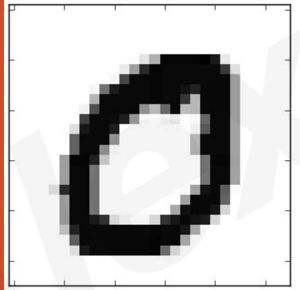
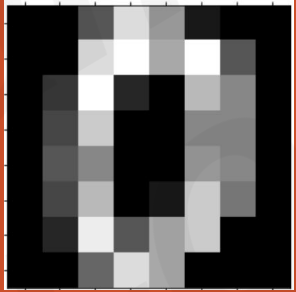
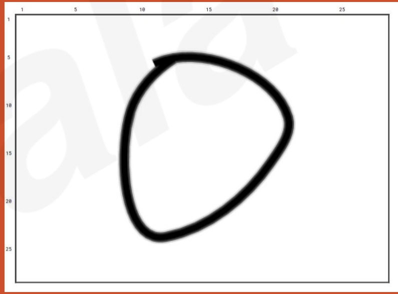
Introduction to Deep Neural Networks



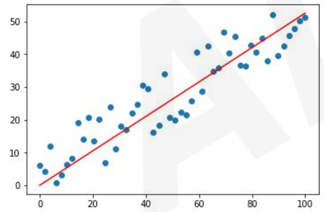


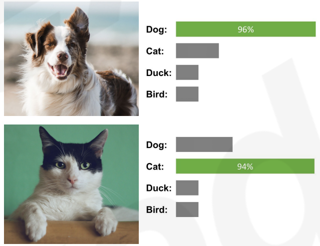


Supervised ML Background

Supervised learning

Regression | Classification





Start with a set of "observations"

$$x$$

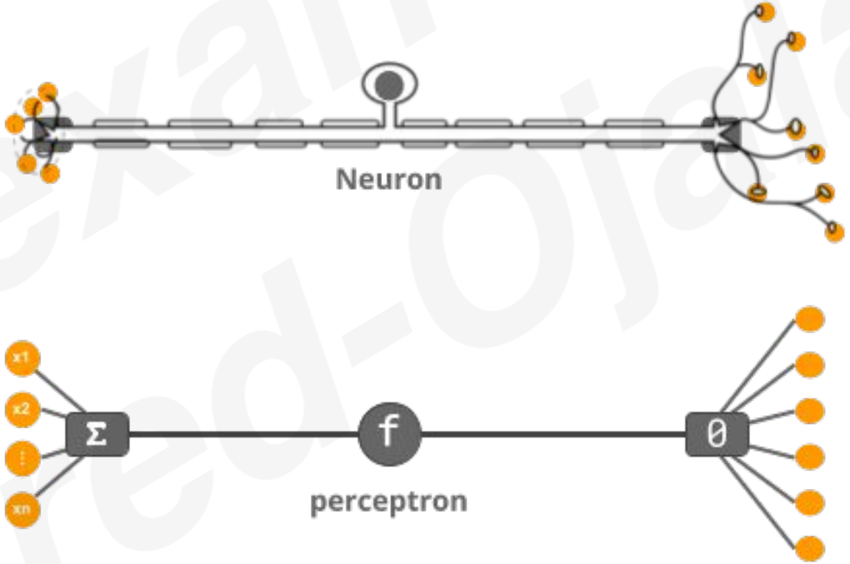
and a space of "targets" (or "labels")

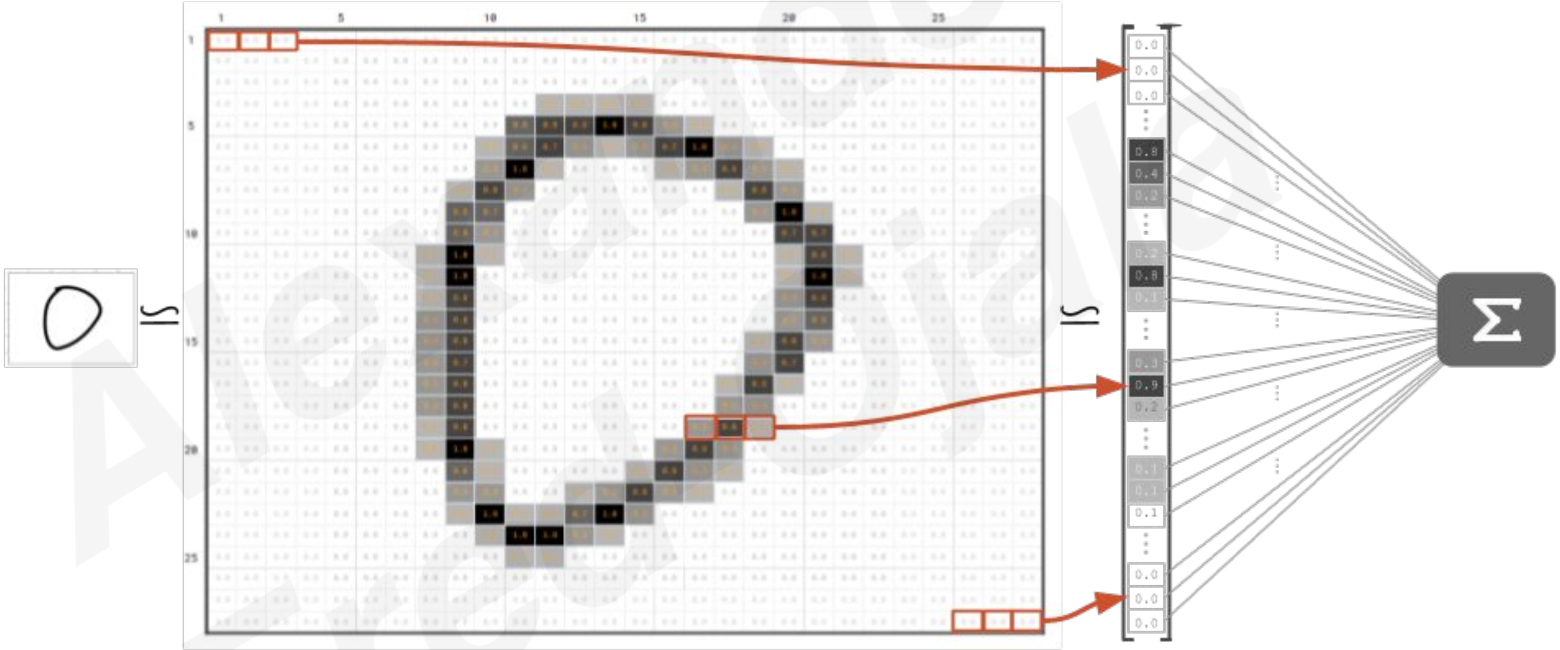
$$y$$

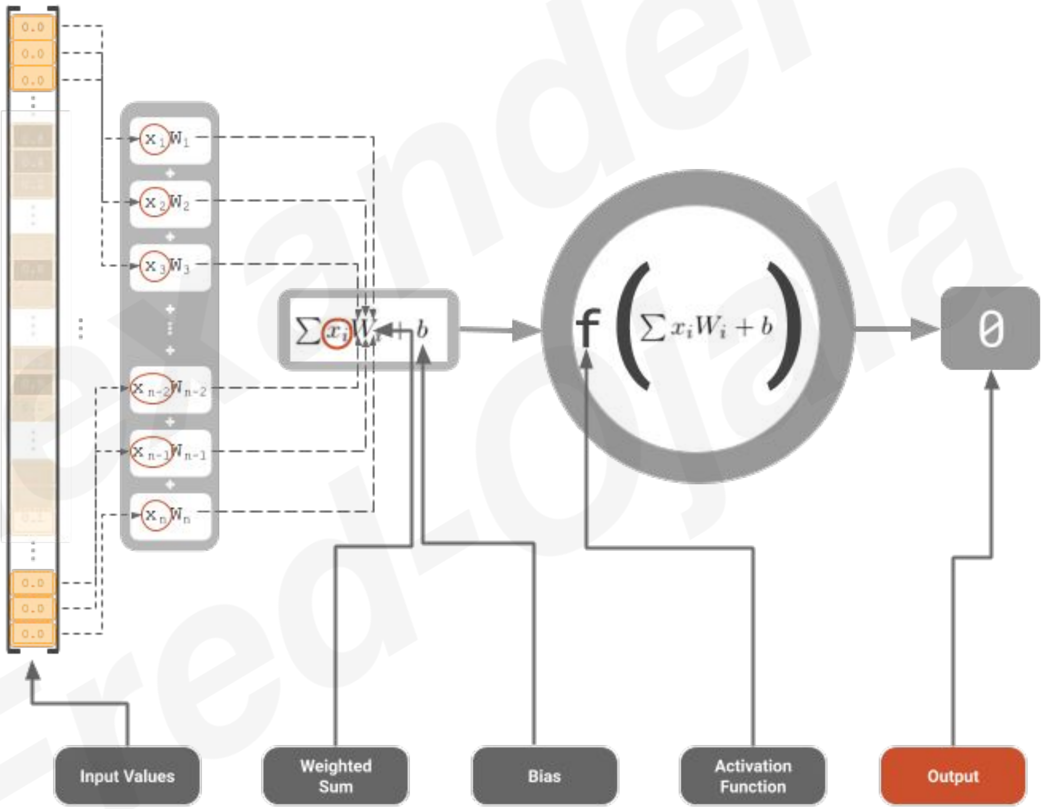
We are interested in finding a model that can map observations to determine its associated target (e.g. prediction, classification etc).

Neural:

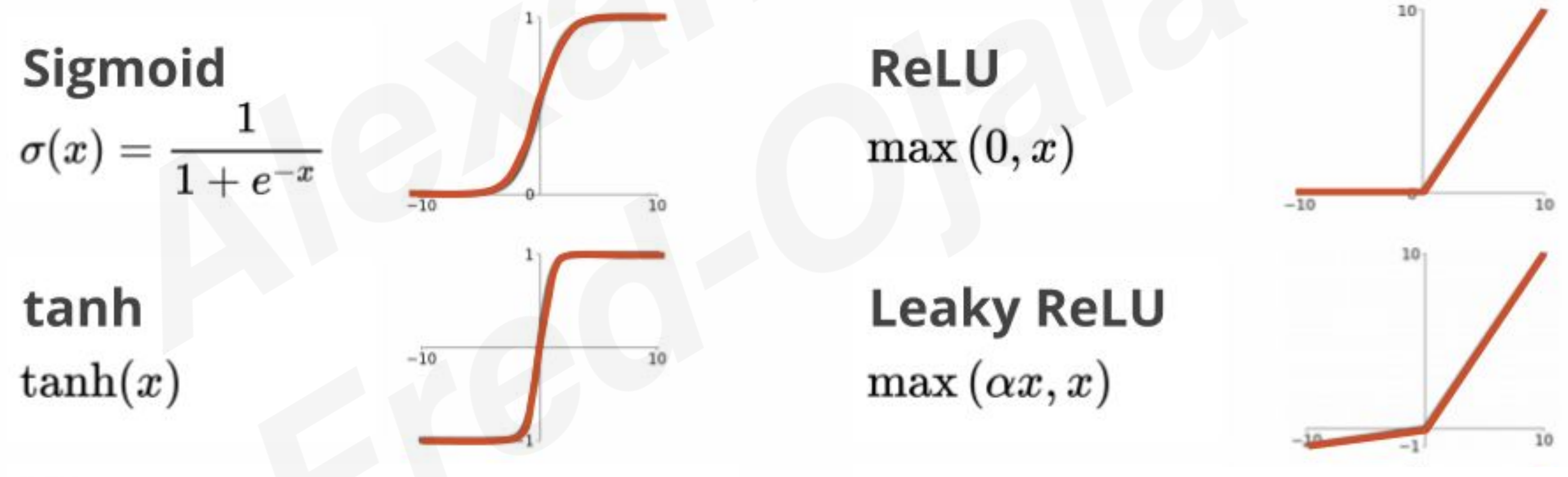
Inspired by the way biological neurons work
in the human brain




1
5
10
15
20
25
0.0
0.8
0.4
0.2
0.8
0.1
0.9
0.1
Σ

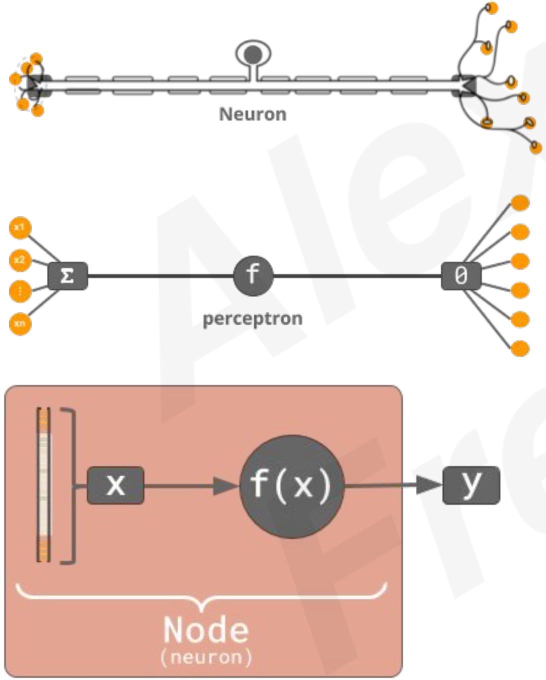

0.0
0.0
0.0
$x_1 W_1$
$x_2 W_2$
$x_3 W_3$
$x_{n-2} W_{n-2}$
$x_{n-1} W_{n-1}$
$x_n W_n$
0.0
0.0
0.0
$\sum x_i W_i + b$
$f\left(\sum x_i W_i + b\right)$
0
Input Values
Weighted Sum
Bias
Activation Function
Output

Activation functions



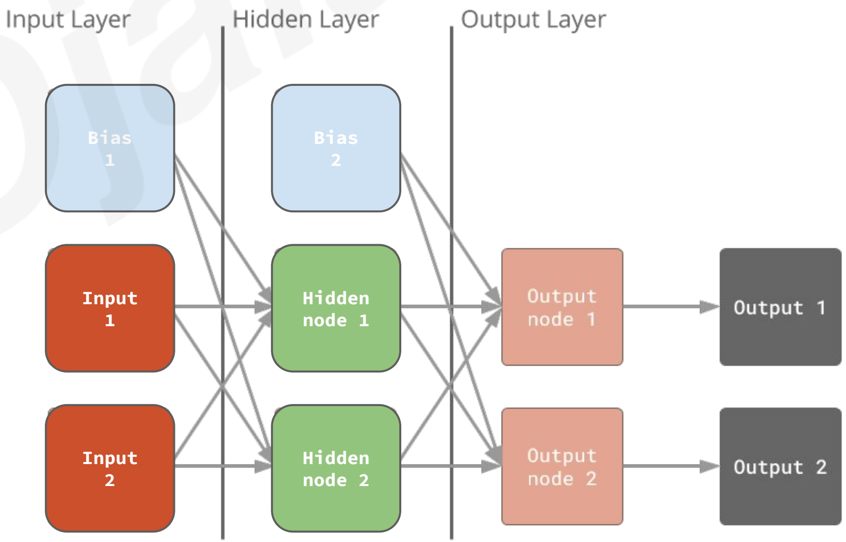
Neural:

Inspired by the way biological neurons work in the human brain process



Network:

A network of neurons/nodes connected by a set of weights



Schematic of multi-layer perceptron

Neural Networks

How does the network know what good weight values are? → it learns them.

Steps:

1. Initialize the network with random weights
2. Input a set of features
3. Calculate the output of the network by feeding the example through all layers (forward pass)
4. Calculate the value of the loss function
5. Backpropagate the error across every layer, calculate the loss gradient and update the weights
6. Repeat from 2 until desired (or acceptable) performance is achieved

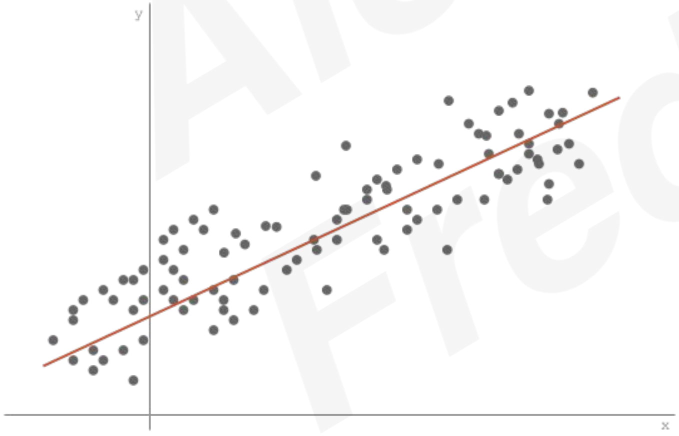
Multi-layer Perceptron (covered)

(covered next)

Neural Network (Deep if hidden layers > 2)

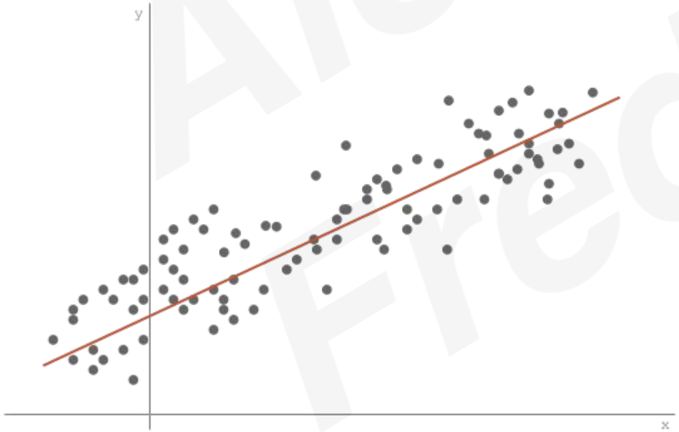
Prediction Functions

$$\hat{y} = f(x) = ax + b$$



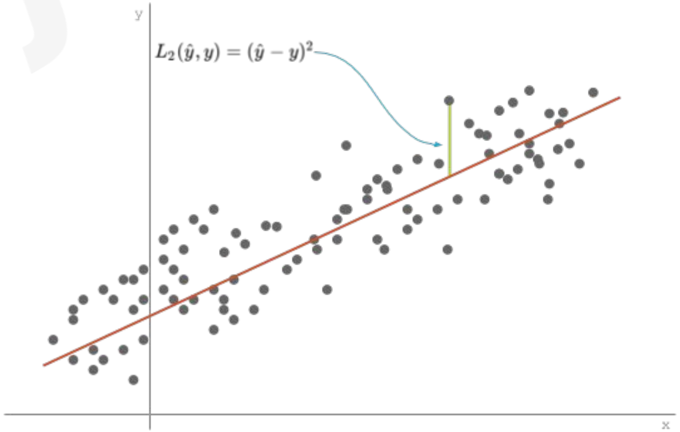
Prediction Functions

$$\hat{y} = f(x) = ax + b$$

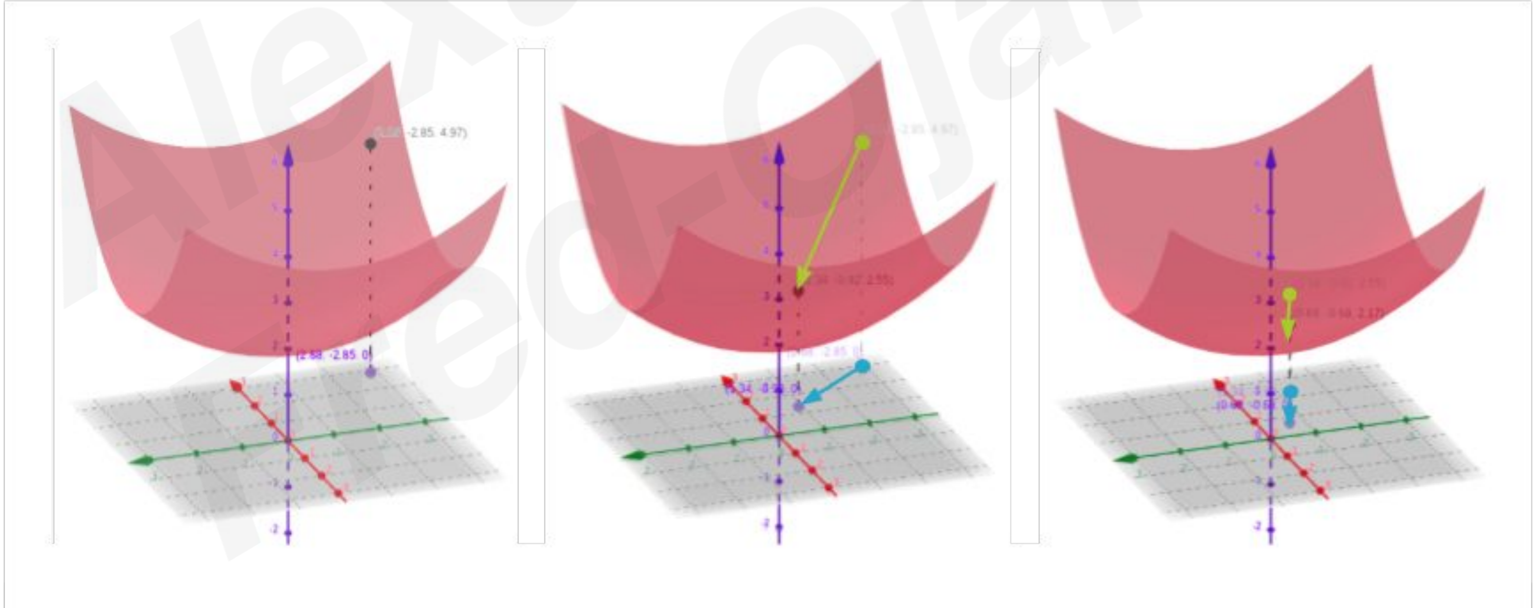


Loss Functions

$$L_2(\hat{y}, y) = (\hat{y} - y)^2$$



Optimization: Gradient Descent



Backpropagation

Chain Rule

For input *x*, target *y*, and parameters *W*, our loss is of the form

$$L\left(f\left(x, W\right), y\right)$$

To compute the gradient of *L* with respect to *W* we need the chain rule

$$\frac{dL}{dW} = \frac{dL}{df}\frac{df}{dW}$$

given *f* is single-valued, *W* a single parameter.

if *W* is a vector of parameters, then

$$\nabla_W L = \frac{dL}{df} \nabla_W f$$

if *f* is vector-valued with *k* values and *W* is a vector of *m* parameters

$$\frac{\partial L}{\partial W_j} = \sum_{i=1}^{k} \frac{\partial L}{\partial f_i}\frac{\partial f_i}{\partial W_j}$$

Backpropagation

Jacobians

$$\frac{\partial L}{\partial W_j} = \sum_{i=1}^{k} \frac{\partial L}{\partial f_i} \frac{\partial f_i}{\partial W_j}$$

For *j = 1,…, m* the above can be written as

$$J_L(W) = J_L(f) J_f(W)$$

where the Jacobian matrix is given by

$$J_f(W)_{ij} = \frac{\partial f_i}{\partial W_j}$$

Thus the Jacobian generalizes the gradient of a scalar-valued function *f* to a *k*-valued function -- which is used to represent a neural layer with *m* inputs and *k* outputs with.

$$J_f(W) = \begin{bmatrix} \frac{\partial f_1}{\partial W_1} & \frac{\partial f_1}{\partial W_2} & \cdots & \frac{\partial f_1}{\partial W_m} \\ \frac{\partial f_2}{\partial W_1} & \frac{\partial f_2}{\partial W_2} & \cdots & \frac{\partial f_2}{\partial W_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_k}{\partial W_1} & \frac{\partial f_k}{\partial W_2} & \cdots & \frac{\partial f_k}{\partial W_m} \end{bmatrix}$$

Backpropagation

N-Step Chain Rule

Now suppose we have a deep network composed of several vector-valued functions

$$A, B, C, \ldots$$

composed in a chain (Input 1, Hidden 1, etc.)

$$W \rightarrow A \rightarrow B \rightarrow C \rightarrow \cdots \rightarrow L$$

Which is represented algebraically as

$$L(W) = L\left(\cdots C\left(B\left(A\left(W\right)\right)\right)\cdots\right)$$

Then we can get the gradient by simple matrix multiplication of the Jacobians

$$J_L(W) = J_L(K) * \cdots * J_C(B) * J_B(A) * J_A(W)$$

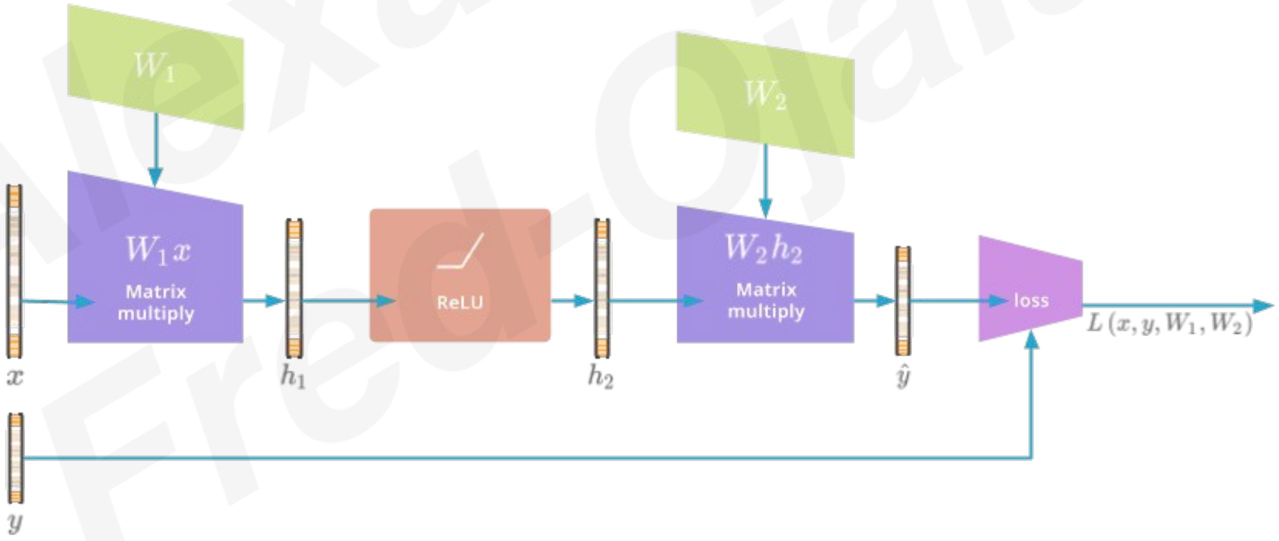
where

$$J_L(W) = (\nabla_W L)^T$$

is the gradient we need to minimize the loss over *W*.

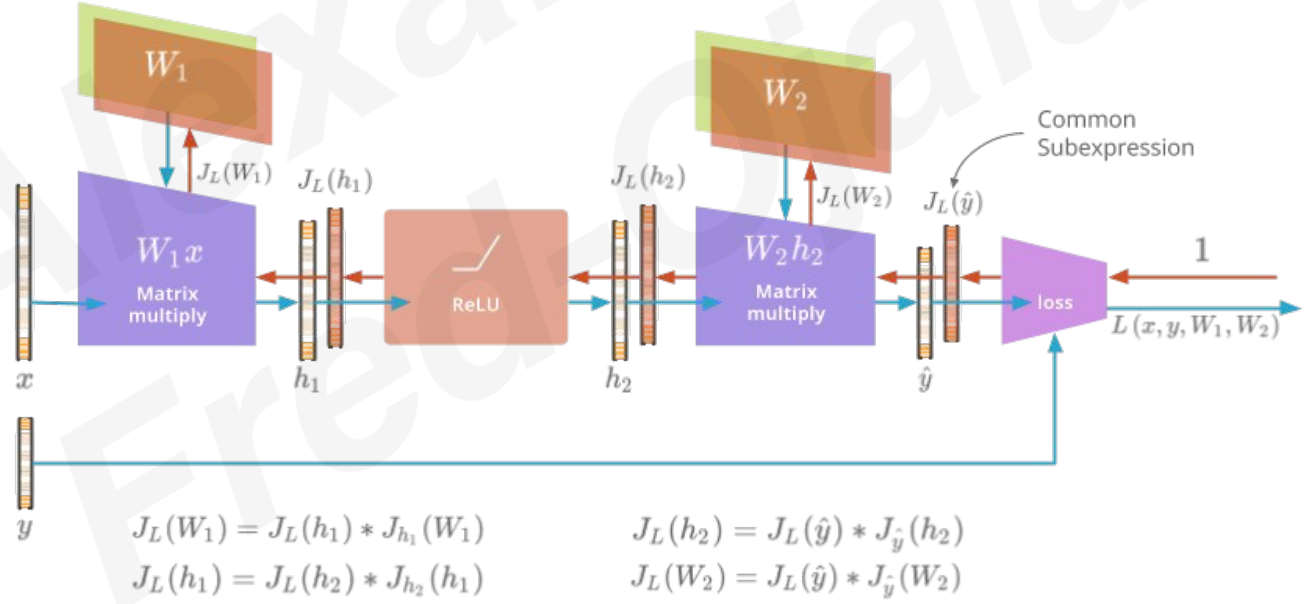
1-Layer Neural Network - forward pass

Zooming out: building a neural network (Tensorflow style)



1-Layer Neural Network - backpropagation

Zooming out: building a neural network (Tensorflow style)



Thank You!