UCL ARTIFICIAL
INTELLIGENCE SOCIETY

TUTORIAL #5

Session 5

Ensemble Methods and Neural Networks

While we wait, sign up at **doxaai.com** for this week's challenge - more at the end!

UCL ARTIFICIAL
INTELLIGENCE SOCIETY

LECTURE OVERVIEW

01

Ensemble models

How can we make our simple models more useful?

02

Gradient descent

Recap of this incredibly important concept

03

Neural Networks

The most famous class of Machine Learning models

04

Competition

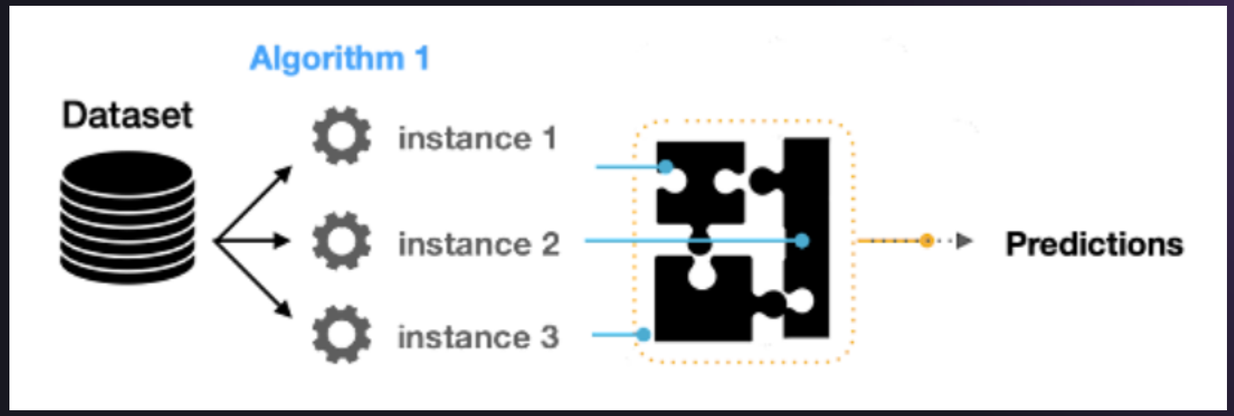
Try out everything you learned so far!

Theory

ENSEMBLE MODELS

What are Ensemble Models?

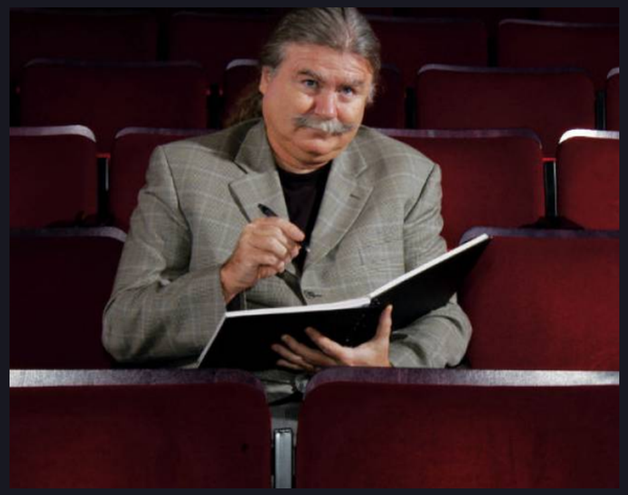
- They are a machine learning approach to c**ombine multiple other models** in the prediction process
- Singular model in EM is usually a weaker predictor than a standard supervised model



Theory

ENSEMBLE MODELS

What are Ensemble Models?



Theory

ENSEMBLE MODELS

Different approaches for Ensemble Models

Simple Ensemble Techniques

- Max Voting
- Averaging
- Weighted Averaging

Advanced Ensemble Techniques

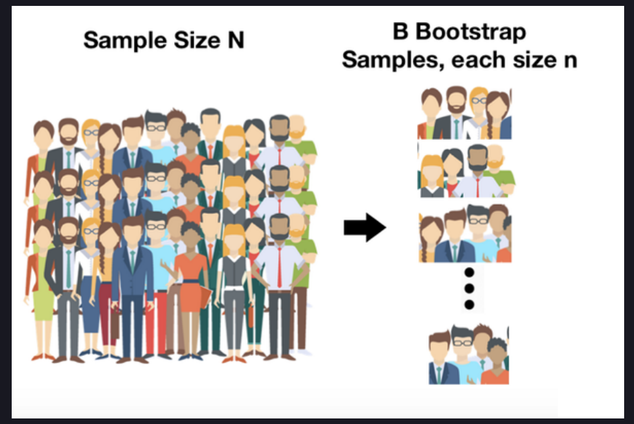
- Stacking
- Blending
- Bagging
- Boosting

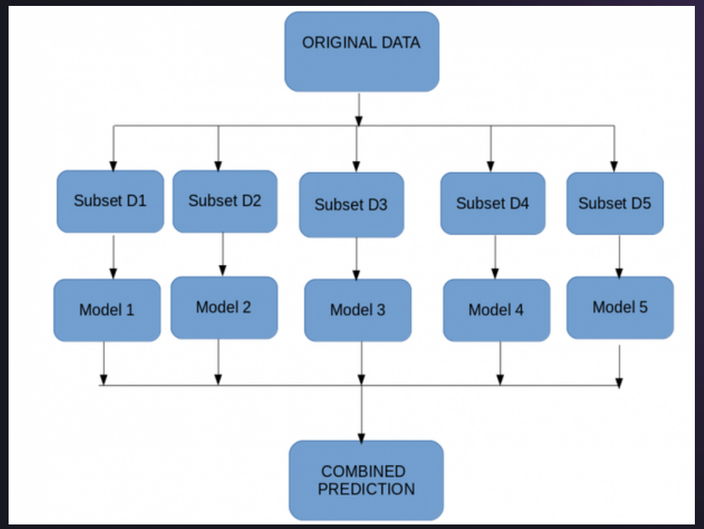
Theory

ENSEMBLE MODELS

Bagging

- Combining results of multiple models (e.g. decision trees)
- Uses the technique of **Bootstrapping**
  - Taking subsets, which overlap
- Models are trained on these subsets
- Predictions from these models are then combined



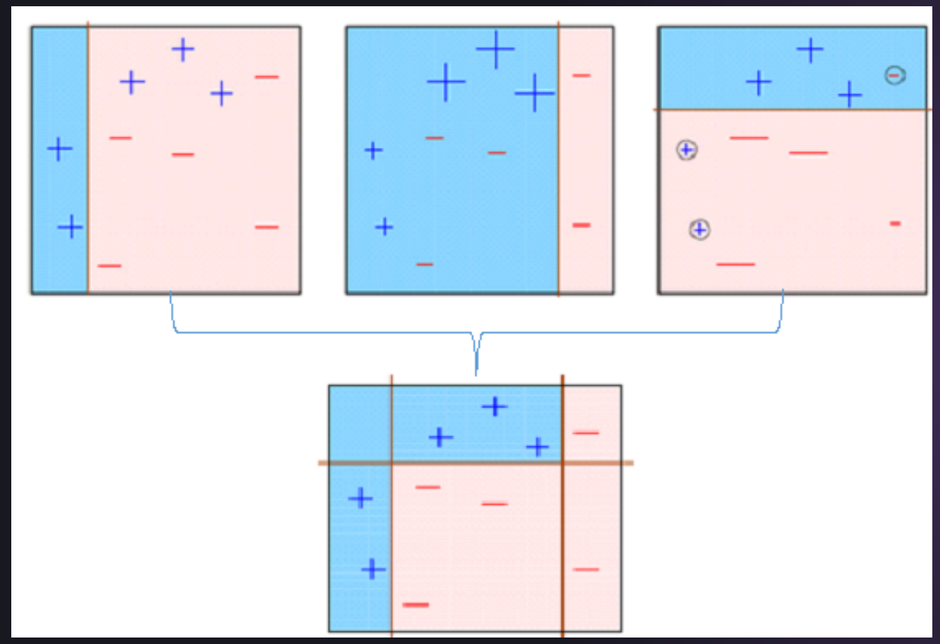


Theory

ENSEMBLE MODELS

Boosting

- Each model tries to **correct the errors of previous models**
- Initially, a **single model is trained** on a subset of the dataset
- It evaluates the whole dataset, and a n**ew subset for training a model is chosen** to counter the errors
- The final model is weighted mean of all the models



Theory

ENSEMBLE MODELS

Random Forest

- Consists of numerous decidion trees trained on bootstrapped data
- Is the primary example of bagging approach

```
from sklearn.ensemble import RandomForestRegressor

reg = RandomForestRegressor()
reg.fit(X_train, y_train)
reg.score(X_test, y_test)
```

Theory

ENSEMBLE MODELS

XGBoost

- XGBoost is a popular and efficient open-source implementation of the gradient boosted trees algorithm
- Gradient boosting is a version of boosting using Gradient Descent

```
from sklearn.ensemble import GradientBoostingRegressor

xgboost = GradientBoostingRegressor(n_estimators=100)
xgboost.fit(X_train, y_train)
xgboost.score(X_test, y_test)
```

Theory

GRADIENT DESCENT

What is Gradient Descent

- Gradient Descent is an **optimization algorithm** used for **minimizing the cost function/loss function** in various machine learning algorithms
- It is used for updating the parameters of the learning model.

Theory

GRADIENT DESCENT

Cost functions/Loss function

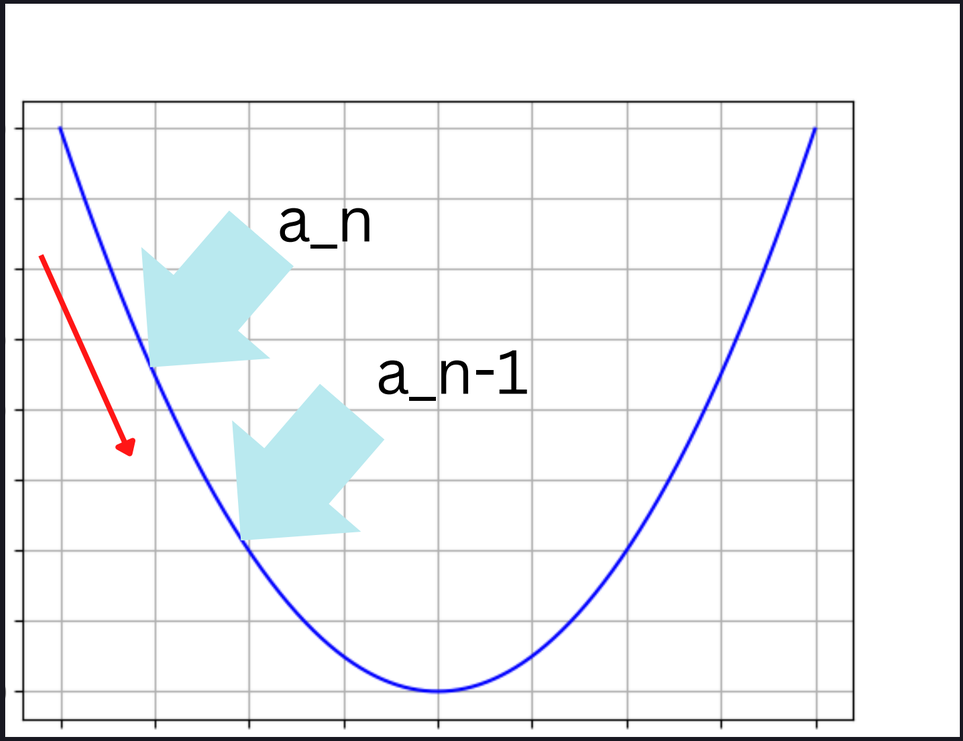
- Loss function aims to **capture the difference between the actual and predicted values** for a single record whereas cost functions aggregate the difference for the entire training dataset.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

$$\text{MAE} = \underbrace{\frac{1}{n}\sum_{i=1}}_{\text{test set}}^{n} \underbrace{|y_i}_{\text{predicted vaue}} - \underbrace{\hat{y}_i|}_{\text{actual value}}$$

Theory

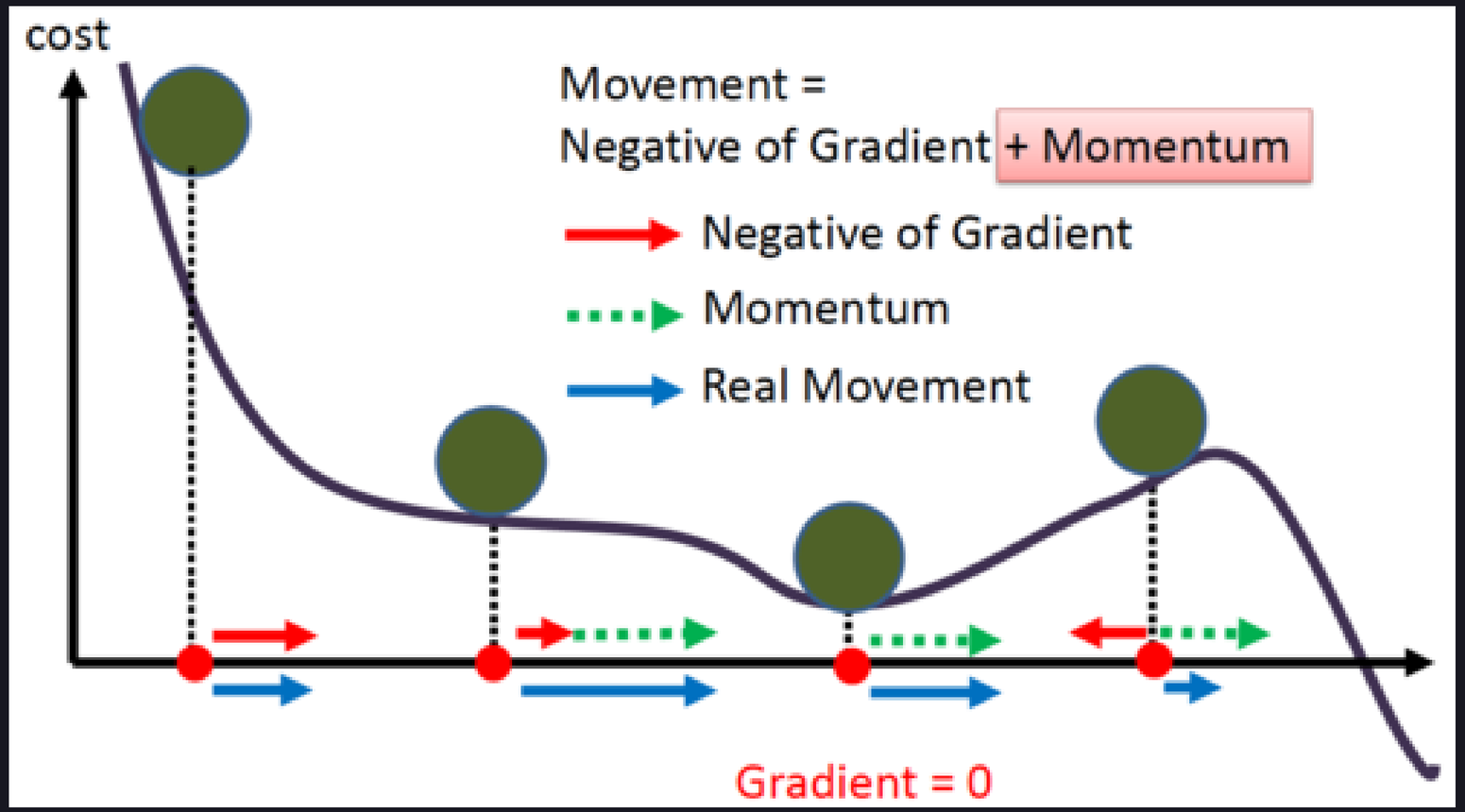
GRADIENT DESCENT



$$\mathbf{a}_{n+1} = \mathbf{a}_n - \gamma \nabla F(\mathbf{a}_n)$$

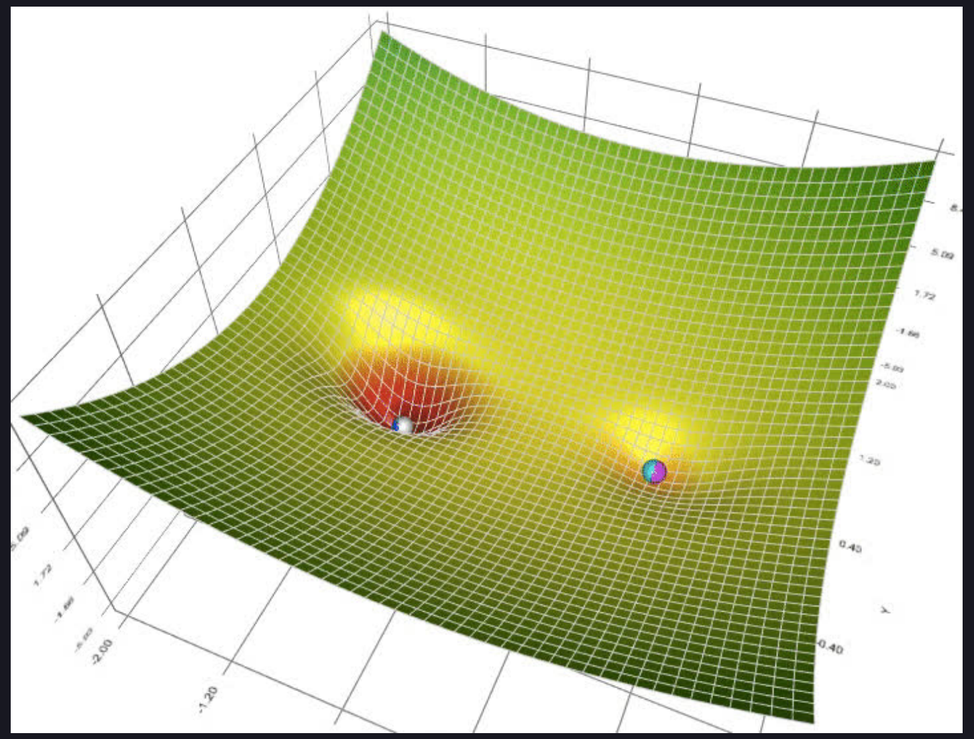
Theory

GRADIENT DESCENT



Theory

GRADIENT DESCENT



Theory

NEURAL NETWORKS

Introduction

- A method of computation based on the interaction of many connected processing elements called neurons.

- A powerful technique for solving many real-world problems.

- The ability to learn from experience in order to improve their performance.

- Ability to deal with incomplete information

Theory

NEURAL NETWORKS

Basics of Neural Networks

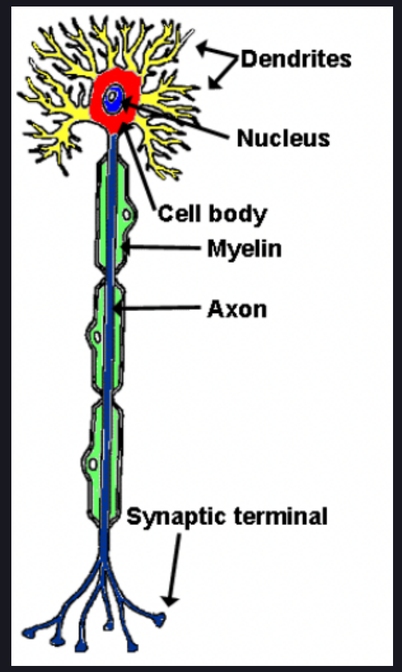
- Biological approach to AI
- Developed in 1943
- Composed of one or more layers of neurons
- Several types, we'll focus on feed-forward and feedback networks

Theory

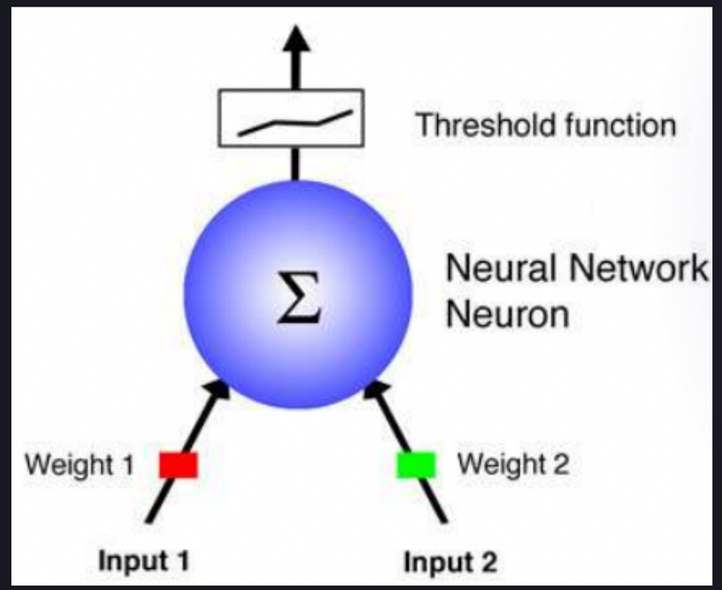
NEURAL NETWORKS

Neurons

Biological



Artificial

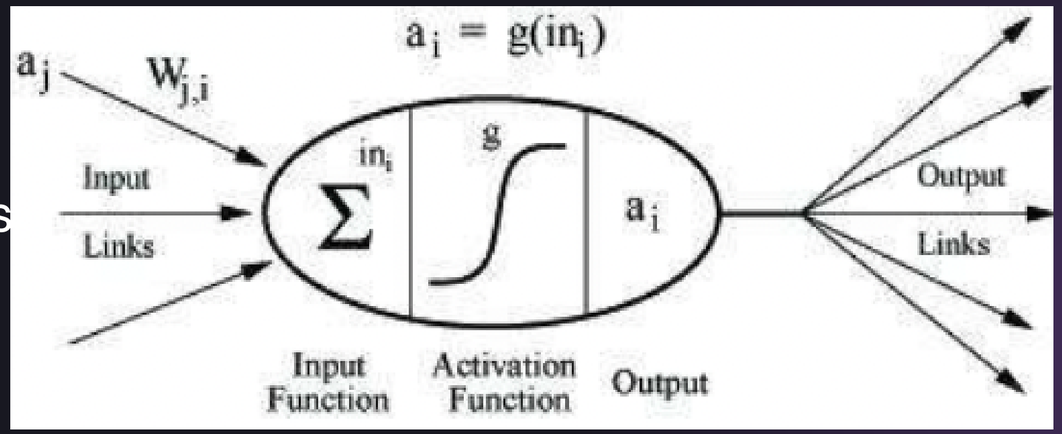


Theory

NEURAL NETWORKS

Artificial Neurons

- Receive n-inputs
- Multiply each input by a weight
- Apply an **activation function** to the sum of results
- Output a result



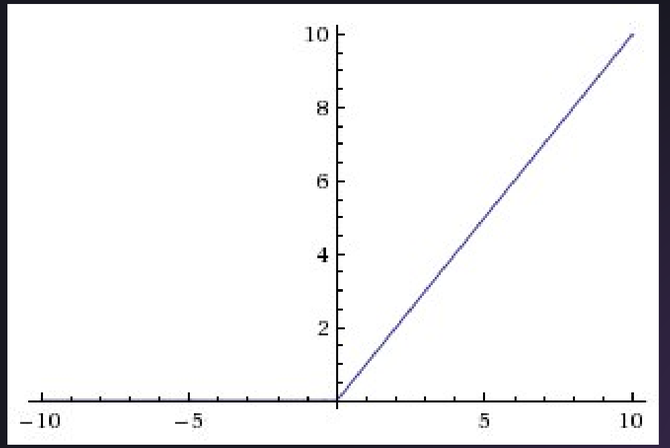
Theory

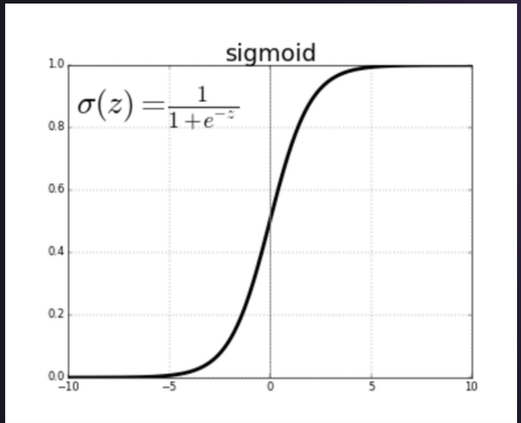
NEURAL NETWORKS

Activation Functions

- Transform the output of neurons
- Control 'active' or 'inactive' a unit is
- Allow non-linear relationships to be modelled

- **Rectified linear unit**: ReLU = max(0, x)
- **Sigmoid function**: σ = 1 / (1 + exp(−x))





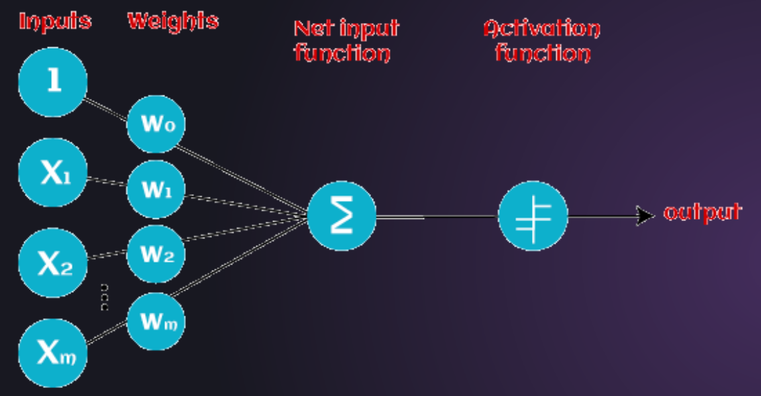
Theory

NEURAL NETWORKS

Perceptron

Frank Rosenblatt introduced the perceptron model, which contains three main components:

- **Input nodes** (forming the **input layer**): accepts the initial data into the system for further processing
- **Weights and biases**: represents the strength of the connection between units
- **An activation function**



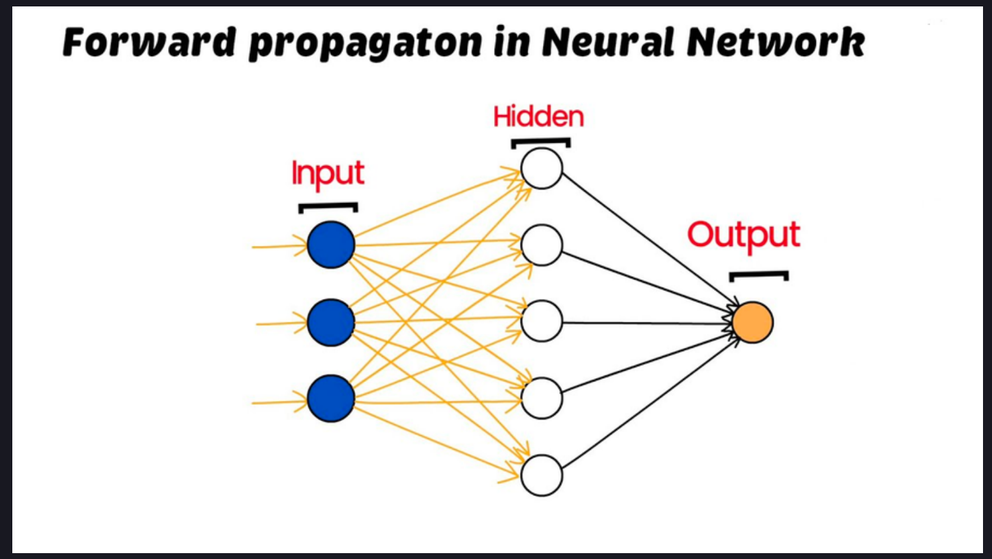
This later inspired the development of the **multi-layer perceptron** model and feedforward neural networks more broadly.

Theory

NEURAL NETWORKS

Forward propagation

- The input data is fed in the forward direction through the network
- Hidden layer accepts the input data, processes it as per the activation function
- Output is passed to the successive layer

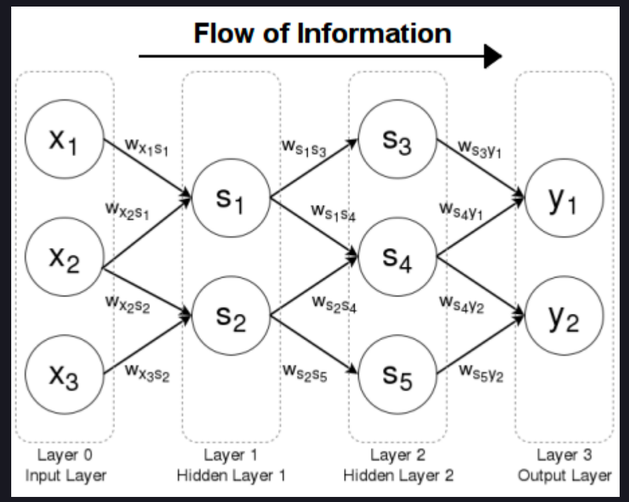


Theory

NEURAL NETWORKS

Feed-forward neural networks

- To generate output, the input data is fed in through the forward direction only.
- The data should not flow in reverse direction - it would form a cycle and the output could never be generated
- Such network configurations are known as feed-forward networks.



Theory

NEURAL NETWORKS

Feed-forward neural networks

The Two Steps:

- **Pre-activation**: it is a weighted sum of inputs, i.e. the linear transformation of weights w.r.t to the inputs available. Based on this aggregated sum and activation function, the neuron makes a decision as to how much information to pass further or not.

- **Activation**: the calculated weighted sum of inputs is passed to the activation function. An activation function is a mathematical function which introduces non-linearities into the network. There are four commonly used and popular activation functions: sigmoid, hyperbolic tangent (tanh, sometimes pronounced 'tanch'), ReLU and softmax.

Theory

NEURAL NETWORKS

```
# define the training procedure
# i.e. one step of gradient descent
# there are lots of steps
# so we encapsulate it in a function
# Note: inputs and labels are torch tensors
def train(model, loss, optimizer, inputs, labels):
  # https://discuss.pytorch.org/t/why-is-it-recommended-to-wrap-your-data-with-variable
  inputs = Variable(inputs, requires_grad=False)
  labels = Variable(labels, requires_grad=False)

  # Reset gradient
  # https://discuss.pytorch.org/t/why-do-we-need-to-set-the-gradients-manually-to-zero-
  optimizer.zero_grad()

  # Forward
  logits = model.forward(inputs)
  output = loss.forward(logits, labels)

  # Backward
  output.backward()

  # Update parameters
  optimizer.step()
```

Theory

NEURAL NETWORKS

Loss calculation

Mean Squared Error

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

Cross-entropy

$$\text{loss} = -\sum_{i \in C} y_i \log(\hat{y}_i)$$

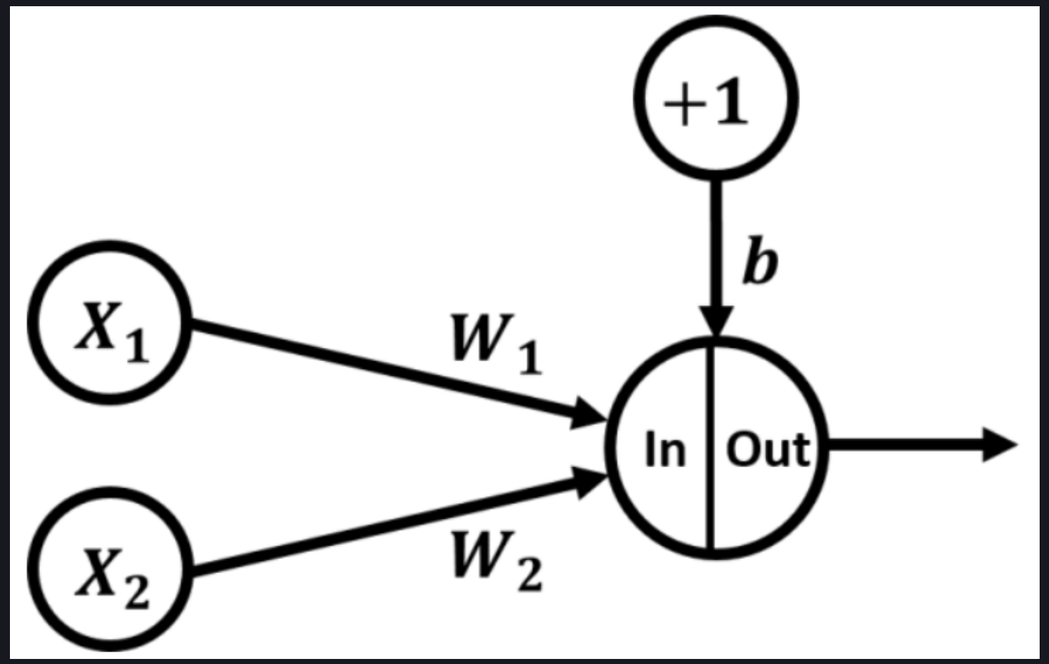
Binary cross-entropy

$$\text{loss} = -y_1 \log(\hat{y}_1) - (1 - y_1)\log(1 - \hat{y}_1)$$

Theory

NEURAL NETWORKS

Backpropagation



- Taking the loss derivative with respect to each weight and bias using the chain rule
- Updating weights (tempered by a **learning rate**)

$$\frac{\partial \text{loss}}{\partial w_1} = \frac{\partial \text{loss}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w_1}$$

$$w_{1_{\text{new}}} = w_1 - \text{LR} \frac{\partial \text{loss}}{\partial w_1}$$

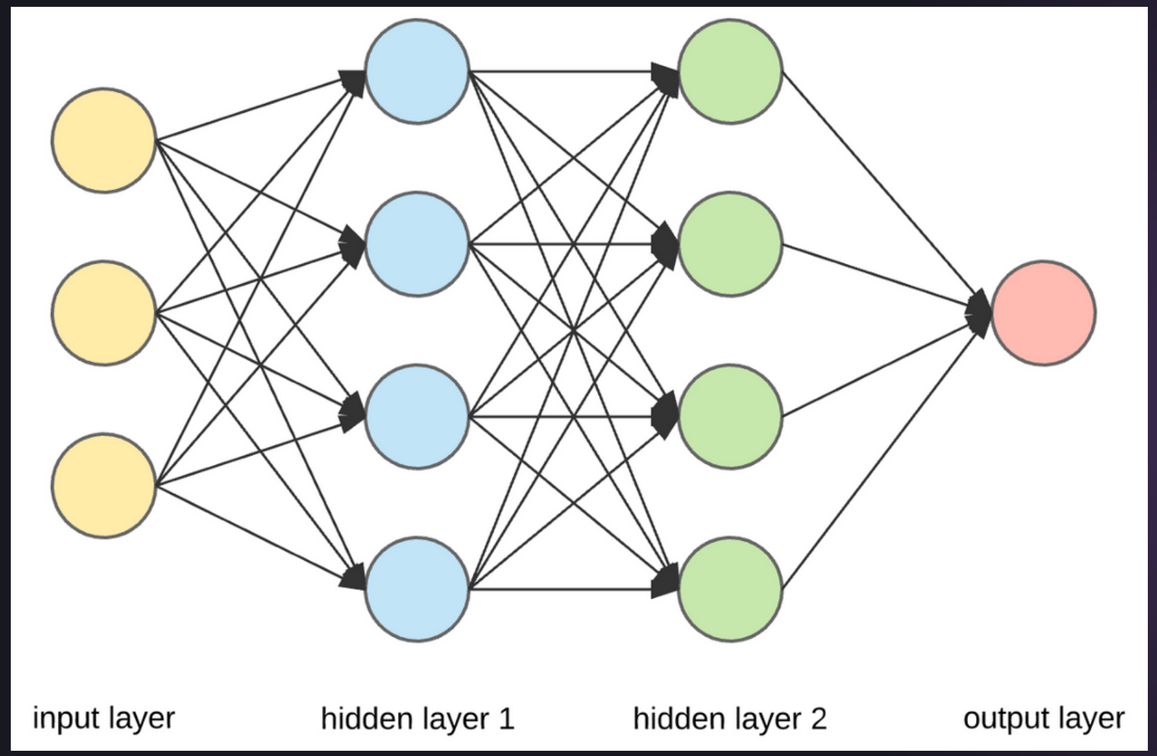
Theory

NEURAL NETWORKS

Multiple layers

- The output from one perceptron becomes the input of another perceptron in the next layer along

- This allows for more complex problems to be tackled and forms the basis of **deep learning**



IT'S TIME FOR YOUR FIRST CHALLENGE

Putting it all into practice

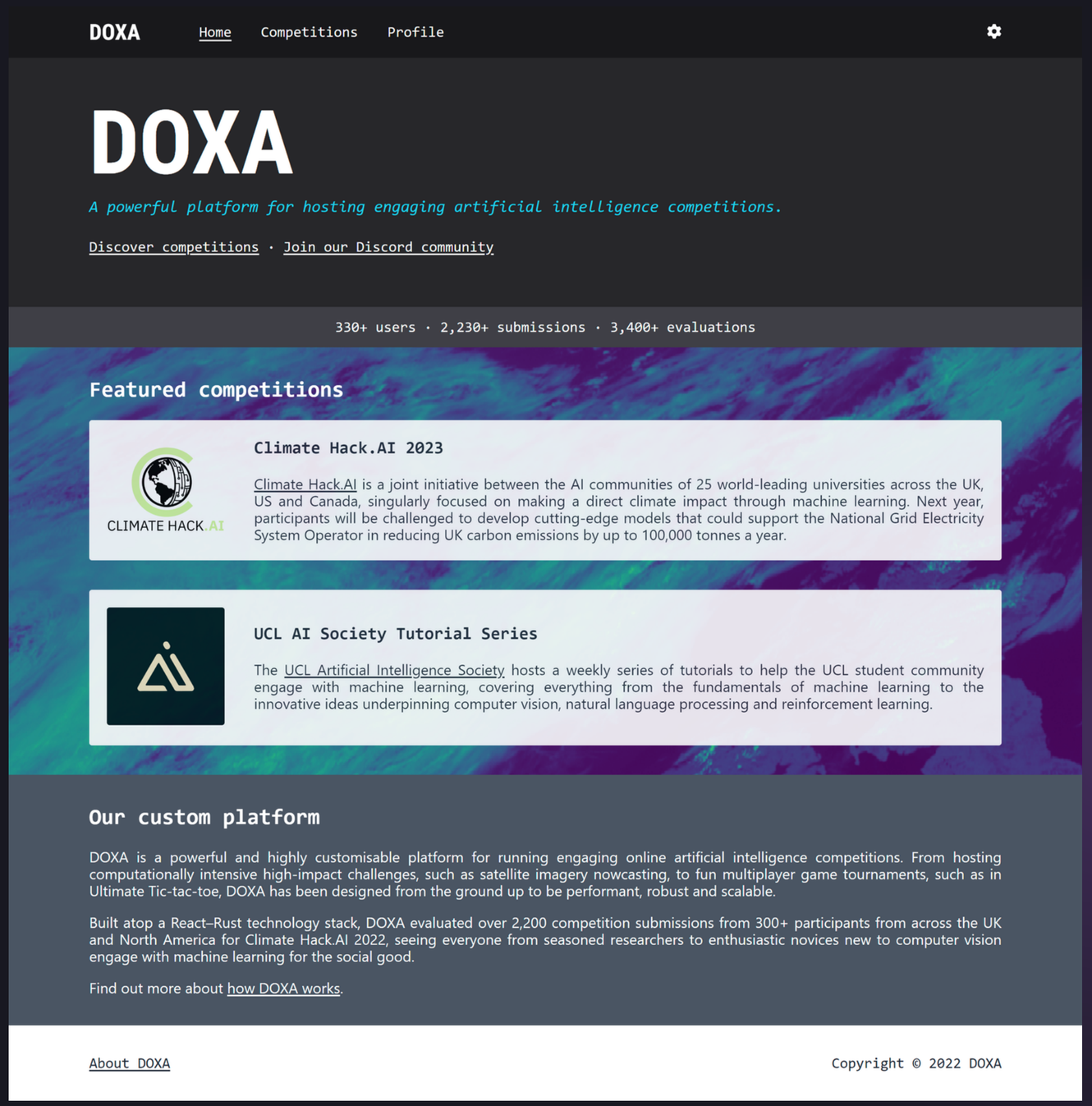
DOXA

A platform for hosting engaging artificial intelligence competitions

We are thrilled to launch the first UCLAIS tutorial series challenge on

doxaai.com

This is a great opportunity to put what you have learned into practice!



UCLAIS DOXA Challenge 1

BRAIN STROKE PREDICTION

Area of Application: Disease Diagnosis

Challenge

Predict whether or not someone has suffered a brain stroke using the machine learning techniques you have learned over the past few tutorial lectures

Training Set: 4500 entries with 10 input features and one binary output label

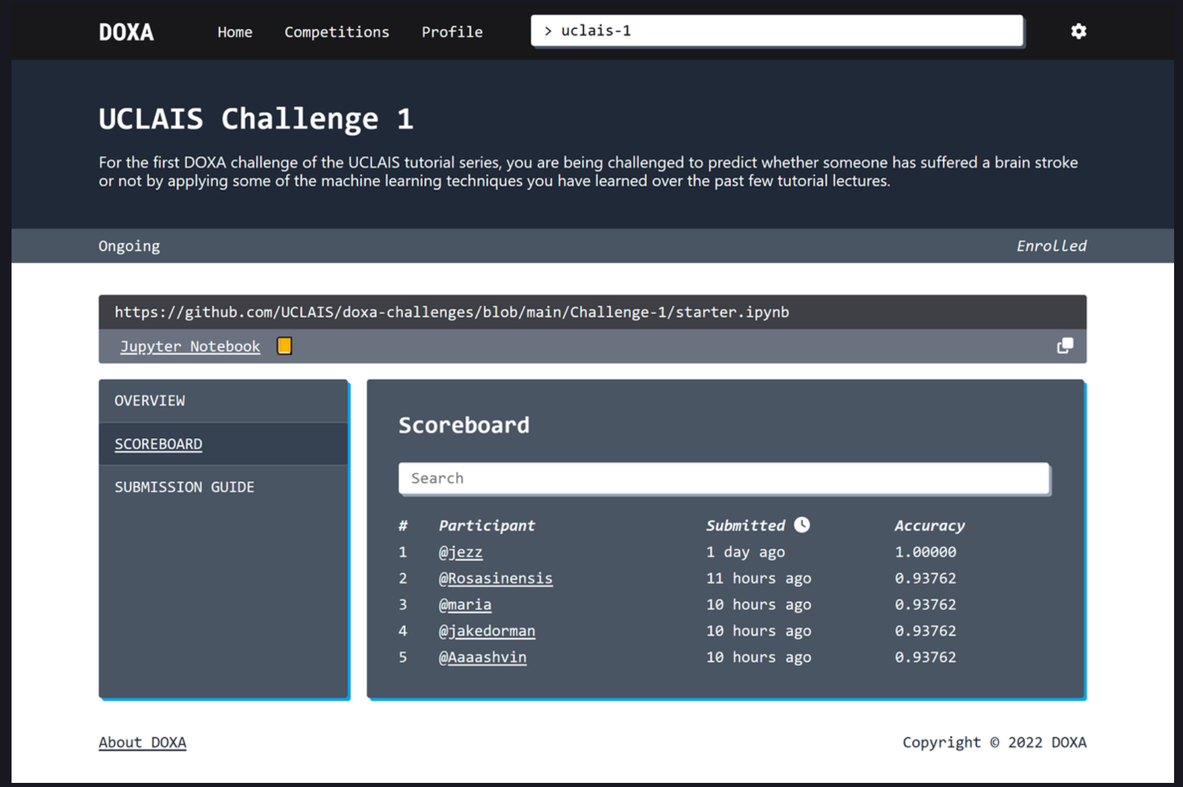
Test Set: 481 entries containing only the 10 input features without the output label

FEATURES

Gender, age, hypertension, heart disease, BMI, marital status, work type, etc...

HOW TO TAKE PART

1. Sign up for an account on **doxaai.com**
2. Enrol for the competition at **https://doxaai.com/competition/uclais-1**
3. Open the Jupyter notebook in Google Colab (**https://colab.research.google.com/**)
4. Run all the cells (making sure to approve access for the DOXA CLI submission tool)
5. Find yourself on the DOXA scoreboard!
6. Improve on the model you were provided with using some of the ideas you have learned over the past few weeks.



DOXA: https://doxaai.com/
GitHub: https://github.com/UCLAIS/doxa-challenges/tree/main/Challenge-1
Notebook: https://github.com/UCLAIS/doxa-challenges/blob/main/Challenge-1/starter.ipynb

A DEMO 

QUESTIONS?

SEE YOU NEXT TIME