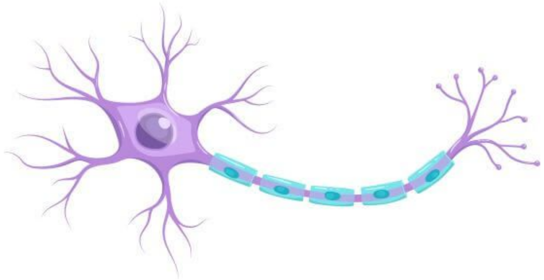
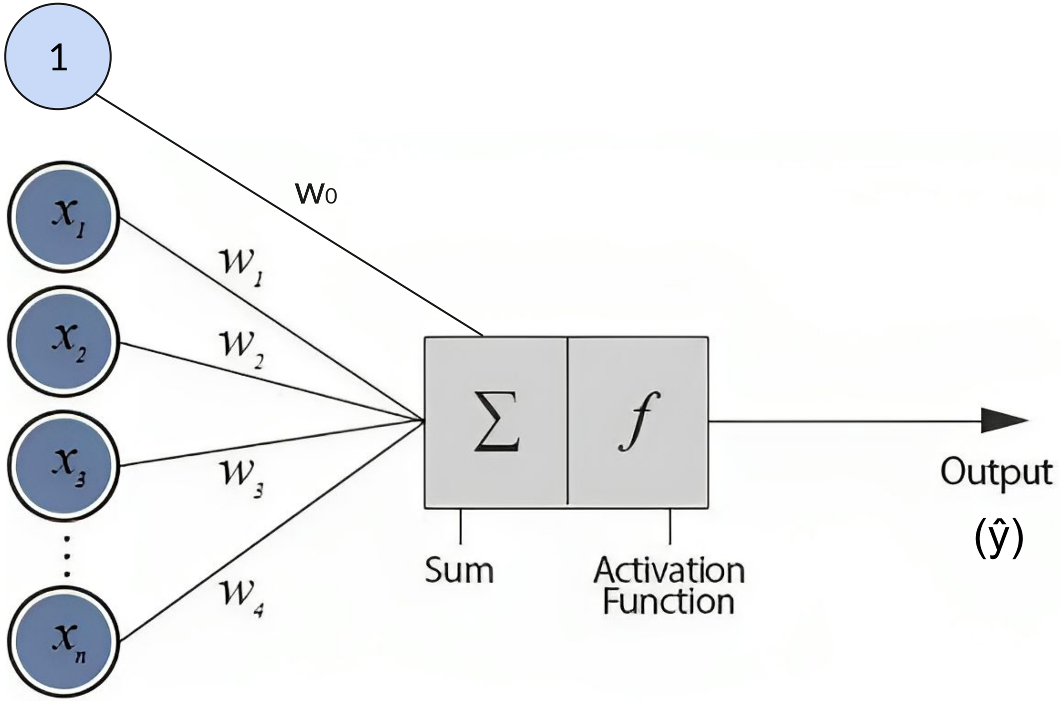
Neural Networks

NN – Intro

- ML method used for regression, classification, etc.
- Inspired by the human brain.
- Very flexible.
- Not easily interpretable.
- Very famous nowadays because of its good performance on:
    - Images.
    - Text.
    - Audio.
    - …

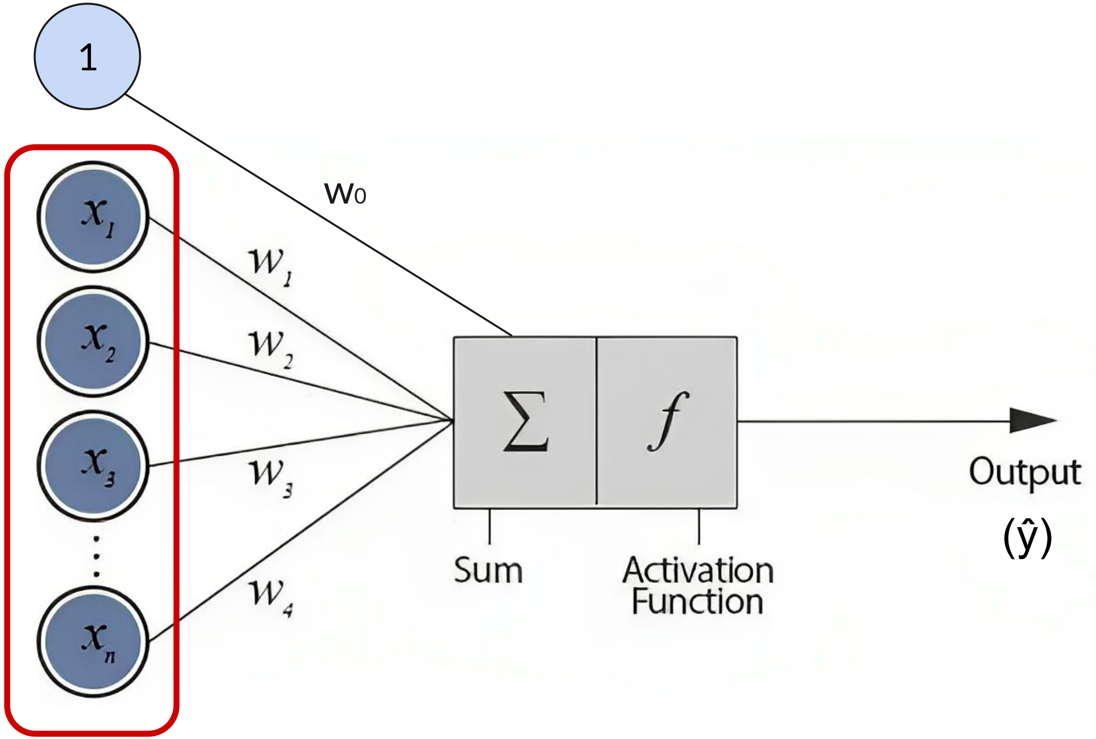


1 Neuron

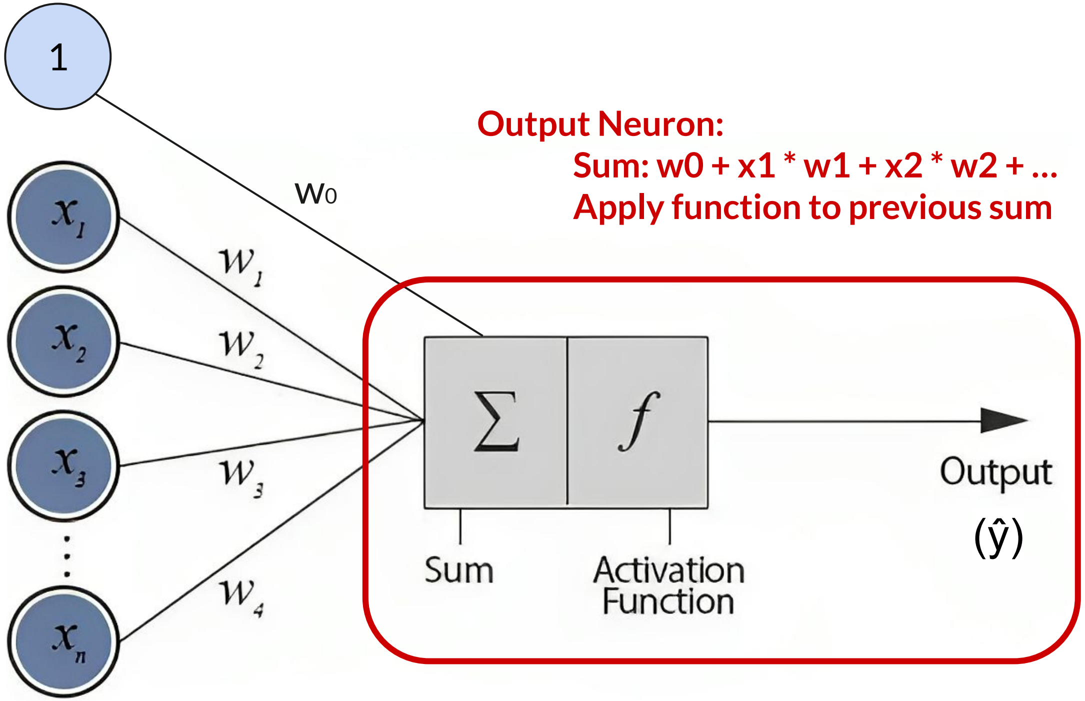

1
w_0
x_1
x_2
x_3
x_n
w_1
w_2
w_3
w_4
Σ
f
Sum
Activation Function
Output
($\hat{y}$)

1 Neuron

Inputs = Features

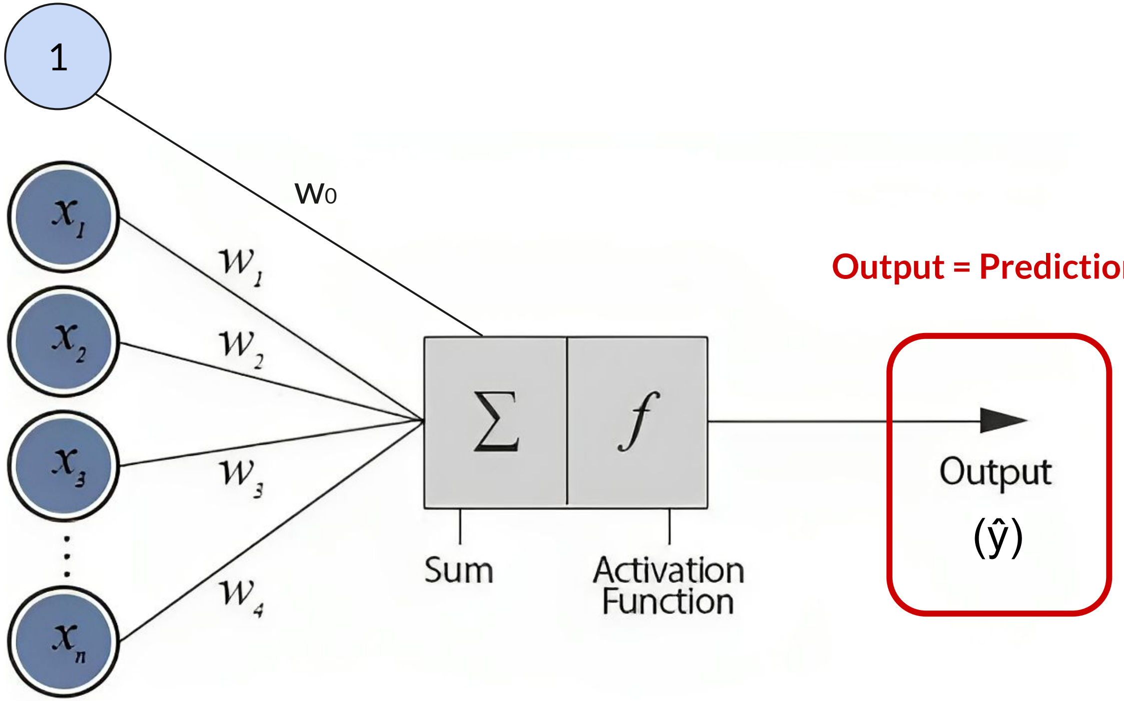


1 Neuron



1
Wo
x_1
x_2
x_3
x_n
w_1
w_2
w_3
w_4
Σ
f
Output Neuron:
Sum: w0 + x1 * w1 + x2 * w2 + ...
Apply function to previous sum
Sum
Activation Function
Output
(ŷ)

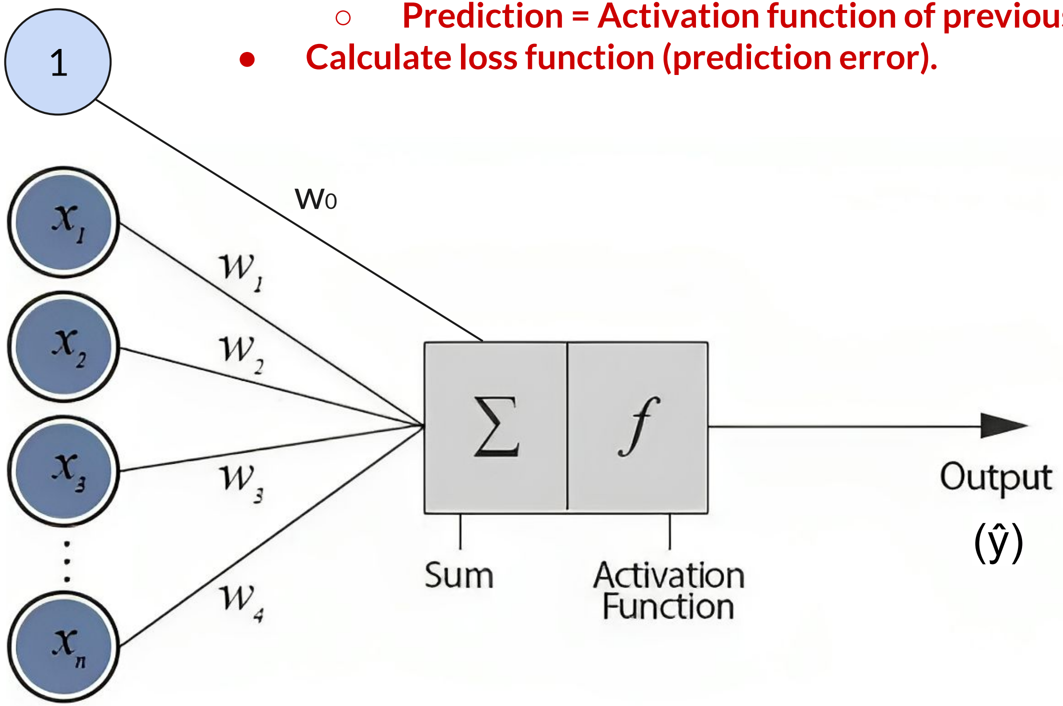
1 Neuron


1
Wo
w_1
w_2
w_3
w_4
x_1
x_2
x_3
x_n
Σ
f
Sum
Activation Function
Output = Prediction
Output
(ŷ)

1 Neuron

Forward propagation:

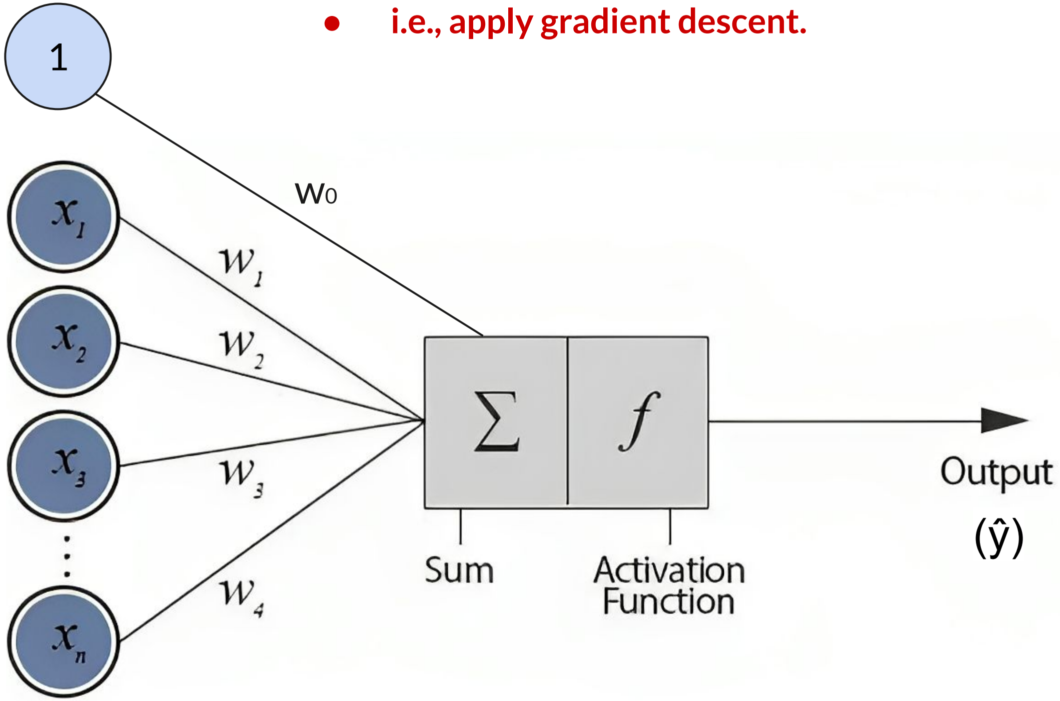
- For each input row:
  - Multiply features with weights and sum.
  - Prediction = Activation function of previous sum.
- Calculate loss function (prediction error).



1 Neuron

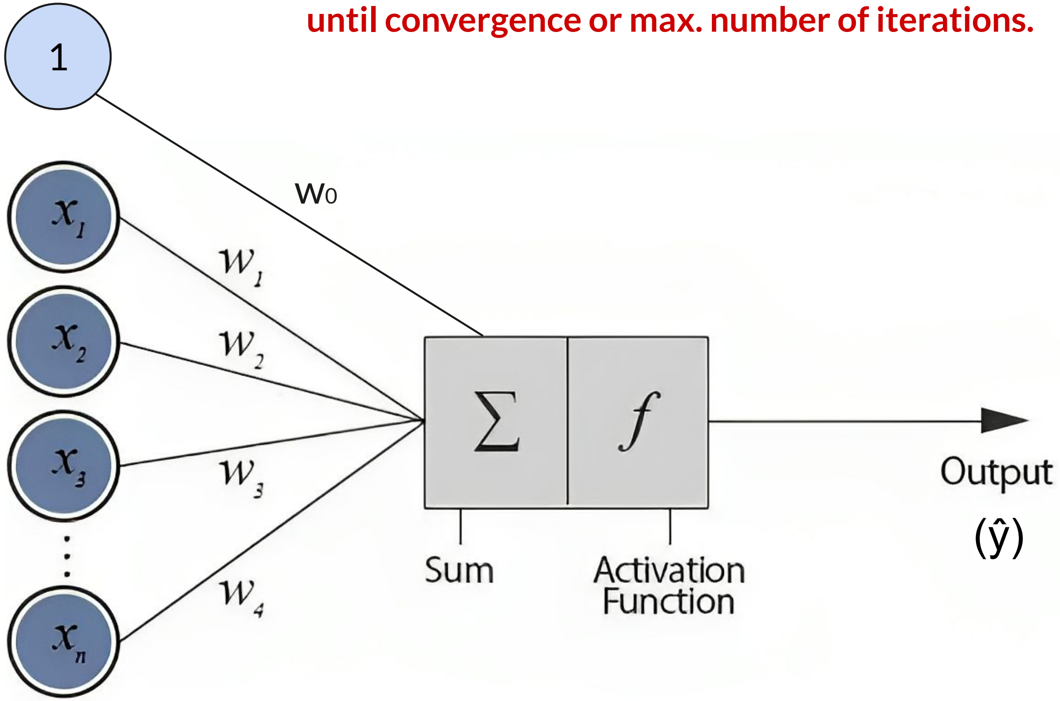
Backward propagation:

- **Update each weight for optimizing loss function.**
- **i.e., apply gradient descent.**



1 Neuron

Keep applying forward and backward propagation until convergence or max. number of iterations.

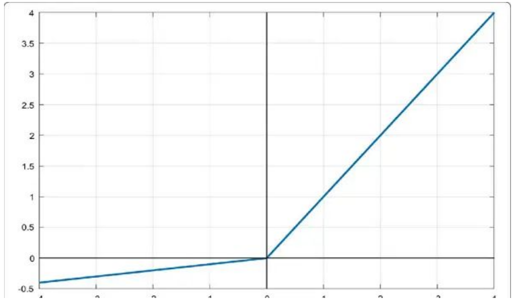


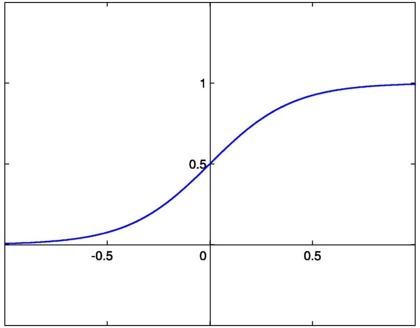
NN – Activation Function

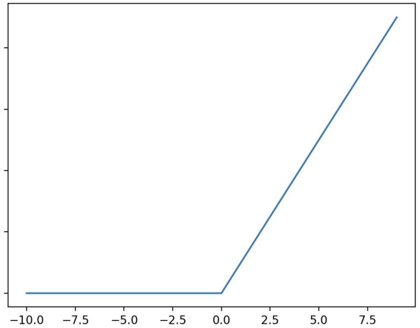
- Aren't we just doing the same as with LR? Yes, kind of.
- Activation functions are used for learning non-linearities between features and target.
- By applying a non-linear function to a linear combination of weights and features.

NN – Common Activation Functions

- Linear
- Sigmoid
- Tanh
- ReLU
- Leaky ReLU
- ELU
- Softmax





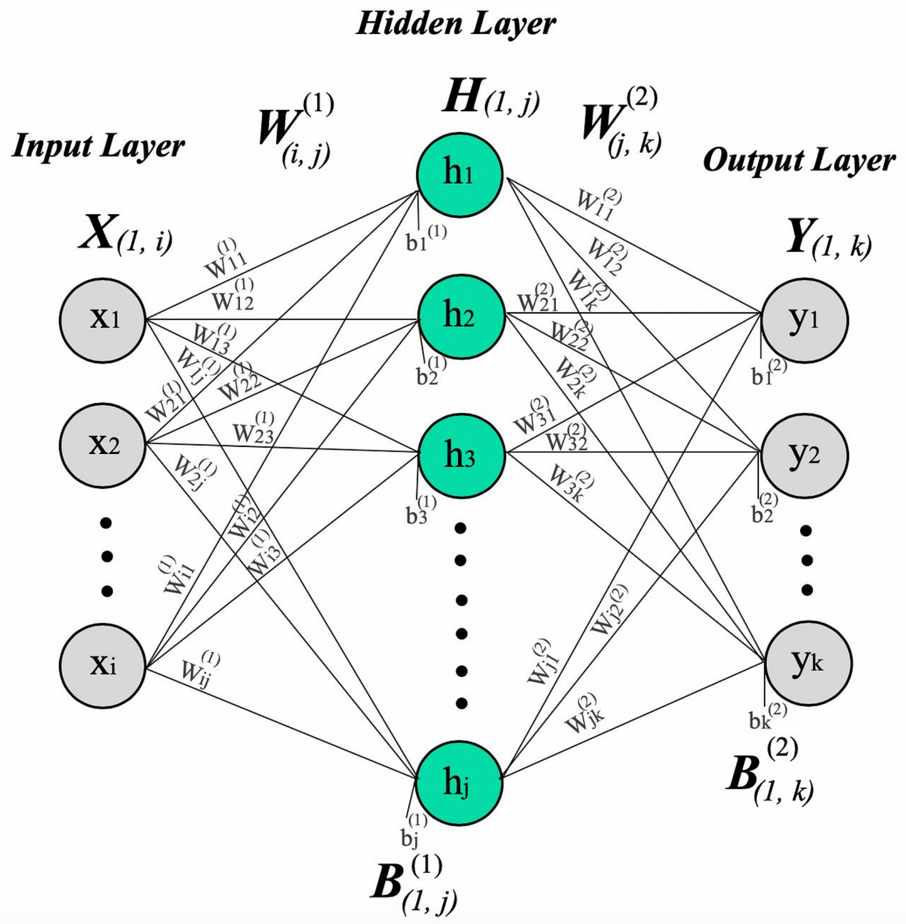


1 Neuron

- Cannot learn non-linear interactions between features.
- Cannot learn complex non-linearities between features and targets.
- This can be solved by using a hidden layer with more neurons.

NN – 3 Layers

- Input
- Hidden
- Output

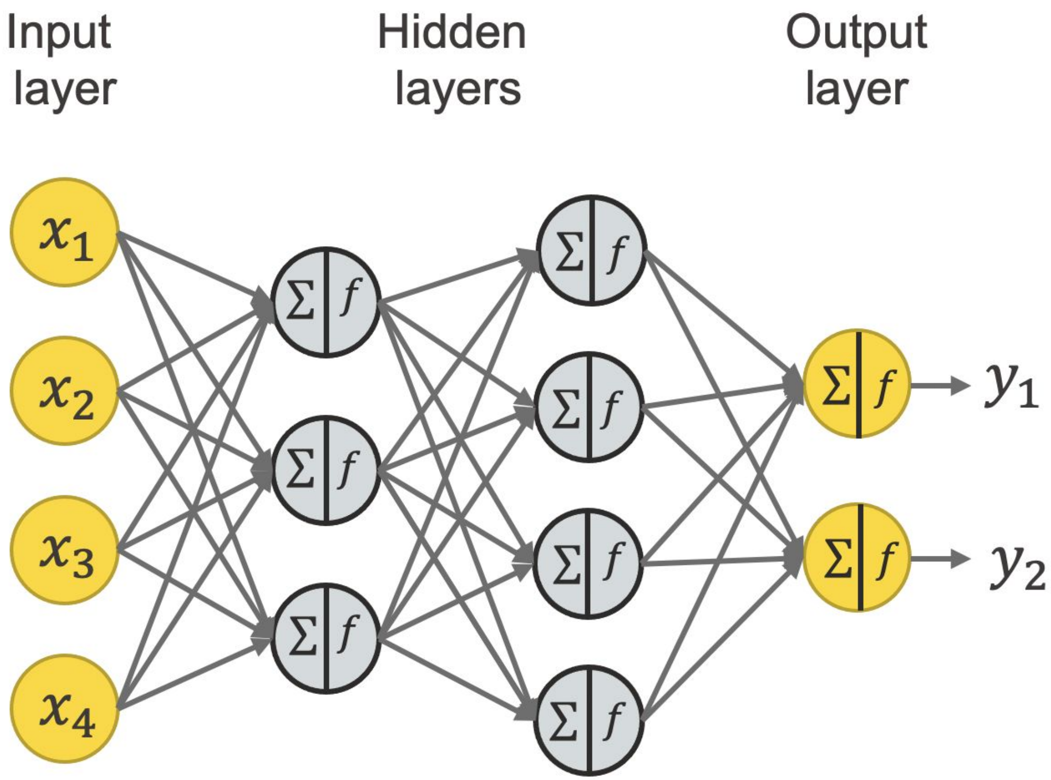


NN – 3 Layer Lengths

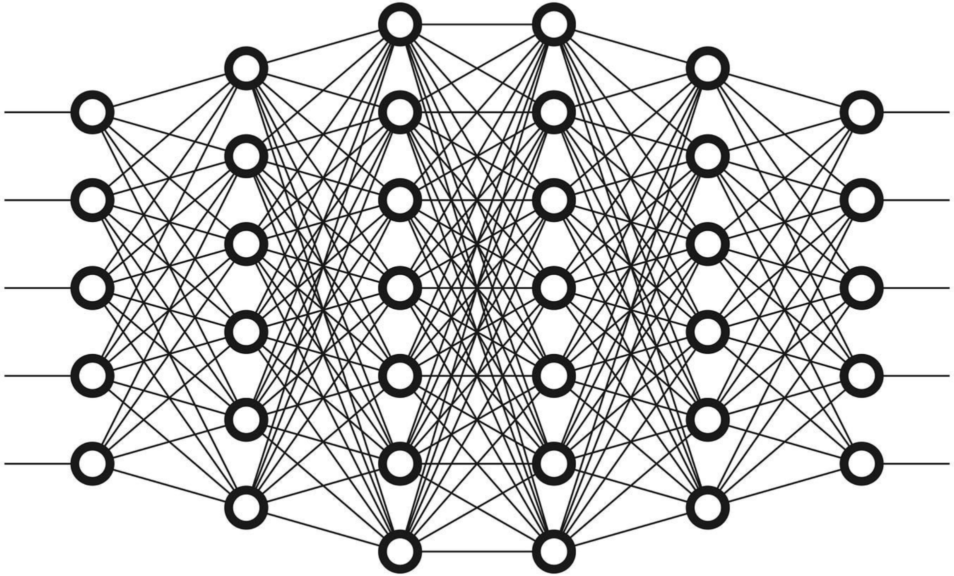
- Input: same length as number of features.
- Hidden: any length we want (see later recommendations).
- Output:
  - 1 (typically linear) neuron for regression problems.
  - N neurons for classification problems with N classes.
    - Softmax if single-label.
    - Sigmoid if multi-label.
  - For binary classification, 1 sigmoid neuron suffices.

NN – 4 Layers

- Input
- Hidden
- Hidden
- Output



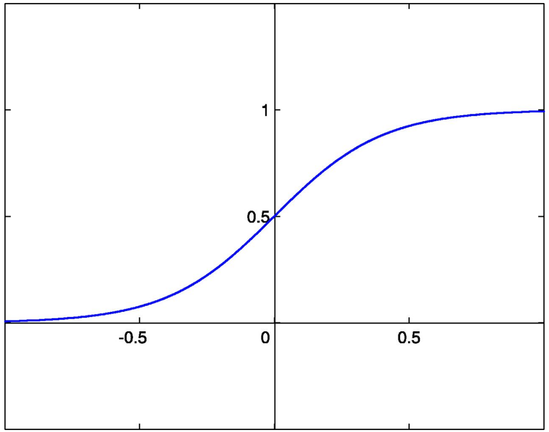
Deep NN (DNN)



NN – Common Activation Functions

- Linear
- Sigmoid
- Tanh
- ReLU
- Leaky ReLU
- ELU
- Softmax

Hidden layers

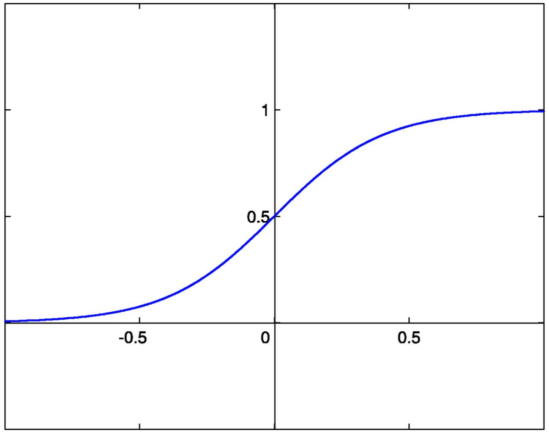


NN – Common Activation Functions

- Linear
- Sigmoid
- Tanh
- ReLU
- Leaky ReLU
- ELU
- Softmax

Output layer

Output layer



NN – Loss

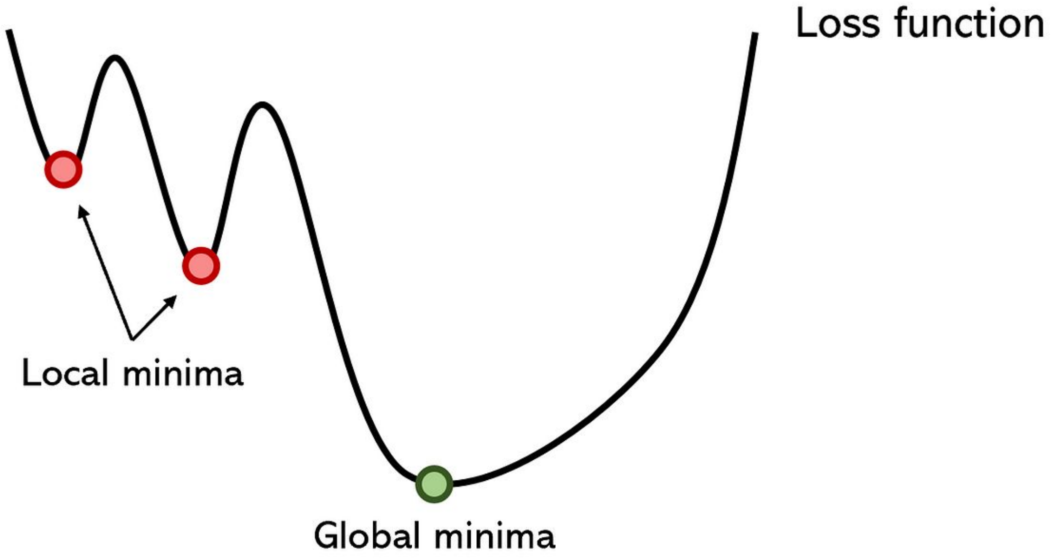
- Regression:
    - MSE
    - MAE
    - Huber loss (robust to outliers).

NN – Loss

- Regression:
  - MSE
  - MAE
  - Huber loss (robust to outliers).
- Classification:
  - Binary cross-entropy (same as log loss).
  - Sparse categorical cross-entropy (integer targets).
  - Categorical cross-entropy (one-hot encoded targets).

NN – Optimizing Loss

- Non-convex.
- Risk of getting stuck.
- Good learning rate.
  - Optimizers:
    - Adam
    - RMSprop
    - AdaGrad
- Good mini-batch for GD.




STOCHASTIC GRADIENT DESCENT
SGD WITH MOMENTUM
RMSPROP
ADAM
imgflip.com

NN – Epochs

- One epoch consists in one pass through all training samples.
- When separated in mini-batches, one epoch iterates over all batches.
- NNs require multiple epochs for a good optimization of the weights.
- Too many epochs can lead to overfitting.
- Too few epochs may result in underfitting.
- Early stopping can be applied.

Building NNs – Recommendations

- Number of layers:
    - One or two hidden for starting.
    - Maybe deeper for more complex data or task.

Building NNs – Recommendations

- Number of layers:
    - One or two hidden for starting.
    - Maybe deeper for more complex data or task.
- Neurons per layer:
    - It is common to start with a length between the input and the output.
    - Or even with more neurons than the input on the first layer.
    - Reducing the hidden sizes as the network goes deeper.

Building NNs – Recommendations

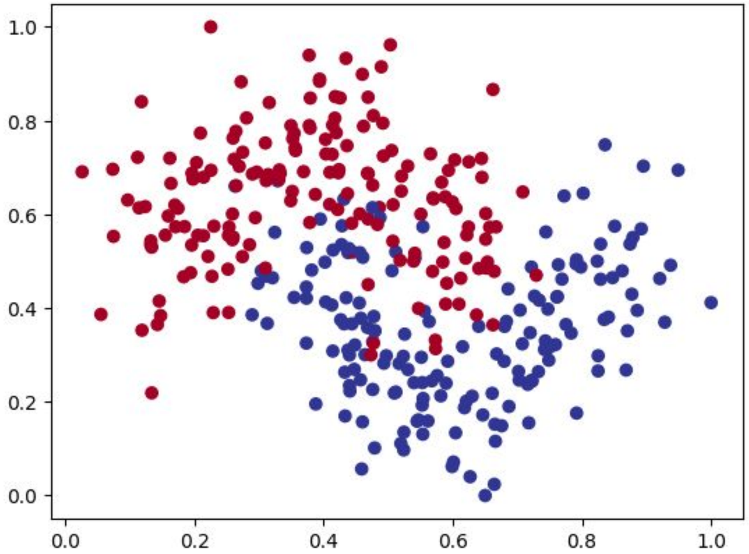
- Number of layers:
  - One or two hidden for starting.
  - Maybe deeper for more complex data or task.
- Neurons per layer:
  - It is common to start with a length between the input and the output.
  - Or even with more neurons than the input on the first layer.
  - Reducing the hidden sizes as the network goes deeper.
- Adam is normally a good start.

Building NNs – Number of Neurons

- More neurons can represent more complex patterns.
- More neurons are slower to train.
- Too many neurons can lead to overfitting.
- Too few neurons can result in underfitting.

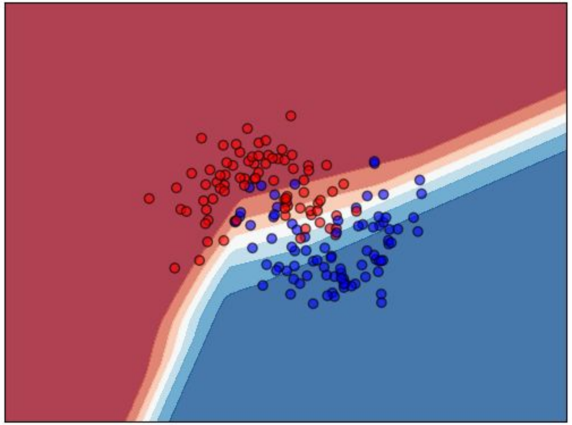
NN – Example

- 4 Layers:
  - Input
  - Hidden 16 neurons
  - Hidden 8 neurons
  - Output 2 neurons with softmax
- Optimizer: Adam(0.01)
- Loss: categorical cross-entropy

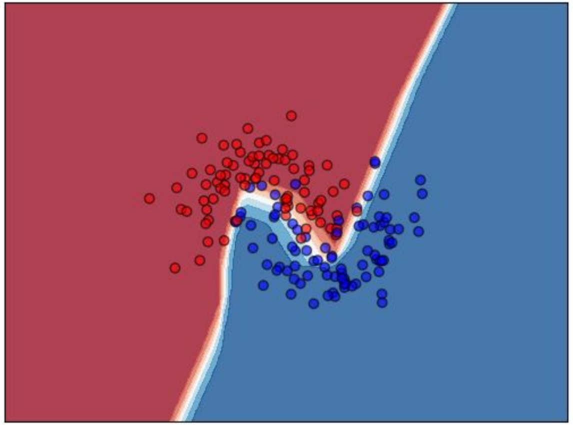


NN – Example

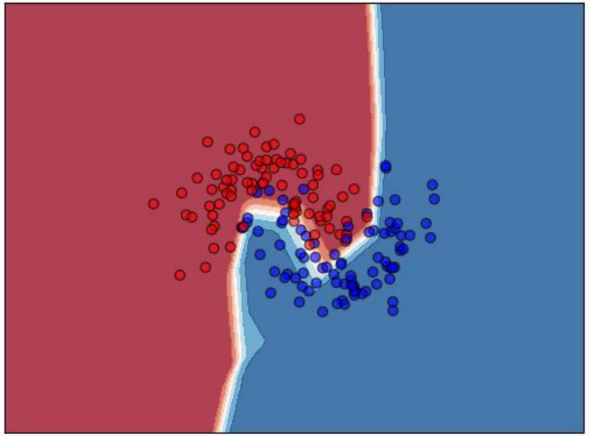
50 epochs



200 epochs



2,000 epochs



NNs – Libraries

- sklearn.neural_network.MLPClassifier or MLPRegressor
- Tensorflow
- Pytorch

Summary of Concepts

- Neural Network (NN).
- Deep NN (DNN).
- Weights (similar as LR coefficients).
- Activation functions.
- Input, Hidden, Output Layers.
- Forward/Backward propagation.
- Learning rate optimizers: Adam, RMSprop, AdaGrad.
- Epoch.