Pattern Classification

EET3053

Lecture 08: Artificial Neural Network

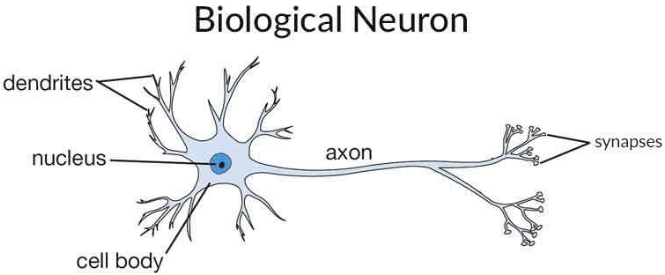
Dr. Kundan Kumar
Associate Professor
Department of ECE

Faculty of Engineering (ITER)
S‘O’A Deemed to be University, Bhubaneswar, India-751030

Neural Networks

What is Neural Network?

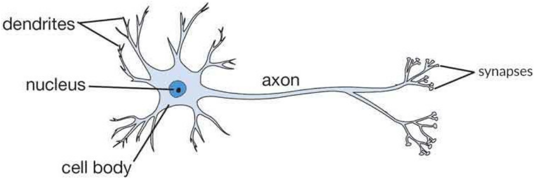
- *Neural Networks* are networks of neurons, for example, as found in real (i.e. biological) brains (86 billion neurons).

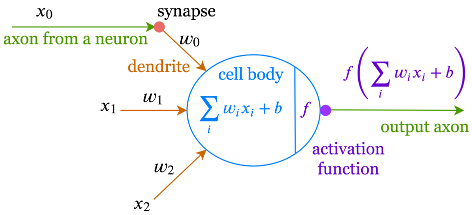


- Dendrite: Receives signals from other neurons
- Soma/Cell body: Processes the information
- Axon: Transmits the output of this neuron
- Synapse: Point of connection to other neurons

Artificial Neural Network (ANNs)

- *Artificial Neural Networks* (ANNs) are network of Artificial Neurons and hence constitute crude approximation to parts of real brains.
- The brain uses chemicals to transmit information; the computer uses electricity.





Why are Artificial Neural Networks worth studying?

- They are extremely powerful computational devices.
- Massive parallelism makes them very efficient.
- They can learn and generalize from training data – so there is no need for enormous programming skill.

Neural Network Applications

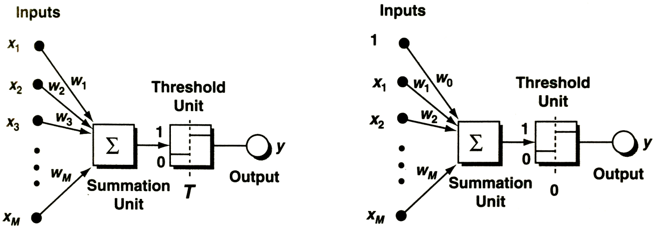
- Real world application
  - Financial modeling – predicting the stock market
  - Time series prediction – climate, weather, seizures
  - Computer games – intelligent agents, chess, backgammon
  - Robotics – autonomous adaptable robots
  - Pattern recognition – speech recognition, seismic activity, sonar signals
  - Data analysis – data compression, data mining
  - Bioinformatics – DNA sequencing, alignment

Brain vs. Computers

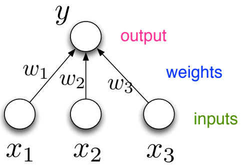
- *Processing elements:* There are 10^{14} synapses in the brain, compared with 10^8 transistors in the computer
- *Processing speed:* 100 Hz for the brain compared to 10^9 Hz for the computer
- *Style of computation:* The brain computes in parallel and distributed mode, whereas the computer mostly serially and centralized.
- *Fault tolerant:* The brain is fault tolerant, whereas the computer is not.
- *Adaptive:* The brain learns fast, whereas the computer doesn't even compare with an infant's learning capabilities
- *Intelligence and consciousness:* The brain is highly intelligent and conscious, whereas the computer shows lack of intelligence
- *Evolution:* The brains have been evolving for tens of millions of years, computers have been evolving for decades.

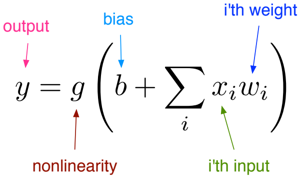
An abstract mathematical model of a neuron

- McCulloch and Pitts given a first attempt to form an abstract mathematical model of a neuron in 1943.
- The binary linear classification model
  - receives a finite number of inputs $x_1, x_2, \ldots, x_d$
  - computes the weighted sum $z = \sum_{i=1}^{d} w_i x_i$ using the weights $w_1, w_2, \ldots, w_d$
  - thresholds s and output 0 or 1 depending on whether the weighted sum is less than or greater than a given threshold value T.

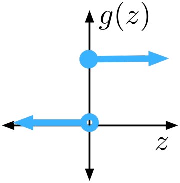


McCulloch and Pitts Model





$$y = g\left(b + \sum_i x_i w_i\right)$$



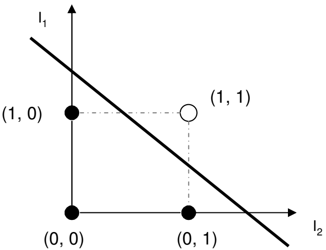
Decision Boundaries for AND and OR

We can now plot the decision boundaries of our logic gates

AND

w1=1, w2=1, θ=1.5

AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1

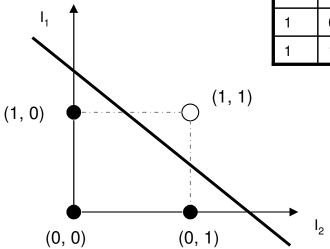


Decision Boundaries for AND and OR

We can now plot the decision boundaries of our logic gates

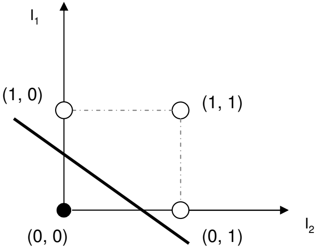
AND
w1=1, w2=1, θ=1.5

AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1



OR
w1=1, w2=1, θ=0.5

OR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	1



Limitations Of McCulloch-Pitts Neuron

- What about non-boolean (say, real) inputs?
- Do we always need to hand code the threshold?
- Are all inputs equally important? What if we want to assign more importance to some inputs?
- What about functions which are not linearly separable? Say XOR function.

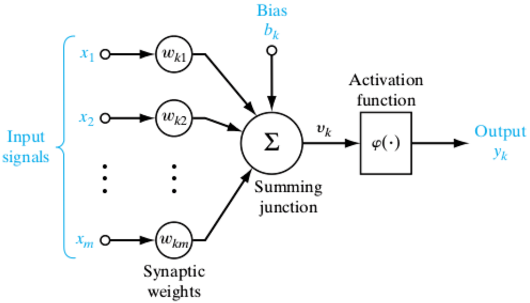
Perceptron Model

Perceptron Model

- Overcoming the limitations of the McCulloch-Pitts model, Frank Rosenblatt proposed the classical perceptron model in 1958.
- Upgraded by Minsky-Papert in 1969.
  - More generalize computational model than McCulloch-Pitts model.
  - Can learn weights and threshold.
- Difference between McCullock Pitts Neuron and Perceptron:
  - In perceptron, weights and bias are allowed to learn (already we did in linear classifiers).
  - Not limited to only boolean inputs.

Basic Neural Model of Perceptron

- Input to neurons:
  - Input x_i arise from other neurons or from outside the network
  - Nodes whose inputs arise outside the network are called input nodes and simply copy values
  - An input may **excite** or **inhibit** the response of the neuron to which it is applied, depending upon the weight of the connection.
- Synaptic efficacy is modeled using real weight w_i
- The response of the neuron is a nonlinear function f of its weighted inputs



Single Layer Perceptron

- The perceptron is an algorithm for supervised learning of binary classifiers.
- The perceptron algorithm is also termed the single-layer perceptron, to distinguish it from a multilayer perceptron.
- The single-layer perceptron is the simplest feedforward neural network.
- Perceptron algorithm
  - Input: A set of examples, $(\mathrm{x}_1, y_1), (\mathrm{x}_2, y_2), \ldots, (\mathrm{x}_n, y_n)$
  - Output: A perceptron model defined by $(w_0, w_1, \ldots, w_d)$

begin initialize $\mathbf{a}, \eta(\cdot), \text{criterion } \theta, k = 0$

  do $k \leftarrow k + 1$

    $\mathbf{a} \leftarrow \mathbf{a} + \eta(k) \sum_{\mathbf{y} \in \mathcal{Y}_k} \mathbf{y}$

  until $\eta(k) \sum_{\mathbf{y} \in \mathcal{Y}_k} \mathbf{y} < \theta$

  return $\mathbf{a}$

end

Single Layer Perceptron

- Other form of Perceptron Learning Rule
  - If $t = 1$ and $z = \mathrm{w}^T\mathrm{x} > 0$
    - then $y = 1$, so no need to change anything.
  - If $t = 1$ and $z < 0$
    - then $y = 0$, so we want to make z larger.
    - Update:

$$\mathrm{w} \leftarrow \mathrm{w}' + \mathrm{x}$$

  - Justification

$$\begin{aligned} \mathrm{w}'^T\mathrm{x} &= (\mathrm{w} + \mathrm{x})^T\mathrm{x} \\ &= \mathrm{w}^T\mathrm{x} + \mathrm{x}^T\mathrm{x} \\ &= \mathrm{w}^T\mathrm{x} + ||\mathrm{x}|| \end{aligned}$$

Perceptron Learning Rule

- For convenience, let targets be $\{-1, 1\}$ instead of our usual $\{0, 1\}$.

$$z = \mathbf{w}^T \mathbf{x}$$

$$y = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

- Perceptron Learning Rule:

$$\text{For each training case } (\mathbf{x}^{(i)}, t^{(i)}),$$

$$z^{(i)} \leftarrow \mathbf{w}^T \mathbf{x}^{(i)}$$

$$\text{If } z^{(i)} t^{(i)} \leq 0,$$

$$\mathbf{w} \leftarrow \mathbf{w} + t^{(i)} \mathbf{x}^{(i)}$$

Perceptron Learning Rule

- How can we define a sensible learning criterion when the dataset isn't linearly separable?
- Why classification error and squared error are problematic cost functions for classification?
- Recall from linear classifier, can we apply gradient descent to update the weights?
- If yes then which cost/criteria function will be appropriate?
- Gradient Descent Algorithm to update the weights and bias:

Algorithm 1 (Basic gradient descent)

begin initialize $\mathbf{a}$, criterion $\theta, \eta(\cdot), k = 0$
  do $k \leftarrow k + 1$
    $\mathbf{a} \leftarrow \mathbf{a} - \eta(k)\nabla J(\mathbf{a})$
  until $\eta(k)\nabla J(\mathbf{a}) < \theta$
return $\mathbf{a}$
end

$0 - 1$ Loss criteria function

$$\mathcal{L}_{0-1}(y,t) = \begin{cases} 0 & \text{if } y = t \\ 1 & \text{otherwise.} \end{cases}$$

- This is the same criteria function that we used earlier.
- Problem: how to optimize?
- Chain rule:

$$\frac{\partial \mathcal{L}_{0-1}}{\partial w_j} = \frac{\partial \mathcal{L}_{0-1}}{\partial z} \frac{\partial z}{\partial w_j}$$

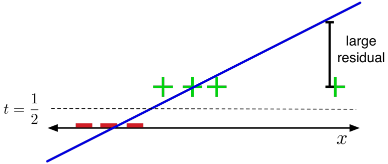
- But $\frac{\partial \mathcal{L}_{0-1}}{\partial z}$ is zero everywhere it's defined!
  - $\frac{\partial \mathcal{L}_{0-1}}{\partial w_j}$ means that changing the weights by a very small amount probably has no effect on the loss.
  - The gradient descent update is a no-op.

Squared Error Loss Function

$$y = \mathbf{w}^\top \mathbf{x} + b$$

$$\mathcal{L}_{\mathrm{SE}}(y, t) = \frac{1}{2}(y - t)^2$$

- Doesn't matter that the targets are actually binary.
- Threshold predictions at $y = 1/2$



- The loss function hates when you make correct predictions with high confidence!
- If $t = 1$, it's more unhappy about $y = 10$ than $y = 0$.

Logistic nonlinearity

- There's obviously no reason to predict values outside $[0, 1]$. Let's squash y into this interval.
- The logistic function is a kind of sigmoidal, or S-shaped, function:

$$z = \mathbf{w}^\top \mathbf{x} + b$$
$$y = \sigma(z)$$
$$\mathcal{L}_{\text{SE}}(y, t) = \frac{1}{2}(y - t)^2.$$

$$z = \mathbf{w}^\top \mathbf{x} + b$$
$$y = \sigma(z)$$
$$\mathcal{L}_{\text{SE}}(y, t) = \frac{1}{2}(y - t)^2.$$

- A linear model with a logistic nonlinearity is known as log-linear:

Logistic nonlinearity: chain rule

- Chain Rule: derivative with respect to the weights

$$\begin{aligned}\frac{\mathrm{d}\mathcal{L}_{\mathrm{SE}}}{\mathrm{d}z} &= \frac{\mathrm{d}\mathcal{L}_{\mathrm{SE}}}{\mathrm{d}y}\frac{\mathrm{d}y}{\mathrm{d}z} \\ &= (y-t)y(1-y) \\ \frac{\partial\mathcal{L}_{\mathrm{SE}}}{\partial w_j} &= \frac{\mathrm{d}\mathcal{L}_{\mathrm{SE}}}{\mathrm{d}z}\frac{\partial z}{\partial w_j} \\ &= \frac{\mathrm{d}\mathcal{L}_{\mathrm{SE}}}{\mathrm{d}z}\cdot x_j.\end{aligned}$$

- derivative with respect to the bias:

$$\begin{aligned}\frac{\mathrm{d}\mathcal{L}_{\mathrm{SE}}}{\mathrm{d}z} &= \frac{\mathrm{d}\mathcal{L}_{\mathrm{SE}}}{\mathrm{d}y}\frac{\mathrm{d}y}{\mathrm{d}z} \\ &= (y-t)y(1-y) \\ \frac{\partial\mathcal{L}_{\mathrm{SE}}}{\partial} &= \frac{\mathrm{d}\mathcal{L}_{\mathrm{SE}}}{\mathrm{d}}\frac{\partial z}{\partial}\end{aligned}$$

Cross-Entropy Loss

- Cross-entropy (CE) is defined as follows:

$$\mathcal{L}_{\text{CE}}(y,t) = \begin{cases} -\log y & \text{if } t = 1 \\ -\log 1 - y & \text{if } t = 0 \end{cases}$$

$$\mathcal{L}_{\text{CE}}(y,t) = -t \log y - (1-t) \log 1 - y.$$

- When we combine the logistic activation function with cross-entropy loss, you get logistic regression:

$$z = \mathbf{w}^\top \mathbf{x} + b$$
$$y = \sigma(z)$$
$$\mathcal{L}_{\text{CE}} = -t \log y - (1-t) \log 1 - y.$$

Cross-Entropy Loss: Chain rule

- Chain rule:

$$z = \mathbf{w}^\top \mathbf{x} + b$$
$$y = \sigma(z)$$
$$\mathcal{L}_{\mathrm{CE}} = -t \log y - (1-t) \log 1 - y.$$

Example on perceptron

Question Compute the updated weights and bias using perceptron algorithm to model the AND gate. Consider the initial weight and bias as 0 and learning rate as 0.5.

Perceptron Model: Multiclass Problem

Multiclass classification

- What about classification tasks with more than two categories?
- Targets form a discrete set $\{1, \ldots, K\}$
- It's often more convenient to represent them as one-hot vectors, or a one-of-K encoding:

$$\mathbf{t} = \underbrace{(0, \ldots, 0, 1, 0, \ldots, 0)}_{\text{entry } k \text{ is } 1}$$

- Now there are d input dimensions and K output dimensions, so we need $K \times d$ weights, which we arrange as a weight matrix W.
- Also, we have a K-dimensional vector b of biases.

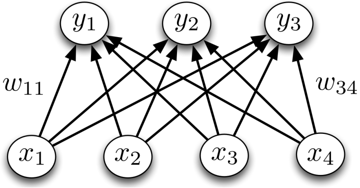
Multiclass classification

- Linear predictions:

$$z_k = \sum_j w_{kj} x_j + b_k$$

- Vectorized:

$$\mathrm{z} = W\mathrm{x} + \mathrm{b}$$



Multiclass classification

- A natural activation function to use is the softmax function, a multivariable generalization of the logistic function:

$$y_k = \text{softmax}(z_1, \ldots, z_K)_k = \frac{e^{z_k}}{\sum_{k'} e^{z_{k'}}}$$

- The inputs z_k are called the logits.
- Properties:
  - Outputs are positive and sum to 1 (so they can be interpreted as probabilities)
  - If one of the z_k's is much larger than the others, softmax(z) is approximately the argmax. (So really it's more like "soft-argmax".)
  - Exercise: how does the case of $K = 2$ relate to the logistic function?
- Note: sometimes $\sigma(z)$ is used to denote the softmax function.

Multiclass classification

- If a model outputs a vector of class probabilities, we can use cross-entropy as the loss function:

$$\mathcal{L}_{\mathrm{CE}}(\mathbf{y}, \mathbf{t}) = -\sum_{k=1}^{K} t_k \log y_k$$
$$= -\mathbf{t}^\top (\log \mathbf{y}).$$

  where the log is applied element wise.

- Just like with logistic regression, we typically combine the softmax and cross-entropy into a softmax-cross-entropy function.

Multiclass classification

- Multiclass logistic regression:

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$
$$\mathbf{y} = \mathrm{softmax}(\mathbf{z})$$
$$\mathcal{L}_{\mathrm{CE}} = -\mathbf{t}^{\top}(\log \mathbf{y})$$

- Tutorial: deriving the gradient descent updates

$$\frac{\partial \mathcal{L}_{\mathrm{SCE}}}{\partial \mathbf{z}} = \mathbf{y} - \mathbf{t}$$

- Softmax regression is an elegant learning algorithm which can work very well in practice.

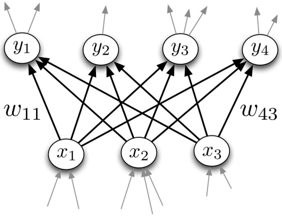
Input to Neurons

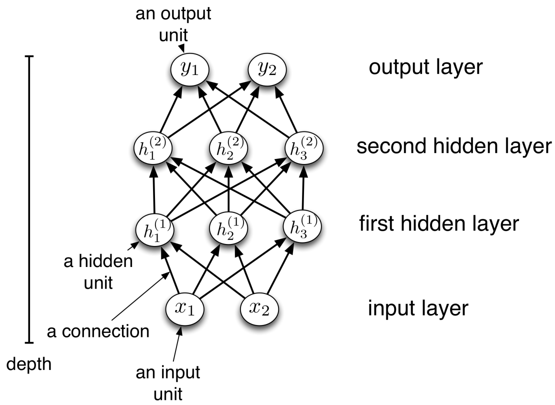
- Arise from other neurons or from outside the network
- Nodes whose inputs arise outside the network are called input nodes and simply copy values
- An input may excite or inhibit the response of the neuron to which it is applied, depending upon the weight of the connection.

Mutlilayer Perceptrons

Mutlilayer Perceptrons

- Feed-forward neural network is a fully connected directed acyclic graph.
- In contrast to recurrent neural networks, which can have cycles (out of the scope of this course).
- Typically, units are grouped together into layers.

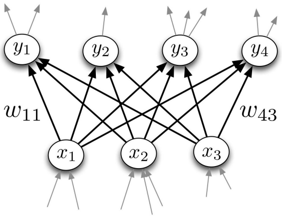


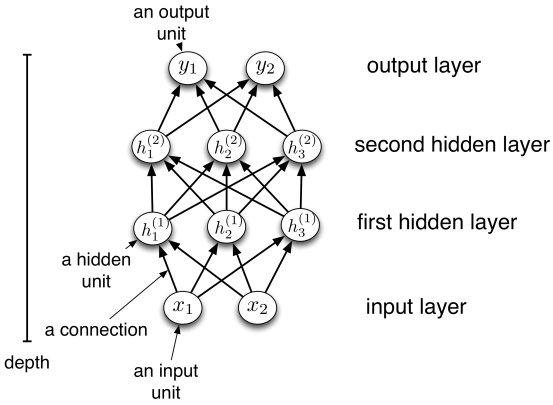


Courtesy: Roger Grosse, Lecture Notes

Mutlilayer Perceptrons

- Each layer connects N input units to M output units. Note: the inputs and outputs for a layer are distinct from the inputs and outputs to the network.
- We need an $M \times N$ weight matrix, W.
- The output units are a function of the input units: $y = f(x) = (Wx + b)$

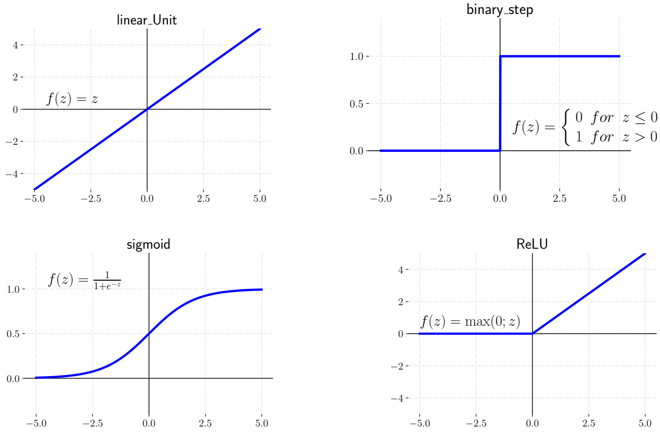




Courtesy: Roger Grosse, Lecture Notes

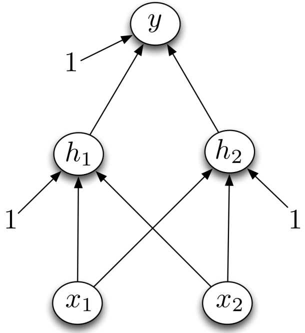
Multilayer Perceptrons

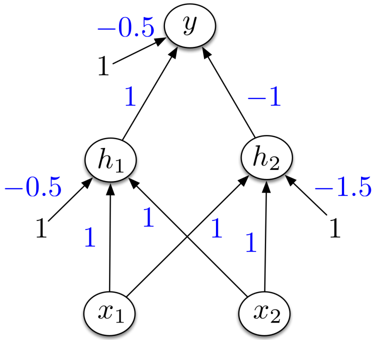
- Some activation functions



Example: Exclusive OR

- Designing a network to compute XOR: Assume hard threshold activation function

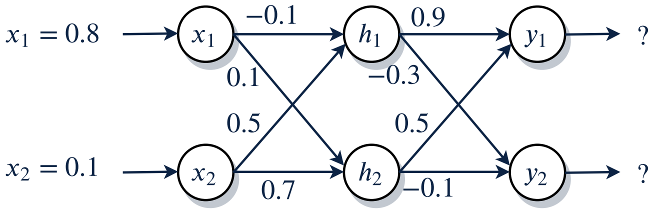




Forward-Propagation

- Propagate the input through the network:
  - Assume sigmoid activation function,
  - Bias is dropped for simplification

$$y_i = f\left(\sum_j w_{ji}^{(2)} f\left(\sum_k w_{kj}^{(1)} x_k\right)\right) \quad \text{for one hidden layer}$$



Backpropagation Learning Algorithm

will update soon

References

[1] Hart, P. E., Stork, D. G., & Duda, R. O. (2000). Pattern classification. Hoboken: Wiley.

[2] Gose, E. (1997). Pattern recognition and image analysis.

Thank you!