# Machine Learning Algorithm Foundations - Day 2

## Topics

- Establish Term Glossary
- The Artificial Neuron
- Open-Source Software
  - The layers of auto-ml
- Rudimentary Components of ML Algorithms
- Walkthrough w/ Real Data

## Goals

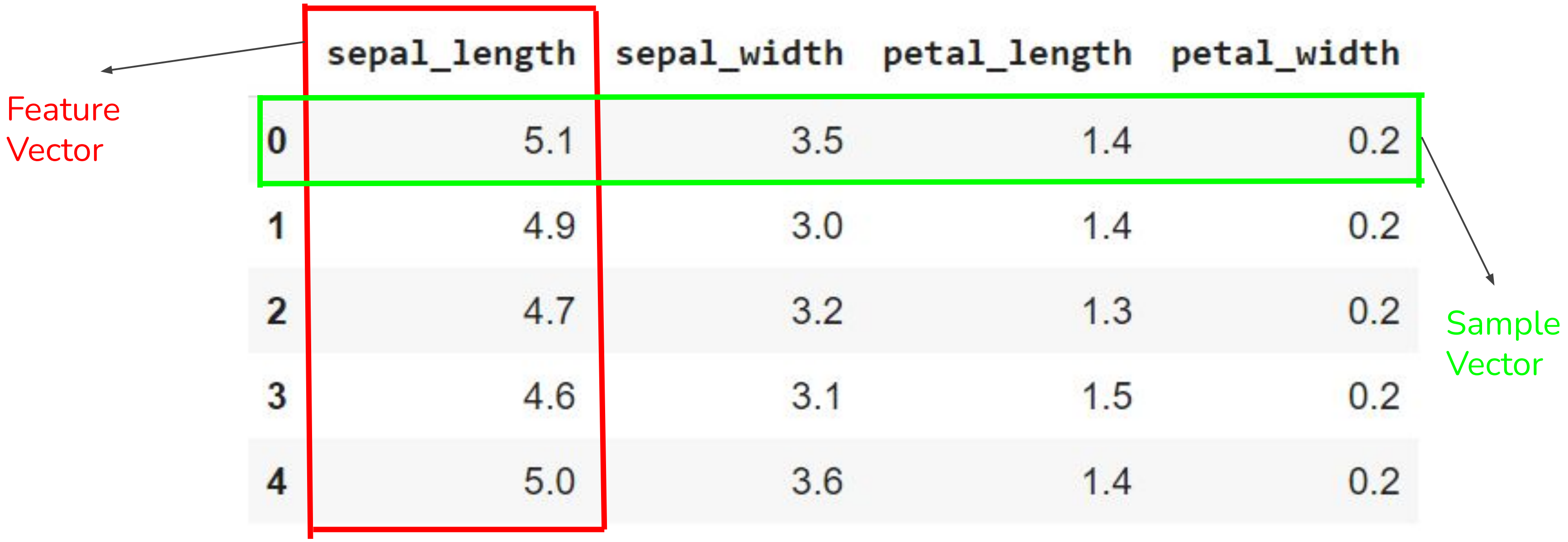
- Become familiar with common ML lingo
- Distinguish biological intelligence vs artificial intelligence
- Understand auto-ml and distinguish between different types
- Become familiar with the ML development process
- Understand the components of a dataset
- Gain an understanding of the components of ML algorithms

## Important Lingo

## Term Glossary

- **Feature:** A column in our dataset
- **Sample:** A row in our dataset
- **Vector:** a single row or column of data
  - vec = [1,2,3,4,5] (1, 5)
- **Matrix:** multiple rows or columns of data
  - mat = [[1,2,3], [1,2,3]] (2, 3)
- **Parameters:** the trainable parameters of our algorithm (W, b)
- **Hyper-Parameters:** the chosen parameters of our algorithm (units, features)
- **Loss:** function used to approximate the algorithms “incorrectness”
- **Activation:** function used to decide if a neuron will “fire” or not
- **Epoch:** How many iterations over the dataset during model training

## Examples

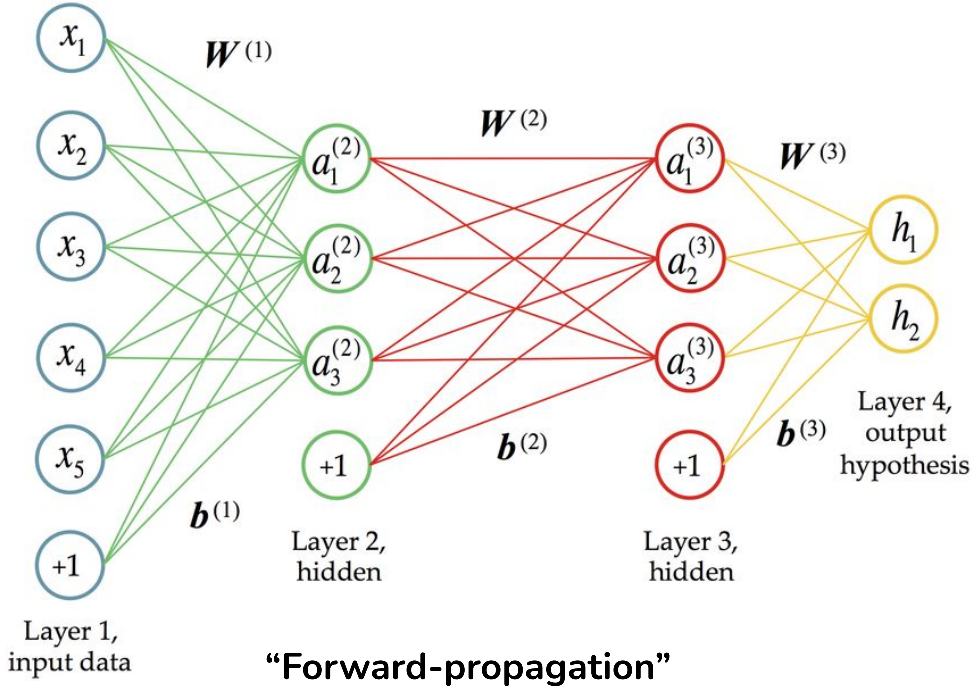


|   | sepal_length | sepal_width | petal_length | petal_width |
|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 |

Feature Vector
Sample Vector

Data Matrix of shape (samples, features), in this case (5,4)

## More Examples



**"Forward-propagation"**

**Parameters:**
- W, b
- These are not controlled by a human, <u>the network learns the parameters itself</u>

Multi-layer neural net, the predictions from one layer are the inputs to the next.

**Hyper-Parameters:**
- a (activation function)
- Num. of hidden layers
- Num. of neurons in each hidden layer
- <u>Human controlled params of the network</u>

## The Artificial Neuron

## Biological Intelligence vs AI

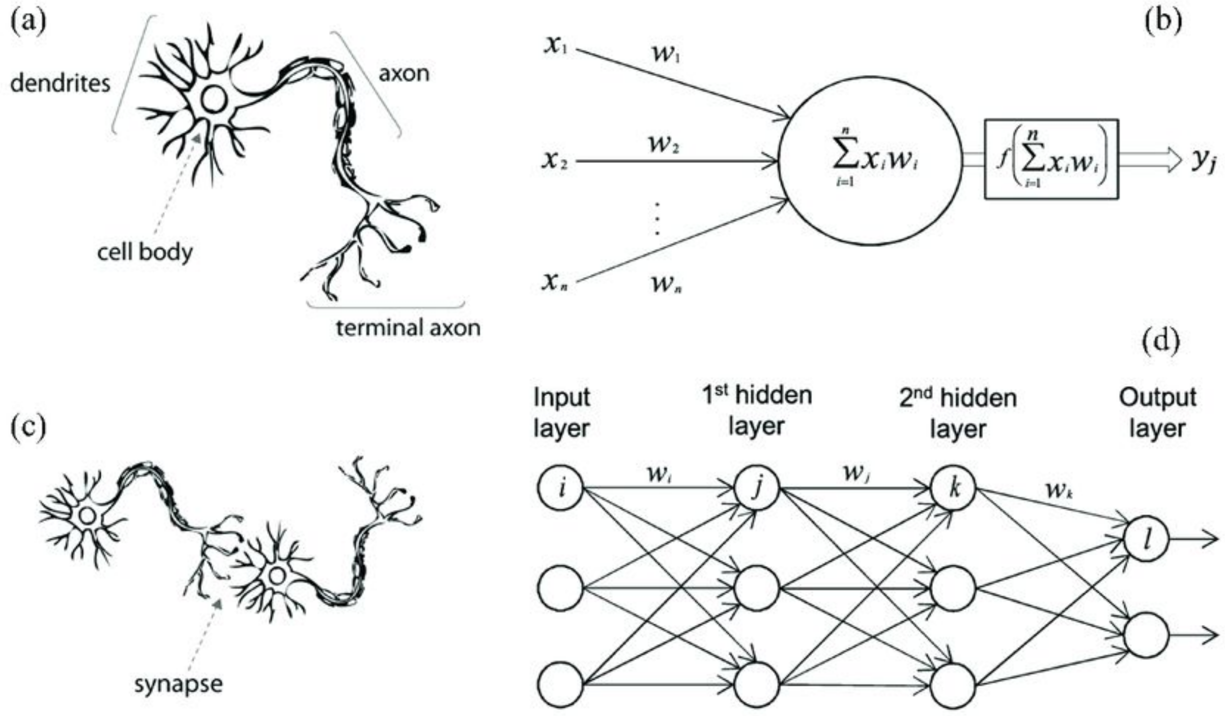


Figure (a): biological neuron

Figure (b): artificial neuron

Figure (c): biological neural-net

Figure (d): artificial neural-net

## Comparison

- Figure A: **Biological neuron.** Dendrites take input, pass it on to the cell body (nucleus), and the axon carries the output.
- Figure B: **Artificial neuron.** [x1, x2, x3…] are the inputs, they get passed into the node, which outputs a prediction.
- Figure C: **Biological neural network.** It depicts the connection, called a synapse, between two biological neurons.
- Figure D: **Artificial neural network, commonly referred to as 'ANN'.** This depicts the connections of artificial neurons. The lines, denoted as W(i), referred to as 'weight-vectors' or 'parameters', are the ANN equivalent of synapses.

## Fun Facts

- Human neural network— “Number of neurons ~ 86 Billion, number of synapses ~ 150 Trillion, another generalization: average number of synapses per neuron ~ 1,744.”
- Specialized artificial neural network — “GPT-3’s full version has a capacity of 175 billion machine learning parameters.”
- (side note — GPT3 is one of the most powerful AIs in the world.)
- Number of Brain Synapses: 150 trillion
- Number of parameters in specialized AIs : ~150 billion
- 150 trillion/150 billion= 1,000. In other words — the most powerful AIs would have to scale by a factor of one thousand for their connections to be on par with the brain…

## Thoughts

- However, it is not the size capacity of AI models that limits their abilities. It is our understanding of the brain which limits the scope of modern machine learning systems. Humans don’t possess a deep enough understanding of the brain to replicate it dependably and completely.
- One of the most renowned computer scientists of our generation and often revered as the ‘father’ of deep learning, Geoffrey Hinton, has many quotes with this same sentiment. “The brain has about ten thousand parameters for every second of experience. We do not have much experience about how systems like that work or how to make them so good at finding structure in data.”

## Open-Source

## The Layers of Auto-ML

- **Layer 1:** You code your own algorithms from scratch. (C++/Python)
- **Layer 2:** You use high-level open-source APIs. (Keras/Sci-Kit Learn)
- **Layer 3:** You use a pre-trained model from a high-level API.
- **Layer 4:** Auto-ML platform. No human intervention. (Alteryx/Data Robot)

## Layer 1 - From Scratch

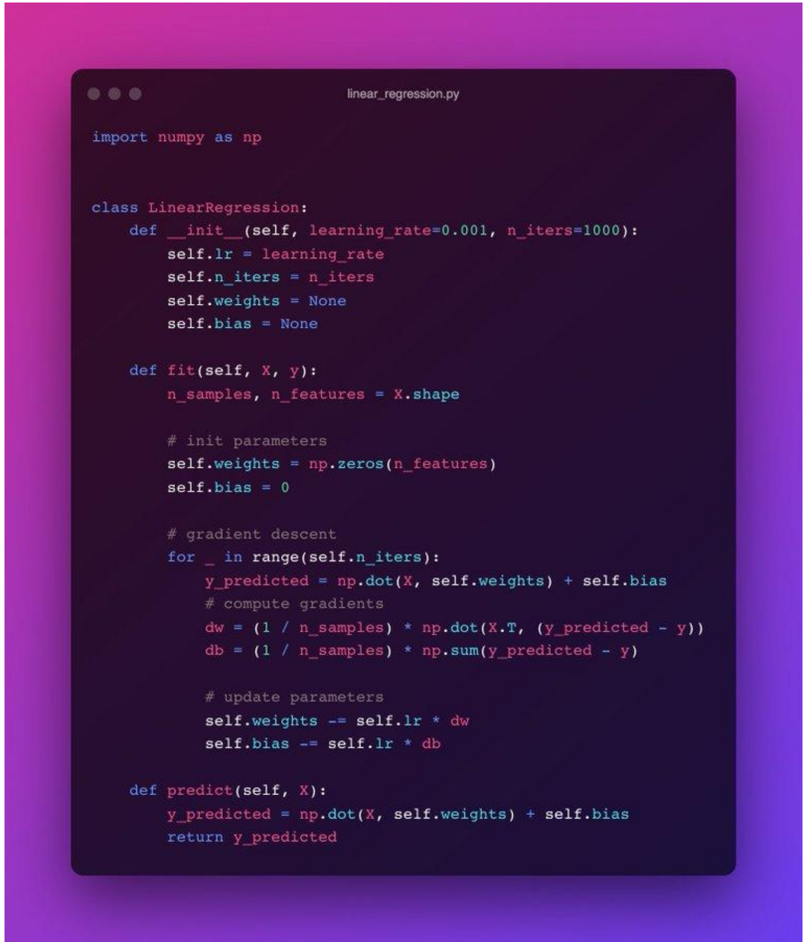
$$\hat{y} = \beta_i X_i + \dots \beta_n X_n + \beta_0$$

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

$$\frac{dL}{d\beta_i} = \frac{\alpha}{n} \sum_{i}^{n} (y_i - \hat{y}_i) * X_i$$

$$\frac{dL}{d\beta_0} = \frac{\alpha}{n} \sum_{i}^{n} (y_i - \hat{y}_i)$$

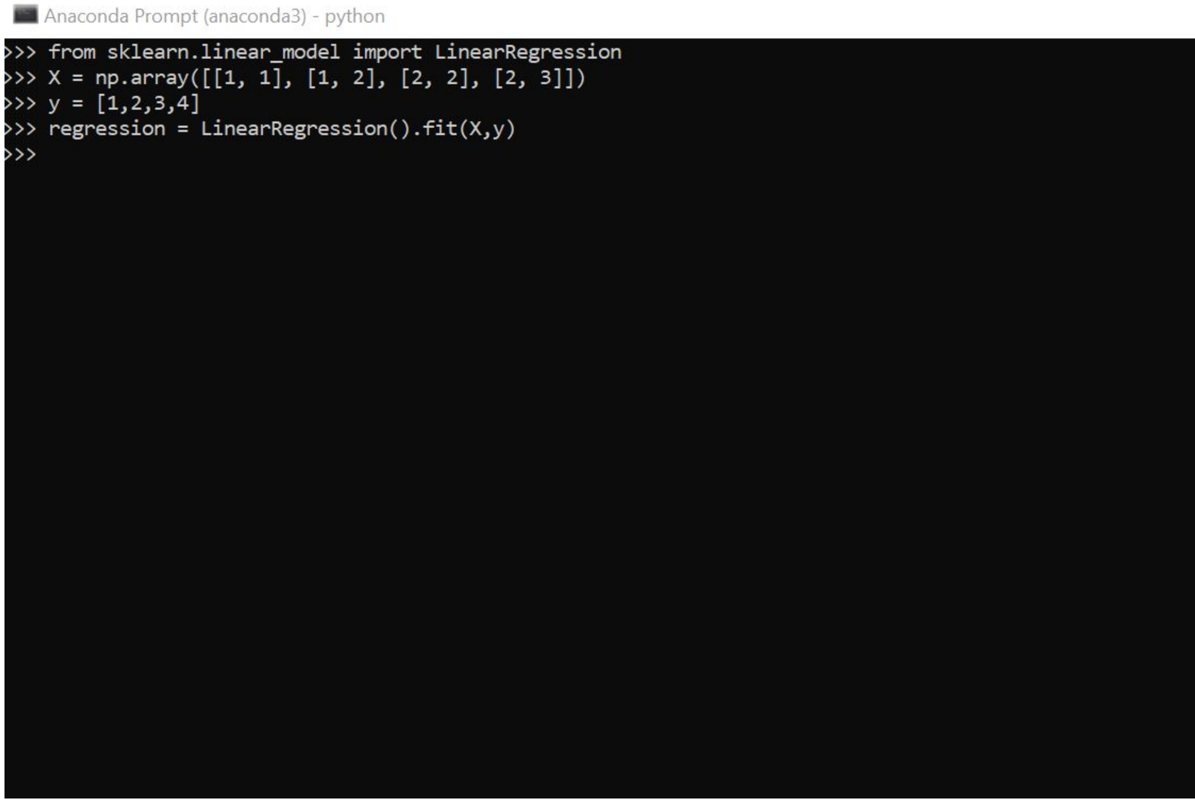
Linear Regression from scratch


linear_regression.py

```python
import numpy as np


class LinearRegression:
    def __init__(self, learning_rate=0.001, n_iters=1000):
        self.lr = learning_rate
        self.n_iters = n_iters
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        n_samples, n_features = X.shape

        # init parameters
        self.weights = np.zeros(n_features)
        self.bias = 0

        # gradient descent
        for _ in range(self.n_iters):
            y_predicted = np.dot(X, self.weights) + self.bias
            # compute gradients
            dw = (1 / n_samples) * np.dot(X.T, (y_predicted - y))
            db = (1 / n_samples) * np.sum(y_predicted - y)

            # update parameters
            self.weights -= self.lr * dw
            self.bias -= self.lr * db

    def predict(self, X):
        y_predicted = np.dot(X, self.weights) + self.bias
        return y_predicted
```

## Layer 2 - API

In just two lines of code, we can accomplish everything in the previous slide.



```
>>> from sklearn.linear_model import LinearRegression
>>> X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
>>> y = [1,2,3,4]
>>> regression = LinearRegression().fit(X,y)
>>>
```

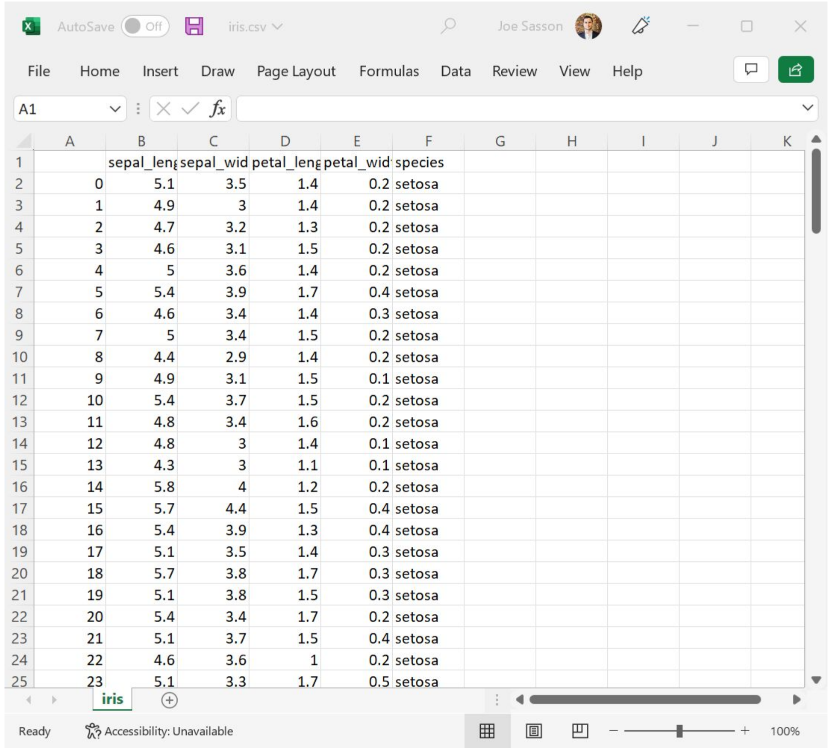
## Layer 3 - Pretrained Model

```
model = ocr_predictor(det_arch='db_resnet50', reco_arch='crnn_vgg16_bn', pretrained=True)
```

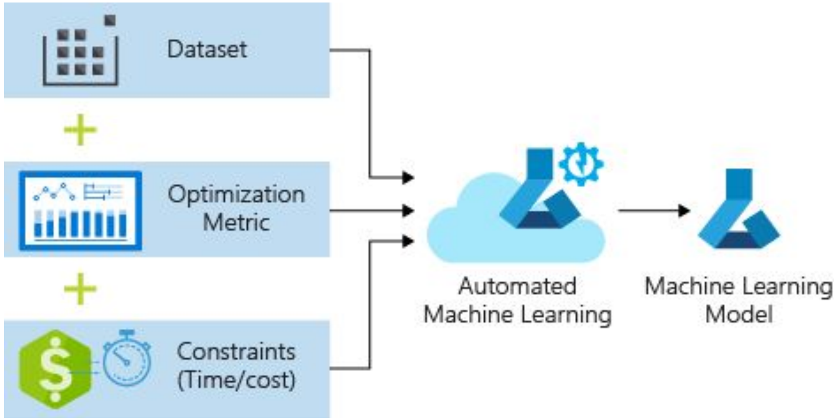
Here we are actually using a high-level API (keras), but we are loading in pre-trained models.

Large organizations & research institutions open-source huge models, trained on hundreds of thousands of TBs of data. We can then utilize these models as is, or fine-tune them for our specific task.

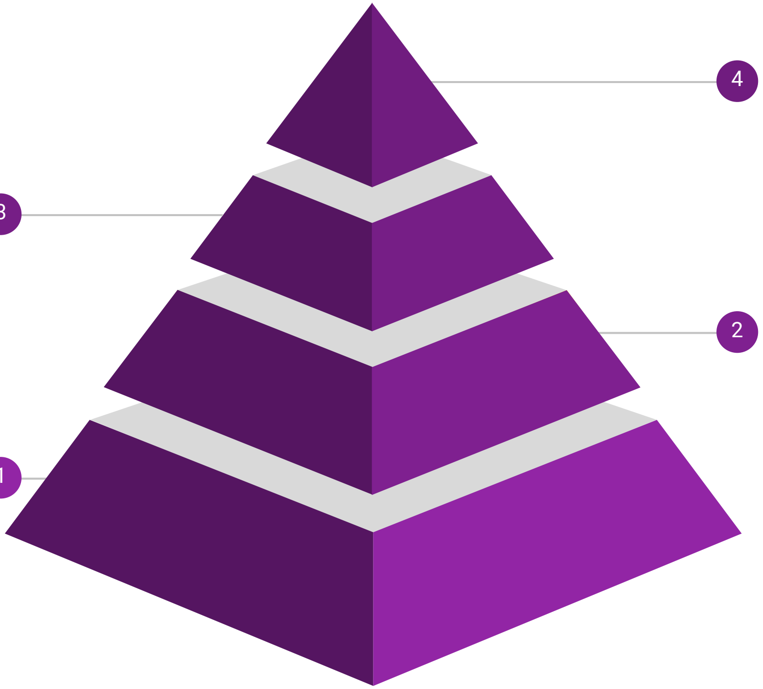
# Layer 4 - Auto ML



| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | | sepal_leng | sepal_wid | petal_leng | petal_wid | species |
| 2 | 0 | 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 3 | 1 | 4.9 | 3 | 1.4 | 0.2 | setosa |
| 4 | 2 | 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 5 | 3 | 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| 6 | 4 | 5 | 3.6 | 1.4 | 0.2 | setosa |
| 7 | 5 | 5.4 | 3.9 | 1.7 | 0.4 | setosa |
| 8 | 6 | 4.6 | 3.4 | 1.4 | 0.3 | setosa |
| 9 | 7 | 5 | 3.4 | 1.5 | 0.2 | setosa |
| 10 | 8 | 4.4 | 2.9 | 1.4 | 0.2 | setosa |
| 11 | 9 | 4.9 | 3.1 | 1.5 | 0.1 | setosa |
| 12 | 10 | 5.4 | 3.7 | 1.5 | 0.2 | setosa |
| 13 | 11 | 4.8 | 3.4 | 1.6 | 0.2 | setosa |
| 14 | 12 | 4.8 | 3 | 1.4 | 0.1 | setosa |
| 15 | 13 | 4.3 | 3 | 1.1 | 0.1 | setosa |
| 16 | 14 | 5.8 | 4 | 1.2 | 0.2 | setosa |
| 17 | 15 | 5.7 | 4.4 | 1.5 | 0.4 | setosa |
| 18 | 16 | 5.4 | 3.9 | 1.3 | 0.4 | setosa |
| 19 | 17 | 5.1 | 3.5 | 1.4 | 0.3 | setosa |
| 20 | 18 | 5.7 | 3.8 | 1.7 | 0.3 | setosa |
| 21 | 19 | 5.1 | 3.8 | 1.5 | 0.3 | setosa |
| 22 | 20 | 5.4 | 3.4 | 1.7 | 0.2 | setosa |
| 23 | 21 | 5.1 | 3.7 | 1.5 | 0.4 | setosa |
| 24 | 22 | 4.6 | 3.6 | 1 | 0.2 | setosa |
| 25 | 23 | 5.1 | 3.3 | 1.7 | 0.5 | setosa |



## Visualization



**Layer 4**

Here we have fully automated machine learning via a MLaaS provider.

**Layer 3**

Here we have pretrained machine learning models via API.

**Layer 2**

Here we have high-level APIs such as Sci-Kit Learn, Keras, and PyTorch. We can utilize this code to custom build a model.

**Layer 1**

Here we have raw code and computational mathematics.

## Pros & Cons

| | **Layer 1 - Scratch** | **Layer 2 - API** | **Layer 3 - Pretrained Model** | **Layer 4 - Automated ML** |
|---|---|---|---|---|
| Pros | Total customization over the entire network parameters and hyper-parameters | No need to worry about equations, it has all been abstracted for you in the most optimal way | You don't even have to train the model!<br><br>SOTA accuracy | No-code machine learning. Simply drag & drop your data |
| Cons | You have to be exceptional at math & coding<br><br>Probably not as optimized as open-source libraries | Less customization over the network and how it trains | Not trained on your specific dataset, the pretrained solution may or may not work! | Zero control over the network and hyper-params |

## Components of ML Algorithms

## Key Components

- **Input Data** (X)
    - The dataset we are using to train the model
    - In the form of a matrix, with shape, (samples, features)
- **Weights and Biases** (W, b)
    - The trainable parameters of our network, which are initialized randomly
- **Activation Function** (sigmoid, tanh, relu)
    - The function used to decide the output of a layer
- **Loss Function (**Crossentropy, MSE)
    - The function used to assess the models error

## Key Components Continued

- **Linear Algebra**
  - Matrices, Vectors
  - Operations surrounding these things are part of linear algebra
  - Example: Computing the weighted sum of X and W
- **Statistics**
  - Loss, Activation functions
  - Operations surrounding output probabilities are part of statistics
  - Example: Computing output probabilities
- **Calculus**
  - Optimization
  - The math enabling ML algorithms to learn is part of calculus
  - Example: Minimizing our loss function

## Inputs → Outputs

**Combine Inputs & Weights, add bias term**

Z = Xi*Wi + b

**Put the weighted sum (Z), through an activation function**

y_pred = activation(Z)

**Calculate Loss**

L = sum((y-y_pred)^2)

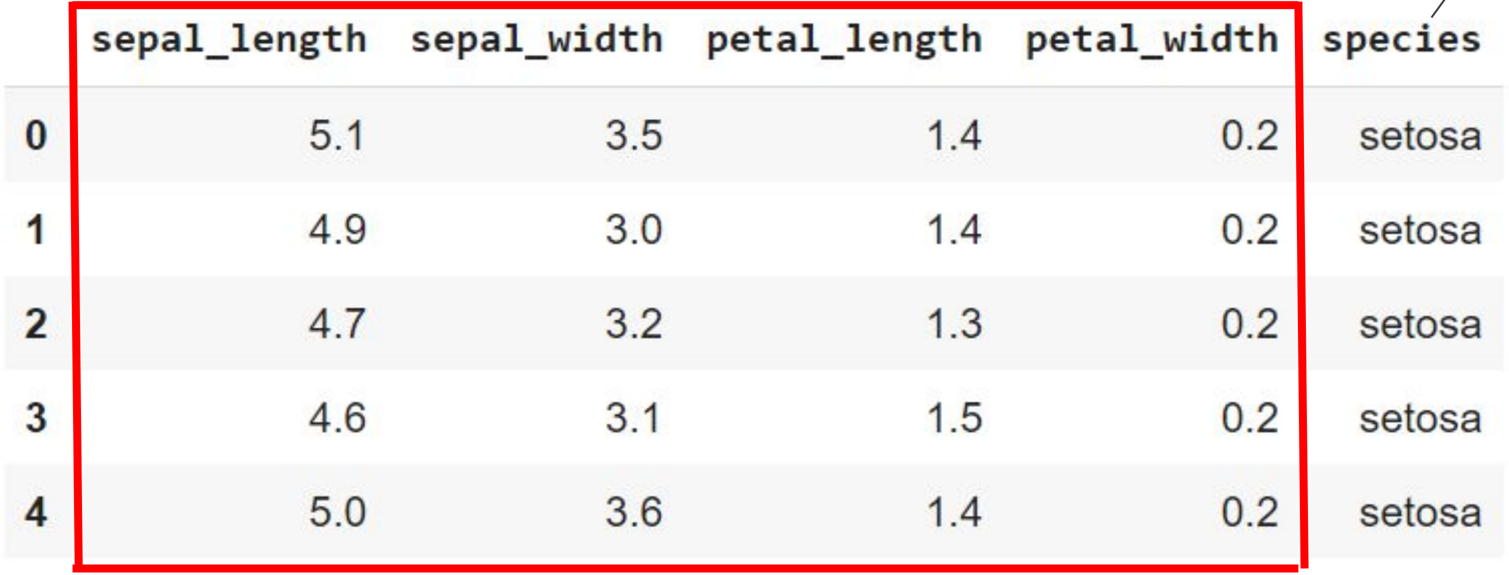
**Calculate Gradients**

dL/dW = sum((y-y_pred))*X

## Walkthrough

## Visualize Data

This is the target (labels)



| | sepal_length | sepal_width | petal_length | petal_width | species |
|---|---|---|---|---|---|
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | setosa |

These are the features

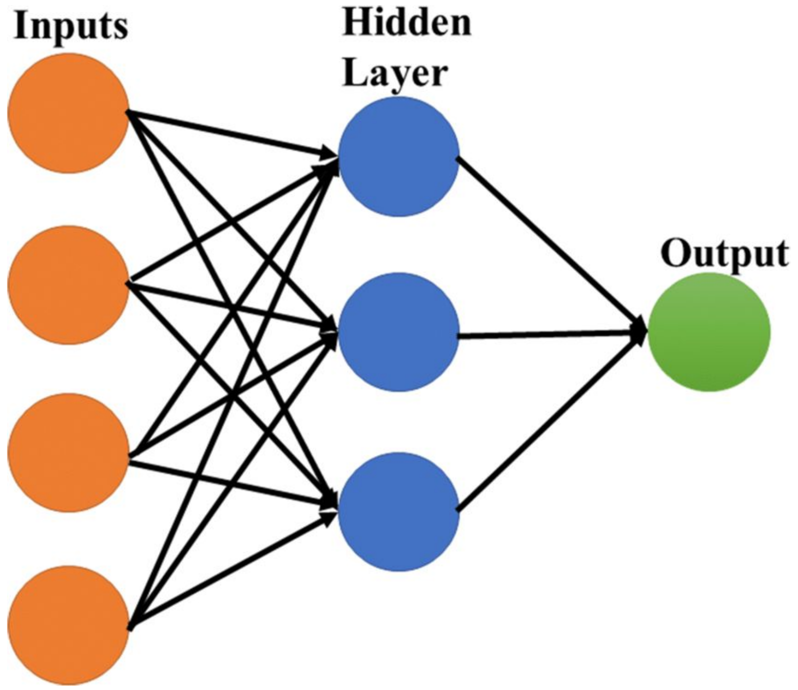
## Task Breakdown

- This is a supervised learning task, because we have our labels
- This is a classification task, because we are trying to classify flower species
- We have three classes: Setosa, Virsicolor, Virginica
- This means for each sample, our model will output three probabilities
- The largest probability in each output will indicate which class the sample is most likely to belong

For this example, I want to showcase how we go from inputs to outputs. Therefore, I will be coding it from scratch - layer 1 of the auto-ml depth.

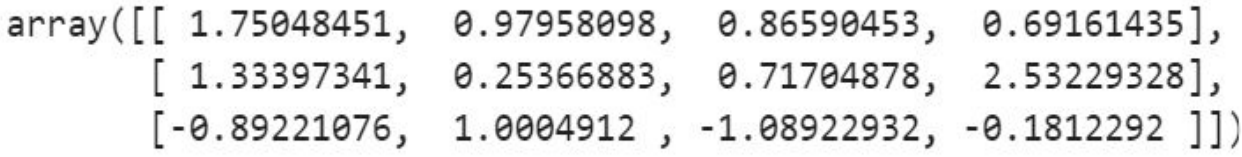
The purpose of coding it from scratch is so we can inspect the inputs & outputs of specific computations happening in the neuron. (they’re actually pretty simple!)

## Visualize Network



The following example will walkthrough a network with the following architecture. One hidden layer.

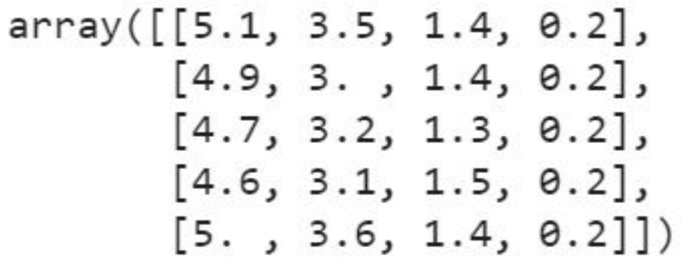
## Setup



```
array([[ 1.75048451,  0.97958098,  0.86590453,  0.69161435],
       [ 1.33397341,  0.25366883,  0.71704878,  2.53229328],
       [-0.89221076,  1.0004912 , -1.08922932, -0.1812292 ]])
```

Weights (W)

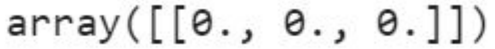
(3 neurons, 4 features)



```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2]])
```

Input Data (X)

(5 samples, 4 features)



```
array([[0., 0., 0.]])
```

Bias (b)

One bias vector for each neuron

# Forward Pass

Z = sum(Xi*Wi) + b

```
array([[13.70659364,  9.20143221, -2.60972253],
       [12.86670625,  8.80780312, -2.93152598],
       [12.62593509,  8.52003732, -2.44406266],
       [12.52610945,  8.50468286, -2.67273657],
       [13.62950329,  9.09340176, -2.42045234]])
```

A = activation(Z)

```
array([[3.28559099e-01, 3.63117141e-03, 2.69480546e-08],
       [1.41858394e-01, 2.44960357e-03, 1.95330453e-08],
       [1.11503745e-01, 1.83704887e-03, 3.18033294e-08],
       [1.00910353e-01, 1.80905741e-03, 2.53023449e-08],
       [3.04182051e-01, 3.25934039e-03, 3.25631508e-08]])
```

We have a prediction for each sample

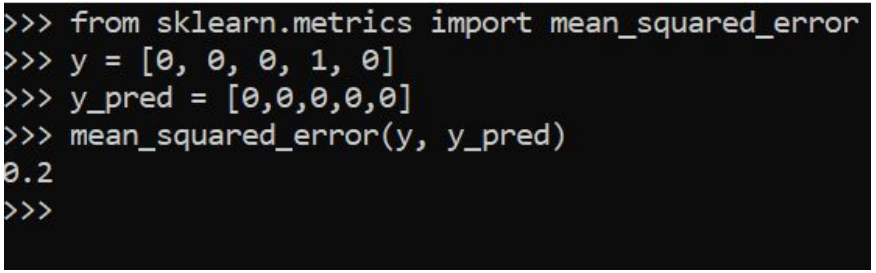
```
[0, 0, 0, 0, 0]
```

Take the position of the largest value in each row

## Calculate Loss

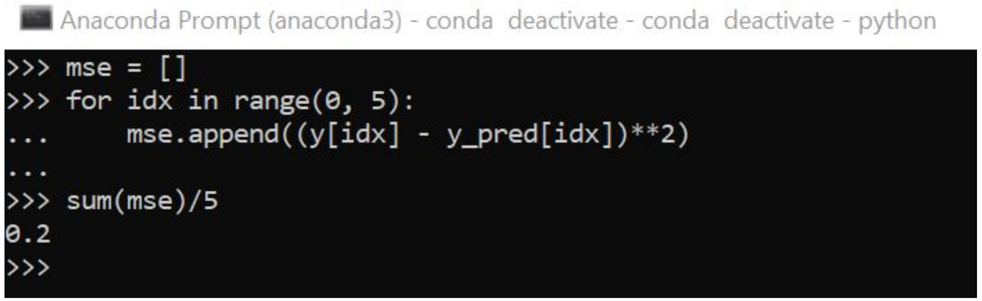
This is why we use open-source APIs

We could calculate the loss from scratch, or we could use an API.



```
>>> from sklearn.metrics import mean_squared_error
>>> y = [0, 0, 0, 1, 0]
>>> y_pred = [0,0,0,0,0]
>>> mean_squared_error(y, y_pred)
0.2
>>>
```

Using an API (one line of code). No math involved.



Anaconda Prompt (anaconda3) - conda deactivate - conda deactivate - python

```
>>> mse = []
>>> for idx in range(0, 5):
...     mse.append((y[idx] - y_pred[idx])**2)
...
>>> sum(mse)/5
0.2
>>>
```

From scratch. Performing computations.

These computations get exponentially more complex as we've already seen in slide 15.

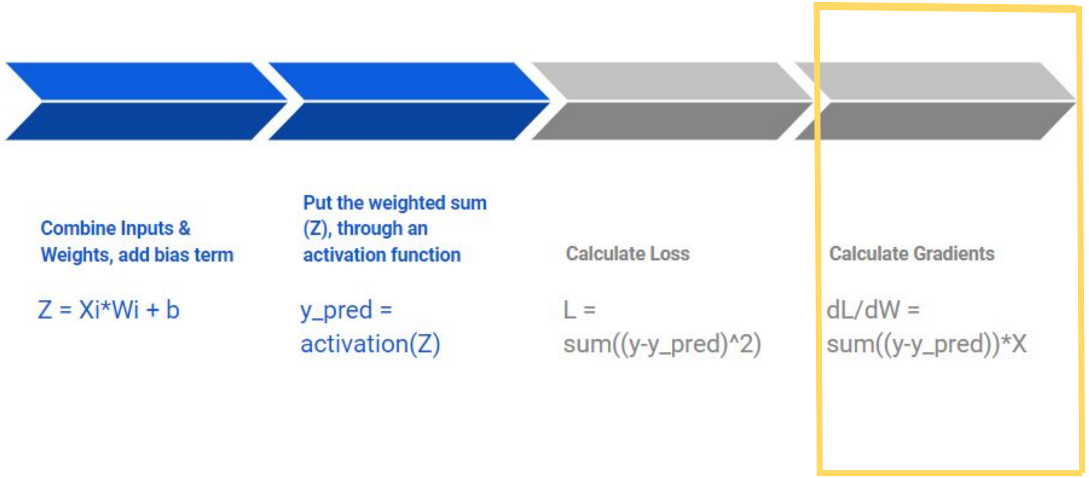
## Backward Pass - Overview 1

- The backward pass, AKA **Backpropagation**, is how we update the parameters of the network.
- Backpropagation is a bit beyond the scope of this course, but is the true essence and brilliance of ML, so it is important to know about.

The previous slide presented a brief walkthrough of how a model makes predictions. But how does the model actually learn the correct parameters, AKA train?

This is called backpropagation.

# Backward Pass Overview 2


Combine Inputs & Weights, add bias term
Z = Xi*Wi + b
Put the weighted sum (Z), through an activation function
y_pred = activation(Z)
Calculate Loss
L = sum((y-y_pred)^2)
Calculate Gradients
dL/dW = sum((y-y_pred))*X

Bringing back this infographic to highlight step 4 (backpropagation)

$$W = W - dL/dW$$

Instead of moving data from the input layer to the output layer, the data flows backwards from output layer to the input layer. Updating the weights, by subtracting the new value obtained in step 4.

## Recap

**Training** $\longrightarrow$

1. We combine our weights with our inputs and add the bias
2. We send the output from step 1 through an activation function to get our prediction
3. We calculate the loss, and perform backpropagation
4. Steps 1-3 are repeated over the number of epochs

| **Hyper-Parameters (controlled by human)** | **Trainable Parameters (not controlled by human)** |
|---|---|
| Epochs | Weights |
| Number of layers | Bias |
| Number of neurons in each layer | |
| Number of features | |

## Conclusion

## Questions

1. Which of the following is a trainable parameter(s)?
   a. Number of hidden layers
   b. Weights & biases
   c. Number of epochs
2. Which of the following occurs during the forward pass of an ANN?
   a. Model outputs predictions
   b. Model updates the parameters
3. Why would I use a pre-trained model over building my own?
   a. To have more control over the network and training
   b. Trained on a massive dataset and has proven accuracy
4. A vector is… finish the sentence!
   a. A large excel sheet filled with data
   b. A row or column of data
   c. The weights of an ANN