ADALINE AND PERCEPTRON REVIEW

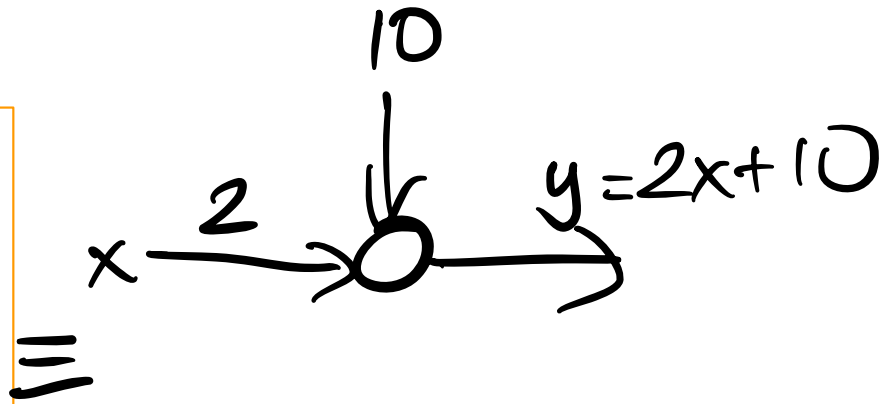
Linear Regression Problems

A single neuron can represent a **linear regression** model**.** The weights and bias correspond to the linear regression coefficients and the activation function is linear**.**

E.g. Neuron computes weighted input:

$y = 2x_1+10$

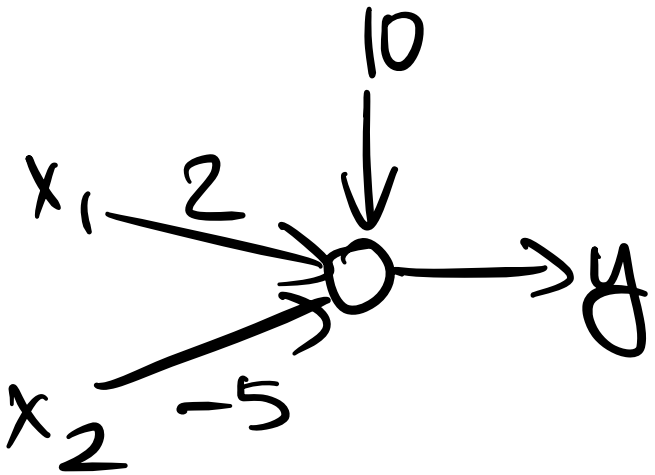
How many input, how many outputs and what are the weights?



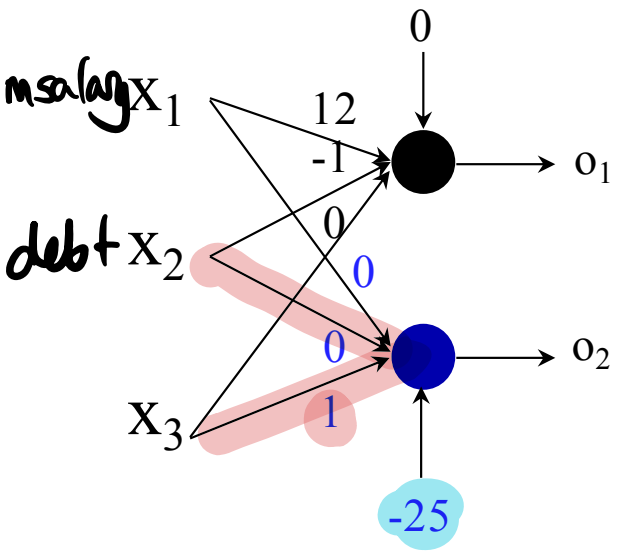
E.g. Neuron computes weighted input:

$y = 2x_1 - 5x_2 + 10$

How many input, how many outputs and what are the weights?



Simple Classification Problem



Lets look at the credit approval problem where we want to output 1 for good applicants and 0 for applications to deny (because looking at the data, we think that they may fail to pay back the received credit).

Lets consider three features x1: monthly salary, x2: existing debt, x3: age

E.g. $Neuron_1$ should output 1 if 12xmonthly salary greater than or equal to debt

$$12x_1 - x_2 \geq 0$$

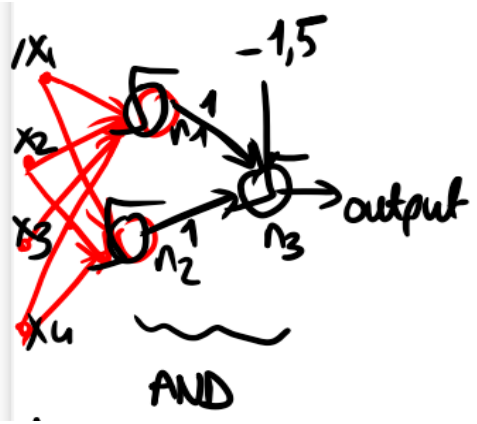
Threshold activation:

f(net) = 1 if net ≥ 0
= 0 if net < 0

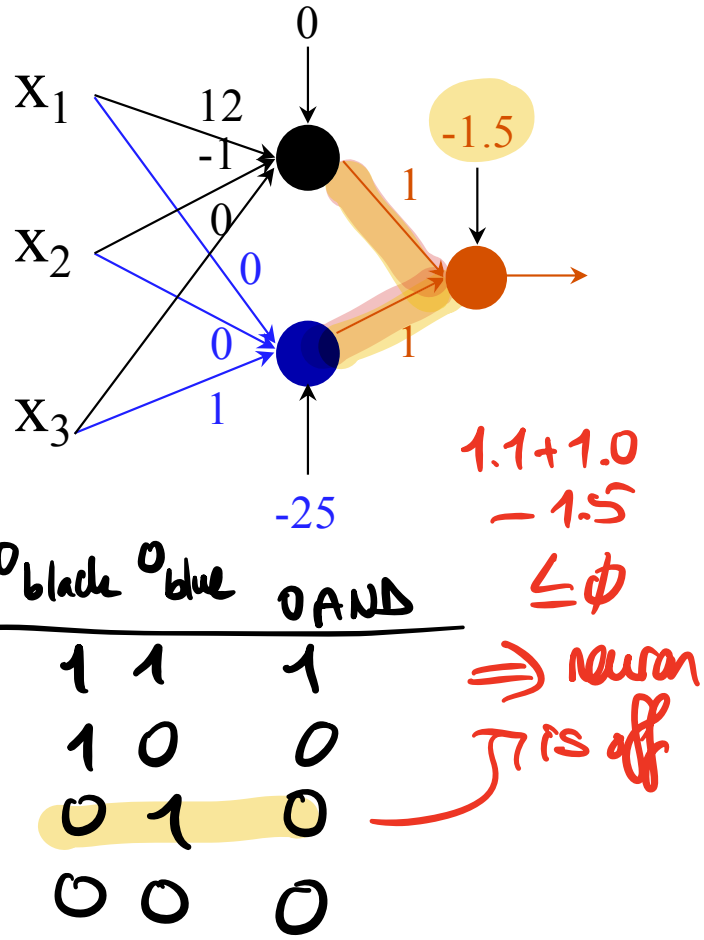
E.g. $Neuron_2$ should output 1 if age is greater than or equal to 25:

$$x_3 - 25 \geq 0$$

Neuron3 now can take the outputs of neuron1 and neuron2 and make a more complex decision. An AND-node outputs 1, if **both** neuron1 and neuron2 outputs 1.

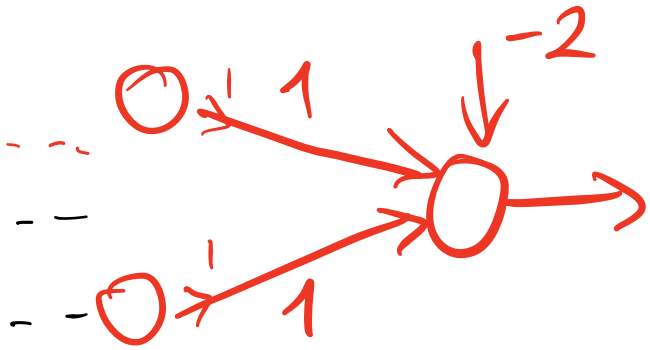


In general, individual neurons each accomplish small tasks, like detecting simple visual patterns or important features in the credit application problem.

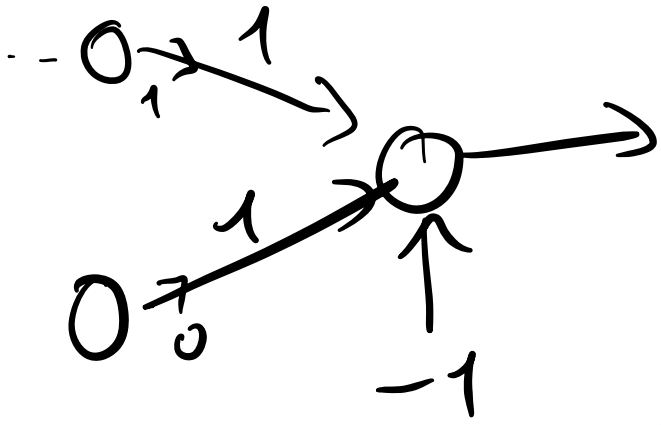


Later layers can detect more complex patterns using the output of the earlier layer

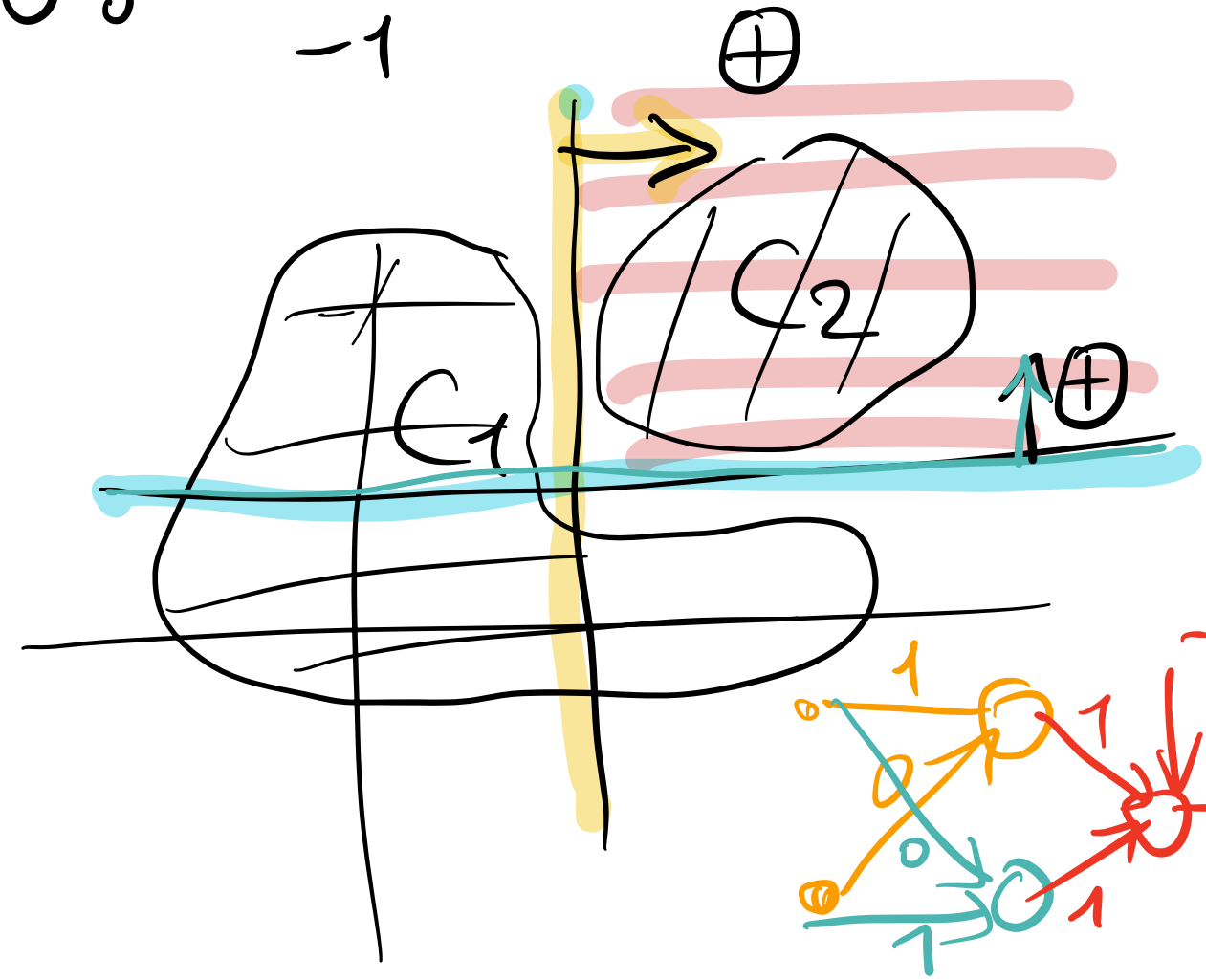
- a neuron in the second layer may be active if both/some neurons in the previous layer are active etc.



AND netw.



OR netw.



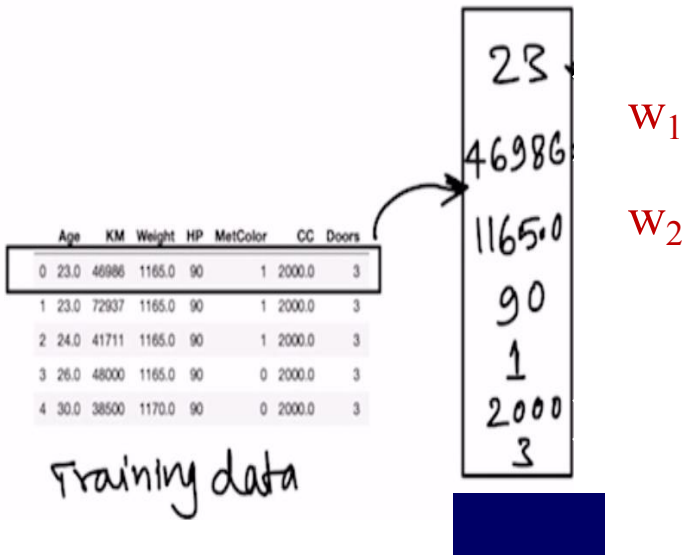
Neural Networks

Given an input (an image, text, vector, or sequence…), each layer of neurons compute simple functions of their input based on the current parameters (weights) of the network, passing their output as input to next layer.

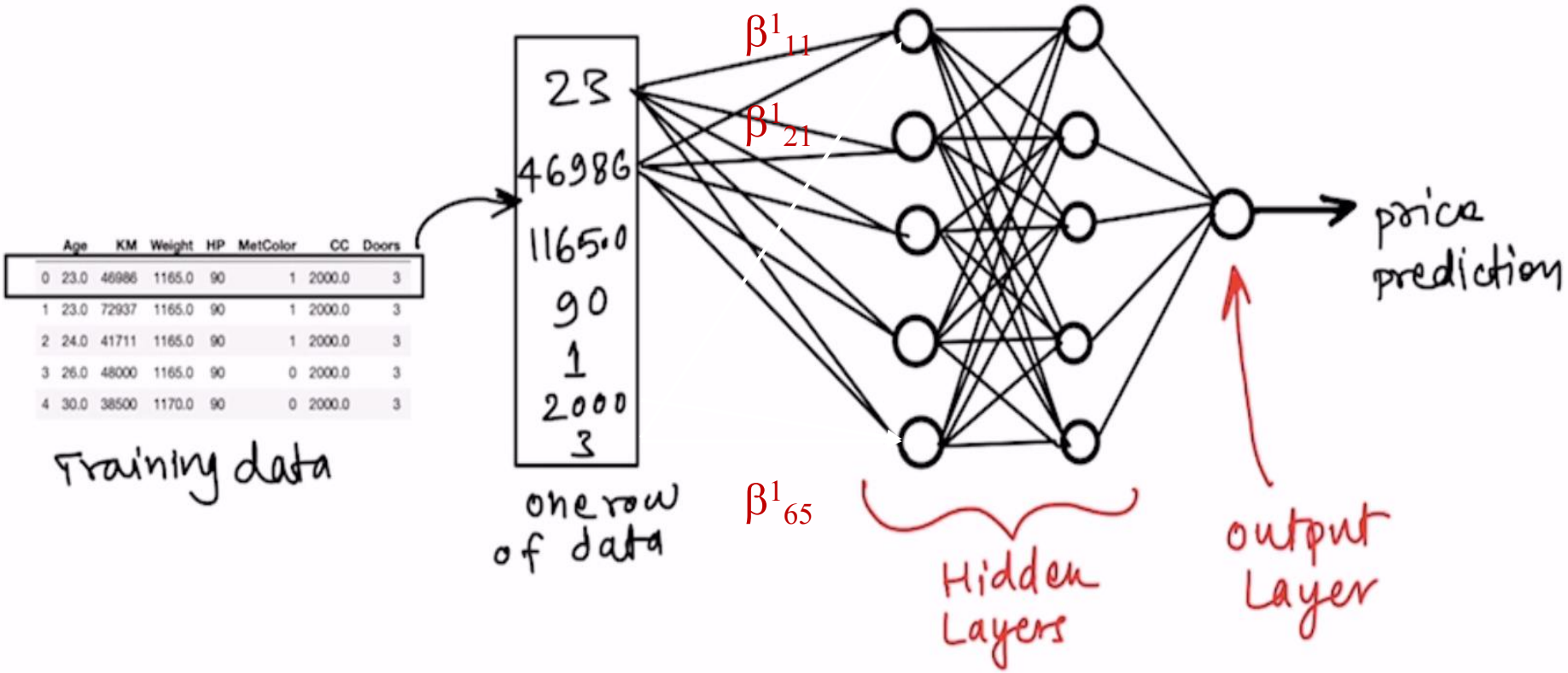
Measure error between target and what is predicted

- E.g. mean squared error

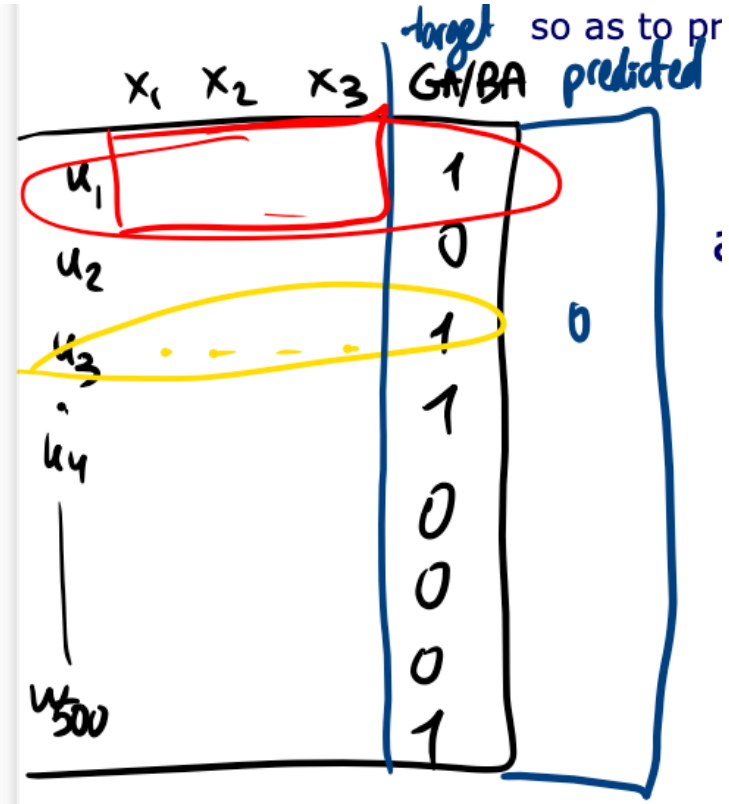
Backpropagation: Modify the network weights so as to reduce the error



Multi-layer regression network



Learning Overview



- Decide on the architecture of the network
- Initialize all weights (incl. biases) to small random numbers
- **Loop until stopping criteria:**
  - Pick a sample point (e.g. u_3)
  - Forward pass (pass through the network)
  - Measure error (e.g. MSE)
  - Update each weight to reduce the error
- **Backpropagation:**
  - Gradient descent to minimize the error with respect to each weight, starting from the output layer all the way to the 1st layer weights

Artificial Neural Networks

MultiLayer Perceptrons

Backpropagation

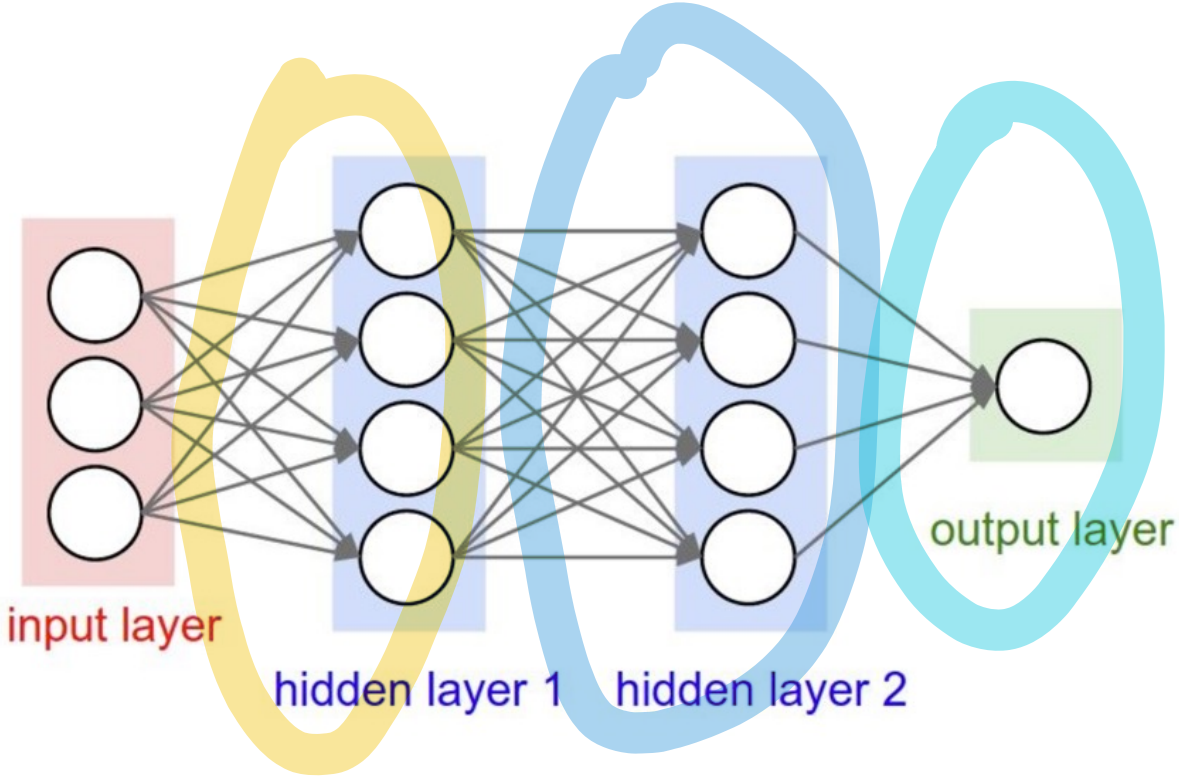
Berrin Yanikoglu

4/2024

Multilayer Perceptron

- In Multilayer perceptrons (a.k.a FeedForward Neural Networks), there may be one or more hidden layer(s) which are called hidden since they are not observed from the outside.

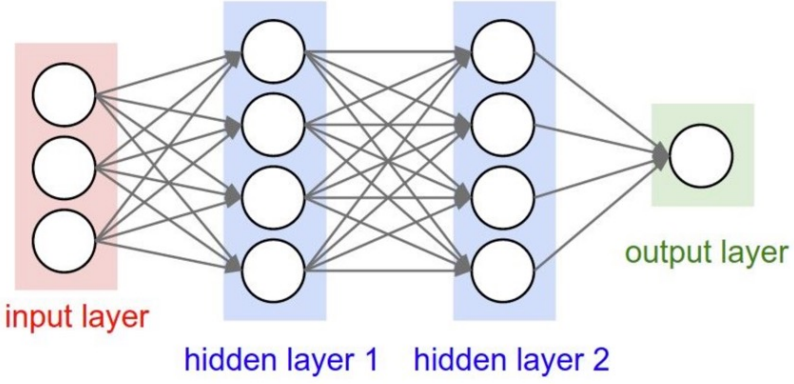
- Activations are passed only from one layer to the next.



Slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

Multilayer Perceptron

- Each layer may have different number of nodes and different activation functions:
  - Commonly, same activation function within one layer
  - Typically,
    - sigmoid/tanh/Relu… activation function is used in the hidden units
    - sigmoid/tanh/Relu… or linear activation functions are used in the output units depending on the problem (classification or function approximation)
- In feedforward networks, outputs are passed only from one layer to the next.



Backpropagation

Capabilities of multilayer NNs were known, but ...

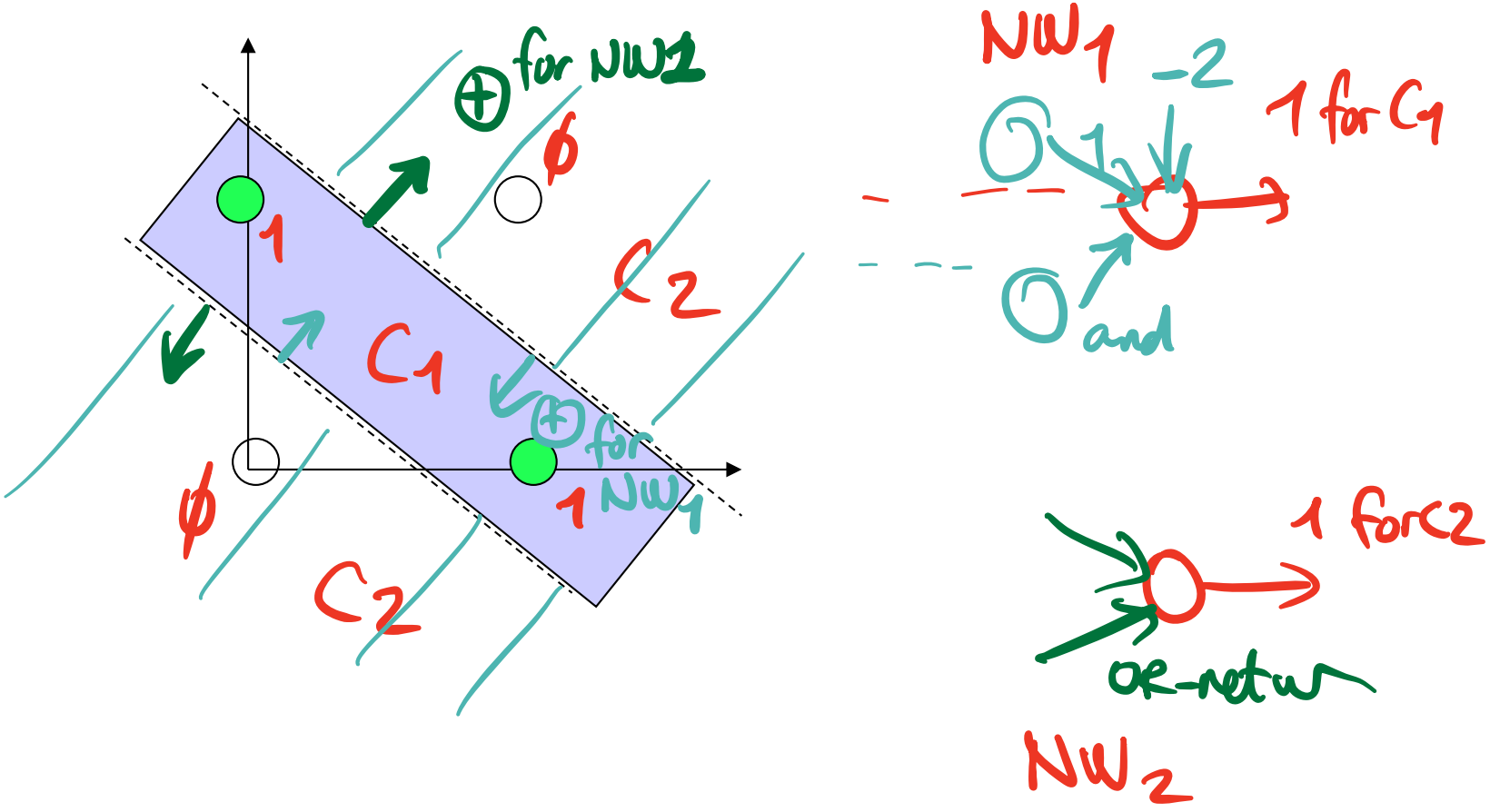
- a learning algorithm was introduced by Werbos (1974)
- made famous by Rumelhart and McClelland (mid 1980s)
- which started massive research in the area

➢ Backpropagation (which is based on gradient descent)

How does MLP solve the XOR problem?

- Learning Boolean functions: 1/0 output can be seen as a 2-class classification problem

- Xor can be solved by a 1-hidden layer network



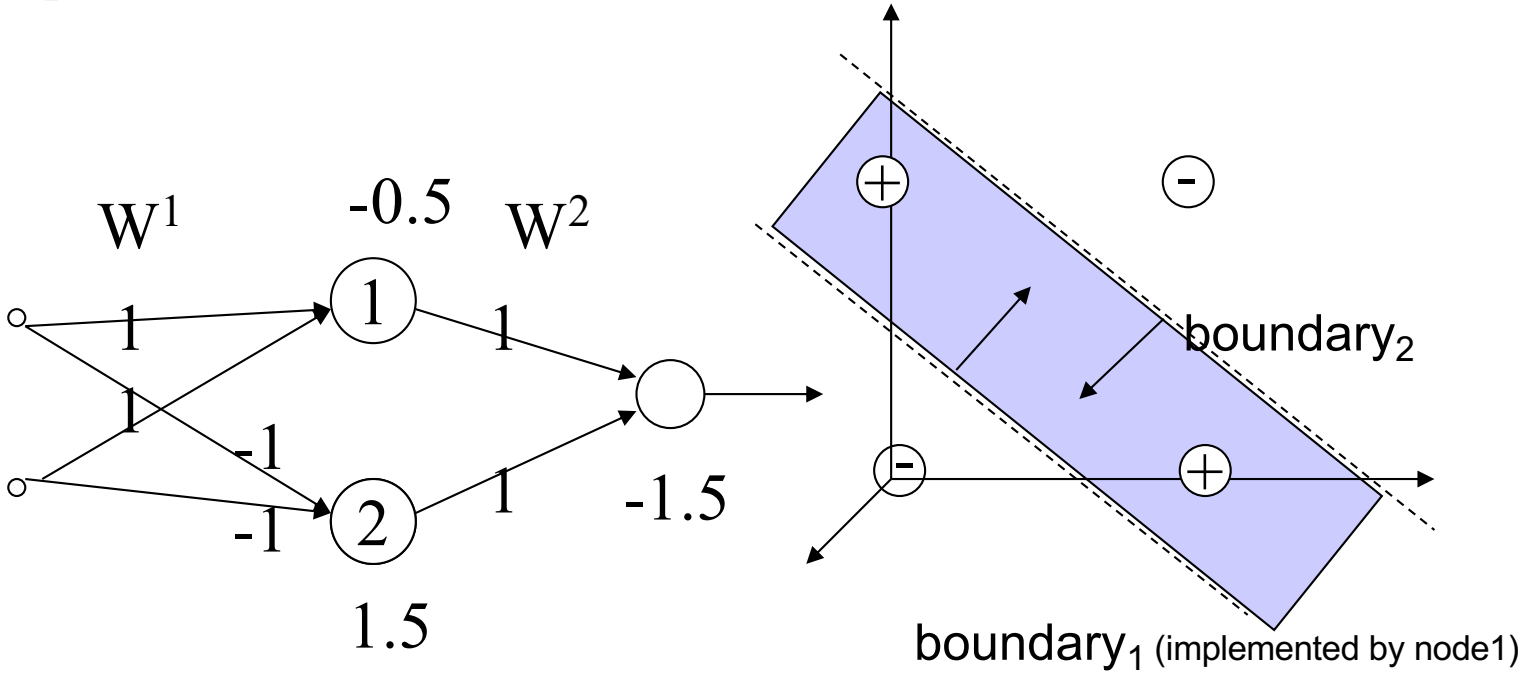
XOR problem Solved!

$W^1_1 = [\ 1\ \ 1\ -0.5]$

$W^1_2 = [-1\ -1\ 1.5]$

$W^2 = [\ 1\ \ 1\ -1.5]$

Notice how each node implements a decision boundary and the output node combines (AND) their result.



Capabilities (Hardlimiting nodes)

Single layer

- Hyperplane boundaries

1-hidden layer

- Can form any, possibly unbounded convex region

2-hidden layers

- Arbitrarily complex decision regions

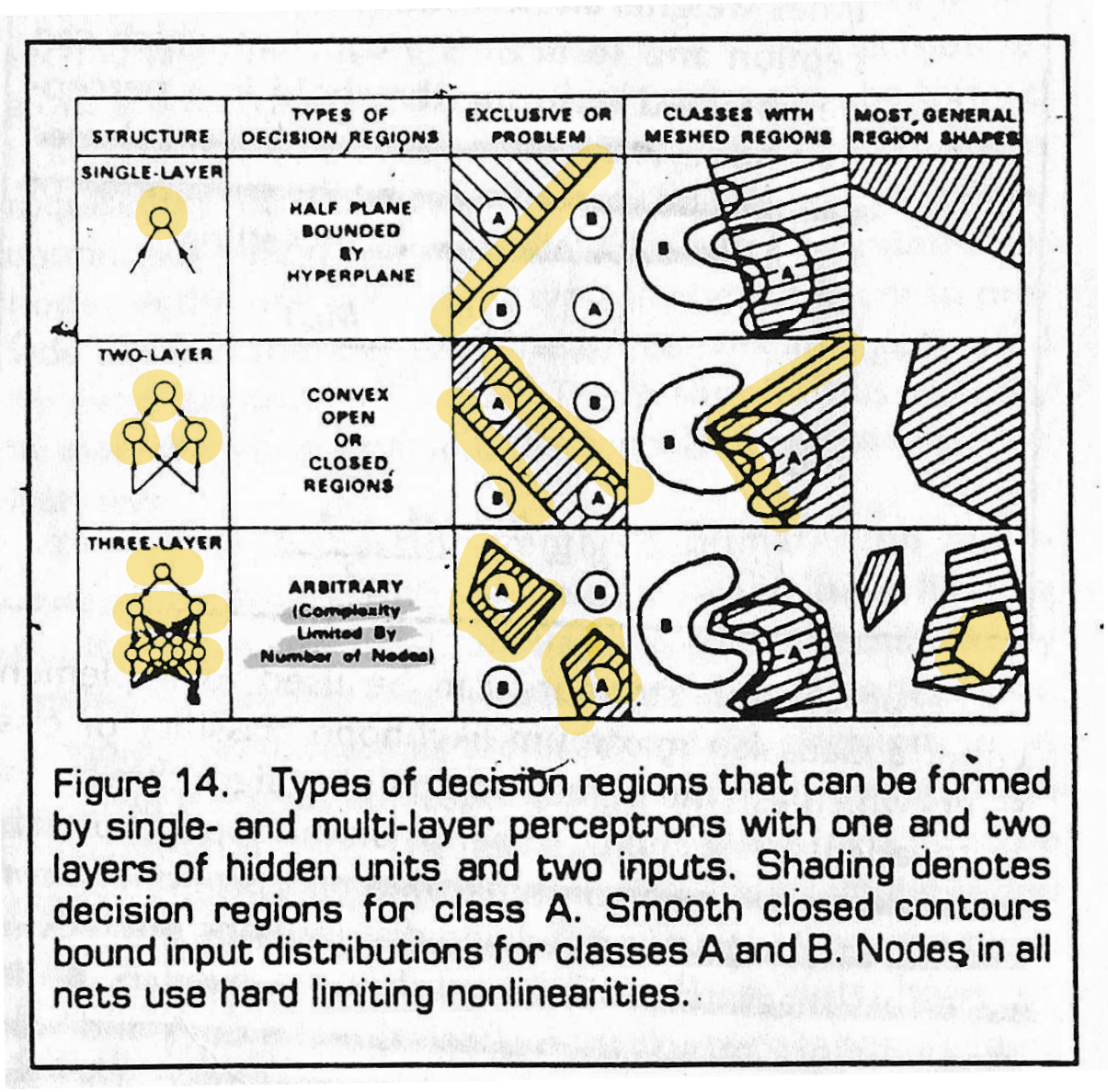


Figure 14. Types of decision regions that can be formed by single- and multi-layer perceptrons with one and two layers of hidden units and two inputs. Shading denotes decision regions for class A. Smooth closed contours bound input distributions for classes A and B. Nodes in all nets use hard limiting nonlinearities.

From Lipmann

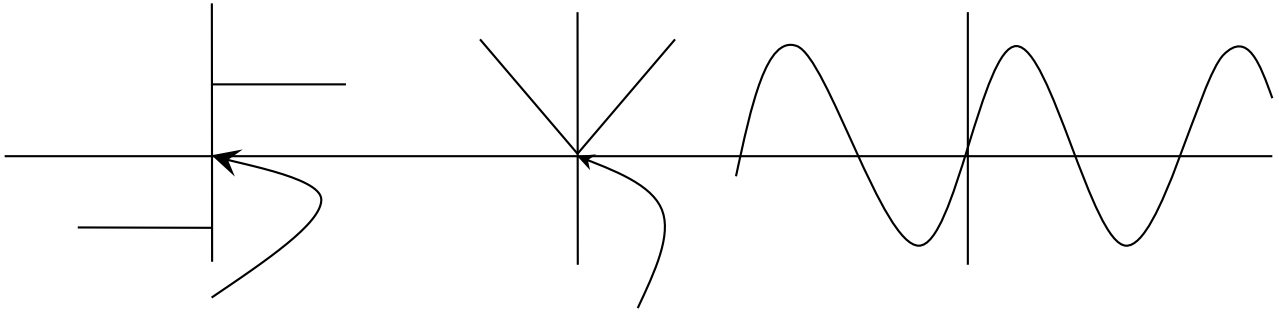
Capabilities

Theorem.(Cybenko 1989, Hornik et al. 1989):

Every bounded continuous function can be approximated arbitrarily accurately by 2 layers of weights (1-hidden layer) and sigmoidal units. All other functions can be learned by 2-hidden layer networks.

- *Proof is based on Kolmogorov's Thm.*
- Discontinuities can be theoretically tolerated for most real life problems.

- **However the fact that such approximation is possible with 2 hidden layers, does not mean it can be learned well in practice.**
- **We now know that deep networks surpass shallow networks clearly.**

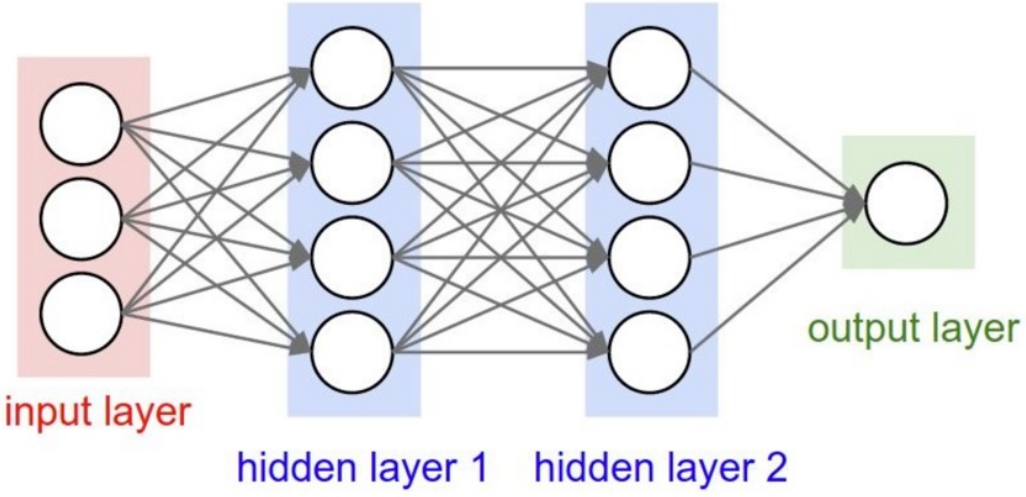


Discontinuity Continuous with discontinuous first derivative

DECIDING ON A NETWORK TOPOLOGY AND LOSS FUNCTION

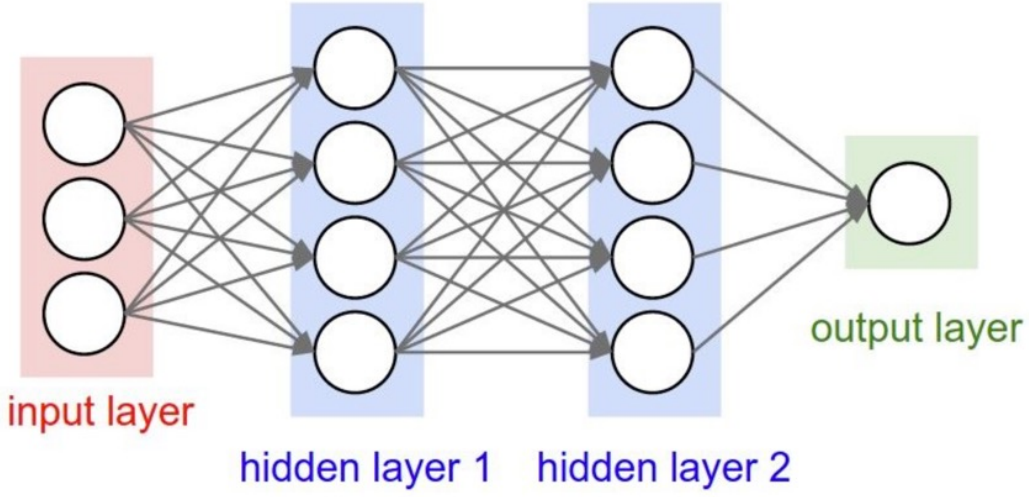
How to Decide on a Network Topology?

- # of input nodes?
  - ……
- # of output nodes?
  - ……
- Activation function?
  - ……
- # of hidden nodes?
  - ……



How to Decide on a Network Topology?

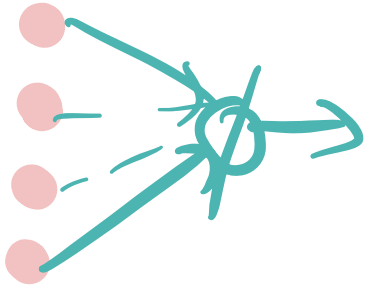
- # of input nodes?
  - Number of features
- # of output nodes?
  - one for regression, 1-of-C for classification…
- transfer function?
  - sigmoid/tanh/ReLU in hidden nodes
  - inear for regression, softmax for classification
- # of hidden nodes?
  - Not exactly known



Network Topology and Loss Function for Regression

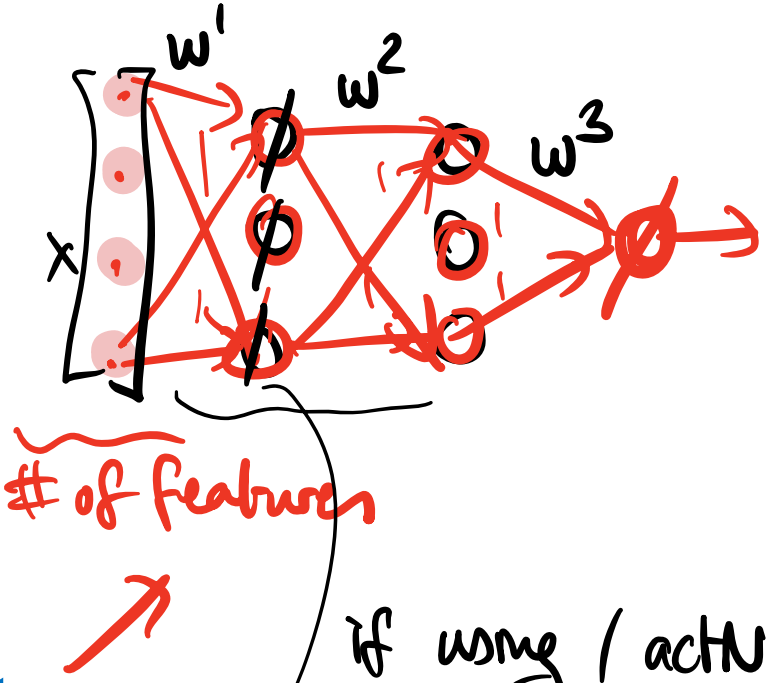
Linear Regression

- Single layer network and a single output node w/ linear activation & MSE loss



Non-Linear Regression

- Multilayer network and a single output node w/ linear activation and sigmoid activation in the hidden layers & MSE loss



if using / actN. f()

"$f(w'x) = w'x$ iff is linear

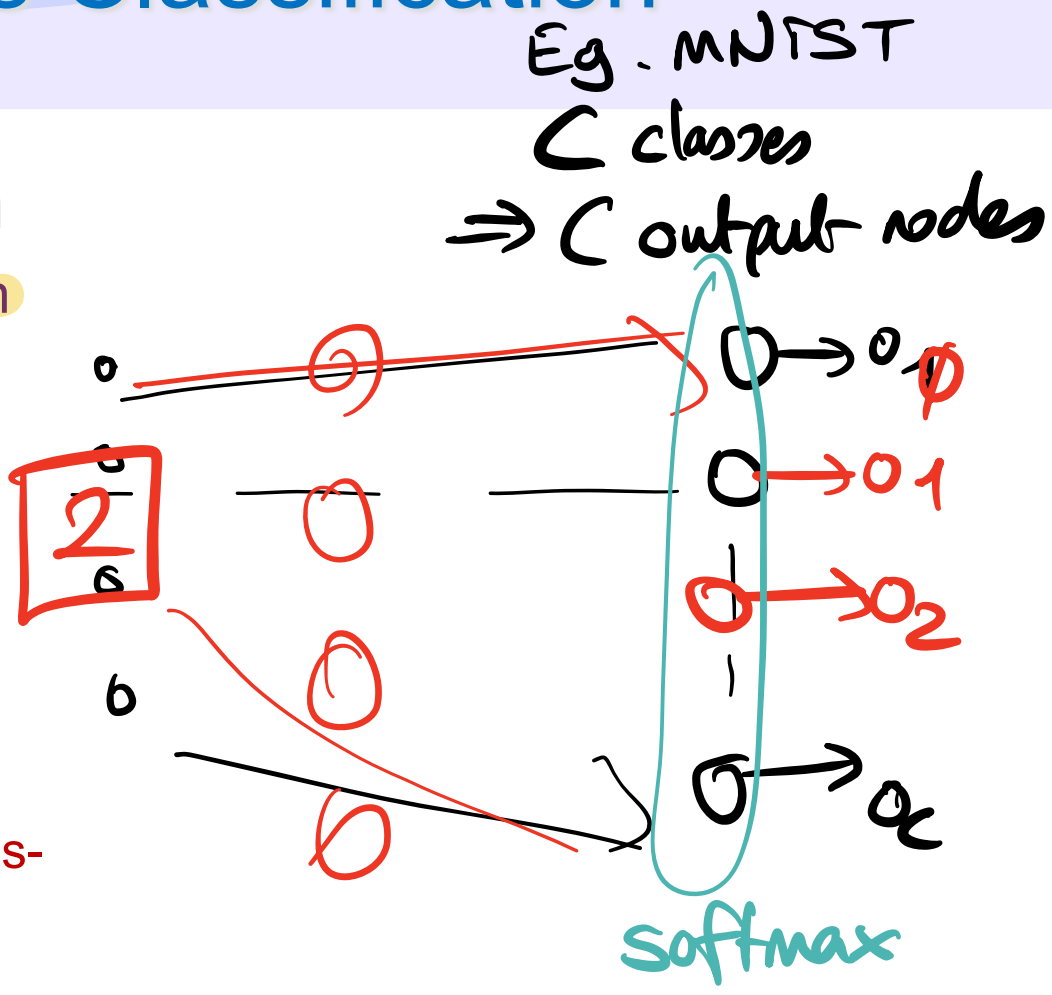
$W^2(W'x)$: one layer too many

$W_{new} = (W^2W')$ to combine 2 layers

Multi-class Classification

Multi-class classification

- K-nodes w/ softmax activation and cross-entropy loss
  - K >= 2



- For two-class problems, you may also use a single output node with sigmoid activation and binary cross-entropy loss (BCE).

These two are **equivalent**, but the first is more general and sometimes required. So you may just adopt that one.

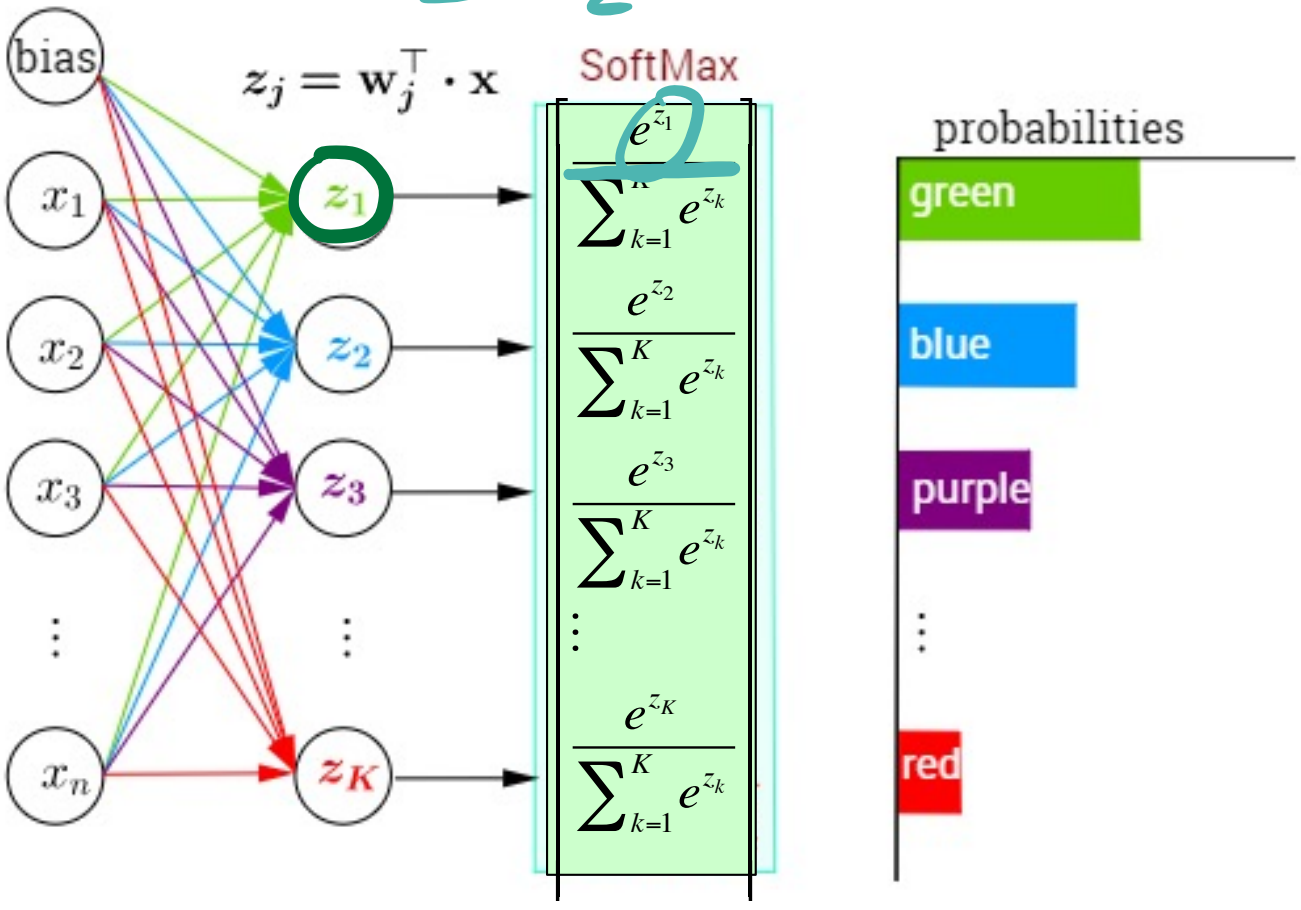
SoftMax

- Given a test input **x**, we want our hypothesis to estimate the probability that P(y=k|**x**) for each value of k=1,…,K.
- I.e., we want to interpret the outputs as the posterior probabilities of the class given x where the weights are $\mathbf{w}_i$.

$z_1 = w_1 \cdot x$

$z_2 = w_2 \cdot x$

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \mathbf{w}_3^\top \\ \vdots \\ \mathbf{w}_K^\top \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

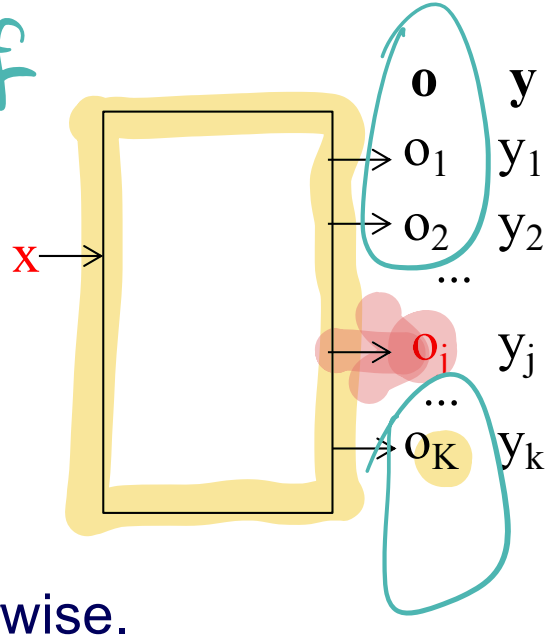


Equivalent Formulation for Categorical Cross-Entropy

For N training samples and K classes, the cross-entropy loss is:

$$J = -\sum_{j=1}^{K} 1\{y_j = 1\} \log o_j$$

where **y** is the target and **o** is the output for input **x**.



Note: Indicator function: 1{...} = 1 if {…} is true; 0 otherwise.

Categorical cross entropy is also (wrongly) called softmax loss sometimes, as it is used with softmax activation function.

Categorical Cross-Entropy: Loss vs Cost

Categorical cross-entropy **loss** for input **x** with target *vector* **y** in a K>2 class problem, is:

$$L = -\mathbf{y}.\log(\hat{\mathbf{y}})$$

(. indicates dot product)

$$L = -\sum_{i=1}^{nout} y_i \log(\hat{y}_i)$$

Equivalent notations

The corresponding **cost** is:

$$J = -\frac{1}{N}\sum_{i=1}^{N} \mathbf{y}.\log(\hat{\mathbf{y}})$$

$$J = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{nout} y_{ij} \log(\hat{y}_{ij})$$

Equivalent notations

true label = [1 0 0 0 0]

predicted = [0.1 0.5 0.1 0.1 0.2]

Cross-entropy loss = ?

true label = [1 0 0 0 0]

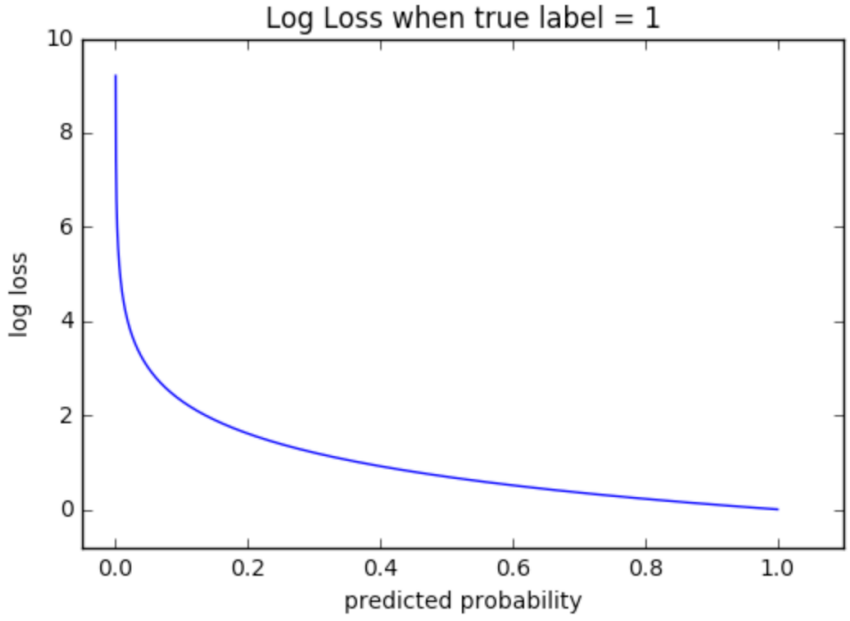
predicted = [0.1 0.5 0.1 0.1 0.2]

Cross-entropy loss = -ln(0.1) = 2.3

Binary Cross Entropy

Target y is set as 1 for one class and 0 for the other

- Prediction p is the posterior probability for the 1 class



Binary Cross Entropy Cost (over N samples)

with target y_i and output p_i

$$C = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log p_i + (1-y_i)\log(1-p_i)]$$

where N is the number of samples,
yi is the target for the ith sample ($y_i \in \{0,1\}$)
p_i is the probability of target label 1 assigned by the model for instance i.

Output Layer and Loss Function for 2-Class Classification

We had suggested to use:

- Single node w/ sigmoid activation & binary cross-entropy loss

```
keras.layers.Dense(1, activation = 'sigmoid')
```

Equivalently you may see that people use (but extra unnecessary computation):

- 2-nodes w/ softmax activation and categorical cross-entropy loss (general case of binary cross-entropy)

```
keras.layers.Dense(2, activation = 'softmax')
```

Multi-Task Learning

Multi-task Learning is different/

- L-nodes with sigmoid activation for L labels and MSE loss over the output nodes