

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.

- $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$
- Minimize the **squared differences** between actual and predicted values.
- $\hat{\beta} = \arg \min_{\beta_0, \dots, \beta_p} \sum_{i=1}^n \left(y^{(i)} - \left(\beta_0 + \sum_{j=1}^p \beta_j x_j^{(i)} \right) \right)^2$
- **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.

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

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

$$\frac{odds(x_k + 1)}{odds(x_k)} = \frac{\exp(\beta_0 + \beta_1 x_1 + \dots + \beta_k (x_k + 1) + \dots + \beta_p x_p)}{\exp(\beta_0 + \beta_1 x_1 + \dots + \beta_k (x_k) + \dots + \beta_p x_p)} = \exp(\beta_k)$$

2. Decision Tree

Source Material: [Chen Huang, School of Business and Social Sciences Aarhus University](#)



Decision Tree

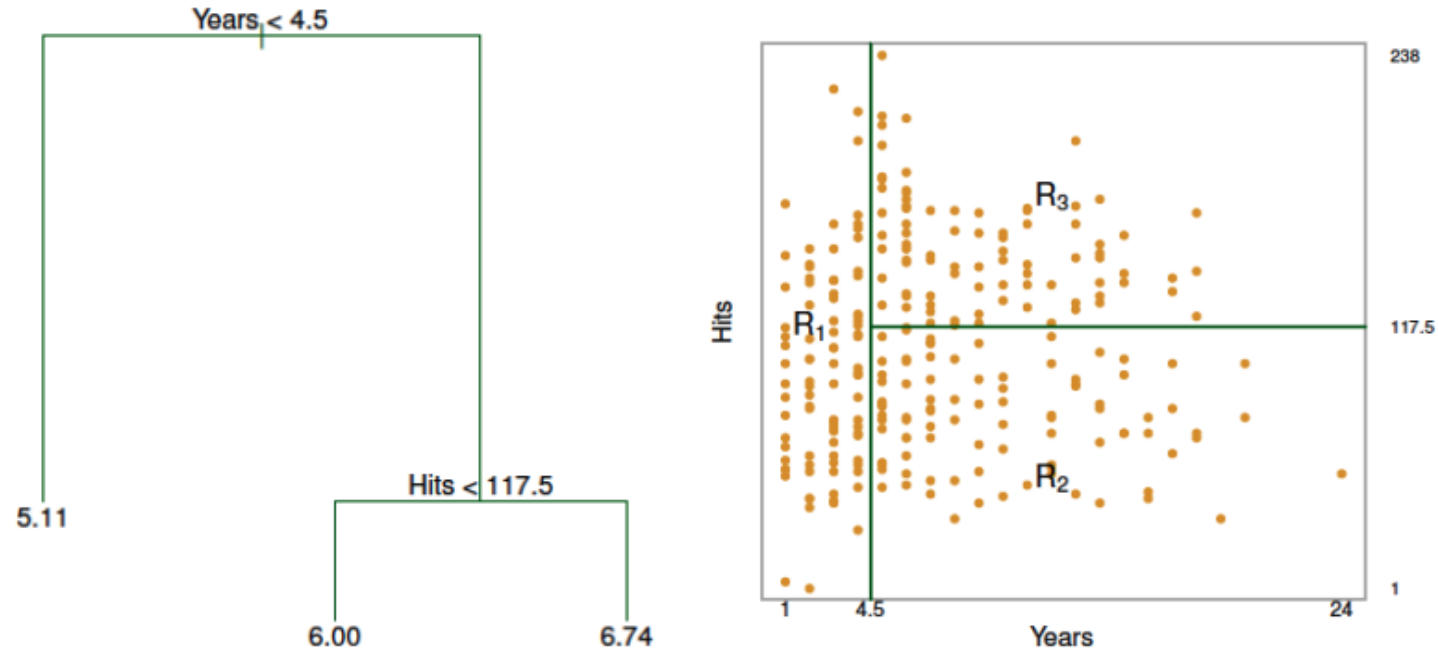


Figure 1: Left: A regression tree for predicting the log salary of a baseball player. Right: The three-region partition from the regression tree. (Figure 4.6 and 4.7 of ISLR)

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 $R_1(j, s) = \{\mathbf{X} : X_j < s\}$ and $R_2(j, s) = \{\mathbf{X} : X_j \geq s\}$, choose j and s that minimize

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

where \hat{y}_{R_k} is the mean response for the training data in $R_m(j, s)$, $m = 1, 2$.

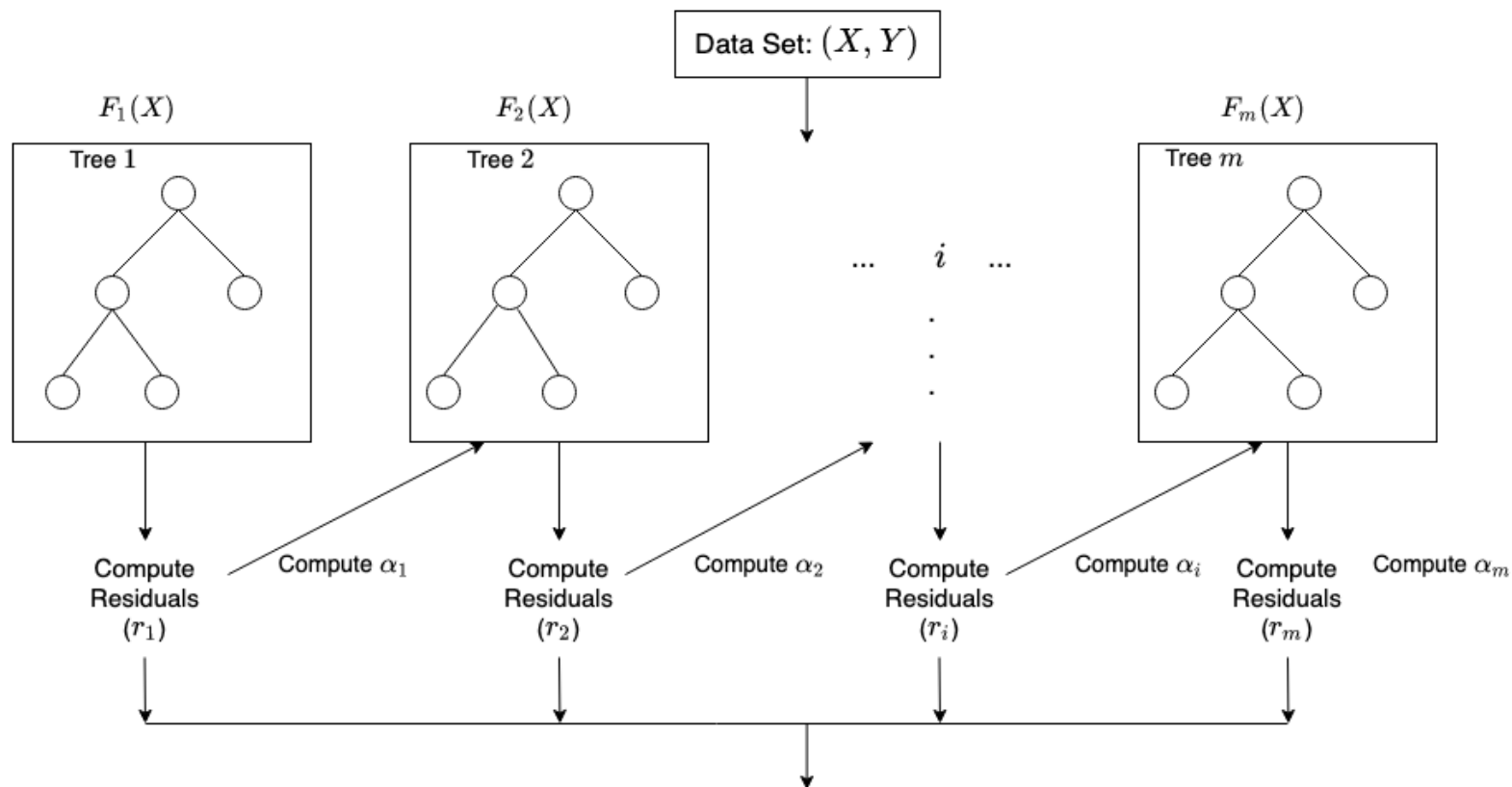
- Step 2: repeat the splitting process on the subregion that RSS reduces the most.
- ...
- Stop splitting when some stopping criterion reached, e.g. a minimum terminal node size.

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 respectively, 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

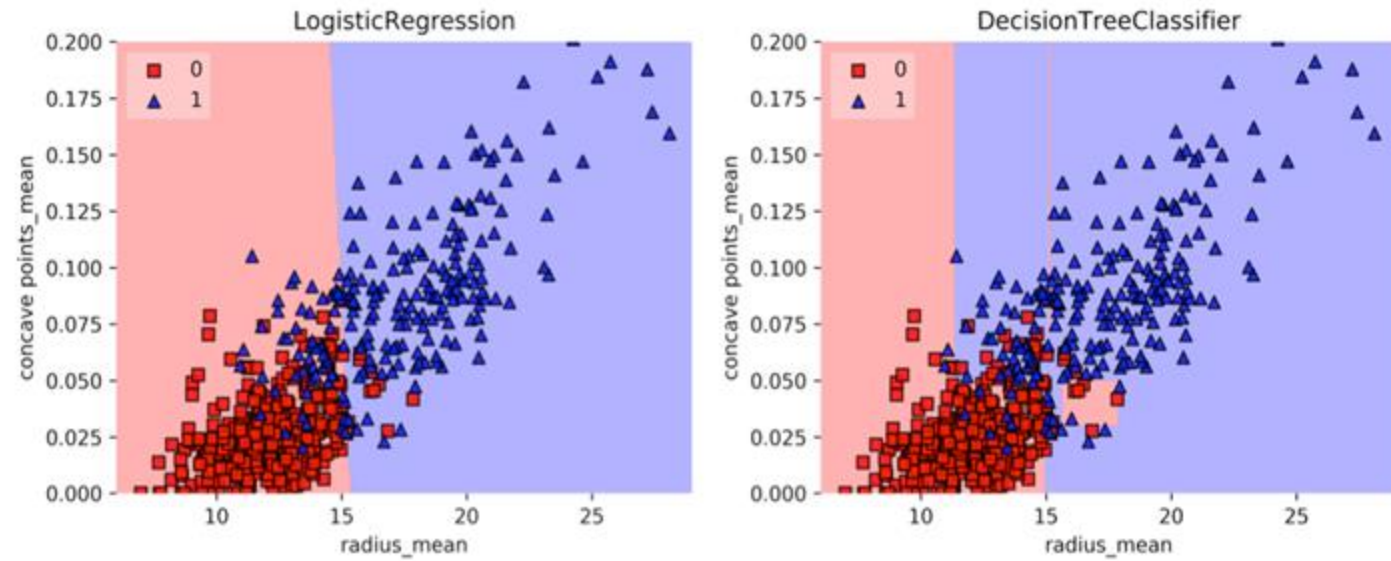
computed, r_i and compute the following: $\arg \min_{\alpha} = \sum_{i=1}^m L(Y_i, F_{i-1}(X_i) + \alpha h_i(X_i, r_{i-1}))$ where

$L(Y, F(X))$ is a differentiable loss function.

4. Evolution of NN

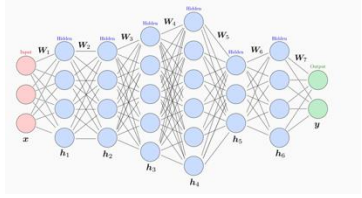
Source Material: Branka Hadji Misheva,
training material (full material available on
request)

Rule-based systems;
logistic regressions,
decision trees



Pre-deep learning

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



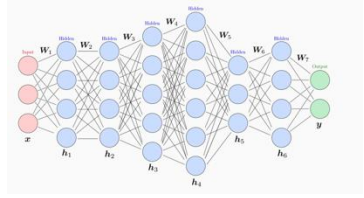
Introduction of backpropagation (Rumelhart et al., 1986) allowed training of **multi-layer neural networks**

No memory of previous inputs

Multi-layer perception



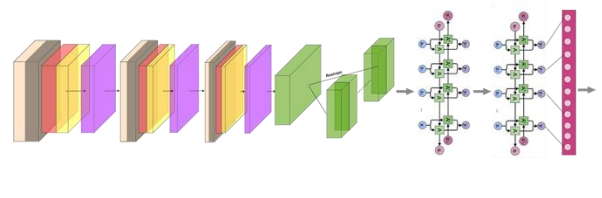
Deep learning



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

No memory of previous inputs

Multi-layer perception

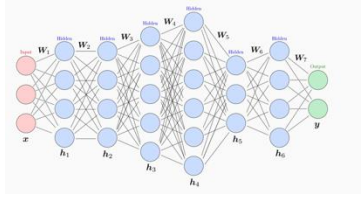


Use of **convolutional layers** with local receptive fields. Great for images and structured spatial data

No memory of previous inputs

Convolutional NN

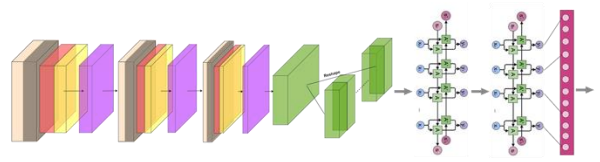
Deep learning



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

No memory of previous inputs

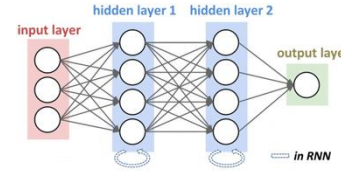
Multi-layer perception



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

No memory of previous inputs

Convolutional NN



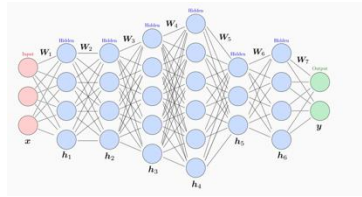
Use of **recurrent layers** that process data step-by-step. Great for sequential data (time series, text speech)

Maintains memory of previous inputs

Recurrent NN

Deep learning

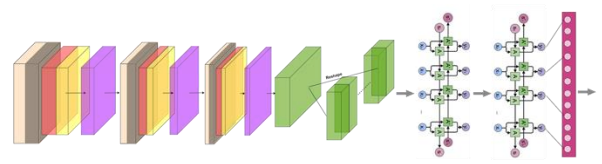
Limitation: Hard to train on long sequences; Sequential processing is slow (can't parallelize easily).



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

No memory of previous inputs

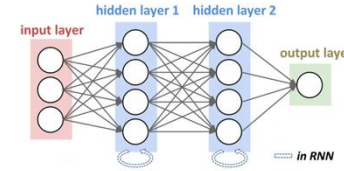
Multi-layer perception



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

No memory of previous inputs

Convolutional NN



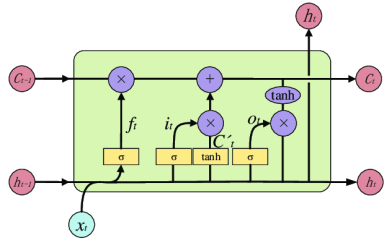
Use of **recurrent layers** that process data step-by-step. Great for sequential data (time series, text speech)

Maintains memory of previous inputs

Recurrent NN

Deep learning

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



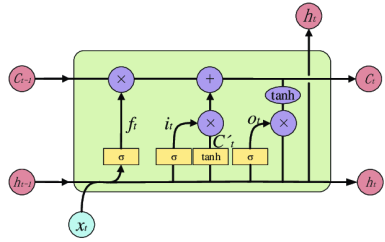
Long Short-Term Memory networks (**LSTMs**) are a special type of RNN designed to remember long-term dependencies without suffering from vanishing gradients.

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.

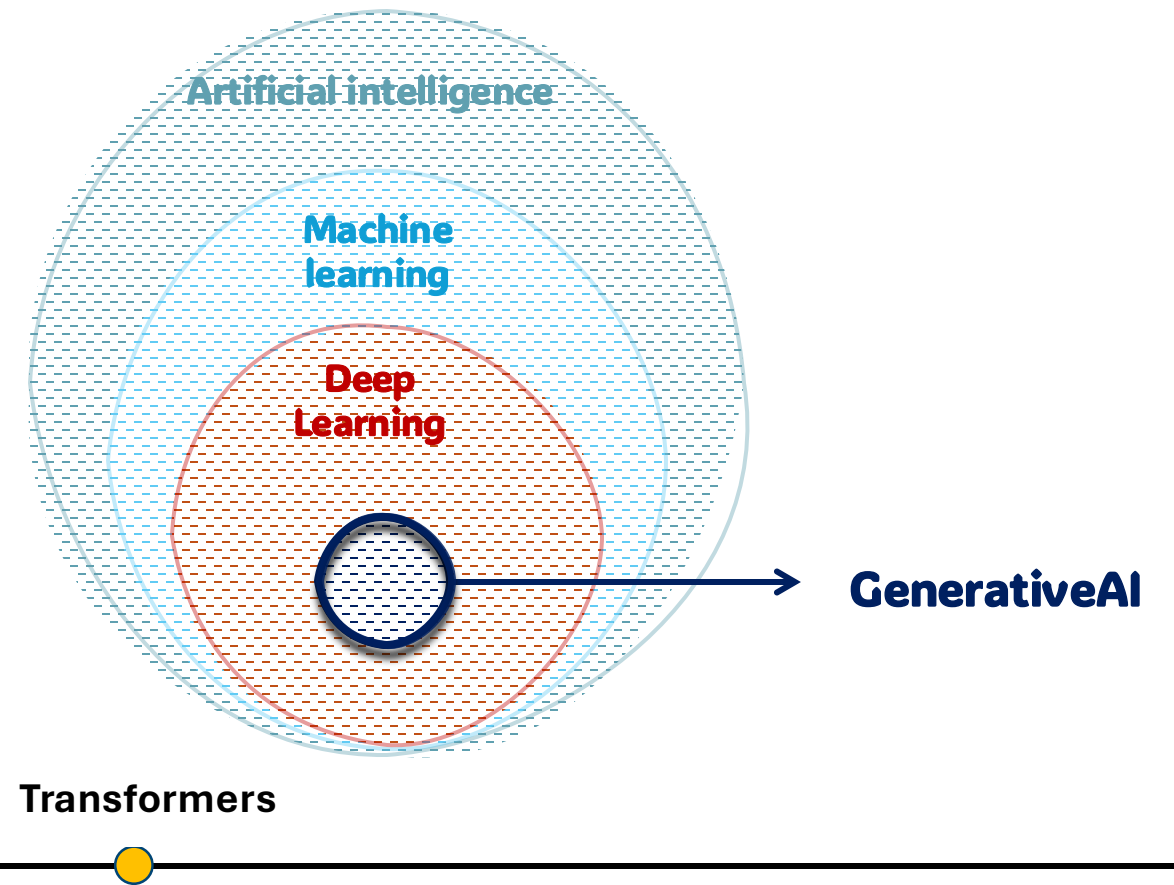


Long Short-Term Memory networks (**LSTMs**) are a special type of RNN designed to remember long-term dependencies without suffering from vanishing gradients.

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

Long Short-Term Memory

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**.



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

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

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

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

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.

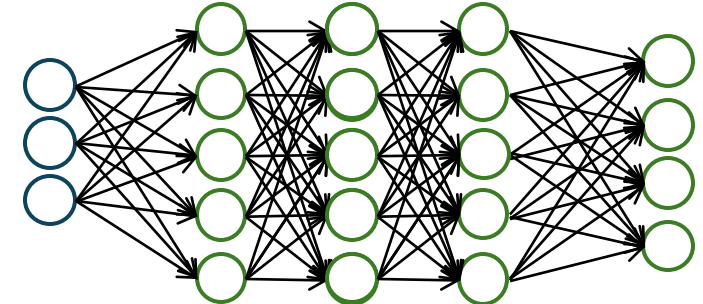
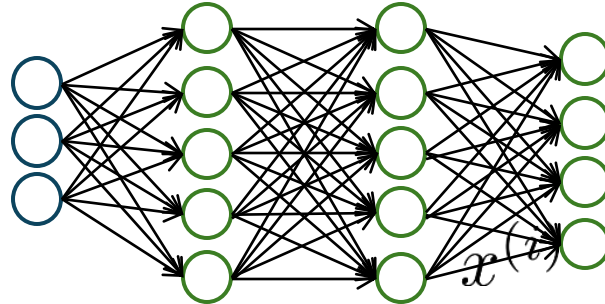
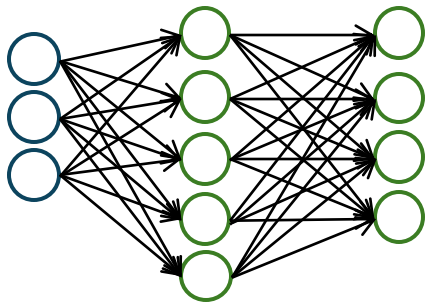
6. Training a NN

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)

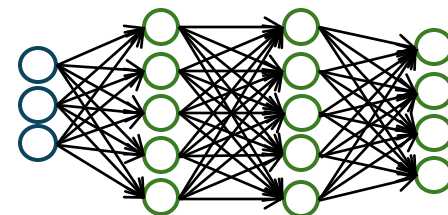
Training a neural network

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 $l = 2, \dots, L$).

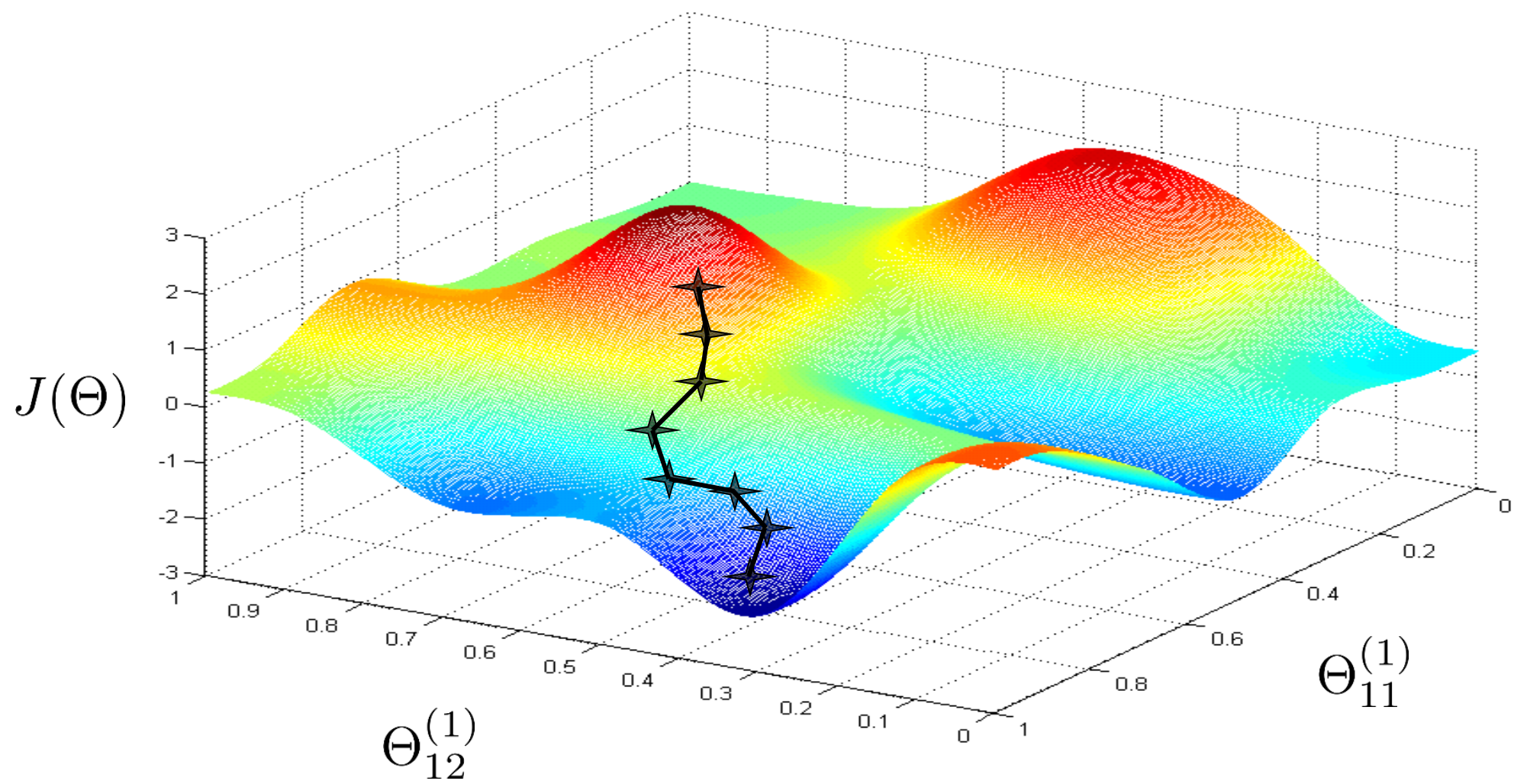


Training a neural network

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



7. Transformer

Source Material: [Poloclub GitHub - Transformer-explainer](#)

TRANSFORMER EXPLAINER

Examples ▾

Data visualization empowers users to **visualize**

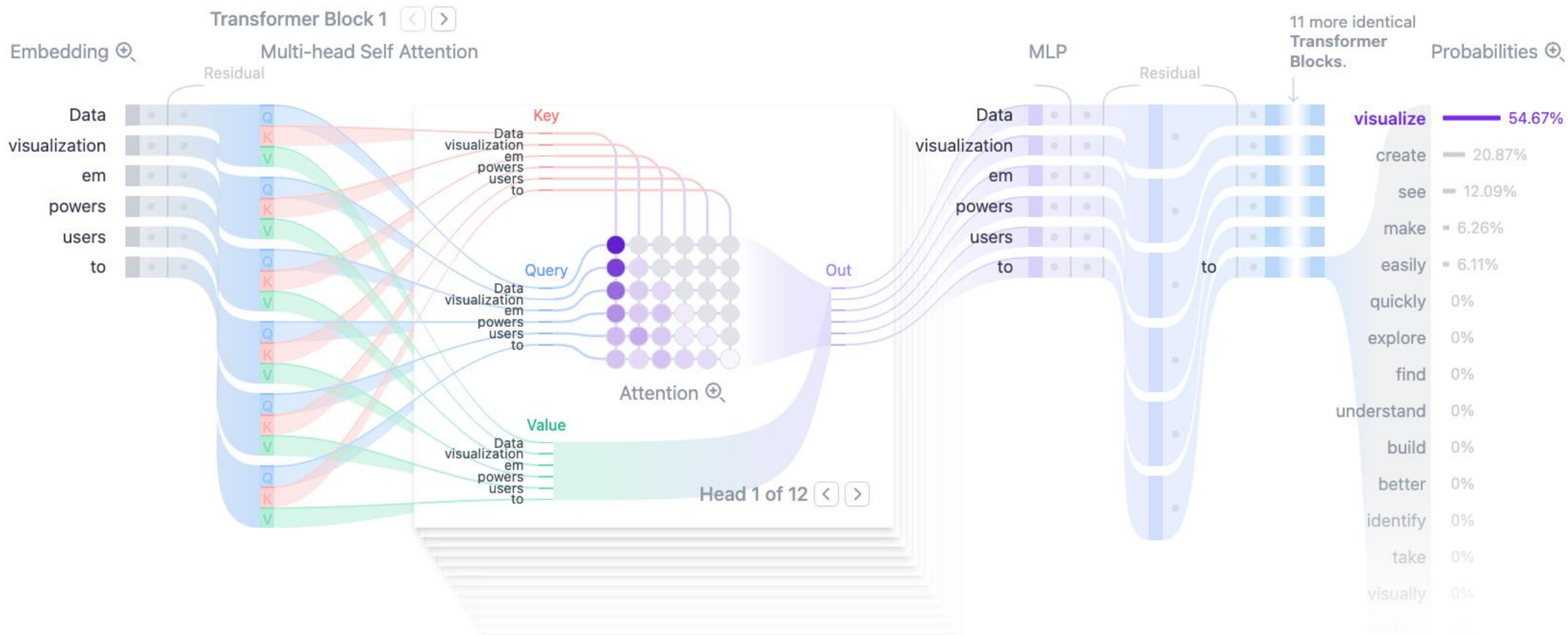
Generate

Temperature ?

0.8

Sampling ☒ Top-k ☐ Top-p

k=5



DEMO

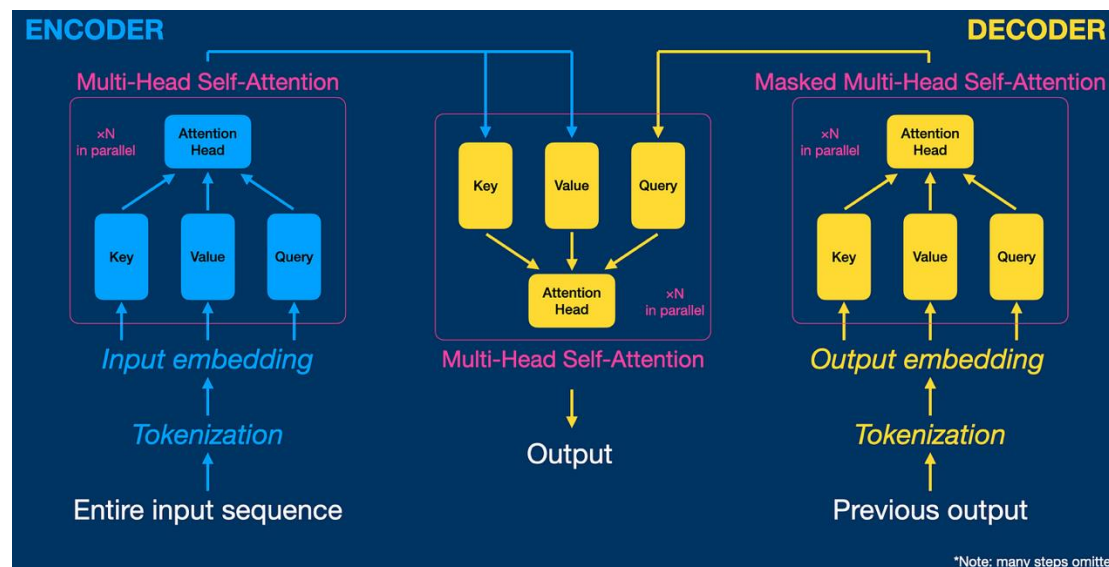
Generative Models

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).



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.

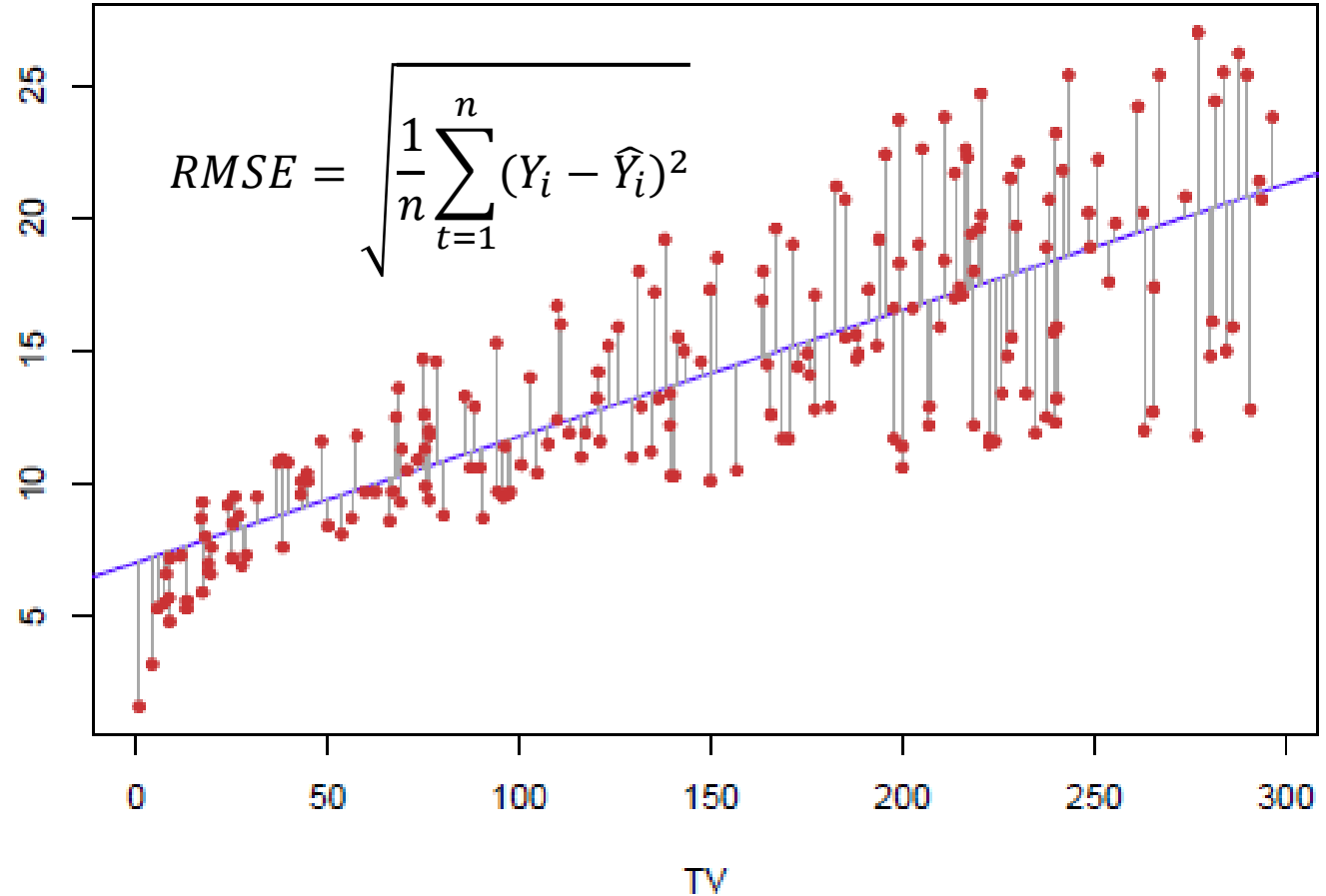
Goal: generate the next likely word, given a sequence of preceding words.

8. Model Evaluation

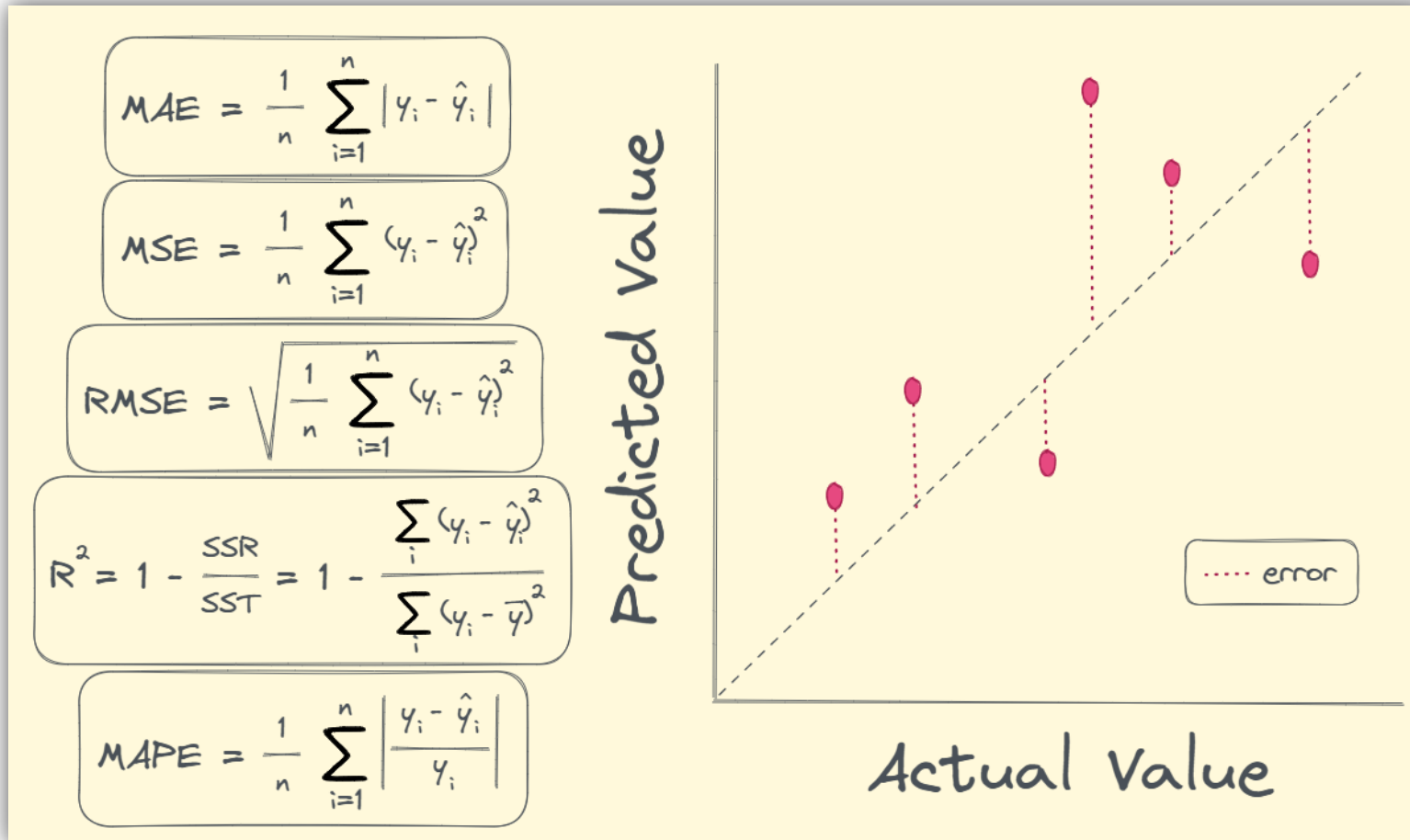
Source Material: Branka Hadji Misheva,
training material (full material available on
request)

Testing preformance: Regression models

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



Testing preformance: Regression models



Testing preformance: **Classification models**

- Confusion matrix

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

- **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:

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

where:

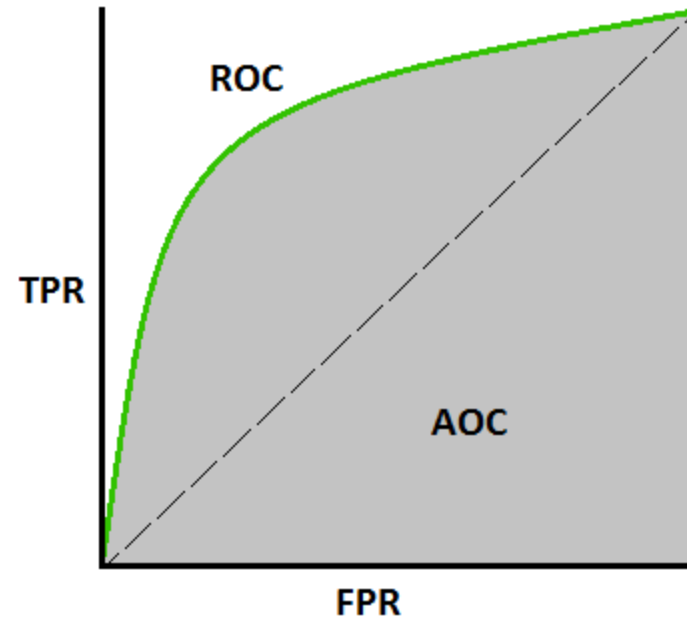
- **True Positive (TP)**: Actual and predicted class is positive
- **True Negative (TN)**: Actual and predicted class is negative
- **False Negative (FN)**: Actual class is positive and predicted negative
- **False Positive (FP)**: Actual class is negative and predicted positive

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

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

False Positive Rate

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

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

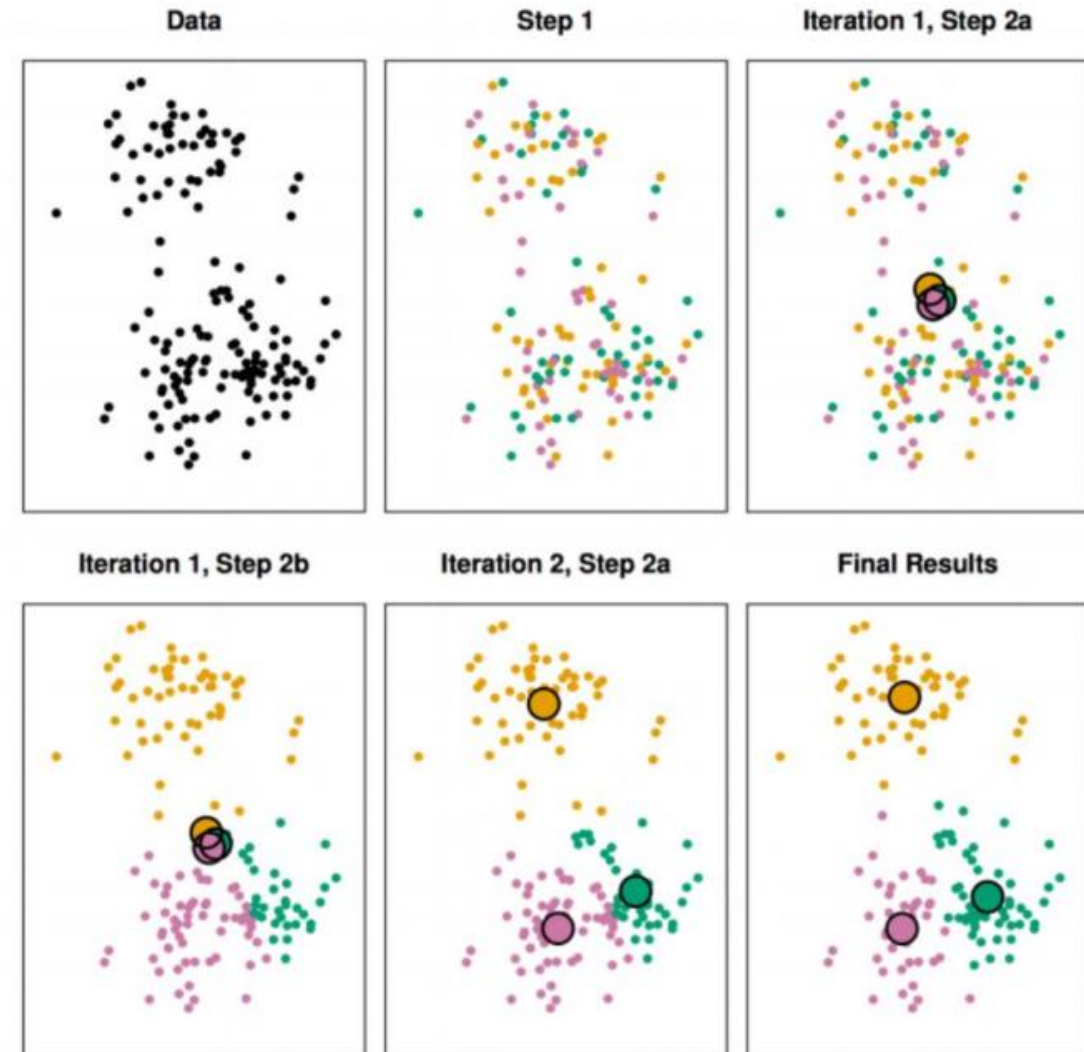
Source Material: Branka Hadji Misheva,
training material (full material available on
request)

K-means

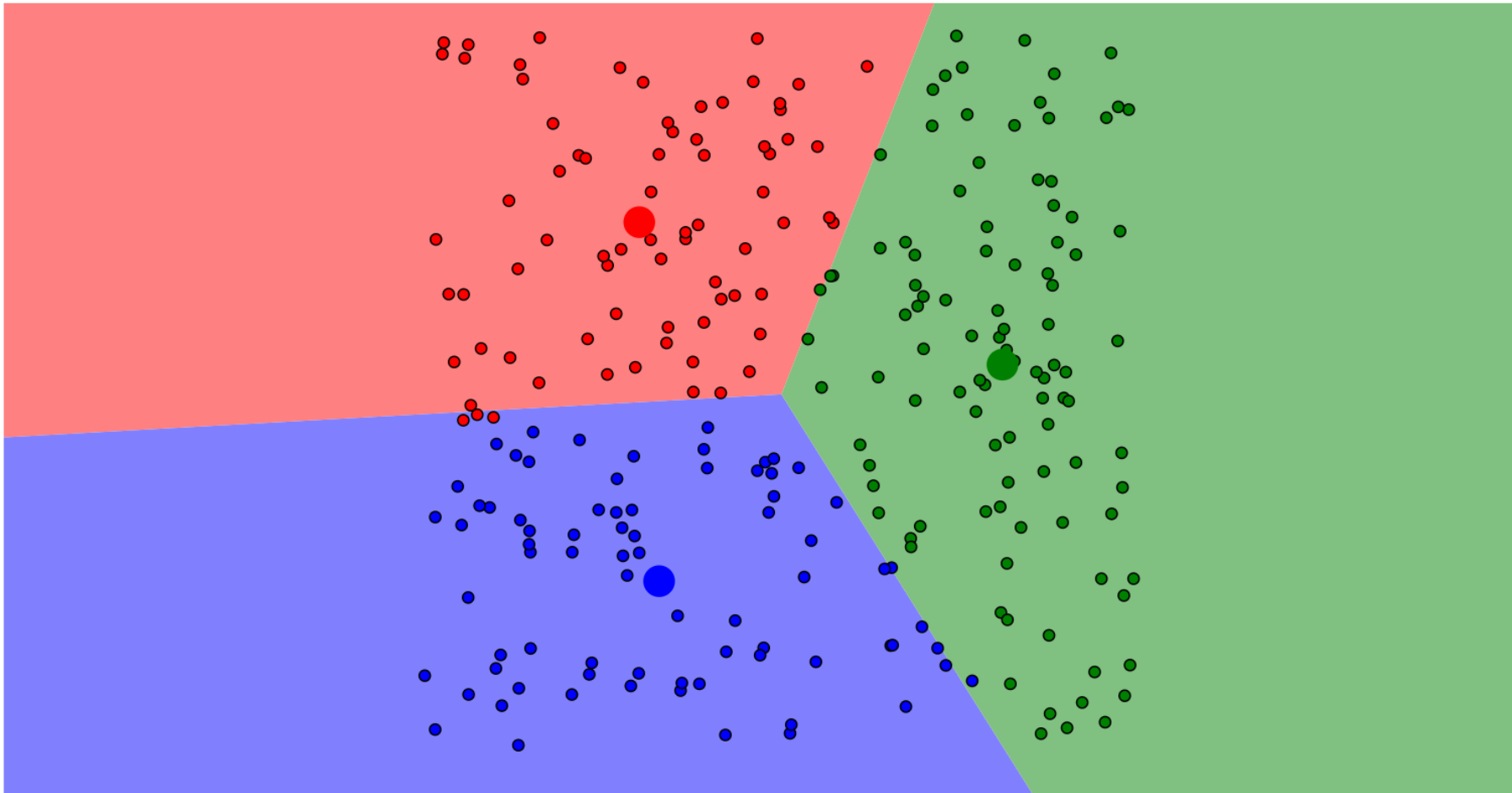
- K-means clustering – simple and very popular unsupervised machine learning algorithm
- Its basic operation is very simple:
 - ❑ Choose **number of clusters = k**
 - ❑ Randomly **choose the centroids** of each cluster
 - ❑ **Two-step:**
 - Step 1: Assignment
 - Step 2: Update



Repeat steps



Interactive Play



[Link to Demo](#)

K-means: DETAILS

- $C_1, \dots, C_k \rightarrow$ 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

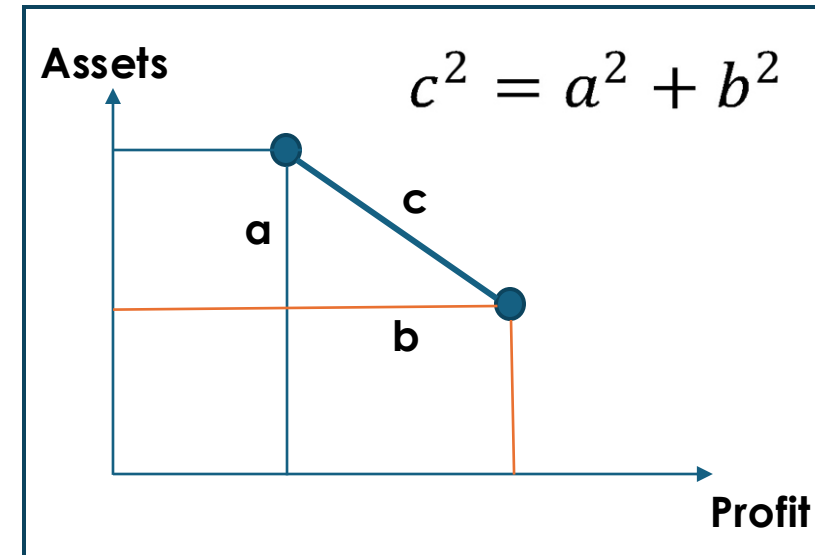
- What is a good clustering?
 - **Within-Cluster Variation** (WCV) is as small as possible
- The within-cluster variation for cluster C_k is a measure $WCV(C_k)$ of the amount by which the observations within a cluster differ from each other
- We are solving the following problem

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

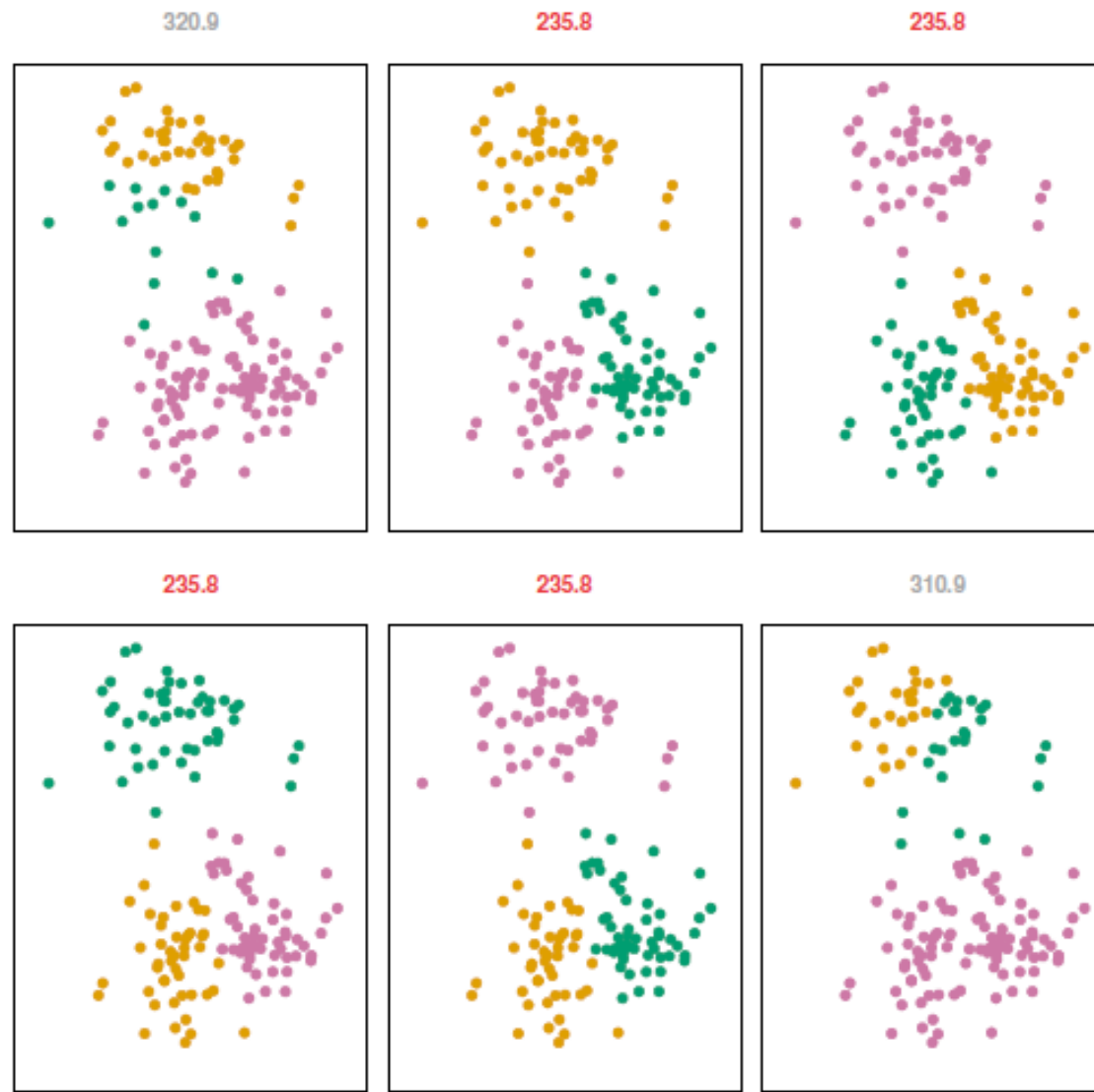
- Hence, we are looking for a partition that can minimize the WCV

K-means: DETAILS

- How do we measure the **within-cluster variation**?
- One option is to use **Euclidean Distance**
- What are some of the properties?
 - $\text{Dist}(x,y)=0$ if $x=y$ (the identity axiom)
 - $\text{Dist}(x,y)=\text{Dist}(y,x)$ (the symmetry axiom).
 - (if 3 points) $\text{Dist}(x,y) + \text{Dist}(y,z) \geq \text{Dist}(x,z)$ (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

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

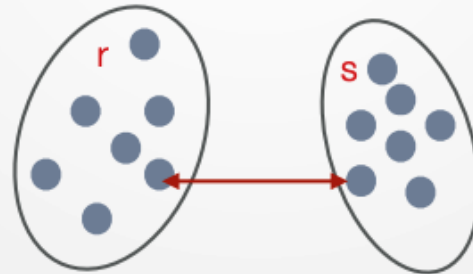
The 8 points example

Single Linkage Algorithm

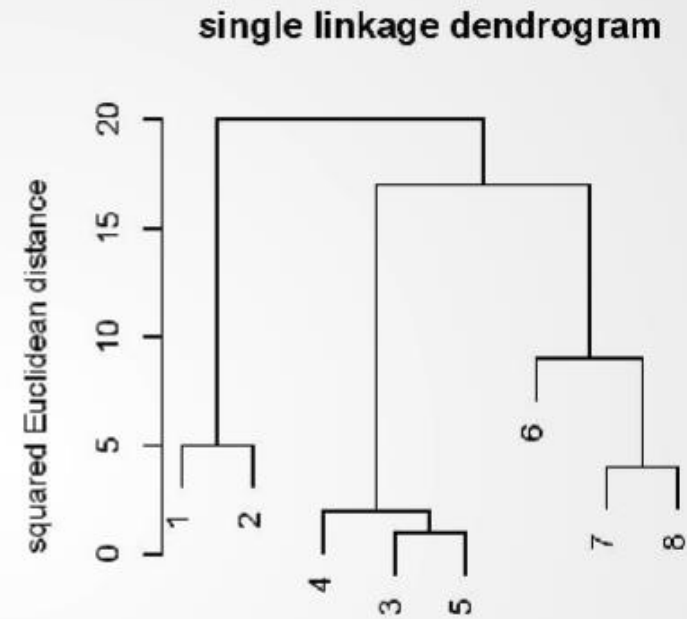
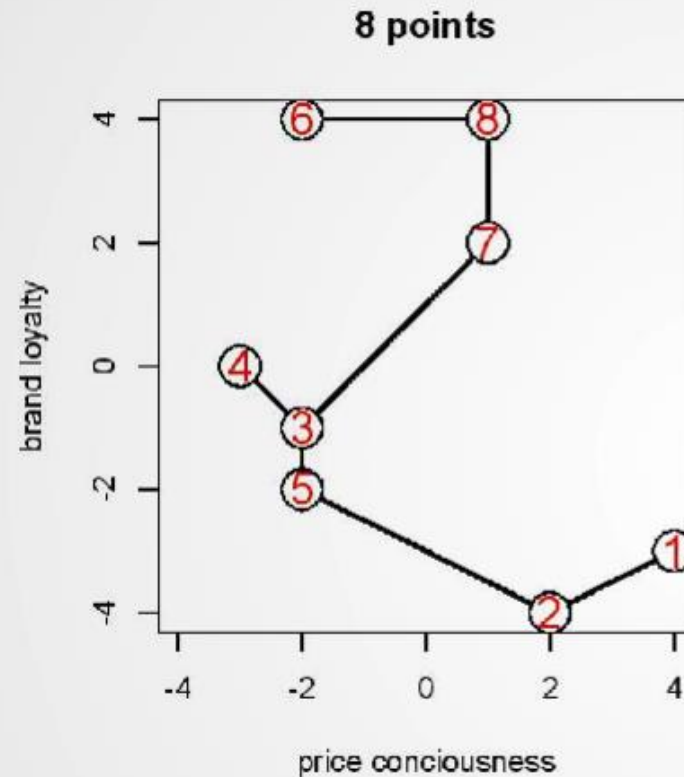
- distance between two clusters r and s : the smallest value of the individual distances.

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

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



A simple example



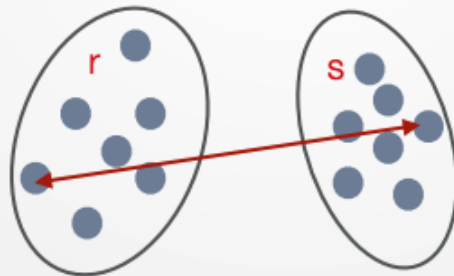
Single linkage algorithm on squared Euclidean distance for 8 point example with dendrogram.  SMSclus8pd

Complete Linkage Algorithm

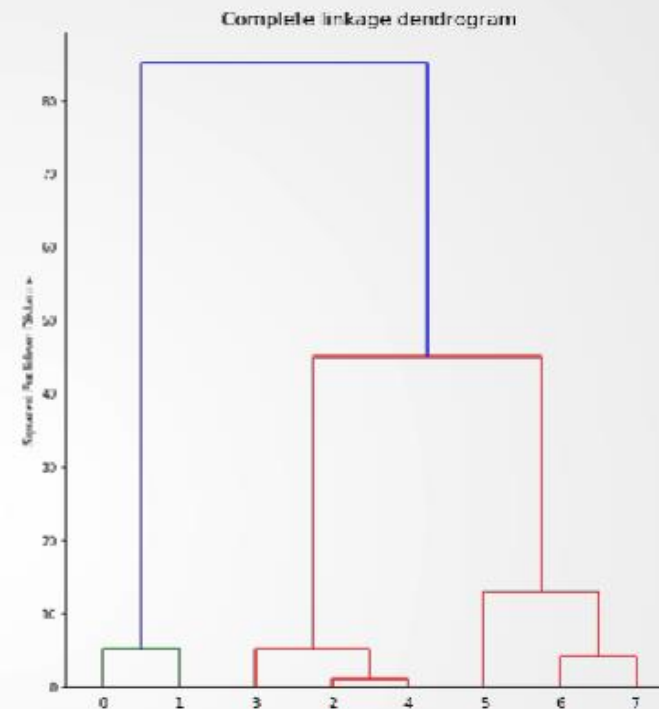
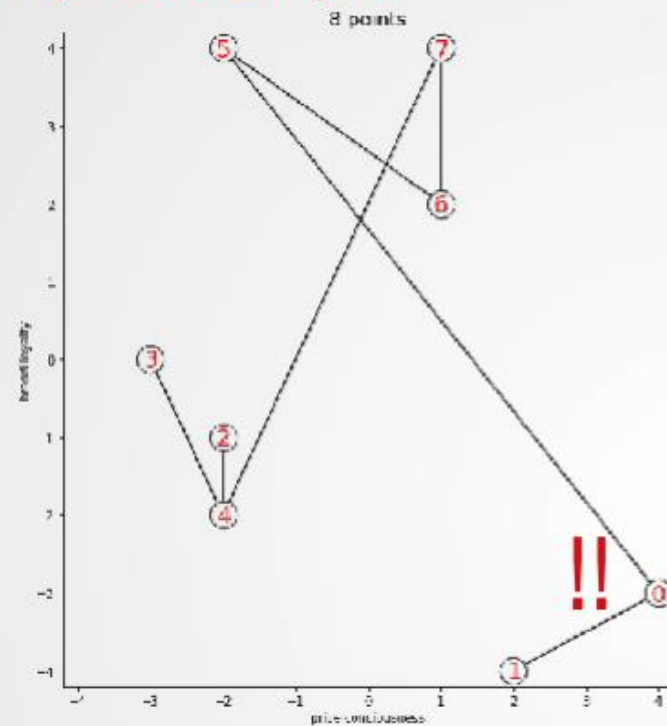
- ▣ Considers the largest (individual) distance

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

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



A simple example



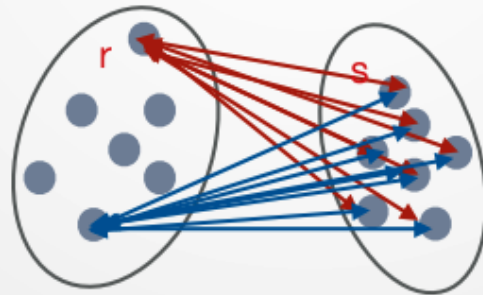
Complete linkage algorithm on squared Euclidean distance for 8 point example with dendrogram.



Average Linkage Algorithm

A compromise between nearest and farthest neighbor distance.
Average all mean distances:

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

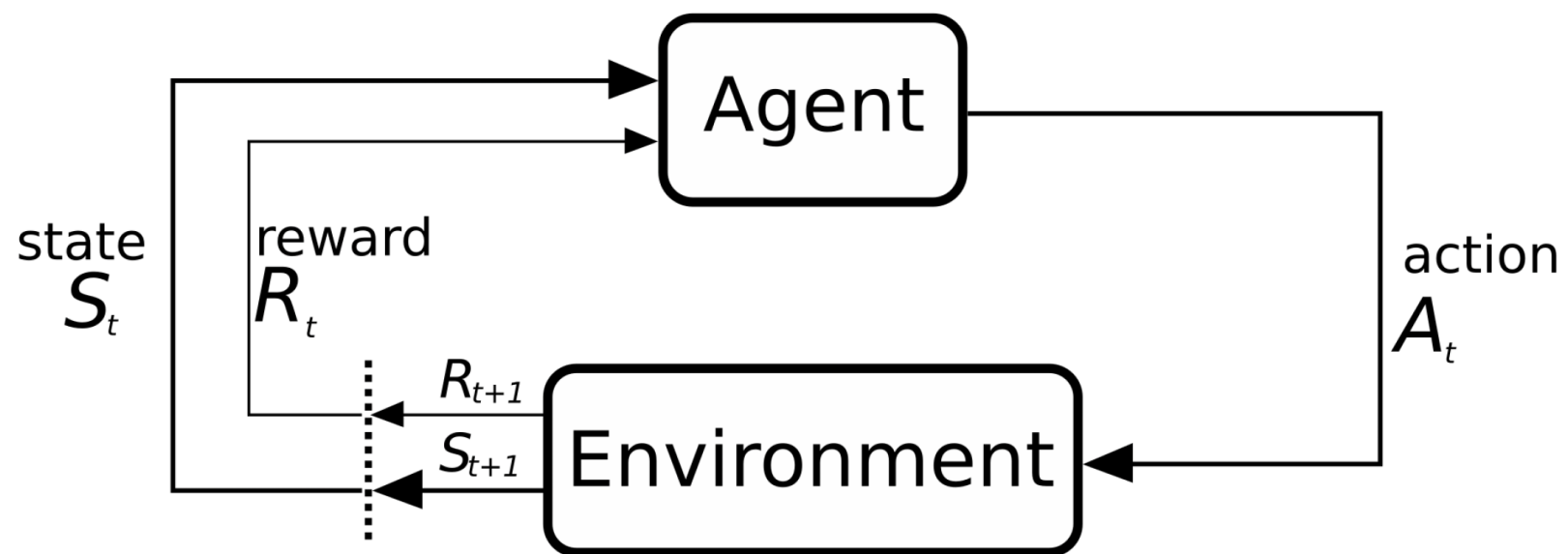


11. Reinforcement Learning: Basic

Source Material:

[Joerg Osterrieder, University of Twente](#)

[Martijn Mes, University of Twente](#)



The Idea of Value-Based RL

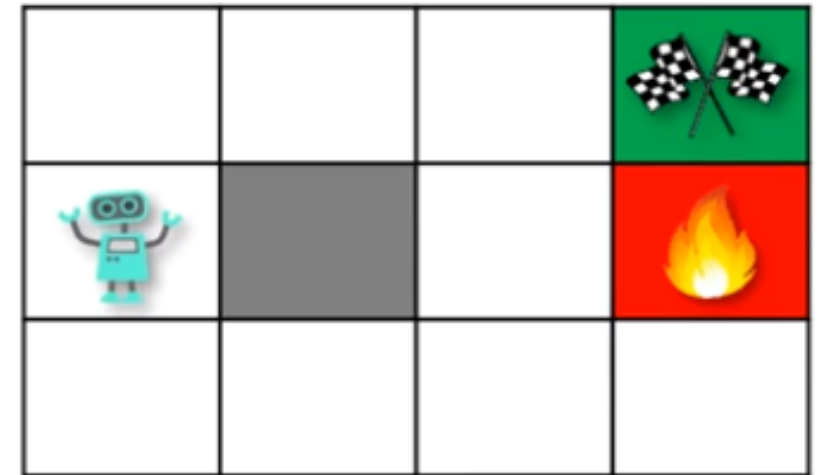
- We have to learn the consequences of our actions (rewards + reachable states):

- $V(S)$ = (discounted) (in)finite future rewards from state S onwards given an optimal policy

$Q(S,a)$ = previous + direct reward of a in S

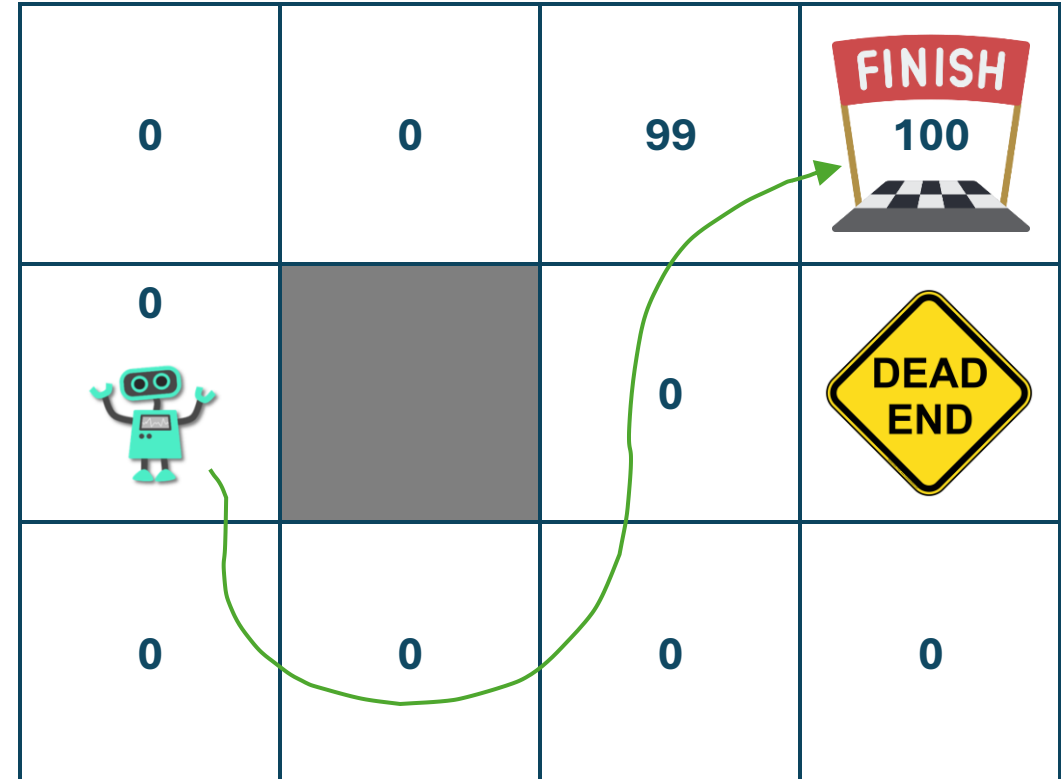
- Ways of learning:

- Look-ahead one step, take action, update $V(S)/Q(S,a) \rightarrow TD(0)$
 - 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 $S \rightarrow TD(1)$



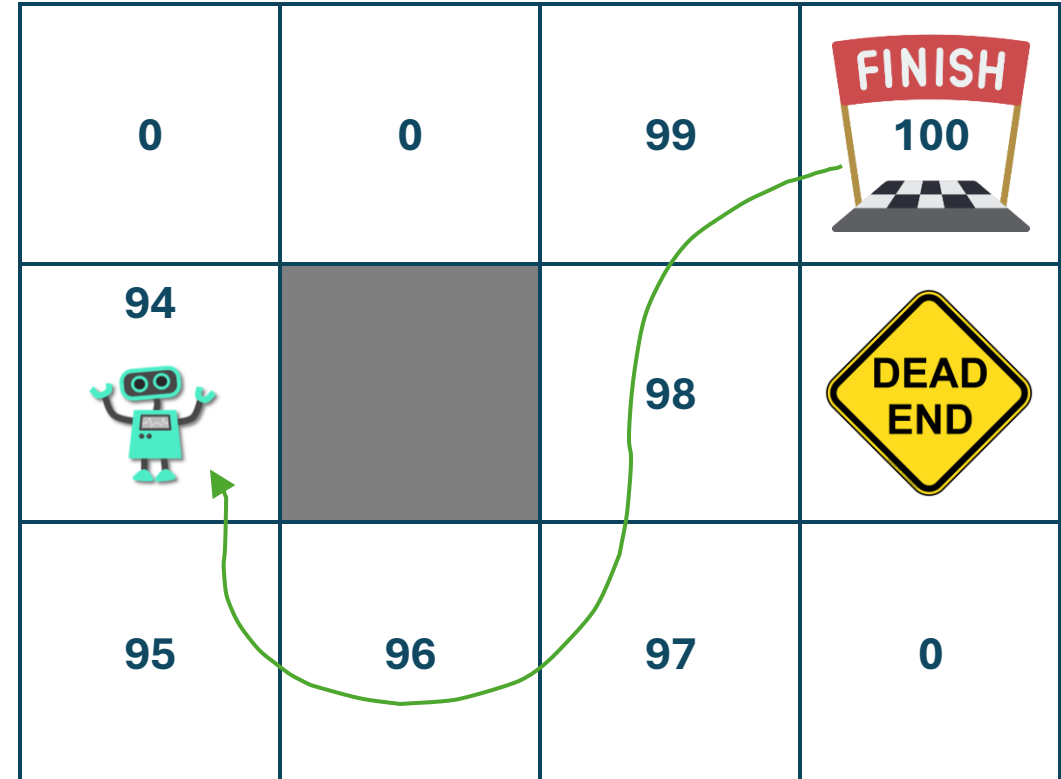
Learning the Value of States

- $V(S)$ = 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



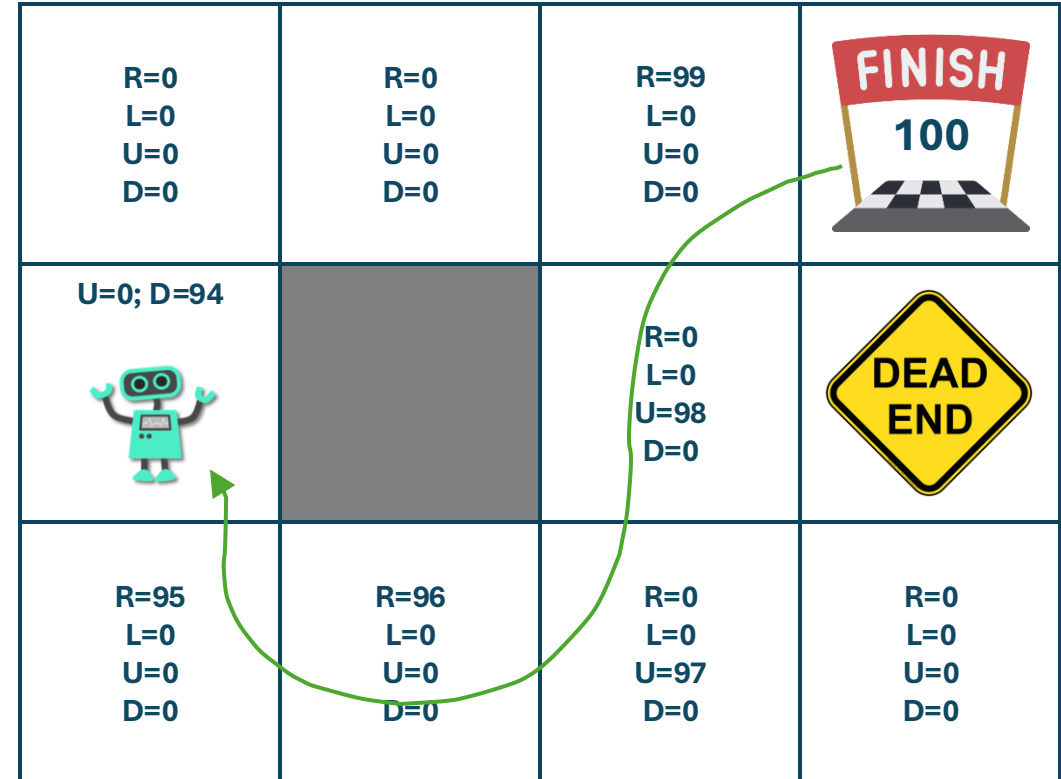
Learning the Value of States

- $V(S)$ = 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



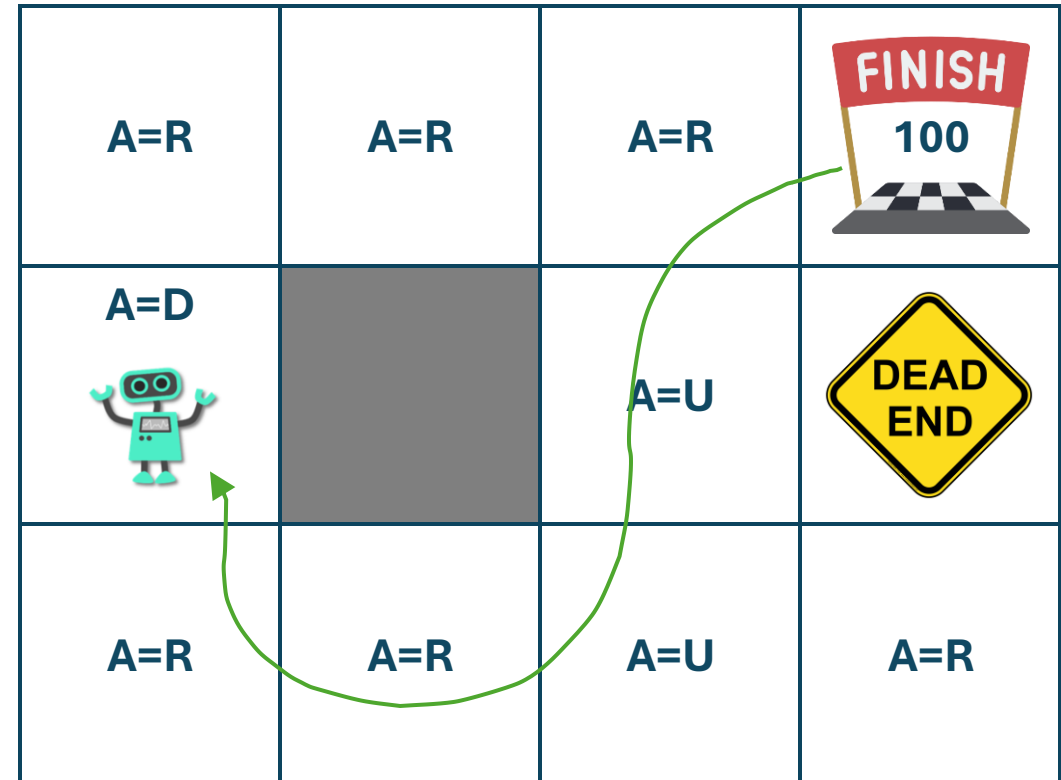
Learning the Value of State-Action Pairs

- $Q(S,a)$ = 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



Learning the Policy Directly

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



LEARNING DIMENSIONS

- **Model-free or model-based:** do we learn a model of the world, i.e., of the rewards and transition probabilities, or solely through interactions with the environment?
- **Online or offline:** do we train based on interactions with the environment (online) or on a historical dataset (offline)?
- **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 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]

- 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!

$$\pi: S \rightarrow a$$

Policy function approximation – PFA vs VFA

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

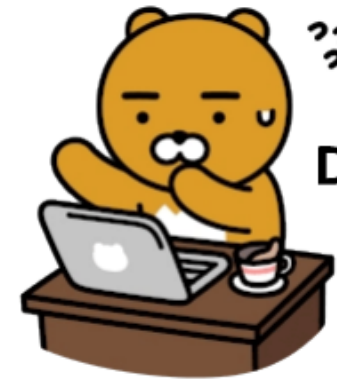
Policy Gradients



Go Right

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

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



Deep Q-Learning

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

$$a = \underset{a \in \mathcal{A}}{\operatorname{argmax}} [r(s, a) + Q(s, a)]$$

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 $\pi: s \rightarrow a$ (deterministic)
 - Now, we will use policies $\pi: s \rightarrow \mathbb{P}(a|s)$ (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



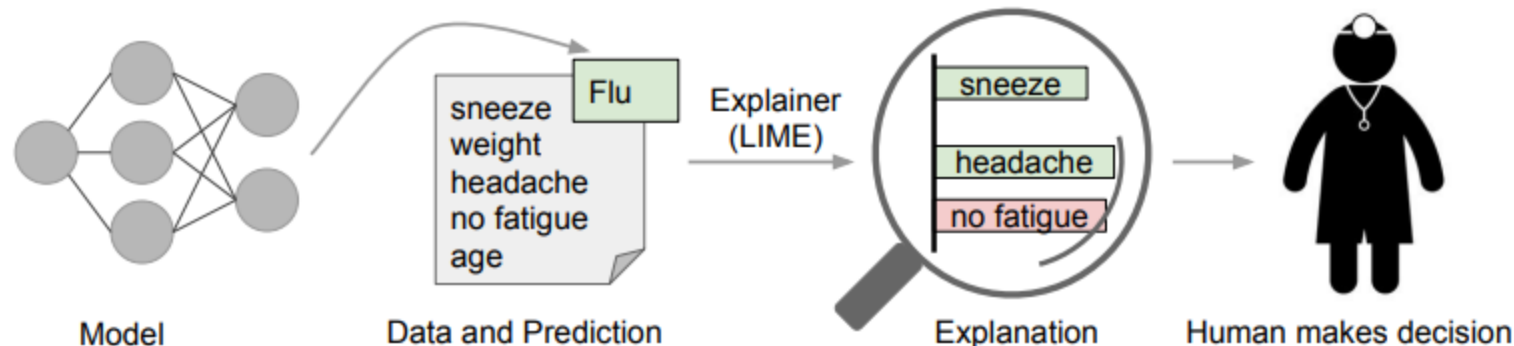
- LIME → explains the prediction of **any machine learning model** by learning an interpretable model **locally** around a specific instance of interest
- Works with classification & regression
- Works with tabular data, text and pictures

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

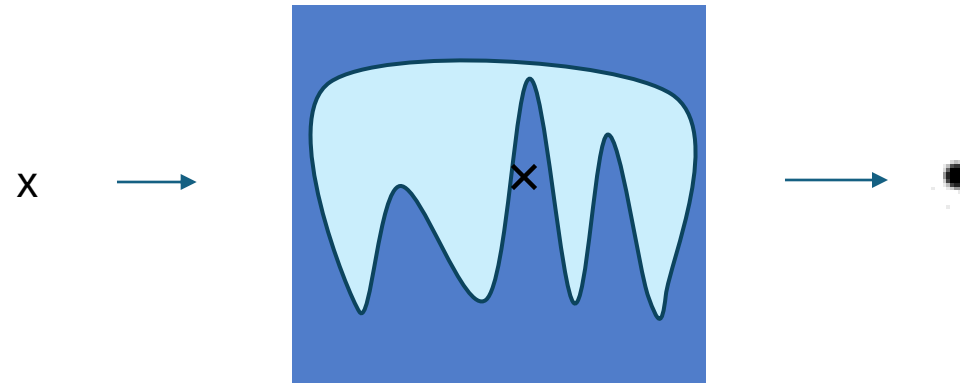
Marco Tulio Ribeiro
University of Washington
Seattle, WA 98105, USA
marcotcr@cs.uw.edu

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

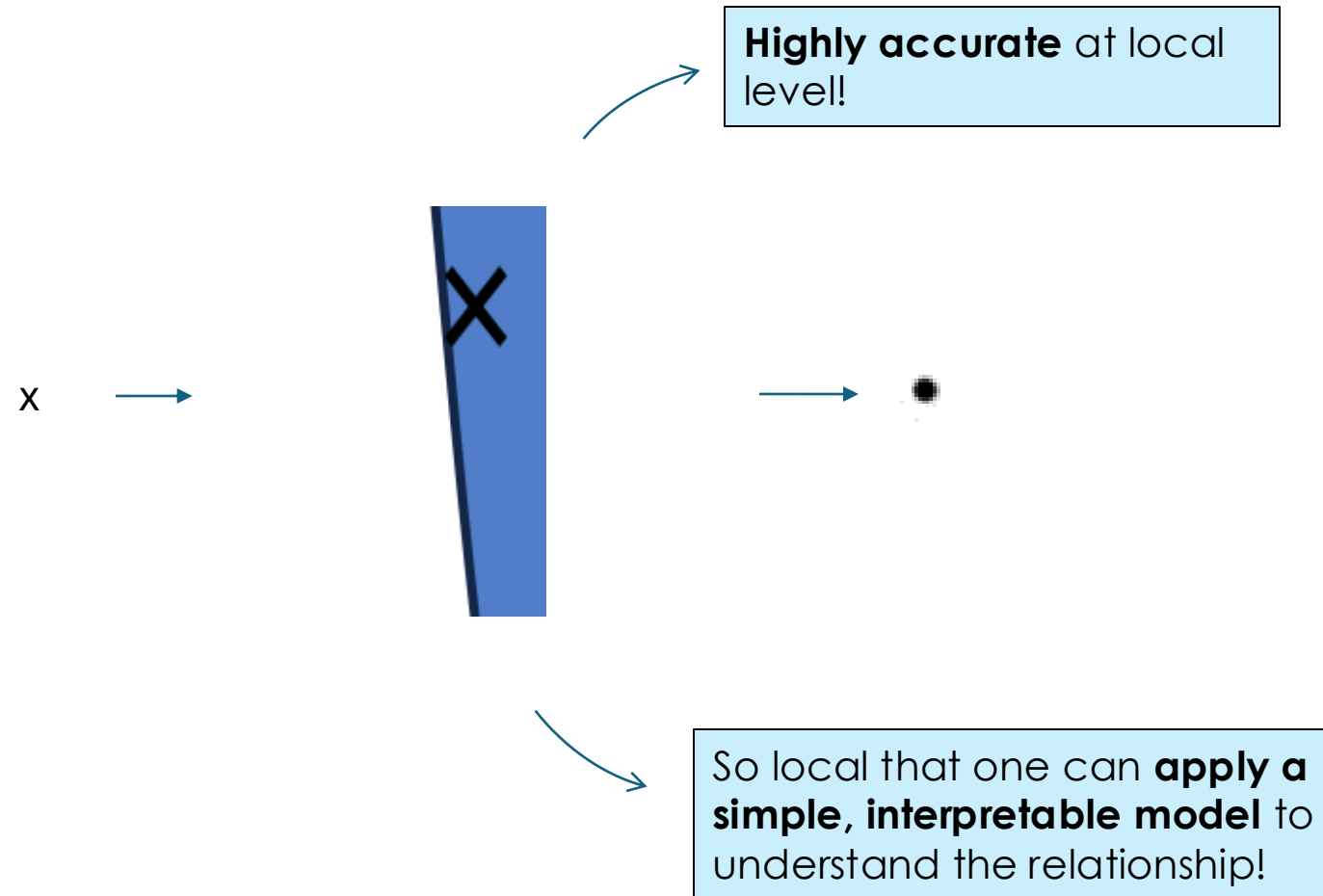
Carlos Guestrin
University of Washington
Seattle, WA 98105, USA
guestrin@cs.uw.edu




LIME – How does it work?



LIME – How does it work?



Steps



For which you require explanations

- **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**.

The Math

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

The Math

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

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

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

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

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

Coalition without feature i

The Math

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

Coalition with feature i

The Math

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

Marginal contribution of i to the coalition

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

The Math

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

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**