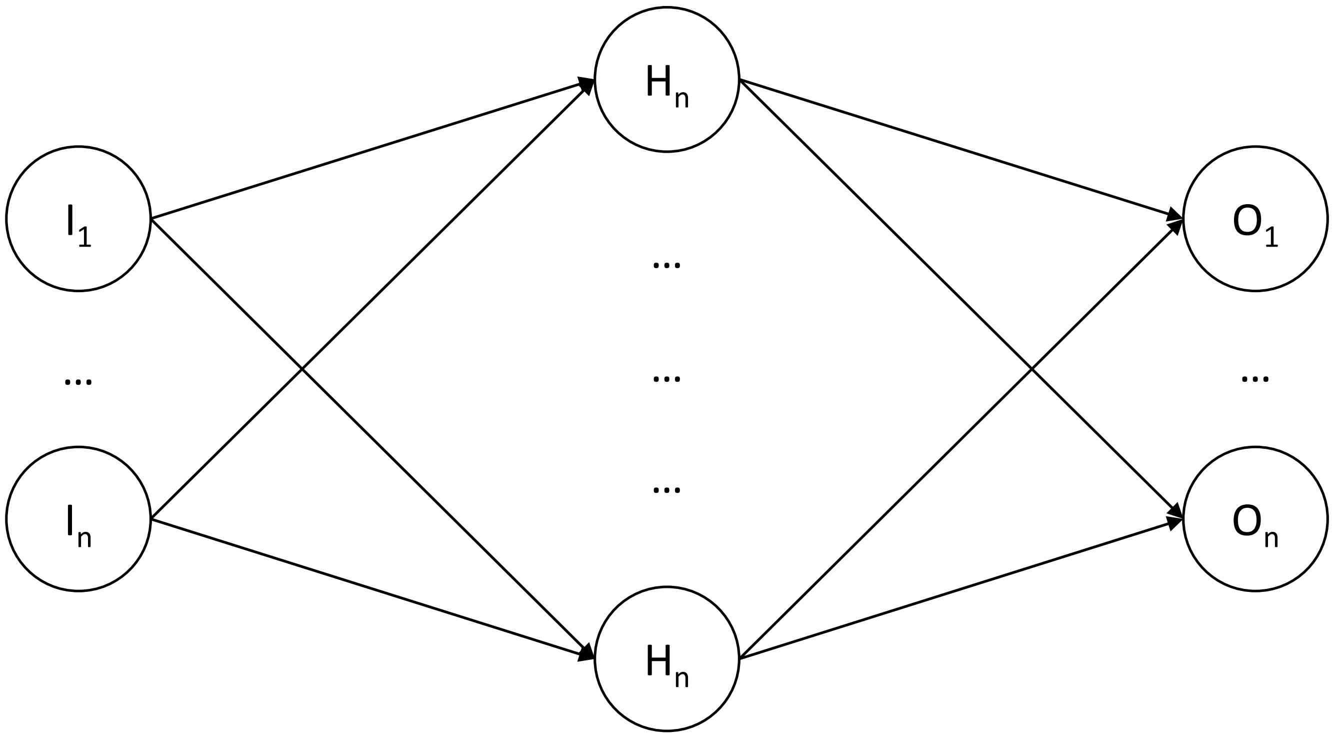
ANN
Activation function - comparison

Dominik Lewy

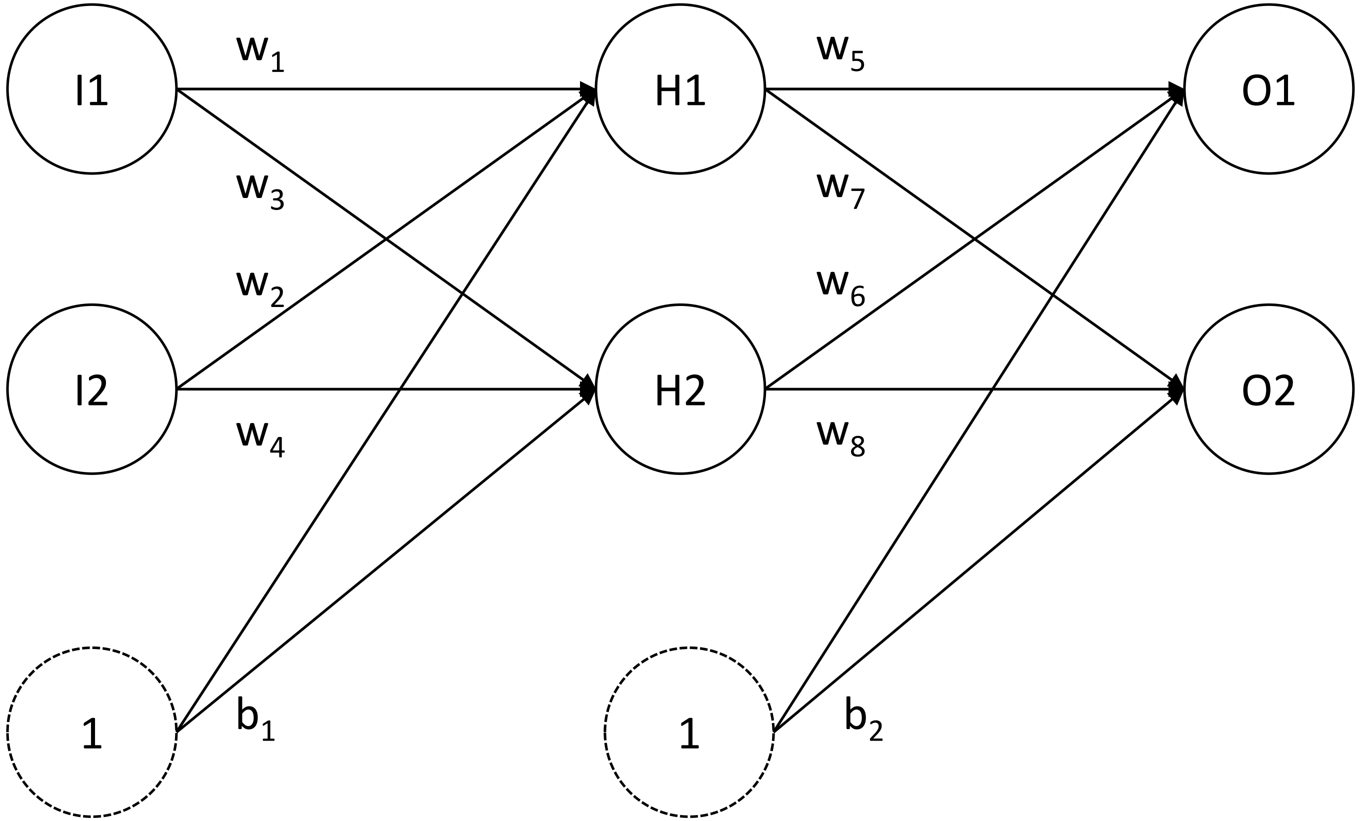
Neural Network – Hyperparameters



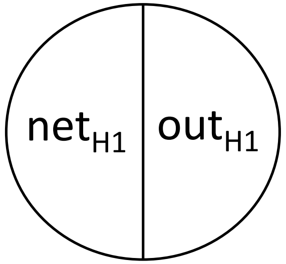
What can be tested:

- Architecture (depth, number of neurons, activation function)
- Learning rate
- Weight initialization
- Regularization

Activation functions – Forward Pass



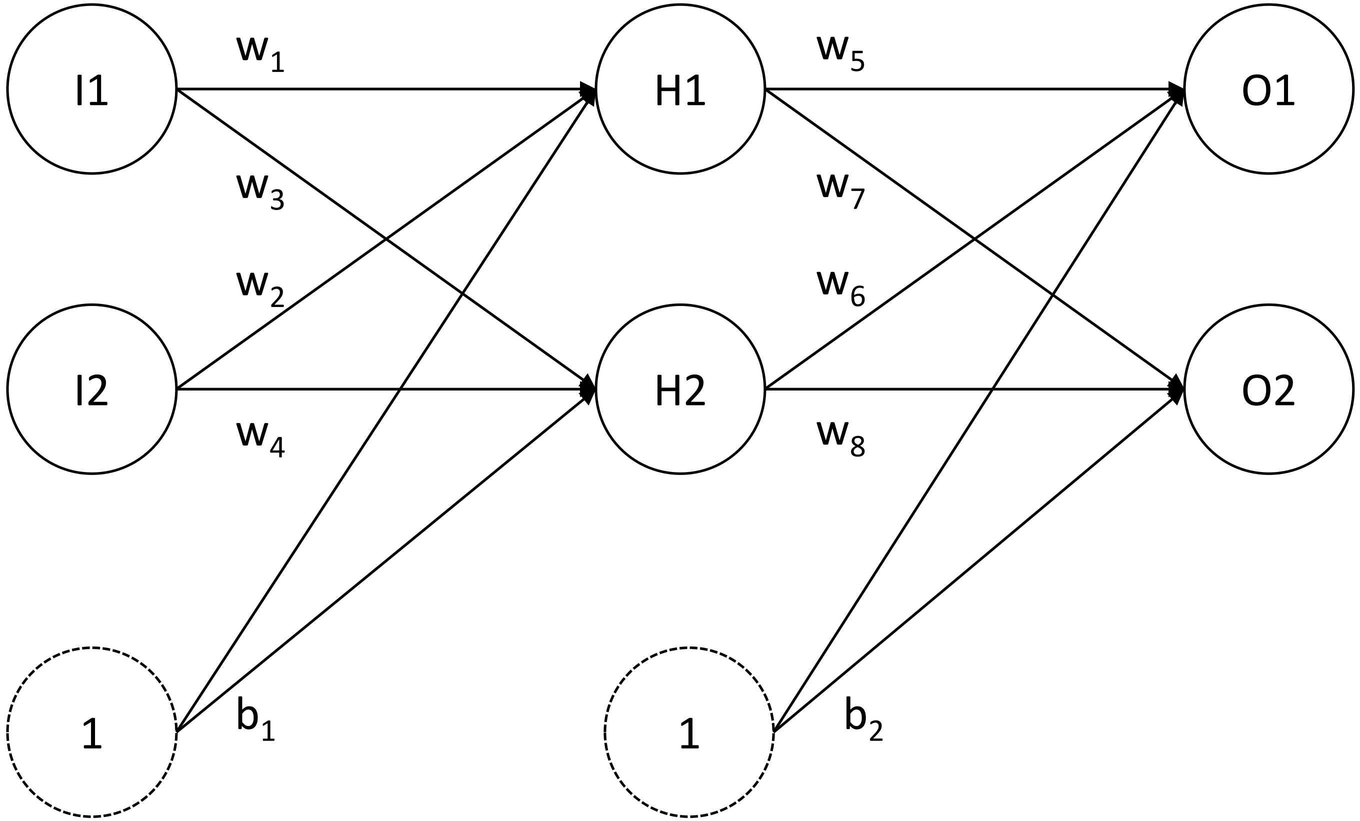
Example: H1



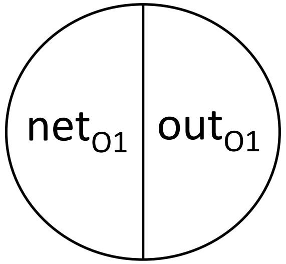
$$net_{H1} = w_1 * I_1 + w_2 * I_2 + b_1 * 1$$

$$out_{H1} = activation(net_{H1})$$

Activation functions – Forward Pass



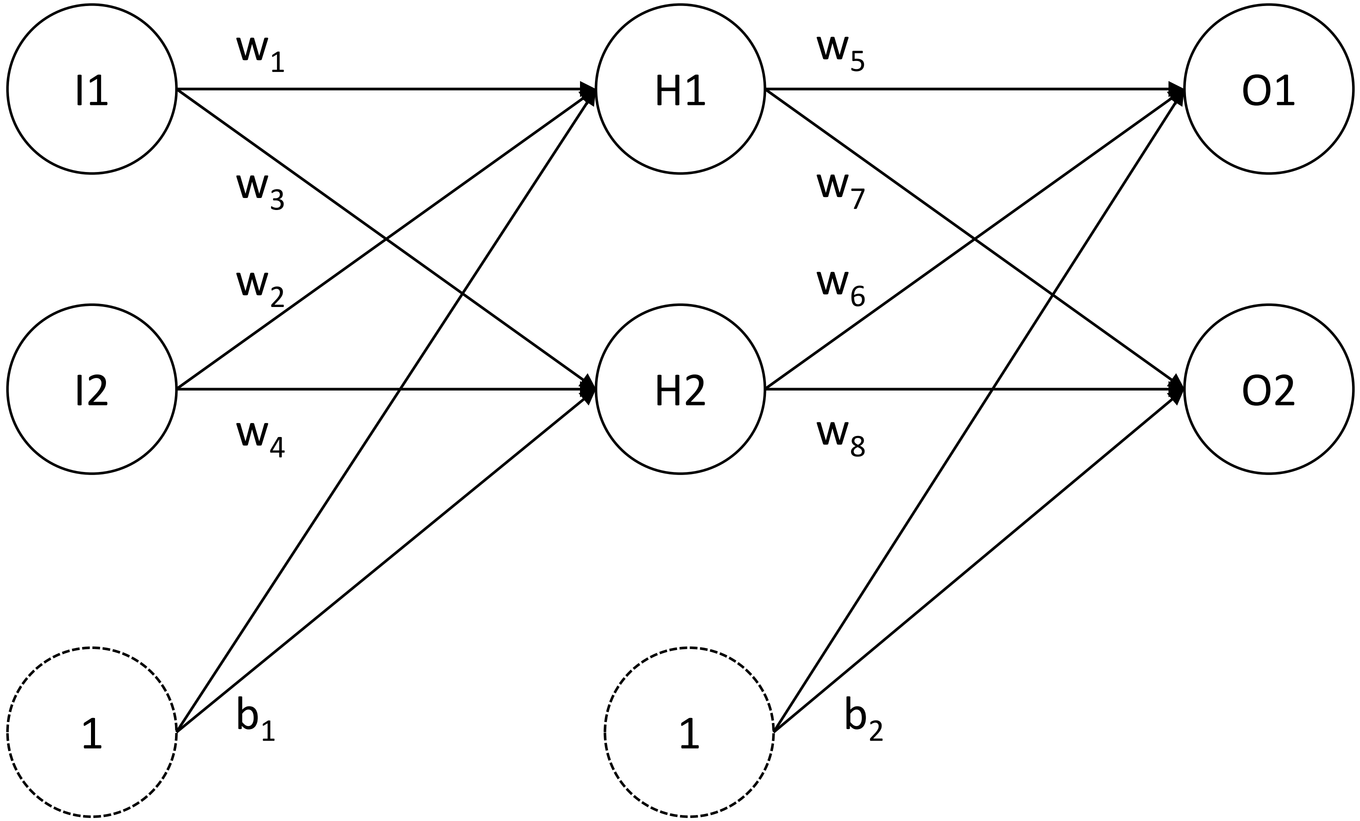
Example: O1



$$net_{O1} = w_5 * out_{H1} + w_6 * out_{H2} + b_1 * 1$$

$$out_{O1} = activation(net_{O1})$$

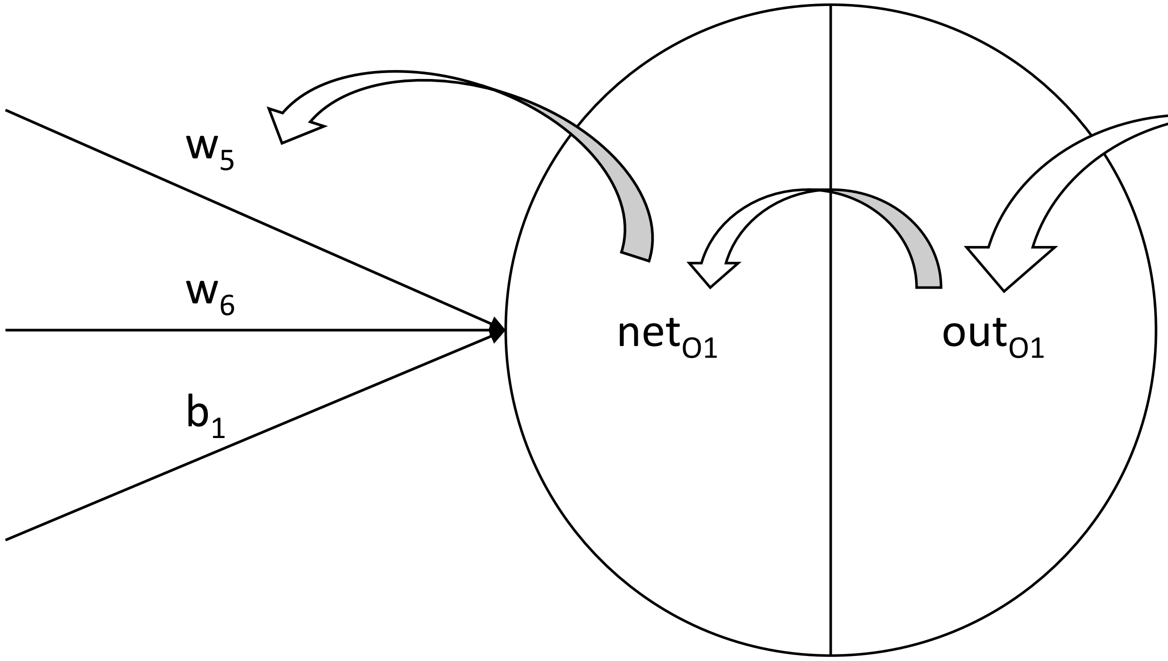
Activation functions – Error



$$E_{O1} = \mathrm{f}(\mathrm{target}_{O1} - \mathrm{output}_{O1})$$

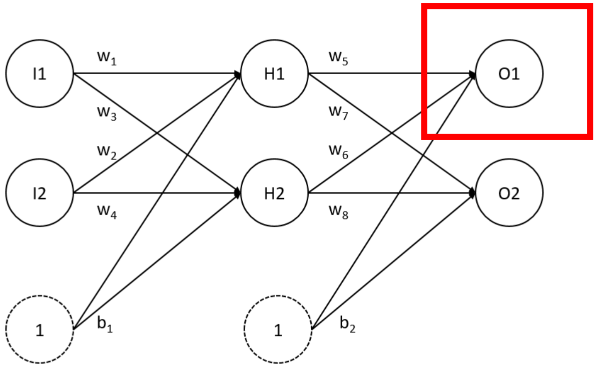
$$E_{total} = E_{O1} + E_{O2}$$

Activation functions – Backward Pass

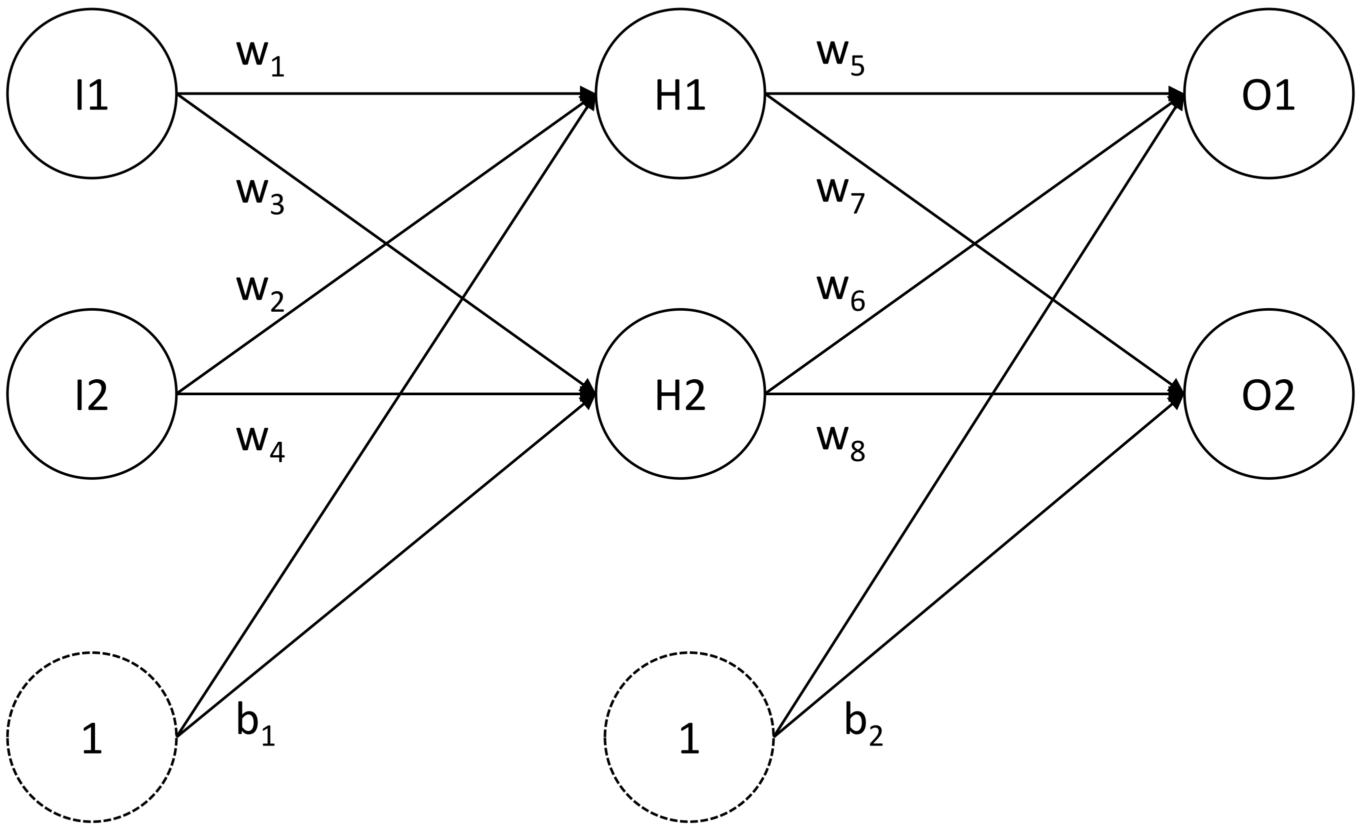


$$E_{total} = E_{O1} + E_{O2}$$

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{O1}}{\partial out_{O1}} * \frac{\partial out_{O1}}{\partial net_{O1}} * \frac{\partial net_{O1}}{\partial w_5}$$



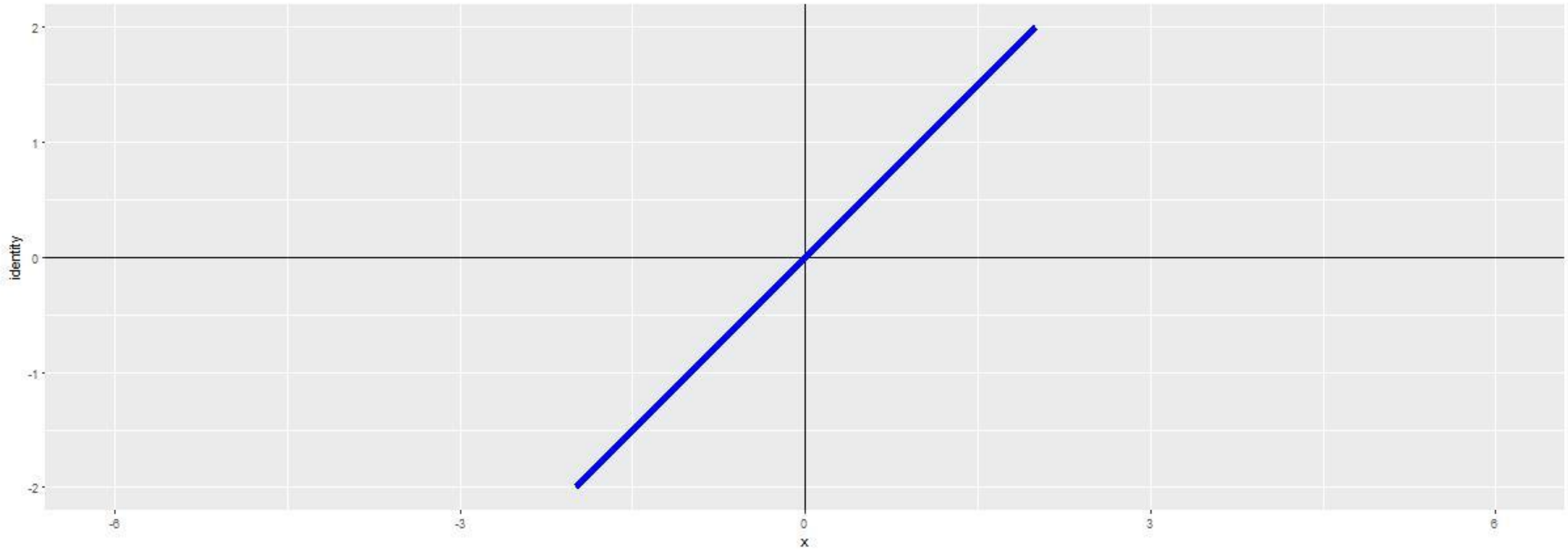
Activation functions – Backward Pass



$$\frac{\partial E_{total}}{\partial w_1} = \left(\frac{\partial E_{O1}}{\partial out_{O1}} * \boxed{\frac{\partial out_{O1}}{\partial net_{O1}}} * \frac{\partial net_{O1}}{\partial out_{h1}} + \frac{\partial E_{O2}}{\partial out_{O2}} * \boxed{\frac{\partial out_{O2}}{\partial net_{O2}}} * \frac{\partial net_{O2}}{\partial out_{h1}}\right) * \boxed{\frac{\partial out_{h1}}{\partial net_{h1}}} * \frac{\partial net_{h1}}{\partial w_1}$$

Activation functions - Identity

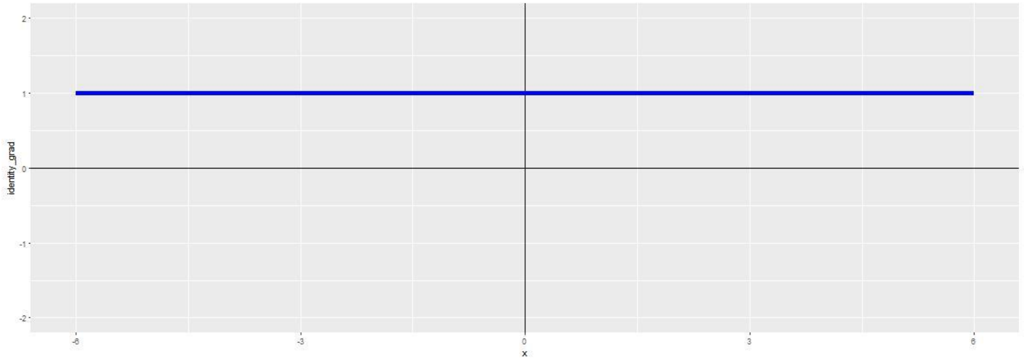
$$a_j^i = \sigma\left(z_j^i\right) = z_j^i$$



Activation functions - Identity

- Constant gradient of identity function – weight updates are independent of input values

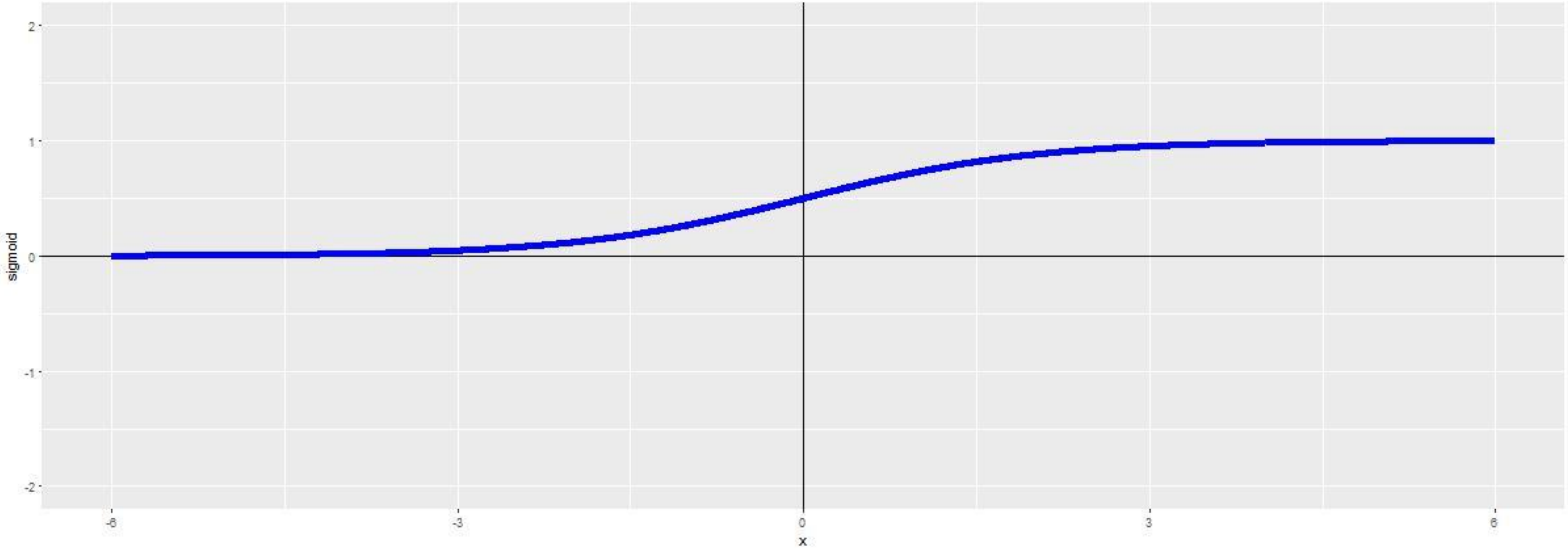
Gradient value of identity function



- No non-linearity – we lose the ability to stack layers because the output of the network remains a linear combination of the input

Activation functions - Sigmoid

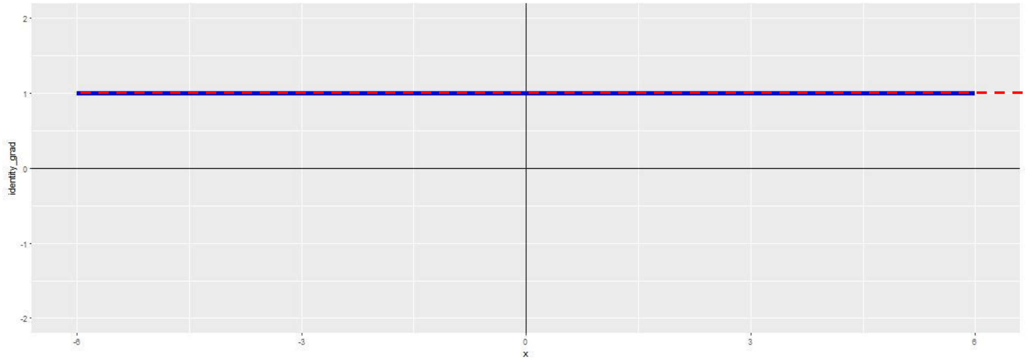
$$a_j^i = \sigma\left(z_j^i\right) = \frac{1}{1+\exp\left(-z_j^i\right)}$$



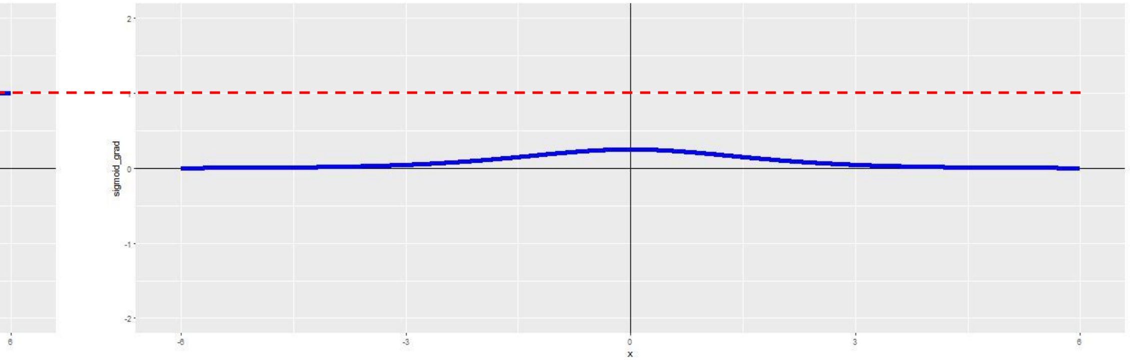
Activation functions - Sigmoid

- Small gradient of sigmoid function – makes it difficult to train the network to acceptable level

Gradient value of identity function

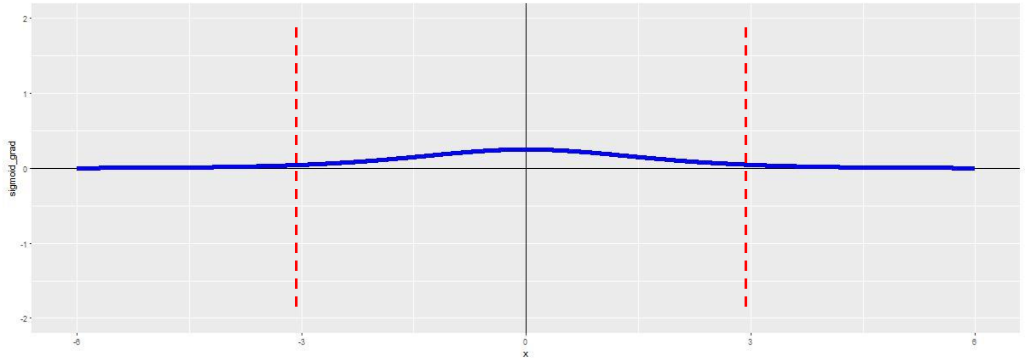


Gradient value of sigmoid function



- Incorrect weight initialization and outliers in data can lead to neuron saturation, where most neurons of the network then become saturated and almost no learning will take place

Gradient value of sigmoid function



For net_{O1} lower than -10 and greater than 10:

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{O1}}{\partial out_{O1}} * \underbrace{\frac{\partial out_{O1}}{\partial net_{O1}}}_{\sim 0} * \frac{\partial net_{O1}}{\partial w_5}$$

Vanishing gradient problem

Max value of $\frac{\partial Sigmoid(z)}{\partial z} = \frac{1}{4}$

One hidden layer:

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial output} * \frac{\partial output}{\partial hidden1} * \frac{\partial hidden1}{\partial w_1}$$

Two hidden layers:

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial output} * \frac{\partial output}{\partial hidden2} * \frac{\partial hidden2}{\partial hidden1} * \frac{\partial hidden1}{\partial w_1}$$

Three hidden layers:

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial output} * \frac{\partial output}{\partial hidden3} * \frac{\partial hidden3}{\partial hidden2} * \frac{\partial hidden2}{\partial hidden1} * \frac{\partial hidden1}{\partial w_1}$$

Best case scenario:

$$\frac{1}{4} * \cdots$$

$$\frac{1}{4} * \frac{1}{4} \cdots$$

$$\frac{1}{4} * \frac{1}{4} * \frac{1}{4} \cdots$$

Activation functions - Sigmoid

- The sigmoid function "squashes" values to the range between 0 and 1 – compared to (-inf, inf) for linear function. So we have our activations bound in a range and the activations won't blow up
  - In accordance with the analogy to human brain and neurons
  - Interpretation in terms of probability

- The output is not zero centered – all the weights will be either increased or decreased because the weights update depends only on upstream gradient

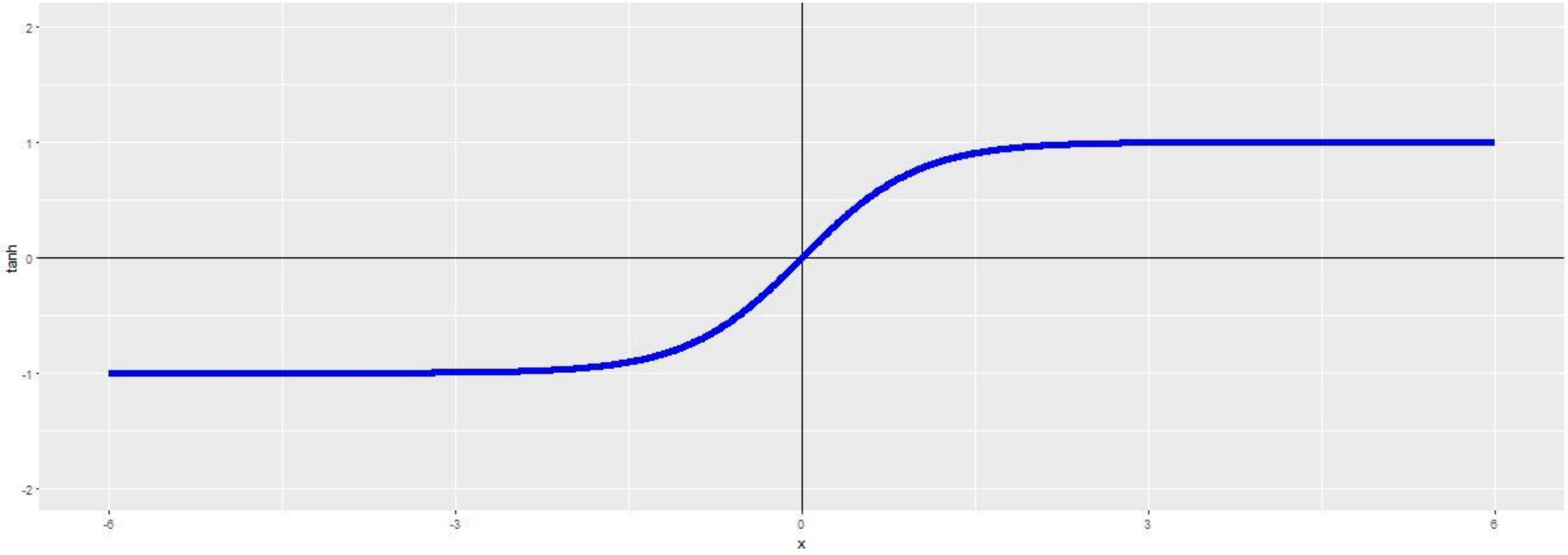
$$\frac{\partial E_{total}}{\partial w_5} = \boxed{\frac{\partial E_{O1}}{\partial out_{O1}} * \frac{\partial out_{O1}}{\partial net_{O1}}} * \underset{\oplus}{\frac{\partial net_{O1}}{\partial w_5}}$$

$$net_{O1} = w_5 * out_{H1} + w_6 * out_{H2} + b_1 * 1$$

$$\frac{\partial net_{O1}}{\partial w_5} = \underset{\oplus}{out_{h1}}$$

Activation functions – Tanh

$$a_j^i = \sigma\left(z_j^i\right) = \tanh\left(z_j^i\right)$$



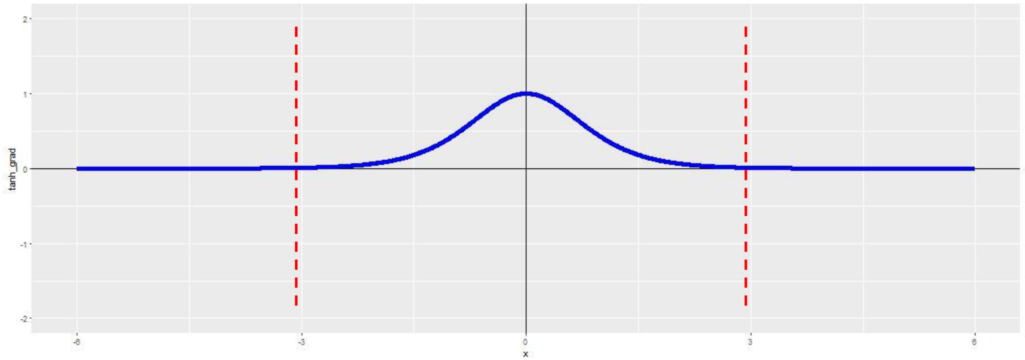
Activation functions - Tanh

- Tanh neuron is simply a scaled sigmoid neuron, in particular the following holds

$$\tanh(x) = 2sigmoid(2x) - 1$$

- Incorrect weight initialization and outliers in data can lead to neuron saturation, where most neurons of the network then become saturated and almost no learning will take place

Gradient value of tanh function



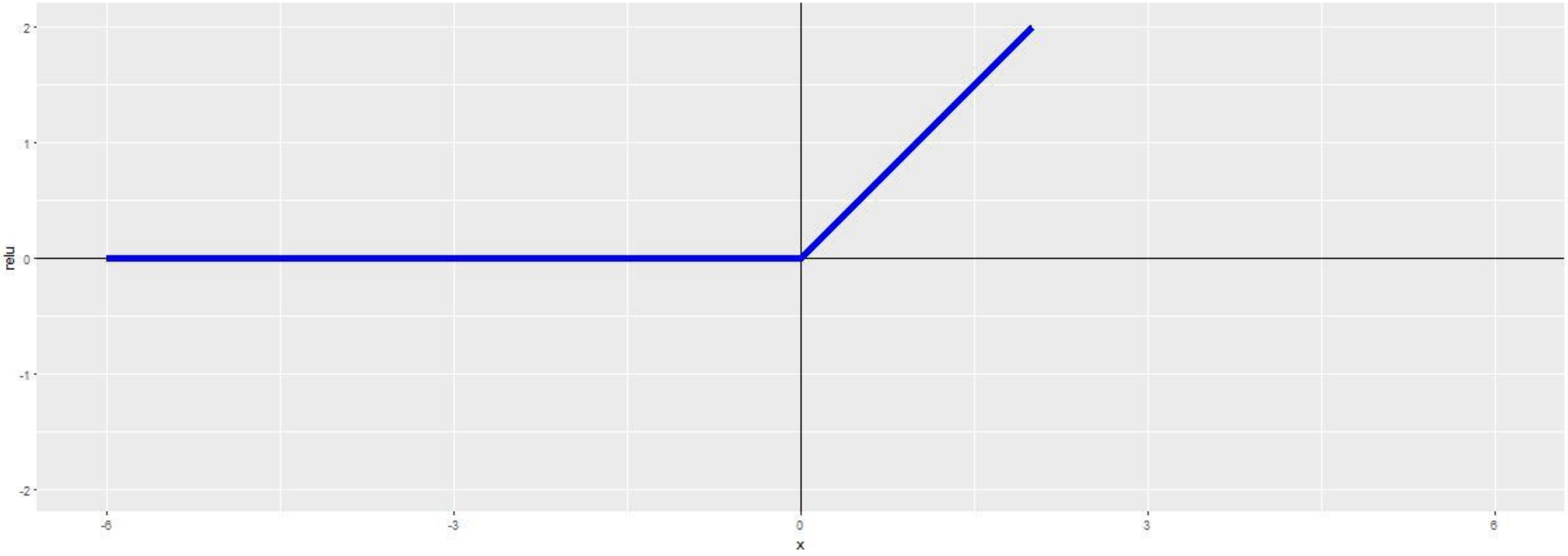
For net_{O1} lower than -10 and greater than 10:

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{O1}}{\partial out_{O1}} * \underbrace{\frac{\partial out_{O1}}{\partial net_{O1}}}_{\sim 0} * \frac{\partial net_{O1}}{\partial w_5}$$

- Problems resolved by Tanh
  - The output is not zero centered
  - Small gradient of sigmoid function

Activation functions - ReLU

$$a_j^i = \sigma\left(z_j^i\right) = \max(0, z_j^i)$$



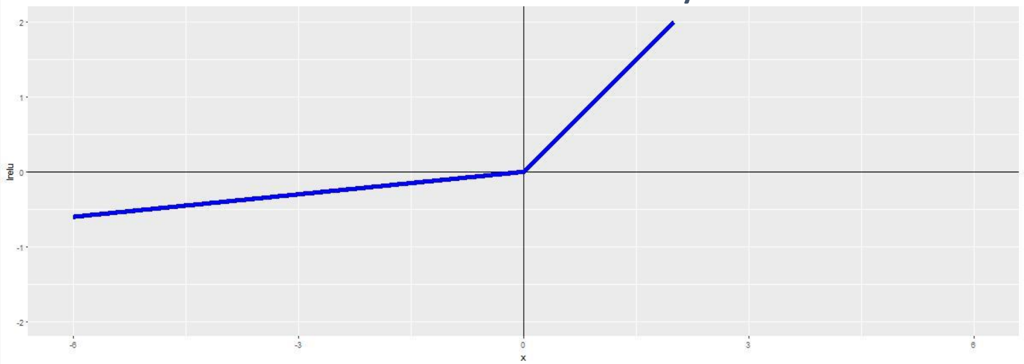
Activation functions - ReLU

- It was found to greatly accelerate the convergence of stochastic gradient descent compared to the sigmoid/tanh functions. It is argued that this is due to its linear, non-saturating form.

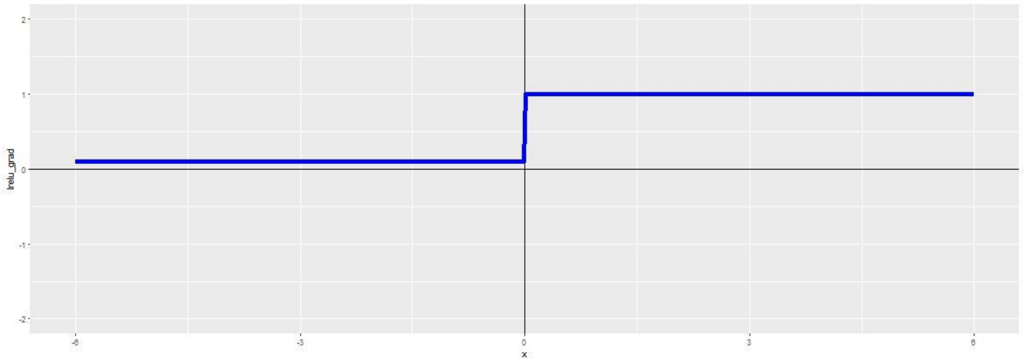
- Compared to tanh/sigmoid neurons that involve expensive operations (exponentials, etc.), the ReLU can be implemented by simply thresholding a matrix of activations at zero.

- Unfortunately, ReLU units can be fragile during training and can "die“
  - On the other hand it introduces sparsity into the net
  - There are variations of ReLU that deal with this problem like Leaky ReLU

Activation function Leaky ReLU



Gradient value of LReLU function



$$a_j^i = \sigma(z_j^i) = \max(0.1z_j^i,\ z_j^i)$$

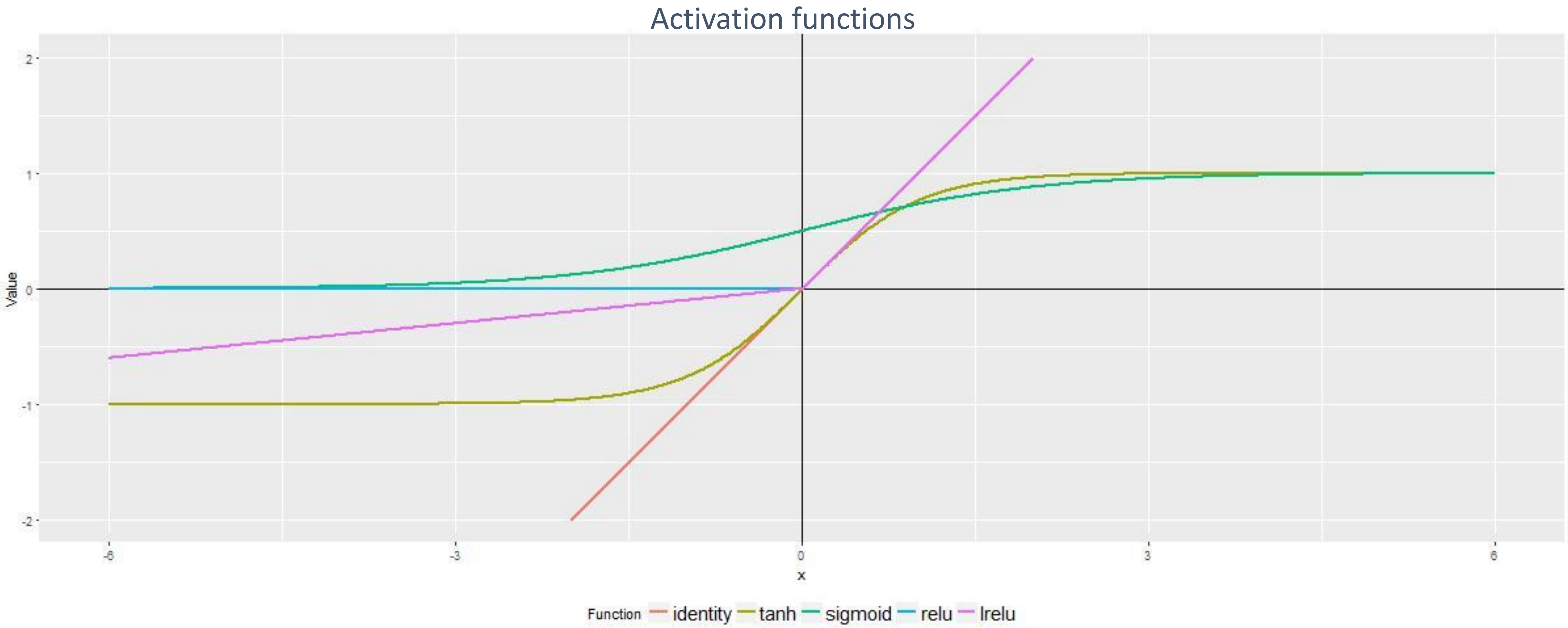
Activation functions - Summary

"*What activation function should I use?*"

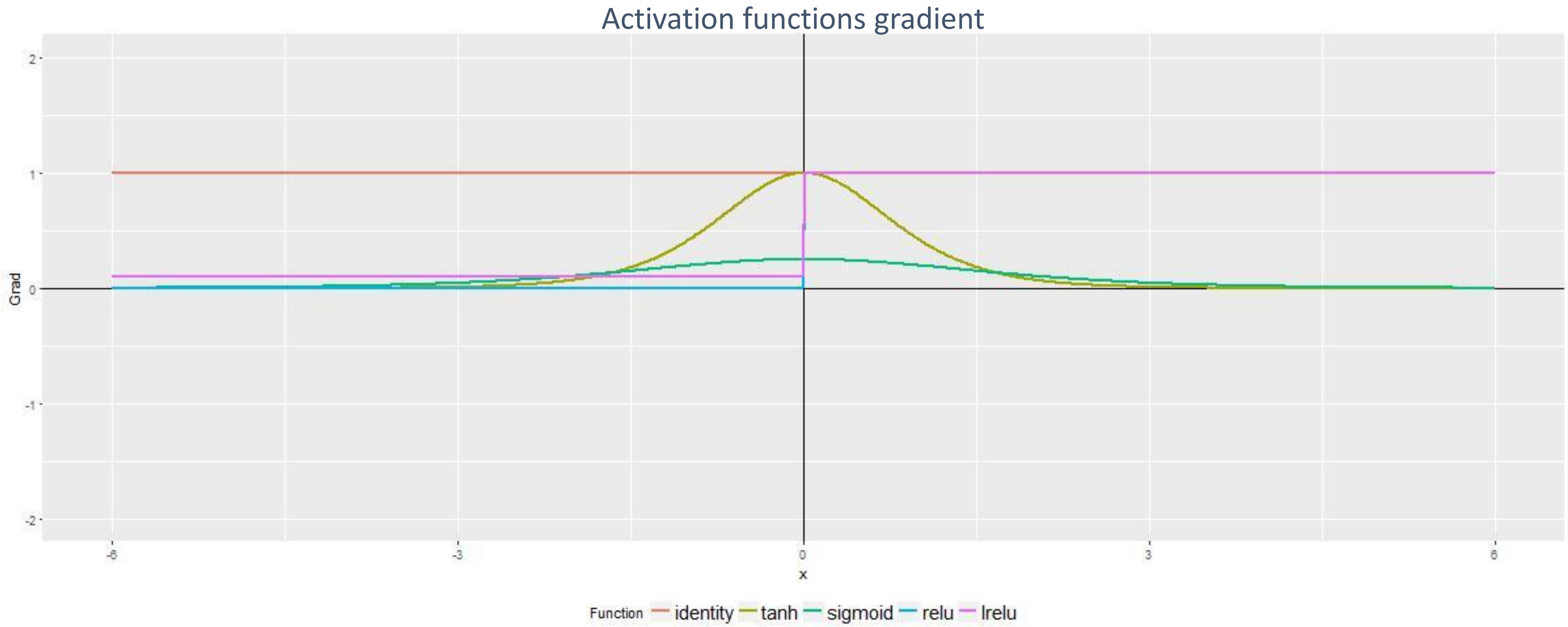
- Treat ReLU as your go to function
  - monitor the fraction of "dead" units in a network
- If the result are not good enough try Leaky ReLU
- Never use sigmoid
- You can try tanh, but expect it to work slower and worse than ReLU

Function	Advantages	Disadvantages
Identity		Blown up activations
Sigmoid	Output in range (0,1)	Saturated Neurons, Not zero centered, Small gradient, Vanishing gradient
Tanh	Zero centered, Output in range (-1,1)	Saturated Neurons
ReLU	Computational efficiency, Accelerated convergence	Dead Neurons, Not zero centered

Activation functions - Summary

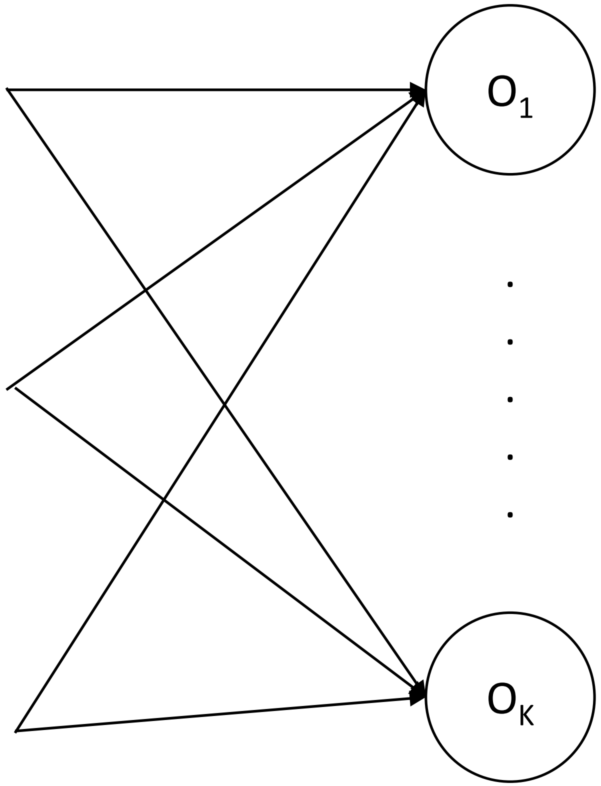


Activation functions - Summary



Activation functions - Softmax

$$a_j^i = \sigma(z_j^i) = \frac{\exp(z_j^i)}{\sum_{k=1}^{K} \exp(z_j^i)}$$



$$\frac{\exp(z_1)}{\sum_{k=1}^{K} \exp(z_k)}$$

$$\frac{\exp(z_K)}{\sum_{k=1}^{K} \exp(z_k)}$$

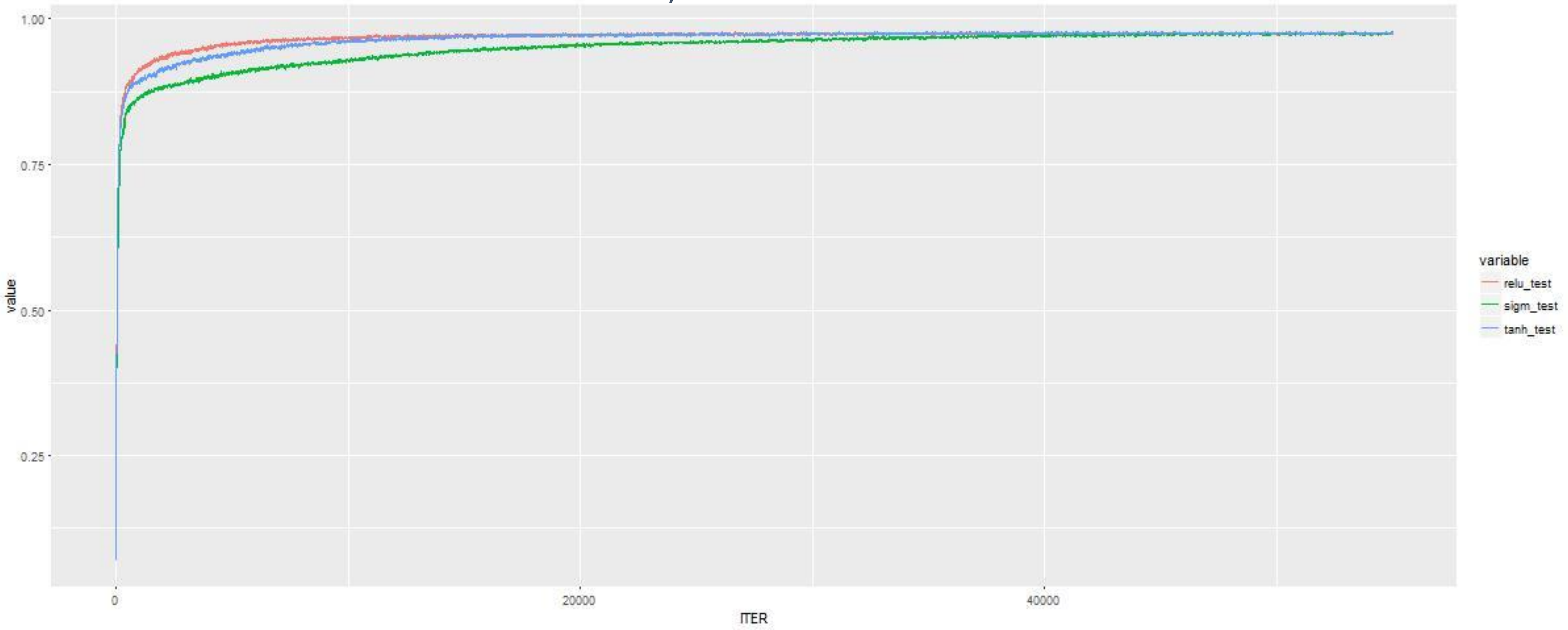
Properties:

- It is used as last layer for categorization problems
- It returns a probability of each output
- All the probabilities sum up to one

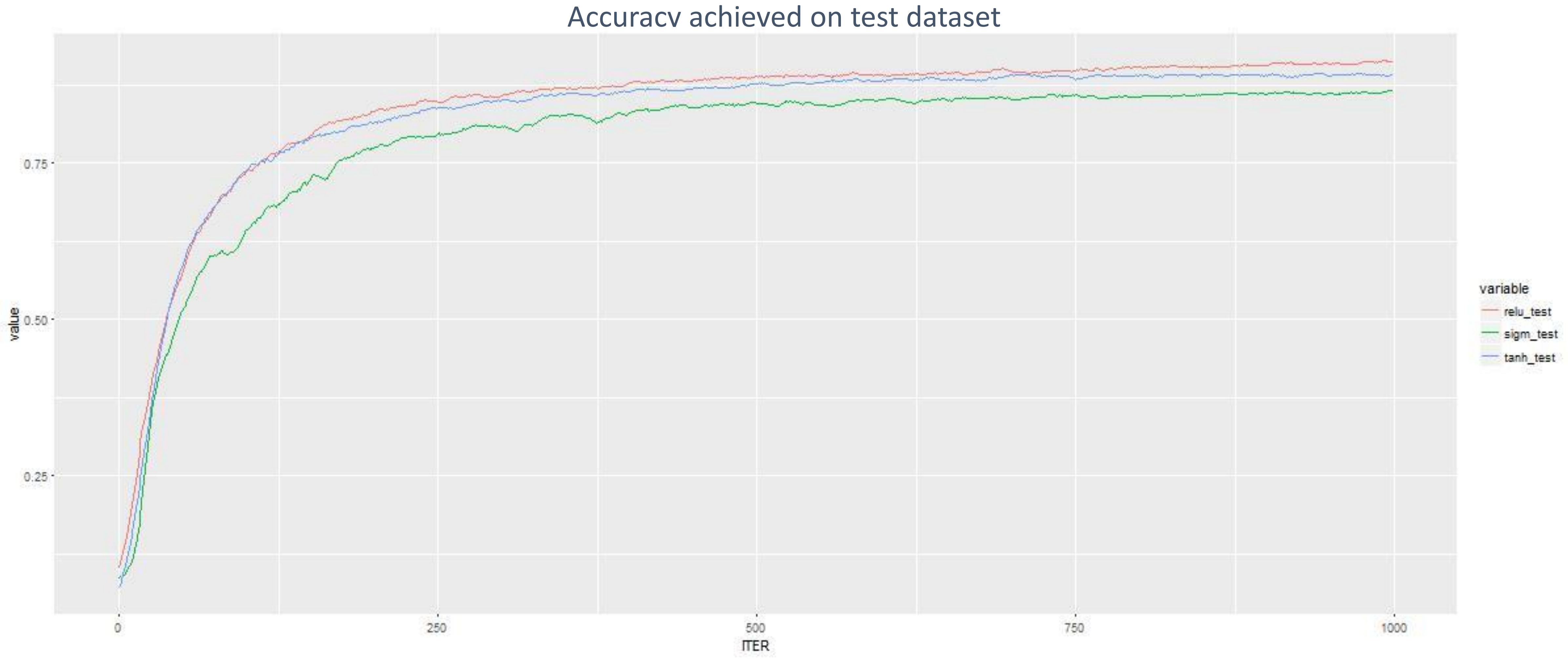
Experiment - image

It's time to look at some experiments. MNIST data set was used for this exercise. Nets with the same architecture, differing only by the activation function used in the hidden layer, were trained for 100 epochs.

Accuracy achieved on test dataset

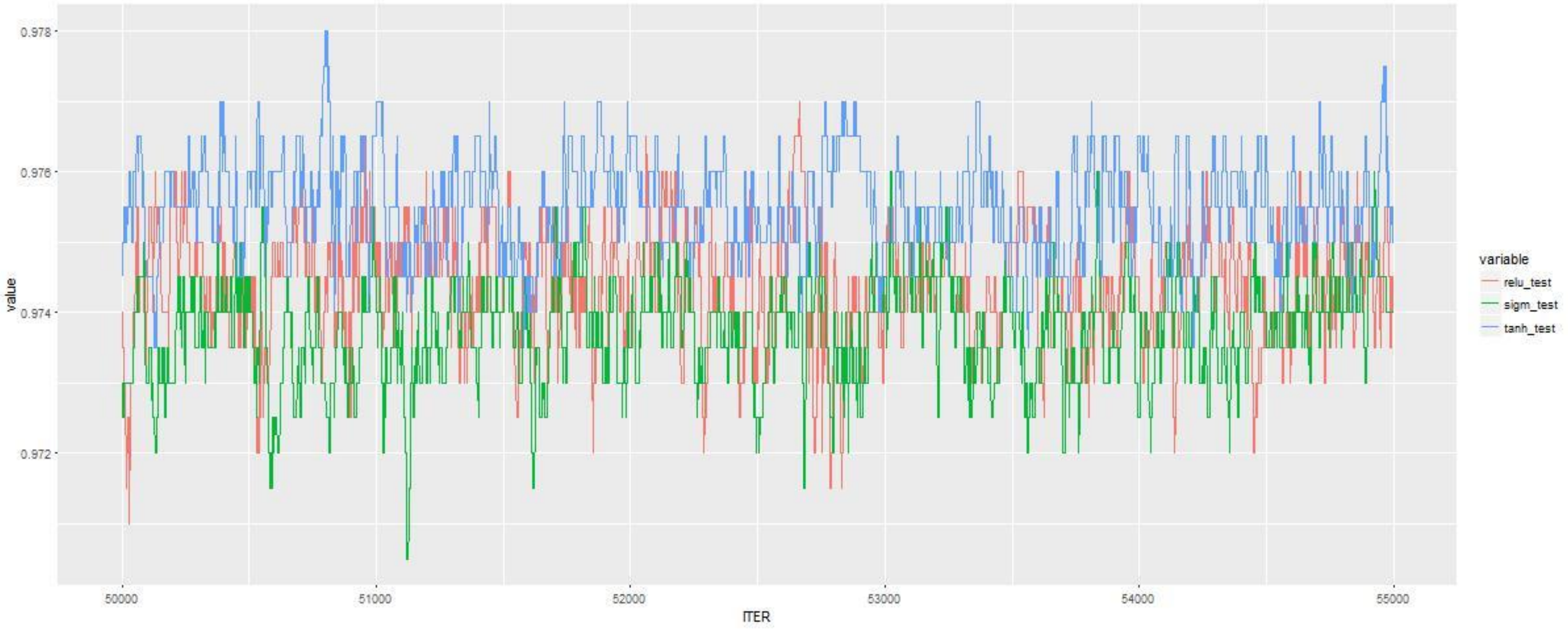


Experiment


Accuracy achieved on test dataset
value
0.25
0.50
0.75
0
250
500
750
1000
ITER
variable
relu_test
sigm_test
tanh_test

Experiment

Accuracy achieved on test dataset



Experiment

Function	Sigmoid	Tanh	ReLU
Iterations to reach 0.8 accuracy	267	174	154
		x1.5 faster	x1.7 faster
Iterations to reach 0.9 accuracy	3852 (3781 first)	1254	765 (692 first)
		x3 faster	x5 faster
Iterations to reach 0.95 accuracy	17113 (16249 first)	6417 (6053 first)	3705 (3694 first)
		x2.7 faster	x4.6 faster
Average accuracy achieved	0.974	0.975	0.975
Training time of 1 epoch	0.0224	0.0236	0.0227

Activation function – Dominik Lewy

Links

https://dashee87.github.io/data%20science/deep%20learning/visualising-activation-functions-in-neural-networks/

Discussion

Gradient Clipping

Gradient Clipping is a technique to prevent exploding gradients in very deep networks, typically Recurrent Neural Networks. There exist various ways to perform gradient clipping, but the a common one is to normalize the gradients of a parameter vector when its L2 norm exceeds a certain threshold according to `new_gradients = gradients * threshold / l2_norm(gradients)`.

$$L2 = \|(x_1, x_2)\|_2 = \left(\sum_{i=1}^{2} x_i^2\right)^{1/2}$$

http://www.cs.toronto.edu/~rgrosse/courses/csc321_2017/readings/L15%20Exploding%20and%20Vanishing%20Gradients.pdf

$$L1 = \|(x_1, x_2)\|_1 = \sum_{i=1}^{2} |x_i|$$

$$L\infty = \|(x_1, x_2)\|_\infty = \max_i |x_i|$$