

NANYANG
TECHNOLOGICAL
UNIVERSITY
SINGAPORE

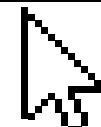
Natural Language Processing

SC4002 / CE4045 / CZ4045
by Wang Wenya

Email: wangwy@ntu.edu.sg

**Click here
for Lecture 1**

Contents adapted from Dr. Joty Shafiq's notes





Outline for today

- 01 Course Logistics
- 02 What is NLP (a quick recap)
- 03 Plan for the second half
- 04 Introduction to ML & DL



01 Course Logistics

Time

- Lecture: Wednesday 9:30 - 11:30 (LT27)
- Tutorial: Wednesday 11:30 - 12:30 (LT27)

Materials: will be uploaded before each lecture

Pre-requisites

- Basic probability and statistics
- Multivariate Calculus, linear algebra
- Fundamentals of machine learning



01 Course Logistics

Objectives

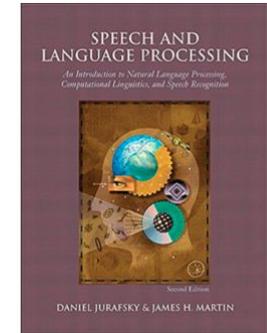
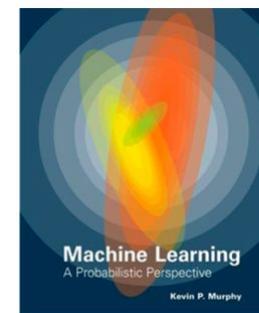
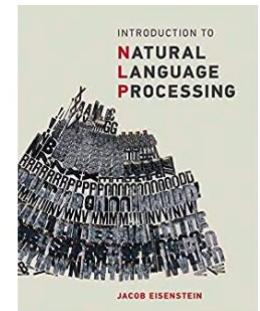
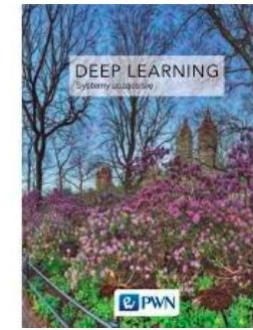
- Understand deep learning methods in NLP
- Be familiar with typical NLP applications
- Be able to build simple deep learning systems to solve NLP problems
- Be aware of and adapt to emerging trends of NLP
- Understand key challenges for current techniques



01 Course Logistics

Optional Textbooks

- Deep Learning by Goodfellow, Bengio, and Courville
- Machine Learning – A Probabilistic Perspective by Kevin Murphy
- Natural Language Processing by Jacob Eisenstein
- Speech and Language Processing by Dan Jurafsky and James H. Martin





01 Course Logistics

Evaluation

- 15% midterm quiz (for 1st half, week 8 tutorial)
- 35% group assignment (released)
- 50% final exam



Outline for today

- 01 Course Logistics**
- 02 What is NLP (a quick recap)**
- 03 Plan for the second half**
- 04 Introduction to ML & DL**



02

What is NLP?

“We study formalisms, models and algorithms to allow computers to perform **useful tasks** involving knowledge about human languages.”



Useful Tasks



Text classification

sentiment analysis, spam detection, topic classification



Information extraction

entity and relation recognition, event extraction



Structured prediction

dependency parsing, semantic role labeling



Text generation

summarization, conversation



Text Classification - Sentiment



The dessert is excellent.



Service was quite slow.



Good for a quick meal, but nothing special.



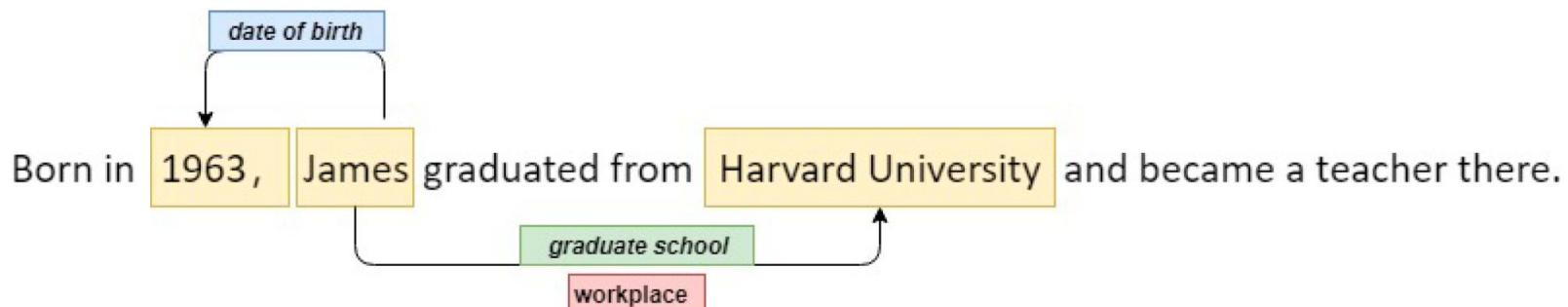
Completely lacking in good taste, good service, and good ambience.





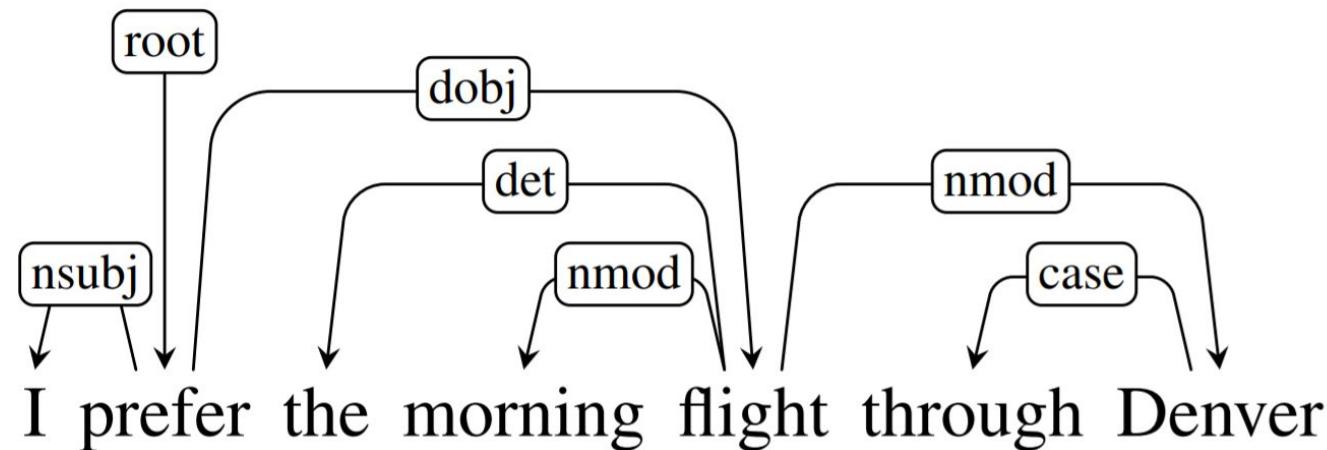
Information Extraction

In December 1903 **DATE** the Royal Swedish Academy of Sciences **ORG** awarded Marie **PERSON** and Pierre Curie **PERSON**, along with Henri Becquerel **PERSON**, the Nobel Prize in Physics **WORK_OF_ART**.





Structured Prediction - Parsing





Text Generation - Summarization

(a) Extractive Summarization

Source Text: Peter and Elizabeth took a taxi to attend the night party in the city.

While in the party, Elizabeth collapsed and was rushed to the hospital.

Summary: Peter and Elizabeth attend party city. Elizabeth rushed hospital.

(b) Abstractive Summarization

Source Text: Peter and Elizabeth took a taxi to attend the night party in the city.

While in the party, Elizabeth collapsed and was rushed to the hospital.

Summary: Elizabeth was hospitalized after attending a party with Peter.



Useful Tasks



Question answering

Reading comprehension,
open-domain



Multimodal

Image-to-text, text-to-speech
synthesis



Machine translation

Source language to target
language



Knowledge graph

KG construction, link
prediction



Reading Comprehension

Passage:

The two halves of the city were actually founded and plotted as separate cities, but soon grew together. The north side is characterized by very diverse and fashionable urban neighborhoods near the city center and sprawling suburbs further north. South Oklahoma City is generally more blue collar working class and significantly more industrial, having grown up around the Stockyards and meat packing plants at the turn of the century, and is currently the center of the city's rapidly growing Latino community.

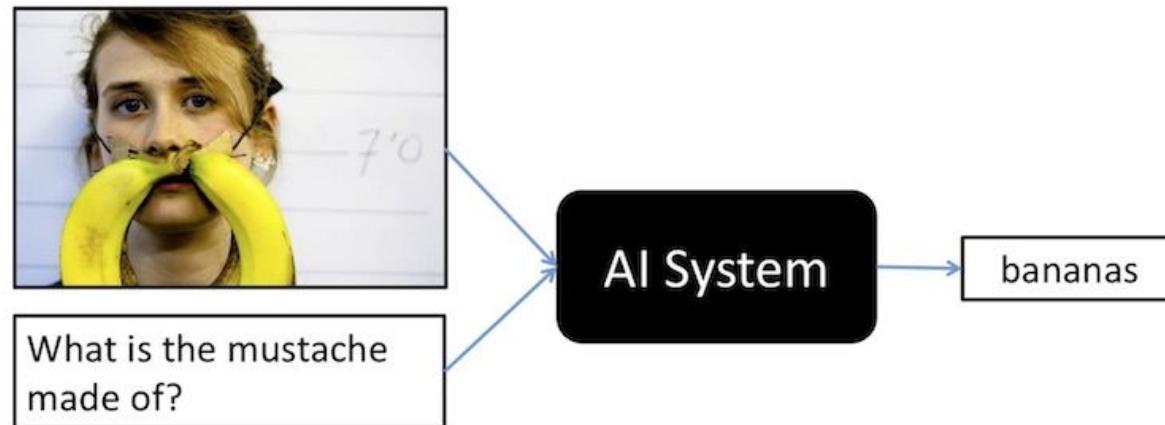
Question:

Which side is more urban and fashionable?

Gold Answer: North Oklahoma City



Multimodal - VQA





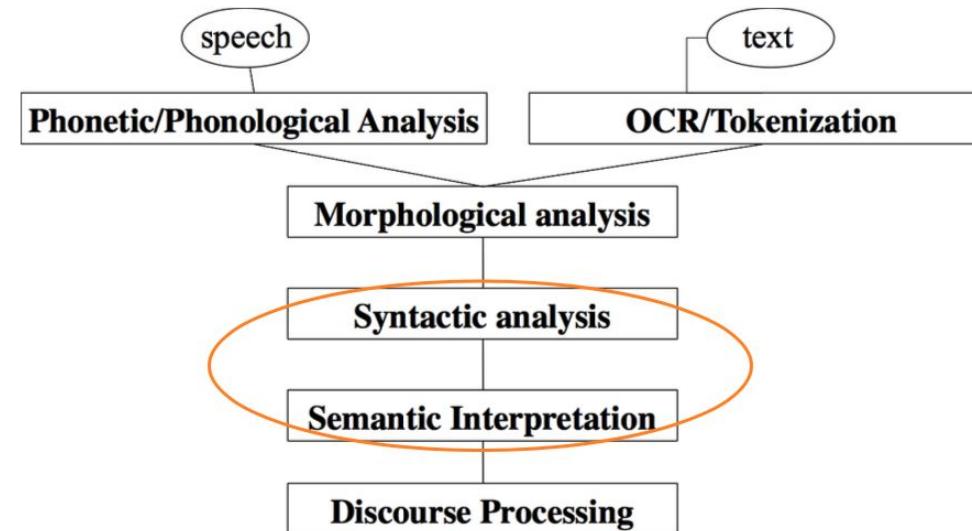
02

What is NLP?

“We study formalisms, models and algorithms to allow computers to perform useful tasks involving knowledge about human languages.”



Knowledge about Languages

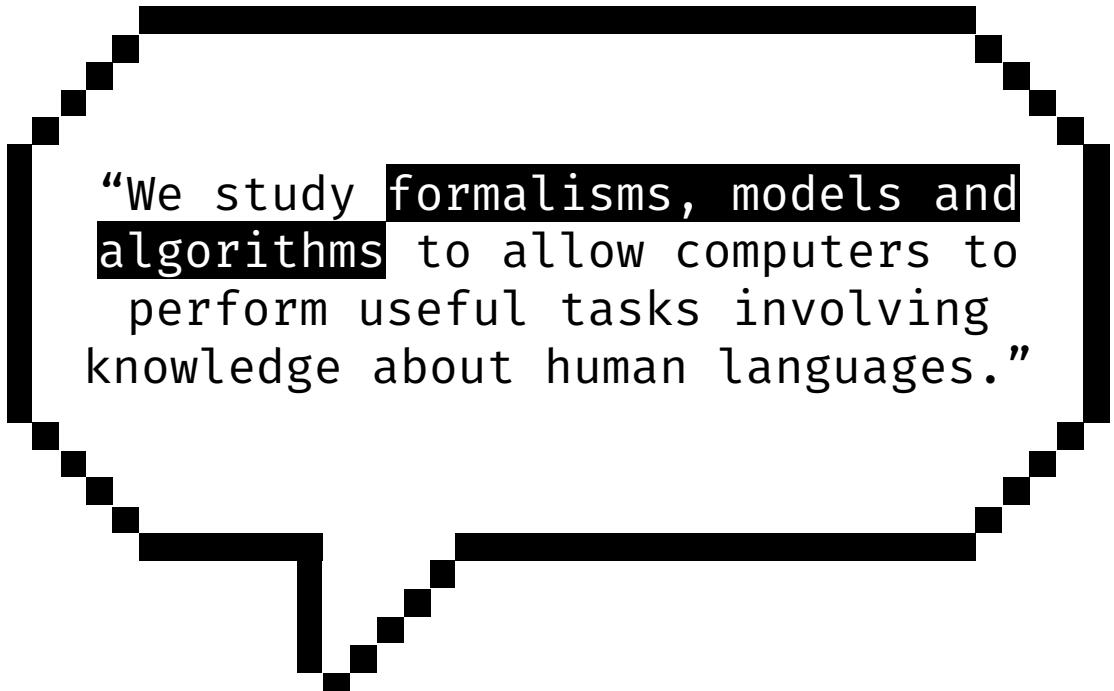


Mainly covered in the first half, not this one



02

What is NLP?



“We study **formalisms, models and algorithms** to allow computers to perform useful tasks involving knowledge about human languages.”

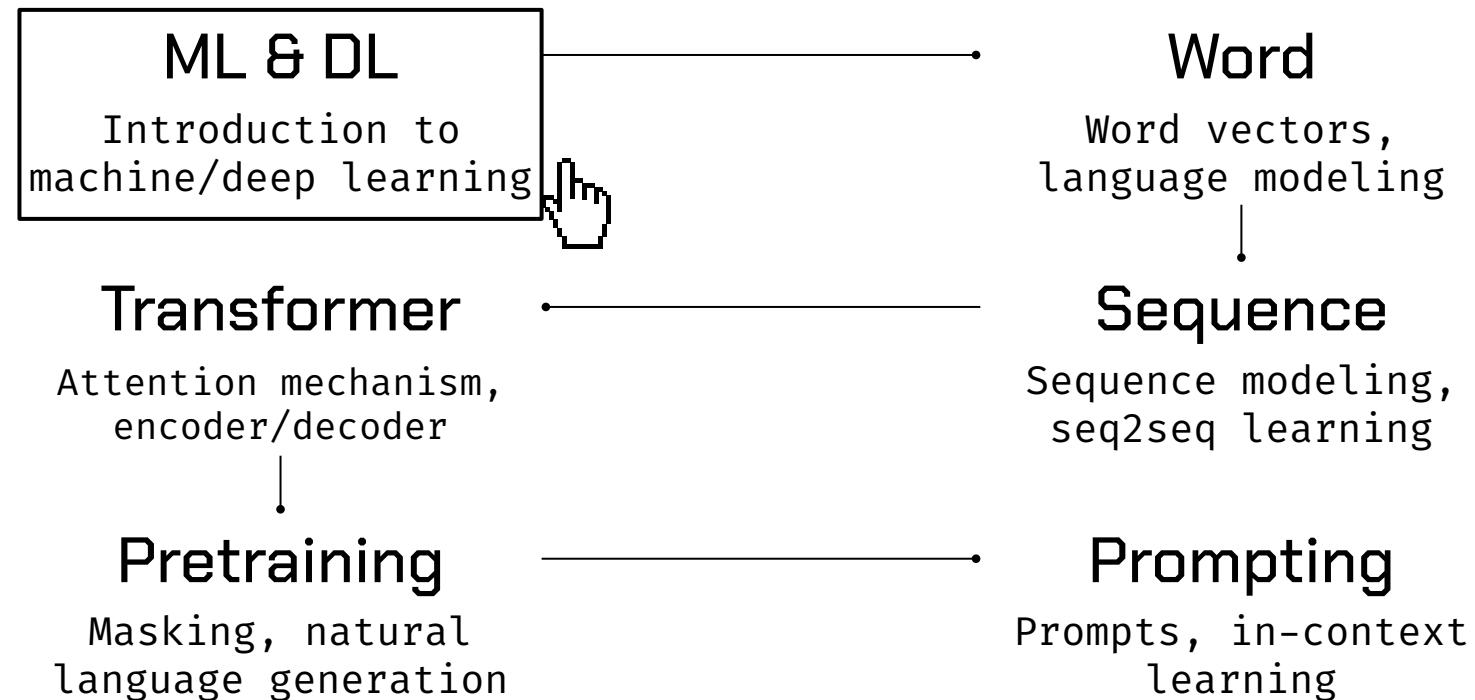


Outline for today

- 01 Course Logistics**
- 02 What is NLP (a quick recap)**
- 03 Plan for the second half**
- 04 Introduction to ML & DL**



Modules we will cover





Outline for today

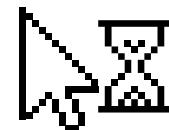
- 01 Course Logistics**
- 02 What is NLP (a quick recap)**
- 03 Plan for the second half**
- 04 Introduction to ML & DL**



Introduction to ML & DL

Machine Learning

- Definition
- Paradigms of machine learning
- Machine learning pipeline
- Typical algorithms
 - Linear regression
 - Logistic regression
- Optimization - gradient descent





What is Machine Learning?

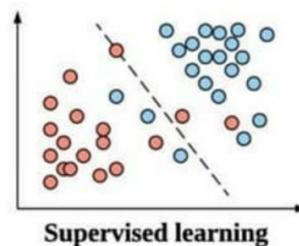
“A set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty.”



Paradigms of Machine Learning

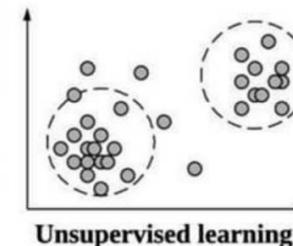
Supervised Learning

Learning a function that maps input data to output data, based on **labeled** examples.



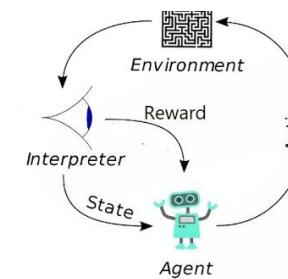
Unsupervised Learning

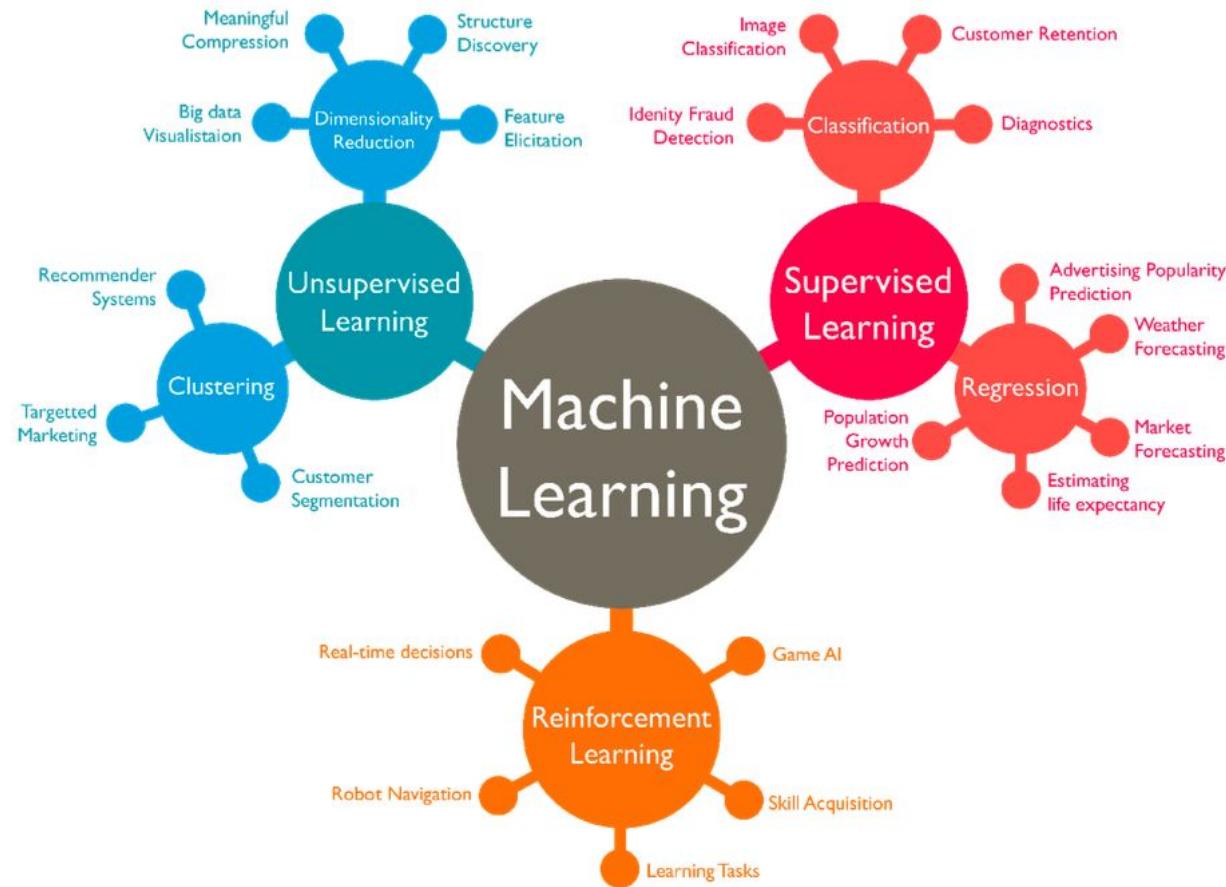
Finding hidden patterns or intrinsic structures in **unlabeled** data.



Reinforcement Learning

learning how to act or behave when given occasional **reward** or punishment signals







Supervised Learning

Classification: To learn a mapping from inputs x to outputs y , where $y \in \{1, \dots, C\}$, with C being the number of classes, which means y is categorical.

- If $C = 2$, this is called **binary** classification (in which case we often assume $y \in \{0, 1\}$)
- If $C > 2$, this is called **multiclass** classification.

Examples

- (1) Document classification and email spam filtering
- (2) Sentiment classification
- (3) Image classification and handwriting recognition
- (4) Face detection and recognition



Supervised Learning

Regression: To learn a mapping from inputs x to outputs y , where y is **real-valued**.

Examples

1. Predict tomorrow's stock market price given current market conditions and other possible side information.
2. Predict the age of a viewer watching a given video on YouTube.
3. Predict the location in 3d space of a robot arm end effector, given control signals (torques) sent to its various motors.
4. Predict the temperature at any location inside a building using weather data, time, door sensors.



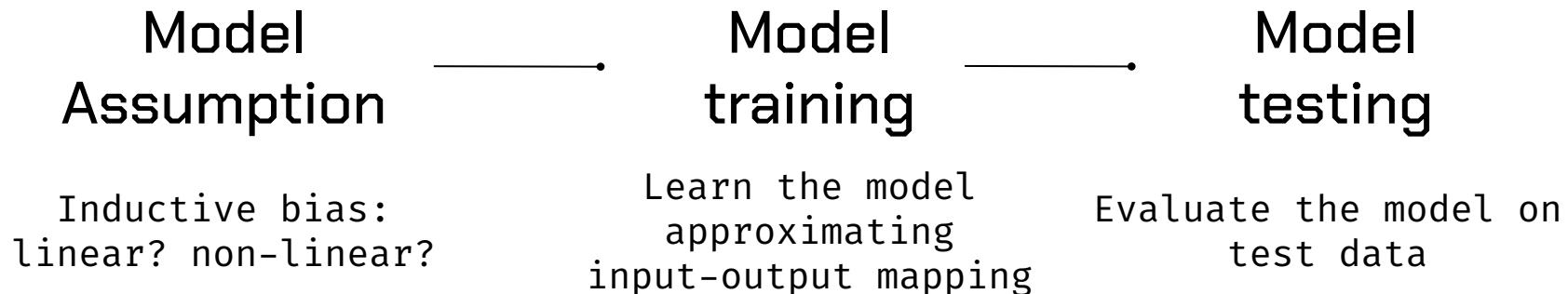
Unsupervised Learning

We are just given input data, without any outputs.
The goal is to discover **interesting structure** in the data; this is sometimes called knowledge discovery.

1. **Discovering clusters:** To cluster data into groups and estimate which cluster each data sample belongs to.
2. **Discovering latent factors:** To project the data to a lower dimensional subspace which captures the essence of the data. It is also called "Dimension Reduction". (eg, PCA)
3. **Discovering graph structure.**



Machine Learning Pipeline





Linear Regression

Problem Setup

Training Data is represented by:

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$$

Supervised

Here,

\mathcal{D} is called the **training set**.

N is the **number of training example**.

\mathbf{x}_i is a **d -dimensional vector of features** (also called **attributes** or **covariates**) i.e. #features is d .

y_i is called **output or response variable**.

For regression: y_i is **real-valued** i.e. $y_i \in \mathbb{R}$



Linear Regression

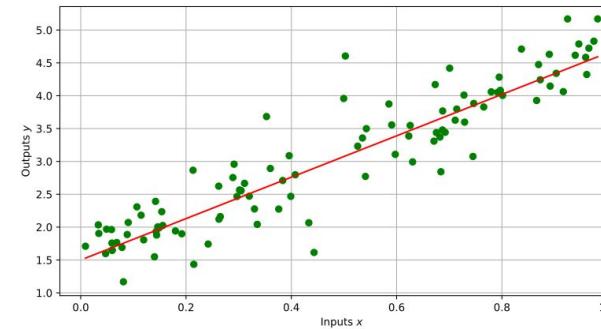
What is it?

The basic idea behind regression is that, you want to model the relationship between a real valued outcome variable y , and a vector of explanatory variables $\mathbf{x} = (x_1, x_2, \dots, x_n)$.

A linear regression relates y to a linear predictor function of \mathbf{x} .

For a given data point i , the linear function is of the form:

$$\hat{y}_i = w_0 + w_1 x_{i1} + \dots + w_d x_{id}$$





Linear Regression

Least Square Formulation of the Loss

In this formulation, the least squares fit is the line that **minimizes the sum of the squared distances** between observed data and predicted values, i.e. it minimizes the **Residual Sum of Squares (RSS)**:

$$\arg \min_{\mathbf{w}} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

Predicted output



Linear Regression

Least Square Formulation

So our **objective function** is:

$$J(\theta) = \arg \min_{\theta} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

$$= \sum_{i=0}^N (y_i - \mathbf{x}_i^\top \mathbf{w})^2 = (\mathbf{y} - \mathbf{X}\mathbf{w})^\top (\mathbf{y} - \mathbf{X}\mathbf{w})$$

A handwritten diagram illustrating vector notation. It shows three vectors: y (a column vector), x (a column vector), and w (a column vector). Below y , there is a bracket labeled \hat{y}_i indicating the predicted value for each y_i . A red arrow points from the label \hat{y}_i to the bracket under y .

Vector notation



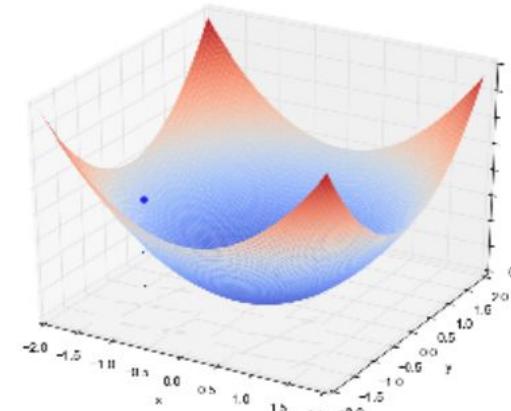
Linear Regression

Least Square Formulation

Notice that, we want to minimize $J(\theta)$.

We see that $J(\theta)$ is **Quadratic**. Hence it will be **Bowl Shaped**.

$$(y_i - x^T w)^2$$





Linear Regression (Optional)

Least Square Formulation

$$\begin{aligned} J(\theta) &= (y - \mathbf{X}\mathbf{w})^\top(y - \mathbf{X}\mathbf{w}) = (y^\top - \mathbf{w}^\top\mathbf{X}^\top)(y - \mathbf{X}\mathbf{w}) \\ &= y^\top y - y^\top \mathbf{X}\mathbf{w} - \mathbf{w}^\top \mathbf{X}^\top y + \mathbf{w}^\top \mathbf{X}^\top \mathbf{X}\mathbf{w} \end{aligned}$$

Taking partial derivative with respect to \mathbf{w} :

$$\begin{aligned} \frac{\partial}{\partial \mathbf{w}} J(\theta) &= \frac{\partial}{\partial \mathbf{w}} (y^\top y - \cancel{y^\top \mathbf{X}\mathbf{w}} - \mathbf{w}^\top \cancel{\mathbf{X}^\top y} + \mathbf{w}^\top \cancel{\mathbf{X}^\top \mathbf{X}\mathbf{w}}) \\ &= \underline{0} - 2\mathbf{X}^\top y + \underline{2\mathbf{X}^\top \mathbf{X}\mathbf{w}} \end{aligned}$$

Equating $\frac{\partial}{\partial \mathbf{w}} J(\theta)$ to 0 we get,

$$\begin{aligned} \frac{\partial}{\partial \mathbf{w}} J(\theta) = 0 &= -2\mathbf{X}^\top y + \underline{2\mathbf{X}^\top \mathbf{X}\mathbf{w}} \\ \mathbf{w} &= (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top y \end{aligned}$$

Closed form Solution



Linear Regression

Probabilistic Formulation

If we ever want to understand linear regression from a **Bayesian perspective** we need to start thinking **probabilistically**.

We need to *flip things over* and instead of thinking about the line minimizing the cost, think about it as maximizing the likelihood of the observed data.

As we will see, this amounts to the **exact same thing** - *mathematically speaking* - it's just a different way of looking at it.



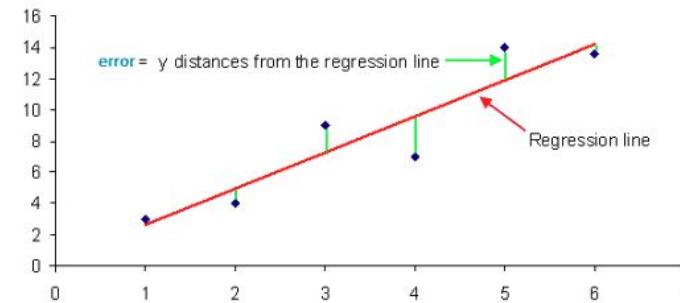
Linear Regression

Least Square Formulation

We can write the **actual value** of the i^{th} observation as:

$$y_i = w_0 + w_1 x_i + \epsilon_i$$

where ϵ_i is the **residual, or error**, we get for the i^{th} observation, i.e. the difference between our linear predictions and the true response.





Linear Regression

Probabilistic Formulation

If we ever want to understand linear regression from a **Bayesian perspective** we need to start thinking **probabilistically**.

We have seen previously:

$$y_i = \mathbf{w}^T \mathbf{x}_i + \epsilon_i$$

We often assume that ϵ_i normally distributed with a mean of 0 i.e.

$$\epsilon_i \sim \mathcal{N}(0, \sigma^2)$$

So, From the above equation we can say that:

$$y_i \sim \mathcal{N}(\mathbf{w}^T \mathbf{x}_i, \sigma^2)$$

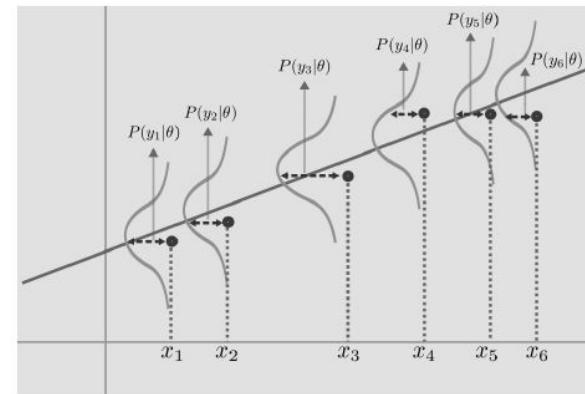


Linear Regression

Probabilistic Formulation

It is common to assume the training examples are **Independent and Identically Distributed**, commonly abbreviated to **IID**.

So, we assume all our training examples are **Normally distributed** with **mean $\mathbf{w}^T \mathbf{x}_i$** and **variance σ^2** .





Linear Regression

Maximum Likelihood Estimation

We can write Linear regression as a model of the form:

$$p(y|\mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(y|\mathbf{w}^T \mathbf{x}, \sigma^2)$$

Maximum likelihood Estimation

A common way to estimate the parameters of a statistical model is to compute the [Maximum likelihood Estimation\(MLE\)](#), which is defined as:

$$\hat{\boldsymbol{\theta}} \triangleq \operatorname{argmax}_{\boldsymbol{\theta}} \log p(\mathcal{D}|\boldsymbol{\theta})$$



Linear Regression

Maximum Likelihood Estimation

$$\begin{aligned}\ell(\theta) &\triangleq \log p(\mathcal{D}|\theta) = \sum_{i=1}^N \log p(y_i|\mathbf{x}_i, \theta) \\ &= \sum_{i=1}^N \log \left[\left(\frac{1}{2\pi\sigma^2} \right)^{\frac{1}{2}} \exp \left(-\frac{1}{2\sigma^2} (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 \right) \right] \\ &= \frac{-1}{2\sigma^2} RSS(\mathbf{w}) - \frac{N}{2} \log(2\pi\sigma^2)\end{aligned}$$

RSS stands for Residual Sum of Squares $RSS(\mathbf{w}) \triangleq \sum_{i=1}^N (y_i - \mathbf{w}^\top \mathbf{x}_i)^2$



Linear Regression

rewatch why we need to use log here

Maximum Likelihood Estimation

$$\begin{aligned}\ell(\theta) &\triangleq \log p(\mathcal{D}|\theta) = \sum_{i=1}^N \log p(y_i|\mathbf{x}_i, \theta) \\ &= \sum_{i=1}^N \log \left[\left(\frac{1}{2\pi\sigma^2} \right)^{\frac{1}{2}} \exp \left(-\frac{1}{2\sigma^2} (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 \right) \right] \\ &= \frac{-1}{2\sigma^2} RSS(\mathbf{w}) - \frac{N}{2} \log(2\pi\sigma^2)\end{aligned}$$

Least Square Formulation

RSS stands for Residual Sum of Squares

$$RSS(\mathbf{w}) \triangleq \sum_{i=1}^N (y_i - \mathbf{w}^\top \mathbf{x}_i)^2$$



Logistic Regression

Classification in NLP

- News stories: politics? sports? business? technology?
- Reviews of films, restaurants, products: positive? negative?
- Email, arXiv submissions: spam? Not spam?
- Will a piece of proposed legislation pass?
- What dialect is a text written in?
- Does the text contain content that will likely offend end people?



Logistic Regression

Problem Setup

Training Data is represented by

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N \quad \text{Supervised}$$

Here

\mathcal{D} is called the **training set**.

N is the **number of training examples**.

x_i is a **d-dimensional vector of features** (also called attributes or covariates) i.e. #features is d.

y_i is called **output or response variable**.

For classification: y_i is **categorical** i.e. $y_i \in \{1, \dots, C\}$

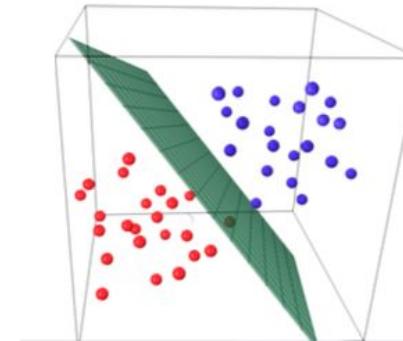


Logistic Regression

y_i is **categorial** i.e. $y_i \in \{1, \dots, C\}$

First, we will consider the case of **binary response variable** that is, where it can take only two values, "0" and "1".

Cases where the response variable has more than two outcome categories may be analyzed in **Multinomial Logistic Regression**.





Logistic Regression

Linear regression for binary outcomes? No!

- The relationship between x and y is not linear. $y \neq x^T w + \epsilon$
- The response y is not normally distributed. $y \notin N(x^T w, \sigma)$
- Linear models cannot constrain y to be 0 or 1. $y \in \mathbb{R}$

What do we need?

- We are interested in $p(y_i = 1 | \mathbf{x}_i, \mathbf{w}) \in [0, 1]$
- Replace normal distribution with Bernoulli distribution, imaging we are tossing a coin, p =probability of success.

$$y \sim \text{Bernoulli}(p) \in \{0, 1\}$$



Logistic Regression

Ensure outputs to be probabilities $p(y_i = 1 | \mathbf{x}_i, \mathbf{w}) \in [0, 1]$

Sigmoid Function

Sigmoid is a **squashing function**.

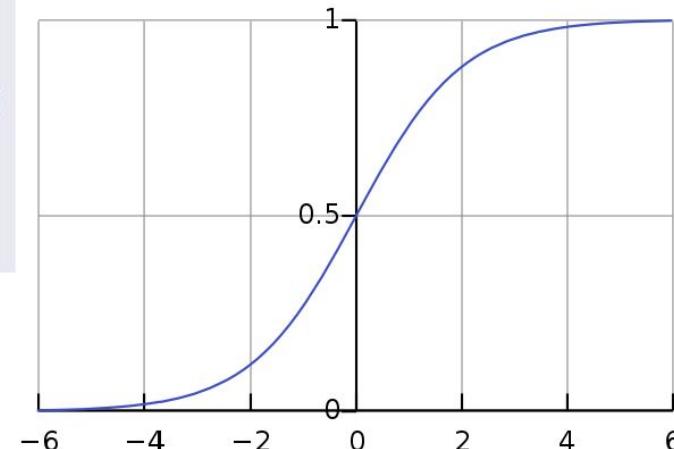
It maps the whole real line to $[0, 1]$.

It is also known as *logistic* or *logit* function

The term “sigmoid” means S-shaped. This is defined as:

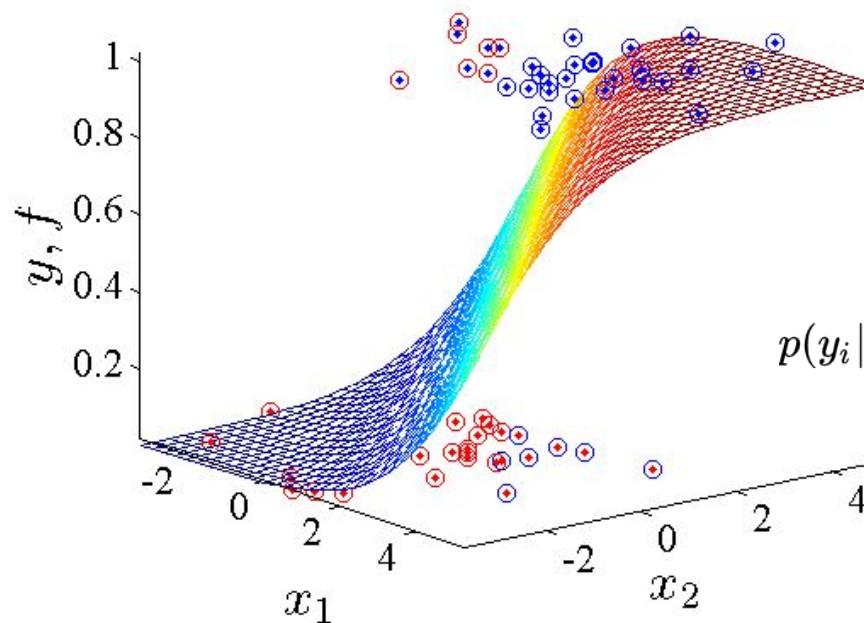
$$\text{sigm}(\eta) \triangleq \frac{1}{1 + \exp(-\eta)} = \frac{e^\eta}{e^\eta + 1}$$

$$\sigma(\eta) = \frac{1 - e^{-\eta}}{1 - e^{-\eta} + e^{-\eta} - e^{-2\eta}}$$



Logistic Regression

Combine sigmoid function with Bernoulli distribution



$$\mathcal{P}(y|\mathbf{x}, \mathbf{w}) = \text{Bernoulli}(y|\text{sigm}(\mathbf{w}^\top \mathbf{x}))$$

$$p(y_i = 1|\mathbf{x}_i, \mathbf{w}) = \text{sigm}(\mathbf{w}^\top \mathbf{x}_i)$$

$$p(y_i = 0|\mathbf{x}_i, \mathbf{w}) = 1 - \text{sigm}(\mathbf{w}^\top \mathbf{x}_i)$$

$$p(y_i|\mathbf{x}_i, \mathbf{w}) = \text{sigm}(\mathbf{w}^\top \mathbf{x}_i)^{y_i} \cdot (1 - \text{sigm}(\mathbf{w}^\top \mathbf{x}_i))^{1-y_i}$$



Logistic Regression

Maximum Likelihood Estimation

if multi-nomial with 3 classes require 3 weights, why binary logistic regression only require 1 weight?

$$\begin{aligned}\mathbf{w}^* &= \operatorname{argmax}_{\mathbf{w}} \log p(\mathcal{D}|\mathbf{w}) \\ &= \operatorname{argmax}_{\mathbf{w}} \sum_{i=1}^N \log p(y_i | \mathbf{x}_i, \mathbf{w}) \quad \boxed{\mu_i = \operatorname{sigm}(\mathbf{w}^\top \mathbf{x}_i)} \\ &= \operatorname{argmax}_{\mathbf{w}} \sum_{i=1}^N \log(\mu_i^{y_i} \cdot (1 - \mu_i)^{1-y_i}) \\ \text{Negative Log Likelihood } \xrightarrow{\text{NLL}} &= \operatorname{argmin}_{\mathbf{w}} - \sum_{i=1}^N (y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i)) \quad \text{note how 1 becomes zero}\end{aligned}$$

This is also called the Cross Entropy Error Function.



Logistic Regression (Optional)

Unlike linear regression, we can no longer write down the MLE in **closed form**. Instead, we need to use an **optimization algorithm** to compute it.

For this, we need to derive the **gradient** and **Hessian**.

$$\mathbf{g} = \frac{d}{d\mathbf{w}} f(\mathbf{w}) = - \sum_i (\mu_i - y_i) \mathbf{x}_i = \mathbf{X}^\top (\boldsymbol{\mu} - \mathbf{y})$$

$$\mathbf{H} = \frac{d}{d\mathbf{w}} \mathbf{g}(\mathbf{w})^\top = - \sum_i (\nabla_{\mathbf{w}} \mu_i) \mathbf{x}_i^\top = - \sum_i \mu_i (1 - \mu_i) \mathbf{x}_i \mathbf{x}_i^\top = \mathbf{X}^\top \mathbf{S} \mathbf{X}$$

$$\text{where } \mathbf{S} \triangleq \text{diag}(\mu_i(1 - \mu_i))$$

One can also show that **H** is **positive definite**.

Hence the *NLL* is **convex** and has a **unique global minimum**.



Multiclass Logistic Regression

Logistic regression can be extended to handle more than two classes.
This is called **Multi-class logistic regression**.

Also called **Multinomial logistic regression**.

In this case, the response variable $y_i \in \{1, 2, \dots, C\}$

Different from binary logistic regression, we model the probability using
Softmax as

$$\mathcal{P}(y_i = c | \mathbf{x}_i, \mathbf{W}) = \mu_{ic} = \frac{\exp(\mathbf{w}_c^\top \mathbf{x}_i)}{\sum_{c'=1}^C \exp(\mathbf{w}_{c'}^\top \mathbf{x}_i)}$$

take note that each weight correspond to a class
so if there are 3 class, W is a matrix of 3

$$\mathcal{P}(y_i | \mathbf{x}_i, \mathbf{W}) = \prod_{c=1}^C \mu_{ic}^{y_{ic}}$$



Multiclass Logistic Regression

Maximum Likelihood Estimation

$$\mathbf{W}^* = \operatorname{argmax}_{\mathbf{W}} \log p(\mathcal{D} | \mathbf{W})$$

$$= \operatorname{argmax}_{\mathbf{W}} \sum_{i=1}^N \log p(y_i | \mathbf{x}_i, \mathbf{W})$$

this is the diff

$$= \operatorname{argmax}_{\mathbf{W}} \sum_{i=1}^N \log \prod_{c=1}^C \mu_{ic}^{y_{ic}}$$

$$\mathbf{W} = [\mathbf{w}_1; \dots; \mathbf{w}_C]$$

$$y_{ic} = \mathbb{I}(y_i = c)$$

$$= \operatorname{argmax}_{\mathbf{W}} \sum_{i=1}^N \sum_{c=1}^C y_{ic} \log \mu_{ic}$$

this is the cross entropy formulation

$$= \operatorname{argmax}_{\mathbf{W}} \sum_{i=1}^N \left[\left(\underbrace{\sum_{c=1}^C y_{ic} \mathbf{w}_c^\top \mathbf{x}_i}_{\text{"hope"}} \right) - \log \left(\underbrace{\sum_{c'=1}^C \exp(\mathbf{w}_{c'}^\top \mathbf{x}_i)}_{\text{"fear"}} \right) \right]$$

“hope”

“fear”



Gradient Descent

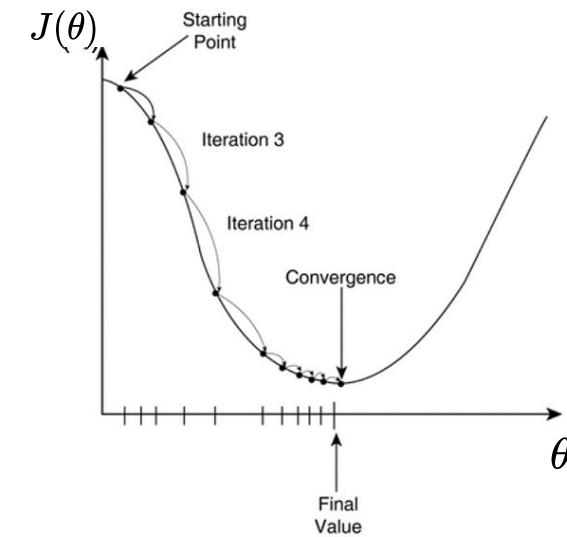
The most commonly used method for unconstrained optimization

Goal: minimize $J(\theta)$ with respect to θ

next step current step

$$\theta_{k+1} = \theta_k - \eta_k g_k$$

update





Gradient Descent

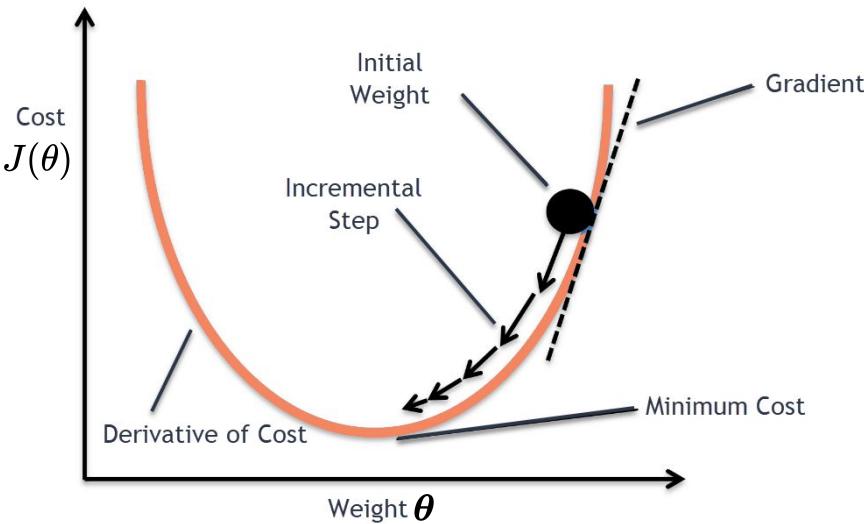
The most commonly used method for unconstrained optimization

Goal: minimize $J(\theta)$ with respect to θ

$$\theta_{k+1} = \theta_k - \eta_k g_k$$

step size / learning rate

gradient





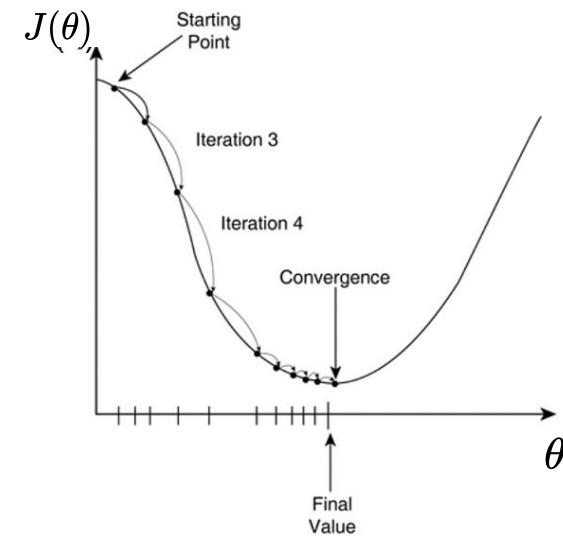
Gradient Descent

The most commonly used method for unconstrained optimization

Goal: minimize $\sum_{i=1}^N J_i(\theta)$ with respect to θ

because we have N data points, we want
minimise for all data points

$J_i(\theta)$ is the loss for the ith data





Gradient Descent

Batch Gradient Descent uses the entire dataset to compute the a single gradient update

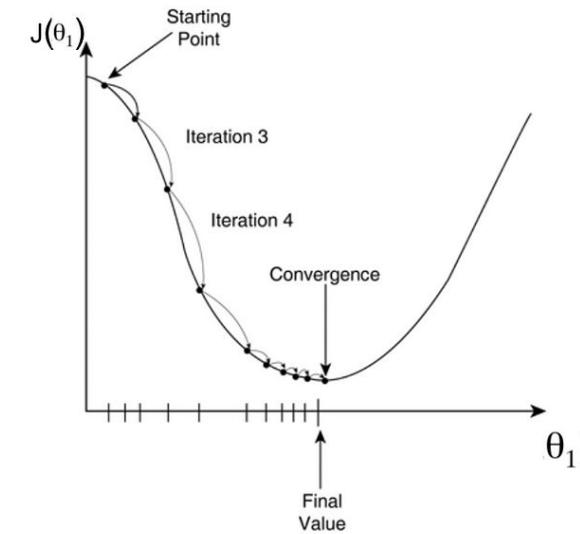
For $t \in \{1, \dots, T\}$:

Epoch index

$$\theta \leftarrow \theta - \eta \cdot \nabla_{\theta} \frac{1}{N} \sum_{i=1}^N J_i$$

1 gradient for entire dataset
make 1 large update

Output: θ





Gradient Descent

Stochastic Gradient Descent (SGD) updates parameters after every single data point.

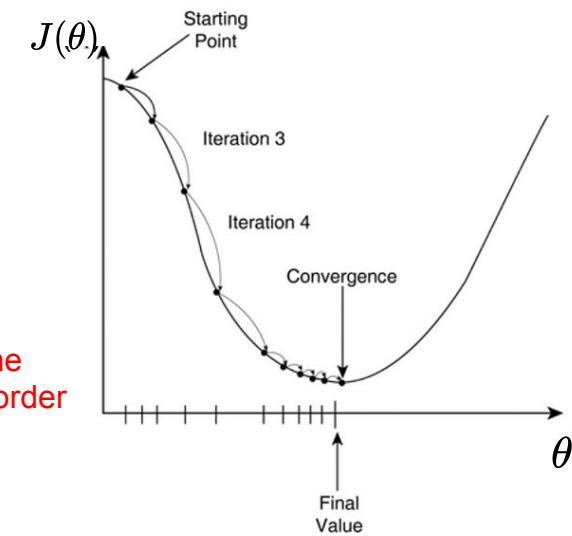
For $t \in \{1, \dots, T\}$:

- ▶ Choose a random permutation π of $\{1, \dots, N\}$
- ▶ For $i \in \{1, \dots, N\}$: total there are N gradients

$$\theta \leftarrow \theta - \eta \cdot \nabla_{\theta} J_{\pi(i)}$$

data is permuted to ensure the model does not memorise the order of the dataset

Output: θ





Gradient Descent

Mini-batch Gradient Descent updates parameters after every batch of data points.

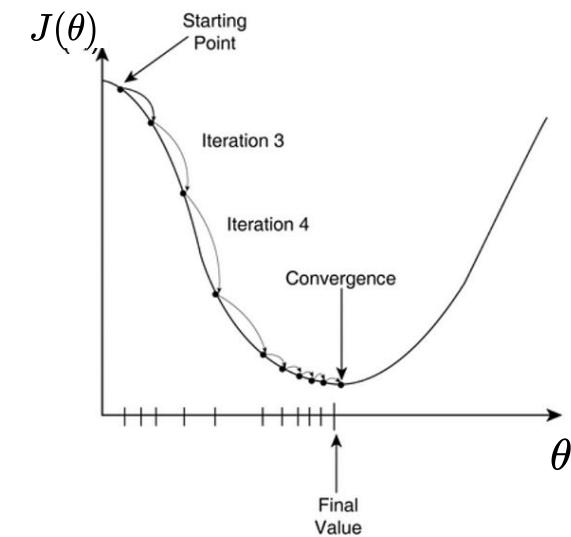
For $t \in \{1, \dots, T\}$: create s batch, so we minimise the gradient
compute

For each batch $\{b_1, \dots, b_S\} \subset \{1, \dots, N\}$

- ▶ Choose a batch of size S $\{x_{b_1}, \dots, x_{b_S}\}$
- ▶ Do batch gradient descent on this batch

$$\theta \leftarrow \theta - \eta \cdot \nabla_{\theta} \frac{1}{S} \sum_{i=b_1}^{b_S} J_i$$

Output: θ





Comparison

Method	Accuracy	Update Speed	Memory Usage	Online Learning
Batch gradient descent	Good	Slow	High	No
Stochastic gradient descent	Good (with annealing)	High	Low	Yes
Mini-batch gradient descent	Good	Medium	Medium	Yes

Table: Comparison of trade-offs of gradient descent variants

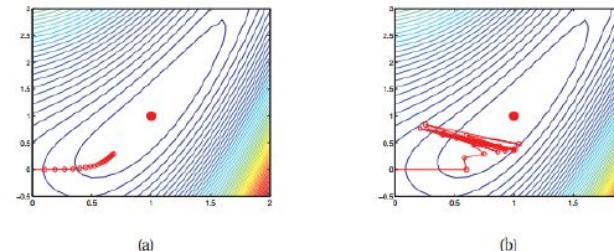


Gradient Descent

The main issue in gradient descent is:

How should we set the step size?

This turns out to be quite tricky. If we use a constant learning rate, but make it **too small**, convergence will be very slow, but if we make it **too large**, the method can fail to converge at all.





Gradient Descent Variants

Several ways to remedy this. learning rate

- Choose the optimal step size η by line-search.
- Add a momentum term to the updates.
- Use methods such as Conjugate Gradient
- Use second-order methods (e.g., Newtons method) to exploit the curvature of the objective function. This requires the Hessian matrix.

Use Adaptive Methods with Stochastic Gradient Descent



Gradient Descent Variants

- ① Momentum
- ② Nesterov accelerated gradient
- ③ Adagrad
- ④ Adadelta
- ⑤ RMSprop
- ⑥ Adam
- ⑦ Adam extensions

See <https://ruder.io/optimizing-gradient-descent/>



Example

Let's consider how to compute MLE for linear regression

The stochastic gradient at iteration k is given by

note this formulation of g here

$$g_k = 2\mathbf{x}_k(\theta_k^\top \mathbf{x}_k - y_k)$$

$$l(\theta) = \sum_{i=1}^N (y_i - \theta^\top \mathbf{x}_i)^2$$

Assuming k is the training example to use at iteration k .

After computing the gradient, we take a step along it as follows:

$$\theta_{k+1} = \theta_k - \eta_k g_k$$



Example

After computing the gradient, we take a step along it as follows:

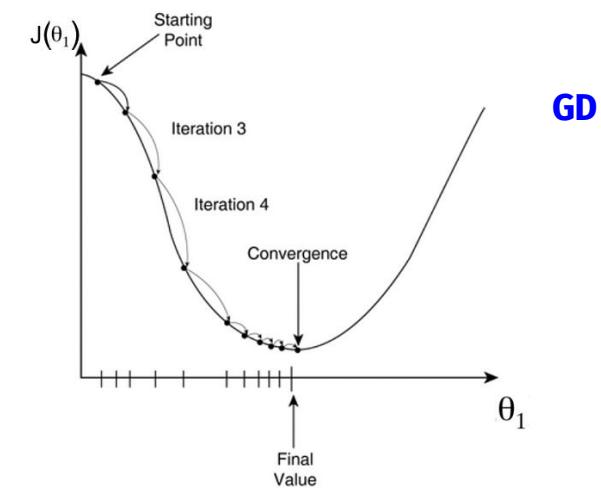
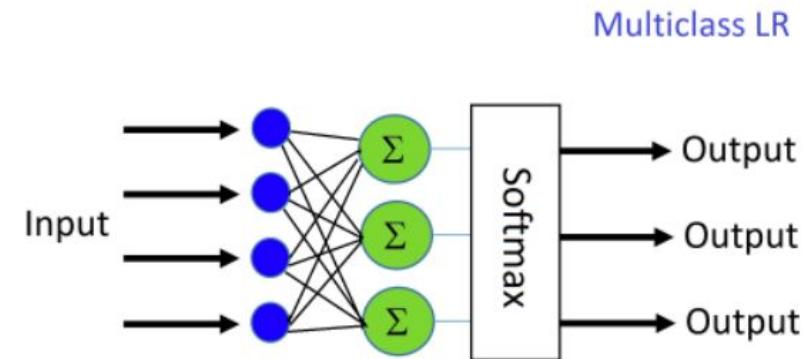
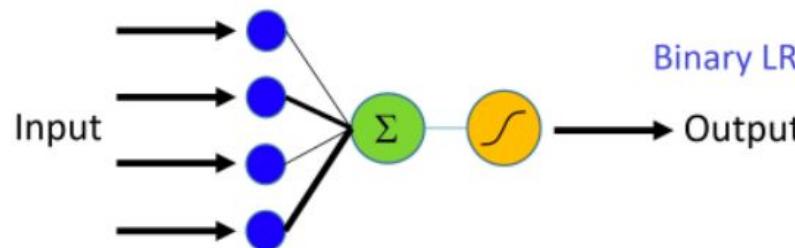
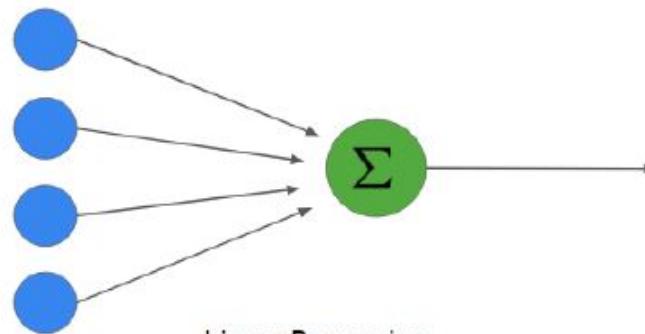
$$\theta_{k+1} = \theta_k - \eta_k g_k \quad g_k = 2\mathbf{x}_k(\theta_k^\top \mathbf{x}_k - y_k)$$

this is the difference of
prediction and truth,
larger gradient gives
larger update

Interpretation: it is the feature vector \mathbf{x}_k weighted by the difference between what we predicted, $\hat{y}_k = \theta_k^\top \mathbf{x}_k$, and the true response, y_k ; hence the gradient acts like an **error signal**.



Summary

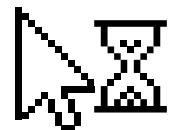




Introduction to ML & DL

Deep Learning

- What is Deep Learning and why
- From Logistic/Linear Regression to Neural Networks
- Non-linearity of Neural Networks
- How to train Neural Networks
- Regularisation (dropout, gradient clipping)





What is Deep Learning and Why

- Deep learning is a subfield of machine learning.
- Most machine learning methods work well because of human-designed representations/input features.
- Machine learning becomes just optimizing feature weights to make a good prediction.

features are learnt rather than engineered by the expert

Feature	NER
Current Word	✓
Previous Word	✓
Next Word	✓
Current Word Character n-gram	all
Current POS Tag	✓
Surrounding POS Tag Sequence	✓
Current Word Shape	✓
Surrounding Word Shape Sequence	✓
Presence of Word in Left Window	size 4
Presence of Word in Right Window	size 4

Features for NER (Finkel et al., 2010)

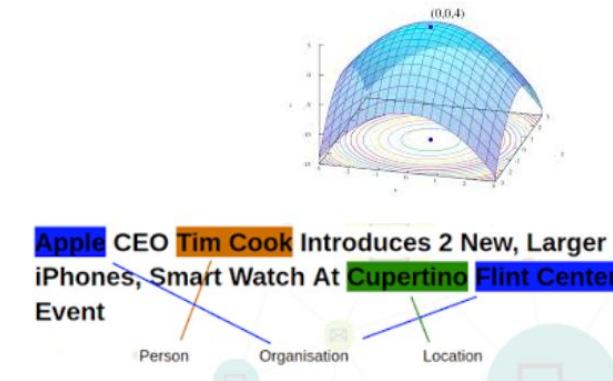
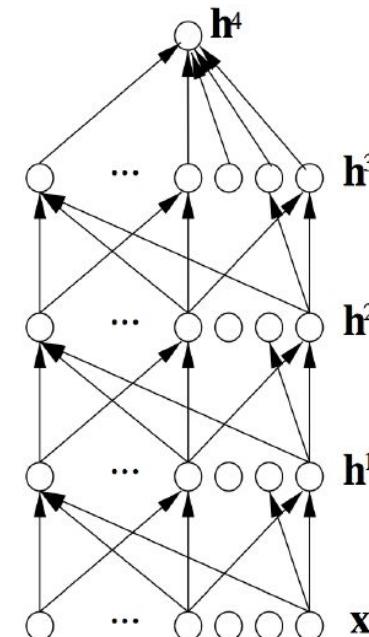


Figure 1: An example of NER application on an example text



What is Deep Learning and Why

- Representation learning attempts to automatically learn good features or representations
- Deep learning algorithms attempt to learn (multiple levels of) representations (here: h^1, h^2, h^3) and an output (h^4)
- From “raw” inputs x (e.g. sound, pixels, characters, or words)

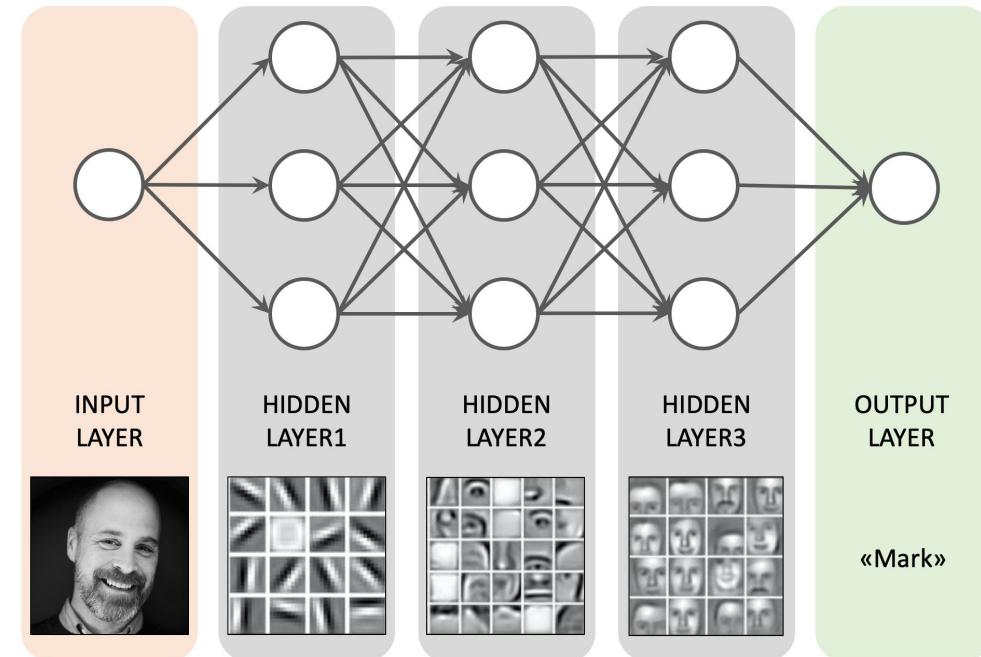




Deep Learning

Multiple hidden layers learn different abstractions of the input:

- Lower layers capture lower-level features, e.g., edges, parts
- Higher layers capture higher-level features, e.g., faces





What is Deep Learning and Why

Source: stanford 224n

- Manually designed features are often over-specified, incomplete and take a long time to design and validate
- **Learned Features** are easy to adapt, fast to learn
- Deep learning provides a very flexible, learnable framework for **representing** world, visual and linguistic information.
- Deep learning can learn in **unsupervised** (from raw text) and **supervised** (with specific labels like positive/negative) settings



What is Deep Learning and Why NLP

Source: stanford 224n

Combine NLP knowledge with representation learning and deep learning

Improvements in recent years in all levels

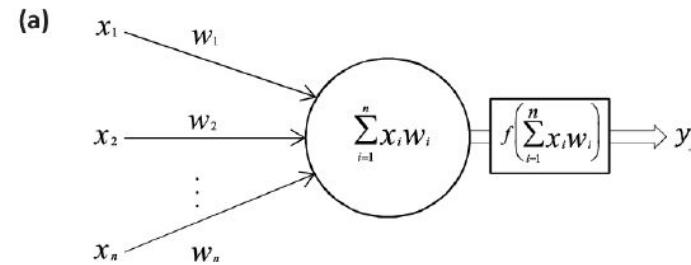
- **Linguistic levels:** speech, words, syntax, semantics, discourse
- **Full applications:** sentiment analysis, question answering, dialogue agents, machine translation



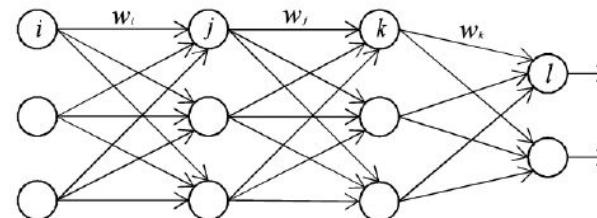
Neuron

if you understand how logistic regression works

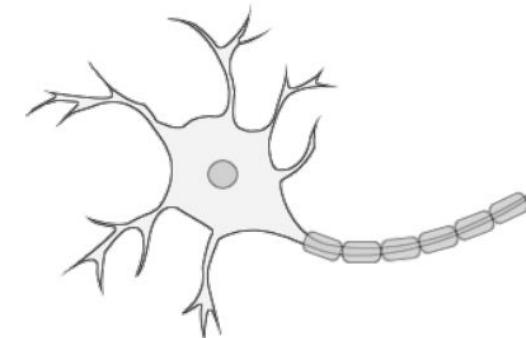
Then **you already understand** the operation of a basic neural network neuron!



(b) Input layer 1st hidden layer 2nd hidden layer Output layer



$$y_j = f\left(\sum x_i w_i\right) \quad y_k = f\left(\sum x_i w_j\right) \quad y_l = f\left(\sum x_i w_k\right)$$





From LR to NN

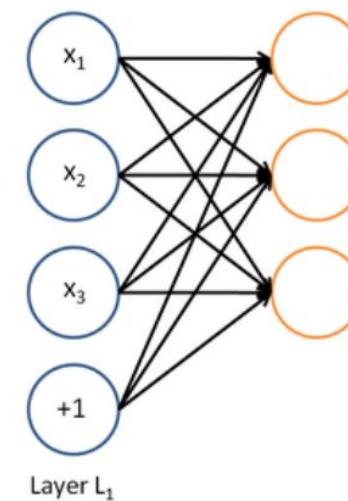
A neural network = running several logistic regressions at the same time

If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs ...

But these are not random variables

what does this mean?

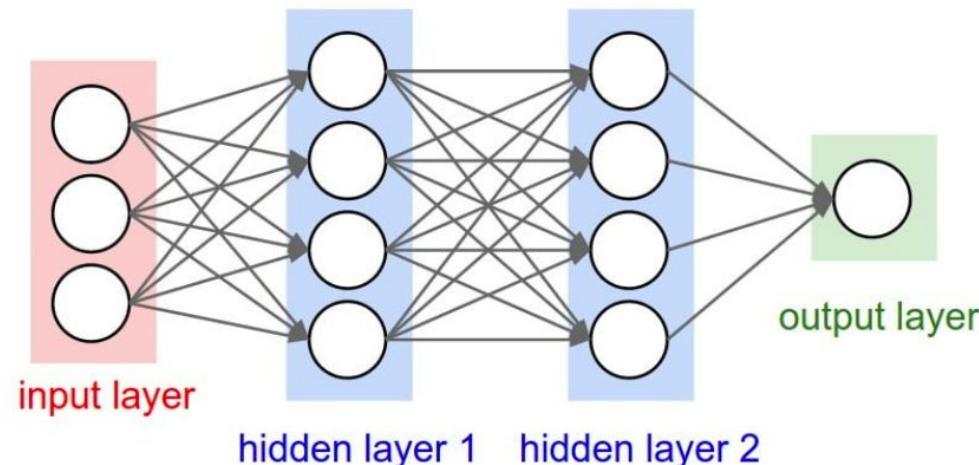
we are using the previous value from the sigmoid compute and this is a real value





From LR to NN

... which we can feed into another logistic regression function



Output layer can be a classification layer or a regression layer



From LR to NN

what if there was no bias term?

$$a_1 = f(W_{11}x_1 + W_{12}x_2 + W_{13}x_3 + b_1)$$

$$a_2 = f(W_{21}x_1 + W_{22}x_2 + W_{23}x_3 + b_2)$$

.....

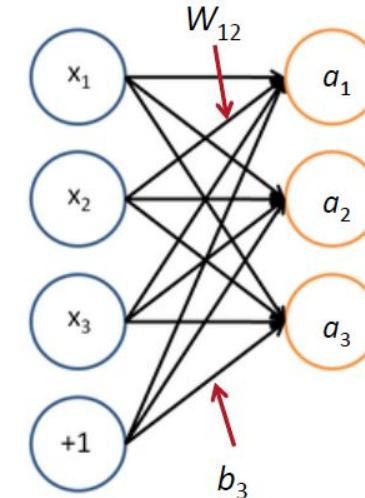
In Matrix Notation

$$z = Wx + b$$

$$a = f(z)$$

Where $f()$ is applied element-wise

$$f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)]$$

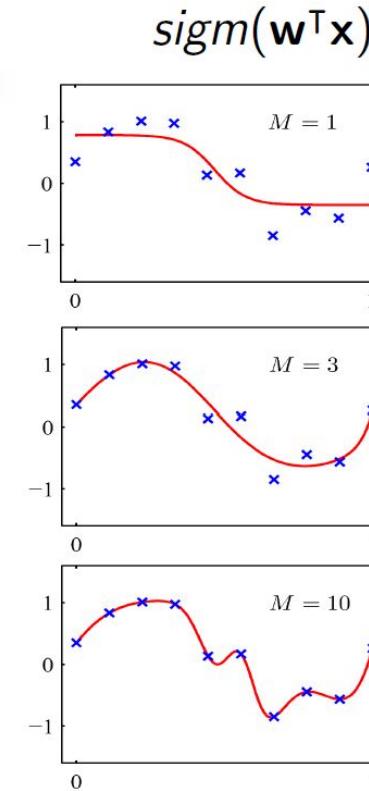


Activation function



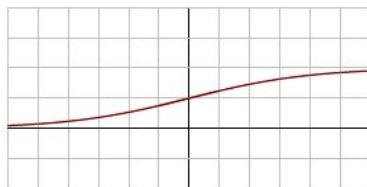
Why non-linearity

- For logistic regression: map to probabilities
- For NN: function approximation, e.g., regression or classification
 - Without non-linearities, NNs can't do anything more than a linear transform
 - Extra layers could just be compiled down into a single linear transform $\mathbf{W}(\mathbf{Wx}) = \mathbf{Vx}$

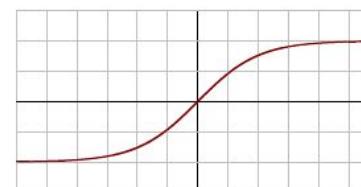




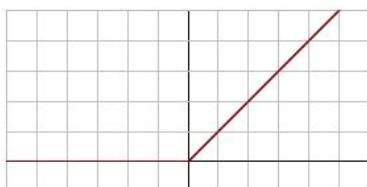
Activation functions



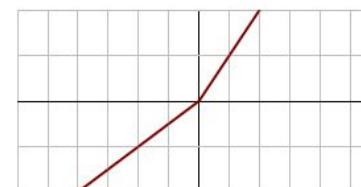
(a) Sigmoid



(b) tanh



(c) ReLU



(d) Leaky ReLU

- Sigmoid

$$\sigma(x) = \frac{1}{1 + \exp^{-x}}$$

- Hyperbolic Tangent:

$$\tanh(x) = \frac{\exp^x - \exp^{-x}}{\exp^x + \exp^{-x}}$$

- Rectified Linear Unit (ReLU):

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

- Leaky ReLU:

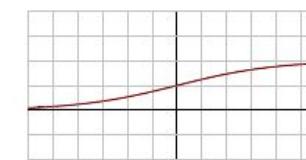
$$f(x) = \begin{cases} \alpha x & x < 0 \\ x & x \geq 0 \end{cases}$$

, where α is small constant

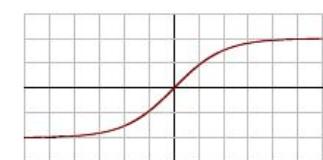


Activation functions

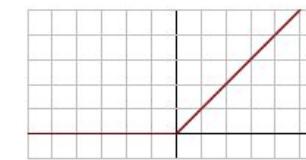
- Sigmoid
 - Biological analogy. Very popular before DL era.
 - Range: $[0, 1]$
 - Gradient vanishing \rightarrow losing popularity in DL era, but still used a lot e.g., LSTM
- Hyperbolic Tangent
 - Centered at 0
 - Gradient vanishing (slightly better than sigmoid function)
- Rectified Linear Unit (ReLU)
 - Alleviate gradient vanishing problem. Safe choice in general.
 - Piece-wise linear
- Leaky ReLU: variant of ReLU



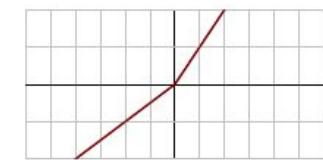
(a) Sigmoid



(b) tanh



(c) ReLU



(d) Leaky ReLU



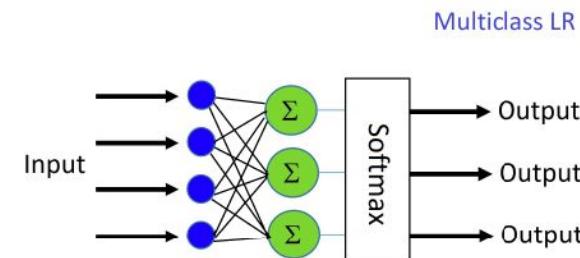
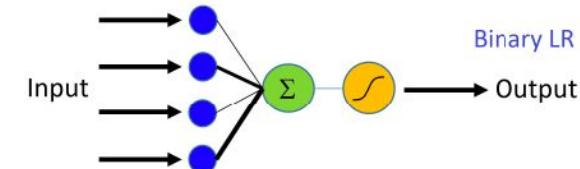
How to Train Neural Networks

If we have only one layer (linear)

It is same as Linear/Logistic Regression

- Use Gradient Descent
- Or L-BFGS (2nd order)
- Or SGD

See the first part of the slides
for details.



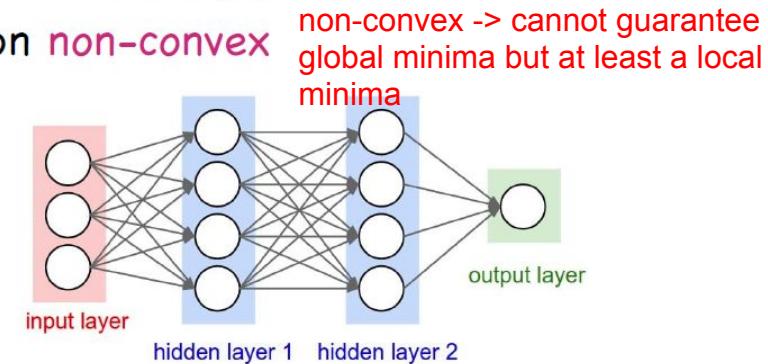


How to Train Neural Networks

But we have internal (hidden) layers in a multi-layer NN

This makes the loss/objective function **non-convex**

- But, we can still use the same ideas/algorithms (just without guarantees)
- Use SGD or its variants
- We “**backpropagate**” error derivatives through the model



Algorithm 2 Stochastic Gradient Descent Algorithm

```
initialize  $\theta, \eta$ 
repeat
    Randomly permute data
    for  $i = 1 : N$  do
         $\mathbf{g} = \nabla f(\theta, \mathbf{z}_i)$ 
         $\theta \leftarrow \text{proj}_{\Theta}(\theta - \eta \mathbf{g})$ 
        Update  $\eta$ 
    end for
until converged
```



How to Train Neural Networks

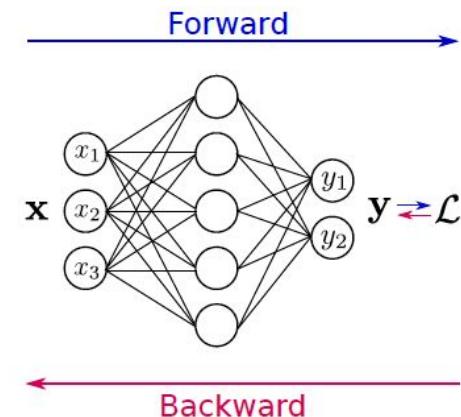
Two information flow directions:

Forward propagation

- NN accepts an input \mathbf{x} and produces an output \mathbf{y}
- During training, it continues onward until it produces a scalar cost \mathcal{L}

Back-propagation

- Information (**gradients** with respect to the parameters) from the cost flows backward through the network



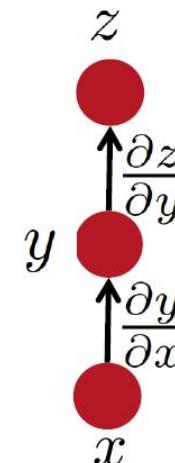


Chain Rule of Derivatives

Compute gradient of example-wise loss wrt parameters

- Simply apply the chain rule of derivative

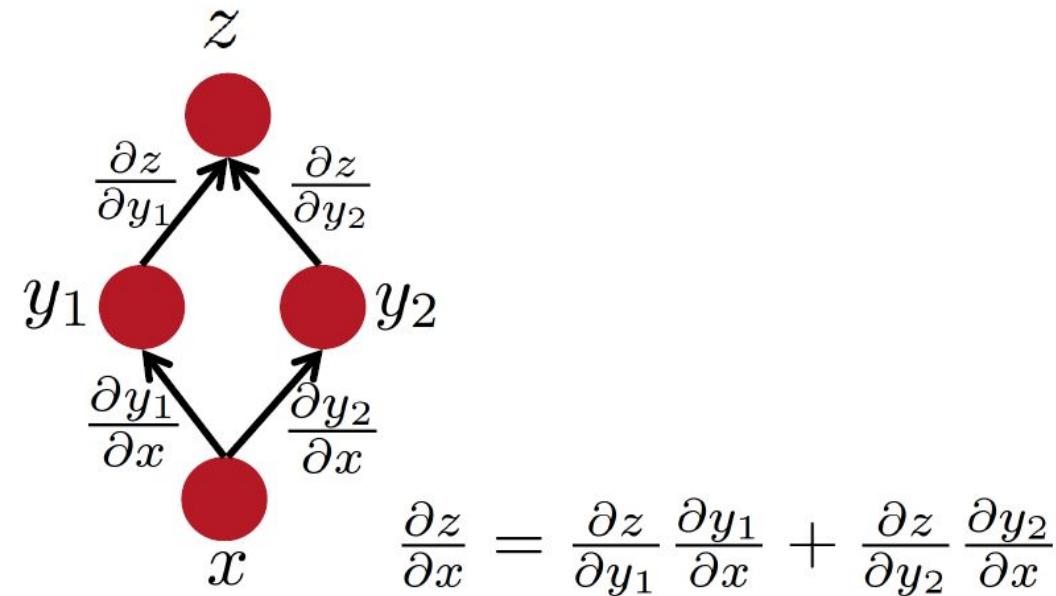
$$z = f(y) \quad y = g(x) \quad \frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$$





Chain Rule of Derivatives

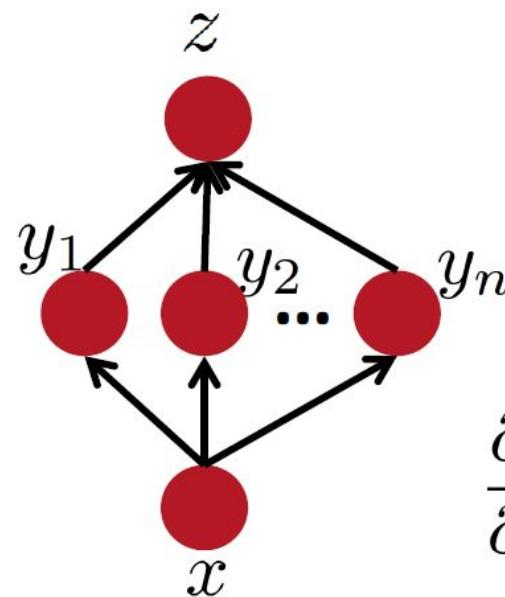
- Multiple-path chain rule





Chain Rule of Derivatives

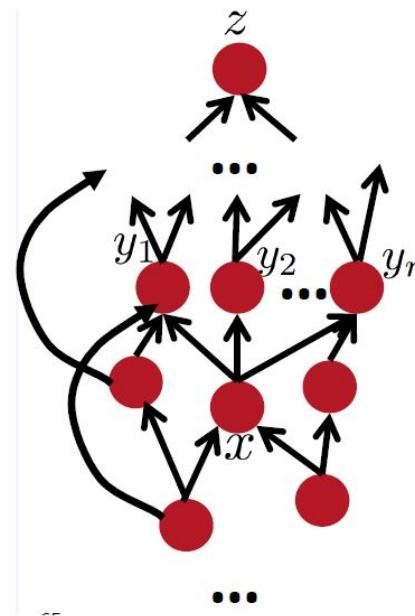
- Multiple-path chain rule (general)



$$\frac{\partial z}{\partial x} = \sum_{i=1}^n \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

Chain Rule of Derivatives

- Chain rule in computational graph



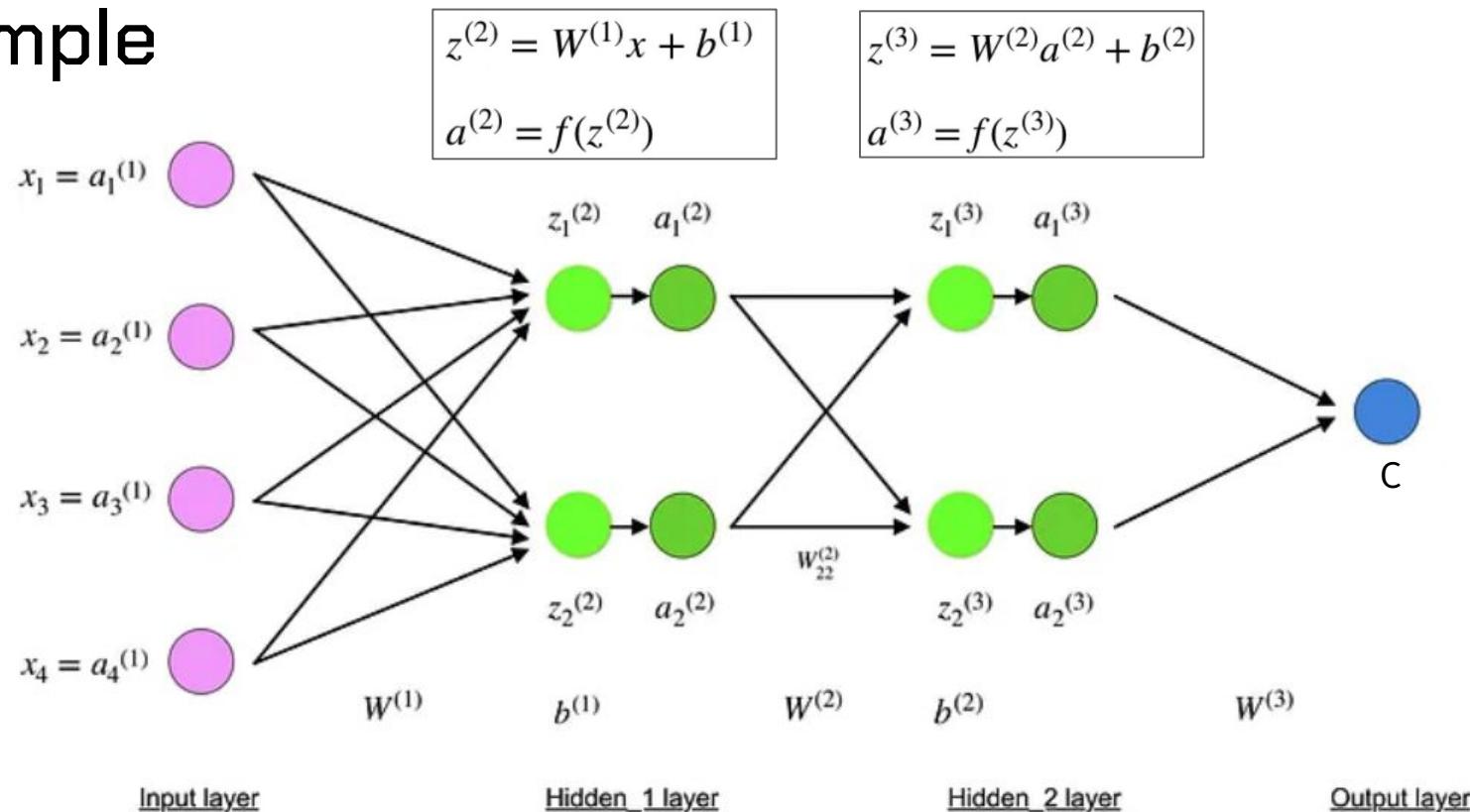
Flow graph: any directed acyclic graph
node = computation result
arc = computation dependency

$\{y_1, y_2, \dots, y_n\}$ = successors of x

$$\frac{\partial z}{\partial x} = \sum_{i=1}^n \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$



Example



Simple 4-layer neural network illustration



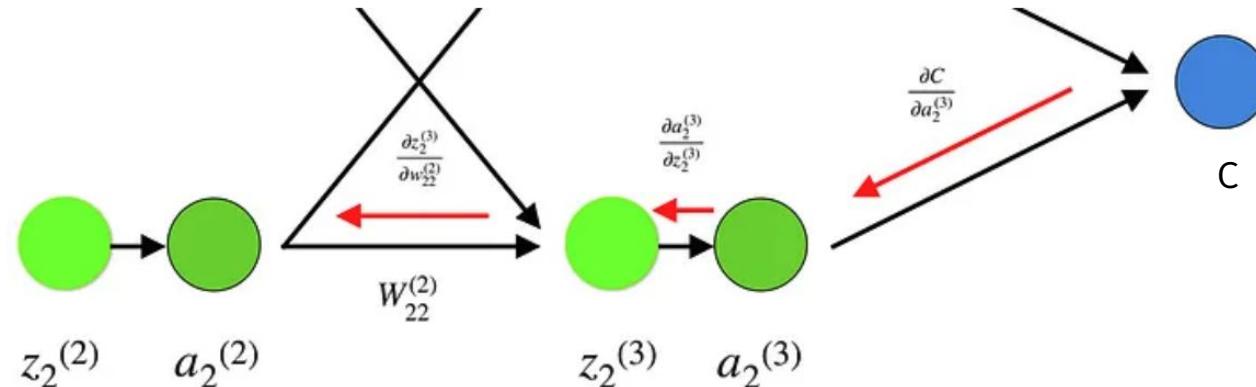
Example

$$z^{(2)} = W^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

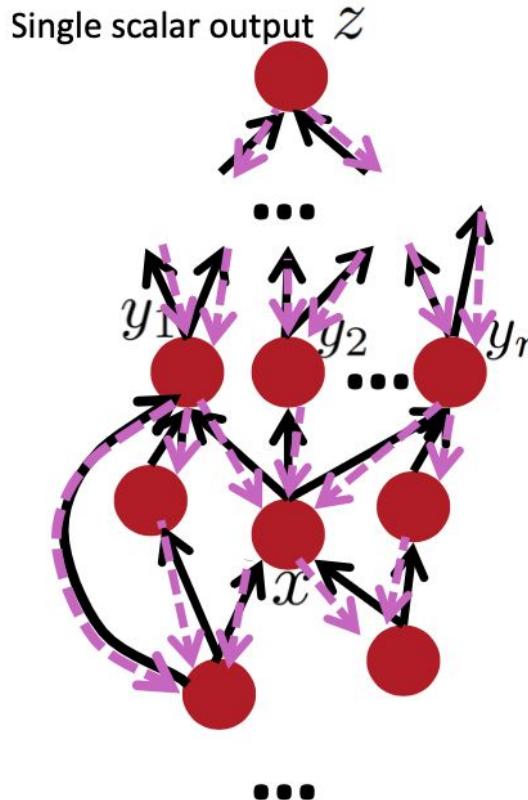
$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$

$$a^{(3)} = f(z^{(3)})$$



$$\frac{\partial C}{\partial w_{22}^{(2)}} = \frac{\partial C}{\partial z_2^{(3)}} \cdot \frac{\partial z_2^{(3)}}{\partial w_{22}^{(2)}} = \frac{\partial C}{\partial a_2^{(3)}} \cdot \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \cdot \frac{\partial z_2^{(3)}}{\partial w_{22}^{(2)}}$$

Backpropagation in General



1. Fprop: visit nodes in topo-sort order
 - Compute value of node given predecessors
2. Bprop:
 - initialize output gradient = 1
 - visit nodes in reverse order:
Compute gradient wrt each node using gradient wrt successors

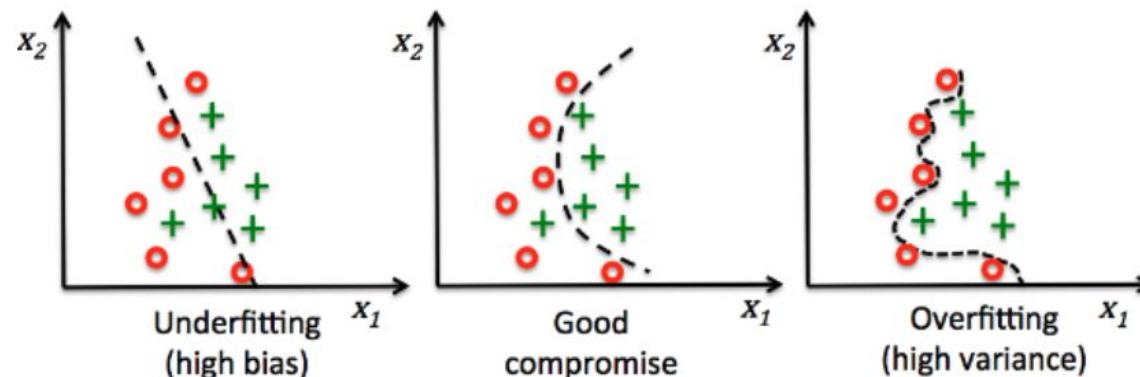
$\{y_1, y_2, \dots, y_n\}$ = successors of x

$$\frac{\partial z}{\partial x} = \sum_{i=1}^n \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$



Regularization

DNNs tend to overfit to the training data → poor generalisation performance.



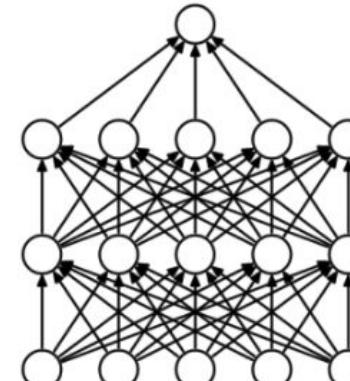
Generalization error increases due to overfitting.

We need to regularise the network

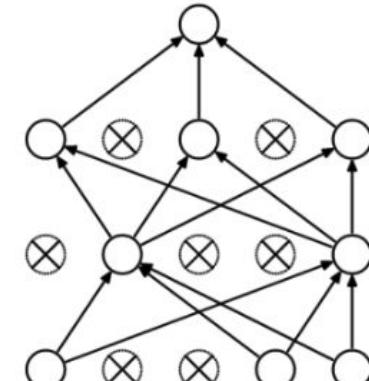
Dropout

note this is done
during training only,
inference uses all!!!

- Simple, but powerful regulariser
- Randomly turnoff some nodes during training.
- Use all activations during inference



(a) Standard Neural Net



(b) After applying dropout.

force model to not depend on every neuron rather on the general pattern

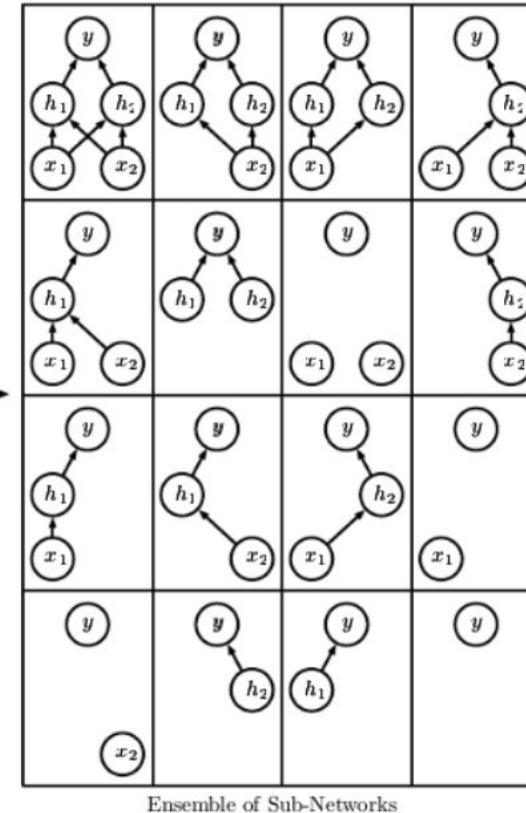
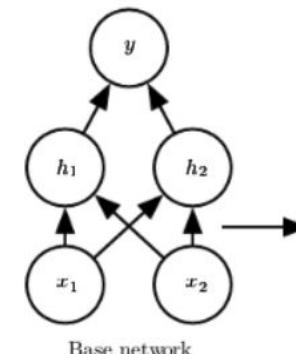
- Dropout has the effect of making the training noisy, forcing nodes within a layer to probabilistically take on more or less responsibility for the inputs.
- Dropout encourages the network to learn a sparse representation
- It breaks-up situations where network layers co-adapt to correct mistakes from prior layers, in turn making the model more robust



Dropout

- Has ensemble effect

what does it mean by ensemble effect ?





L1/2 Regularization

If a particular feature f_j is usually positive, then it always improves the loss to increase θ_j .

Regularization: discourage every θ_j from getting too large in magnitude.

[get formula of L1 and L2 norm](#)

$$\arg \min_{\boldsymbol{\theta}} \text{loss}(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_p^p$$

where $\lambda > 0$ is a “hyperparameter” and $p = 2$ or 1 .



L1/2 Regularization

$$\min_{\mathbf{w}} \text{loss}(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_1$$

Note that:

$$\|\boldsymbol{\theta}\|_1 = \sum_{j=1}^d |\theta_j|$$

- ▶ This results in **sparsity** (i.e., many $\theta_j = 0$).



Gradient Clipping

- If the gradient becomes too big, then the SGD update step becomes too big:

$$\theta^{new} = \theta^{old} - \underbrace{\alpha \nabla_{\theta} J(\theta)}_{\text{gradient}}^{\text{learning rate}}$$

- This can cause **bad updates**: we take too large a step and reach a bad parameter configuration (with large loss)
- In the worst case, this will result in **Inf** or **NaN** in your network (then you have to restart training from an earlier checkpoint)



Gradient Clipping

Gradient clipping: if the norm of the gradient is greater than some threshold, scale it down before applying SGD update

Algorithm 1 Pseudo-code for norm clipping

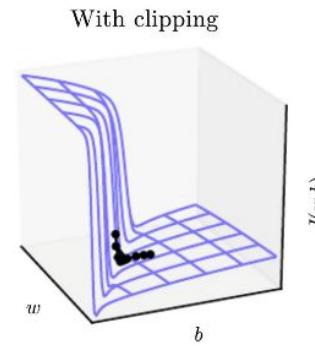
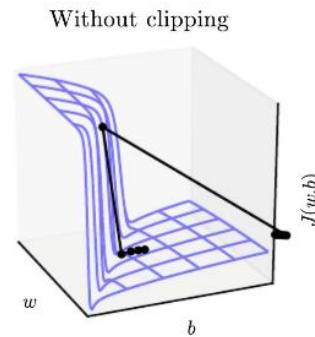
```
hat{g} ← ∂E/∂θ
if ||hat{g}|| ≥ threshold then
    g ← threshold / ||hat{g}|| hat{g}
end if
```

- Intuition: take a step in the **same direction**, but a **smaller step**



Gradient Clipping

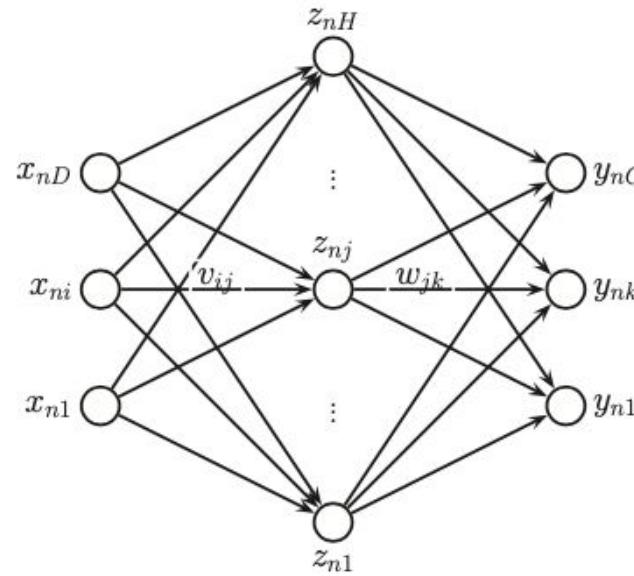
- This shows the loss surface of a simple RNN
(hidden state is a scalar not a vector)
- The “cliff” is dangerous because it has steep gradient
- In the Fig. at the top, gradient descent takes two very big steps due to steep gradient, resulting in climbing the cliff then shooting off to the right (both bad updates)
- At the bottom, gradient clipping reduces the size of those steps, so effect is less drastic





Summary

MLP



BackProp

