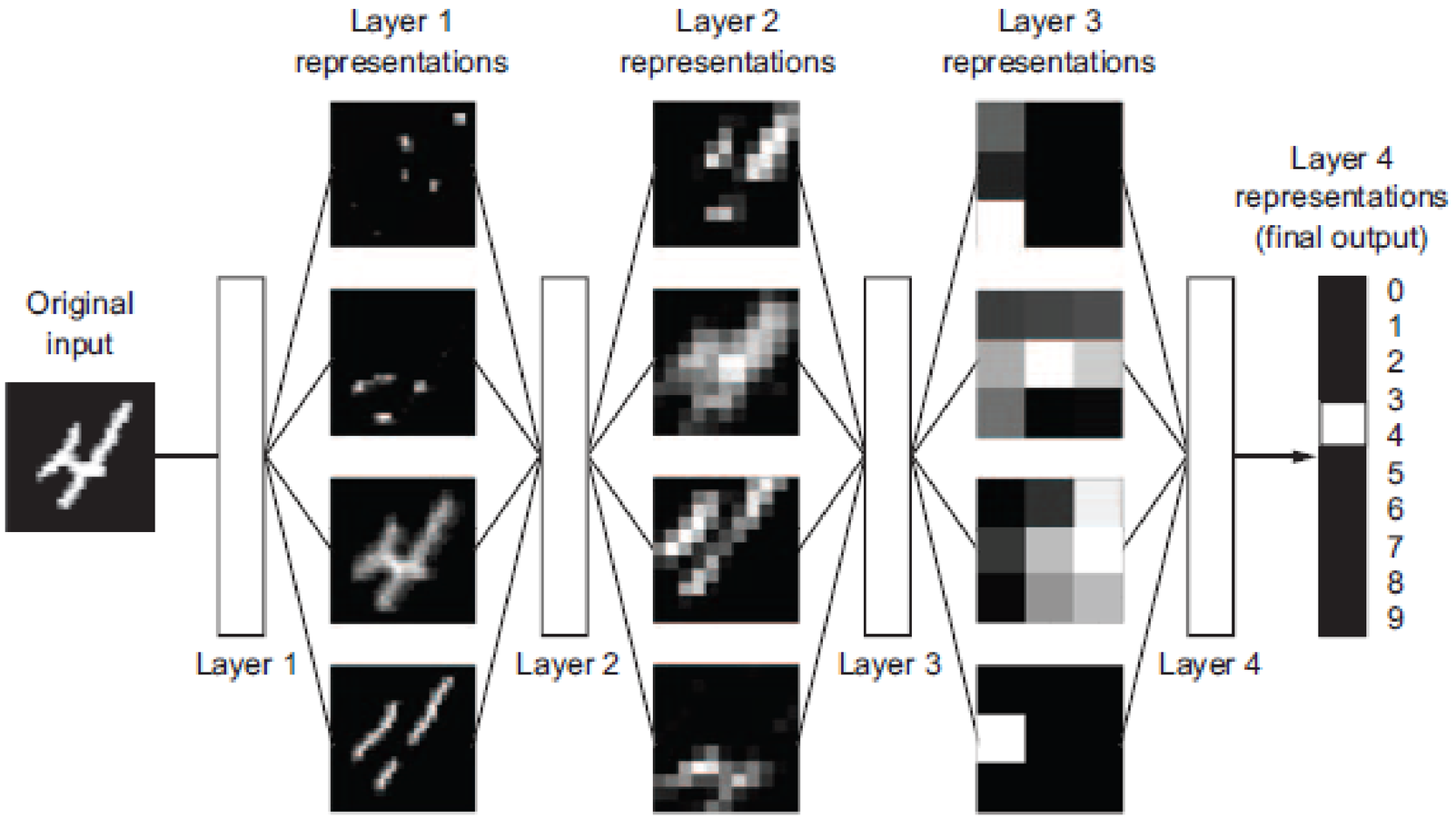
Neural Networks Model Development

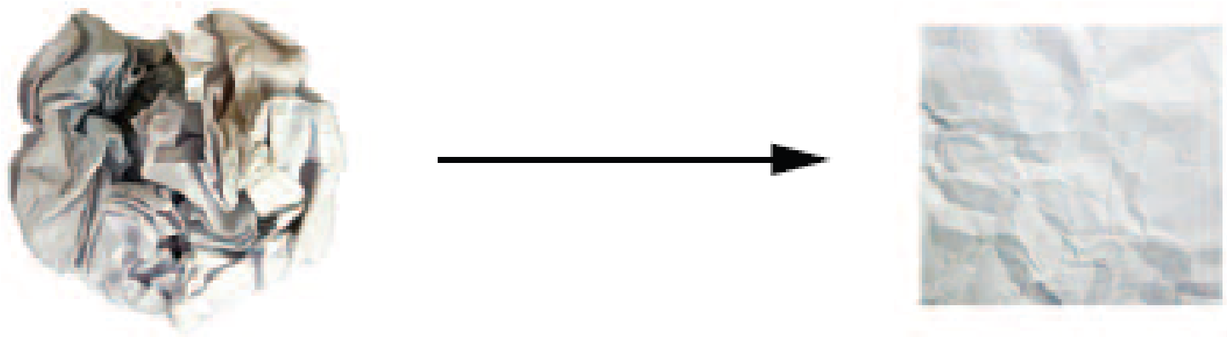
Instructor:
Revendranath T

Motivations to Build Neural Networks Model



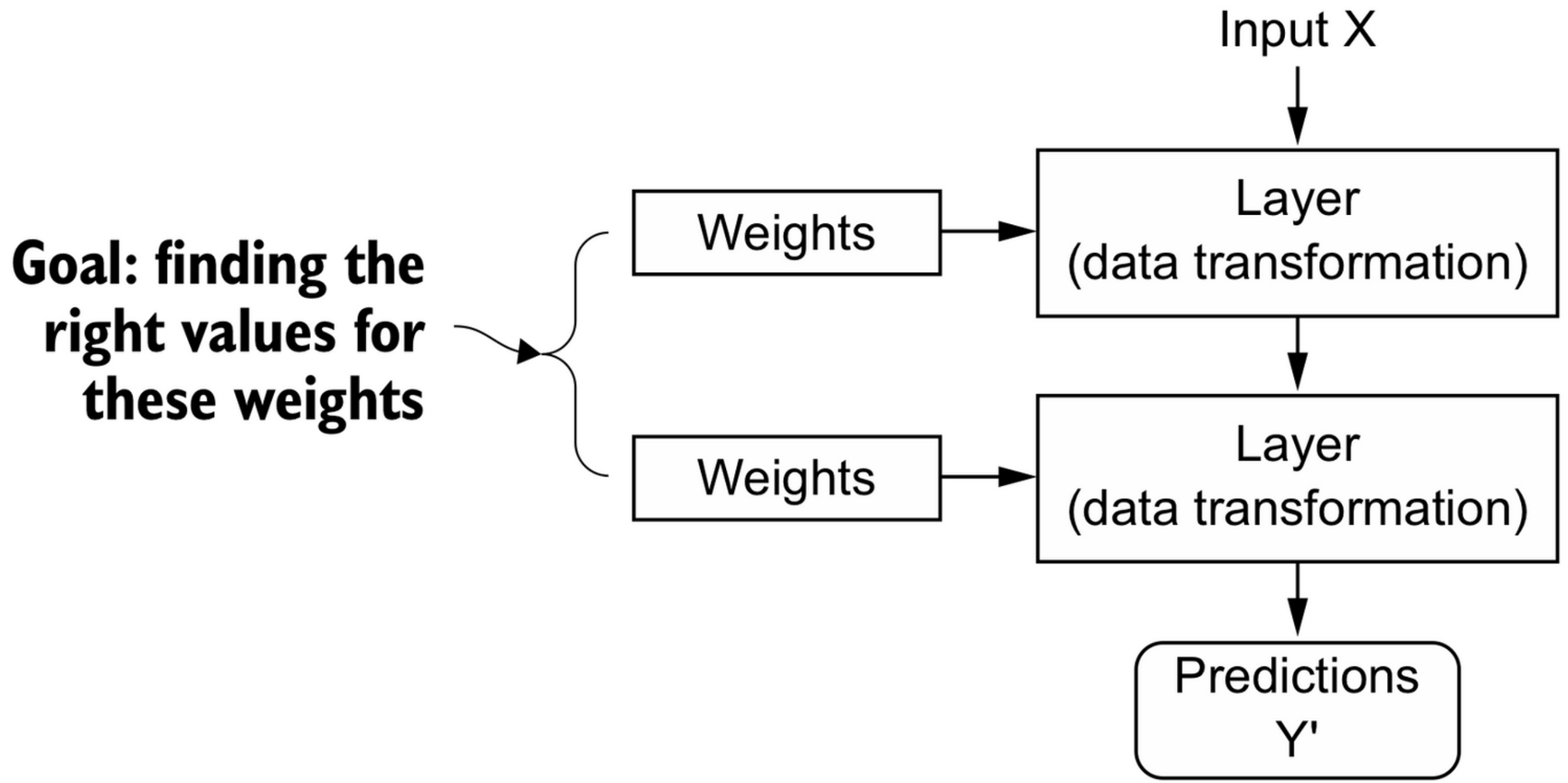
Pattern recognition ability in each layer

Motivations to Build Neural Networks Model

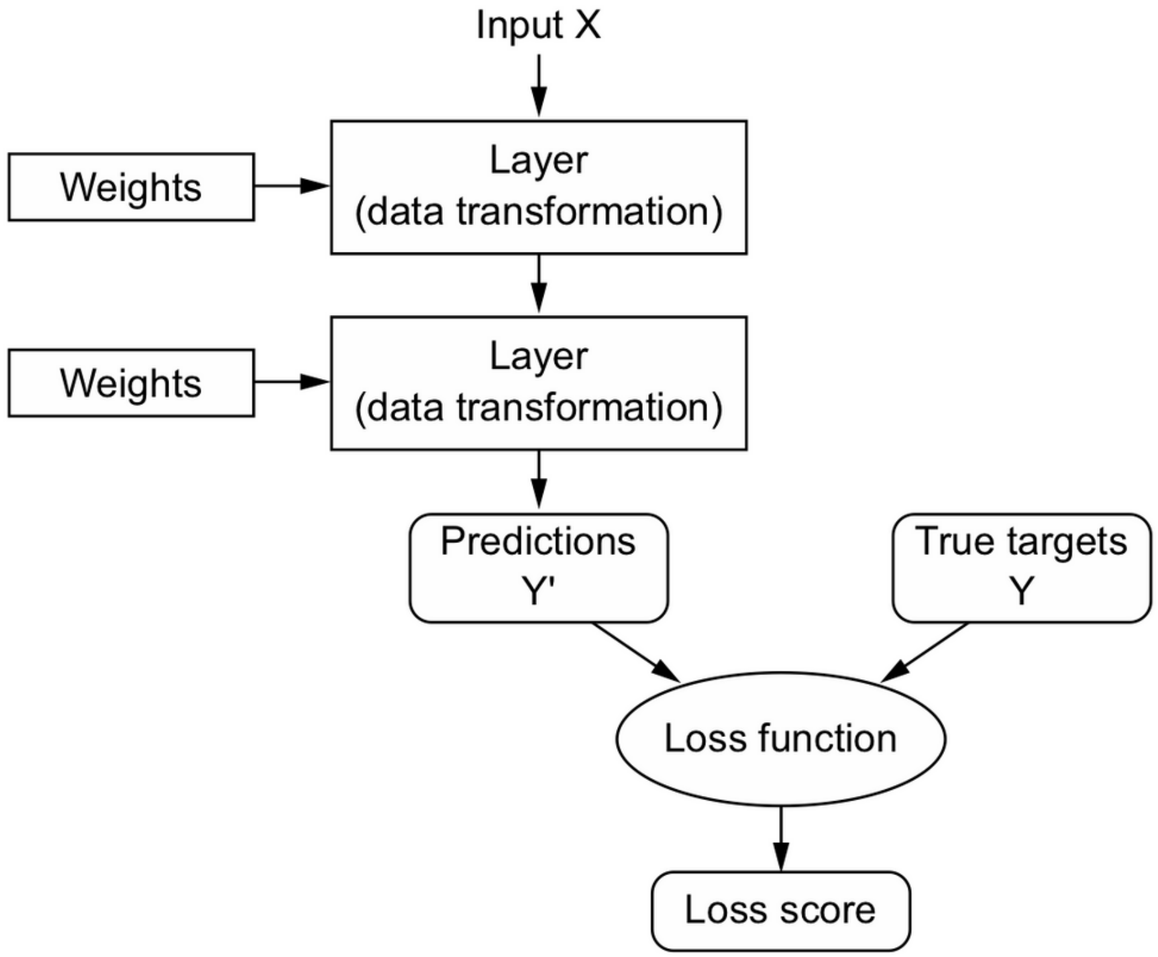


Uncrumpling a
complicated manifold of data

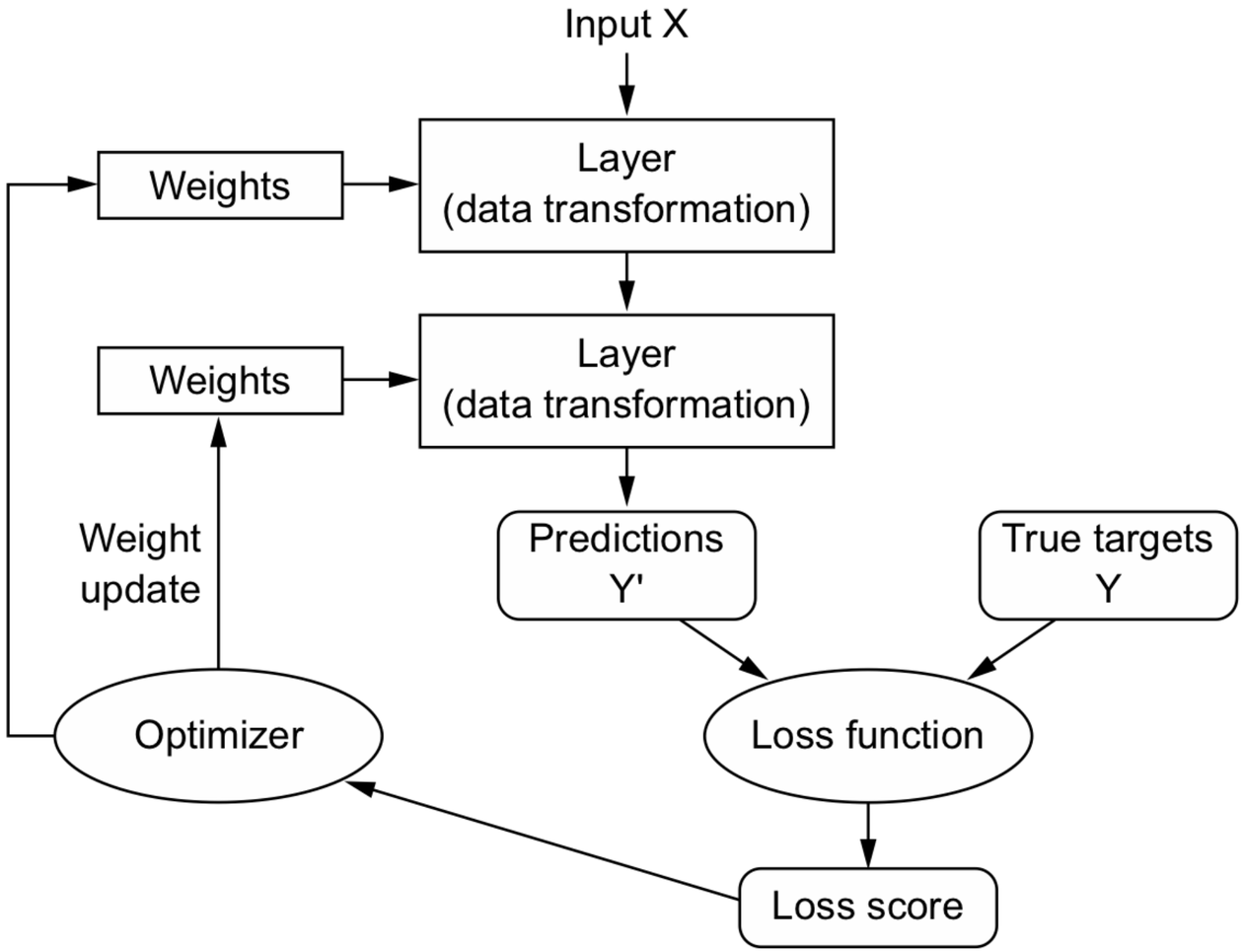
Find weights. How?



How to reduce loss function?

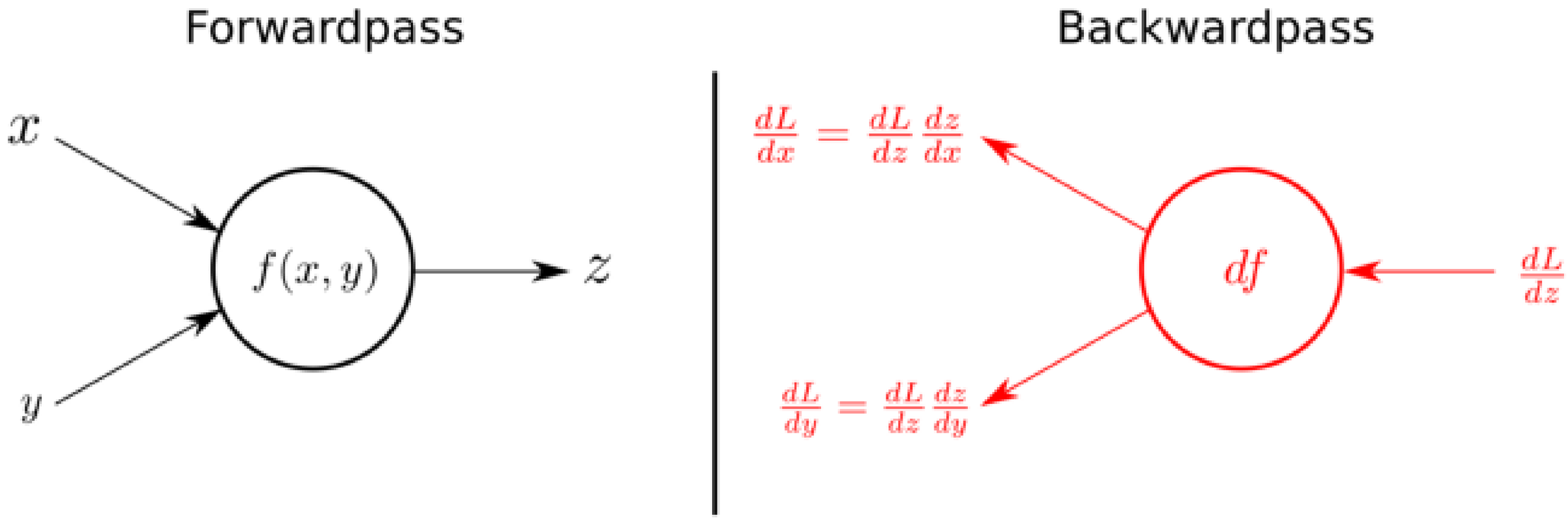


Gradient-based optimization to reduce loss function?



Gradient-based optimization is the enginer of neural networks

Forward Pass & Backward Pass

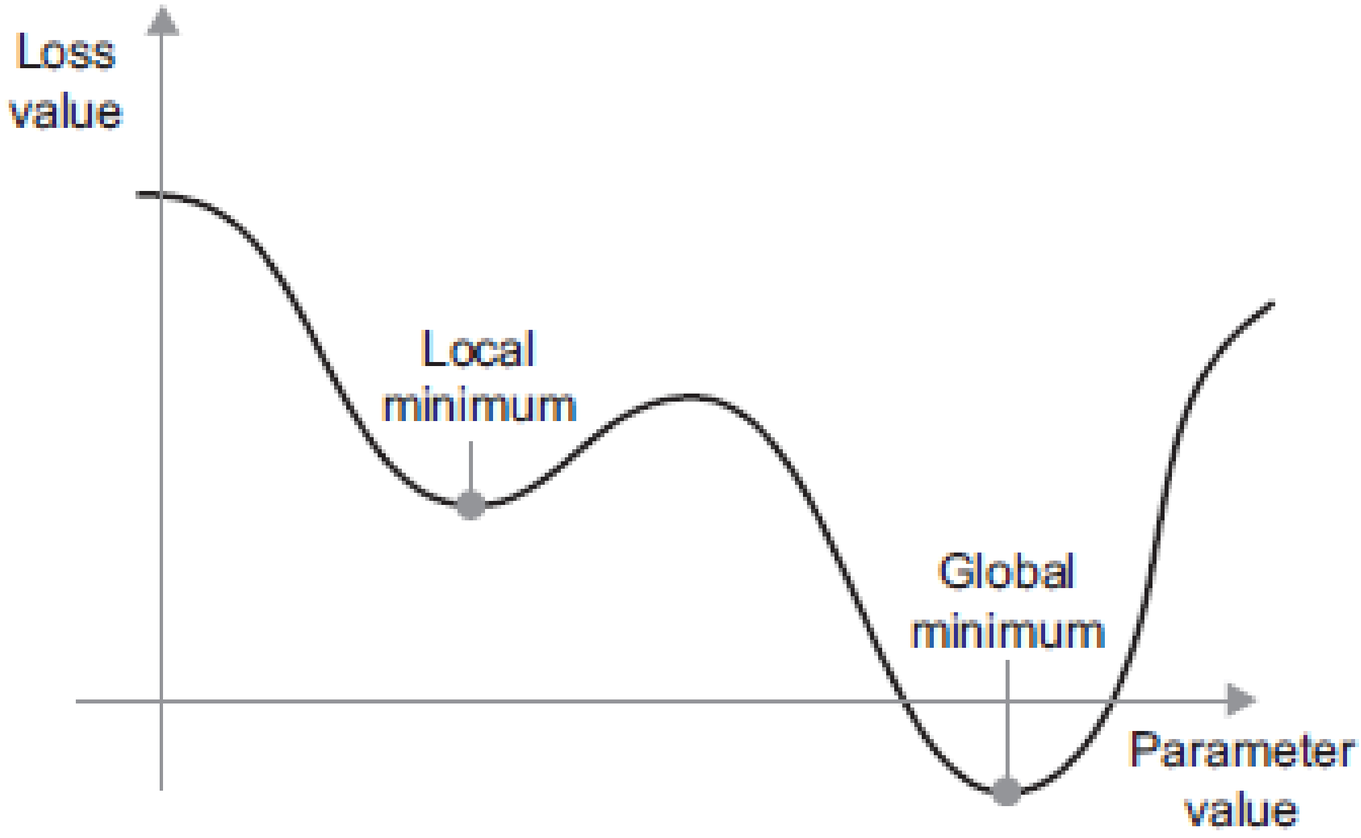


Steps in Training

1. Draw a **batch** of training samples **x** and corresponding targets **y**.
2. Run the network on **x** (a step called the forward pass) to obtain predictions **y_pred**.
3. Compute the **loss** of the network on the **batch**, a measure of the mismatch between **y_pred** and **y**.
4. **Update** all **weights** of the network in a way that slightly **reduces** the loss on this **batch**.

The Gradient: Derivative of a tensor operation

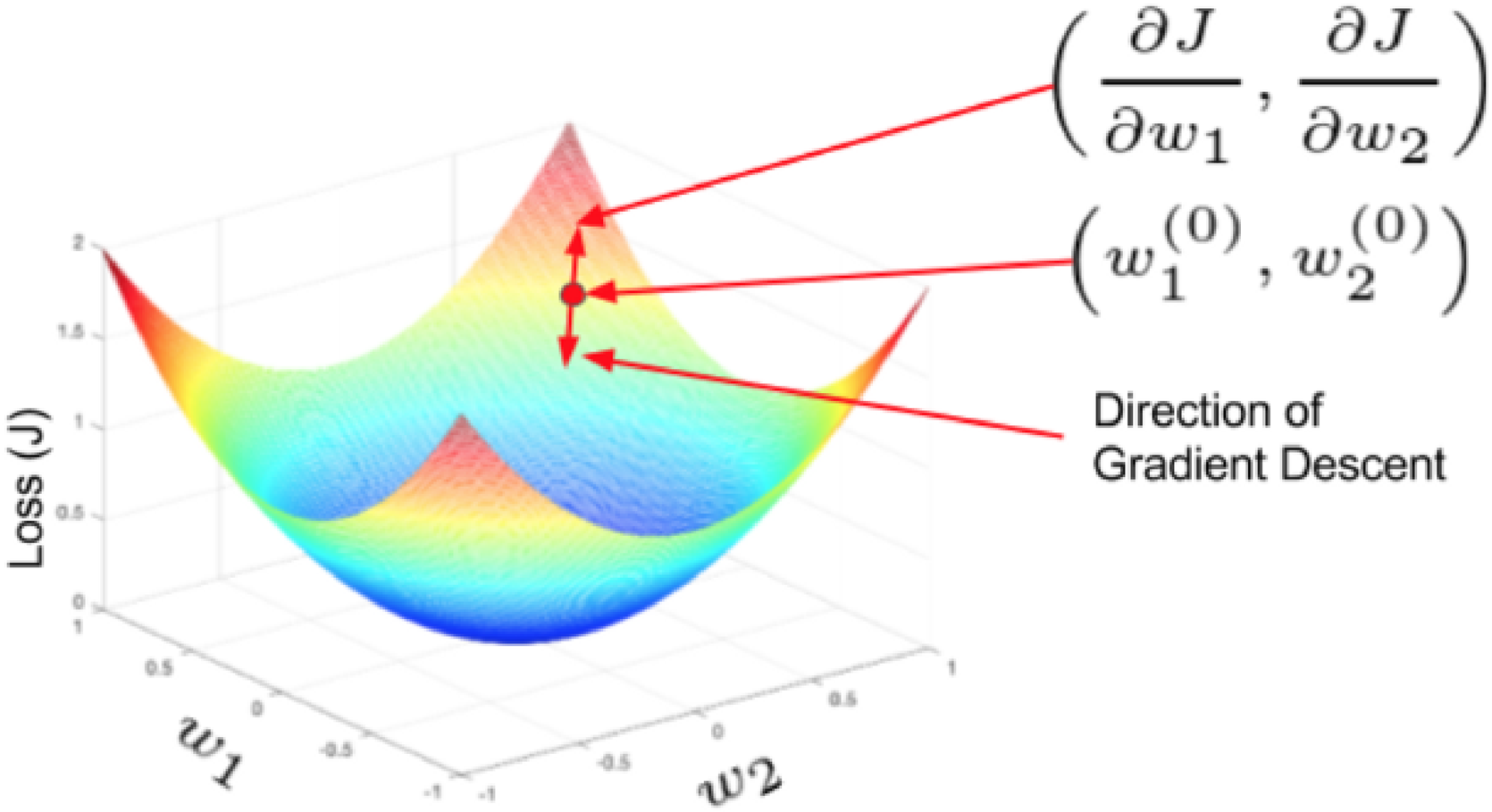
1. Gradient is the generalization of the concept of derivatives to functions of multidimensional inputs: that is, to functions that take tensors as inputs.



Mini-batch stochastic gradient descent (minibatch SGD)

1. Draw a **batch** of training samples **x** and corresponding targets **y**.
2. Run the network on **x** to obtain predictions **y_pred**.
3. Compute the **loss** of the network on the **batch**, a measure of the **mismatch** between **y_pred** and **y**.
4. Compute the **gradient of the loss** with regard to the network's parameters (a backward pass).
5. Move the parameters a little in the **opposite direction** from the gradient—for example **W -= step * gradient**—thus reducing the loss on the batch a bit

2-D representation of gradient descent



Gradient Descent

- Uses efficient methods to compute gradients automatically in two passes through the neural network
- One forward pass & one backward pass
- The back-propagation algorithm computes the gradient of the neural network's error on every single model parameter.
- Back-propagation process finds out how much the weights and biases of neural networks per layer should be adjusted to reduce the error and makes adjustments.
- Each step of the back-propagation is termed as Gradient Descent step, and whole process is repeated till the model converges
- Automatically computing gradients is called automatic differentiation (autodiff)

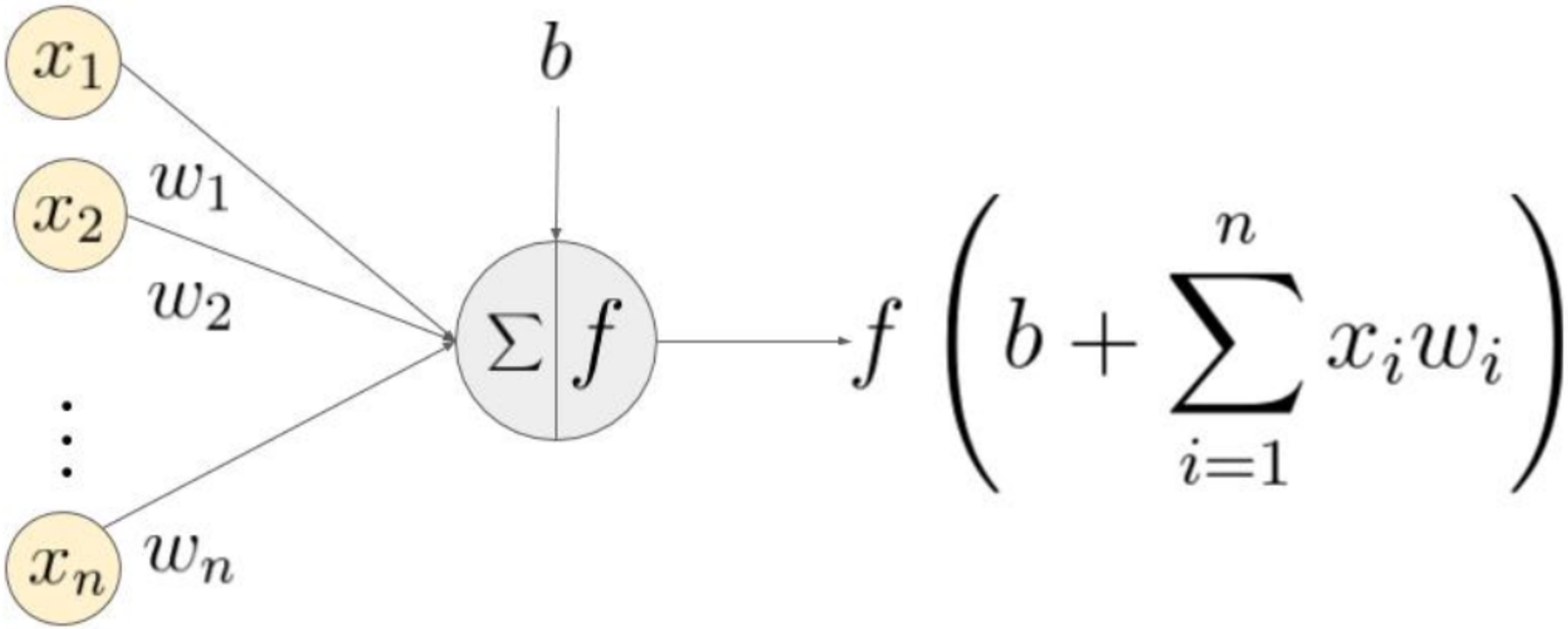
Back-propagation Algorithm

- Handles one mini-batch at a time (for example, containing 32 instances each), and goes through the full training set multiple times. Each pass is called an **epoch**.
- **Forward pass**: Each mini-batch is passed to the network's input layer, which sends it to the first hidden layer and so on till the output layer final values are calculated (for every instance in the mini-batch).
  - Forward pass enables prediction.
  - All intermediate results at every layer are preserved (as they are required for backward pass)
- The algorithm measures the neural network's output error (y_actual - y_pred) to identify how much each output connection contributed to the error

Back-propagation Algorithm

- Chain rule is applied to calculate the contribution of each layer to the error.
- Chain rule is nothing but applying differentiation to the error component.
- **Backward pass**: The algorithm uses the error from the output layer, calculates error contribution from each layer, and makes corrections in the backward direction to the layers until the input layer is reached.
- This process continues till the model converges.

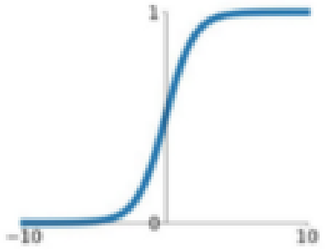
Neural Networks & Activation Function



Activation Functions

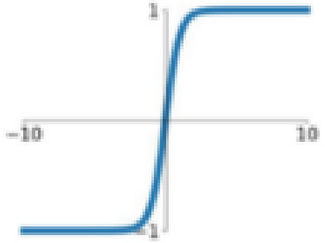
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



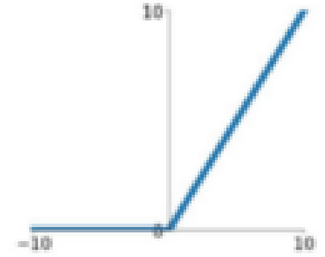
tanh

$$\tanh(x)$$



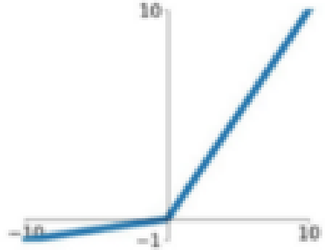
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

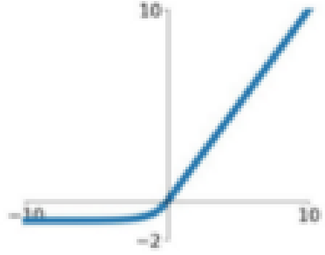


Maxout

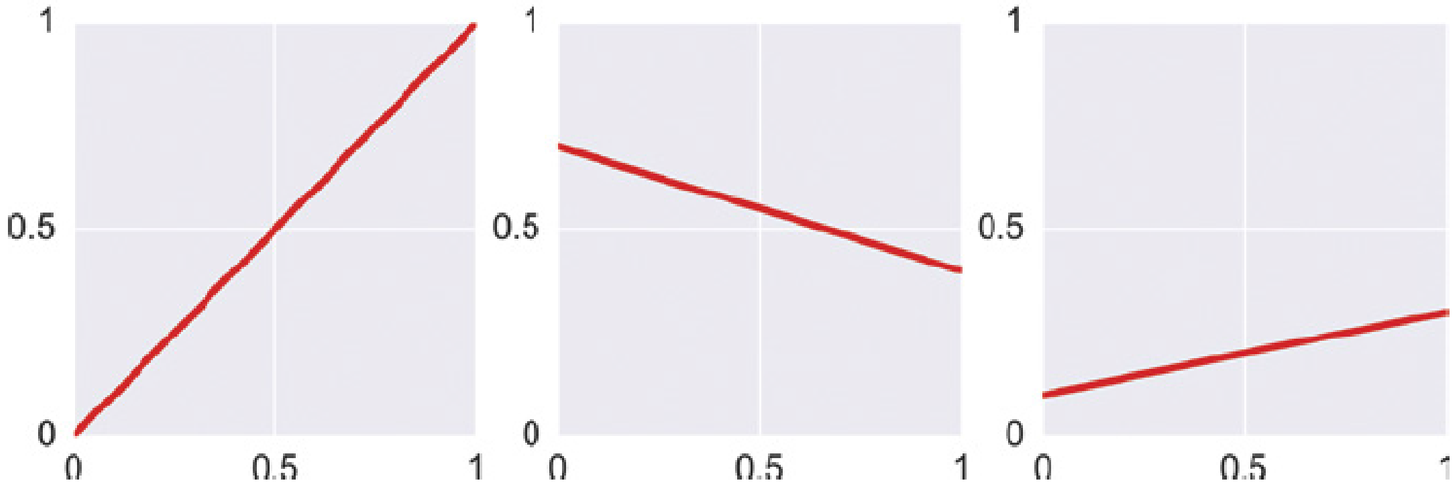
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



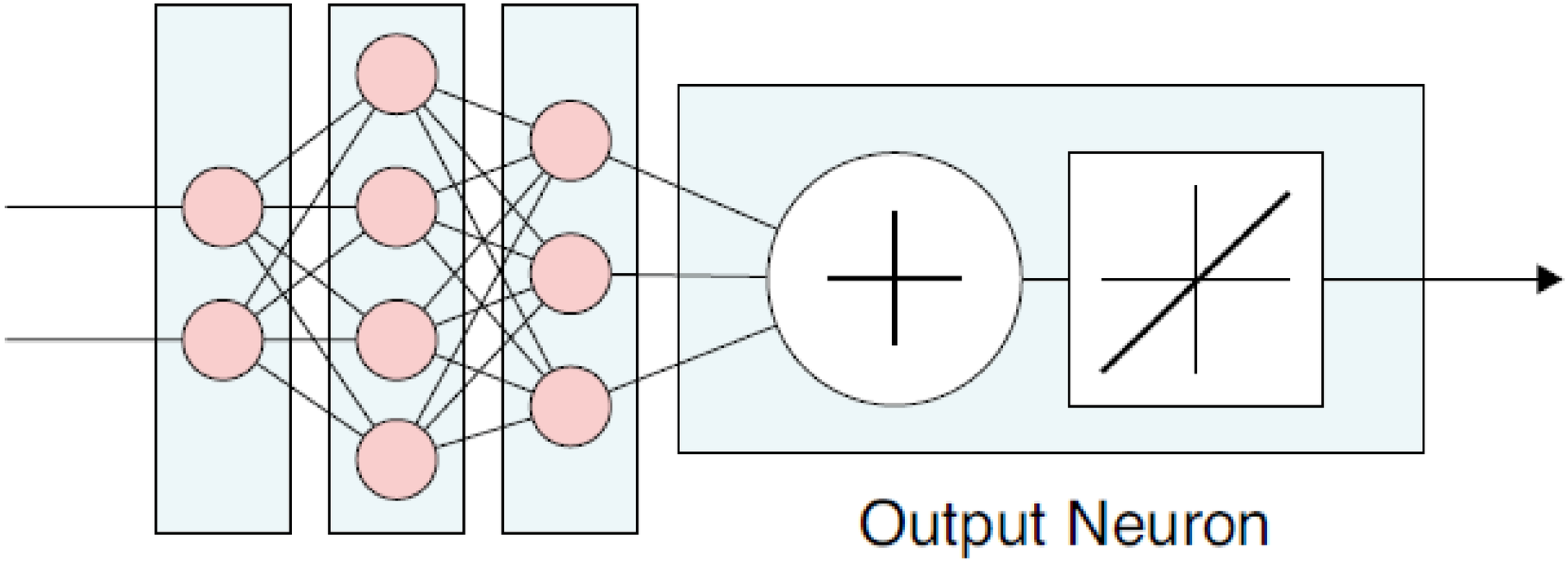
Straight Line Functions



Identify Function Linear Curve

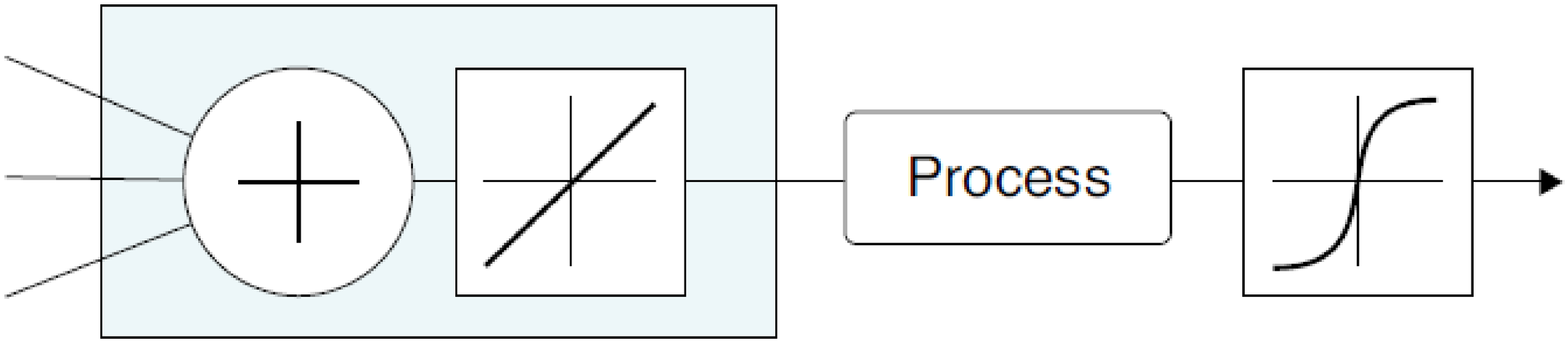
- Do not prevent network collapse
- Used at neuron output where the network does not collapse
- Used to insert some processing between the summation step in a
- neuron and its activation function

Applications of Straight Line Activation Function



Neuron output where the network does not collapse

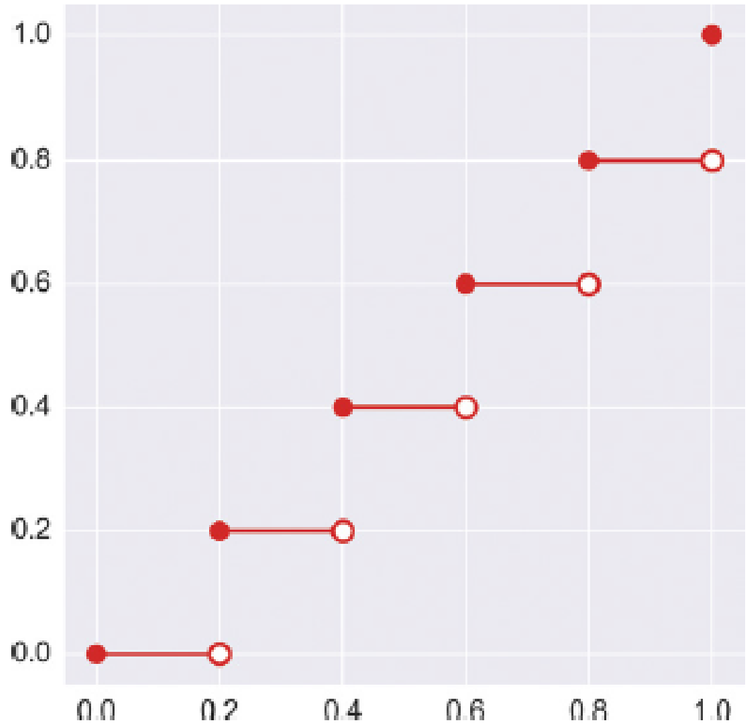
Applications of Straight Line Activation Function

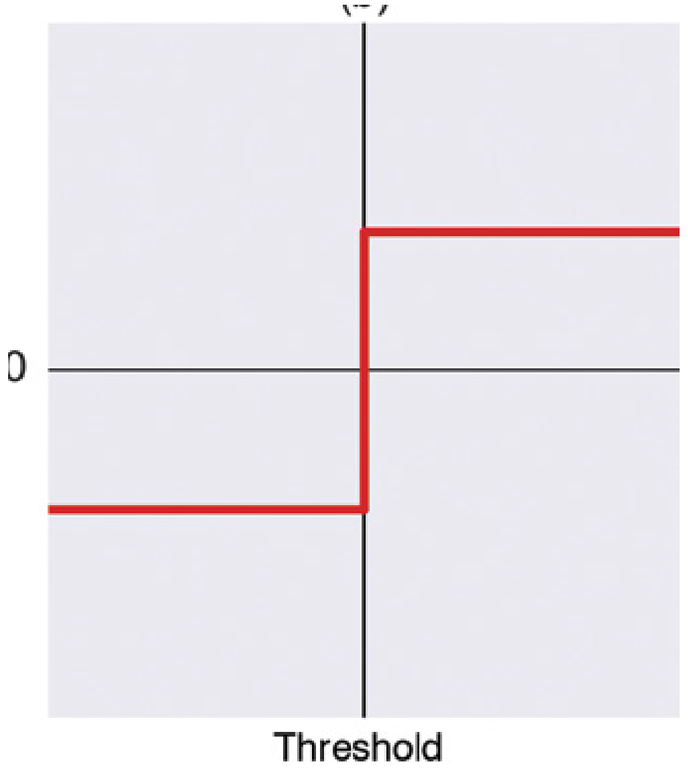


Insert some processing between the summation step in a neuron and its activation function

Step Functions

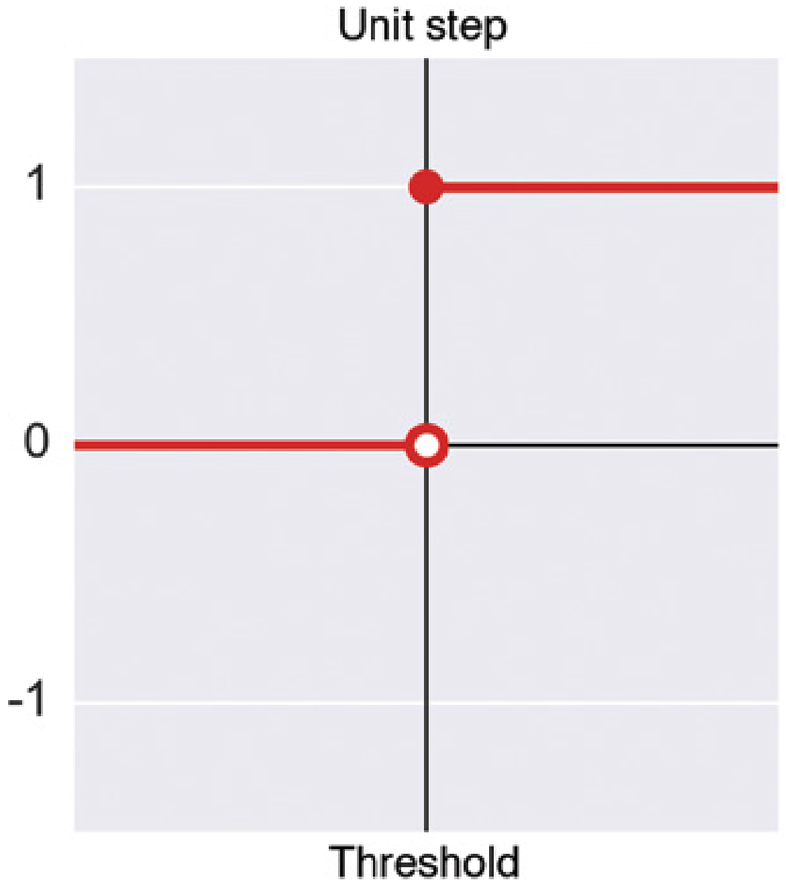
- Require a straight line, that needs to be elevated at each step
- Stair-case step function
- Step function

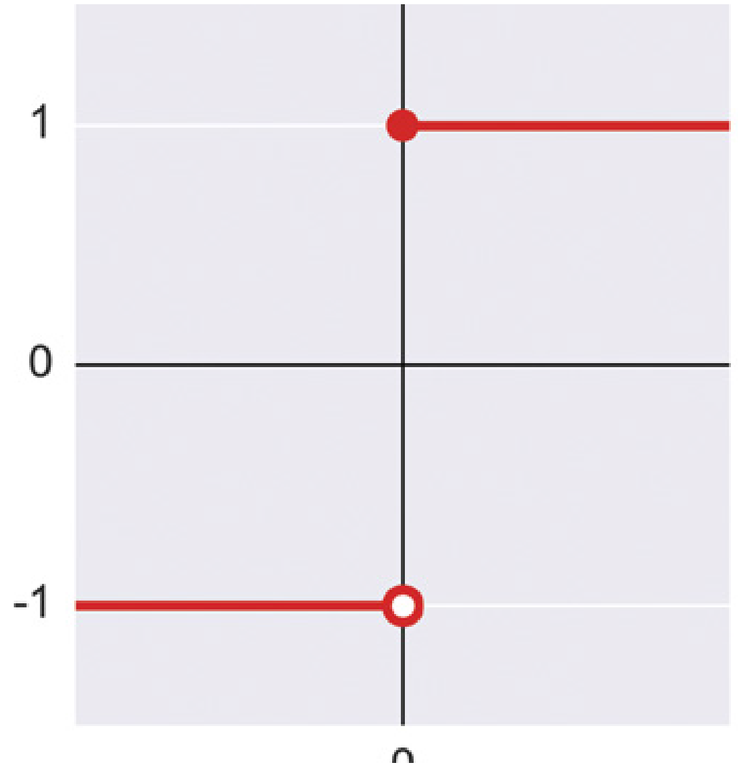




Step Functions

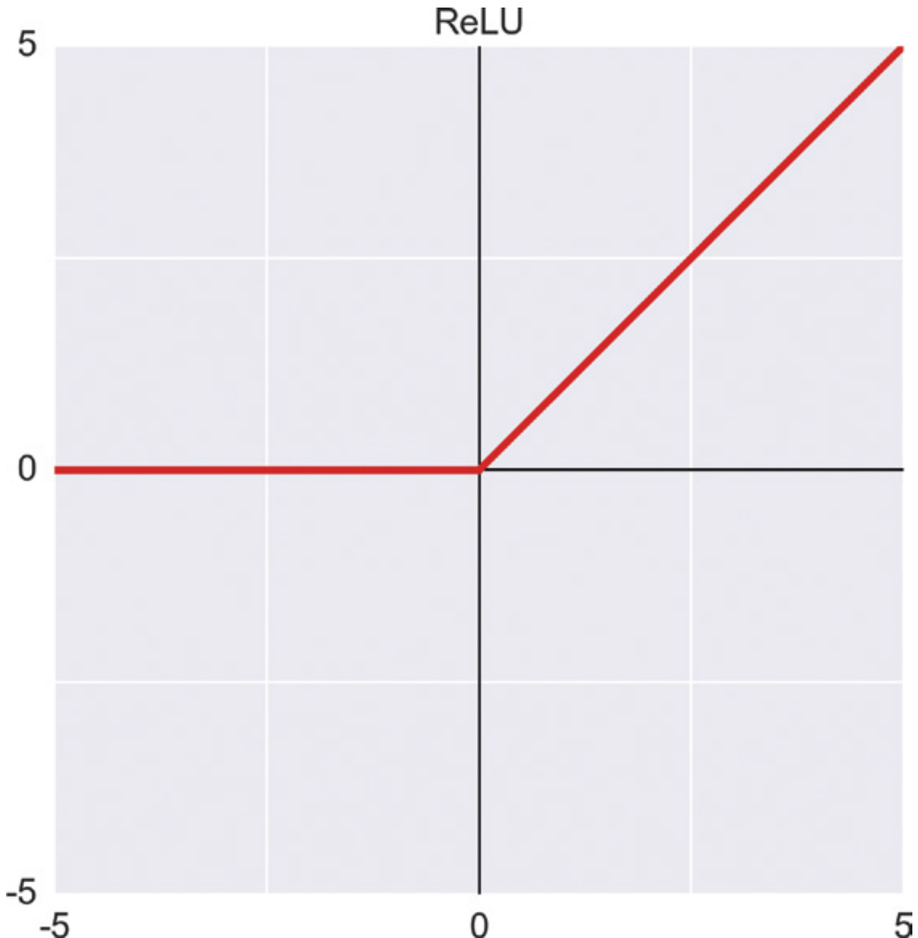
- Unit step function
- Sign step function

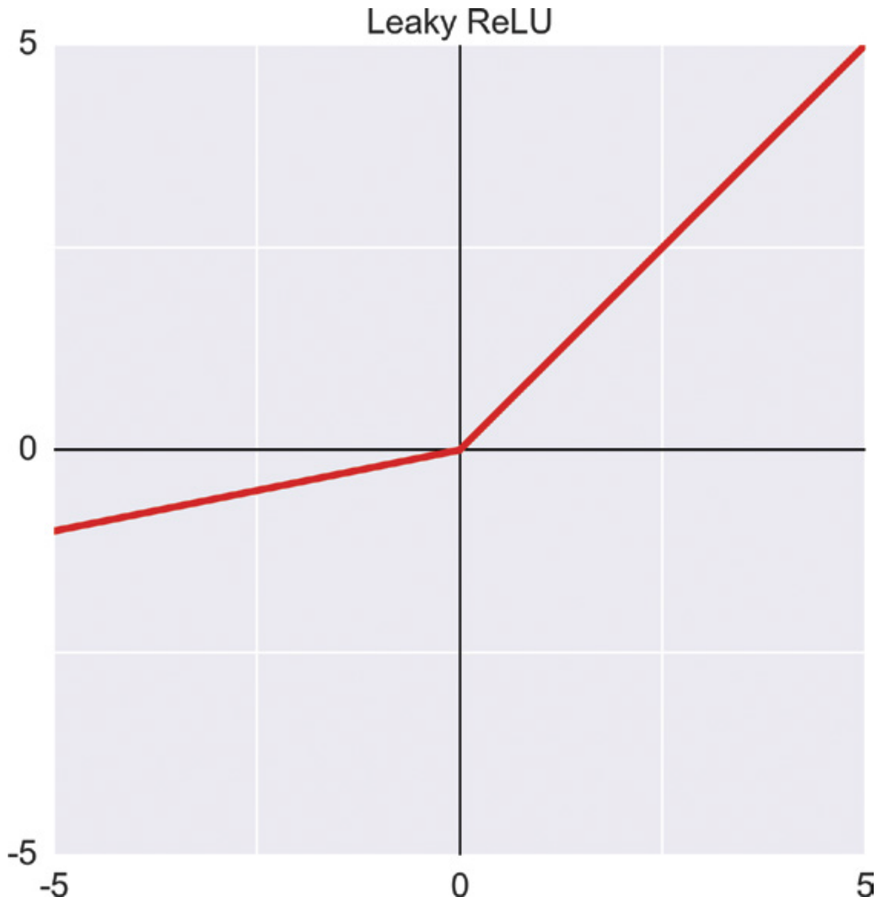




Piecewise Linear Functions

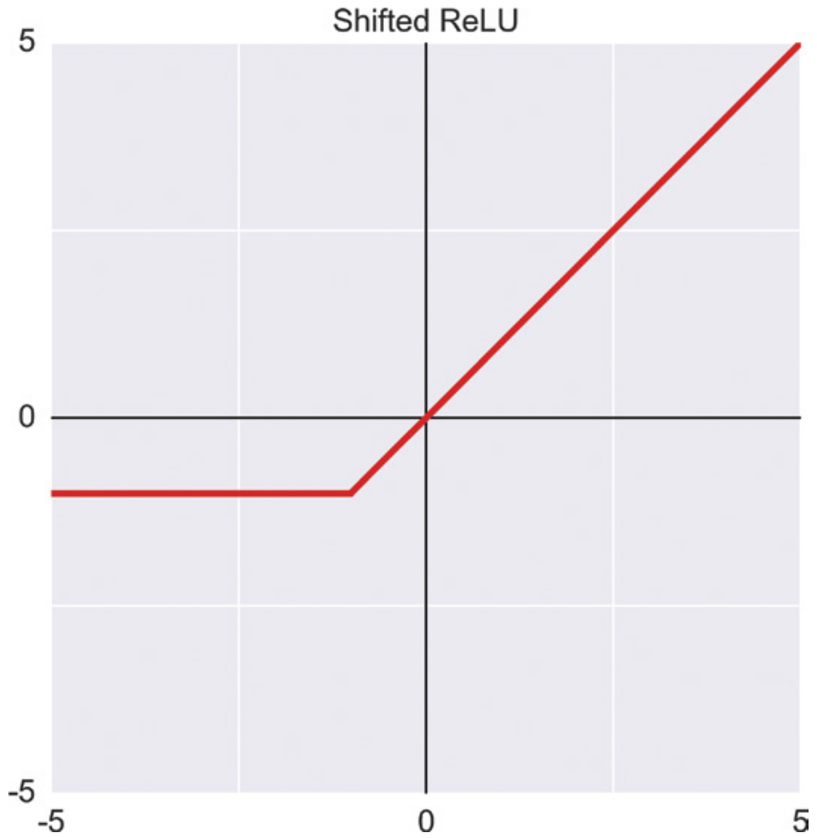
- ReLU: rectified linear unit
- Leaky ReLU: When small negative values are required



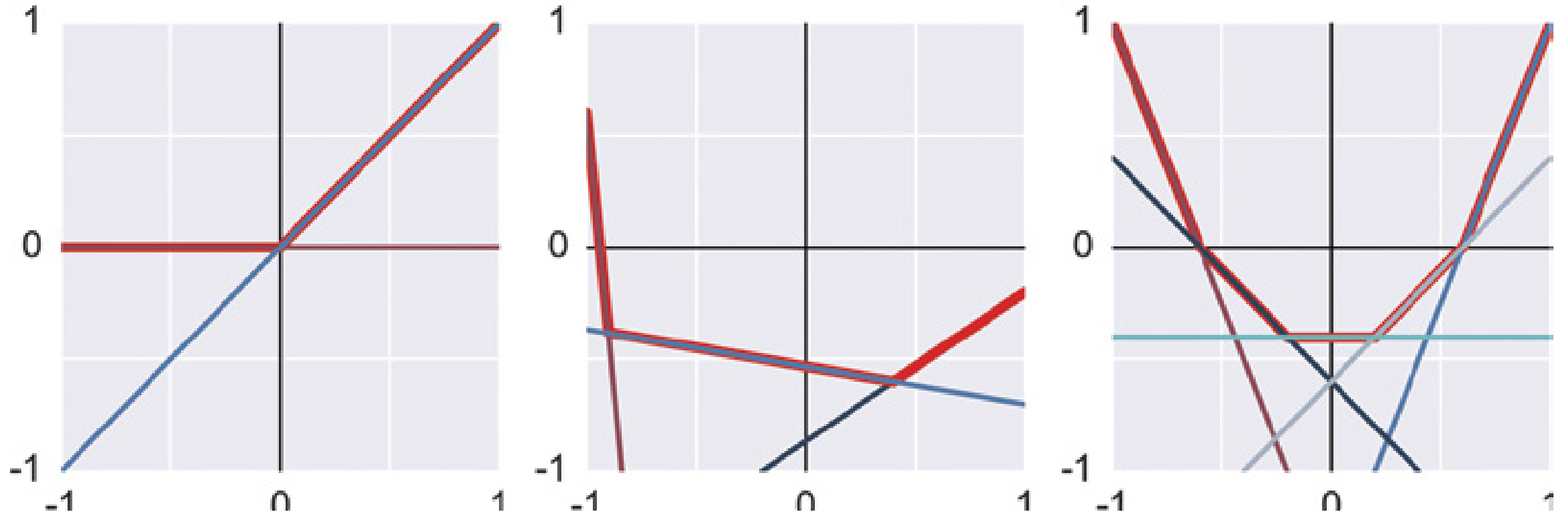


Piecewise Linear Functions

Shifted ReLu



Maxout ReLu with two or more lines

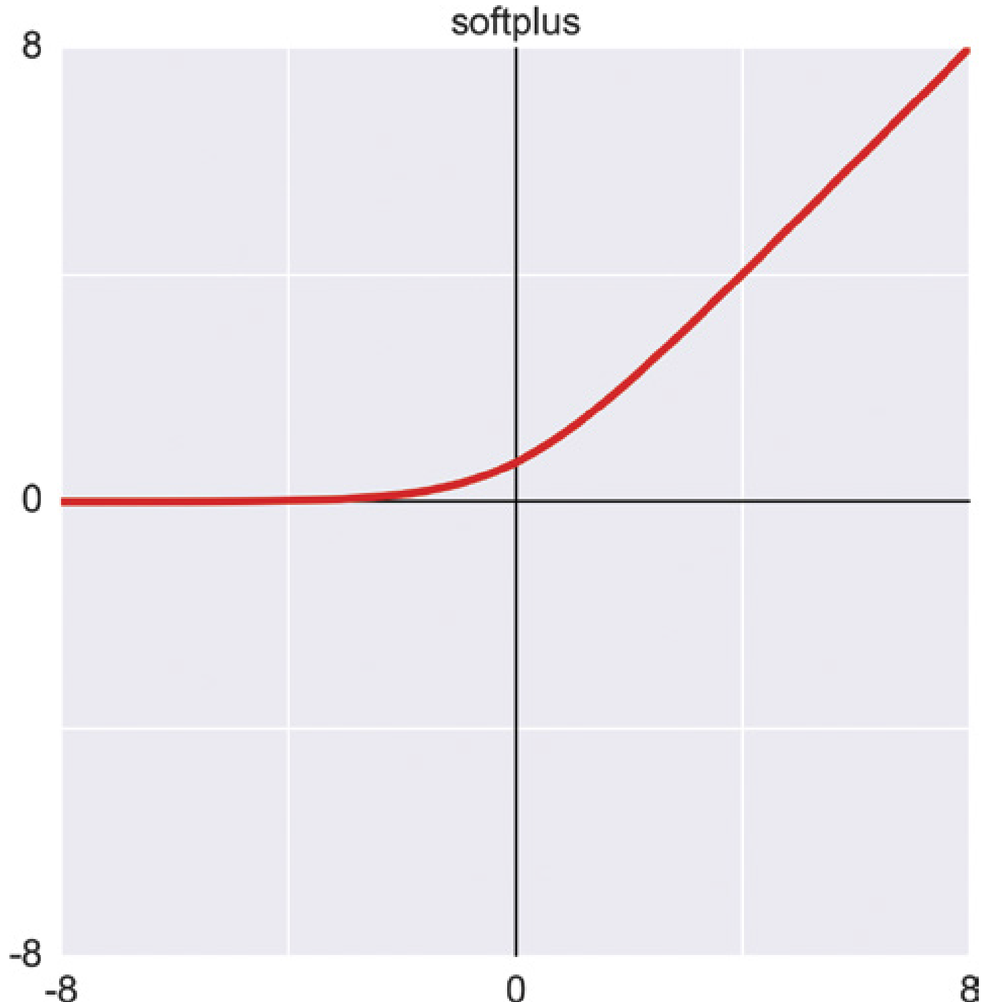


Smooth Functions

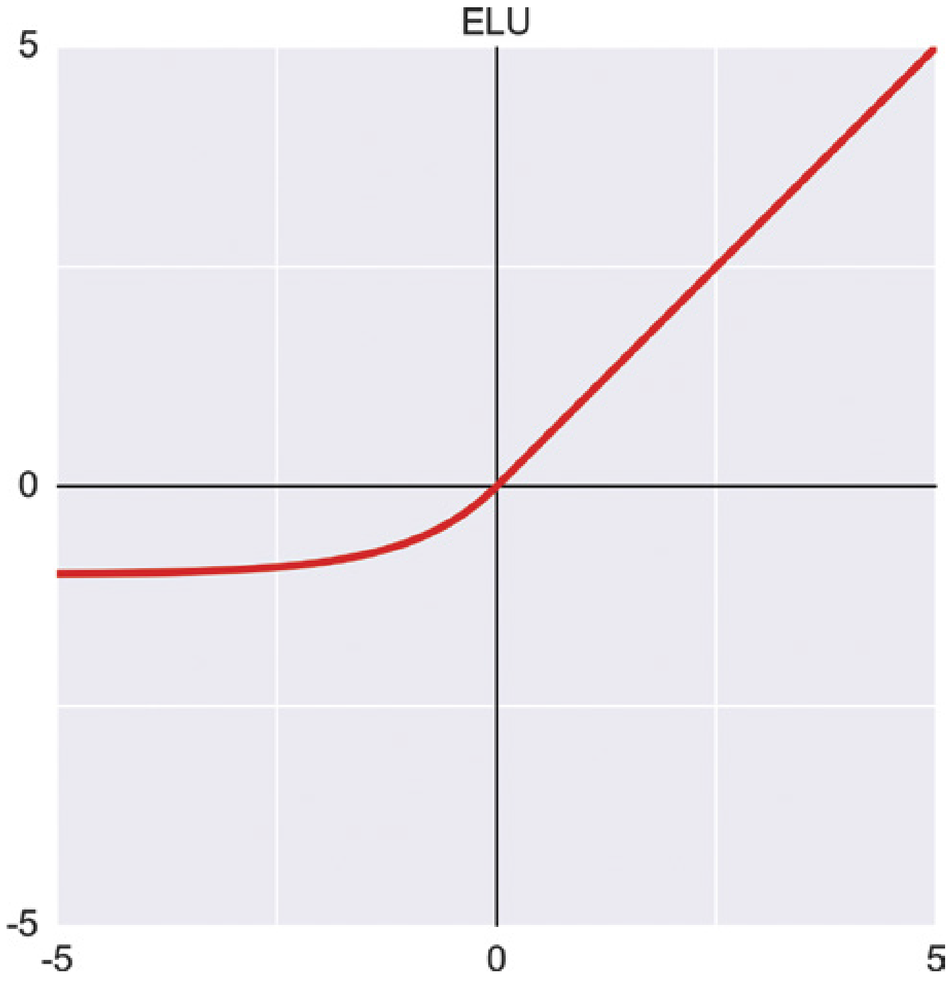
- The straight line, step function and piecewise linear activation functions have edges that make it challenging to arrive at derivatives.
- Leads to complexity in training models.
- Smoothness at the edges enables better mathematical derivatives
- Smooth functions that inherently have a derivative everywhere. i.e., they’re smooth everywhere.
- Softplus and Exponential ReLU smoothes the ReLU activation functions

Smooth Functions

Softplus

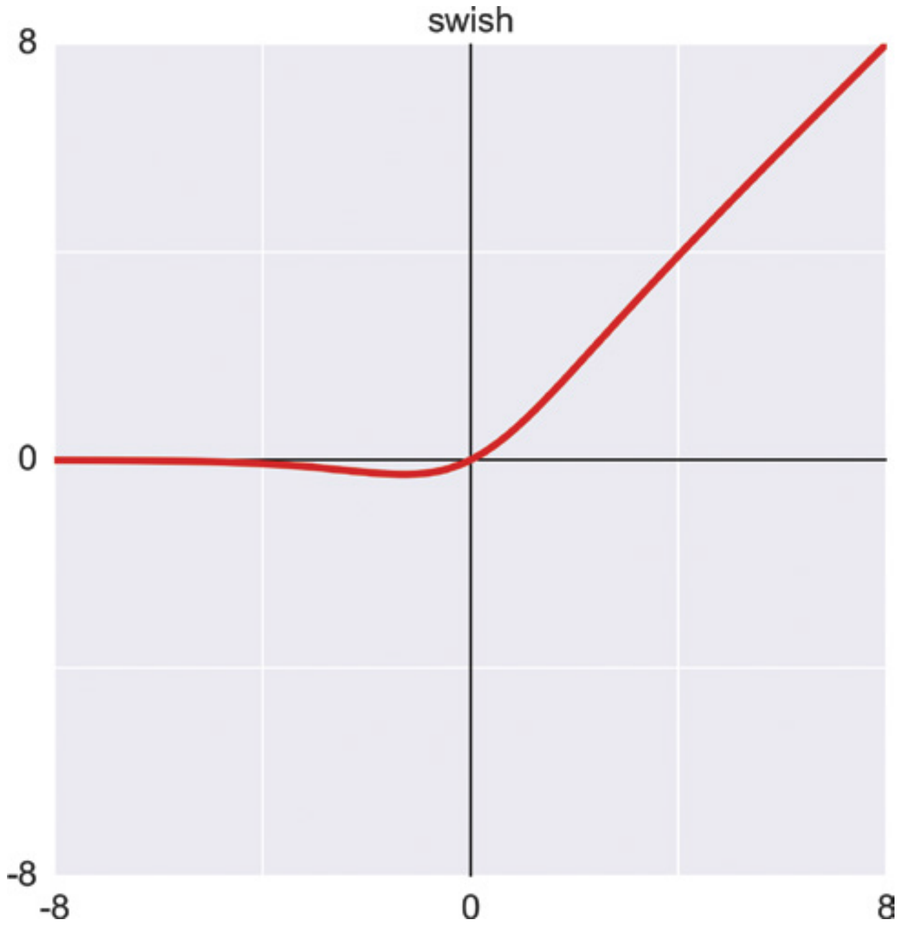


Exponential ReLu

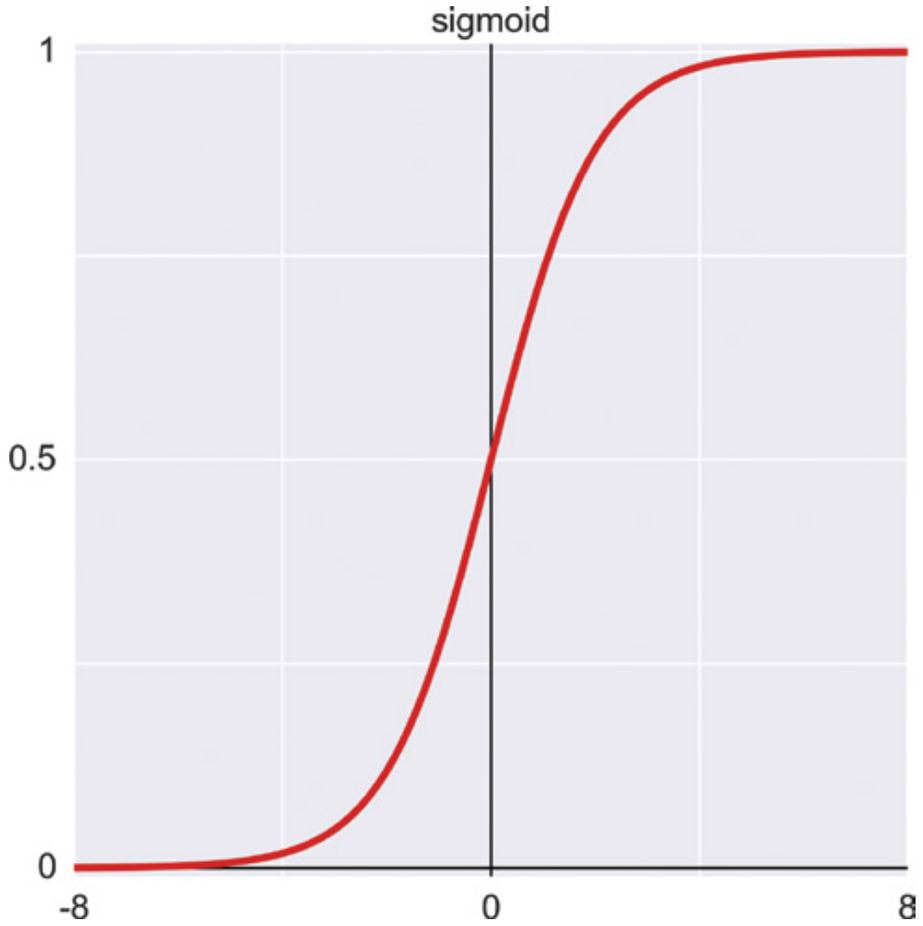


Smooth Functions

Swish

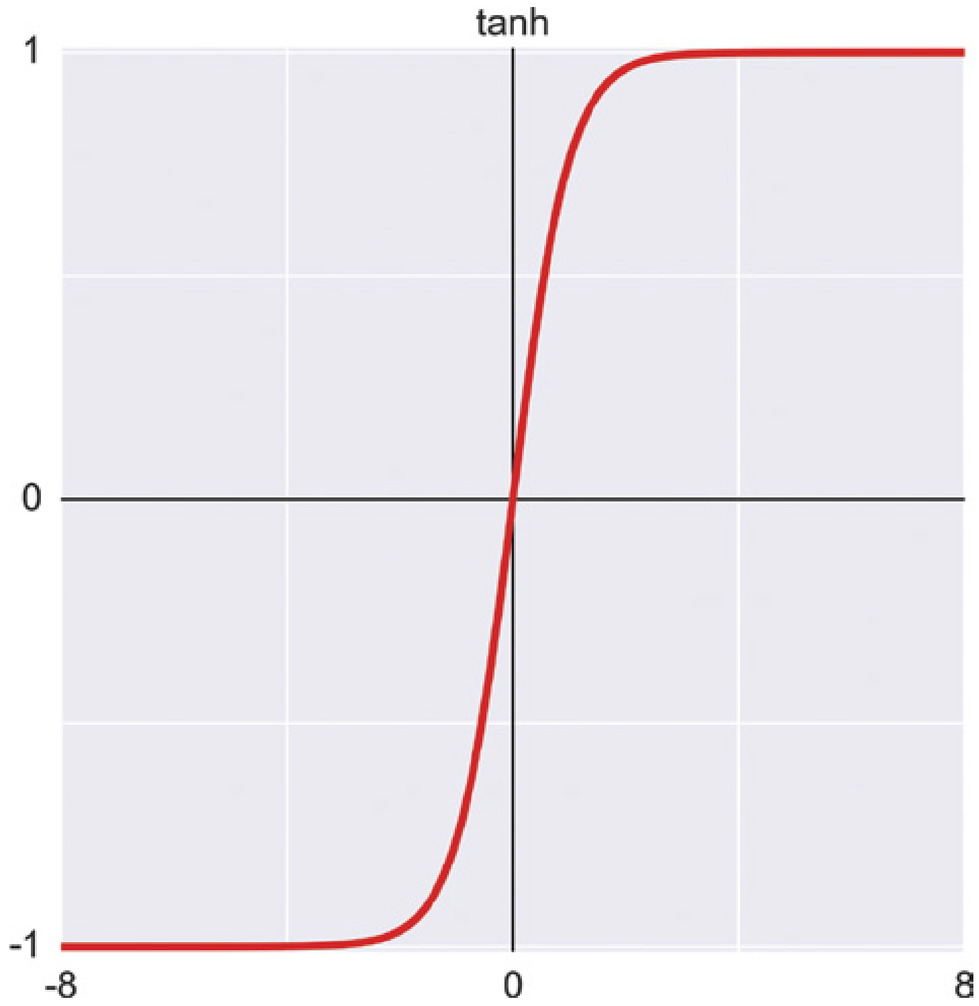


Sigmoid

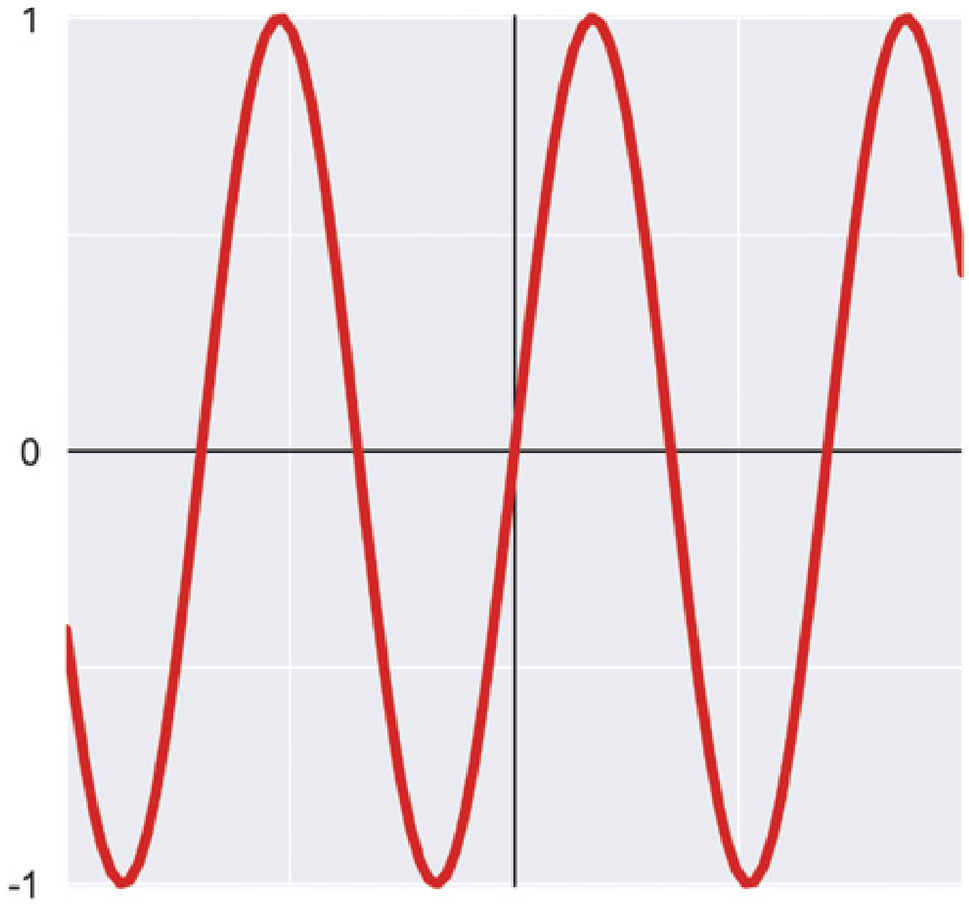


Smooth Functions

tanh

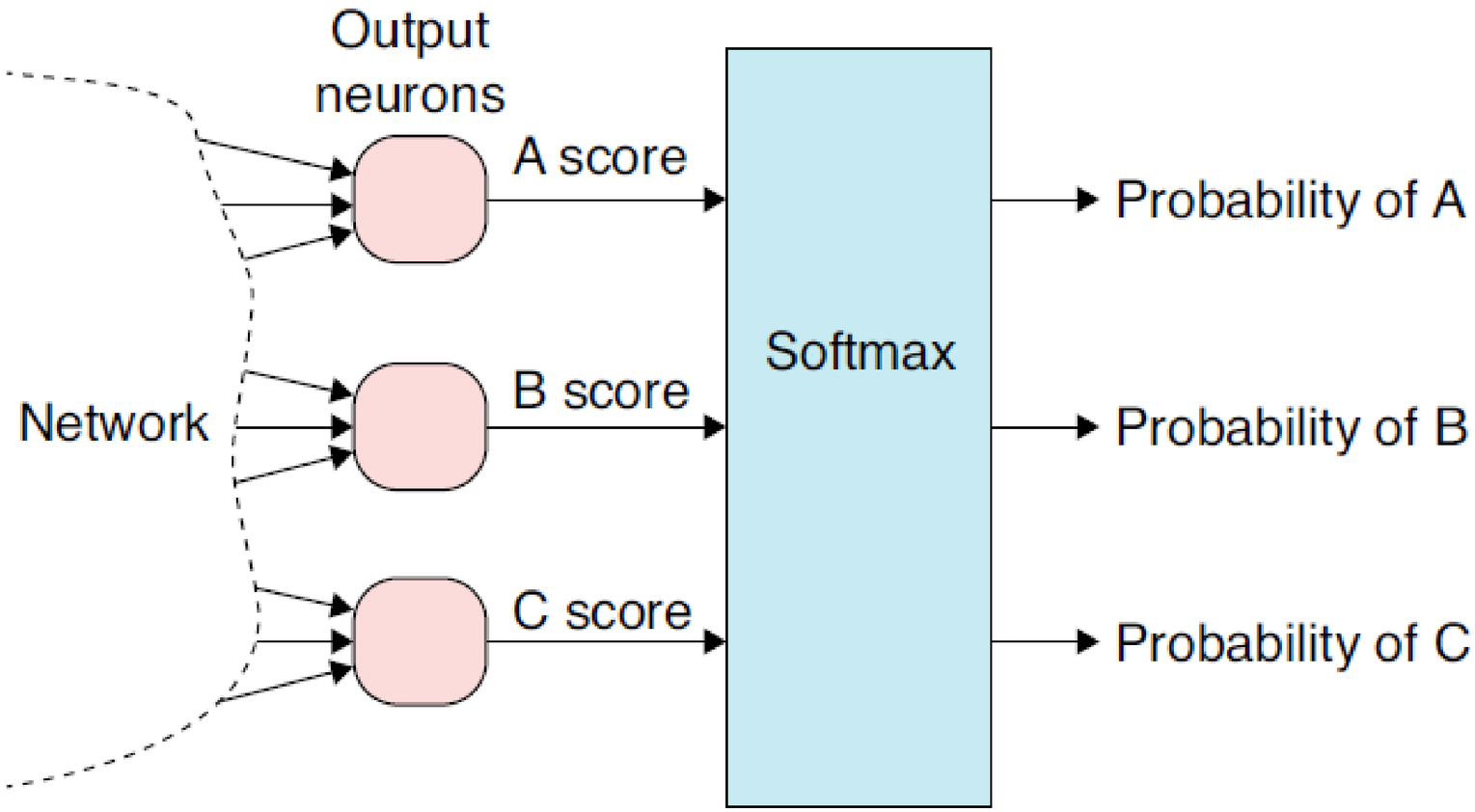


Sinusoid



Smooth Functions: Softmax

- Softmax is a technique used in classification outputs of neural networks
- Softmax transforms the raw numbers that come out of a classification network into class probabilities.



Softmax: Illustration

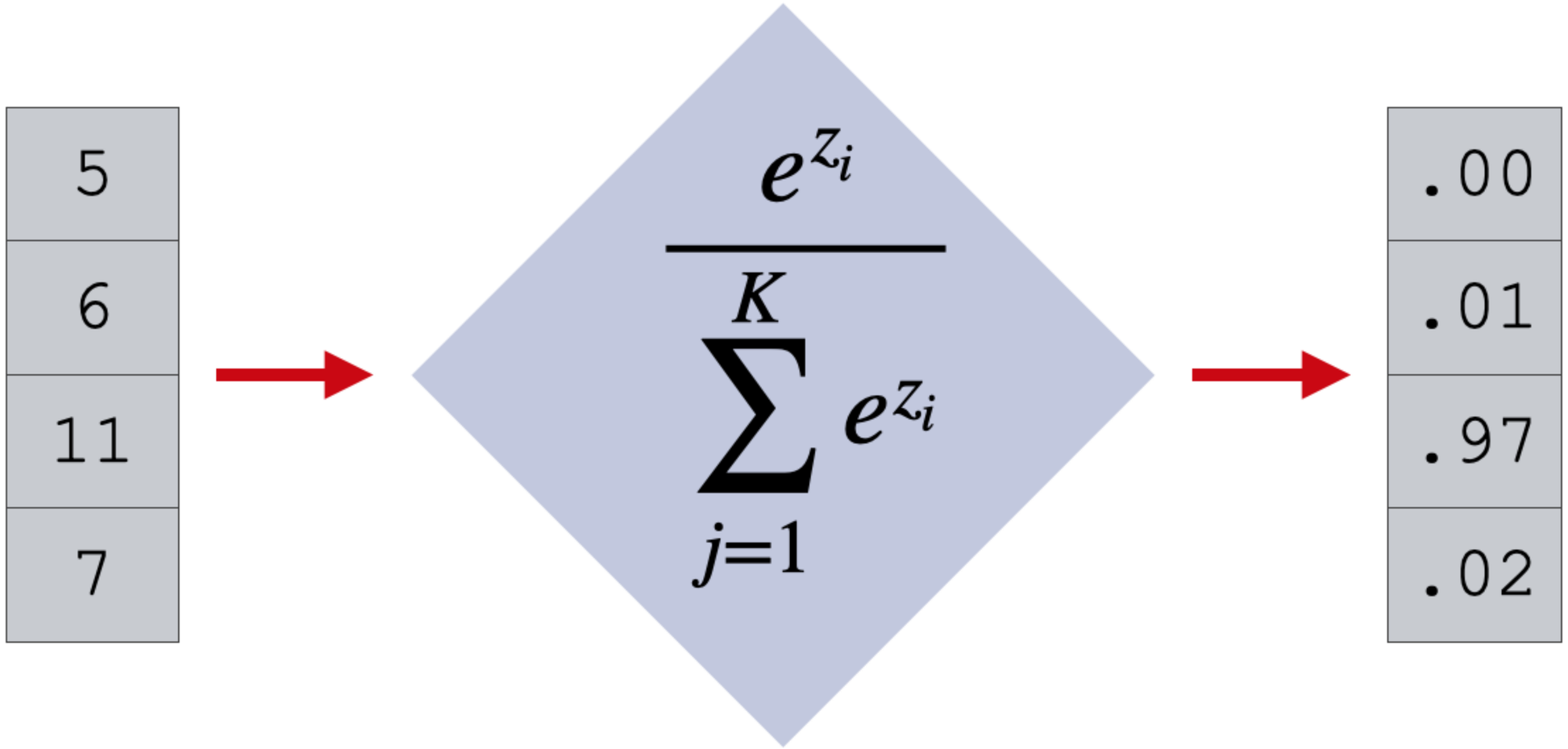


Illustration: Refer to **Softmax Illustration.ipynb**

Initializing Weights

- After data pre-processing and selection of neural network model, the first step in training of NN is initializing weights.
- In practice, initialization of weights impacts the speed of training NNs
- Algorithms to initialize weights of neural networks
  - Uniform distribution
    - The LeCun Uniform,
    - Glorot Uniform (or Xavier Uniform)
    - He Uniform
  - Normal Distribution
    - LeCun Normal,
    - Glorot Normal (or Xavier Normal),
    - He Normal

Challenges to Train DNN

- **Vanishing Gradients Problem**
  - Gradients may grow smaller or larger when glowing backwards through DNN during training.
  - Makes it hard to train a model
- Not enough data to train a large DNN
- Large data is available. However, data may be costly to label
- Training may be extremely slow
- Model with millions of parameters may overfit

Vanishing Gradient Problem

- The backpropagation algorithm goes from the output layer to the input layer, propagating the error gradient along the way.