CS578 – Interactive and Transparent Machine Learning

Topic: Neural Networks

Mustafa Bilgic

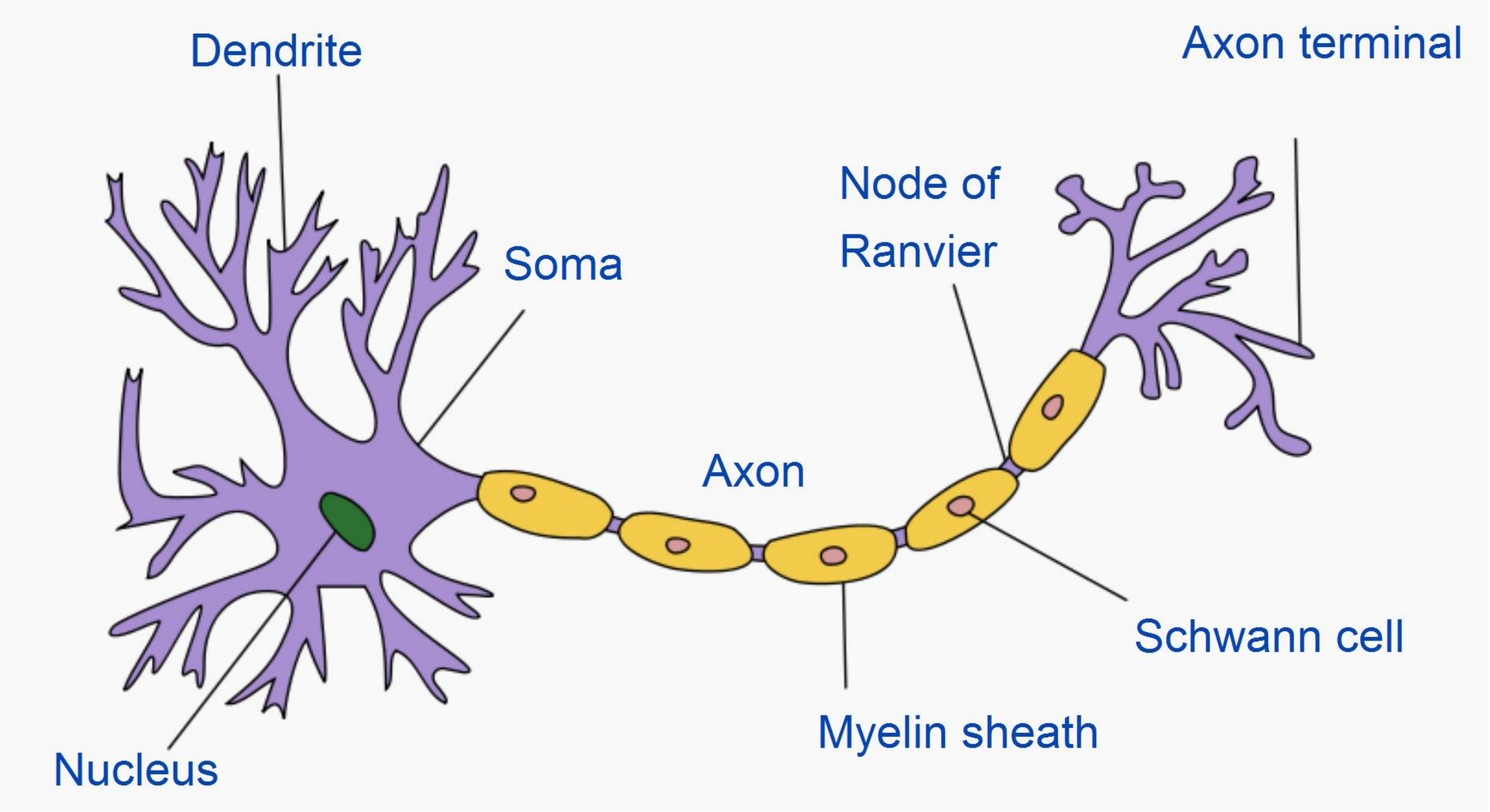
http://www.cs.iit.edu/~mbilgic

https://twitter.com/bilgicm

Motivation

- Inspired by neurons in the brain

Neuron



By Quasar Jarosz at English Wikipedia, CC BY-SA 3.0,
https://commons.wikimedia.org/w/index.php?curid=7616130

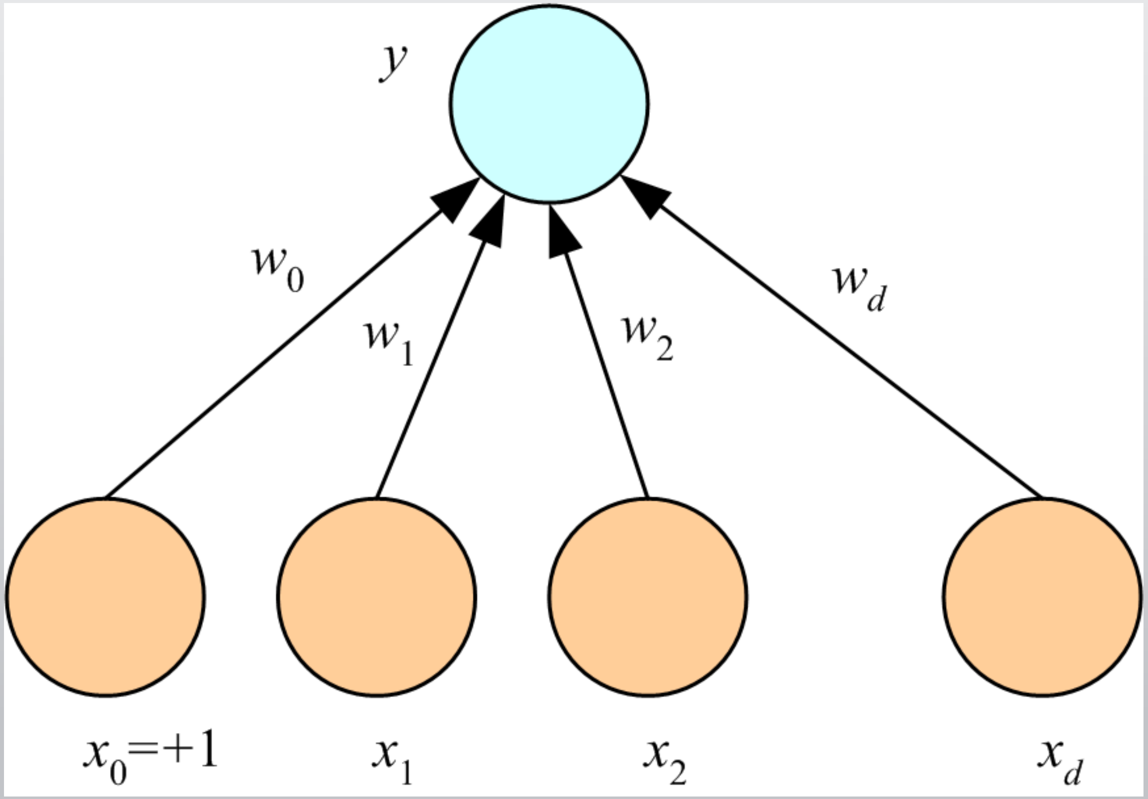
NEURON

- Neurons can have multiple dendrites and at most one axon
- Typical connections are from an axon of a neuron to dendrites of other neurons
- Synaptic signals are received through dendrites and somas; signals are transmitted through axons
- Signals can excite or inhibit the receiving neuron
- A neuron fires when the excitement is above a threshold

Artificial Neural Networks

- Artificial neural networks are inspired by real neurons
- 1943 – One of the first neural computational models was proposed by McCulloch and Pits
- 1958 – Rosenblatt proposed perceptron
- 1969 – A paper by Minsky and Papert almost killed the entire field
  - Perceptrons are incapable of representing XOR
  - Computational resources are too great
- 1975 – Backpropagation algorithm renewed interest in neural networks
- 1980s – parallel architectures were popular
- Late 1990s and 2000s – other methods, such as support vector machines, became more popular
- 2010s – neural networks of several hidden layers are back with the new name “deep learning”

PERCEPTRON



$$y = sign(w_0 + \sum w_i x_i)$$

Credit: Ethem Alpaydin. Introduction to Machine Learning. 3rd Edition. http://www.cmpe.boun.edu.tr/~ethem/i2ml3e

Error Function For Perceptrons

- $y \in \{-1, +1\}$
- For $y = +1$, we want $\boldsymbol{w}^T x > 0$ and for y = -1, we want $\boldsymbol{w}^T x < 0$. That is, want $y\boldsymbol{w}^T x > 0$
- $Error(\boldsymbol{w}) = -\sum_{\langle \boldsymbol{x}, y\rangle \in misclassified} \boldsymbol{w}^T xy$
- Take derivative of *Error(**w**)* with respect to ***w*** and perform gradient optimization

EXAMPLES

- Logical AND
- Logical OR
- Logical XOR

Simple Multilayer Network for XOR

- $XOR(A, B) = (A \wedge \neg B) \vee (\neg A \wedge B)$
- One perceptron for $(A \wedge \neg B)$
- One perceptron for $(\neg A \wedge B)$
- One perceptron for combining the outputs, through OR, of the two previous perceptrons

What an Artificial Neuron Does

- Takes a weighted sum of its inputs
  - $w_0 + \sum_{i=1}^{k} w_i x_i$
  - Assume that there is always a constant input 1, that is, $x_0 = 1$. Then,
  - $\sum_{i=0}^{k} w_i x_i$
- Passes this sum through its activation function
  - $f\left(\sum_{i=0}^{k} w_i x_i\right)$

Various Activation Functions

- Identity function
- Bipolar step function
- Binary sigmoid
- Bipolar sigmoid
- Hyperbolic tangent

Identity Function

- $f\left(\sum_{i=0}^{k} w_i x_i\right) = \sum_{i=0}^{k} w_i x_i$
- Typically used for the output neurons, when the task is regression
- Beware of using the identity function in the hidden layers
  - Linear combination of linear functions is another linear function

Bipolar Step Function

- $f\left(\sum_{i=0}^{k} w_i x_i\right) = sign\left(\sum_{i=0}^{k} w_i x_i\right)$
- Returns either +1 or -1 (except right on the decision boundary)
- Useful for both hidden layers and output layer
- However, its discontinuous and it is problematic for learning algorithms that require taking its derivative

BINARY SIGMOID

- $f\left(\sum_{i=0}^{k} w_i x_i\right) = \frac{1}{1+e^{-\sum_{i=0}^{k} w_i x_i}}$
- This is the logistic function that we used
  - Except, notice the minus sign in front of the sum
    - This is only a convention and does not change much
- The output of the binary sigmoid is between 0 and 1
  - Useful for output layer when the task is classification
  - The output can be interpreted as a probability

Hyperbolic Tangent

- $f\left(\sum_{i=0}^{k} w_i x_i\right) = \frac{e^{\sum_{i=0}^{k} w_i x_i} - e^{-\sum_{i=0}^{k} w_i x_i}}{e^{\sum_{i=0}^{k} w_i x_i} + e^{-\sum_{i=0}^{k} w_i x_i}}$
- The output of tanh is between -1 and +1
  - Useful for output layer when the task is classification
  - Useful for both hidden and output layers

Error Functions

- Regression – squared error
  - $\frac{1}{2}(t-y)^2$
  - t: the true target value
  - y: the predicted value
- Classification – log-loss, negative CLL
  - $-(1-t) \times ln(1-y) - t \times ln(y)$
  - t: the true target value (0/1)
  - y: probability of class 1
- Obviously, there are other error functions as well; these are two of the most commonly used ones

BACKPROPAGATION ALGORITHM

- In a nutshell
  - Take the derivative of the error at the output
  - Backpropagate the error in the direction of the input
- Derived in the CS584 class

Various Network Architectures

- Autoencoder
  - Input and Output are the same
  - Input->Hidden Layer: encoding, Hidden Layer -> Output: decoding
- Convolutional neural networks (CNNs)
  - Captures local features
  - Especially common for image analysis, but can be used for sequence data as well
- Recurrent neural networks
  - Backward connections
  - Serves as memory
  - Especially useful for sequence data (e.g., time-series, text, etc.)
  - Example: Long Short-Term Memory (LSTM) networks

OVERFITTING

- Neural networks are pretty powerful models
- Even with a single hidden layer, they are "universal approximators", i.e., they can approximate arbitrary functions arbitrarily close
- Therefore, it is very easy to overfit them
- To prevent overfitting, utilize
  - Domain knowledge
  - Shared parameters
  - Validation data
  - Regularization

Deep Learning

- Several hidden layers
  - Millions of parameters
- Big data, big computation
- Strongly-recommended reading
  - "Why and When Can Deep -- but Not Shallow -- Networks Avoid the Curse of Dimensionality: a Review" https://arxiv.org/abs/1611.00740

TRANSPARENCY

- In increasing order of opaqueness
  - Rules
  - Decision trees
  - Linear models
  - Non-linear models
    - SVMs with non-linear kernels
    - Neural networks
- There is a large body of recent work on increasing transparency for non-linear models

A Small Sample of Transparency Papers

- "Why should I trust you?: explaining predictions of any classifier" https://arxiv.org/abs/1602.04938
- "Axiomatic attribution for deep networks" https://arxiv.org/abs/1703.01365
- "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps" https://arxiv.org/abs/1312.6034
- "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation" https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0130140

A Few Libraries

- Scikit-learn
  - https://scikit-learn.org/stable/modules/neural_networks_supervised.html
- Keras
  - https://keras.io/
- Tensorflow
  - https://github.com/tensorflow/tensorflow
- CNTK
  - https://github.com/Microsoft/cntk
- Theano
  - https://github.com/Theano/Theano
- PyTorch
  - https://github.com/pytorch/pytorch
- Fast.ai
  - https://www.fast.ai/
- Others
  - Use your favorite search engine