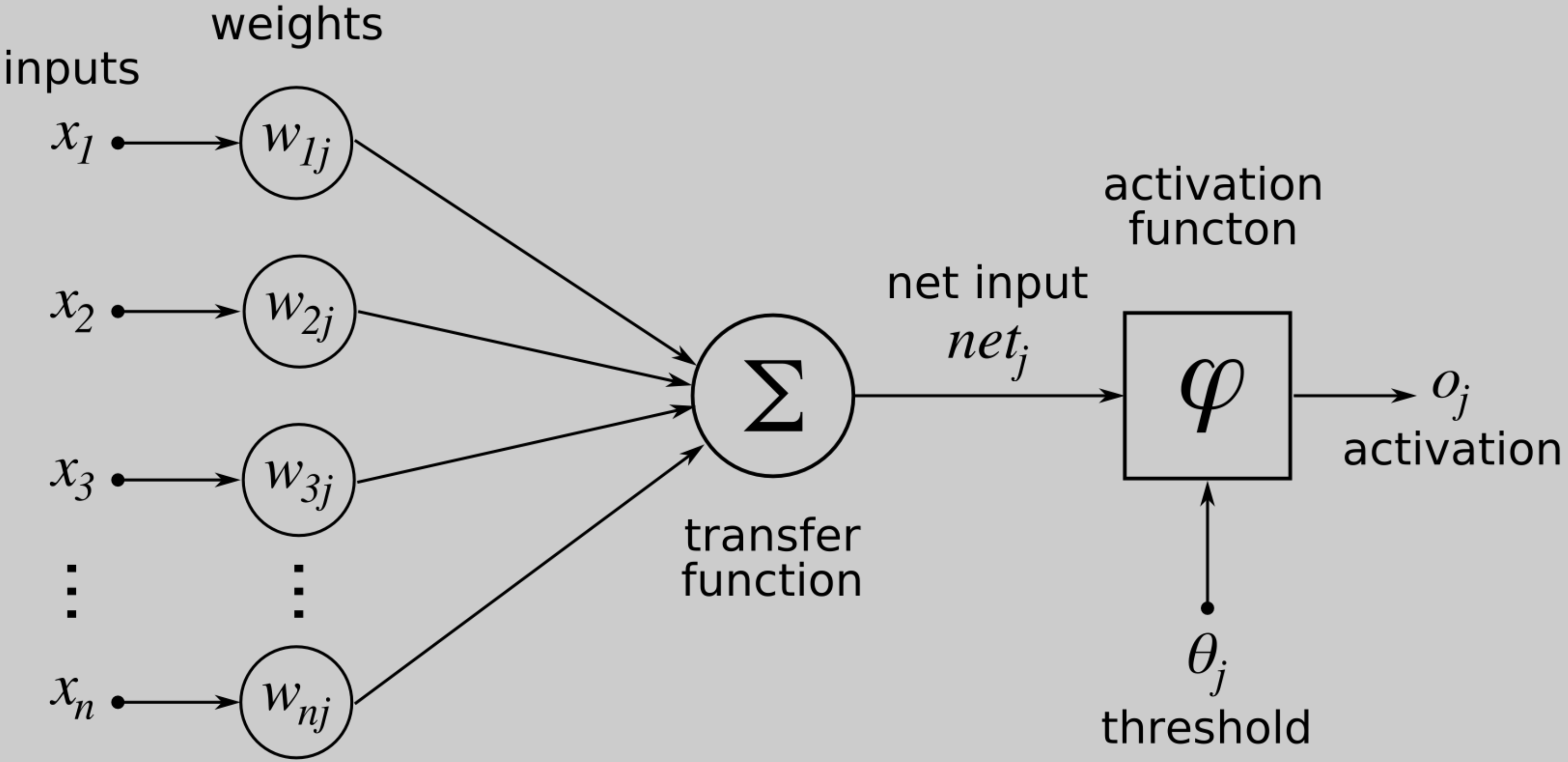
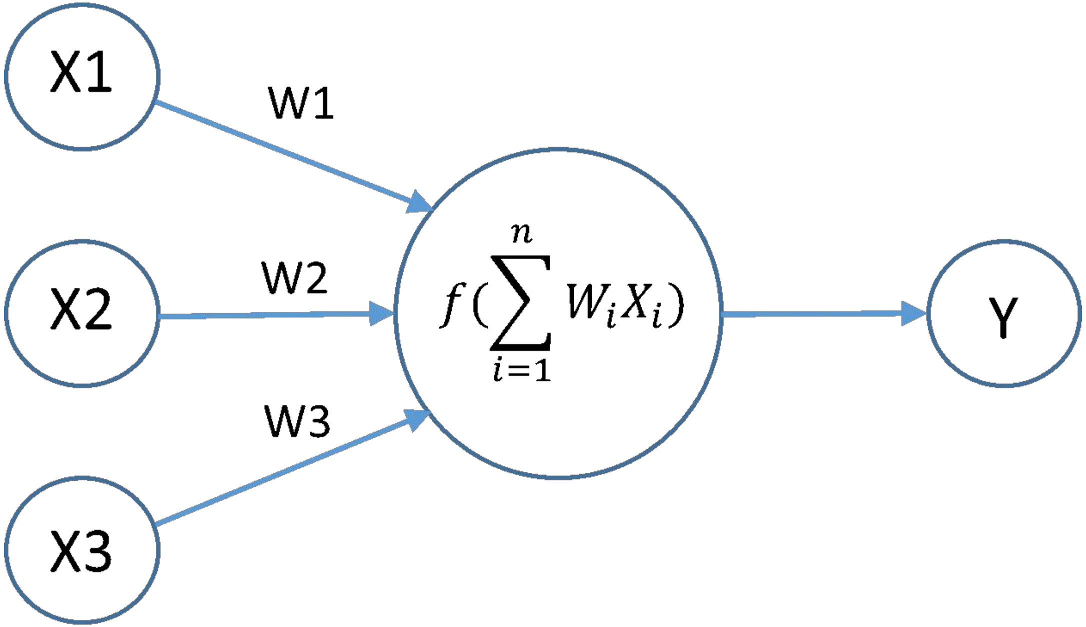
Neural Networks - No Smoke

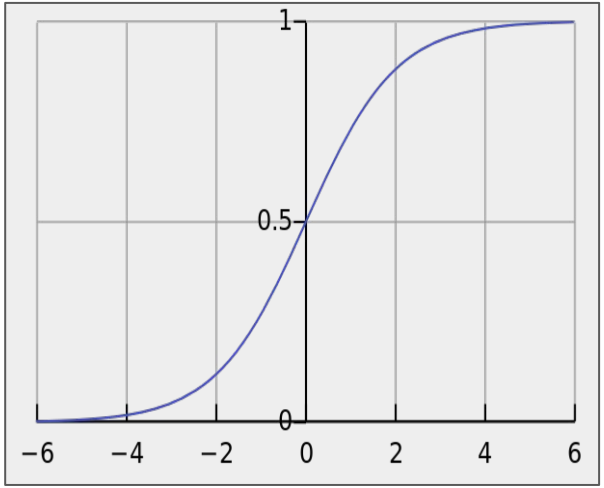


How it really Works?

How it Works? - Neurons

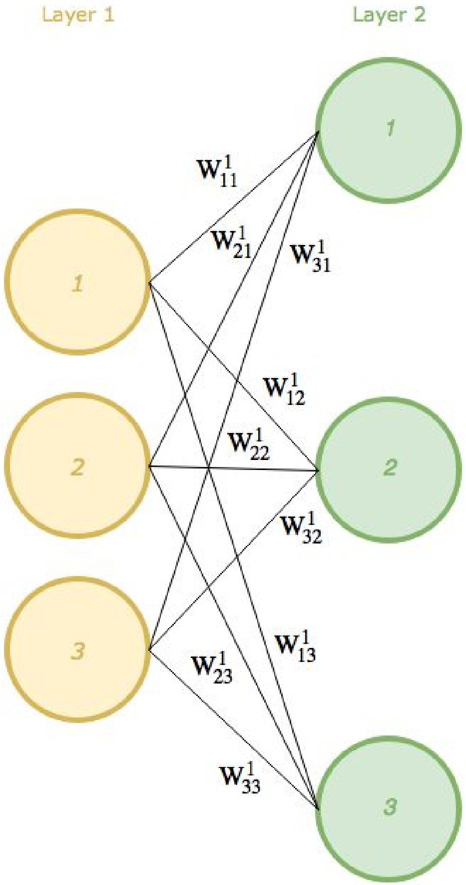


Activation Functions



Sigmoid

How it Works? - Matrix Approach



$$h_{in_1} = \sum (x_i \cdot w_{1i})$$

h1 = x1 * w11 + x2 * w21 + x3 * w31

h2 = x1 * w12 + x2 * w22 + x3 * w32

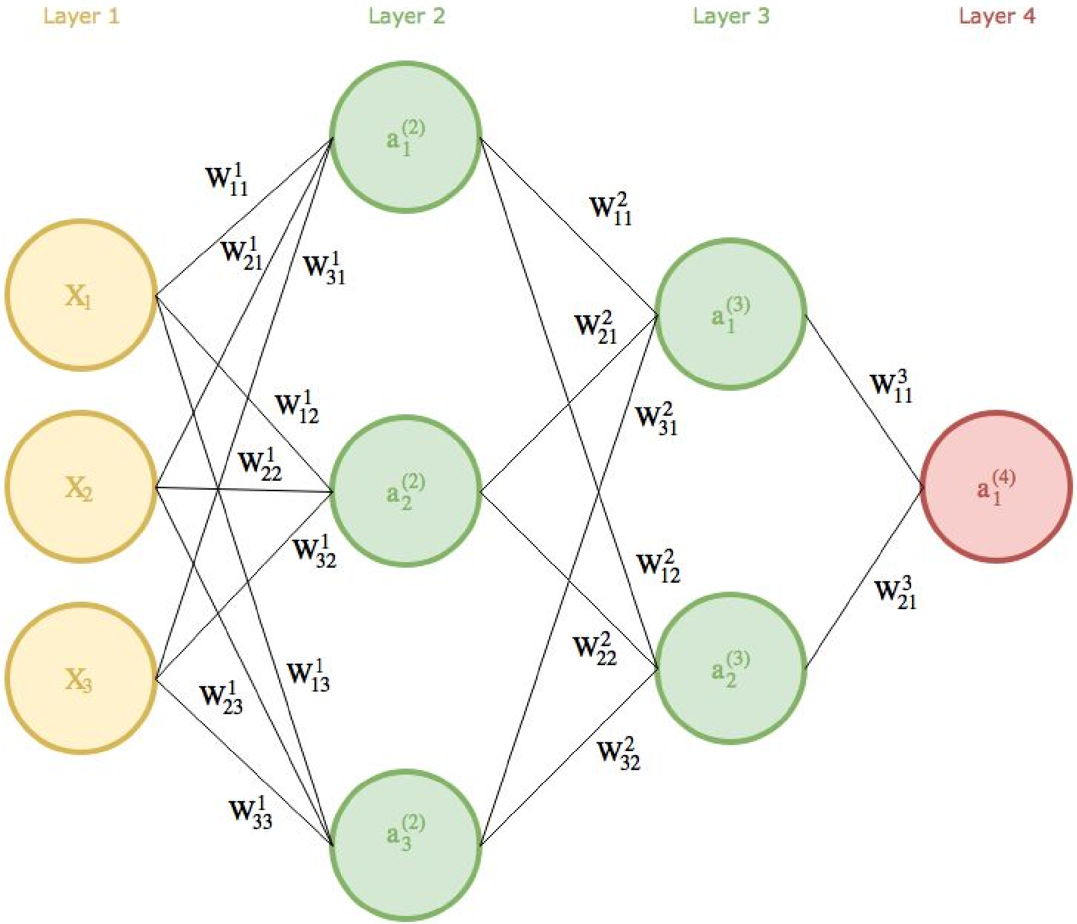
h3 = x1 * w13 + x2 * w23 + x3 * w33

$$W_1 = \begin{bmatrix} 0.01 & 0.05 & 0.07 \\ 0.2 & 0.041 & 0.11 \\ 0.04 & 0.56 & 0.13 \end{bmatrix}$$

$$\times$$

$$X_1 = \begin{bmatrix} 1.4 & -1 & 0.4 \end{bmatrix} \quad \begin{bmatrix} Z^{(2)}_1 & Z^{(2)}_2 & Z^{(2)}_3 \end{bmatrix}$$

How it Works? - Layer Cascading



L2 = Act_Funct(X*W1)

L3 = Act_Funct(L2*W2)

L4 = Act_Funct(L3*W3)

How it Works? - Backpropagation

Error = Y - Ý   [Real Output - Estimated]

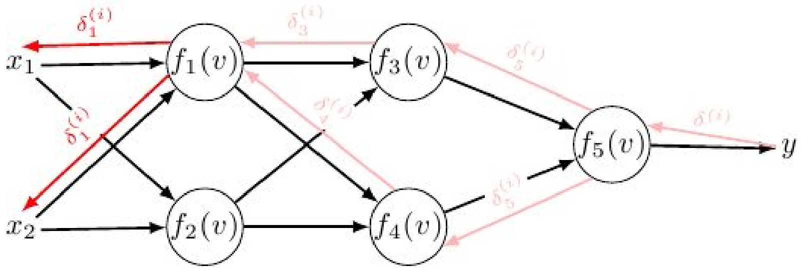
Delta = Error x SigmoidDerived(Ý) [scalar mult]

Now we need to get how much each synapse contributed to that error. To get that we do the reverse calculation from the forward propagation.

Reverse Calculation [Matrix Properties]

LayerOutput^{T} * Delta = W_{error}
[How much each weight contributed to the error]

W -= alpha * W_{error} [alpha is the learning rate]



$$J = \sum \alpha (y - \hat{y})^2$$

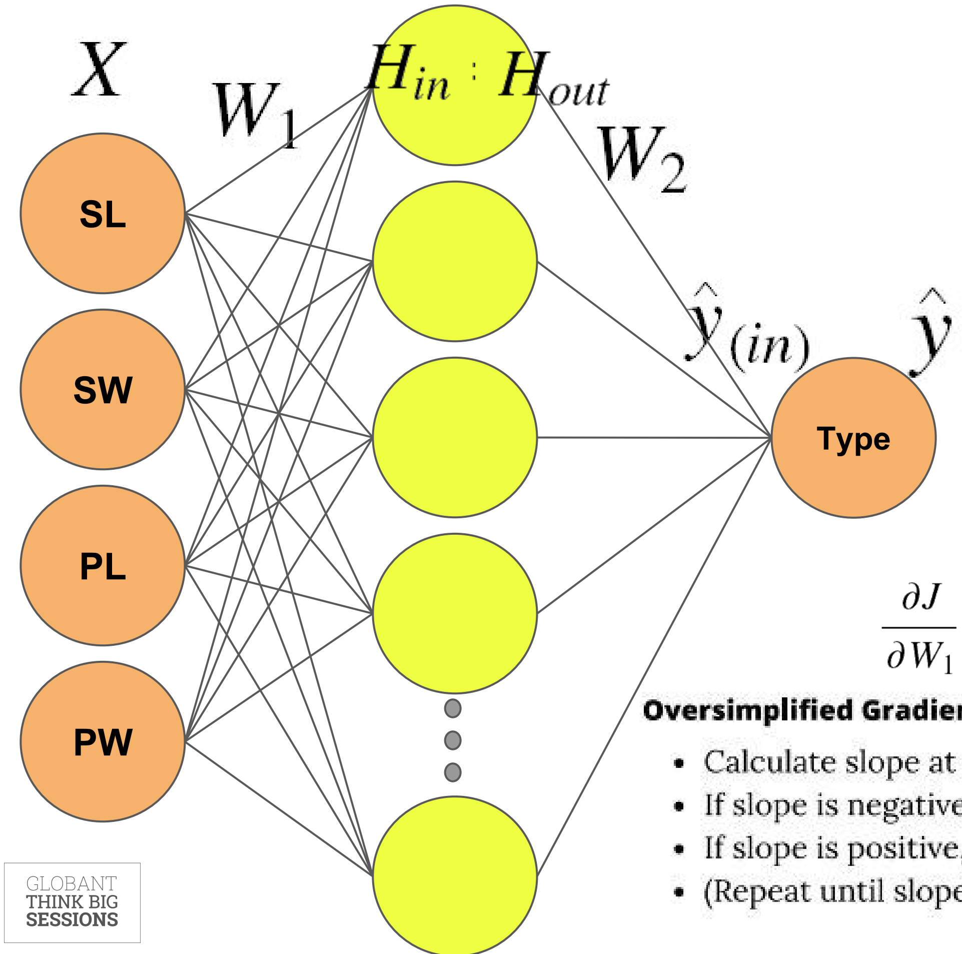
$$J = \sum \alpha (y - f(f(X \cdot W_1) \cdot W_2))^2$$

$$\frac{\partial J}{\partial W_2} = \frac{\partial \left(\sum \alpha (y - f(f(X \cdot W_1) \cdot W_2))^2 \right)}{\partial W_2}$$

$$\frac{\partial J}{\partial W_2} = \ldots = -2\alpha (y - \hat{y}) * f'(Z_3) \cdot A_2$$

$$= -2\alpha \cdot (A_2)^T \cdot (y - \hat{y}) * f'(Z_3)$$

Gradient Descent



$$J = \sum \alpha (y - \hat{y})^2$$

$$J = \sum \alpha (y - f(f(X \cdot W_1) \cdot W_2))^2$$

$$\frac{\partial J}{\partial W_2} = \frac{\partial \sum \alpha (y - f(f(X \cdot W_1) \cdot W_2))^2}{\partial W_2}$$

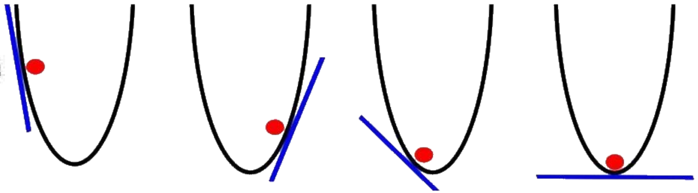
$$\frac{\partial J}{\partial W_2} = \ldots = -2\alpha (y - \hat{y}) * f'(H_{out} \cdot W_2) \cdot H_{out}$$

$$\frac{\partial J}{\partial W_1} = \frac{\partial \sum \alpha (y - f(f(X \cdot W_1) \cdot W_2))^2}{\partial W_1}$$

$$\frac{\partial J}{\partial W_1} = \ldots = -2\alpha (y - \hat{y}) * X^T \cdot f'(H_{out} \cdot W_2) \cdot W_{2^T} \cdot f'(X \cdot W_1)$$

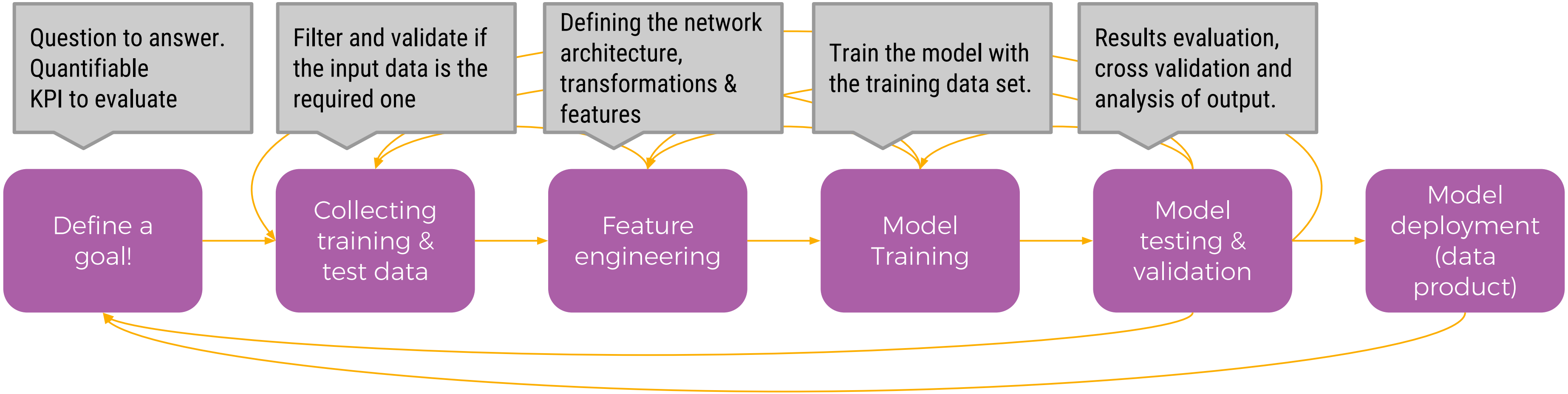
Oversimplified Gradient Descent:

- Calculate slope at current position
- If slope is negative, move right
- If slope is positive, move left
- (Repeat until slope == 0)



The design process

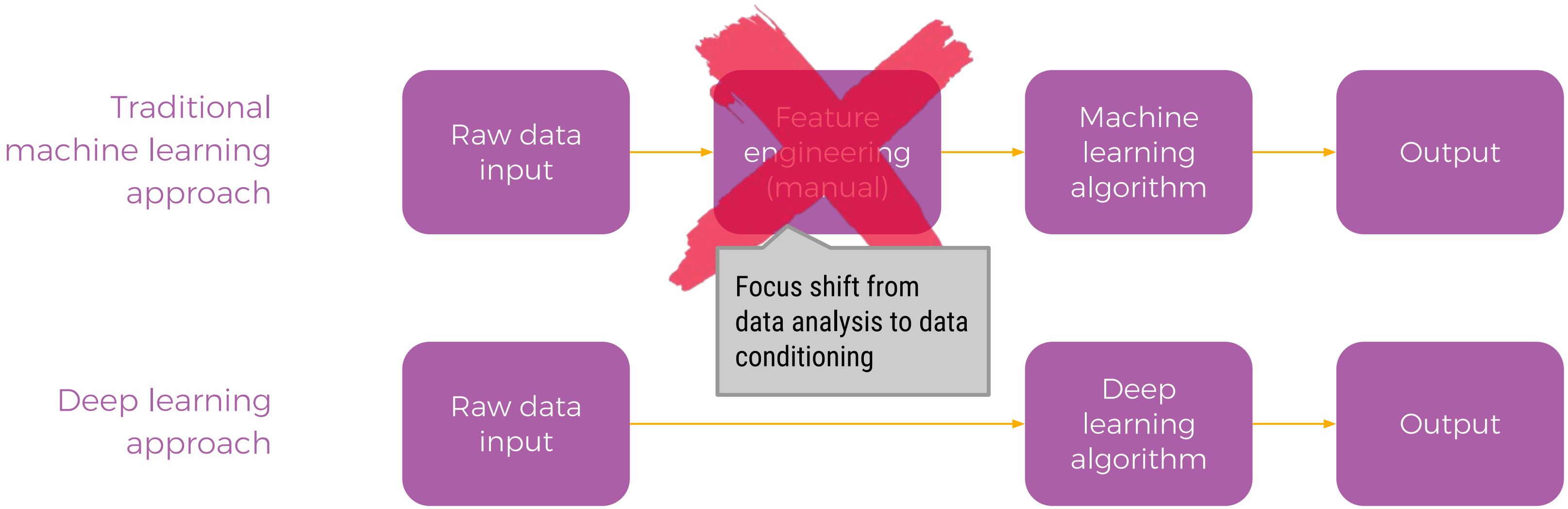
Machine learning workflow



The process of developing a machine learning based data product is hardly ever a linear process.

…learning representations of data

- Deep Neural Networks learn representations of the data!



Let’s think of a simple example

Define a goal!

Collecting training & test data

Feature engineering

Model Training

Model testing & validation

Determine the specie of Iris Flowers based on their physical dimensions.

Define a goal!

Collecting training & test data

Feature engineering

Model Training

Model testing & validation

In this case we will get ourr information from https://en.wikipedia.org/wiki/Iris_flower_data_set that provides us a large set of field measurements of each Iris flower specie.

Training set: 60 samples
Testing set: 35 samples

Define a goal!

Collecting training & test data

~~Feature engineering~~

Model Training

Model testing & validation

We will do no data filtering, data conditioning or pre-prossesing.

Define a goal!

Collecting training & test data

Feature engineering

Model Training

Model testing & validation

- Gradient descent
- Backpropagation

Define a goal!

Collecting training & test data

Feature engineering

Model Training

Model testing & validation

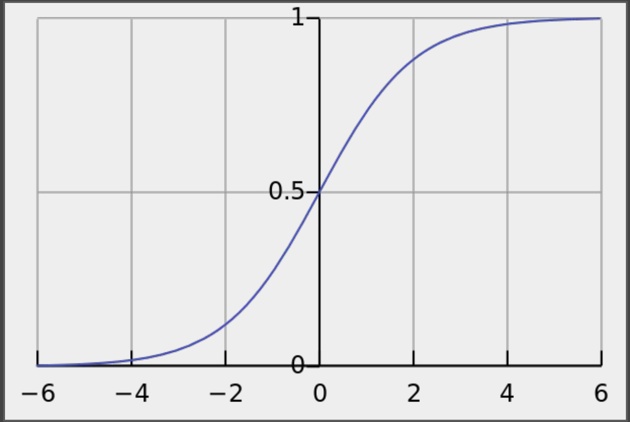
- Predictions V.S. Truth
- Error Rate
- Average prediction error

How hard could it be?

Let's Do It!

Forward propagation (processing the sample)

```
public Matrix<double> Forward(Matrix<double> input)
{
    Matrix<double> layer1Output = Sigmoid(input * synapse0);
    Matrix<double> layer2Output = Sigmoid(layer1Output * synapse1);
    return layer2Output;
}
```



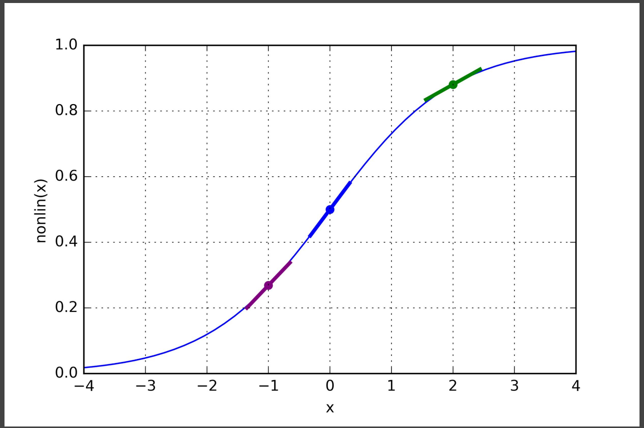

```
//Activation Function
static Matrix<double> Sigmoid(Matrix<double> matrix)
{
     //Output: 1/(1 + e^-x) for every element of the input matrix.
     Matrix<double> outputMatrix = Matrix<double>.Build.Dense(matrix.RowCount, matrix.ColumnCount);

     foreach (var tuple in matrix.EnumerateIndexed())
          outputMatrix.At(tuple.Item1, tuple.Item2, (double)(1 / (1 + Math.Exp(-tuple.Item3))));

     return outputMatrix;
}
```

Training the network

```
for (int i = 0; i < EpochsIterations; i++)
{
    //Process inputs
    Matrix<double> layer1Output = Sigmoid(trainingSetInput * synapse0);
    Matrix<double> layer2Output = Sigmoid(layer1Output * synapse1);

    //Calculate Layers Error
    Matrix<double> layer2Error = layer2Output - trainingSetOutput;
    Matrix<double> layer2Delta = BackpropagateLayerError(layer2Output, layer2Error);

    Matrix<double> layer1Error = layer2Delta * synapse1.Transpose();
    Matrix<double> layer1Delta = BackpropagateLayerError(layer1Output, layer1Error);

    //Update Synapses
    synapse1 -= Alpha * (layer1Output.Transpose() * layer2Delta);
    synapse0 -= Alpha * (trainingSetInput.Transpose() * layer1Delta);
}
```

Backpropagation (computing the error)

```
private Matrix<double> BackpropagateLayerError(Matrix<double> outputCalculated, Matrix<double> error)
{
    Matrix<double> der = SigmoidDerived(outputCalculated);
    return error.EscalarMultiplication(der);
}
```




```
static Matrix<double> SigmoidDerived(Matrix<double> matrix)
{
     //Returns the value of the sigmoid function derivative f'(x) = f(x)(1 - f(x)),
     Matrix<double> outputMatrix = Matrix<double>.Build.Dense(matrix.RowCount, matrix.ColumnCount);

     foreach (var tuple in matrix.EnumerateIndexed())
          outputMatrix.At(tuple.Item1, tuple.Item2, tuple.Item3 * (1 - tuple.Item3));

     return outputMatrix;
}
```

Thank You!

Globant
we are ready

For questions: diego.brihuega@globant.com

Check out some code samples at: https://github.com/diegosfb/NNDemo