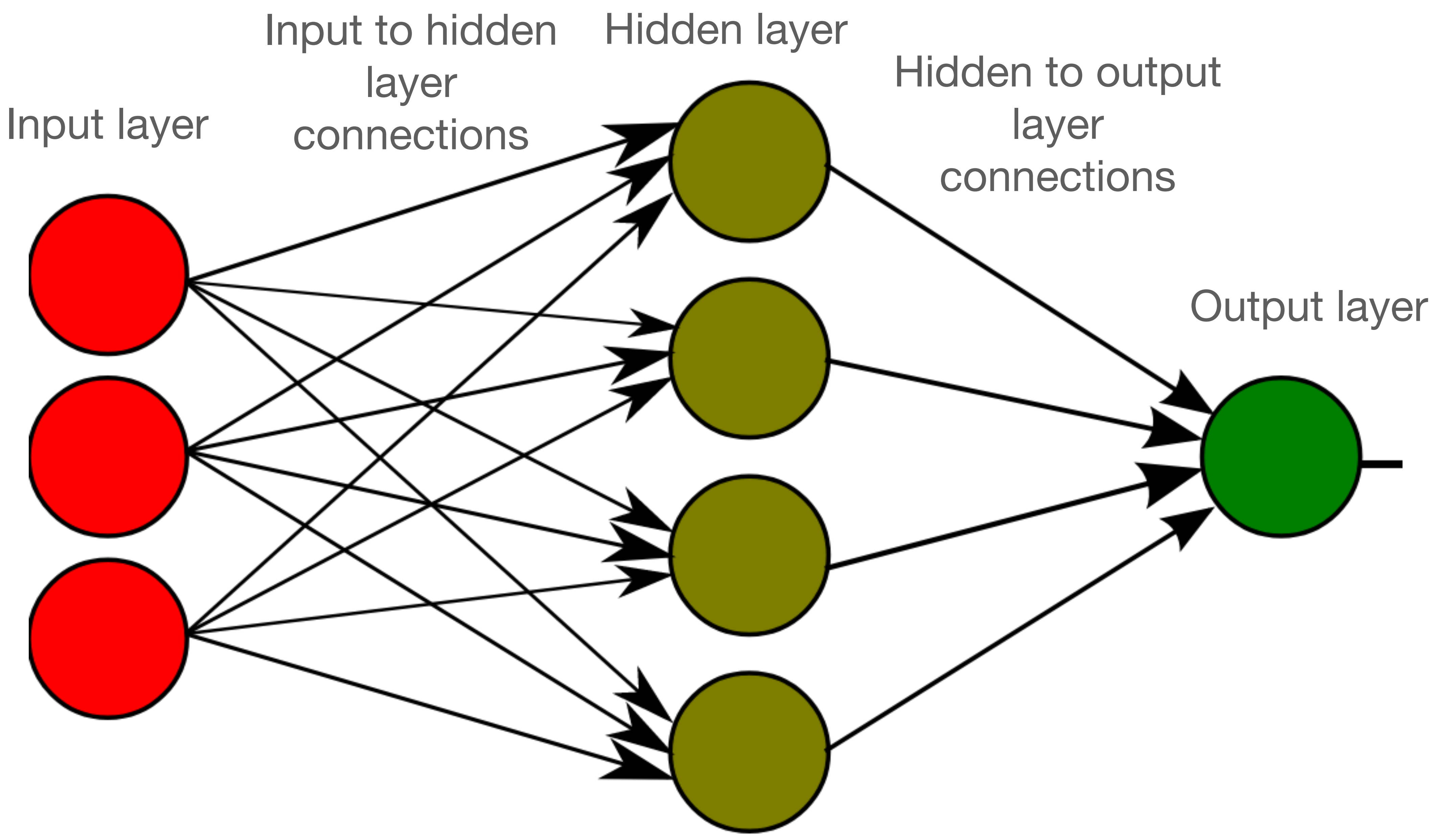
Building deep learning training pipelines

Perumadura De Silva

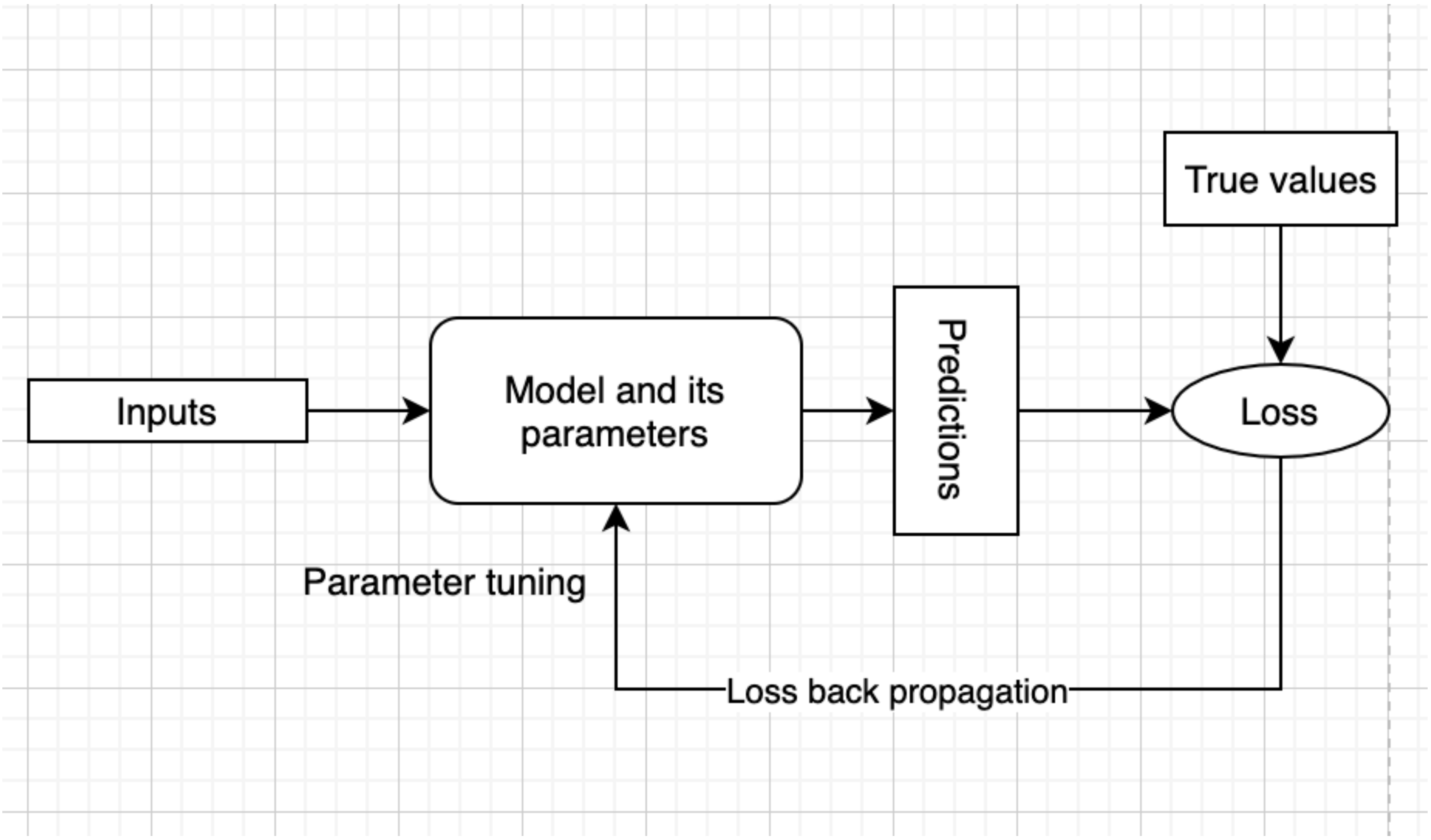
Content

- Structure of a deep learning training pipeline
- Training process of a neural network
- Introduction to model optimisation
- Deep neural network optimisation

Structure of a deep learning training pipeline



Ref.1 Multi-layer Neural Network



Components involved in supervise learning

Structure of a deep learning training pipeline

- Input layer:
  - Inputs to the neural network such as images, audio, text embedding etc.
  - Example:

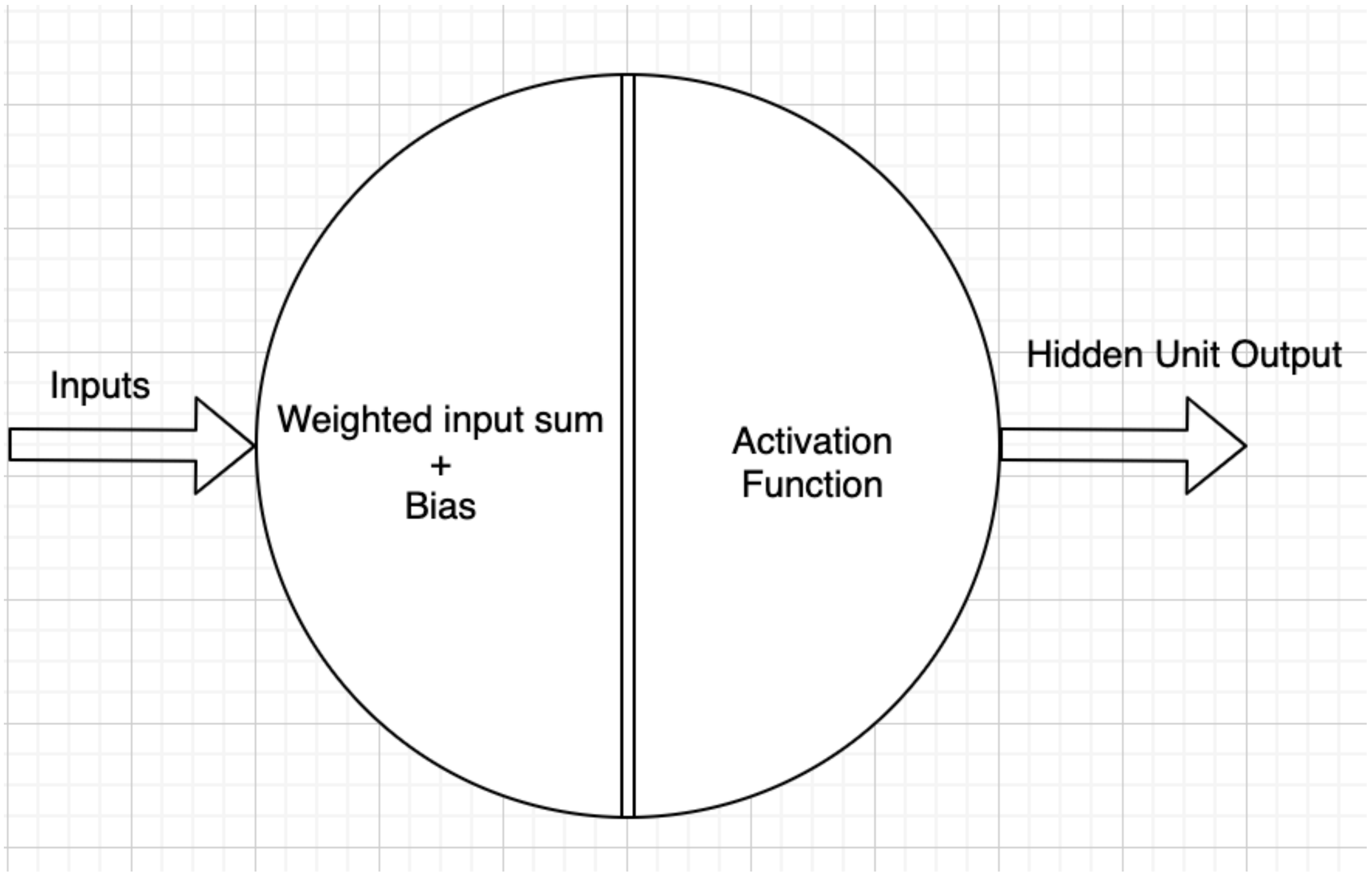
$$
\begin{array}{c}
\begin{array}{cccccc} & & & & & \textbf{Inputs} \end{array} \\
\begin{bmatrix}
0.5 & 0.2 & 0.1 & 0.33 & 0.51 & 0.14 \\
0.32 & 0.53 & 0.34 & 0.34 & 0.13 & 0.44 \\
0.14 & 0.56 & 0.44 & 0.41 & 0.46 & 0.65
\end{bmatrix}
\end{array}
$$

$\longleftarrow$ Number of data samples

- Each input sample is fed into the neural network:
  - Batch-wise (A set of samples together for a training iteration)
  - single example (one sample per training iteration)

Structure of a deep learning training pipeline

- Hidden layer:
  - Inputs connects to hidden layer through the weights
  - Extract the features of the inputs by means of weights and biases
  - This feature extraction encodes the information about an input
  - Then propagate this this information towards output layer
  - Activation function: Performs a nonlinear transformation on the inputs



Structure of a deep learning training pipeline

- Output layer:
  - Takes the inputs from the last hidden layer
  - Produce predictions
  - Similar to the inputs, the outputs has the same size in the batch dimension
  - The output dimension may differ

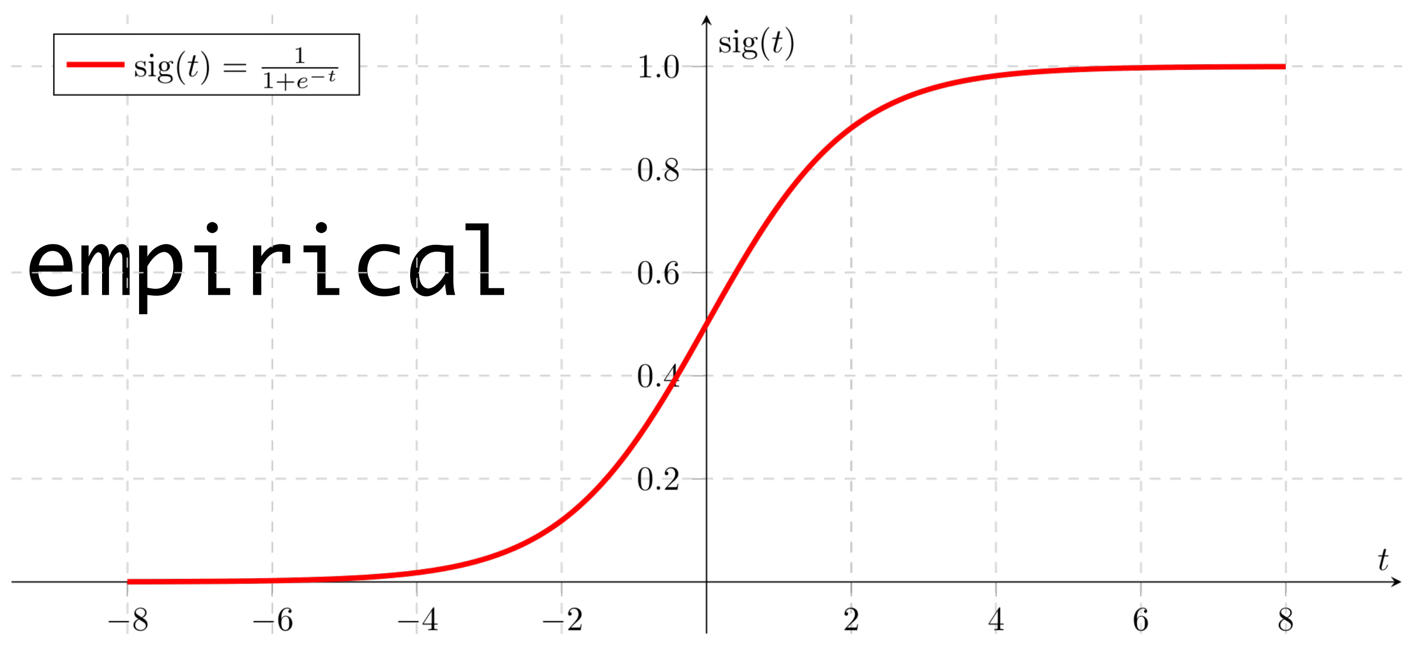
Training process of a neural network

- Three main components involved in training
  - Loss function
    - Compute a distance/encoding between the predictions and true values
    - Used in both supervised, semi supervised and self supervised cases
    - Example MEAN SQUARED ERROR:

$$mse = \frac{1}{2N} \sum_{i}^{N} (\hat{y} - y)^2$$

Training process of a neural network

- Neuron activation
  - Introduces nonlinearity to the layer outputs
  - Selection of the activation function is somewhat empirical
  - Example (Sigmoid function): $\sigma(t) = \dfrac{1}{1+e^{-t}}$
  - Must be differentiable
- Optimisation strategy
  - Two major components: Gradient descent and error back propagation
  - Gradient descent: Update parameters so the loss descend towards a global minima
  - Error backdrop: Propagate the loss gradients towards the parameters for gradient descent computation



Ref.4: Sigmoid function

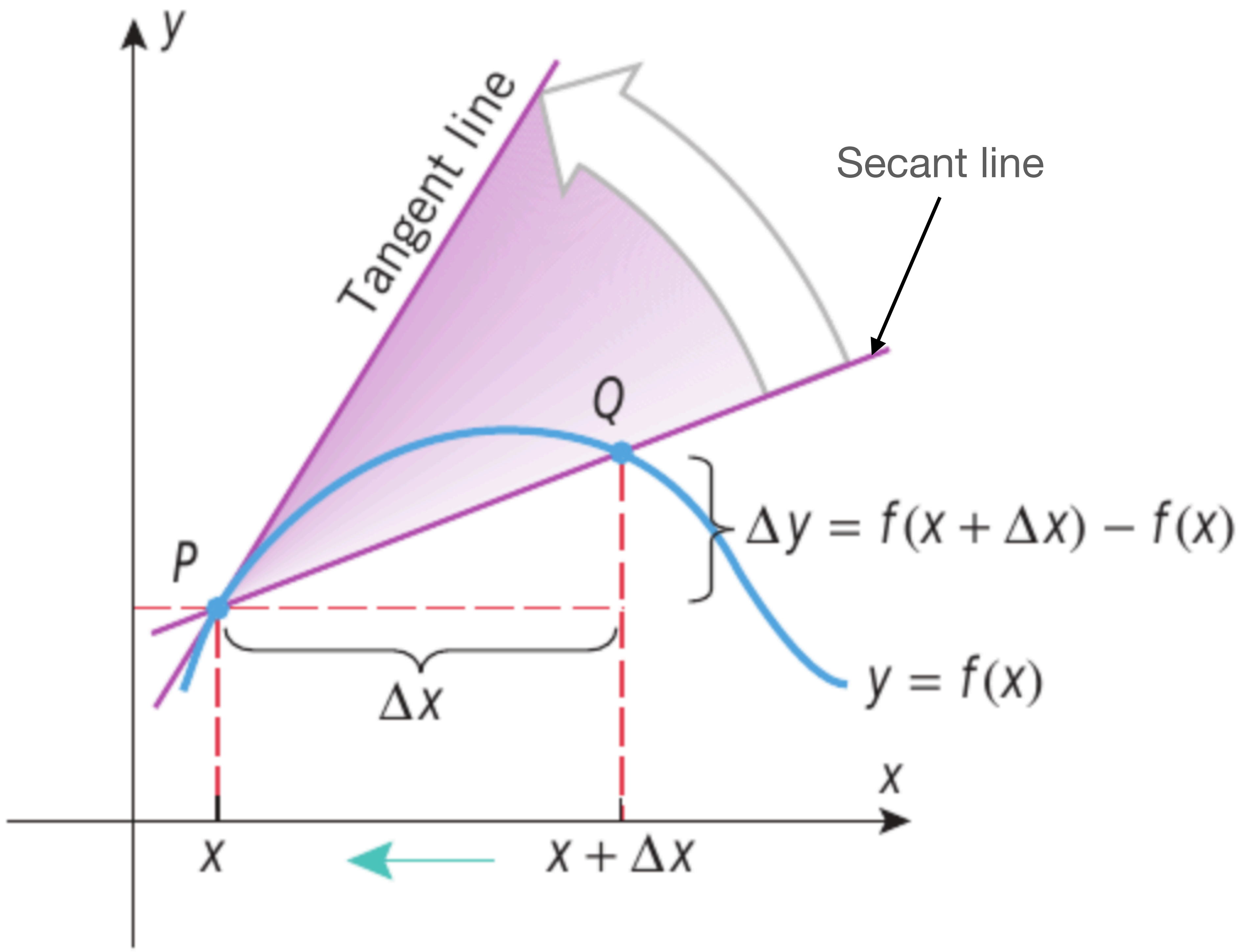
Colab Exercise

Training process of a neural network

- Training pseudo code:
   1. For each epoch:
      1. For each data batch:
         1. Perform a forward pass (prediction step)
         2. Evaluate the error
         3. Perform a backward pass (optimisation step)
         4. Check termination condition

Introduction to model optimisation

- Derivative:
  - Measures the slope of the graph of a function at a given point
- Given a function $f(x)$, the derivative of the function is giver by
  - $f'(x) = lim_{x \to 0} \frac{f(x + \Delta x)}{\Delta x} = \frac{\Delta y}{\Delta x}$
- To find a derivative of a function, the function must not contain any vertical slopes or breaks



Ref.2: Secant and tangent lines

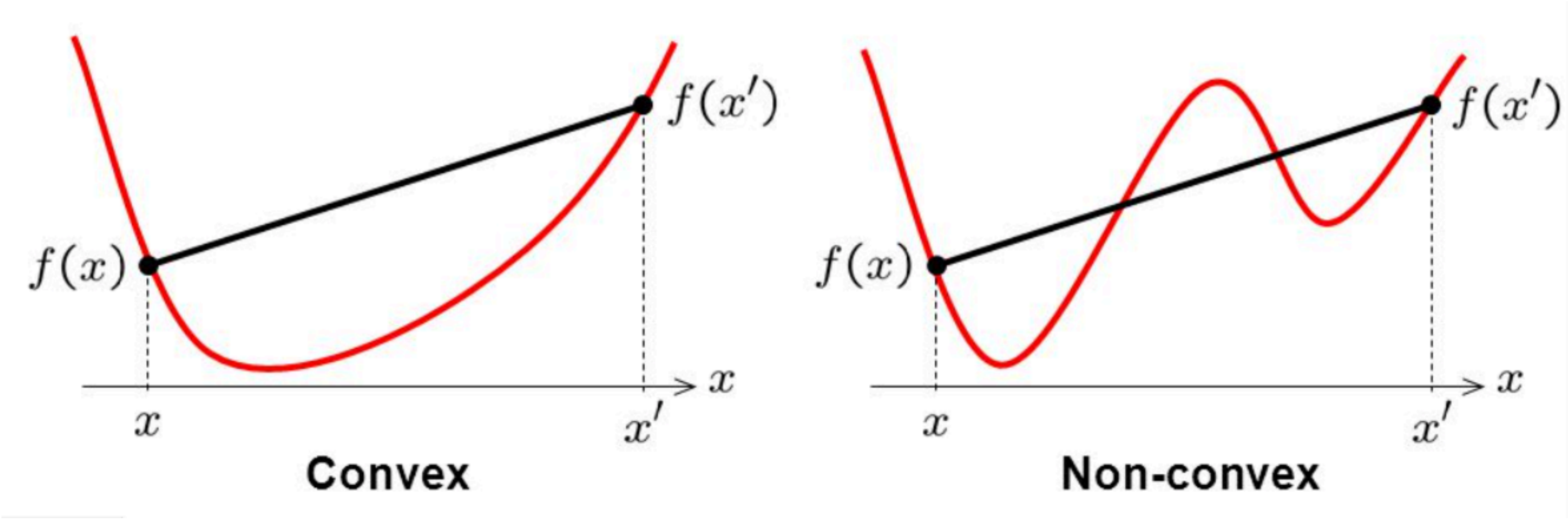
Introduction to model optimisation

- Rules of Differentiation:
  - Sum Rule: $\frac{d[f(x)+g(x)]}{dx} = \frac{df(x)}{dx} + \frac{dg(x)}{dx}$
  - Difference Rule: $\frac{d[f(x)-g(x)]}{dx} = \frac{df(x)}{dx} - \frac{dg(x)}{dx}$
  - Constant multiplication: $\frac{d[kf(x)]}{dx} = k\frac{df(x)}{dx}$
  - Constant Rule : $\frac{d[C]}{dx} = 0$
  - Product Rule: $\frac{d[f(x) \,.\, g(x)]}{dx} = f(x)\frac{dg(x)}{dx} + g(x)\frac{df(x)}{dx}$
  - The quotient Rule: $\frac{d[\frac{f(x)}{g(x)}]}{dx} = \frac{f(x)\frac{dg(x)}{dx} + g(x)\frac{df(x)}{dx}}{v^2}$

Introduction to model optimisation

- Chain Rule of Derivation:
  - A composition function is:
    - $(f \circ g)(x) = f(g(x))$ and $(g \circ f)(x) = g(f(x))$
  - The chain rule is the derivative of a composition function
    - $\dfrac{d[f(g(x))]}{dx} = (f \circ g)'(x) = f'(g(x))g'(x)$

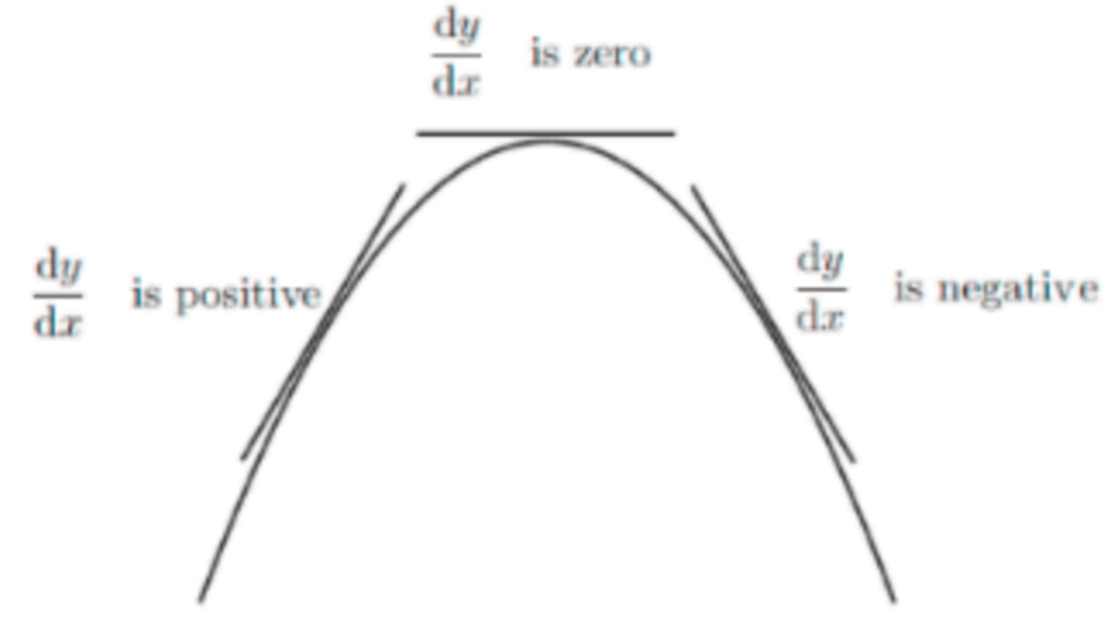
Introduction to model optimisation



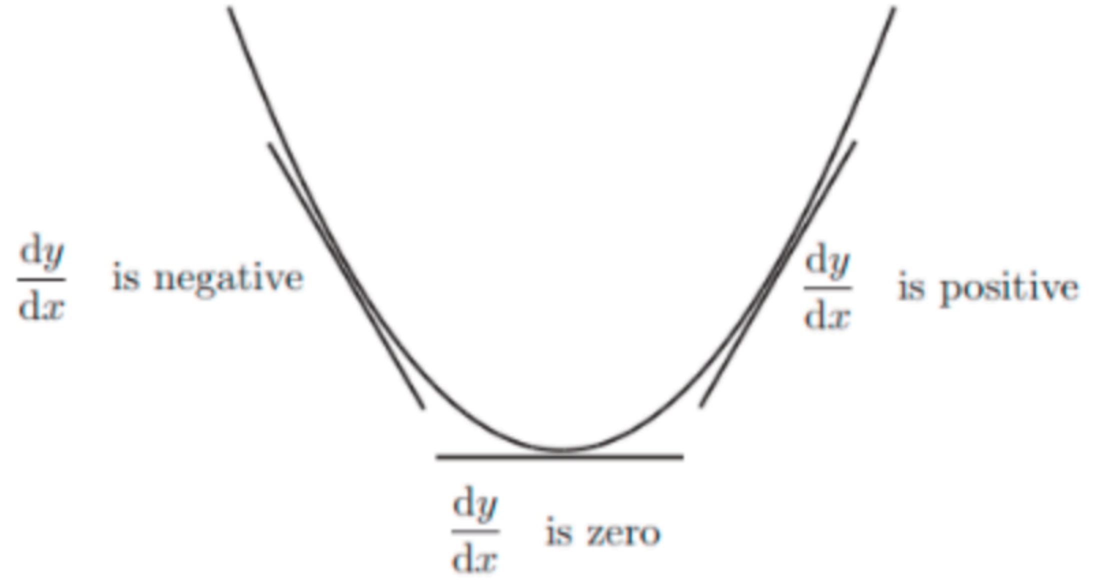
Ref.3 Convex functions has a global minima/maxima and Non-Convex functions have multiple local minimas and a global minima

Introduction to model optimisation

- Function minima/maxima
    - On the ascending direction, the gradient is positive
    - On the descending direction, the gradient is negative
    - On maxima/minima the gradient is “0”
    - Goal of optimisation:
        - Find the function parameters that either minimise or maximise the function (parameters that leads to a $\frac{dy}{dx} = 0$)



Ref.2: Maxima of a function



Ref.2: Minima of a function

Introduction to model optimisation

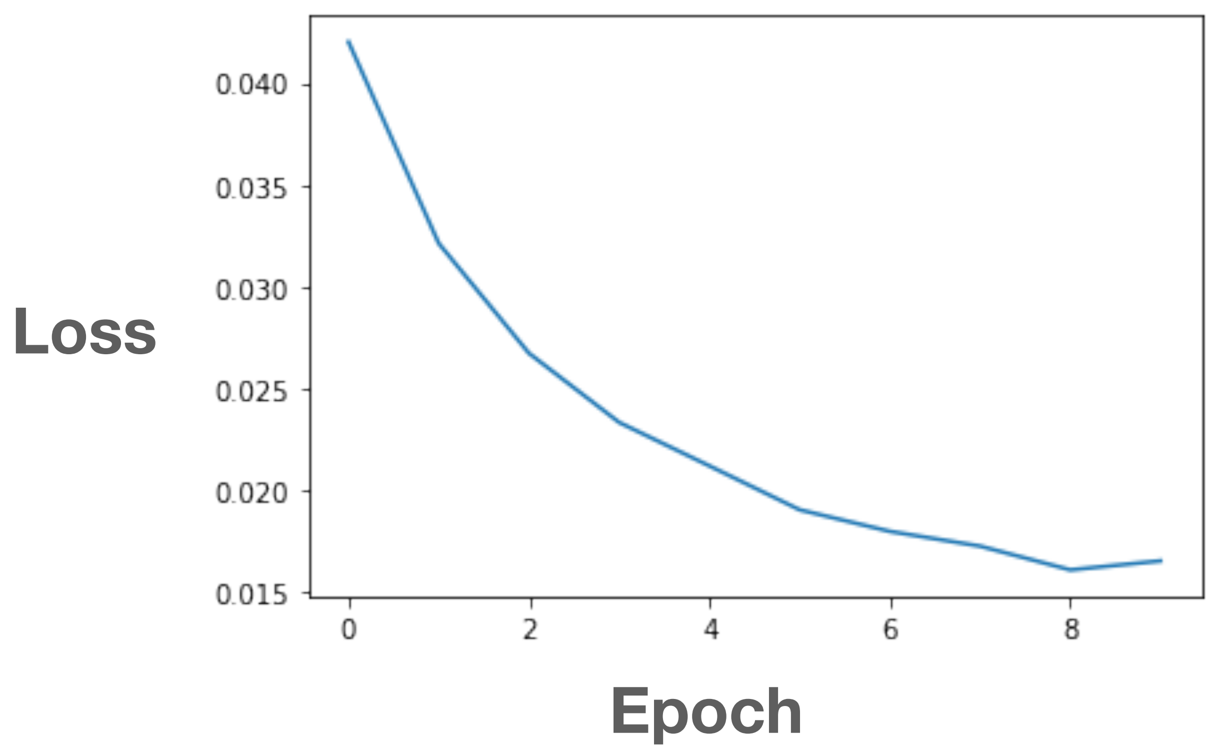
- Intuition on stepping towards a minima:
  - Take a function $f(\theta)$, a set of parameters θ and a loss function L
  - Parameter update equation:
    - $\theta_{new} = \theta_{old} - \alpha \dfrac{dL}{d\theta}$
      - $\dfrac{dL}{d\theta}$ is the loss derivative w.r.t parameters and α is the step size
  - So when the gradient is positive (ascending) we subtract the gradient so we move in the opposite direction
  - If the gradient is negative (descending) we keep moving on that direction (towards the minima)
  - Step size determines how big of a step we should take when calculating the next iteration parameters

Colab Exercise

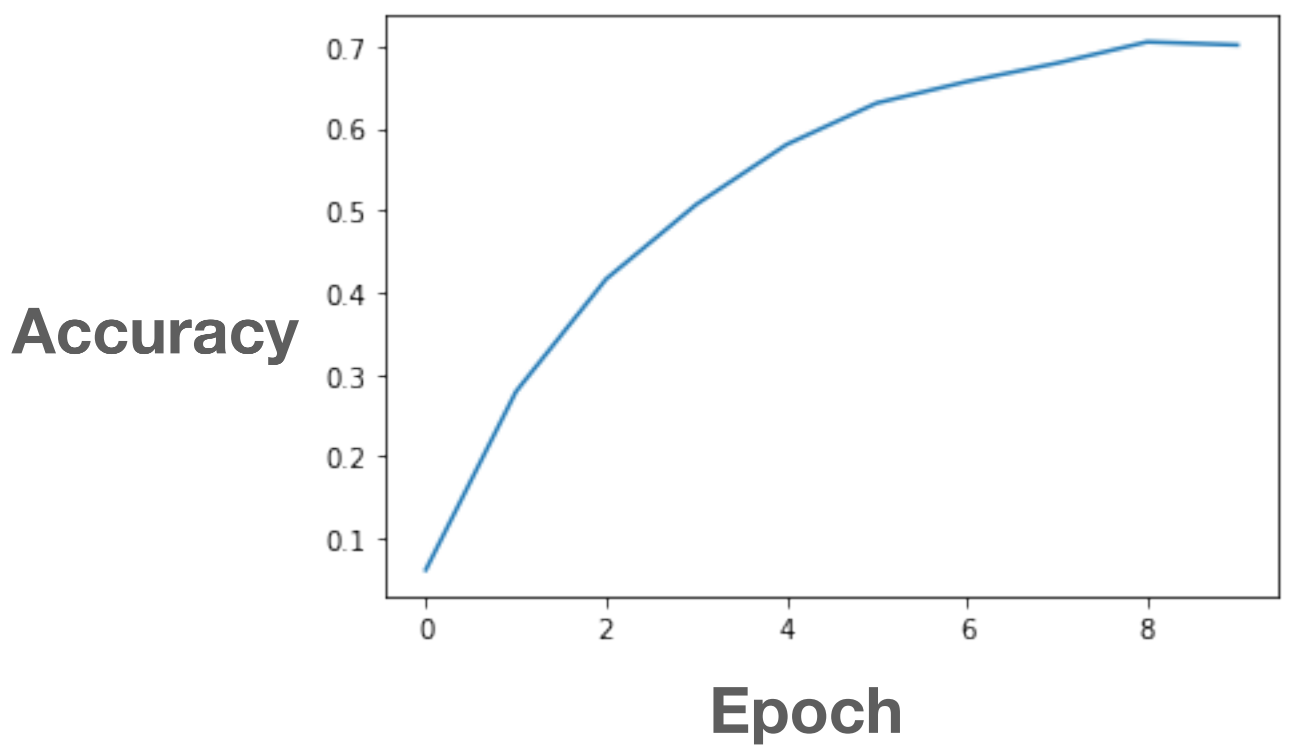
Data Normalization

- What is Normalization:
  - Bring all features to a standard range while preserving their strength of representation
- Why normalise data:
  - Feature in the different ranges could poorly weigh outputs
  - The unnormalised data can lead to an unexpected behaviour in the gradients

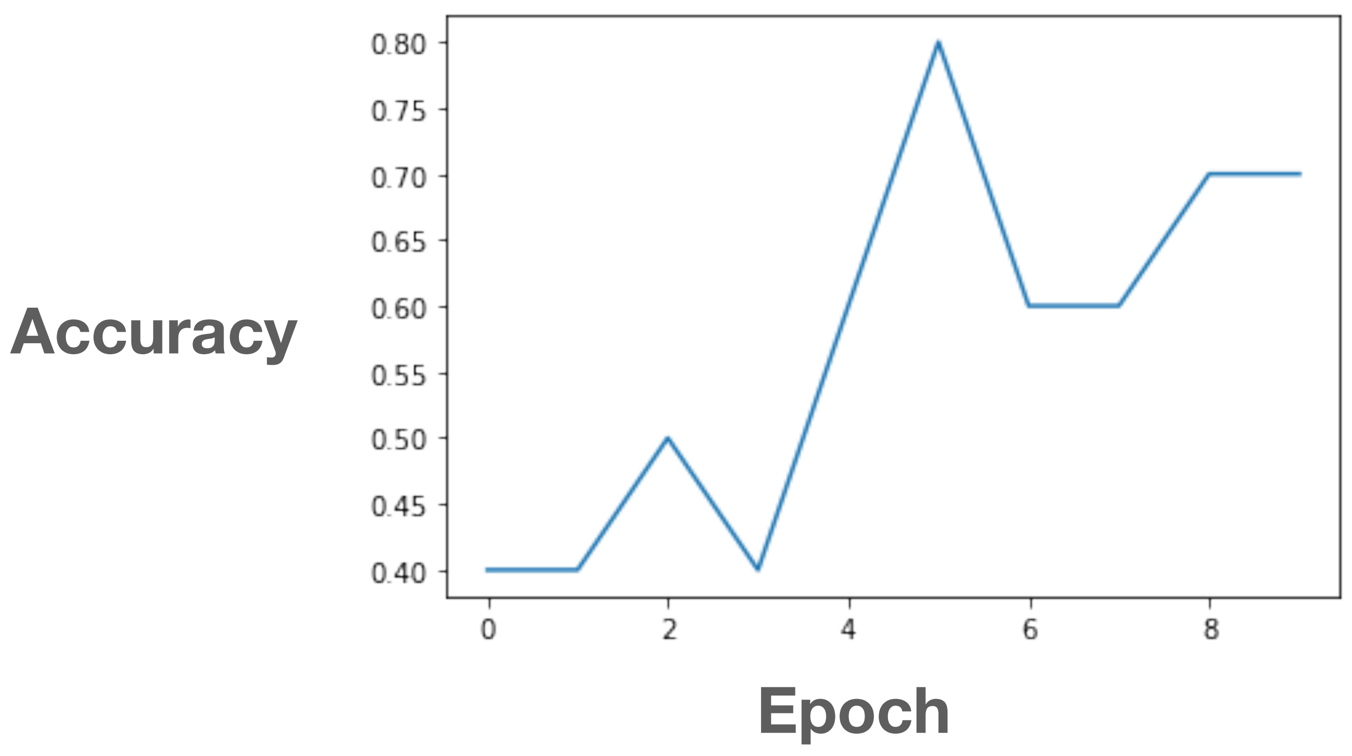
Data Normalization



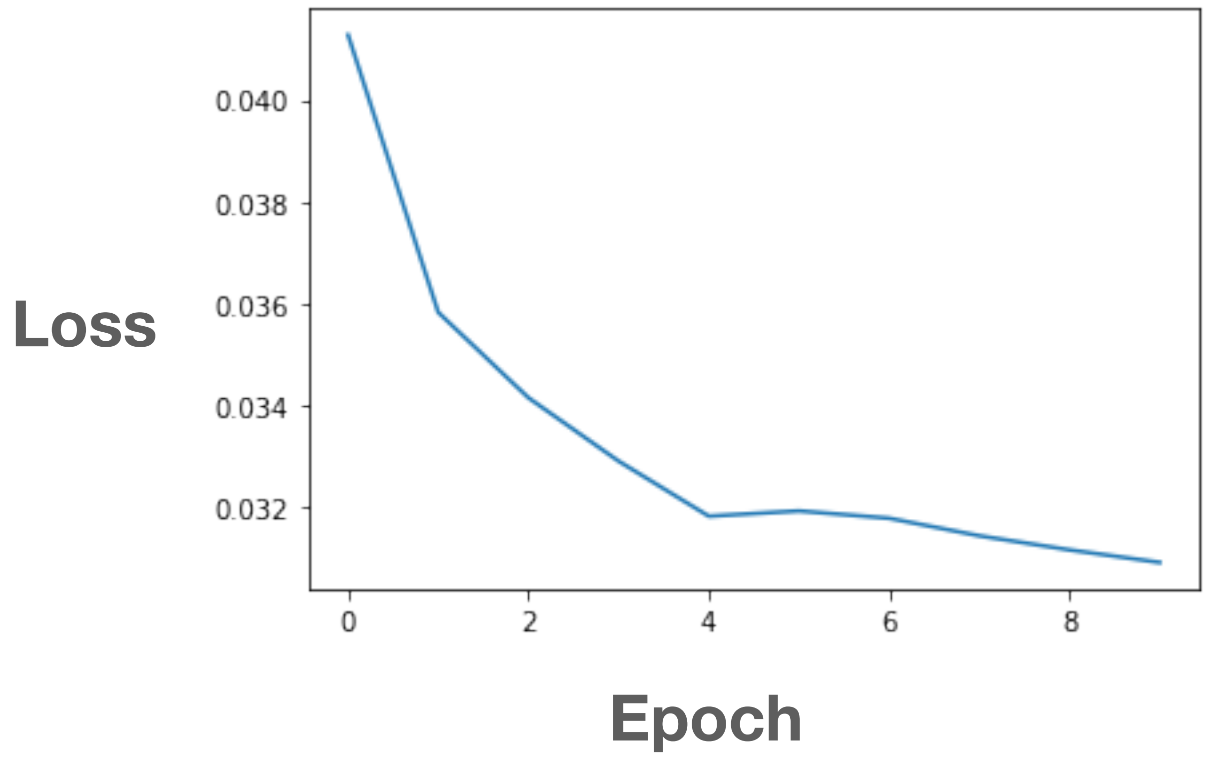
Train loss when inputs are 0-255



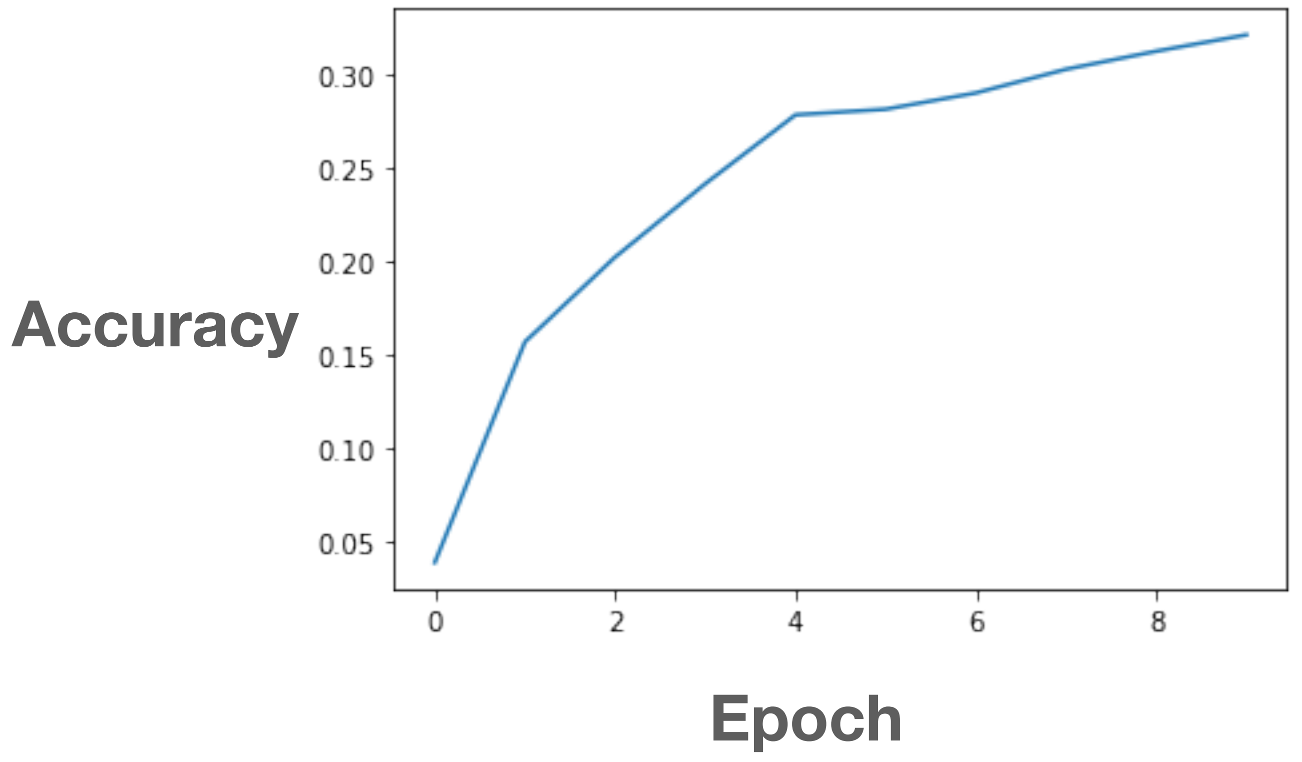
Train accuracy when inputs are 0-255



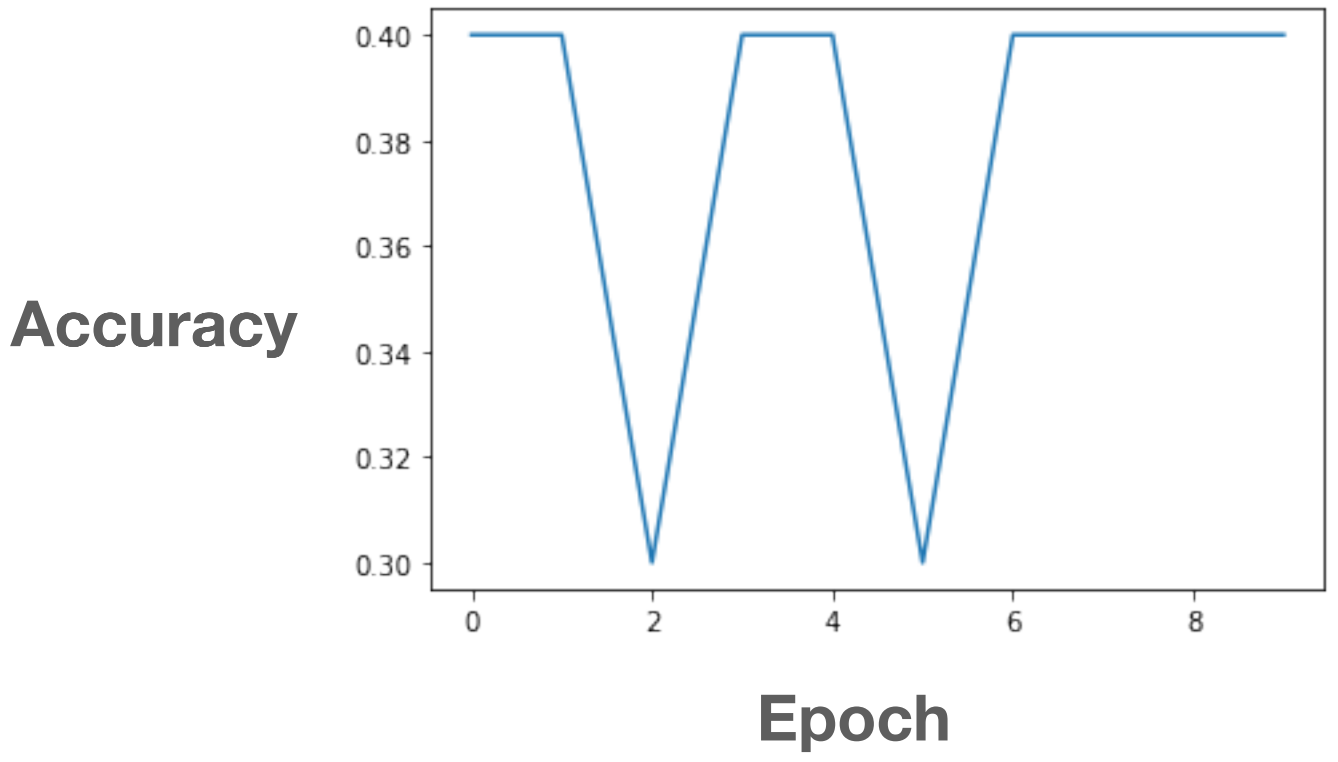
Test accuracy when inputs are 0-255



Train loss when inputs are 0-255*255=65025



Train accuracy when inputs are 0-255*255



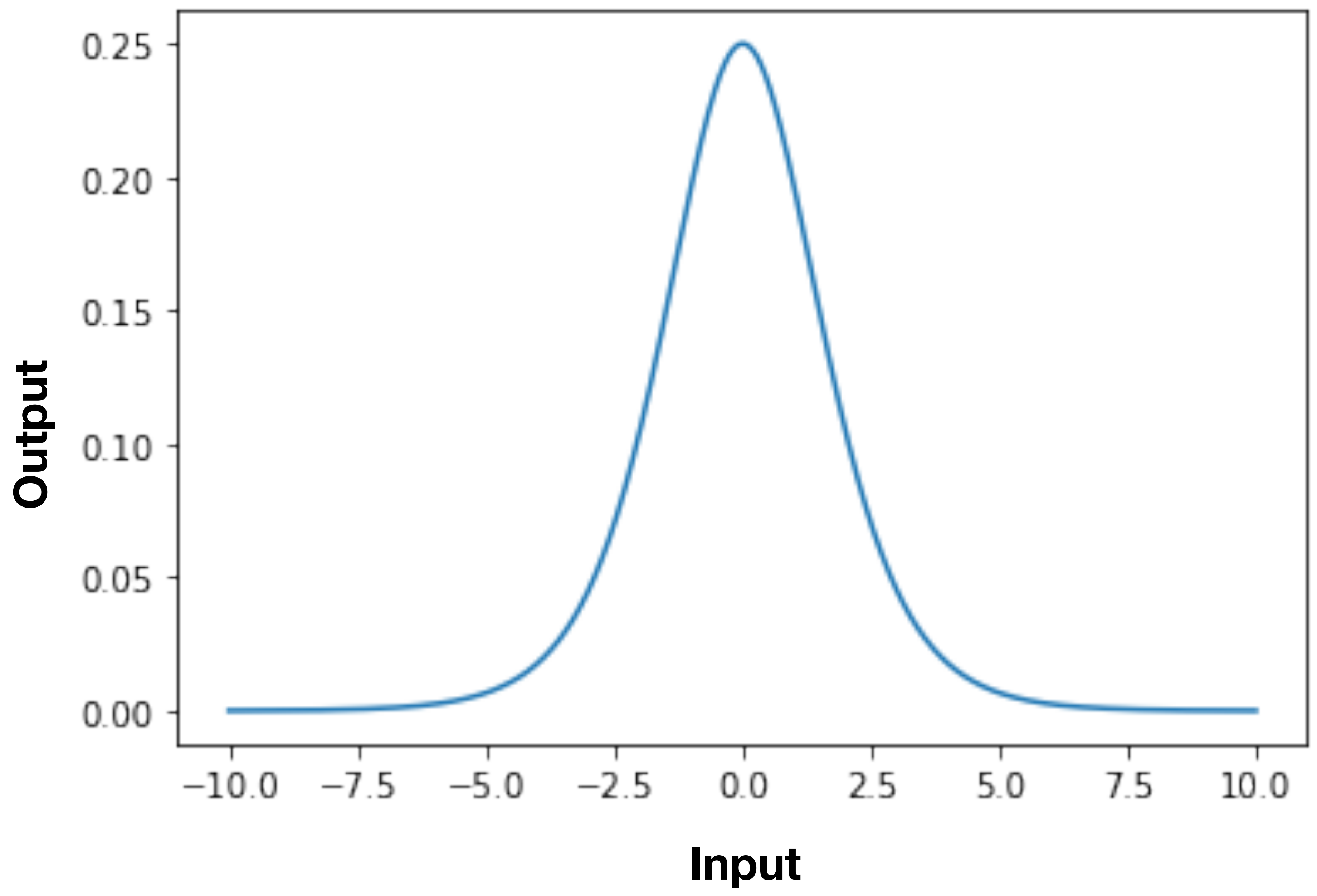
Test accuracy when inputs are 0-255*255

Data Normalization

- What happens when data is not normalised:
  - Take gradient descent when there is no squashing function at output
  - When inputs are large the outputs will be large
  - Take a model $\hat{y} = Wx + b$, a sample loss: $L = \frac{1}{2}(\hat{y} - y)^2$ and its derivative $\frac{\partial L}{\partial W} = (\hat{y} - y)x$
  - So when x is large $\frac{\partial L}{\partial W}$ will be large

Data Normalization

- The parameter update is given by $w^{k+1} = w^k - \alpha \frac{\partial L}{\partial W}$
- When the $\frac{\partial L}{\partial W}$ is very large, the update step would be very large
- This could overshoots the step and not converge towards a minima
- This is an exploding gradient issue
- If there is a squashing function like sigmoid, the gradients will vanish



Behaviour of the sigmoid derivative $\frac{\partial \sigma(x)}{\partial x}$ under different inputs

Data Normalization

- Data Normalization methods:
  - Min/Max normalisation:
    - Find the minimum and maximum of an input
    - Normalise the input X using $X_{normalized} = \frac{X - X_{min}}{X_{max} - X_{min}}$
  - Mean/Stdv normalisation (z-score method):
    - Why?: When trained neural nets learn an underlying distribution of the input data
    - By mean/stdv normalisation we make the distribution as simple as possible
    - First subtract the mean from data (zero centering)
    - The divide by standard deviation (scale the data)
    - $X_{normalized} = \frac{X - \mu}{\sigma}$
    - Centering prevents vanishing gradients and scaling improves the convergence speed

Data Normalization

- Batch normalisation:
  - Normalizes the inputs to a layer and applied on mini-batches
  - Implemented in very deep neural networks
  - Why?:
    - Take two consecutive layers l and $l+1$.
    - $l+1$ makes outputs based on the inputs from l
    - In backprop $l+1$ is updated before l but based on the inputs from l and then the l is updated
    - Now in the next iteration $l+1$ should model the distribution according to updated l
    - This issue makes $l+1$ chase a moving target
    - Therefore we need a standardisation of l layer inputs
  - Batch normalisation is applied only during the training process

Data Normalization

- Batch normalisation equations:
  - Input mini-batch: $B = [x_{1\ldots m}]$ where m is the batch size
  - Output: $y_i = BN_{\gamma,\beta}(x_i)$ where γ, β are trainable parameters (backprop learning)
  - γ, β trained to find the optimal distribution that minimises the error
  - Computation
    1. Find mean of the mini-batch: $\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$
    2. Find the variance of the mini-batch: $\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_B)^2$
    3. Compute the normalised output: $\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$ where ϵ is a regulariser to prevent division by "0"
    4. Shift and scale the normalised output for optimal distribution: $y_i = \gamma \hat{x}_i + \beta = BN_{\gamma,\beta}(x_i)$

Reference

- Ref.1:https://commons.wikimedia.org/wiki/File:MultiLayerPerceptron.svg
- Ref.2: https://medium.com/analytics-vidhya/concepts-of-differential-calculus-for-understanding-derivation-of-gradient-descent-in-linear-de59a17496a3
- Ref.3: https://slideplayer.com/slide/4916524/
- Ref.4: https://de.wikipedia.org/wiki/Datei:Sigmoid-function-2.svg