all :-)

P1 = 0.166, P2 = 0.166, P3 = 0.166, P4 = 0.166, P5 = 0.17, P6= 0.166
The output prediction is [0, 0, 0, 0, 1, 0]



Slide credit: Wang Wei

Multi-class single-label classification

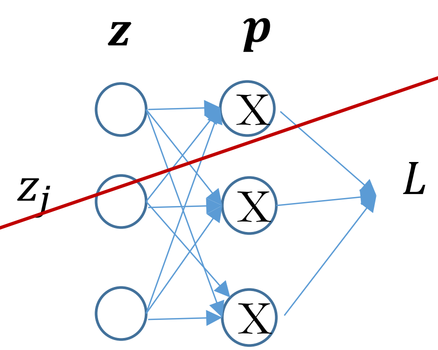
- Loss function: $L_{ce}(\boldsymbol{x}, \boldsymbol{y}) = \sum_i -y_i \log p_i \; = -\boldsymbol{y}^T \log \boldsymbol{p}$
  - Each row of X is the feature vector $\boldsymbol{x}$
  - Each row of Y is the target one-hot vector $\boldsymbol{y}$

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}^{(1)^T} \\ \boldsymbol{x}^{(2)^T} \\ \dots \\ \boldsymbol{x}^{(N)^T} \end{pmatrix} \qquad \boldsymbol{Y} = \begin{pmatrix} \boldsymbol{y}^{(1)^T} \\ \boldsymbol{y}^{(2)^T} \\ \dots \\ \boldsymbol{y}^{(N)^T} \end{pmatrix}$$

- $\frac{\partial L_{ce}}{\partial \boldsymbol{W}} =$? $\frac{\partial L_{ce}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{w}}$

Slide credit: Wang Wei

$$\frac{\partial L}{\partial \boldsymbol{z}} = \frac{\partial L}{\partial \boldsymbol{p}}\frac{\partial \boldsymbol{p}}{\partial \boldsymbol{z}} \qquad \boxed{= \boldsymbol{p} - \boldsymbol{y}}$$



$$\boxed{\frac{\partial L}{\partial p_i} = -\frac{y_i}{p_i}, \qquad \frac{\partial L}{\partial \boldsymbol{p}} = -\frac{\boldsymbol{y}}{\boldsymbol{p}}}$$

$$\frac{\partial L}{\partial z_j} = \sum_i \frac{\partial L}{\partial p_i}\frac{\partial p_i}{\partial z_j}$$

$$= \frac{\partial L}{\partial p_j}\frac{\partial p_j}{\partial z_j} + \sum_{i\neq j}\frac{\partial L}{\partial p_i}\frac{\partial p_i}{\partial z_j}$$

$$= -\frac{y_j}{p_j}(p_j - p_j^2) + \sum_{i\neq j} -\frac{y_i}{p_i}(-p_i p_j)$$

$$= -y_j(1 - p_j) + \sum_{i\neq j} y_i p_j$$

$$= -y_j + y_j p_j + p_j \sum_{i\neq j} y_i \qquad \left(\sum y_i = 1\right)$$

$$= -y_j + y_j p_j + p_j(1 - y_j) \boxed{= p_j - y_j}$$

$$p_i = \frac{e^{z_i}}{\sum_k e^{z_k}}$$

$$h(\mathrm{x}) = \frac{f(\mathrm{x})}{g(x)} \rightarrow h'(x) = \frac{f'(x)g(x) - f(x)g'(x)}{g^2(x)}$$

$$\frac{\partial \sum_k e^{z_k}}{\partial z_j} = \frac{\partial e^{z_j}}{\partial z_j} = e^{z_j}$$

$$\frac{\partial p_i}{\partial z_j} = \begin{cases} i = j, \dfrac{e^{z_j}\sum_k e^{z_k} - e^{z_j}e^{z_j}}{\left(\sum_k e^{z_k}\right)^2} = p_j - p_j^2 \\ i \neq j, \dfrac{0\sum_k e^{z_k} - e^{z_i}e^{z_j}}{\left(\sum_k e^{z_k}\right)^2} = -p_i p_j \end{cases}$$

Slide credit: Wang Wei

3-minute Quiz

- Now you need to train a Neural Network model by Gradient Descent.
- This model has 300 Billion parameters.
- How much hardware memory do you need for your computer?
- Please answer in B, e.g. 300 MB, 600 GB, 900 TB
- You just need to process 1 sample each iteration. 1 sample costs 1 GB.

3-minute Quiz

- If you use single precision: **3601 GB**
- If you use double precision: **7201 GB**
- Explanation for single precision (the same idea as double precision):
  - You need to save parameters, gradients, activations, and input sample
  - 1 parameter costs 4 bytes or 4B, 300 Billion parameters cost 1200 GB
  - Gradients cost the same memory as parameters because they have same shape
  - Activations can't cost more memory than parameters (<= 1200 GB)
  - So 1200 GB + 1200 GB + 1200 GB + 1 GB should be enough!

3-minute Quiz (part 2)

- Now you need to train a Neural Network model by Gradient Descent.
- This model has 300 Billion parameters.
- How much hardware memory do you need for your computer?
- Please answer in B, e.g. 300 MB, 600 GB, 900 TB
- ~~You just need to process 1 sample each iteration. 1 sample costs 1 GB.~~
    - You need to process 1000 samples each iteration. 1 sample costs 1 GB.

3-minute Quiz (part 2)

- If you use single precision: **1203400 GB = 1.15 PB**
- If you use double precision: **2405800 GB = 2.29 PB**
- Explanation for double precision (the same idea as single precision):
  - You need to save parameters, gradients, activations, and input sample
  - 1 parameter costs 8 bytes or 8B, 300 Billion parameters cost 2400 GB
  - Gradients cost the same memory as parameters because they have same shape
    - Why don't save 1000 copies of different gradients?

      a tiny amount of memory from previous layer gradients
    - Because you only need the average, so you can just use 1 copy's space
  - Activations can't cost more memory than parameters (<= 2400 GB)
    - But you need to save 1000 copies of different activations
  - So 2400 GB + 2400 GB + 2400*1000 GB + 1*1000 GB should be enough!

Summary

- Single-unit perceptrons are weighted sums followed by an activation
- Neural networks are built from multiple perceptron units stacked on top of each other (multi-layer perceptrons or MLPs)
- Over/under-fitting may arise due to too little / too much model capacity
- Split your data into training / (validation) / testing; do not learn model on the test!
- Regression vs. classification as basic machine learning tasks
- Using logistic regression to approximate the decision for classification
- Logistic functions should be learned with cross-entropy and not L2 loss due to gradient vanishing problem
- cross-entropy loss tries to minimize the difference between the output distribution and the (ground truth) label distribution