Let’s **FLIP!**

1. Linear & Logistic Regressions

Source Material: Faizan Ahmed (full material available on request)

Linear Regression

A **linear regression model** predicts a target variable as a **weighted sum of input features**. It assumes a linear relationship between inputs and output.

- $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$

- Minimize the **squared differences** between actual and predicted values.
- $\hat{\beta} = \arg\min_{\beta_0,\cdots,\beta_p} \sum_{i=1}^{n} \left(y^{(i)} - \left(\beta_0 + \sum_{j=1}^{p} \beta_j x_j^{(i)} \right) \right)^2$
- **Assumption:** Linearity, normality, Homoscedasticity (constant variance), Independence, Fixed features, Absence of multicollinearity

Logistic Regression

- Linear Regression Models do not work well for Classification
- Probability=reflecting the confidence of the output and classification.

$$P(Y=1) = \frac{1}{\exp\left(-(\beta_0 + \beta_1 x_1 \; + \cdots + \beta_p x_p \;)\right)}$$

$$\left(\frac{P(Y=1)}{P(Y=0)}\right) = odds = exp\left(\beta_0 + \beta_1 x_1 \; + \cdots + \beta_p x_p \;\right)$$

How the prediction changes when one of the features x_k is changed by 1 unit.

$$\frac{odds(x_k+1)}{odds(x_k)} = \frac{exp\left(\beta_0 + \beta_1 x_1 \; + \cdots + \beta_k\,(x_k+1) + \cdots + \beta_p x_p \;\right)}{exp\left(\beta_0 + \beta_1 x_1 \; + \cdots + \beta_k\,(x_k) + \cdots + \beta_p x_p \;\right)} = \exp(\beta_k)$$

2. Decision Tree

Source Material: Chen Huang, School of Business and Social Sciences Aarhus University

Decision Tree

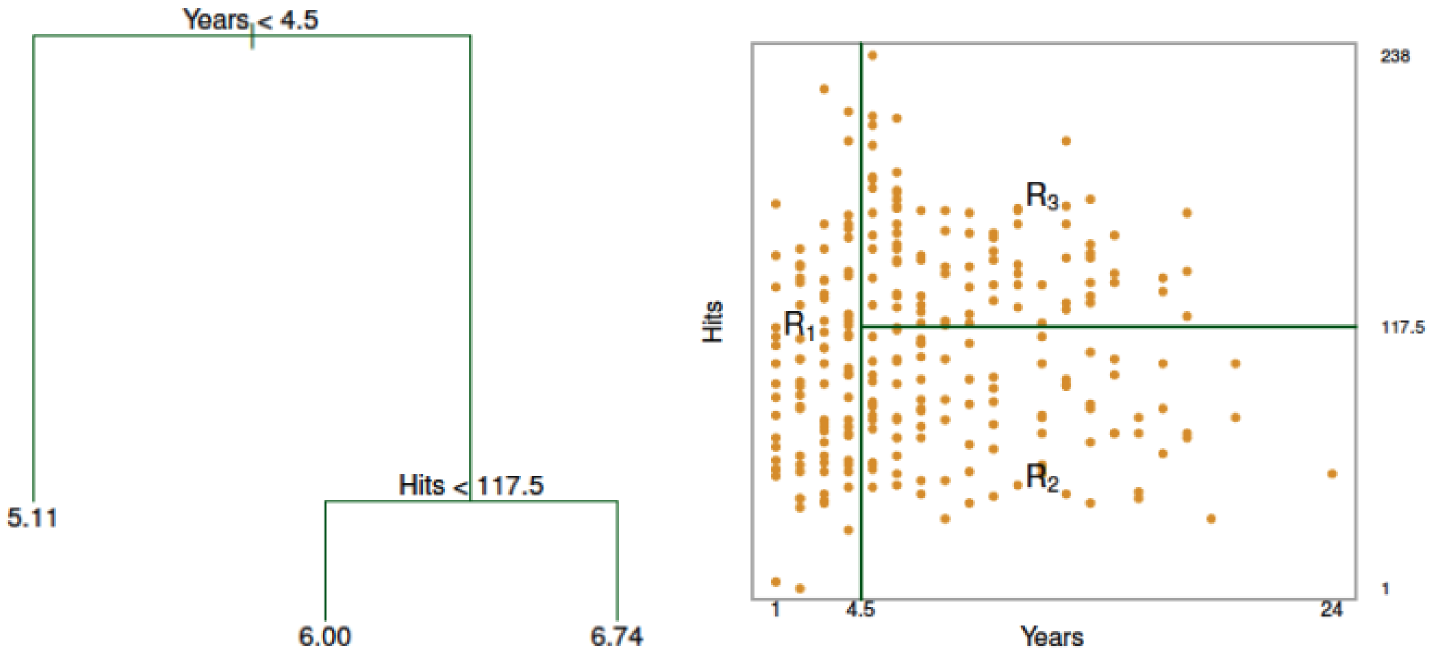


Figure 1: Left: A regression tree for predicting the log salary of a baseball player. Right: The three-region partition from the regression tree. (Figure 4.6 and 4.7 of ISLR)

Basics of Decision Tress

- Basic idea: use a set of splitting rules to **segment** the feature space.
- The decision trees are typically drawn upside down.
  - **Terminal nodes or leaves**: the regions R_1, R_2, and R_3.
  - **Internal nodes**: the points where the feature space is split: `Years<4.5` and `Hits<117.5`.
  - **Branches**: the segment of the trees that connect the nodes.
- The number in each leaf is the average of the response for the observations falling there.

Building a Regression Tree

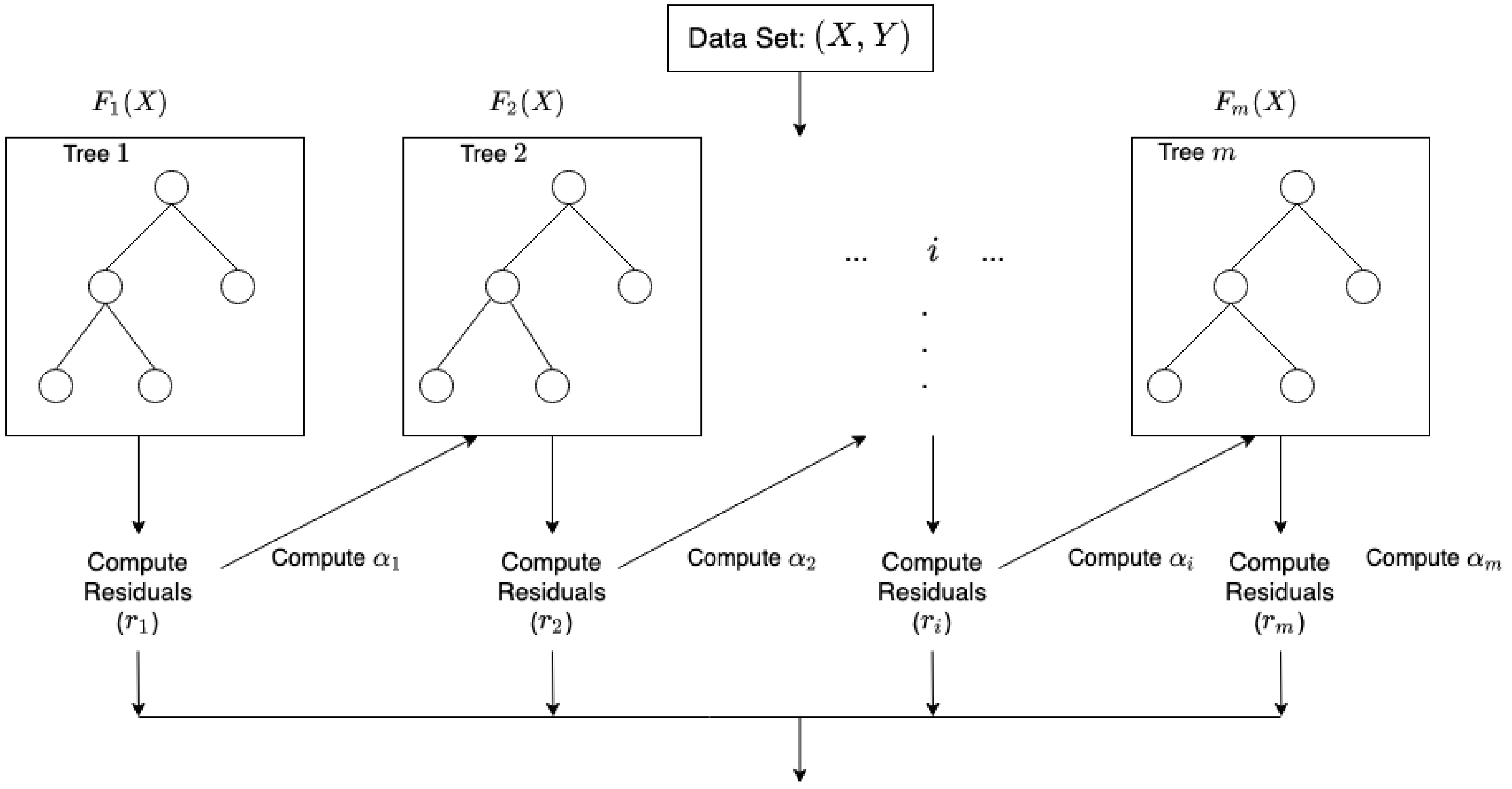
- Goal: use the data to form a decision tree using recursive binary partitions.
- Step 1: let $R_1(j,s) = \{\boldsymbol{X} : X_j < s\}$ and $R_2(j,s) = \{\boldsymbol{X} : X_j \geq s\}$, choose j and s that minimize

  $$\sum_{i:\boldsymbol{x}_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 \quad + \sum_{i:\boldsymbol{x}_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2,$$

  where $\hat{y}_{R_k}$ is the mean response for the training data in $R_m(j,s)$, $m = 1, 2$.
- Step 2: repeat the splitting process on the subregion that RSS reduces the most.
- ...
- Stop splitting when some stopping criterion reached, e.g. a minimum terminal node size.

3. Gradient Boosting

Source Material: How XGBoost works. [Online]. Available: https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost-HowItWorks.html



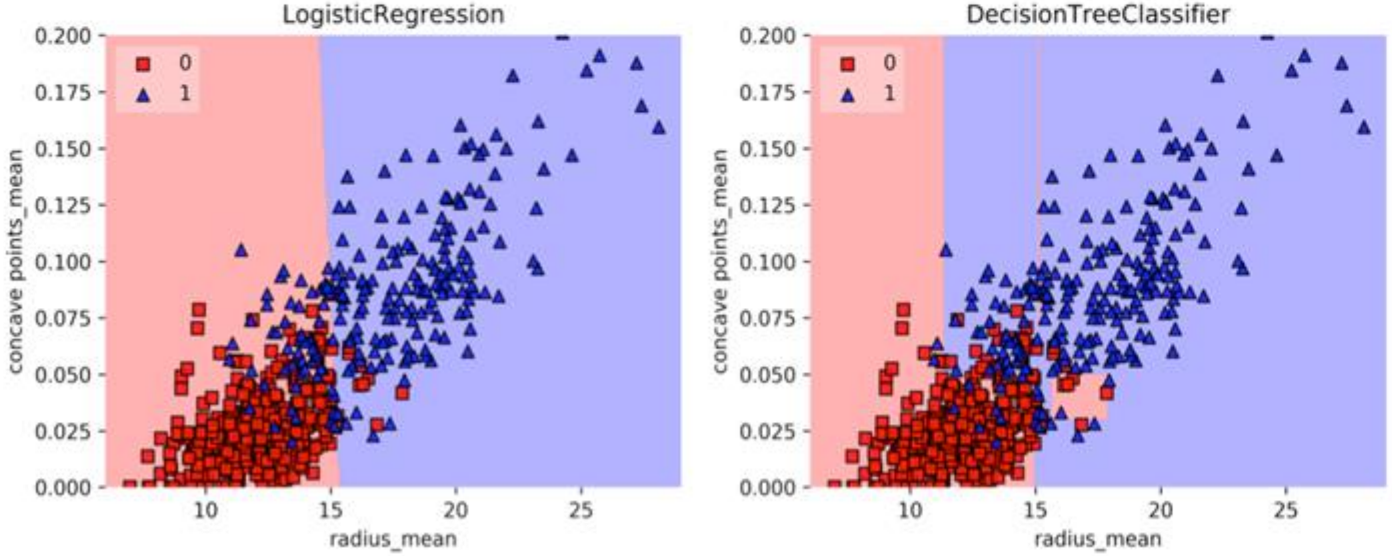
$$F_m(X) \ = \ F_{m-1}(X) + \alpha_m h_m(X, r_{m-1}),$$

where α_i, and r_i are the regularization parameters and residuals computed with the i^{th} tree respectfully, and h_i is a function that is trained to predict residuals, r_i using X for the i^{th} tree. To compute α_i we use the residuals computed, r_i and compute the following: $arg \ \min\limits_{\alpha} \ = \sum\limits_{i=1}^{m} L(Y_i, F_{i-1}(X_i) + \alpha h_i(X_i, r_{i-1}))$ where $L(Y, F(X))$ is a differentiable loss function.

4. Evolution of NN

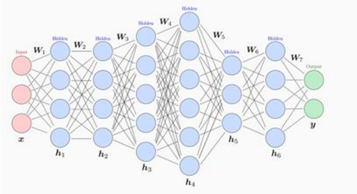
Source Material: Branka Hadji Misheva, training material (full material available on request)

Rule-based systems;
logistic regressions,
decision trees



Pre-deep learning

Limitations: Couldn’t handle complex patterns or large unstructured data (like images or text)

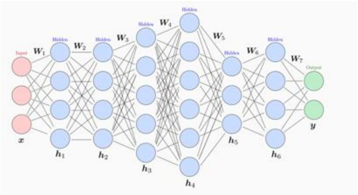


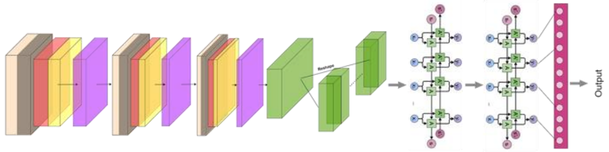
Introduction of backpropagation (Rumelhart et al., 1986) allowed training of **multi-layer neural networks**

No memory of previous inputs

Multi-layer perception

Deep learning





Introduction of backpropagation (Rumelhart et al., 1986) allowed training of **multi-layer neural networks**

Use of **convolutional layers** with local receptive fields. Great for images and structured spatial data

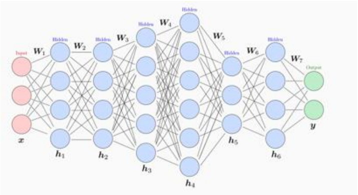
No memory of previous inputs

No memory of previous inputs

Multi-layer perception

Convolutional NN

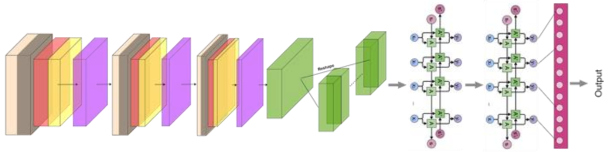
Deep learning



Introduction of backpropagation (Rumelhart et al., 1986) allowed training of **multi-layer neural networks**

No memory of previous inputs

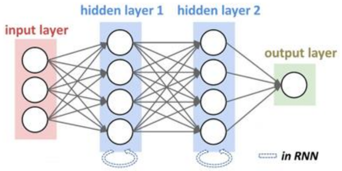
Multi-layer perception



Use of **convolutional layers** with local receptive fields. Great for images and structured spatial data

No memory of previous inputs

Convolutional NN



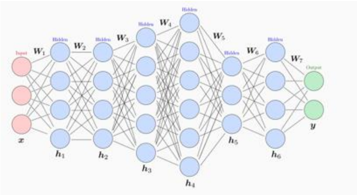
Use of **recurrent layers** that process data step-by-step. Great for sequential data (time series, text speech)

Maintains memory of previous inputs

Recurrent NN

Deep learning

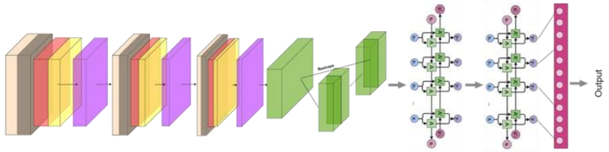
Limitation: Hard to train on long sequences; Sequential processing is slow (can't parallelize easily).



Introduction of backpropagation (Rumelhart et al., 1986) allowed training of **multi-layer neural networks**

No memory of previous inputs

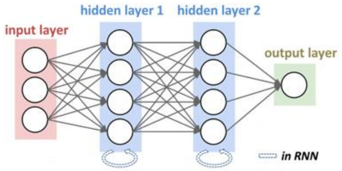
Multi-layer perception



Use of **convolutional layers** with local receptive fields. Great for images and structured spatial data

No memory of previous inputs

Convolutional NN



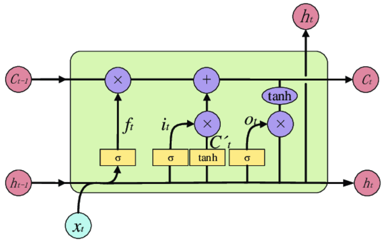
Use of **recurrent layers** that process data step-by-step. Great for sequential data (time series, text speech)

Maintains memory of previous inputs

Recurrent NN

Deep learning

Limitation: Hard to train on long sequences; Sequential processing is slow (can't parallelize easily).



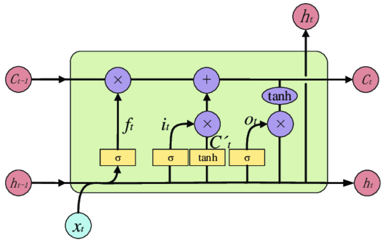
Long Short-Term Memory networks (**LSTMs**) are a special type of RNN designed to remember long-term dependencies without suffering from vanishing gradients.

More sophisticated memory system using gates. These gates control what information should be remembered or forgotten.

Long Short-Term Memory

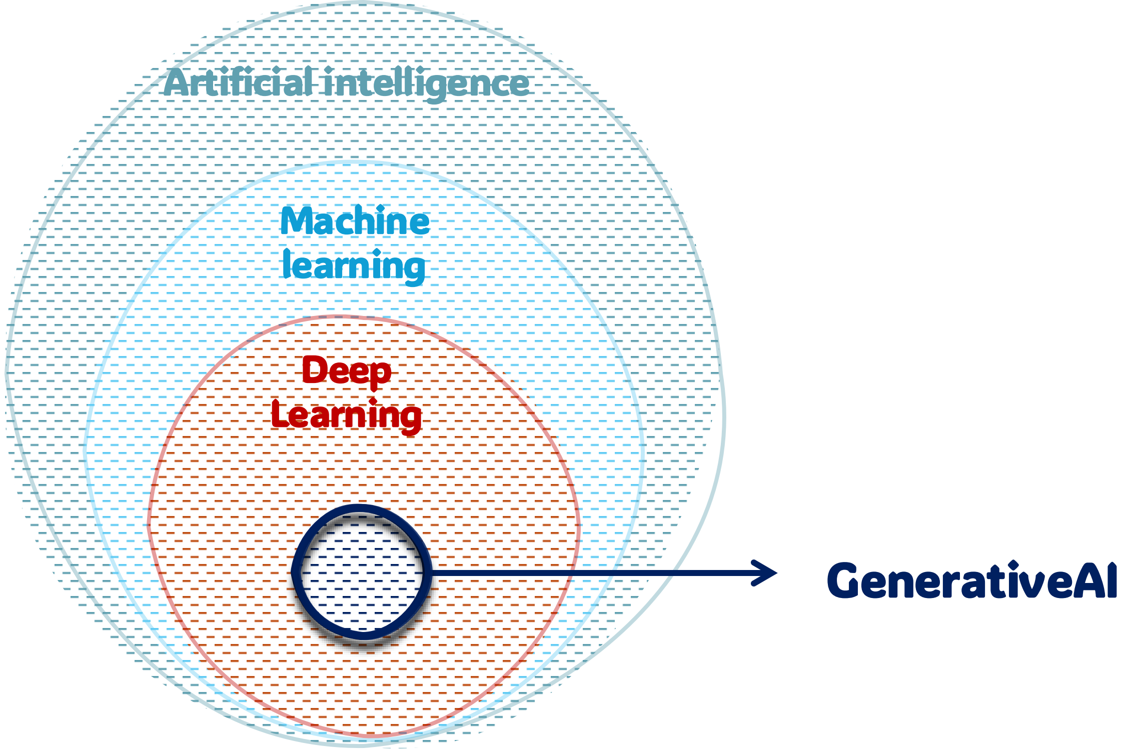
Deep learning

Limitations: still processes sequentially, which limits speed and scalability.



Long Short-Term Memory networks (**LSTMs**) are a special type of RNN designed to remember long-term dependencies without suffering from vanishing gradients.

More sophisticated memory system using gates. These gates control what information should be remembered or forgotten.



Long Short-Term Memory

Transformers

Limitations: still processes sequentially, which limits speed and scalability.

Generative Models

Self-attention

Vaswani et al. in the paper "Attention Is All You Need", fundamentally changed natural language processing (NLP) by replacing traditional RNNs and LSTMs with a self-attention mechanism that **allows parallelization and better handling of long-range dependencies**.

Google

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

1 Introduction

Recurrent neural networks, long short-term memory [13] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and

*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

†Work performed while at Google Brain.

‡Work performed while at Google Research.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

RNNs process words sequentially, meaning each word only has context from past words. **Self-Attention allows all words to interact simultaneously.**

Computes attention scores to weigh different words in a sentence based on their importance to each other.

Transformers

Transformer models

5. NN: Basics

Source Material: Wouter Van Heeswijk, University of Twente

Neural Network Basics (I)

Feedforward Networks

- Typically composed of multiple layers: input, hidden, and output.
- Common activation functions include Linear, ReLU, Sigmoid, and Tanh.

Forward Pass

$$\mathbf{h}^{(1)} = \sigma\big(W^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\big),$$

$$\mathbf{h}^{(2)} = \sigma\big(W^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}\big), \ldots$$

$$\mathbf{y} = W^{(L)}\mathbf{h}^{(L-1)} + \mathbf{b}^{(L)}.$$

Parameter Space

- Weights $\{W^{(l)}\}$ and biases $\{b^{(l)}\}$ define each layer.
- Typically optimized via gradient-based methods (e.g., SGD, Adam).

Neural Network Basics (II)

Backpropagation

Definition: Algorithm applying chain rule to compute partial derivatives $\frac{\partial \mathcal{L}}{\partial W^{(l)}}$ and $\frac{\partial \mathcal{L}}{\partial b^{(l)}}$, where $\mathcal{L}$ is a loss function (e.g., MSE or cross-entropy).

Gradient-Based Updates

$$W \leftarrow W - \eta \, \frac{\partial \mathcal{L}}{\partial W}, \quad b \leftarrow b - \eta \, \frac{\partial \mathcal{L}}{\partial b},$$

where η is the learning rate.

Common Optimizers

- SGD, Momentum-based methods
- Adam, RMSProp (adaptive learning rates)

Practical Note

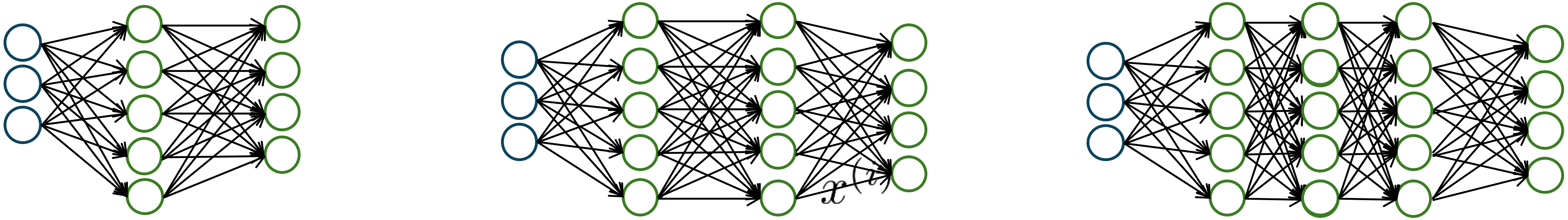
Large networks can be prone to vanishing or exploding gradients. Careful initialization (e.g. Xavier or Kaiming (He)) and normalization (e.g. BatchNorm) are widely used to address these issues.

6. Training a NN

Source Material: Machine Learning By Prof. Andrew Ng

Training a neural network

Pick a network architecture (connectivity pattern between neurons)



No. of input units: Dimension of features

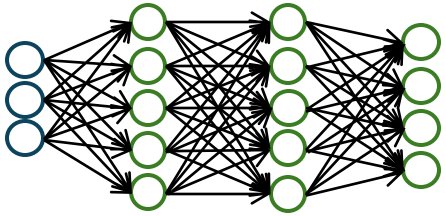
No. output units: Number of classes

Reasonable default: 1 hidden layer, or if >1 hidden layer, have same no. of hidden units in every layer (usually the more the better)

Training a neural network

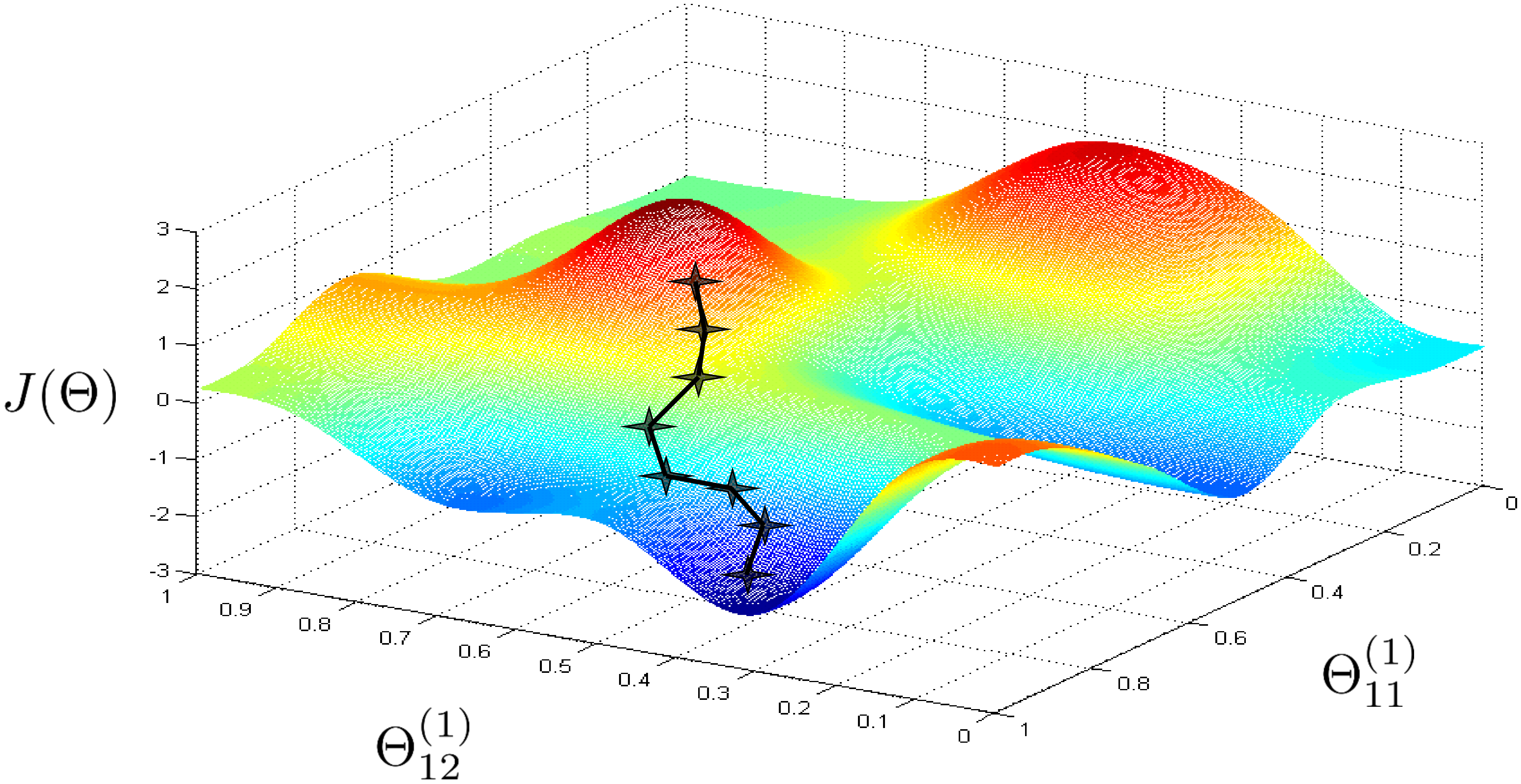
1. Randomly initialize weights
2. Implement forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$
3. Implement code to compute cost function $J(\Theta)$
4. Implement backprop to compute partial derivatives $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

```
for i = 1:m
    Perform forward propagation and backpropagation using example $(x^{(i)}, y^{(i)})$
    (Get activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l = 2, \ldots, L$).
```



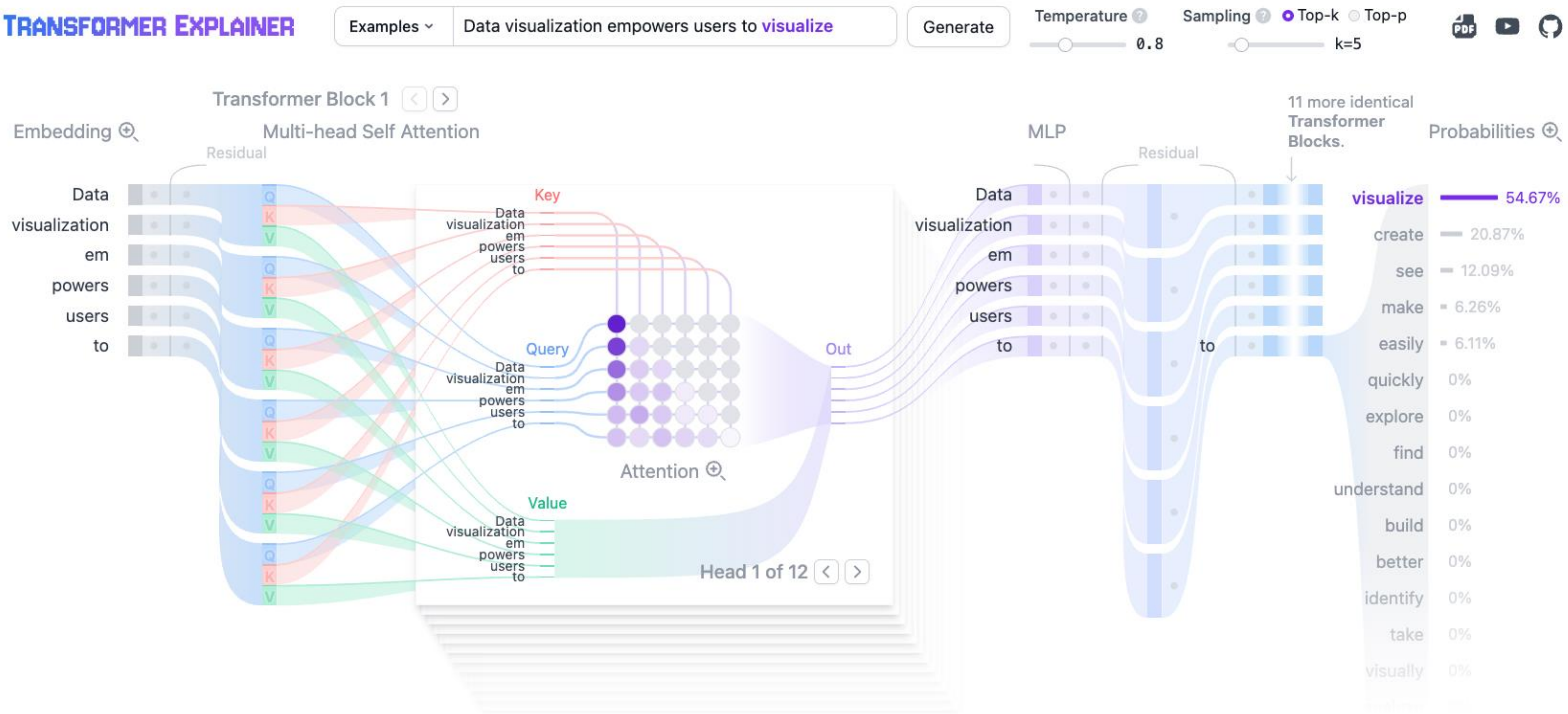
Training a neural network

5. Use gradient checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ computed using backpropagation vs. using numerical estimate of gradient of $J(\Theta)$.
   Then disable gradient checking code.
6. Use gradient descent or advanced optimization method with backpropagation to try to minimize $J(\Theta)$ as a function of parameters Θ


$J(\Theta)$
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
$\Theta_{12}^{(1)}$
0
0.2
0.4
0.6
0.8
1
$\Theta_{11}^{(1)}$

7. Transformer

Source Material: Poloclub GitHub - Tranformer-explainer



Examples ⌄ Data visualization empowers users to visualize

Generate

Temperature 0.8

Sampling ● Top-k ○ Top-p k=5

DEMO

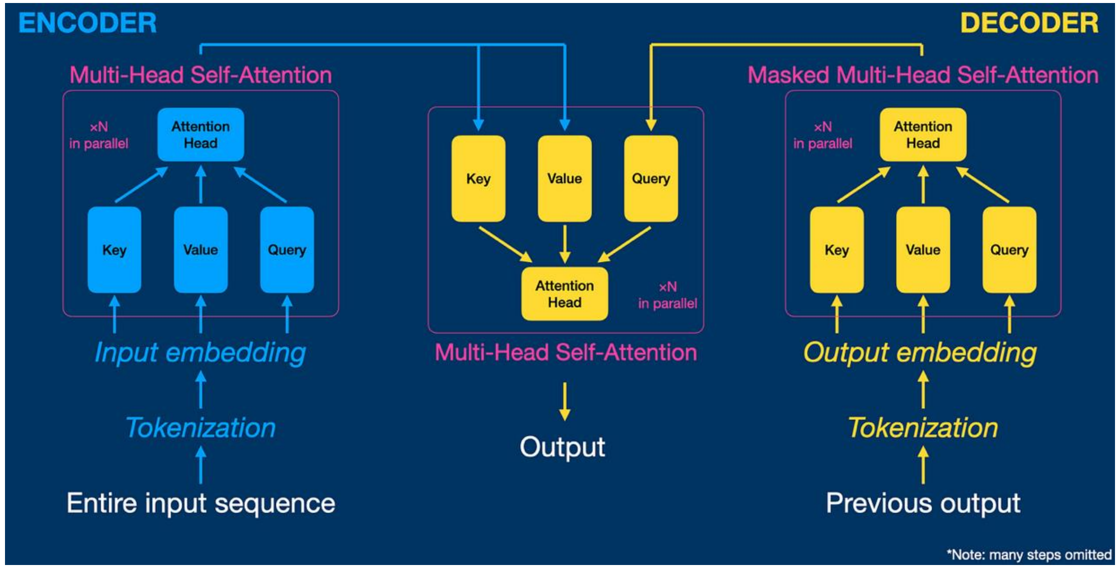
Generative Models

BERT

□ Encoder-only

Input → Tokenizer → Embedding → Encoder → Output

The encoder uses **multi-head self-attention** to understand the full input **bidirectionally** (it sees all words at once, left and right).



*Note: many steps omitted

GPT

□ Decoder-only

Previous output → Tokenization → Output Embedding → Decoder → Next Word

GPT uses masked multi-head self-attention to ensure it only sees past words

Transformer models

Goal: produce a rich understanding of the whole input like figuring out how all the words relate to each other in context.

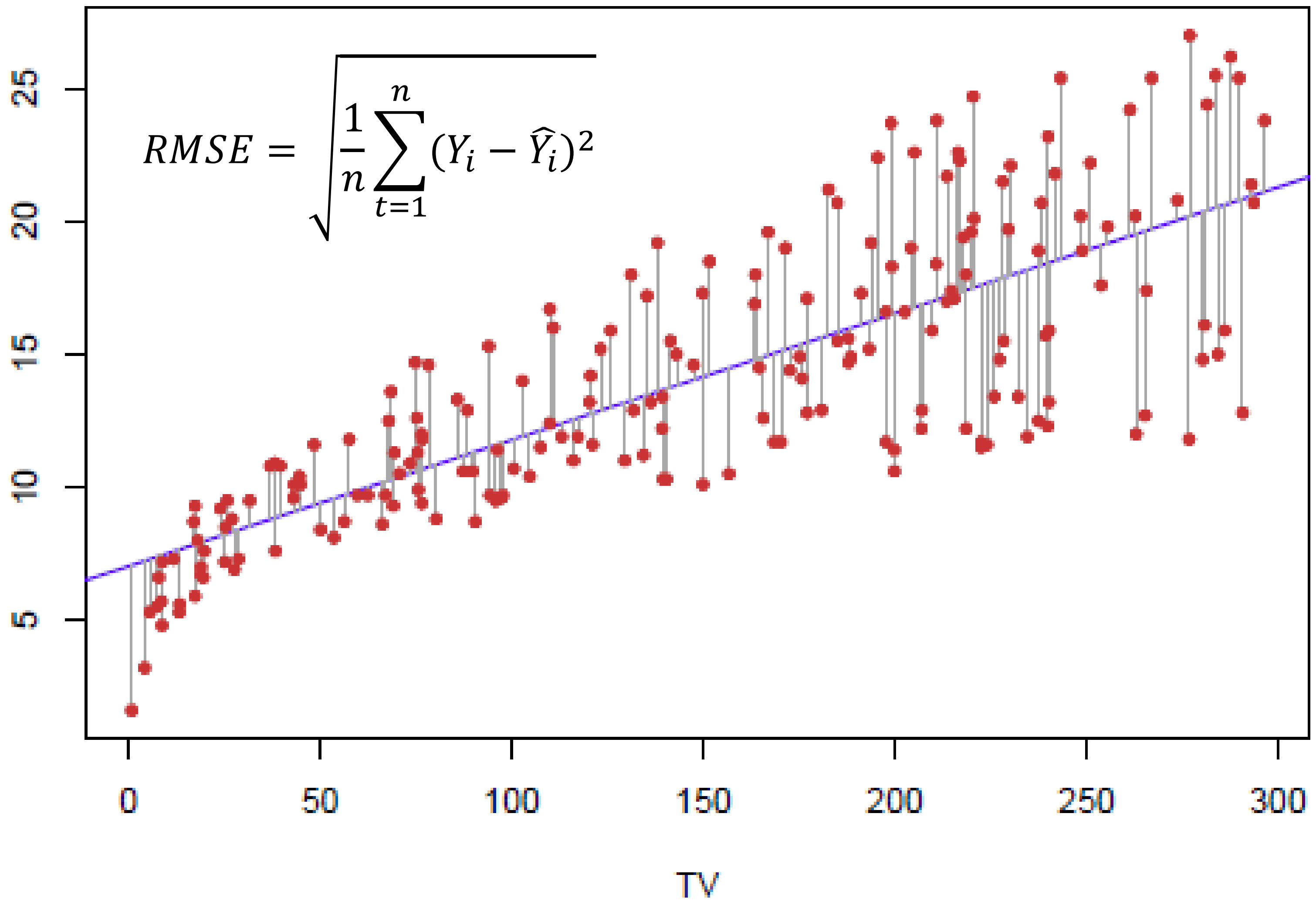
Goal: generate the next likely word, given a sequence of preceding words.

8. Model Evaluation

Source Material: Branka Hadji Misheva, training material (full material available on request)

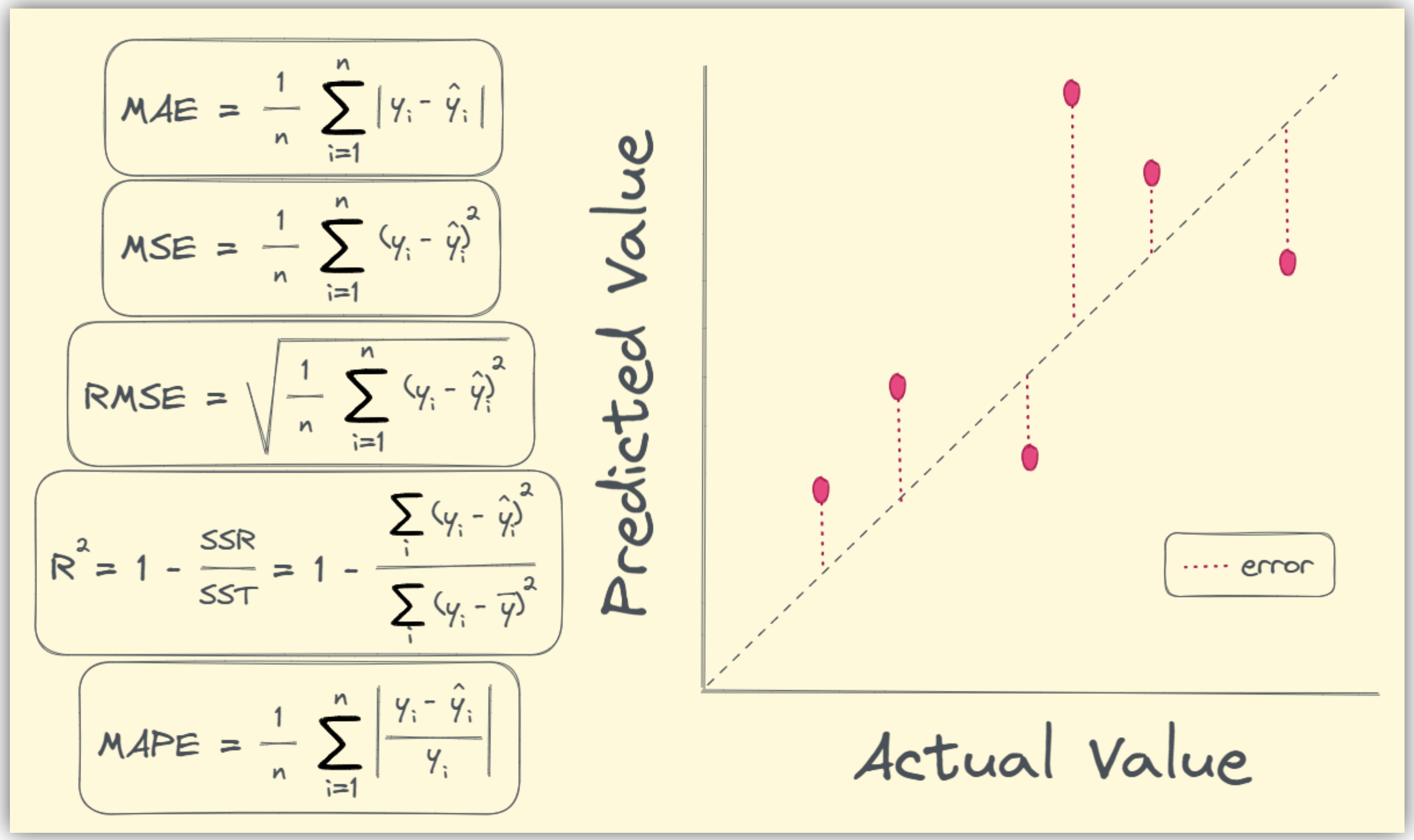
Testing preformance: Regression models

- Comparison between the **real and predicted values**



$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(Y_i - \widehat{Y}_i)^2}$$

Testing preformance: Regression models



$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

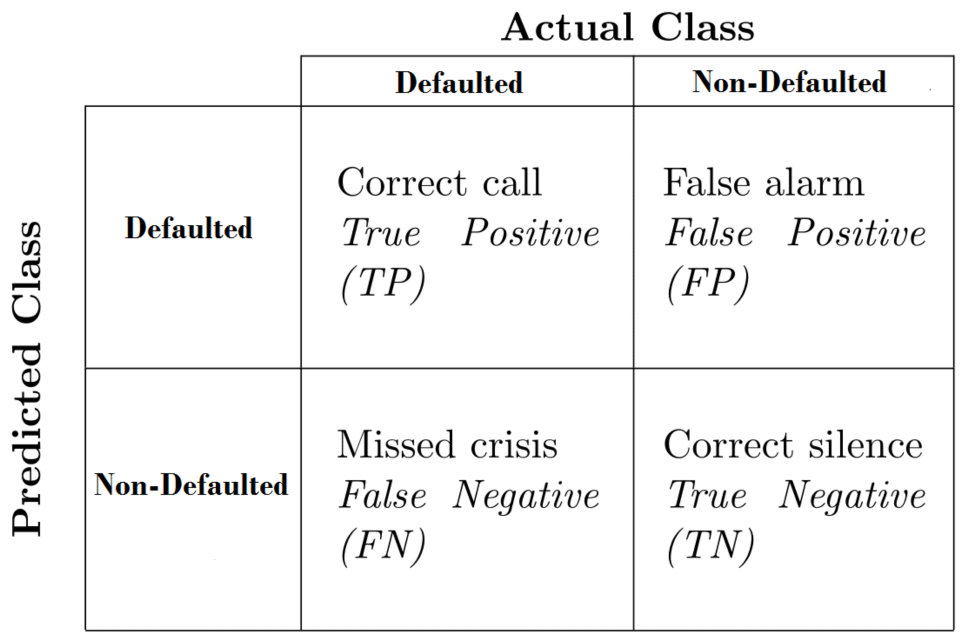
$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

$$R^2 = 1 - \frac{SSR}{SST} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

Testing preformance: **Classification models**

- Confusion matrix



		Actual Class	
		Defaulted	**Non-Defaulted**
Predicted Class	**Defaulted**	Correct call *True Positive (TP)*	False alarm *False Positive (FP)*
	Non-Defaulted	Missed crisis *False Negative (FN)*	Correct silence *True Negative (TN)*

- **Rows** – predicted class values
- **Columns** – predicted class values
- **Numbers on main diagonal** – correctly classified samples
- Numbers off the main diagonal – misclassified samples

Testing preformance: Classification models

- The **overall accuracy** of the model can be computed as:

$$\textbf{ACC} = \frac{TP+TN}{TP+TN+FP+FN}$$

where:

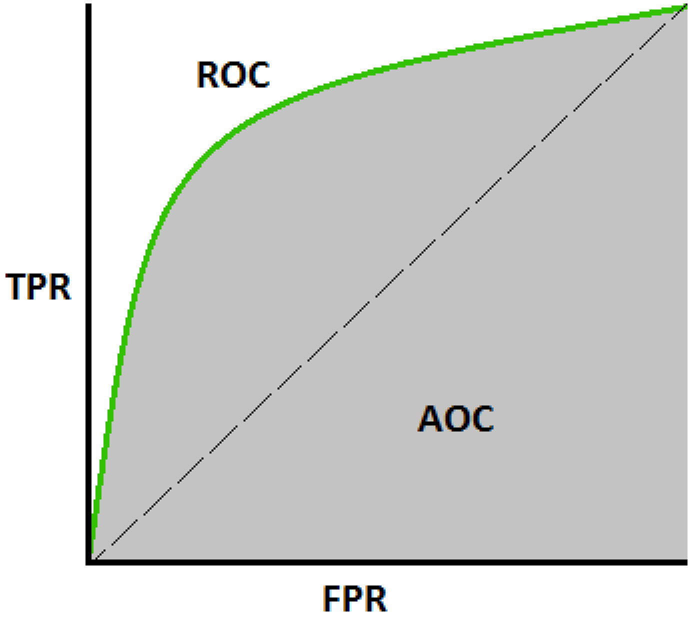
- **True Positive (TP):** Actual and predicted class is positive
- **True Negative (TN):** Actual and predicted class is negative
- **False Negative (FN):** Actual class is positive and predicted negative
- **False Positive (FP):** Actual class is negative and predicted positive

Downsides:

- Only considers the performance in general and not for the different classes
- Therefore, not informative when the class distribution is unbalanced

Testing preformance: Classification models

- The ROC Curve shows the false positive rate and true positive rate for different threshold values:
  - **False positive rate (FPR)**
    - negative events incorrectly classified as positive
  - **True positive rate (TPR)**
    - positive events correctly classified as positive
- AUC – Area under the ROC curve is a performance measurement for classification problem at various thresholds settings
  - Range 0 to 1
  - Closer to 1 it is, the better the classifier is at identifying 0s as 0s and 1s as 1s



True Positive Rate

$$\text{TPR} = \frac{TP}{TP+FN}$$

False Positive Rate

$$\text{FPR} = \frac{FP}{FP+TN}$$

Testing preformance: **Classification models**

- **The F1 Score** is the harmonic mean of precision and recall, and it provides a single metric that balances both precision & recall.

- **Precision**
  - How many predicted positives are actual positives
  - Precision = $\frac{TP}{TP+FP}$

- **Recall** (same as TPR)
  - How many actual positives are correctly predicted
  - Recall = $\frac{TP}{TP+FN}$

$$\mathbf{F1} = 2 * \frac{Precision\ * Recall}{Precision + Recall}$$

F1 captures how well the model identifies the **minority** class, without being misled by the majority class.

9. K-means

Source Material: Branka Hadji Misheva, training material (full material available on request)

K-means

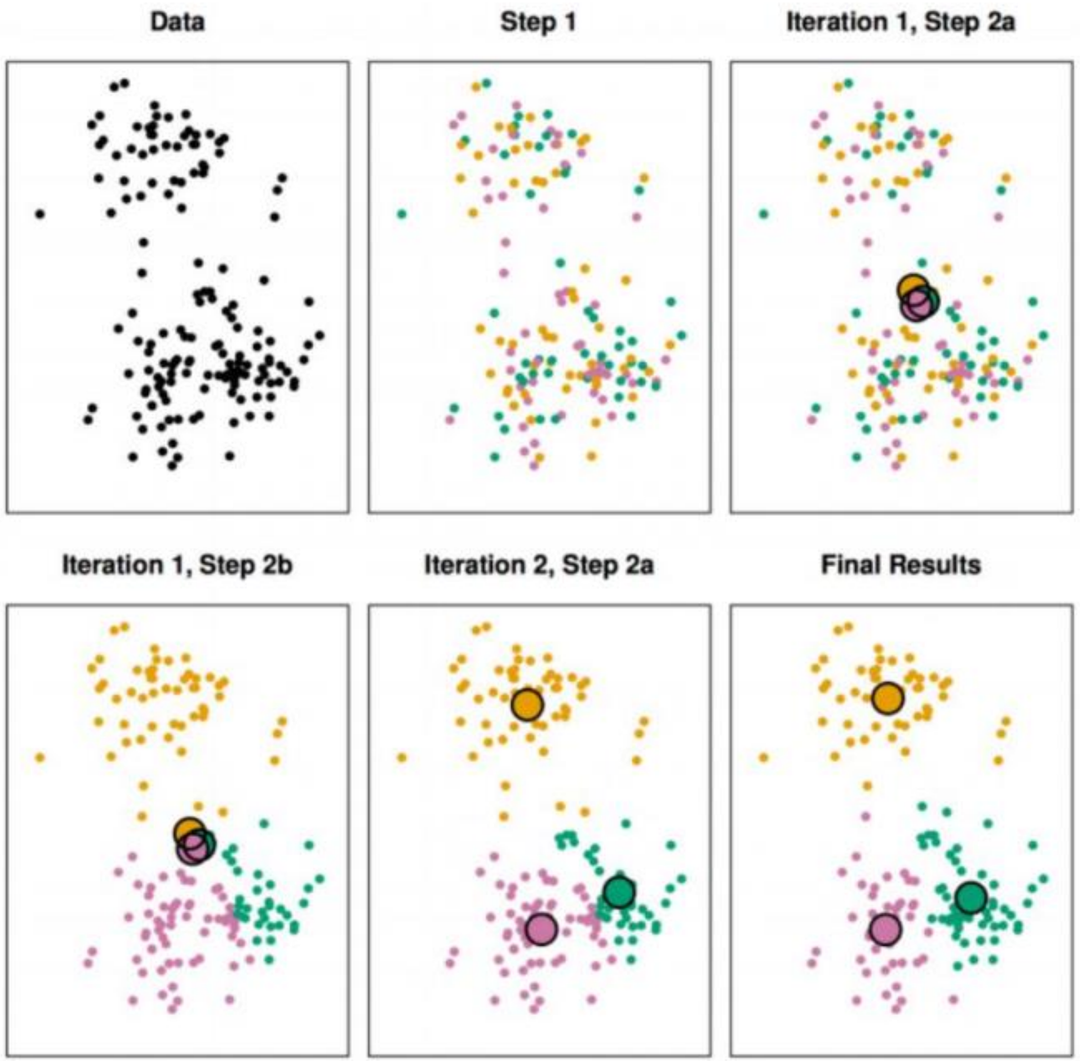
- K-means clustering – simple and very popular unsupervised machine learning algorithm

- Its basic operation is very simple:
  - ❑ Choose **number of clusters = k**
  - ❑ Randomly **choose the centroids** of each cluster
  - ❑ **Two-step:**

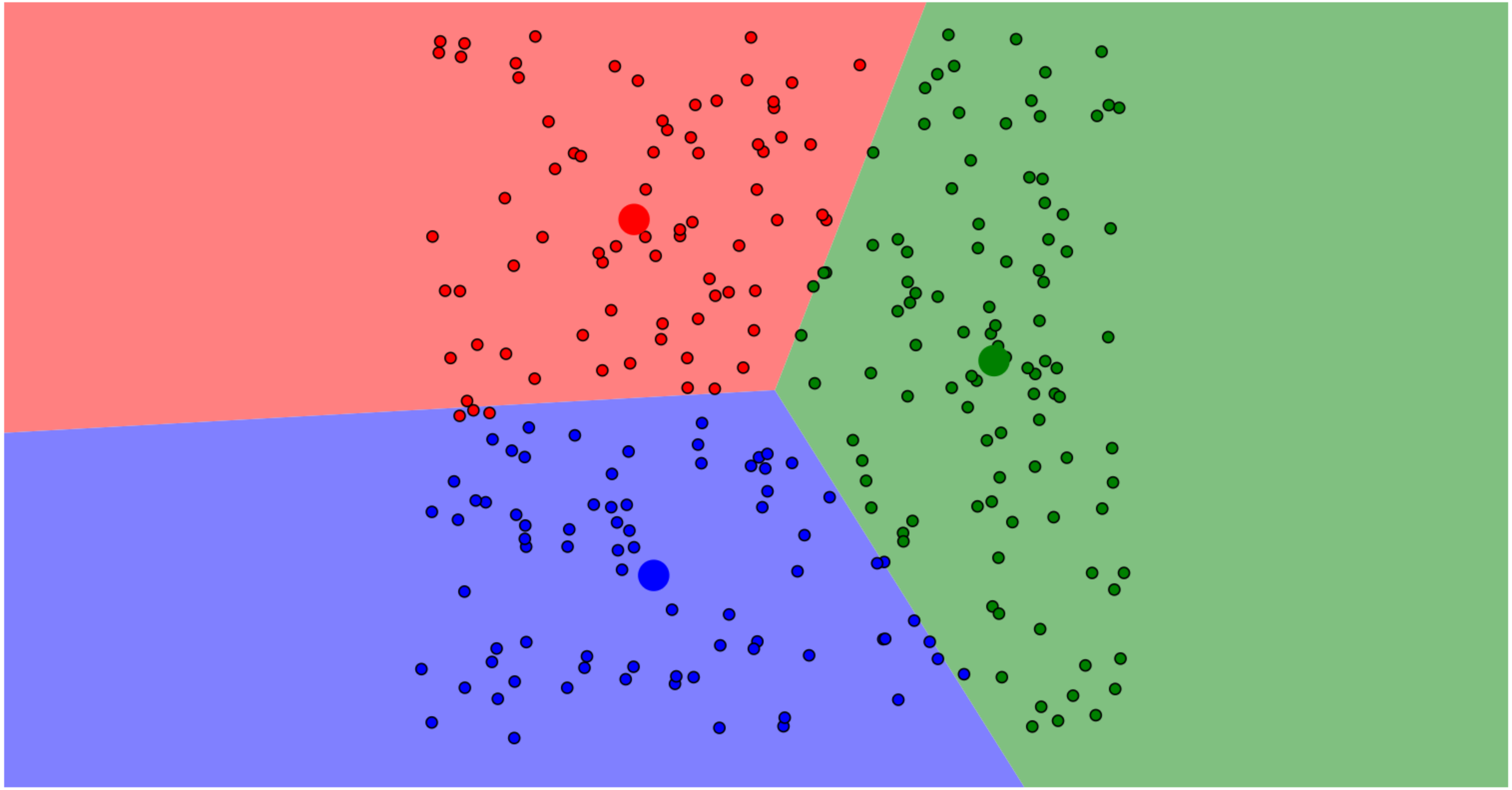
    Step 1: Assignment

    Step 2: Update

Repeat steps



Interactive Play



Link to Demo

K-means: DETAILS

- $C_1, \dots. C_k$ → sets containing the indices of the observation in each cluster. These sets satisfy two main properties:

1. $C_1 \cup C_2 \cup C_3 \dots \cup C_k = \{1, \dots., n\}$ - with the union of all clusters, you have included all observations, i.e. each observation has to be assigned to a cluster.

2. $C_1 \cap C_2 \cap C_3 \dots \cap C_k = \emptyset$ for all $k \neq k^{;}$- the intersection of the clusters is empty i.e. the clusters are not overlapping.

K-means: **DETAILS**

- What is a good clustering?
    - **Within-Cluster Variation** (WCV) is as small as possible

- The within-cluster variation for cluster C_k is a measure $WCV(C_k)$ of the amount by which the observations within a cluster differ from each other

- We are solving the following problem

$$minimize \left\{ \sum_{k=1}^{K} WCV(C_k) \right\} \qquad (1)$$

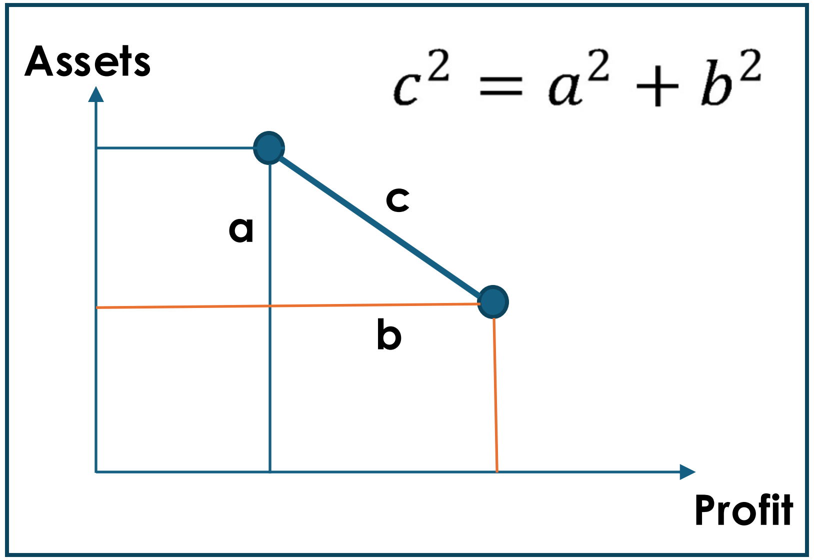
- Hence, we are looking for a partition that can minimize the WCV

K-means: **DETAILS**

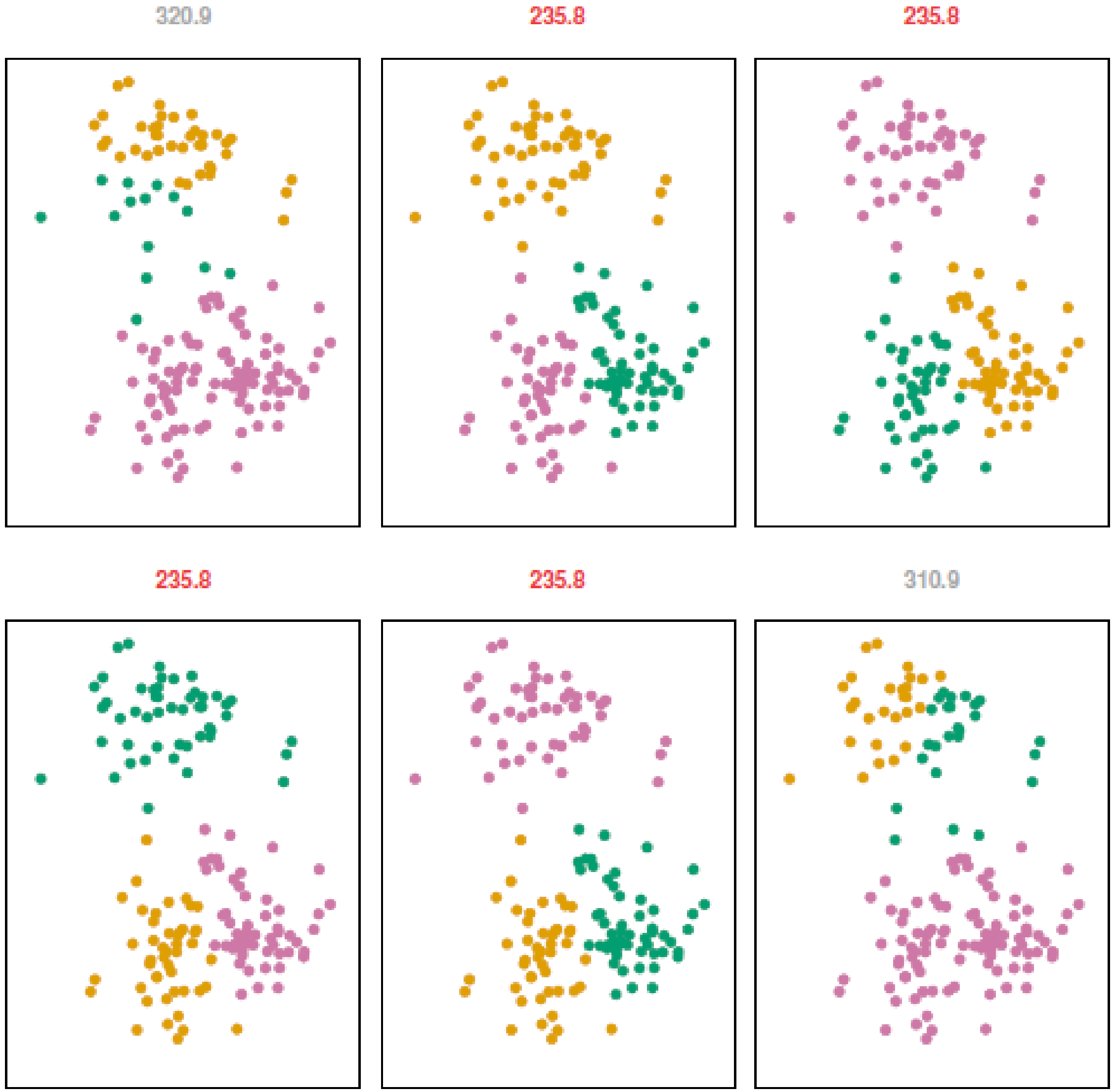
- How do we measure the **within-cluster variation**?

- One option is to use **Euclidean Distance**

- What are some of the properties?
  - Dist(x,y)=0 if x=y (the identity axiom)
  - Dist(x,y)=Dist(y,x) (the symmetry axiom).
  - (if 3 points) Dist(x,y) + Dist(y,z) ≥ Dist(x,z) (the triangle axiom)



STARTING DIFFERENT VALUES



10. Linkage algos (Clustering)

Source Material: Wolfgang Karl Härdle
Elizaveta Zinovyeva

Example The distance matrix D (L_2 distances) is

$$D = \begin{pmatrix} 0 & 10 & 53 & 73 & 50 & 98 & 41 & 65 \\ & 0 & 25 & 41 & 20 & 80 & 37 & 65 \\ & & 0 & 2 & 1 & 25 & 18 & 34 \\ & & & 05 & 17 & 20 & 32 & \\ & & & & 0 & 36 & 25 & 45 \\ & & & & & 0 & 13 & 9 \\ & & & & & & 0 & 4 \\ & & & & & & & 0 \end{pmatrix}$$

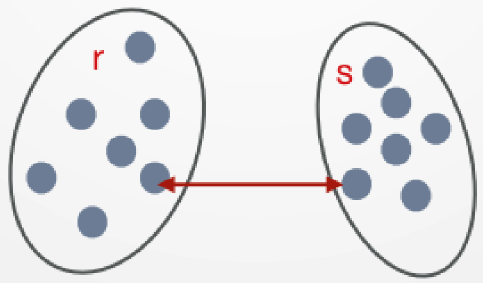
The 8 points example

Single Linkage Algorithm

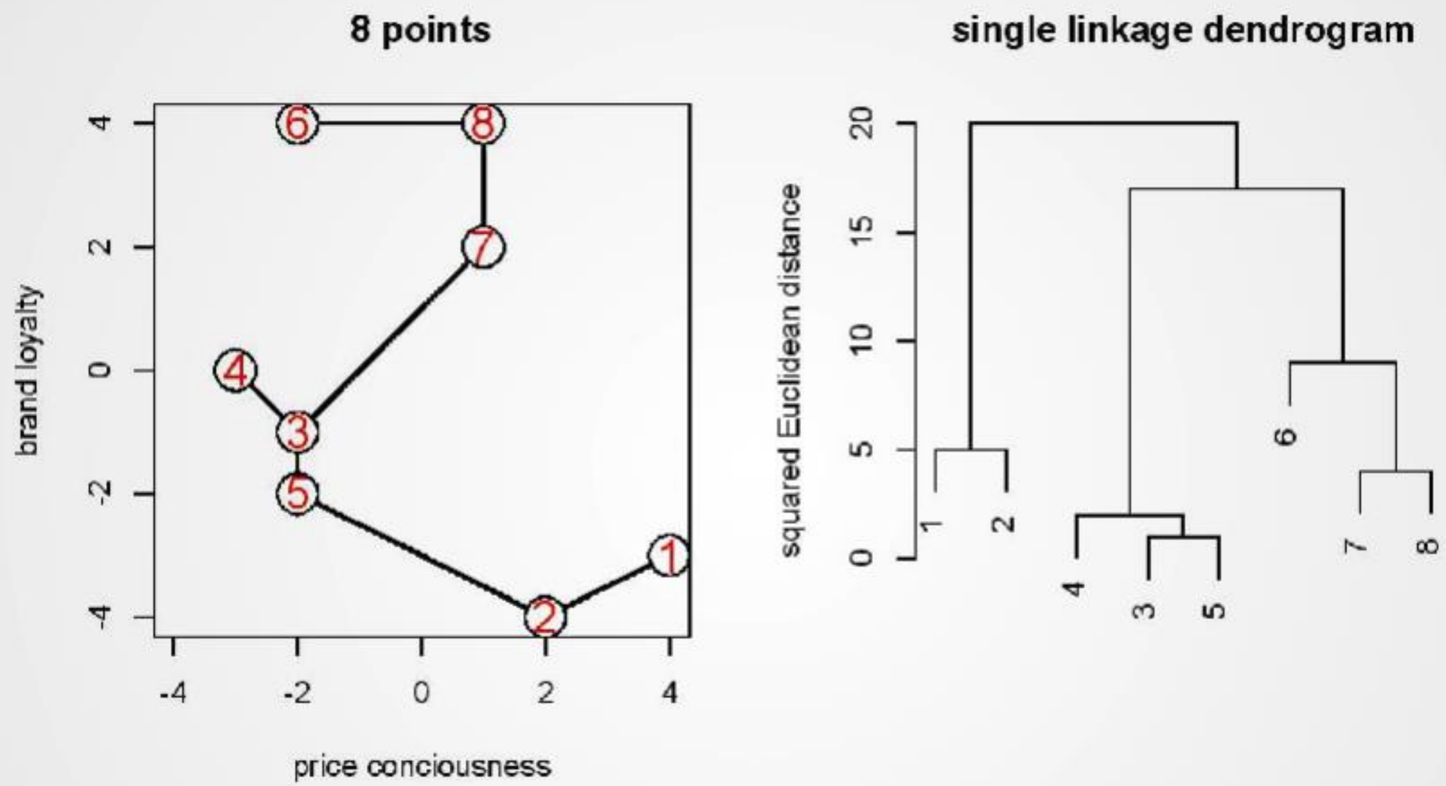
- distance between two clusters r and s : the smallest value of the individual distances.

$$L(r,s) = \min\{D(x_{ri}, x_{sj})\}$$

- Also called the Nearest Neighbor algorithm.
- Single linkage Algo tends to build large groups.



A simple example



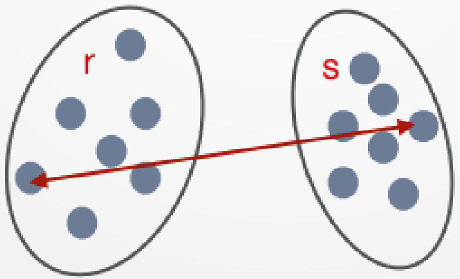
Single linkage algorithm on squared Euclidean distance for 8 point example with dendrogram. SMSclus8pd

Complete Linkage Algorithm

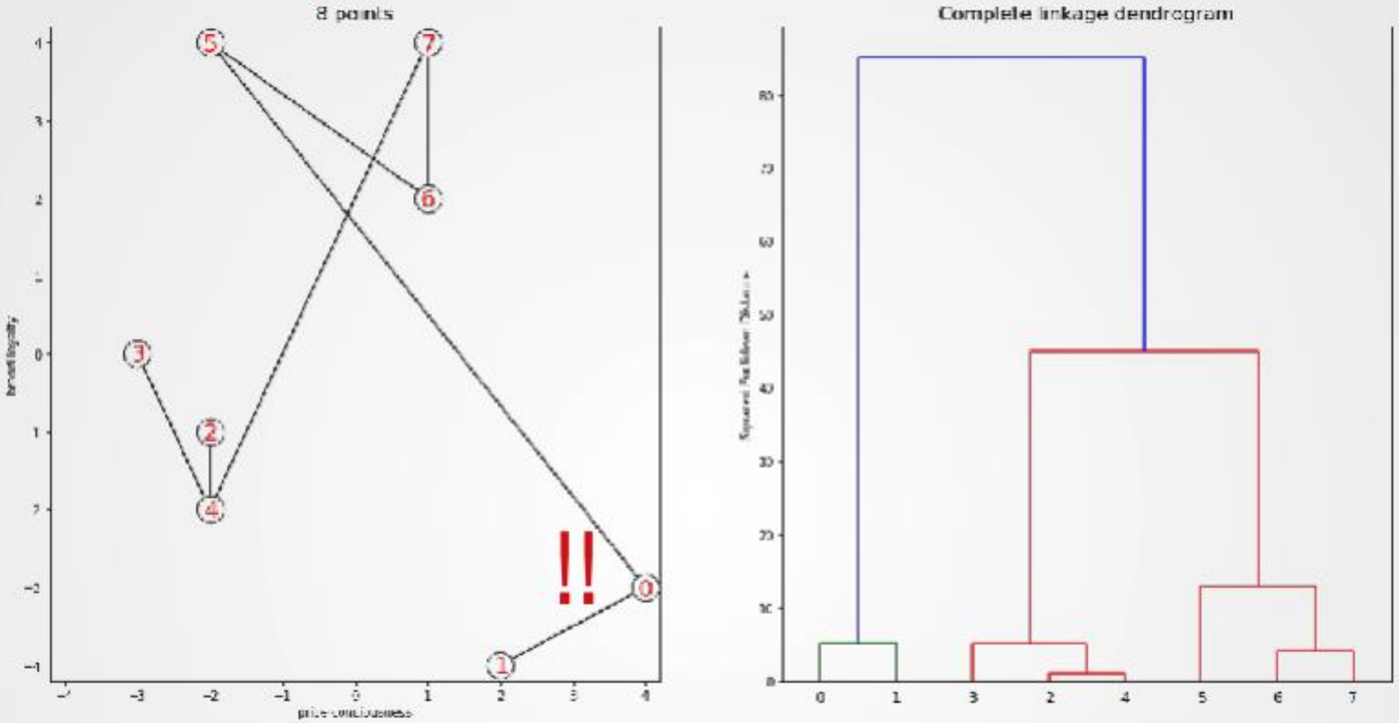
- Considers the largest (individual) distance

$$L(r, s) = \max\{D(x_{ri}, x_{sj})\}$$

- Also called Farthest Neighbor algorithm.
- Will cluster groups where all the points are proximate, since it compares the largest distances.



A simple example



!!

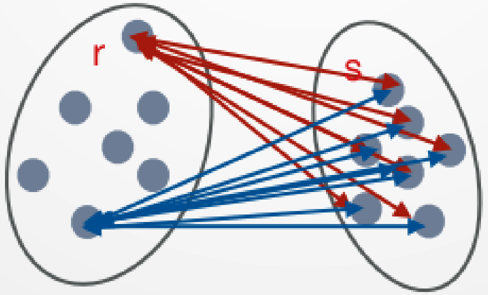
Complete linkage algorithm on squared Euclidean distance for 8 point example with dendrogram. SMSclus8pd

Average Linkage Algorithm

A compromise between nearest and farthest neighbor distance.

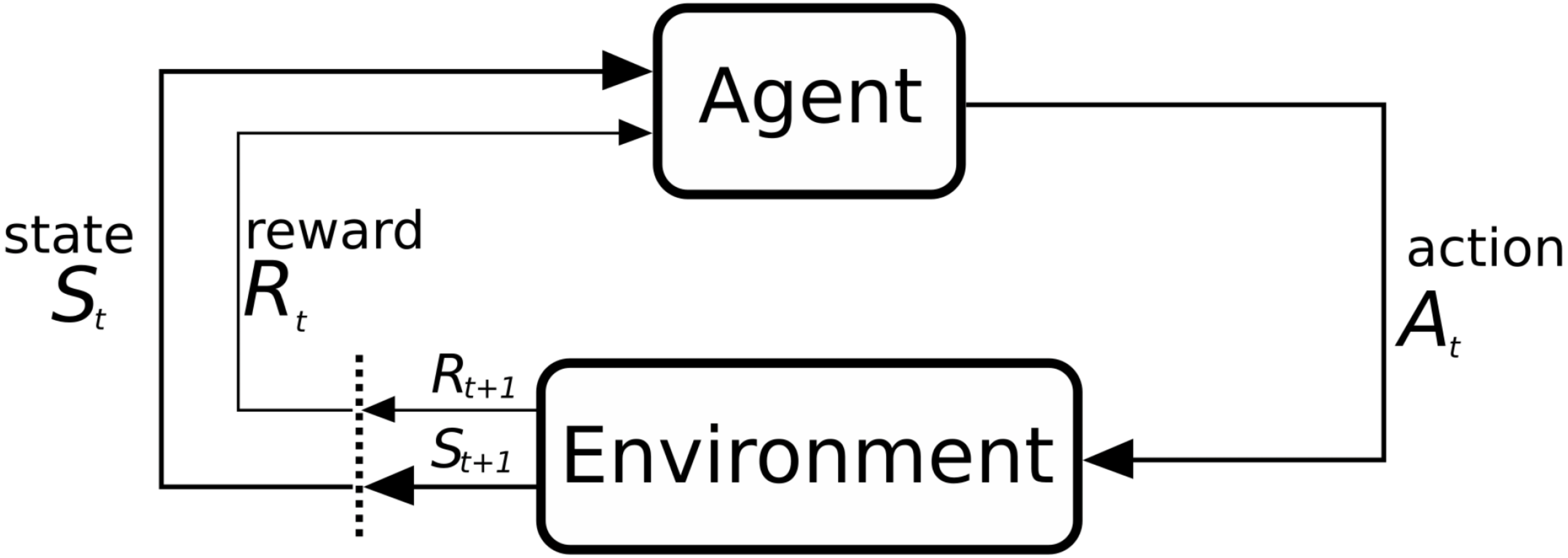
Average all mean distances:

$$L(r,s) = \frac{1}{n_r n_s} \sum_{i=1}^{n_r} \sum_{j=1}^{n_s} D(x_{ri}, x_{sj})$$



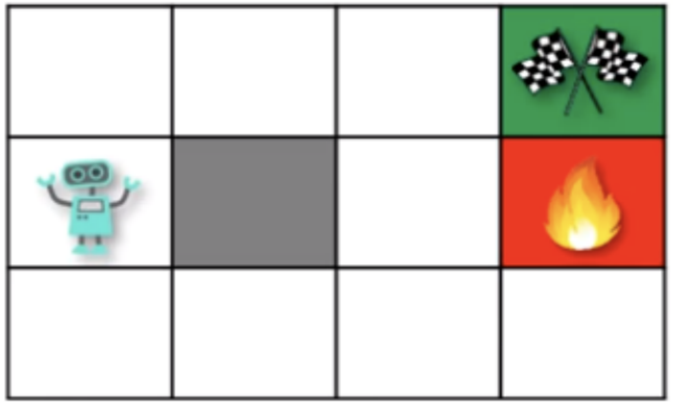
11. Reinforcement Learning: Basic

Source Material:
Joerg Osterrieder, University of Twente
Martijn Mes, University of Twente


Agent
state
S_t
reward
R_t
R_{t+1}
S_{t+1}
Environment
action
A_t

The Idea of Value-Based RL

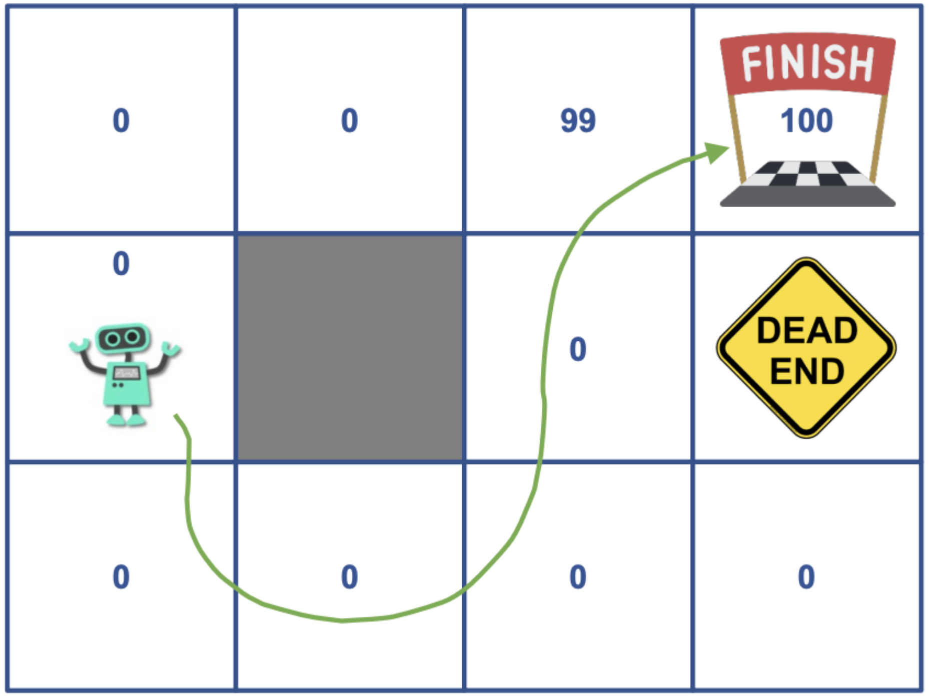
- We have to learn the consequences of our actions (rewards + reachable states):
  - *V(S)* = (discounted) (in)finite future rewards from state *S* onwards given an optimal policy
    Q(S,a) = previous + direct reward of *a* in *S*



- Ways of learning:
  - Look-ahead one step, take action, update *V(S)/Q(S,a)* → *TD(0)*
  - Play out an entire episode with a "given policy" (probably using Monte Carlo simulation) and propagate values to update the Q-values for observed *(S,a)* combinations or *V(S)* for observed *S* → *TD(1)*

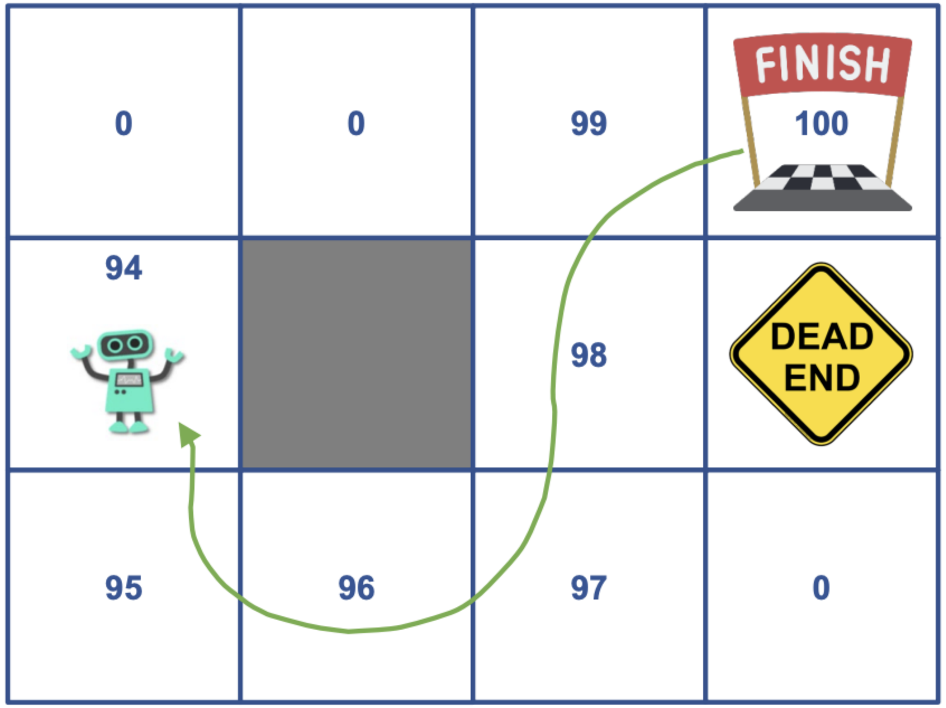
Learning the Value of States

- *V(S)* = value of *S*
- Ways of learning:
  1. Look-ahead one step, take action, update *V(S)*
  2. Play out entire episode and propagate values to update *V(S)* for all encountered *S*



Learning the Value of States

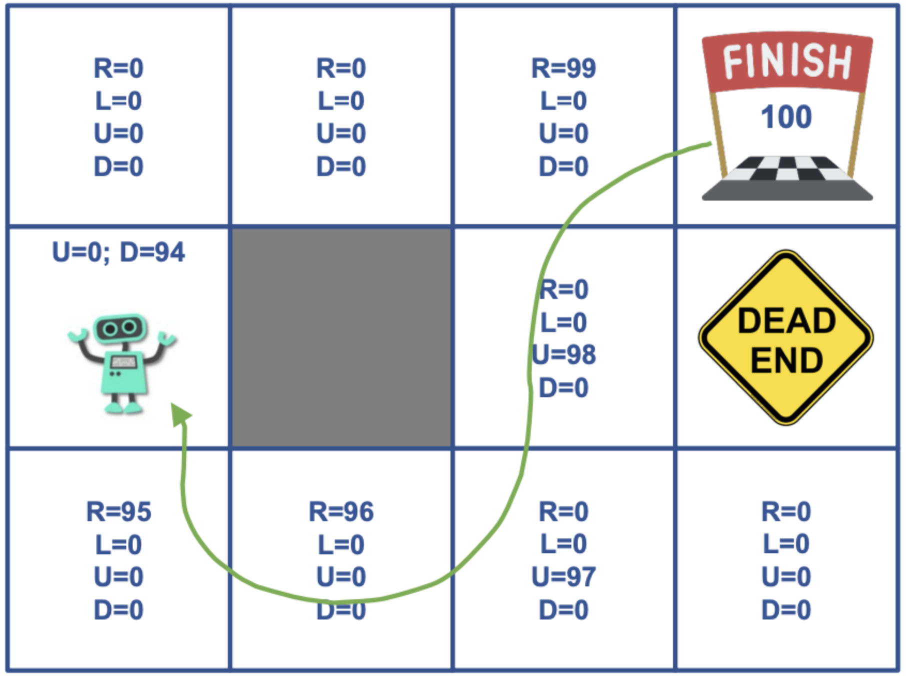
- *V(S)* = value of *S*
- Ways of learning:
  1. Look-ahead one step, take action, update *V(S)*
  2. Play out entire episode and propagate values to update *V(S)* for all encountered *S*



0	0	99	FINISH 100
94		98	DEAD END
95	96	97	0

Learning the Value of State-Action Pairs

- *Q(S,a)* = value of action *a* in *S*
- Ways of learning:
  - Look-ahead one step, take action, update *Q(S,a)*
  - Play out entire episode with a "given policy" and propagate values to update *Q(S,a)* for all encountered *(S,a)* combinations



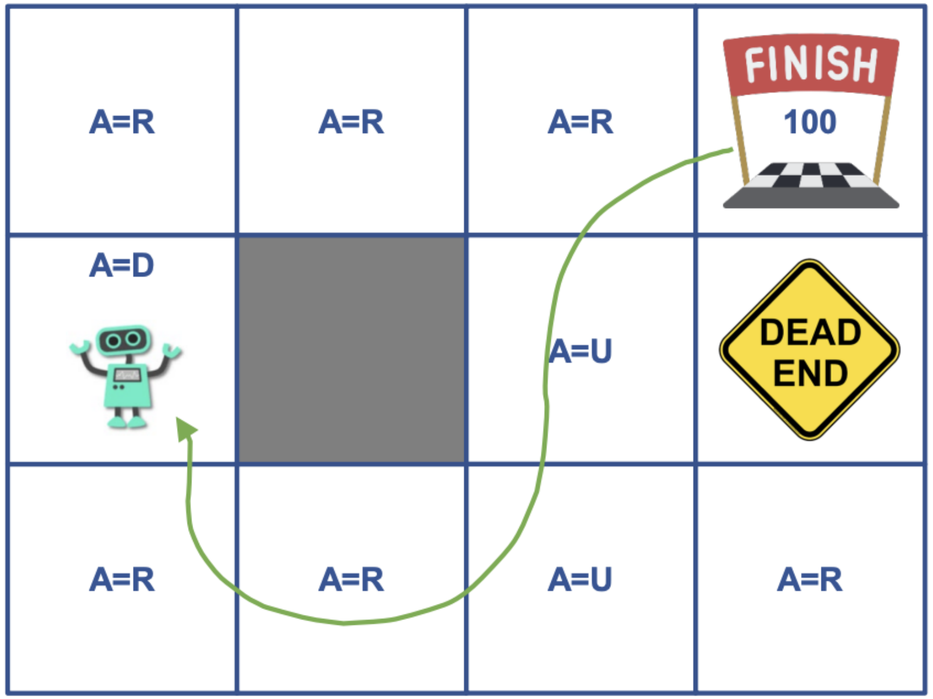
R=0
L=0
U=0
D=0
	R=0
	L=0
	U=0
	D=0
		R=99
		L=0
		U=0
		D=0
			FINISH
			100

U=0; D=94		R=0
		L=0
		U=98
		D=0
			DEAD END

R=95
L=0
U=0
D=0
	R=96
	L=0
	U=0
	D=0
		R=0
		L=0
		U=97
		D=0
			R=0
			L=0
			U=0
			D=0

Learning the Policy Directly

- *f(S)* = *a*
- Ways of learning:
  - Evaluate long term impact of all decisions in a state
  - Update policy function approximation *f(S)*



A=R	A=R	A=R	FINISH 100
A=D		A=U	DEAD END
A=R	A=R	A=U	A=R

LEARNING DIMENSIONS

- Model-free or model-based: do we have a model of the world, i.e., of the rewards and transition probabilities?
- Real-world (online) or simulator (offline): can we train offline in a simulator before implementing our decision/policy in the real world?
- Active or passive learning: do we simultaneously need to learn the value functions and the policy (active) or is the policy already given (passive)?
- On-policy or off-policy: do we learn the optimal policy independently on the agent's actions (off) or does the agent learn the value of the policy followed including the exploration steps (on)? The latter constrains our learning process, as we need an exploration strategy that is built into the policy itself.

12. Policy-based RL

Source Material: Wouter van Heeswijk, University of Twente

Value function approximation [1/2]

- So far, we stayed close to Dynamic Programming paradigm:
  - Replace true value functions V with approximation $\bar{V}$
  - Several ways to find suitable $\bar{V}$ (e.g., Q-table, features)
  - Finding optimal value functions equates finding optimal policy

- Disadvantages of VFA:
  - Falls apart for continuous- and large action spaces
    - Must evaluate $\bar{V}(s, a)$ for every action a in state s.
  - Indirect and unnatural way of decision-making
    - Dynamic Programming not intuitive for everyone

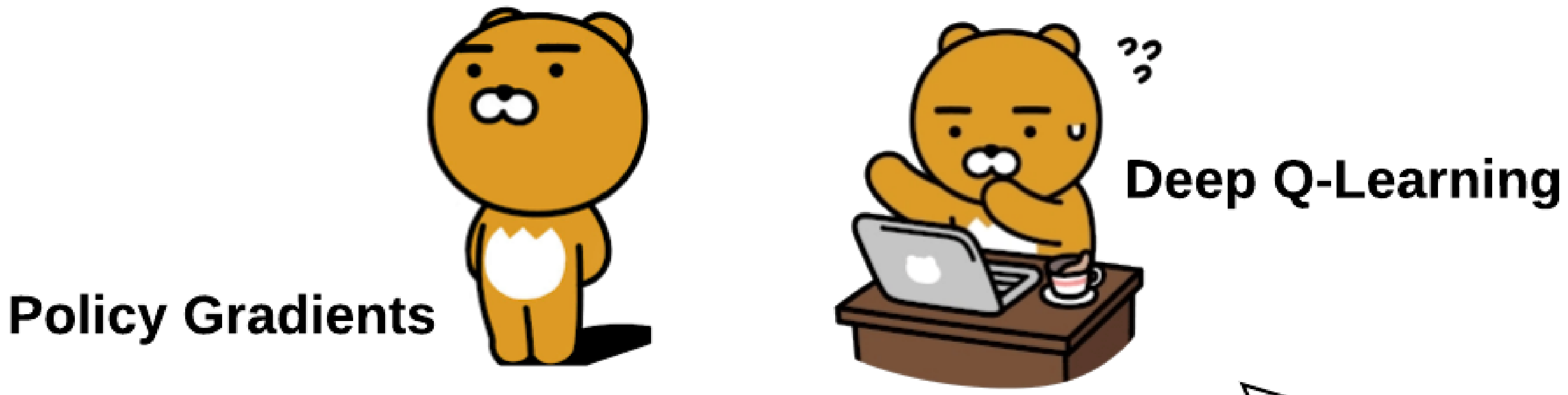
Value function approximation [2/2]

- We already abandoned optimality, no need to stick to Dynamic Programming approach
- Objective is not to solve Bellman equation, but to maximize reward over certain time horizon!
- Alternative: Directly adjust decision-making policy
  - Often more natural
  - Value functions are just a *means* to improve policy
- Recall: policy simply maps state to action!

$$\pi: s \to a$$

Policy function approximation – PFA vs VFA

Source: https://pylessons.com/Beyond-DQN



Note: discrete policy gradients still require full enumeration of action probabilities!

Go Right

Please wait, I am still calculating Q value, only 41891 actions left...

$$a = \pi_\theta(s)$$

$$a = \underset{\boxed{a\in\mathcal{A}}}{\arg\max}[r(s,a) + Q(s,a)]$$

Policy function approximation

- How do we improve a policy?

Policy function approximation

- How do we improve a policy?

- Basic mechanism:
  - Define policy π_θ with tunable parameters θ
  - Take actions according to policy
  - Observe corresponding rewards
    - Typically reward trajectory $r(\tau) = \sum_{t=0}^{T} r_t$
  - Adjust policy π_θ (i.e., adjust parameters θ)
  - Observe whether rewards improve
  - Repeat

Policy function approximation

- But: how do we know in what direction to update policy?
    - Sell higher/lower? Keep less/more inventory?

- A possible solution is to work with stochastic policies.
    - Allows measuring the *difference* between actions
    - So far, we used policies $\pi: s \to a$ (deterministic)
    - Now, we will use policies $\pi: s \to \mathbb{P}(a|s)$ (stochastic)

- We have two sources of information: (i) reward trajectory and (ii) probability of trajectory
    - Intuition: increase probability of high-reward trajectories

Policy function approximation

- We adjust the tunable policy π_θ based on observed reward- and probability trajectories.
  - Mathematically speaking, we compute the gradient
    - Gradient is simply a vector of partial derivatives for each θ

- We can express the gradient as an *expectation*, thus we can use simulation (*sampling*) to approximate it

- PFA *may* tackle both large state- and action spaces
  - However, there are drawbacks as well

Policy function approximation

- Gradient method not the only way to tailor policy

- Non-gradient solutions:
  - Genetic algorithms
  - SIMPLEX
  - Hill climbing

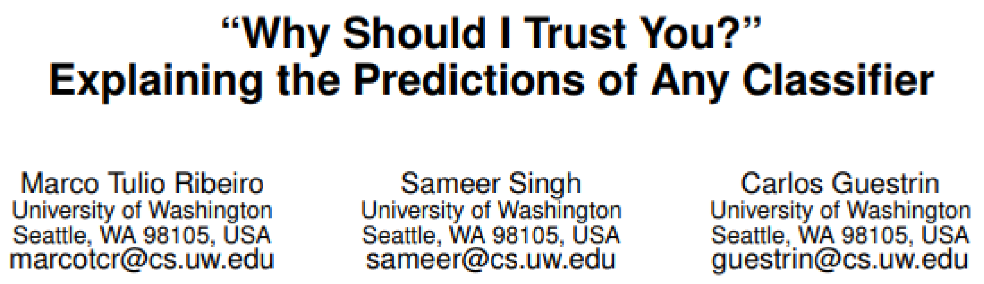
- Gradient methods often more efficient
  - Stochastic gradient descent
  - Newton’s method

13. LIME: Basics

Source Material: Branka Hadji Misheva RL Training Week

Local Interpretable Model-Agnostic Explanations

- LIME → explains the prediction of **any machine learning model** by learning an interpretable model **locally** around a specific instance of interest
- Works with classification & regression
- Works with tabular data, text and pictures



“Why Should I Trust You?”
Explaining the Predictions of Any Classifier

Marco Tulio Ribeiro
University of Washington
Seattle, WA 98105, USA
marcotcr@cs.uw.edu

Sameer Singh
University of Washington
Seattle, WA 98105, USA
sameer@cs.uw.edu

Carlos Guestrin
University of Washington
Seattle, WA 98105, USA
guestrin@cs.uw.edu

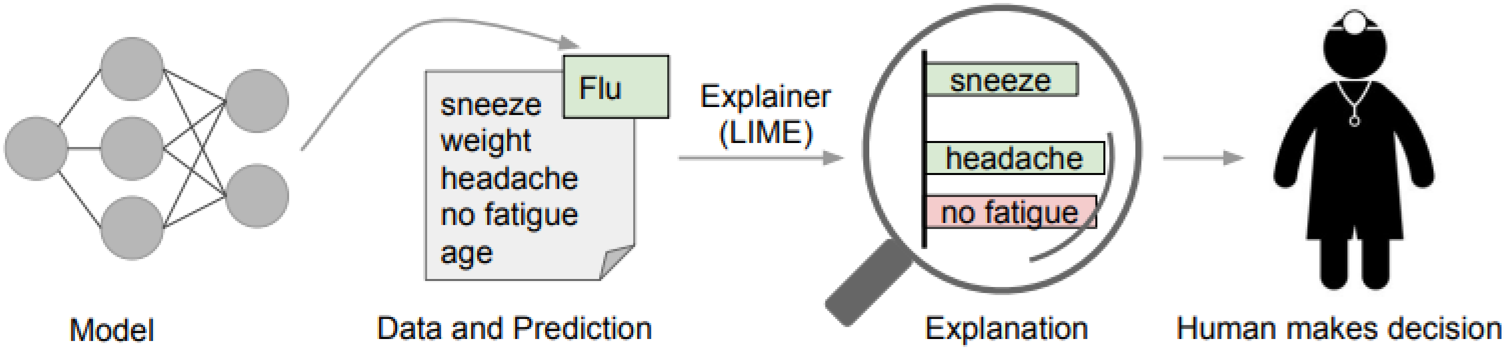
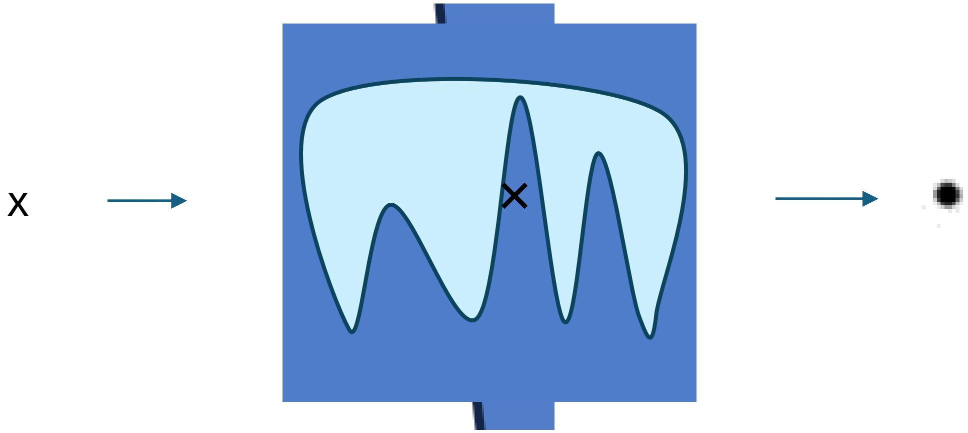


Image source: Ribeiro et al. (2016)

LIME – How does it work?



Highly accurate at local level!

So local that one can **apply a simple, interpretable model** to understand the relationship!

Steps

For which you require explainations

- **Pick an observation**, create and permute data;
- Calculate similarity between the original observations and the permutations;
- Make predictions on new data using your black box;
- **Fit a simple model** to the permuted data with n features and similarity scores as weights;
- **Coefficients from the simple model serve as an explanation of the model behavior** at the local level.

14. Shapley: Basics

Source Material: Branka Hadji Misheva RL Training Week

Shapley Values: **DETAILS**

- Given:
  - A set N of n players: $N = \{1,2,\dots,n\}$
  - A characteristic function v that assigns a value to every coalition (subset of players)

The **Shapley value for a player i** is a measure of the **average contribution of i to all possible coalitions**.

Kumar (2020)

The Math

$$\phi_i(v) = \sum_{S \subseteq N\{i\}} \frac{|S|!(n-|S|-1)!}{n!}(v(S \cup \{i\}) - v(S))$$

The Math

$$\boxed{\phi_i(v)} = \sum_{S \subseteq N\{i\}} \frac{|S|!(n-|S|-1)!}{n!}(v(S \cup \{i\}) - v(S))$$

Shapley value for a given feature i

We calculate the contribution of each feature to a prediction by considering all possible subsets of features and computing the marginal contribution of each feature across these subsets

The Math

$$\phi_i(v) = \boxed{\sum_{S \subseteq N\{i\}}} \frac{|S|!(n - |S| - 1)!}{n!}(v(S \cup \{i\}) - v(S))$$

Sum over all possible coalitions that do not contain i

The Shapley value aims to measure the average contribution of feature i to the prediction, **considering all possible scenarios where i could join a coalition**

The Math

$$\phi_i(v) = \sum_{S\subseteq N\{i\}} \frac{|S|!(n-|S|-1)!}{n!}(v(S\cup\{i\}) - \boxed{v(S))}$$

Coalition without feature i

The Math

$$\phi_i(v) = \sum_{S \subseteq N\{i\}} \frac{|S|!\,(n-|S|-1)!}{n!} \boxed{(v(S \cup \{i\})} - v(S))$$

Coalition with feature i

The Math

$$\phi_i(v) = \sum_{S \subseteq N\{i\}} \frac{|S|!(n-|S|-1)!}{n!} \boxed{(v(S \cup \{i\}) - v(S))}$$

Marginal contribution of i to the coalition

Marginal change in the model's score **after adding feature i**

The Math

$$\phi_i(v) = \sum_{S \subseteq N\{i\}} \boxed{\frac{|S|!(n-|S|-1)!}{n!}} (v(S \cup \{i\}) - v(S))$$

Weighting the contributions of i by its share in the number of total coalitions

- |S| is the **size of the coalition** S (excluding feature i)
- n is the total **number of feature**