ECG782: MULTIDIMENSIONAL DIGITAL SIGNAL PROCESSING

INTRO TO ARTIFICIAL NEURAL NETWORKS

Géron Chapter 10

http://www.ee.unlv.edu/~b1morris/ecg782

OUTLINE

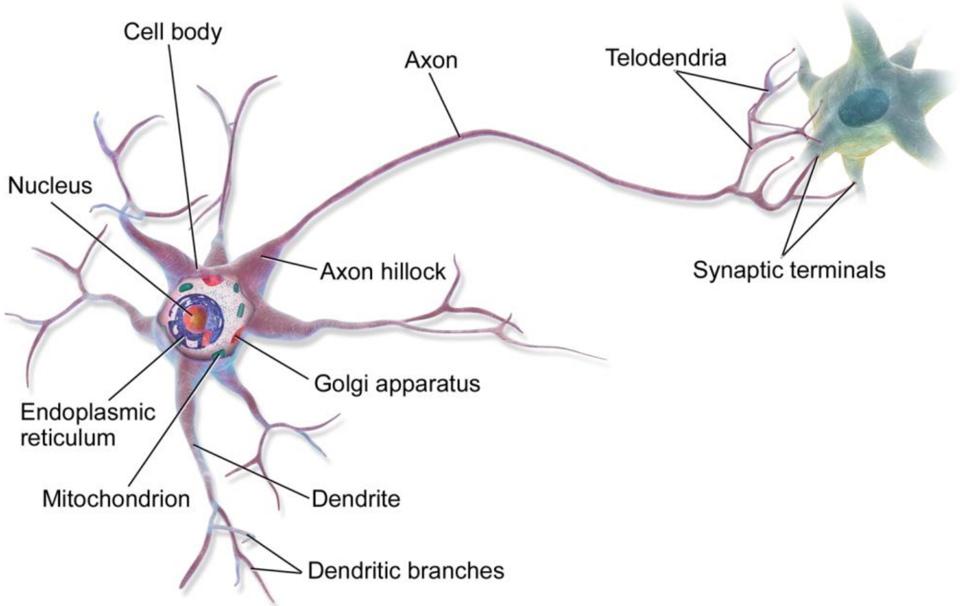
- Biological Inspiration
- Logic Computation with Neurons
- The Perceptron
- The Multilayer Perceptron and Backpropagation
- Regression MLPs
- Classification MLPs

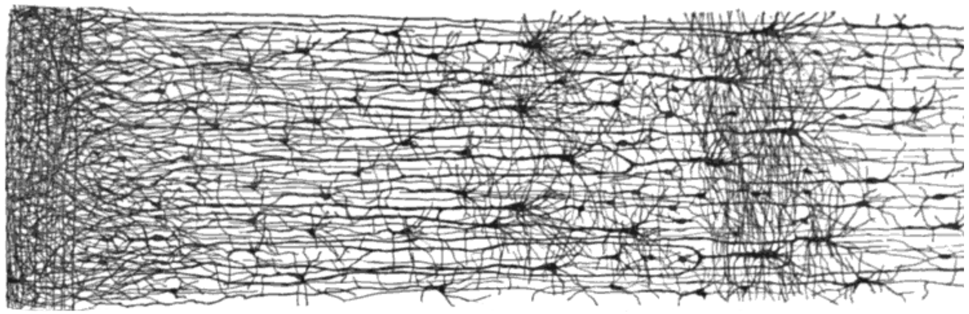
FROM BIOLOGICAL TO ARTIFICIAL NEURONS

- Artificial neural networks (ANNs) first introduced in 1943
- Excitement with ANNs waned in the 1960s
- 1980s had renewed interest but was overtaken in the 1990s with ML techniques such as SVM
- Since 2010s major renewed interest
  - Huge quantities of data are available to train networks
  - Major computing power increases for reduced training times (GPU and cloud)
  - Improved training algorithms
  - Local optima issue rare
  - Lots of funding in ANNs (Artificial Intelligence/Deep Learning)

BIOLOGICAL NEURONS

- Cell mostly found in animal brains
- Produce short electrical impulses (action potentials, APs, or signals) to make synapses release chemical signals (neurotransmitters)
- When a neuron receives enough neurotransmitters it fires its own electrical pulses
- Individual neurons are simple but arranged into vast networks of billions
  - Each neuron connected to thousands of other neurons
  - Neurons seem to be organized in consecutive layers

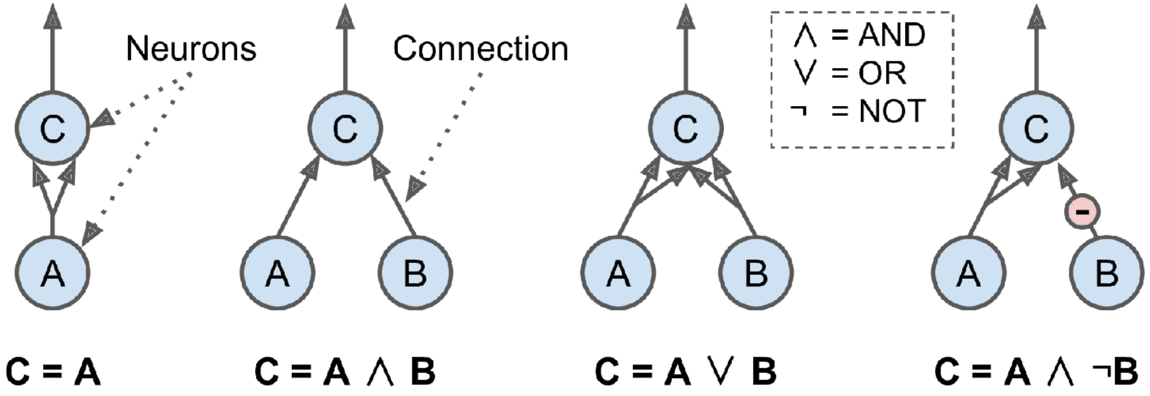




LOGICAL COMPUTATIONS WITH NEURONS

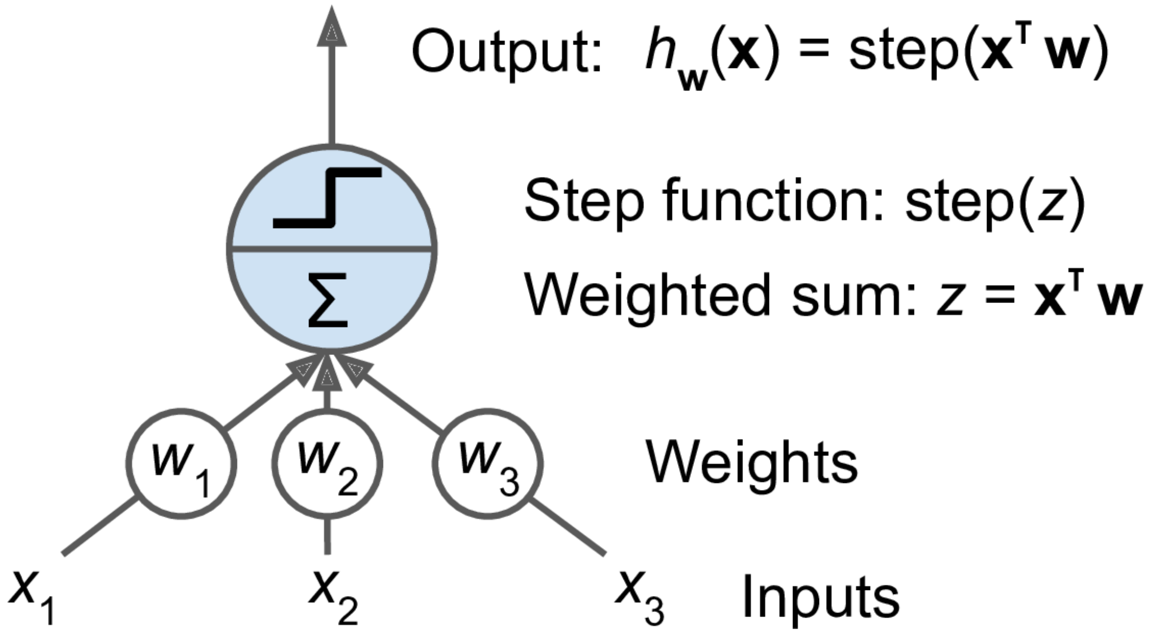
- Artificial neuron proposed by McCulloch and Pitts
  - Simple binary inputs and one binary output
  - Activates output when certain number of inputs on/active
- Even with the simple model, any logical proposition can be computed
- Basic building block networks can be combined for more complex logical expressions

- Building block networks
  - Implement basic logic functions



THE PERCEPTRON I (TLU)

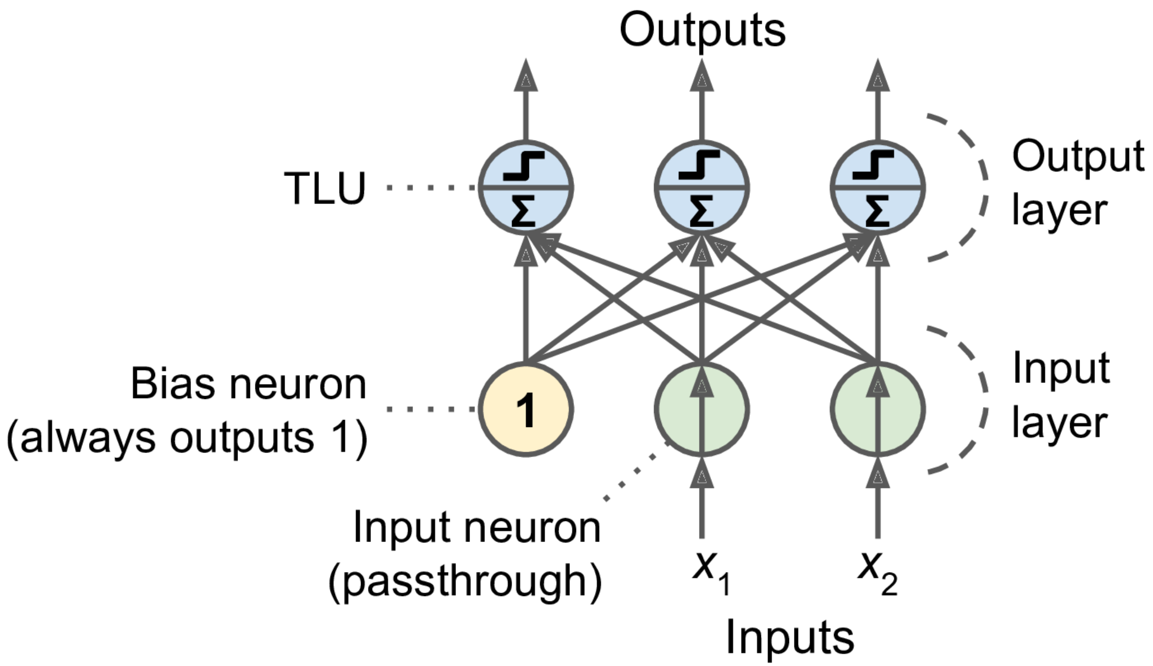
- Invented by Frank Rosenblatt in 1957
- Inputs/outputs are numbers (instead of binary)
- Based on threshold logic unit (TLU) or linear threshold unit
- Inputs associated with a weight
- TLU computes weighted sum of input
  - $z = w_1 x_1 + w_2 x_2 + w_3 x_3$
- Output after a step (threshold) function
  - Heavyside of sign function
- TLU can be used as a simple linear binary classifier



THE PERCEPTRON II

- Perceptron is a layer for TLU
  - Fully connected (dense) layer – all inputs connected to all neurons
- Input neuron – pass value through unchanged
- Bias neuron – always outputs 1

- Example: Multilabel classifier
  - 2 inputs 3 outputs
  - Can classify into three binary classes based on two input values



THE PERCEPTRON III

- Output of fully connected layer

$$h_{W,b}(X) = \phi(XW + b)$$

  - X – matrix of input features
  - W – weight matrix (all weights between input and neurons)
    - One row per input neuron
    - One column per neuron layer
  - b – bias (weights) vector
  - ϕ – activation function (e.g. step)

- Produces linear (non-complex) decision boundary

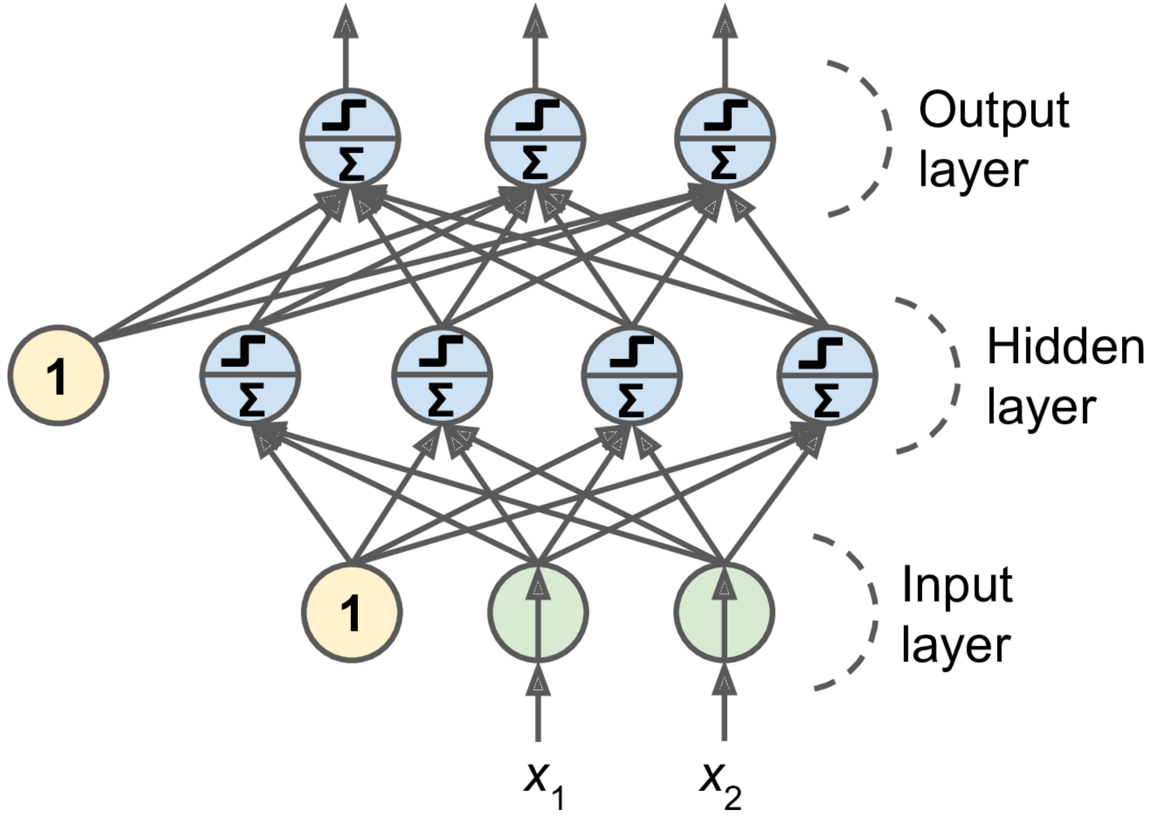
- Perceptron training – reinforce connections that reduce prediction error

$$w_{i,j}^{(next\ step)} = w_{i,j} + \eta\left(y_j - \hat{y}_j\right)x_i$$

  - $w_{i,j}$ - connection weight between ith input and jth output neuron
  - x_i - ith input value
  - $\hat{y}_j$ - perceptron output of jth neuron
  - y_j - target (ground truth) output of jth neuron
  - η – learning rate

MULTILAYER PERCEPTRON (MLP)

- Stack TLU layers for more complicated functions
  - Input layer - passthrough
  - Hidden layer – intermediate TLU layer
  - Output layer – final fully connected TLU layer
- Lower layers – closer to input
- Upper layers – closer to output
- Deep neural network (DNN) has many hidden layers



BACKPROPAGATION I

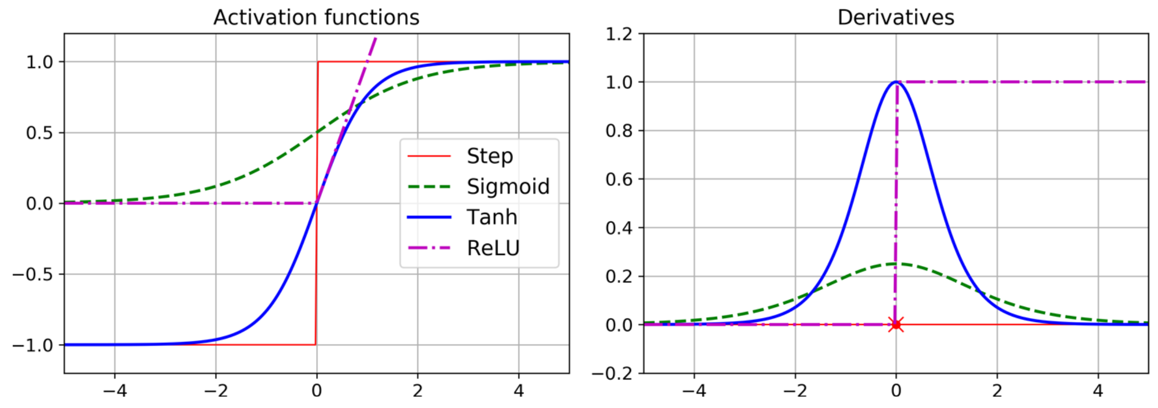
- Effective method to train a MLP developed in 1986
  - Gradient Descent method with efficient gradient computation technique
  - Single forward-backward pass through network to compute gradient of network error for all model parameters
    - Can update all connection weights and bias terms
- Backpropagation uses reverse-mode autodiff to automatically compute gradients (Appendix D)

BACKPROPAGATION II

- Process full dataset each epoch
  - Use mini-batch at each iteration – larger more efficient and more stable gradient but requires more memory
- Mini-batch of input is sent through the MLP in a forward pass (from input to output prediction)
  - All intermediate results (from hidden layers) are saved for backward pass
- Measure current network prediction error
  - Use of loss function to define error metric
- Compute contribution of each connection to the total error
  - Performed backward from output through hidden layers back to input using the chain rule
- Perform Gradient Descent step to adjust all connection weights
  - Using the error gradients from the backward pass

ACTIVATION FUNCTIONS

- Cannot use step for activation since it has no gradient information
- Sigmoid (logistic) function
  - $\sigma(z) = 1/(1 + \exp(-z))$
  - S-shaped between [0, 1]
- Hyperbolic tangent function
  - $\tanh(z) = 2\sigma(2z) - 1$
  - Output between [-1,1] helps speed convergence
- Rectified Linear Unit function
  - $ReLU(z) = \max(0, z)$
  - Not differentiable, but works well and fast so popular



Activation functions add non-linearity!

REGRESSION MLPs

- Single output neuron
  - Mulivariate regression requires an output neuron for each output dimension
    - 2: (x,y) for center of object
    - 4: (x,y,h,w) for a bounding box around object
- Output activation
  - No activation – no limits on output range of value
  - ReLU or softplus (smooth ReLU) – positive output only
  - Scaled sigmoid/tanh – fixed output range
- Loss function
  - Mean squared error (L2 norm)
  - Mean absolute error (L1 norm) when there are a lot of outliers
  - Huber loss is a combination
- Regression MLP summary

Table 10-1. Typical regression MLP architecture

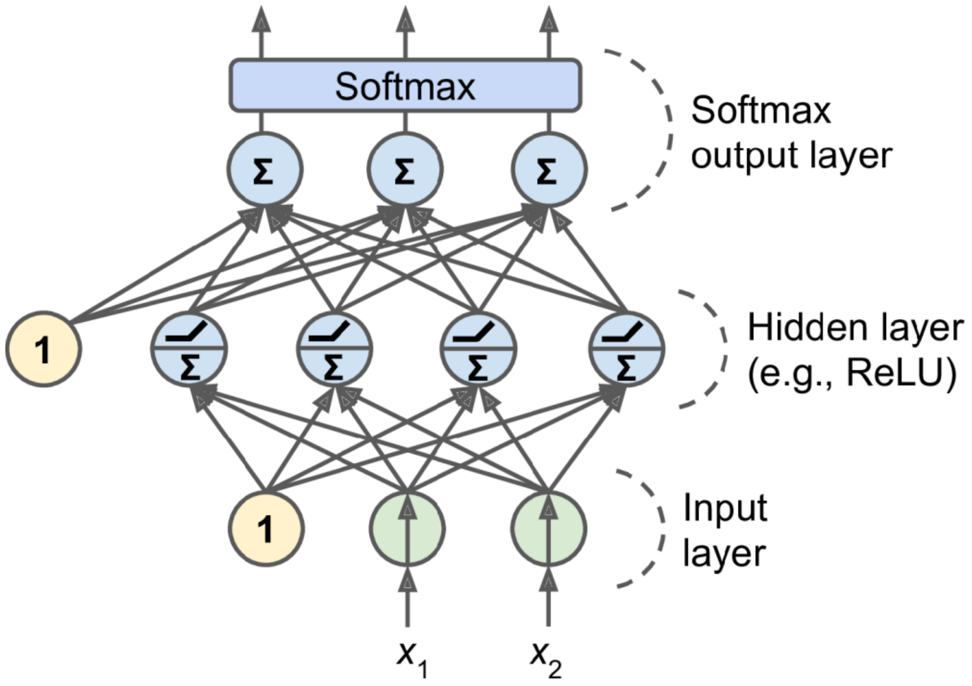
Hyperparameter	Typical value
# input neurons	One per input feature (e.g., 28 x 28 = 784 for MNIST)
# hidden layers	Depends on the problem, but typically 1 to 5
# neurons per hidden layer	Depends on the problem, but typically 10 to 100
# output neurons	1 per prediction dimension
Hidden activation	ReLU (or SELU, see Chapter 11)
Output activation	None, or ReLU/softplus (if positive outputs) or logistic/tanh (if bounded outputs)
Loss function	MSE or MAE/Huber (if outliers)

CLASSIFICATION MLPs I

- Single class (binary) – single output neuron
  - Output between [0,1] using sigmoid
  - Estimate probability of positive class (confidence)
- Multilabel binary – output neuron for every binary classification
  - Output between [0,1] using sigmoid
  - Output probabilities do not sum to one
  - Combinational output space

CLASSIFICATION MLPs II

- Multiclass classification – multiple possible classes (e.g. number 0-9)
  - Each input instance can only belong to a single class (>2)
  - One output neuron per class
  - Softmax activation on the full output layer (Chapter 4 pg 148)
    - $\hat{p}_k = \sigma(s(x))_k = \frac{\exp(s_k(x))}{\sum_j \exp(s_j(x))}$
    - $s_k(x) = (\theta^{(k)})^T x$
    - Estimated probabilities between [0,1] and sum to 1
- Cross entropy loss
  - $J(\theta) = -\frac{1}{m}\sum_i \sum_k y_k^{(i)} \log\left(\hat{p}_k^{(i)}\right)$
  - Penalizes models with low probability estimate for the ground truth class



- Classification summary

Table 10-2. Typical classification MLP architecture

Hyperparameter	Binary classification	Multilabel binary classification	Multiclass classification
Input and hidden layers	Same as regression	Same as regression	Same as regression
# output neurons	1	1 per label	1 per class
Output layer activation	Logistic	Logistic	Softmax
Loss function	Cross entropy	Cross entropy	Cross entropy

IMPLEMENTATION

- Follow Chapter 2 for machine setup (Get the Data section)
  - Highly recommend use of Anaconda Python for setting up your sandbox
  - Google Colab is convenient and free with GPU access
  - Additional notes from Stanford
- Read and follow Implementing MLPs with Keras section → installation of Keras and TensorFlow2

FINE-TUNING HYPERPARAMETERS

- Many hyperparameters must be tweaked for good model performance
- Grid search can evaluate different hyperparameter combinations → slow
  - Book gives other libraries for hyperparam optimization
  - These typically explore more in good hyperparameter space
- Number of hidden layers → deeper is better
  - Transfer learning – reuse lower layers from network trained on large dataset (good initialization and avoid cost of learning from scratch)
- Number of neurons per hidden layers → use fixed size
- Activation function → ReLU works well
- Learning rate – very important parameter, need learning schedule
- Optimizer – more than just mini-batch gradient descent (e.g. Adam)
- Batch size – significant impact on model performance and training time
  - Large batch – efficiently process for reduced training time → maximize for GPU with learning rate warm-up (schedule)
  - Small batch – more stable early in learning and good generalization