Machine Learning and Deep Learning
Lecture-9

By: Somnath Mazumdar
Assistant Professor
sma.digi@cbs.dk

Overview

- Intro to Artificial Neural Networks
- Perceptron
  - TLU
  - MLP
  - Backpropagation

Artificial
Neural Networks

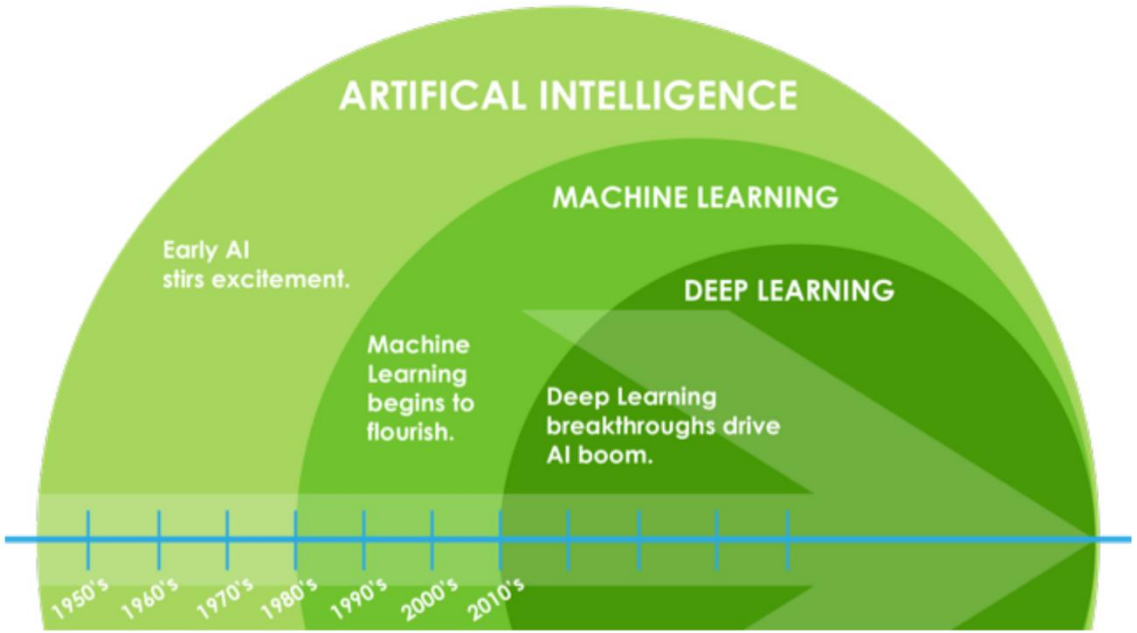
ML and Deep Learning History

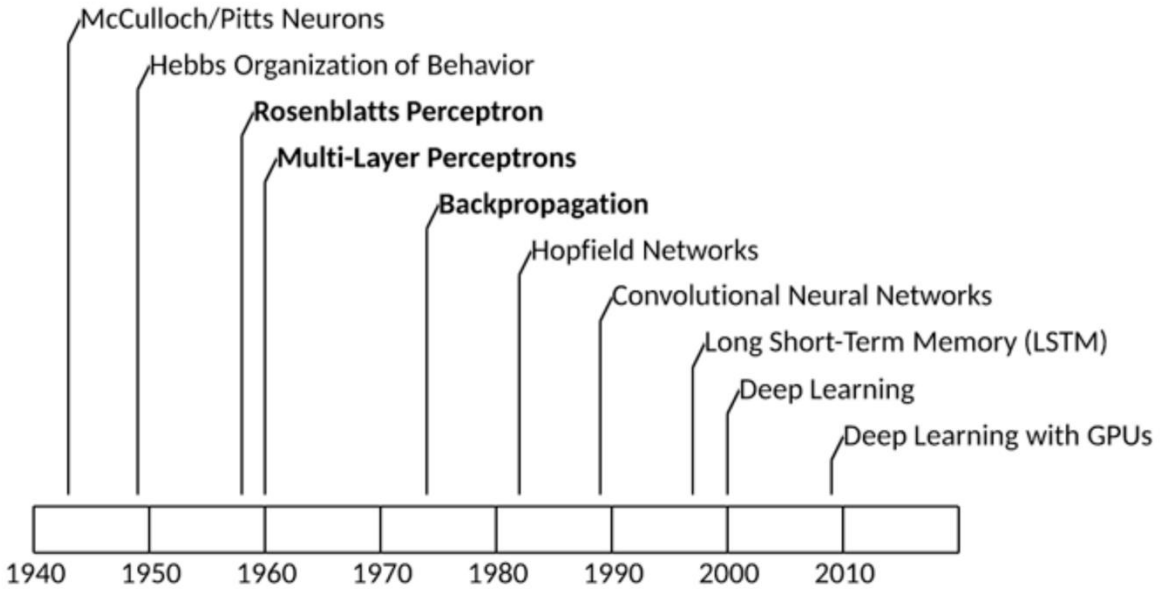
- 1950 and 1960s: Initial excitement.
  - Perceptron: linear classifier and stochastic gradient (roughly).
  - Drop in popularity: Realized limitations of linear models.

- 1970 and 1980s: Connectionism (brain-inspired ML)
  - Want “connected networks of simple units”.
  - Adding hidden layers z_i increases expressive power.
  - Success in optical character recognition.

- 1990s and early-2000s: drop in popularity.
  - Rise in popularity of logistic regression and SVMs with regularization and kernels.

ML Evolution





Overview of ML

- Components: Data, Model, Objective Function, Optimization Algorithm.
- Data: Input and output data
  - Common way of describing an input dataset is with a design matrix.
  - Issues: underfitting, overfitting, training error and test error.

- Model: To generate output.
  - Parametric models (linear regression)
  - Non-parametric models.

- Objective function: Real valued function that provides a measure of how wrong model is in terms of its ability to estimate relationship between inputs and outputs.

Overview of ML

- Objective function: real values function that provides a measure of how wrong the model is in terms of its ability to estimate the relationship between inputs and outputs.
  - Mean square error function, cross-entropy function (classification problems), Maximum Likelihood estimation.

- Optimization Algorithm: Minimizes or maximizes function f (x) by altering x.
  - Gradient descent: reduce f(x) by moving x in opposite sign of derivative.

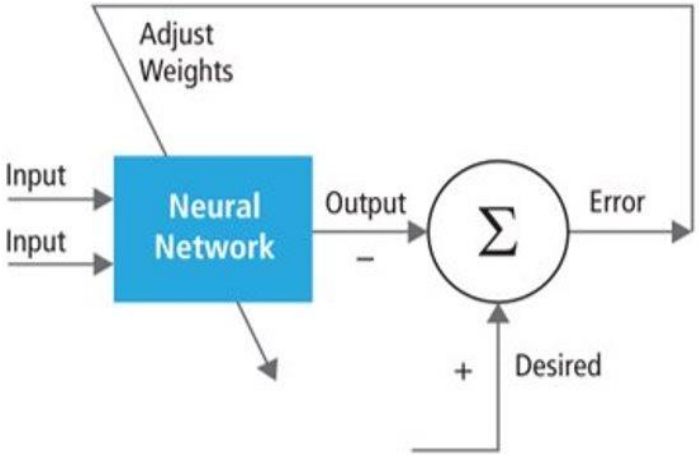
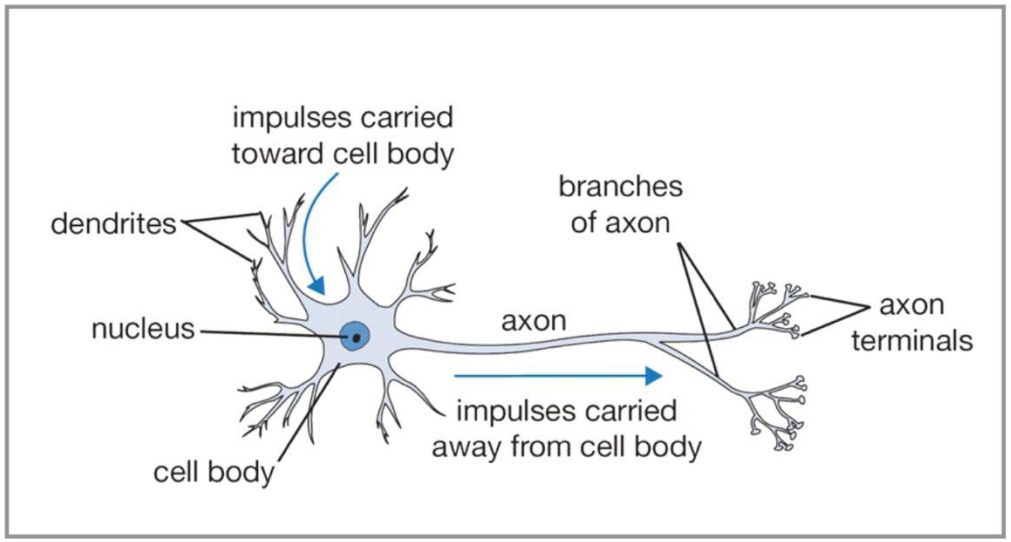
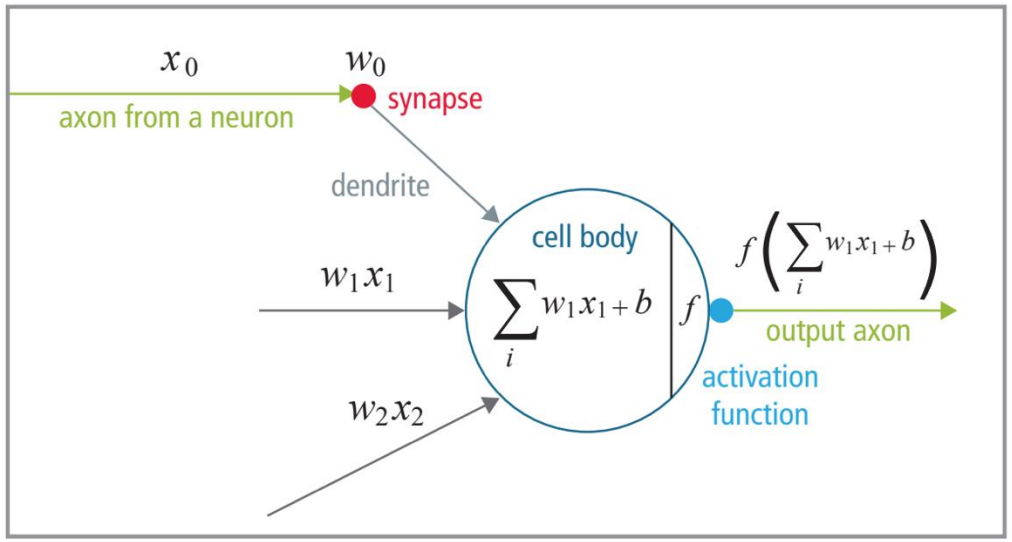


Image Src: Using Convolutional Neural Networks for Image Recognition By Samer Hijazi, Rishi Kumar, and Chris Rowen, IP Group, Cadence

From Biological to Artificial Neurons

- First introduced* in 1943 by Warren McCulloch and Walter Pitts.
- First ANN: To show of how biological neurons might work together in animal brains to perform complex computations using propositional logic.





- "Dendrites" takes input signal.
- "Axon" sends an output signal.

*Warren S. McCulloch and Walter Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity," The Bulletin of Mathematical Biology 5, no. 4 (1943): 115–113.
Image Src: Using Convolutional Neural Networks for Image Recognition By Samer Hijazi, Rishi Kumar, and Chris Rowen, IP Group, Cadence

Notation for Neural Networks

We have our usual supervised learning notation:

$$X = \begin{bmatrix} — x_1^T — \\ — x_2^T — \\ \vdots \\ — x_n^T — \end{bmatrix} \quad n \times d \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad n \times 1$$

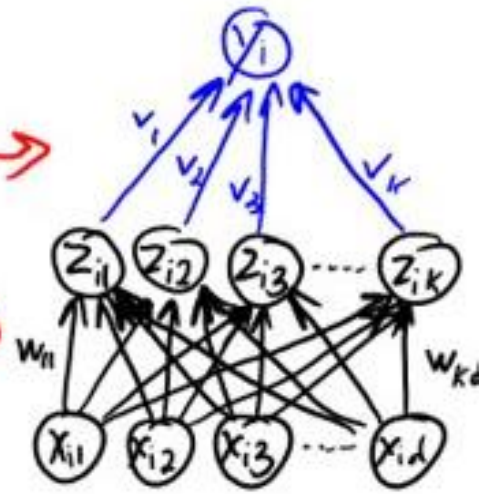
We have our latent features:

$$Z = \begin{bmatrix} — z_1^T — \\ — z_2^T — \\ \vdots \\ — z_n^T — \end{bmatrix} \quad n \times K$$

We have two sets of parameters:

$$V = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_K \end{bmatrix} \quad K \times 1 \qquad W = \begin{bmatrix} — w_1 — \\ — w_2 — \\ \vdots \\ — w_K — \end{bmatrix} \quad K \times d$$

Latent (**'hidden'**) features are computed from observed features using matrix factorization.



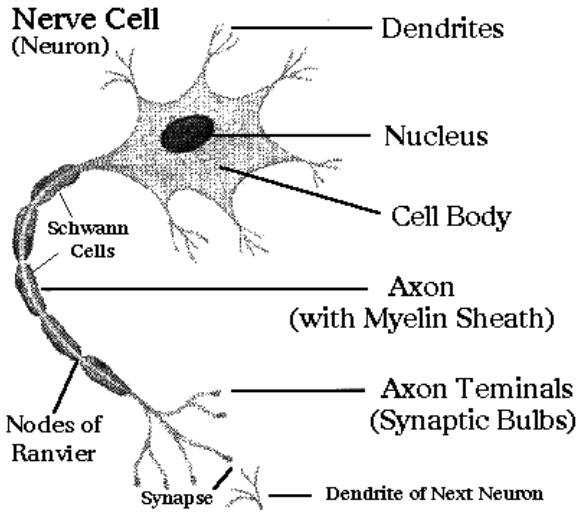
Linearity and Non-Linearity

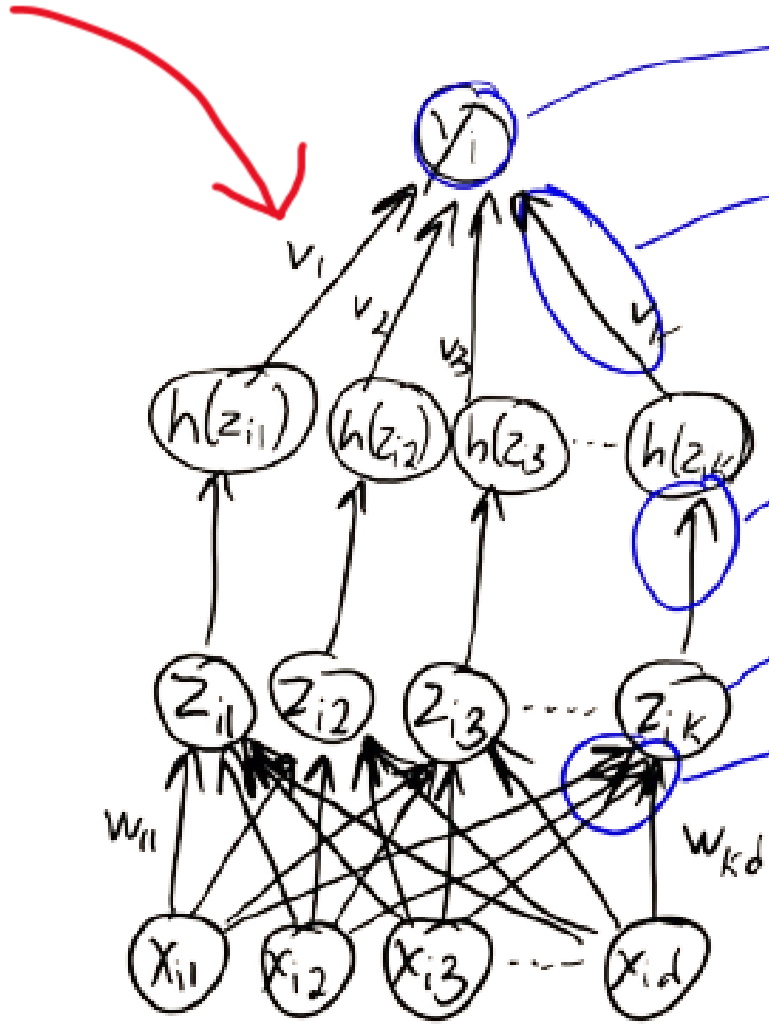
- Linear latent-factor model with linear regression.
  - Use features from latent-factor model: $z_i=Wx_i$
  - Make prediction using linear model: $y_i=v^Tz_i$

- Non-linearity: Transform z_i by non-linear function 'h': $z_i=Wx_i$ ==> $y_i=v^Th(z_i)$
  - Function 'h' transforms 'k' inputs to 'k' outputs.
  - Common choice for 'h': Sigmoid function

- Training: Apply SGD: Compute gradient of random example 'i', update both 'v' and 'W'.

Why “Neural Network”?





Predictions based on aggregation $v^T h(Wx_i)$ at y_i "neuron"

Synapse between z_{ik} and y_i "neuron"

binary signal $h(w_c^T x_i)$ sent along "axon"

Neuron aggregates signals: $w_c^T x_i$

"dendrites" for z_{ik} "neuron" are receiving x_{ij} values

Deep Learning

Linear model:

$$\hat{y}_i = w^T x_i$$

Neural network with 1 hidden layer:

$$\hat{y}_i = v^T h(Wx_i)$$

where $z_i = Wx_i$

Neural network with 2 hidden layers:

$$\hat{y}_i = v^T h(W^{(2)} h(W^{(1)} x_i))$$

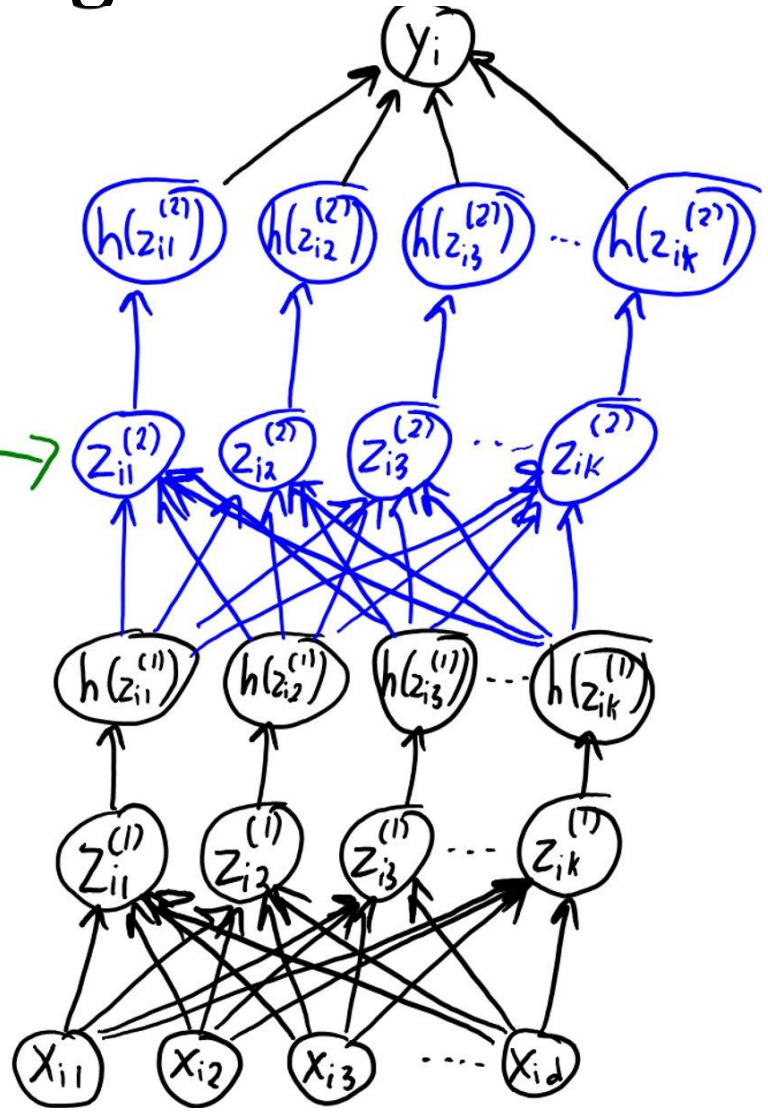
where $z_i^{(1)} = W^{(1)} x_i$ and $z_i^{(2)} = W^{(2)} h(W^{(1)} x_i)$

Neural network with 3 hidden layers

$$\hat{y}_i = v^T h(W^{(3)} h(W^{(2)} h(W^{(1)} x_i)))$$

where $z_i^{(1)} = W^{(1)} x_i$, $z_i^{(2)} = W^{(2)} h(W^{(1)} x_i)$, and $z_i^{(3)} = W^{(3)} h(W^{(2)} h(W^{(1)} x_i))$

Deep learning:



Second "layer" of latent features

You can add more "layers" to go "deeper"

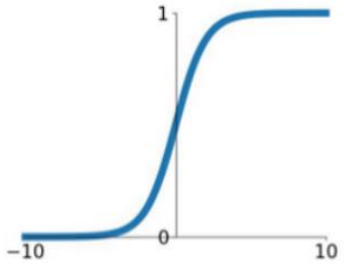
- For 'm' layers, we could use: $\hat{y}_i = w^T \left(\prod_{\ell=1}^{m} h(W^{(\ell)} x_i) \right)$

- NOTE: Non-linearity: Transform z_i by non-linear function 'h': $z_i = Wx_i$ ==> $y_i = v^T h(z_i)$

Activation Functions
(See Bonus Slides for more)

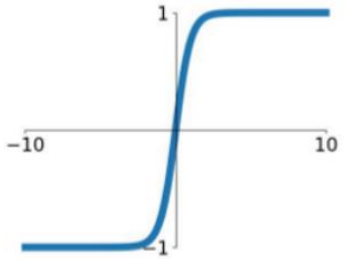
Sigmoid

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



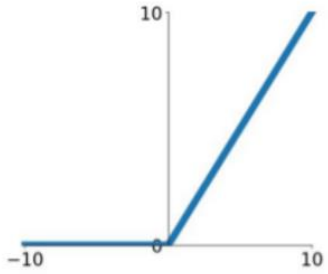
tanh

$$\tanh(x)$$



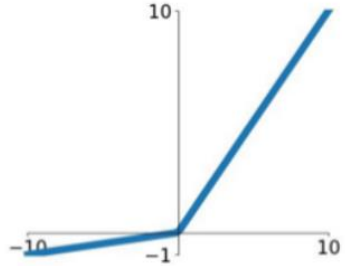
ReLU

$$\max(0, x)$$



Leaky ReLU

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

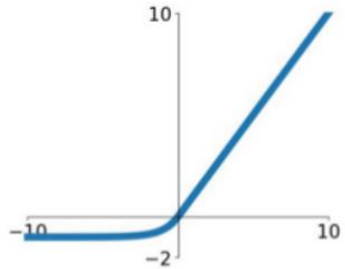


Maxout

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

ELU

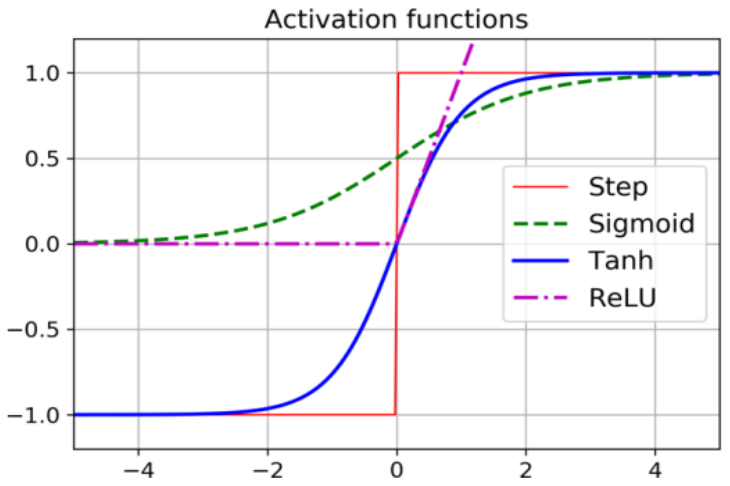
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

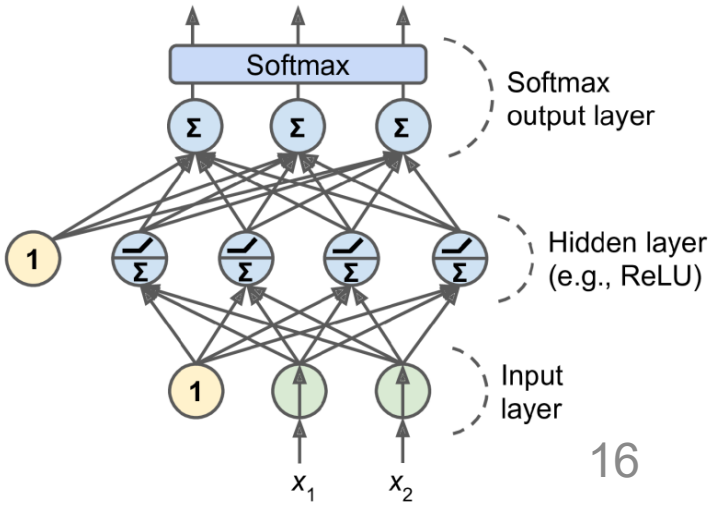


Activation Function

- Defines how weighted sum of input is transformed to output from a node(s).
  - All hidden layers use same activation function.
  - Output layer use a different activation function from hidden layers.

- Large DNN with nonlinear activations can theoretically approximate any continuous function.
  - Sigmoid (S-shaped) activation function was good.
  - ReLU works better in ANNs.
  - Softmax ensure all estimated probabilities between 0 and 1 and they add up to 1 (required for exclusive classes).





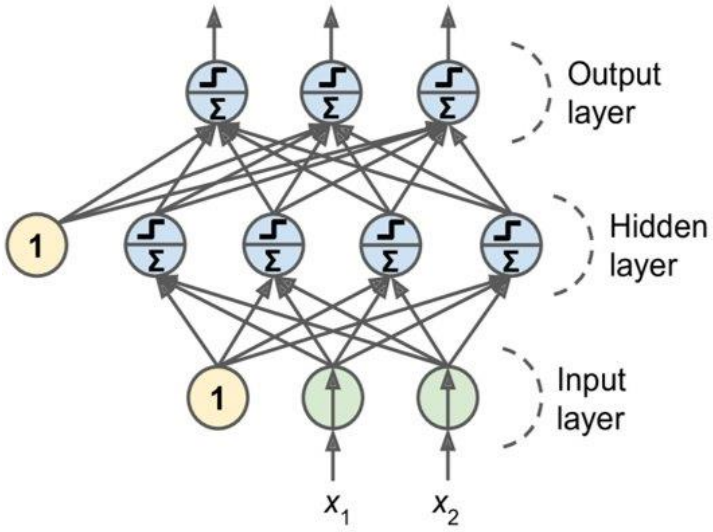
Bias

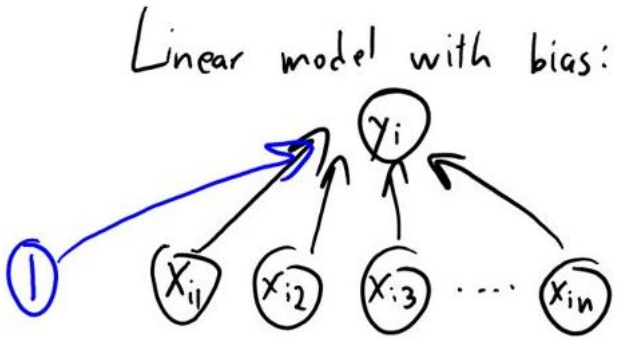
- Generalization error consists of three different errors: Bias, Variance, Irreducible error (due to noisiness of data).

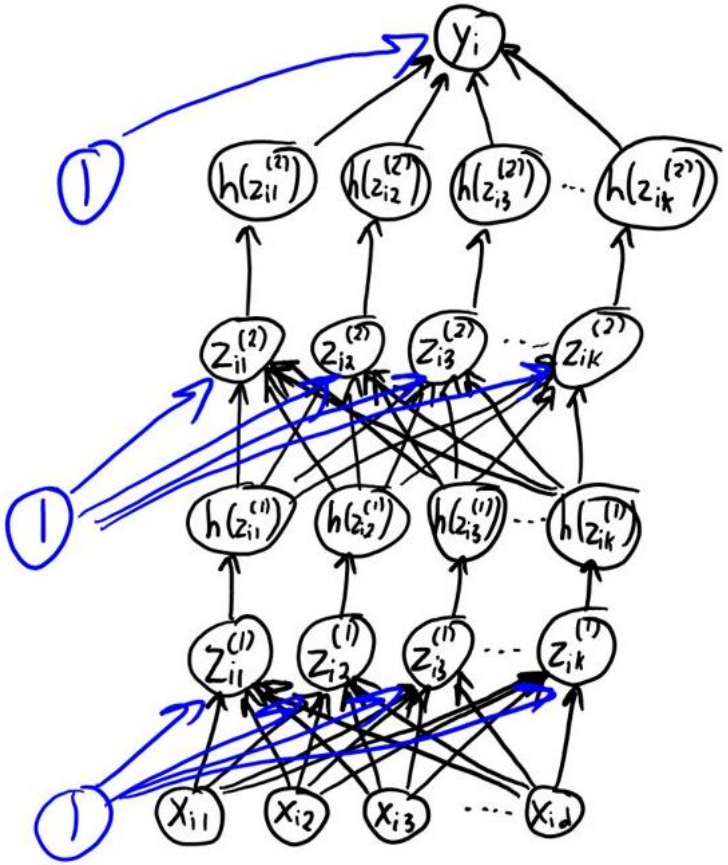
- Bias happen due to wrong assumptions.
- High-bias model is most likely to underfit the training data.
- Example: Assuming data is linear when it is quadratic.

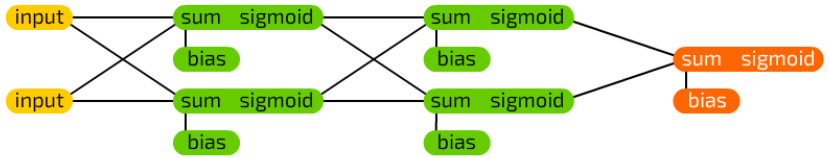
- Note: If you trains a linear SVM model using the LinearSVC class (of Sci-kit).
  - LinearSVC class regularizes bias term.

Adding Bias

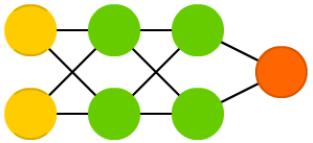

Output layer
Hidden layer
Input layer
1
1
x_1
x_2

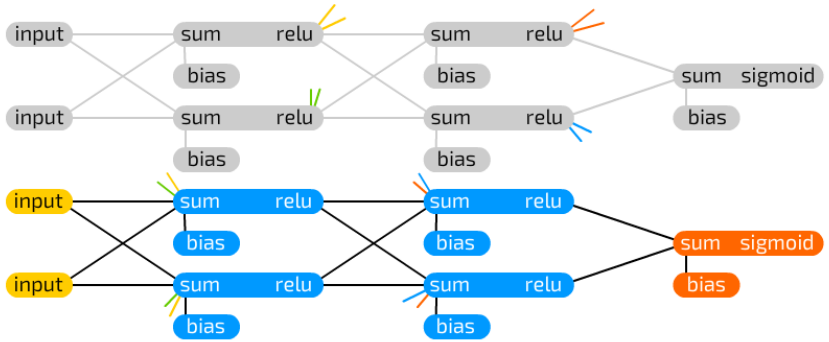

Linear model with bias:
y_i
1
x_{i1}
x_{i2}
x_{i3}
x_{in}


y_i
1
$h(z_{i1}^{(2)})$
$h(z_{i2}^{(2)})$
$h(z_{i3}^{(2)})$
$h(z_{ik}^{(2)})$
$z_{i1}^{(2)}$
$z_{i2}^{(2)}$
$z_{i3}^{(2)}$
$z_{ik}^{(2)}$
1
$h(z_{i1}^{(1)})$
$h(z_{i2}^{(1)})$
$h(z_{i3}^{(1)})$
$h(z_{ik}^{(1)})$
$z_{i1}^{(1)}$
$z_{i2}^{(1)}$
$z_{i3}^{(1)}$
$z_{ik}^{(1)}$
1
x_{i1}
x_{i2}
x_{i3}
x_{id}

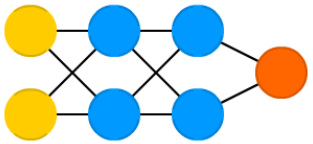


Deep Feed Forward Example

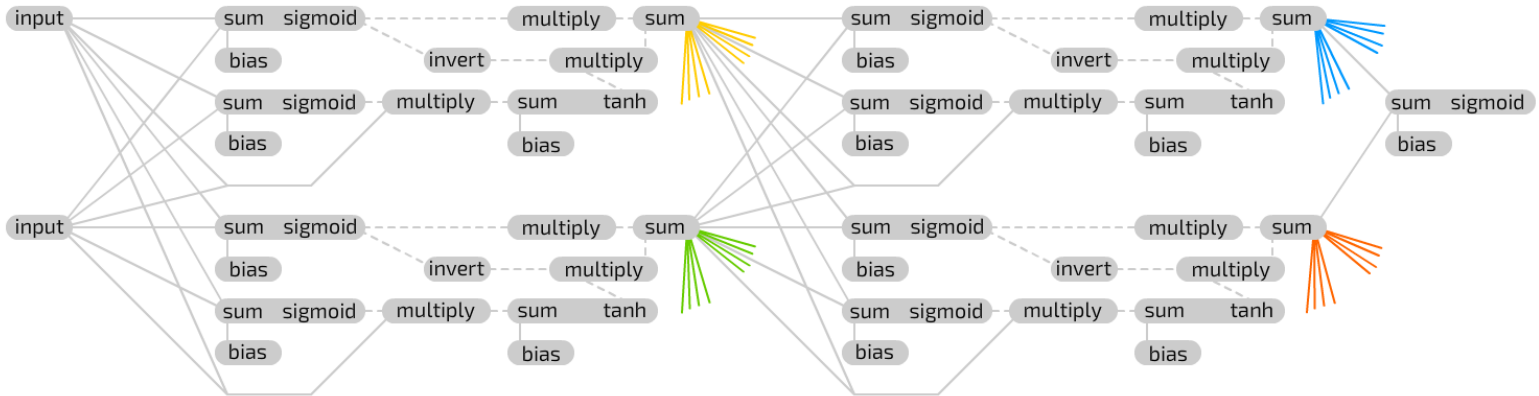




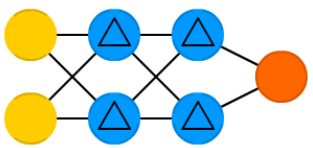
Deep Recurrent Example
(previous iteration)

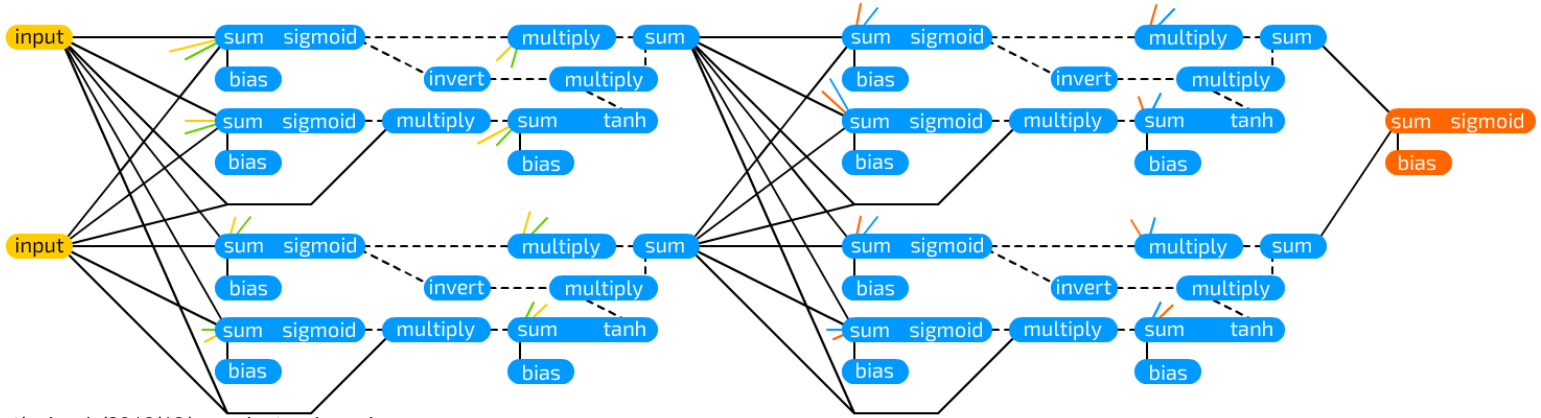


Deep Recurrent Example



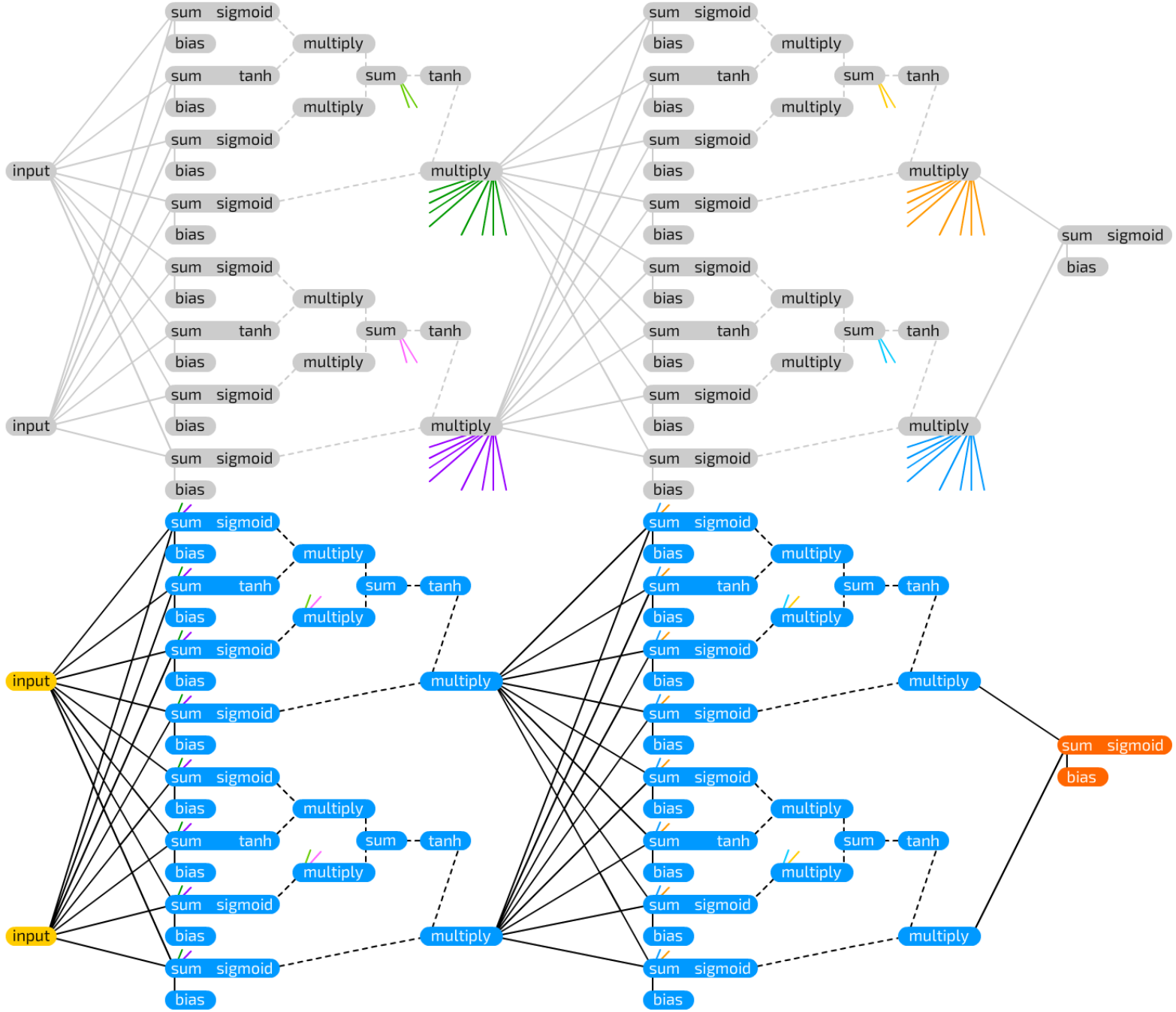
Deep GRU Example
(previous iteration)



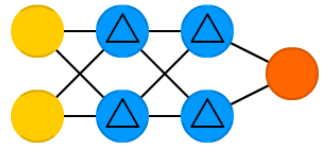


Deep GRU Example

https://www.asimovinstitute.org/wp-content/uploads/2016/12/neuralnetworkgraphs.png



Deep LSTM Example
(previous iteration)



Deep LSTM Example

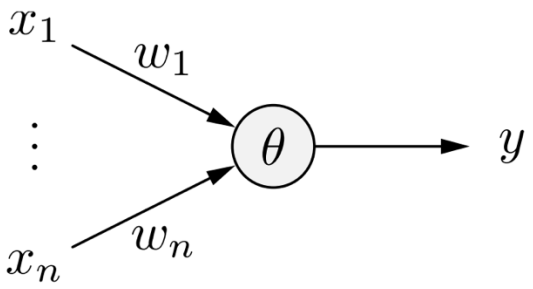
https://www.asimovinstitute.org/wp-content/uploads/2016/12/neuralnetworkgraphs.png

Threshold Logic Units, Perceptron, & Multilayer Perceptron

Threshold Logic Units (TLUs)

- TLU is a processing unit for numbers with n inputs $x_1,...,x_n$ and one output y.
- Unit has a threshold θ and each input x_i is associated with a weight w_i.
- Threshold logic unit computes the function.

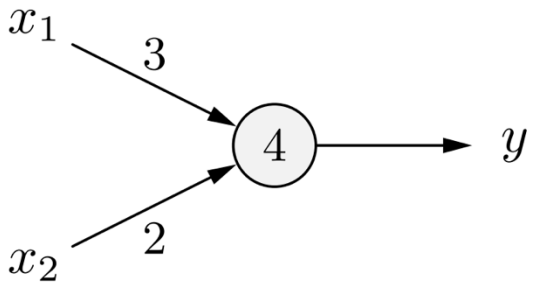
$$y = \begin{cases} 1, & \text{if} \quad \sum_{i=1}^{n} w_i x_i \geq \theta, \\ 0, & \text{otherwise.} \end{cases}$$



- TLUs mimic thresholding behaviour of biological neurons in a (very) simple fashion.

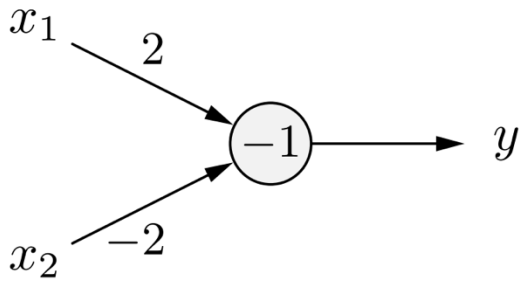
Example

- Threshold logic unit for the conjunction x1∧x2.



x_1	x_2	$3x_1 + 2x_2$	y
0	0	0	0
1	0	3	0
0	1	2	0
1	1	5	1

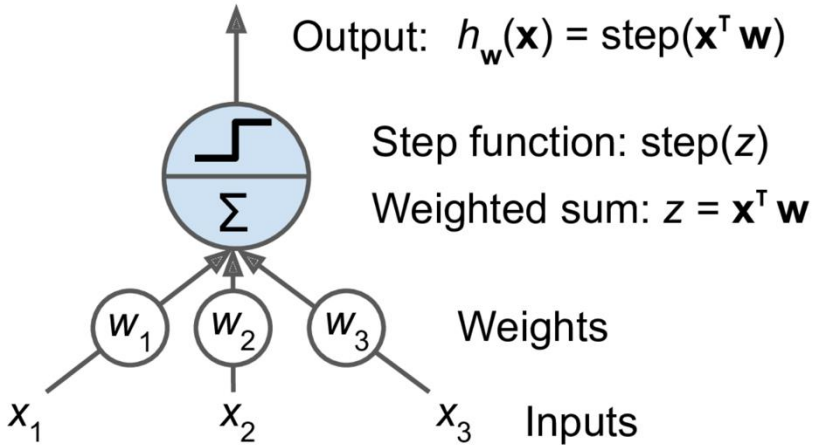
- Threshold logic unit for the implication x2→x1.



x_1	x_2	$2x_1 - 2x_2$	y
0	0	0	1
1	0	2	1
0	1	−2	0
1	1	0	1

Perceptron

- Simplest ANN architectures.
- Invented in 1957 by Frank Rosenblatt.
- Based on a slightly different artificial neuron called a threshold logic unit (TLU), or linear threshold unit (LTU).



- Inputs and output are numbers, and each input connection is associated with a weight.
- TLU computes a weighted sum of its inputs, then applies a step function to that sum and outputs result.

Perceptron

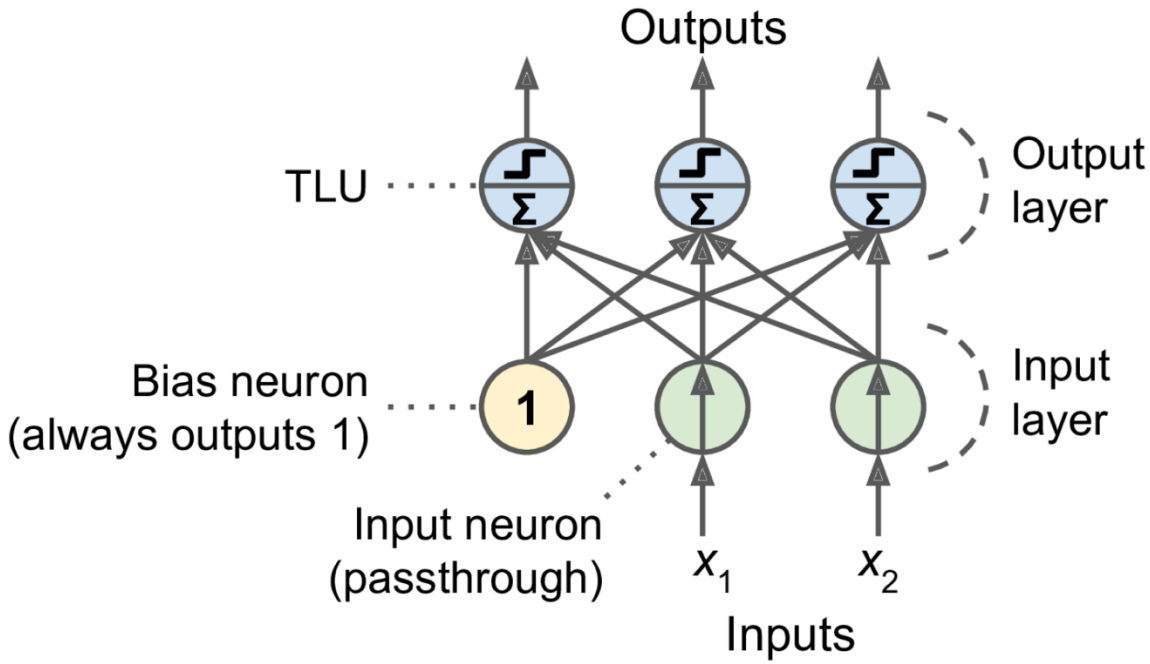
- Single TLU can be used for simple linear binary classification.
- Perceptron is composed of a single layer of TLUs.
- Each TLU connected to all inputs.
  - TLU computes a linear combination of inputs.
  - If result exceeds a threshold, it outputs positive class.

  - Fully connected /dense layer: All neurons in a layer connected to every neuron in previous layer.

- An extra bias feature is generally added ($x_0 = 1$):
  - Typically represented using a special type of neuron called a bias neuron.
  - Outputs 1 always.

Example

- Perceptron with two inputs and three outputs
- This Perceptron can classify instances simultaneously into three different binary classes, which makes it a multi-output classifier.



Perceptron

- How is a Perceptron trained?
  - Perceptron training algorithm was largely inspired by Hebb's rule (or Hebbian learning).
- Process:
  1. Perceptron is fed one training instance at a time, and for each instance it makes its predictions.
  2. For every output neuron that produced a wrong prediction, it reinforces the connection weights from the inputs that would have contributed to the correct prediction.
- Heaviside step function used in Perceptrons.
- Common step functions used in Perceptrons (assuming threshold = 0)

$$\text{heaviside}\,(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \qquad \text{sgn}\,(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$

signum function extracts the sign of a real number z.

- Perceptron and SGDClassifier both have similar implementation.

Perceptron

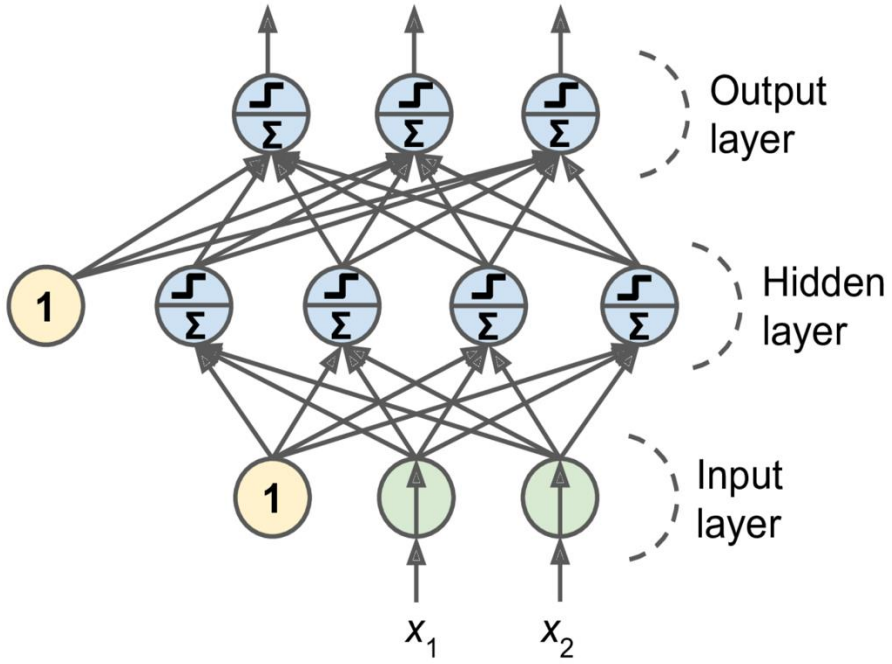
- Perceptron incapable of learning complex patterns:
  - Because decision boundary of each output neuron is linear (just like Logistic Regression classifiers).

- Logistic Regression Vs Perceptrons:
  - Perceptrons do not output a class probability.
  - Perceptrons make predictions based on a hard threshold.

- Weaknesses:
  - Incapable of solving Exclusive OR (XOR) classification problem
  - Soln: Can be solved by stacking multiple Perceptrons
    - Resulting ANN is called a Multilayer Perceptron (MLP).

Multilayer Perceptron: MLP

- MLP is composed of one input layer, one or more layers of TLUs, called hidden layers*, and a layer of TLU called output layer.
- Layers close to input layer are called lower layers
- Layers close to outputs are called upper layers.
- Every input/hidden layer includes a bias neuron and is fully connected to next layer.



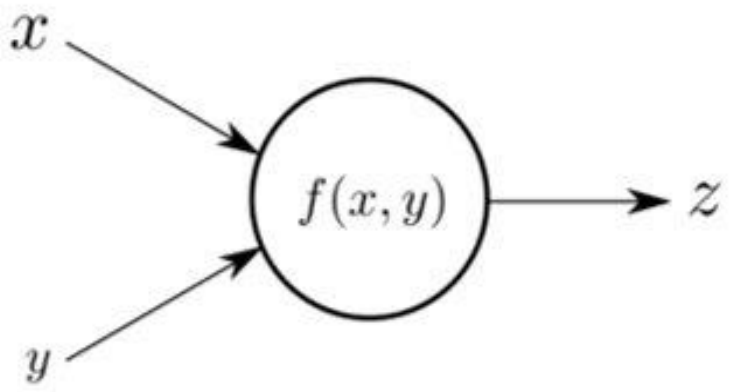
MLP with 2-inputs, one hidden layer of 4-neurons, and 3-output neurons

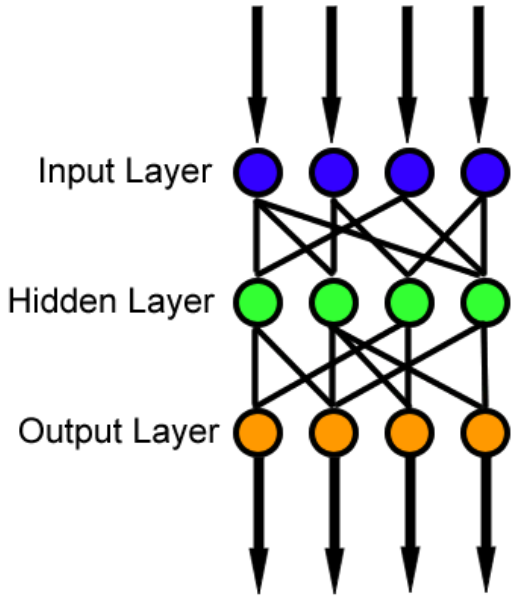
*When an ANN contains a deep stack of hidden layers it is called a deep neural network (DNN).

Feed-Forward Networks
& Backpropagation

Feed-Forward Networks (Prediction)

- There is no cyclic connection between network layers in Feed-forward networks.
- Input flows forward towards output via several hidden layers.
- Feedforward NN: The signal flows in one direction (inputs-->outputs).

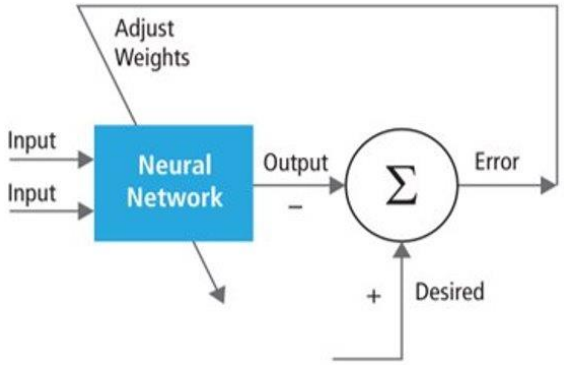




https://en.wikipedia.org/wiki/Feedforward_neural_network

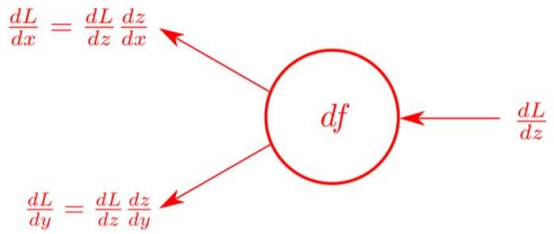
Backpropagation

- Backpropagation computes neural network gradient via chain rule.
- Computing gradient is known as “backpropagation”.



- With squared loss, objective function is: $f(v, W) = \frac{1}{2}\sum_{i=1}^{n}(v^T h(Wx_i) - y_i)^2$
- Usual training procedure: stochastic gradient.
  - Compute gradient of random example ‘i’, update both ‘v’ and ‘W’.
  - Highly non-convex and can be difficult to tune.

- Backward pass:
  - Initialize all hidden layers’ connection weights randomly, or else training will fail.
  - For this algorithm to work for MLP’s architecture the step function was replaced with the logistic (sigmoid) function, σ(z) = 1 / (1 + exp(–z)).



*David Rumelhart et al. “Learning Internal Representations by Error Propagation,” (Defense Technical Information Center technical report, September 1985).

Backpropagation

- Cost of backpropagation:
  - Forward pass dominated by matrix multiplications by W(1), W(2), W(3), and 'v'.
  - If you have 'm' layers and all z_i have 'k' elements, cost would be $O(dk + mk^2)$.
  - Backward pass has same cost as forward pass.
  - For multi-class or multi-label classification, you replace 'v' by a matrix.

Linear model:

$\hat{y}_i = w^T x_i$

Neural network with 1 hidden layer:

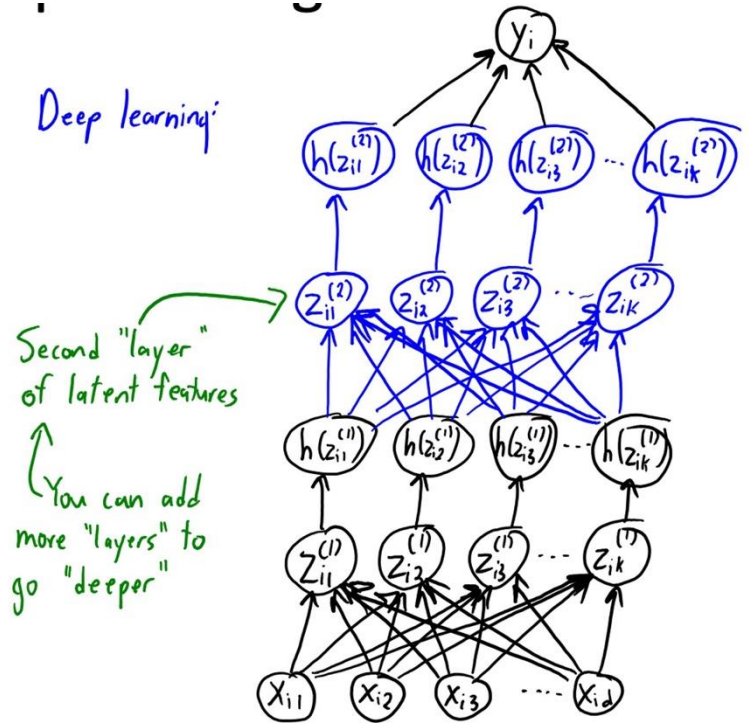
$\hat{y}_i = v^T h(W x_i)$, with $z_i = h(W x_i)$

Neural network with 2 hidden layers:

$\hat{y}_i = v^T h(W^{(2)} h(W^{(1)} x_i))$, with $z_i^{(1)} = h(W^{(1)} x_i)$ and $z_i^{(2)} = h(W^{(2)} z_i^{(1)})$

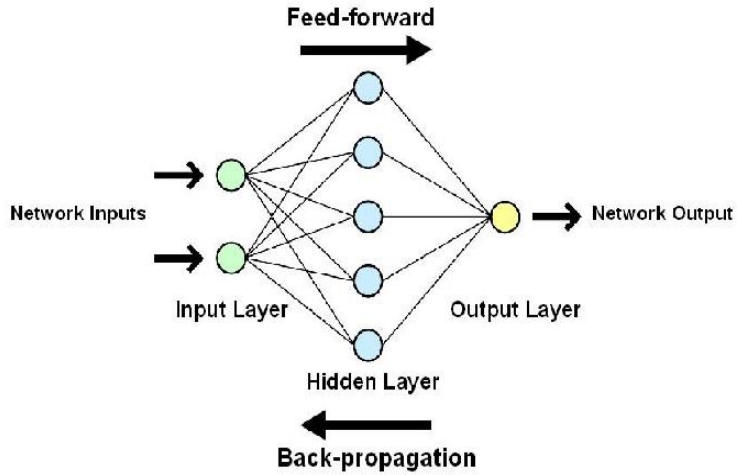
Neural network with 3 hidden layers

$\hat{y}_i = v^T h(W^{(3)} h(W^{(2)} h(W^{(1)} x_i)))$, with $z_i^{(1)} = h(W^{(1)} x_i)$, $z_i^{(2)} = h(W^{(2)} z_i^{(1)})$, and $z_i^{(3)} = h(W^{(3)} z_i^{(2)})$



Backpropagation (Computing gradients)

- An efficient technique for computing gradients automatically.
- Backpropagation training algorithm* to train MLPs.
  1. For each training instance: backpropagation algorithm first makes a prediction (forward pass) and measures error.
  2. Then goes through each layer in reverse to measure error contribution from each connection (backward pass).
  3. Finally tweaks connection weights to reduce error (Gradient Descent step).



*David Rumelhart et al. “Learning Internal Representations by Error Propagation,” (Defense Technical Information Center technical report, September 1985).

Gradient Problem

- Vanishing Gradient Problem: When gradients are very small, they can diminish as they are propagated back through network, leading to minimal or no updates to weights in initial layers.
- Reasons: Choice of activation functions, NN architecture.
- Solution: Activation Functions: ReLU, Proper weight initialization, Batch Norm.

- Exploding Gradient Problem: When gradients of network's loss with respect to parameters (weights) become excessively large. The "explosion" of the gradient can lead to numerical instability and inability of the network to converge to a suitable solution.
- How to check? Monitore gradients and check loss function showing unusually large fluctuations or becoming NaN.
- Solution: Proper activation function choice, Batch Norm.

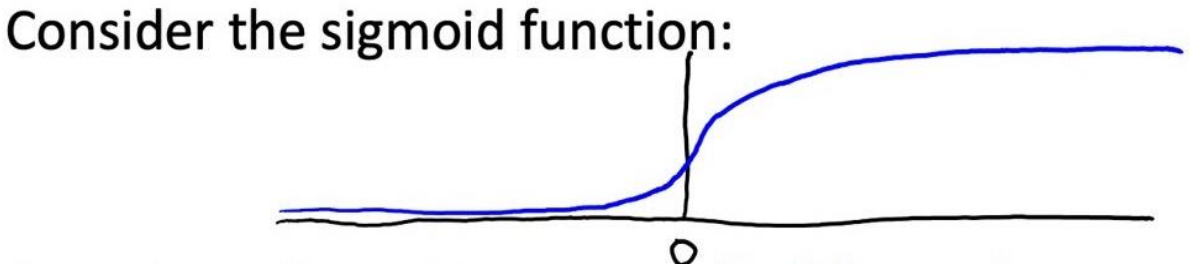
Extra Backpropagation Materials

- Example: https://playground.tensorflow.org/
- Video : https://www.youtube.com/watch?v=Ilg3gGewQ5U
- Step by step Example: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

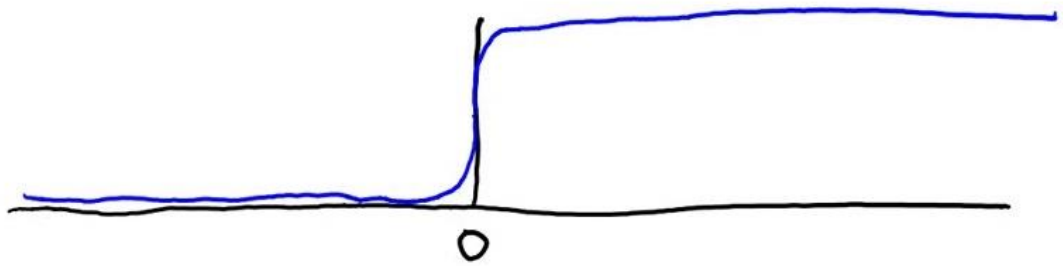
Bonus Slides

Vanishing Gradient Problem

- Vanishing gradients make it difficult to know which direction the parameters should move to improve the cost function.
- Exploding gradients can make learning unstable.



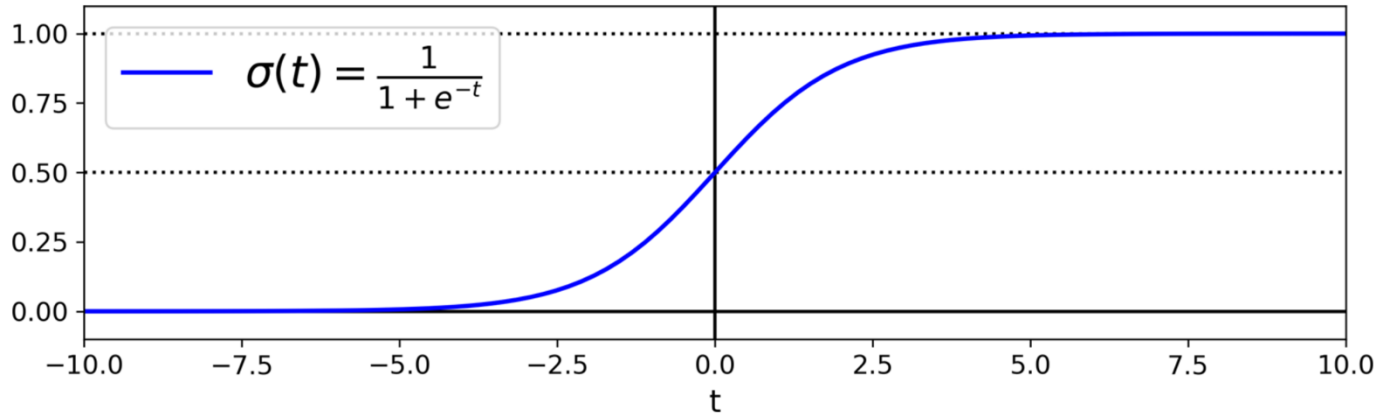
- Away from the origin, the gradient is nearly zero.
- The problem gets worse when you take the sigmoid of a sigmoid:



- In deep networks, many gradients can be nearly zero everywhere.

Sigmoid Function

- Logistic (noted σ(·)) is a sigmoid function (i.e., S-shaped) that outputs between 0 and 1 (remember Estimating Probabilities by Logistic Regression).

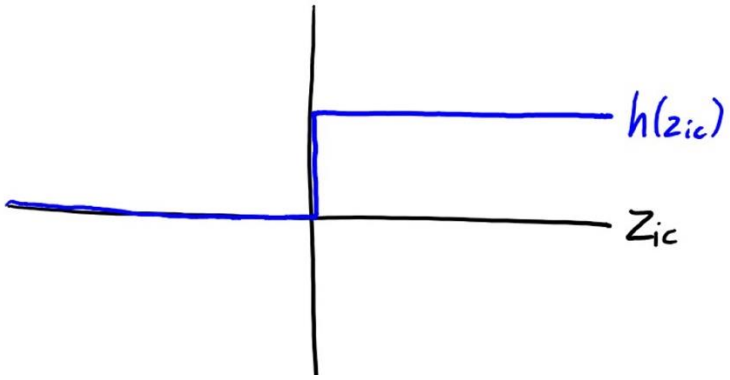


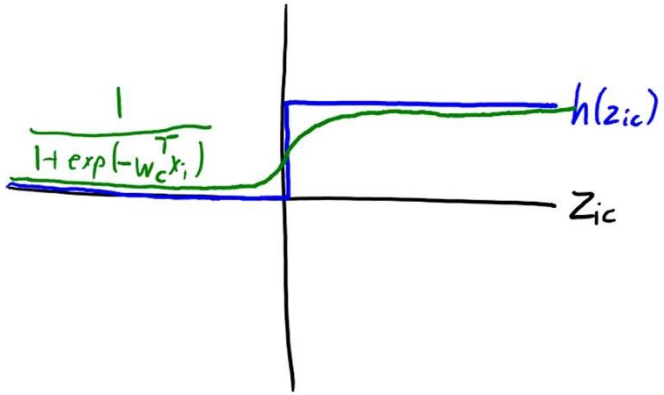
- Sigmoid function is a type of activation function (or squashing function).
- Squashing functions limit output to a range between 0 and 1.
- An activation function determines output behaviour of each node, or “neuron” in an ANN.
- Used in testing of ANNs.

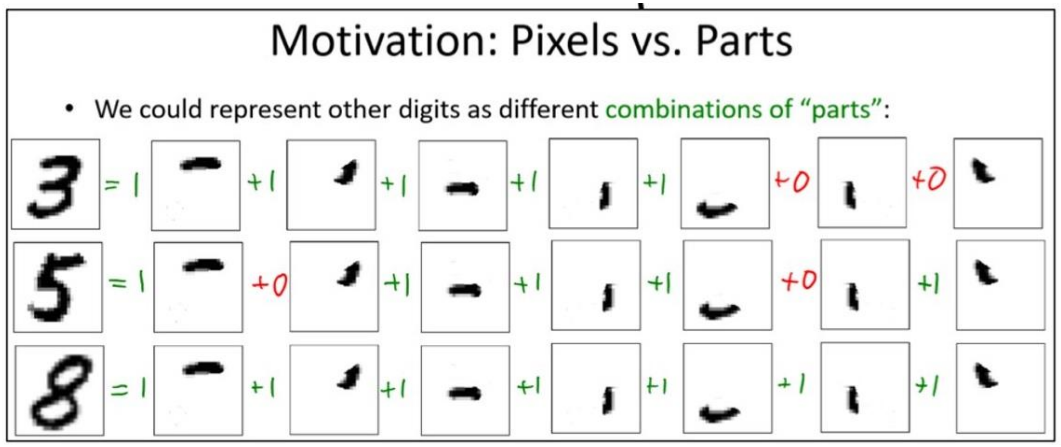
Sigmoid

- Consider setting ‘h’ to define binary features z_i using:

$$h(z_{ic}) = \begin{cases} 1 & \text{if } z_{ic} \geq 0 \\ 0 & \text{if } z_{ic} < 0 \end{cases}$$



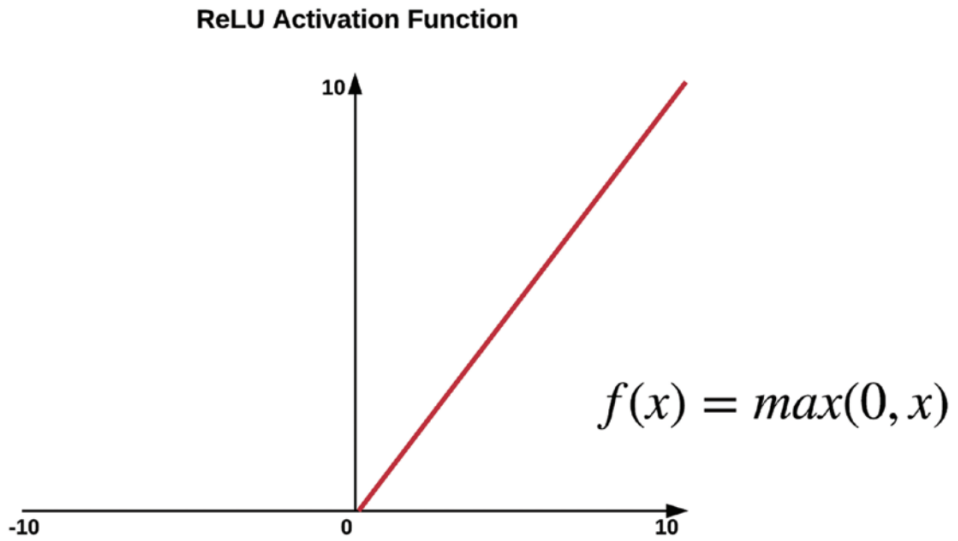




- Each $h(z_i)$ can be viewed as binary feature.
- We can make 2^k objects by all the possible “part combinations”.
- Hard to optimize (non-differentiable/discontinuous).
- Sigmoid is a smooth approximation to these binary features.

Rectified Linear Units (ReLU)

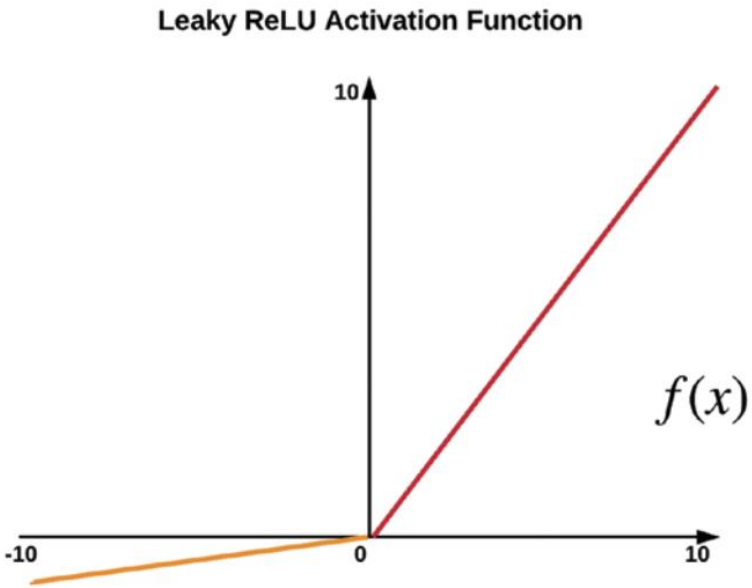
- ReLU mitigate the vanishing and exploding gradient problem.
- ReLU offers improvement on the tanh and sigmoid activation functions.
- ReLU works:
  - Set the activation to 0 for values x < 0
  - Set a linear slope of 1 when values x > 0



Bisong E. (2019) More on Optimization Techniques. In: Building Machine Learning and Deep Learning Models on Google Cloud Platform. Apress, Berkeley, CA

Leaky ReLU

- Some gradients can still die out during backpropagation with a large learning rate –-> use Leaky ReLU.
- Leaky ReLU works by setting the activation to a small negative slope when the value $x < 0$.



$$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

Bisong E. (2019) More on Optimization Techniques. In: Building Machine Learning and Deep Learning Models on Google Cloud Platform. Apress, Berkeley, CA