Lecture 5

Introduction to Artificial Neural Networks

Physical Address for Prof. Ilias Tagkopoulos
Computer Science:
Office: 3063 Kemper Hall
Phone: (530) 752-4821
Fax: (530) 752-4767

Genome and Biomedical Sciences Facility:
Office: 5313 GBSF
Phone: (530) 752-7707
Fax: (530) 754-9658

Instructor: Ilias Tagkopoulos
iliast@ucdavis.edu

(Murphy's book, sections 8.6.1, 16.5

From last lecture: Logistic Regression

- Logistic regression is a classification method that uses the logistic function

$$g(x; w) = sigm(w^T x) = \frac{1}{1 + e^{-w^T x}}$$

to classify any new sample i (assign $y^{(i)}$ depending whether $g(x^{(i)}; w)$ is more or less than zero).

- To find the weights (parameters) **w** we once again resort to gradient decent:

$$w_j := w_j + a\frac{\partial l(w)}{\partial w_j} = w_j + a\left(y^{(i)} - g(x^{(i)}; w)\right)x_j^{(i)}$$

From last lecture: Logistic Regression

- w updates are based on the following rule:

$$w_j := w_j + a\frac{\partial l(w)}{\partial w_j}$$

Or

$$w_j := w_j + a\left(y^{(i)} - g(x^{(i)}; w)\right)x_j^{(i)}$$

- Btw, gradient descent is not the only method to find the parameter w. For example we can use Newton's method

$$w_j := w_j + a\frac{\frac{\partial l(w)}{\partial w_j}}{\frac{\partial^2 l(w)}{(\partial w_j)^2}}$$

Perceptron

- Actually if we use the same update rule but with hard boundaries, forcing the output to be {0,1}, we have the perceptron learning algorithm

$$w_j := w_j + a\left(y^{(i)} - g(x^{(i)}; w)\right)x_j^{(i)}$$

with

$$g(z) = \begin{cases} 0 & if \quad z < threshold \\ 1 & if \quad z \geq threshold \end{cases}$$

Threshold can be any scalar (e.g. 0).

- Generally Stochastic Gradient Descent on logistic regression is faster and has a better performance.

Perceptron

- The Perceptron Learning Algorithm is the same as logistic regression, but it enforces $g(x^{(i)}; w)$ to be a step function, instead of the (smooth) sigmoid function:

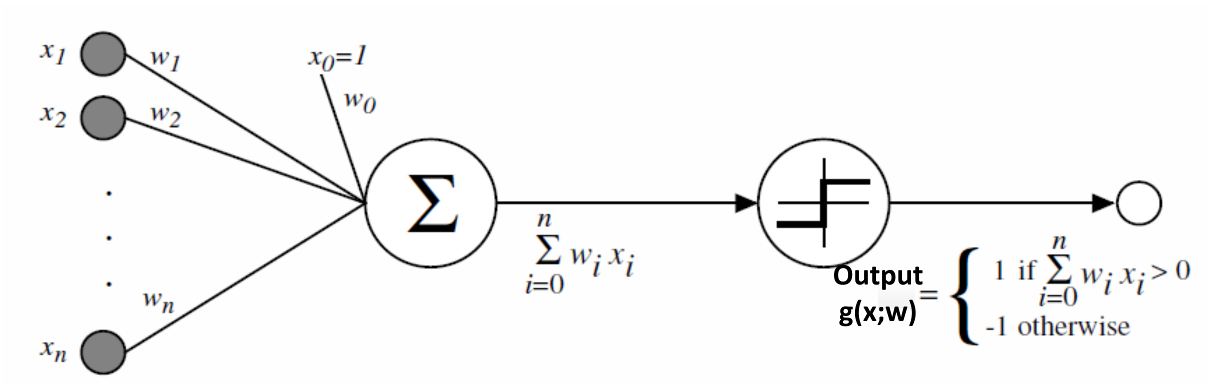
$$w_j := w_j + a\left(y^{(i)} - g(x^{(i)}; w)\right)x_j^{(i)}$$

with

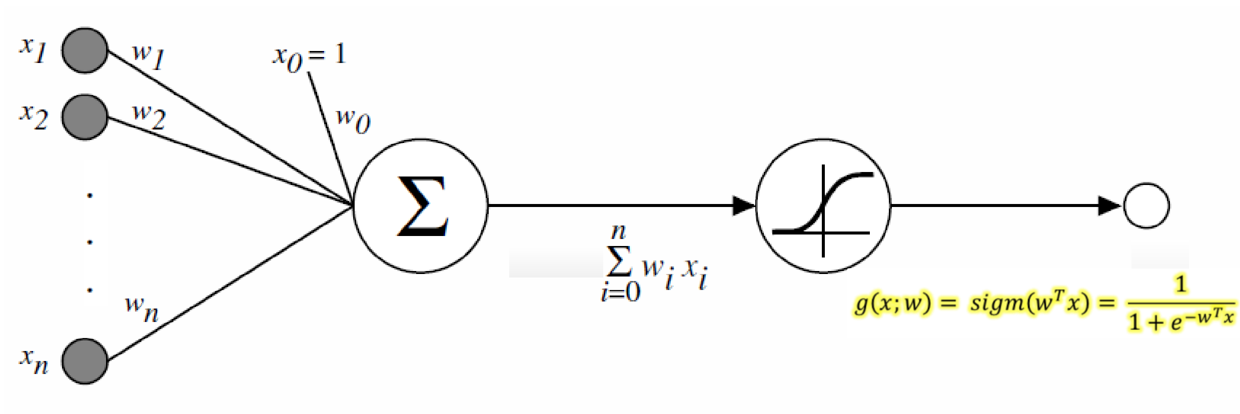
$$g(x^{(i)}; w) = \begin{cases} 0 & if \;\; w^T x^{(i)} < 0 \\ 1 & if \;\; w^T x^{(i)} \geq 0 \end{cases}$$

Graphical comparison between perceptron and logistic unit

Perceptron



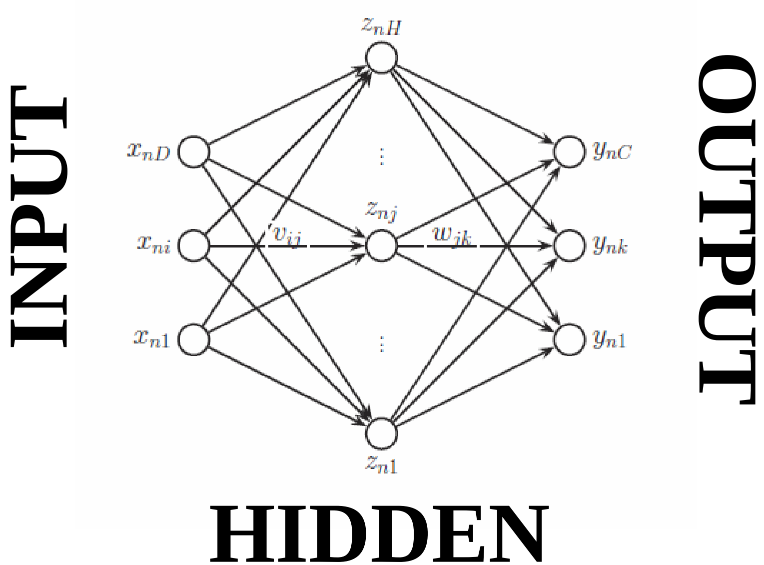
Logistic unit



Can you create an 2-input AND gate with a perceptron? What about an XOR gate?

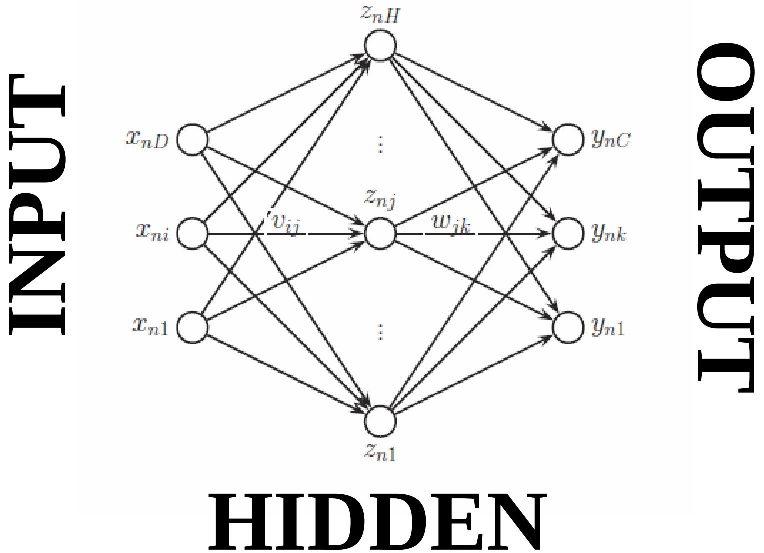
Feed-Forward Neural Network

- The Feed-forward Neural Network (FFNN) is a network with two or more layers of neurons, usually either perceptrons or logistic regression.
  - The final layer can be another logistic regression/perceptron or a linear regression model depending whether it is a classification or regression problem.
  - Also called multi-layer perceptron (MLP)
  - Remarkably, the MLP is a "universal approximator": It can model any smooth function to any accuracy level (for specific activation functions, including sigmoidal).



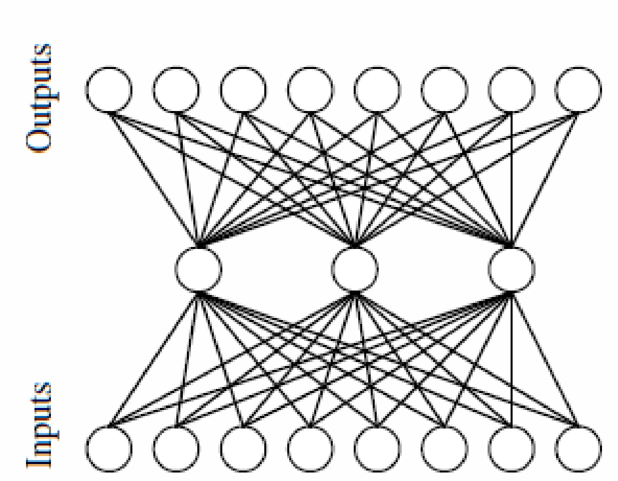
Feed-Forward Neural Network

- How many hidden layers do you need?
  - Generally 1-2 are enough
  - Too many hidden layers will increase training time, especially in cases when the "error correction" is propagated backwards.
- How many hidden nodes per layer?
  - Not easy to know
  - You need just the right amount: too many will take forever to train; too few and the model complexity will be too low to learn the underlying structure.
  - Trial-and-error…



Example of a FFNN

- Imagine a Feed-Forward Neural Network



- And a target function:

Input		Output
10000000	$\rightarrow$	10000000
01000000	$\rightarrow$	01000000
00100000	$\rightarrow$	00100000
00010000	$\rightarrow$	00010000
00001000	$\rightarrow$	00001000
00000100	$\rightarrow$	00000100
00000010	$\rightarrow$	00000010
00000001	$\rightarrow$	00000001

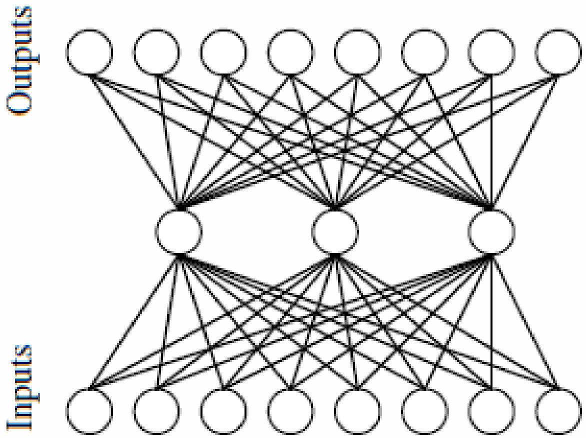
Example from T. Michell's Book "Machine Learning"

Example of a FFNN

- Learned hidden layer representation

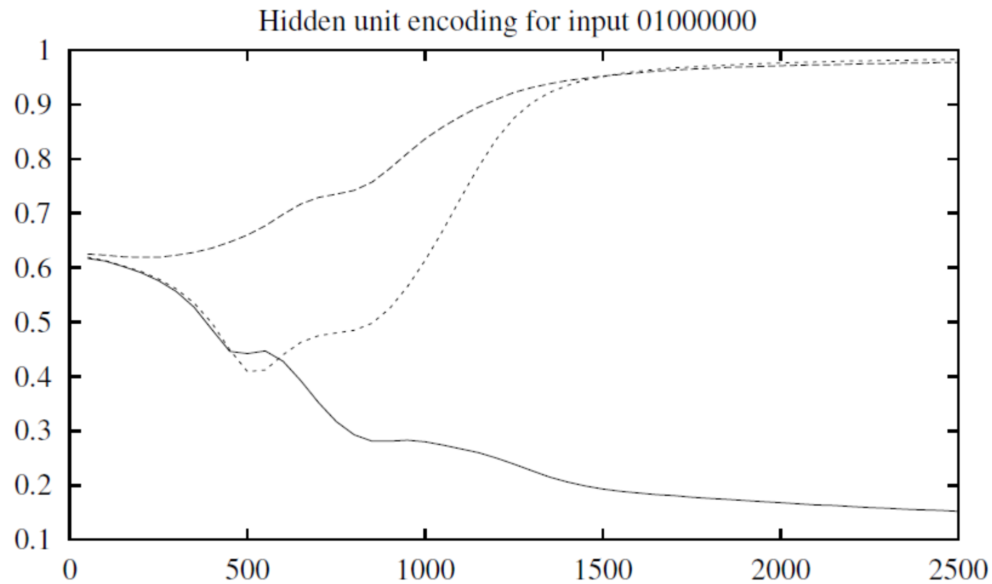
Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.01	.11	.88	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.22	.99	.99	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

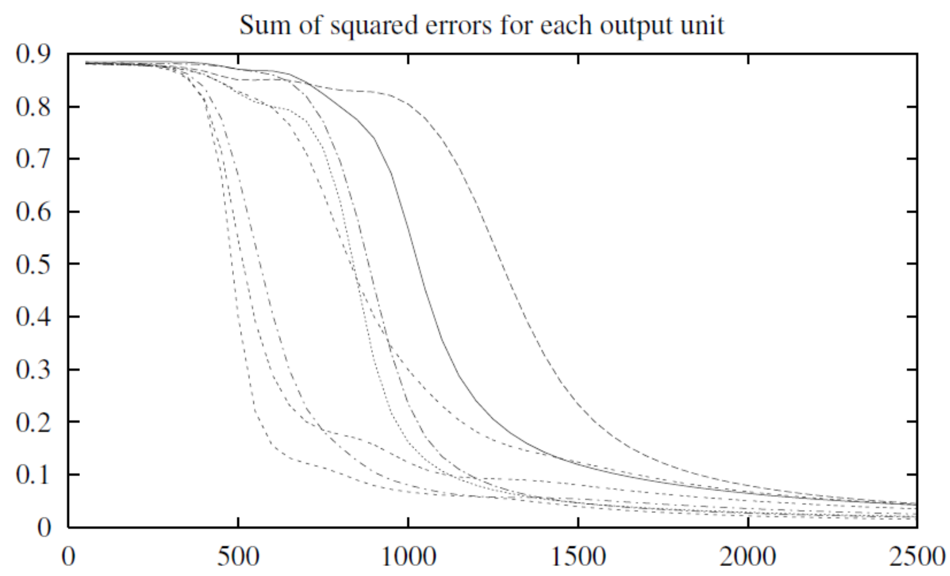
- And a target function:



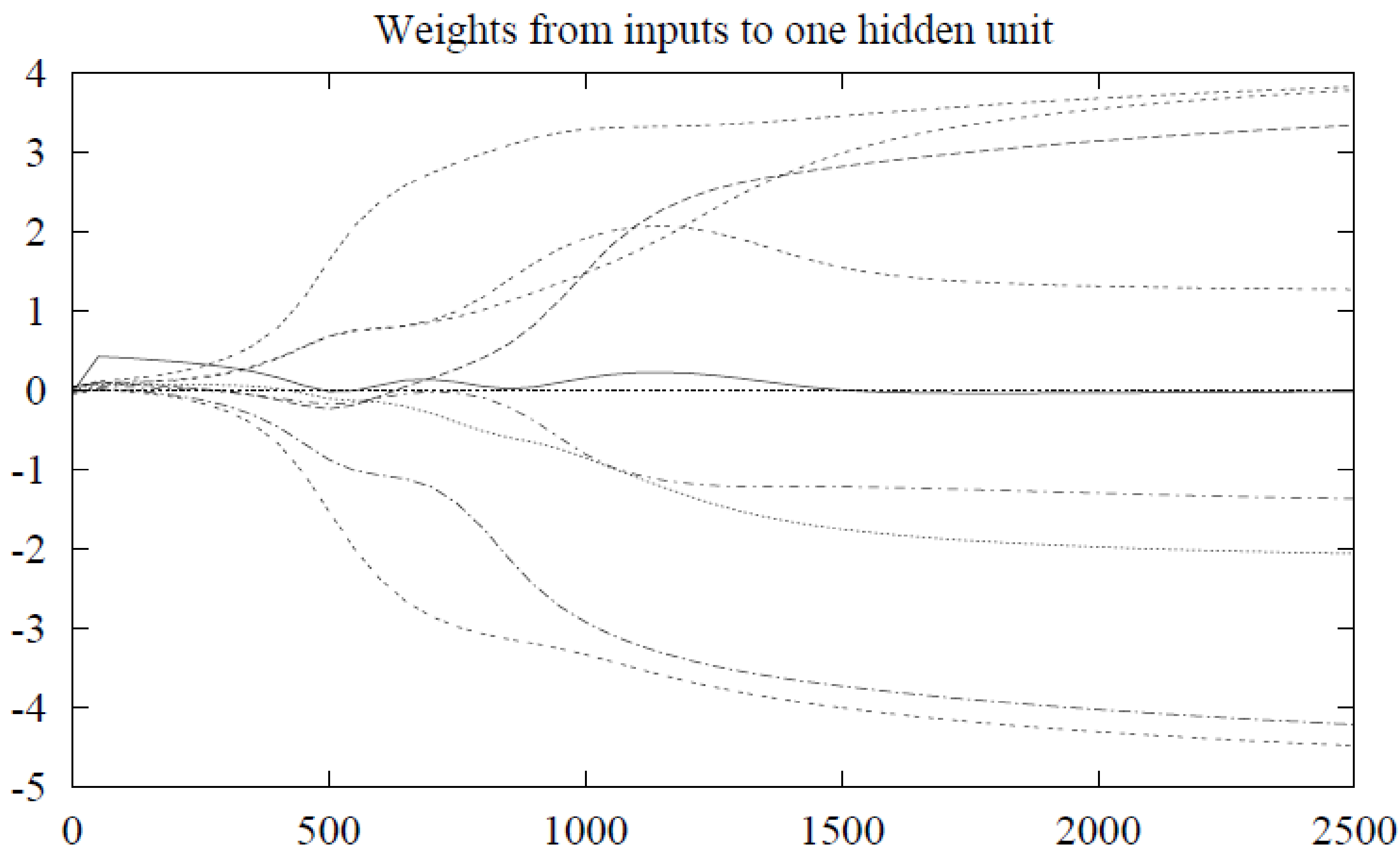
Example from T. Michell's Book "Machine Learning"

Encoding for input 0100000

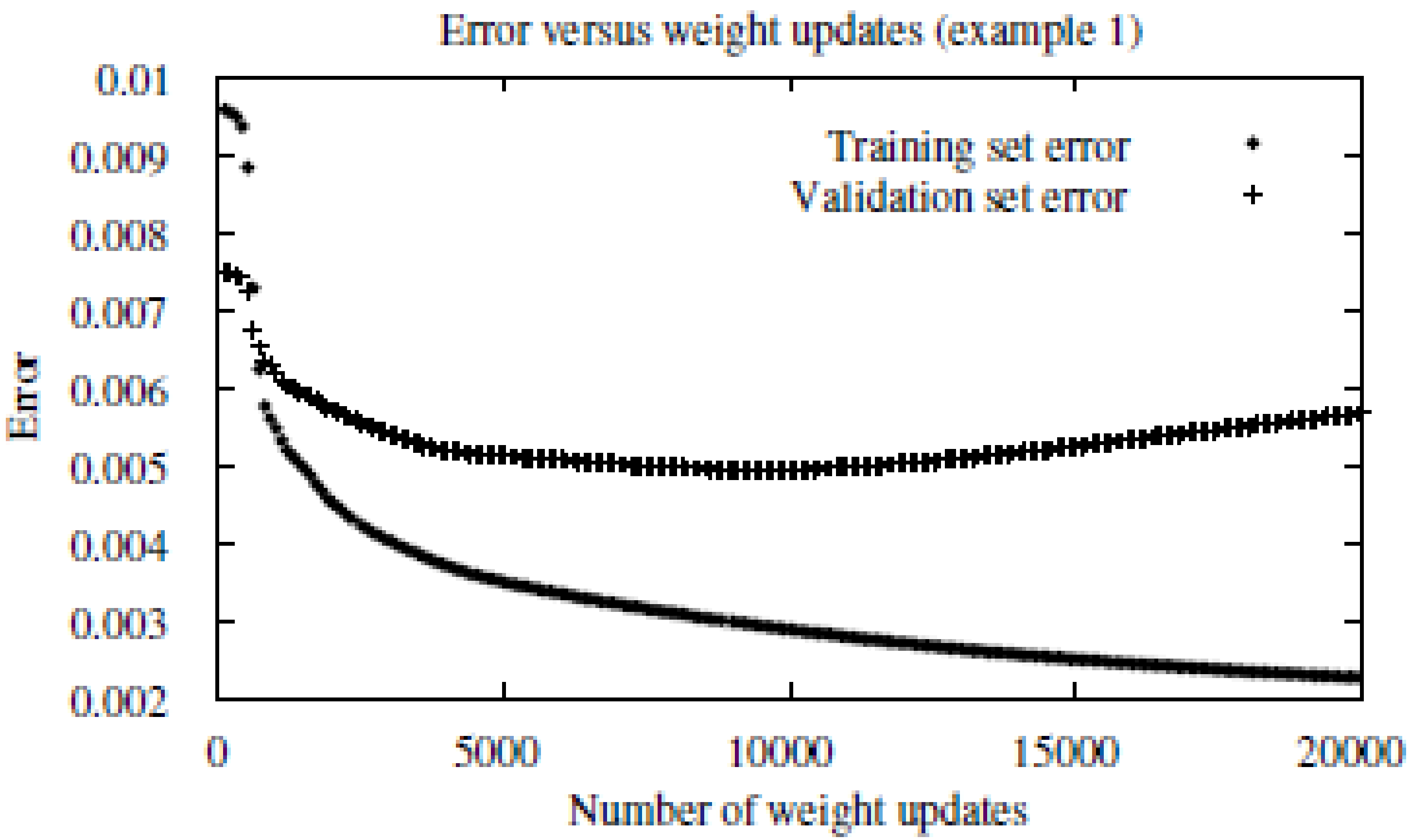




Weights from inputs to one hidden unit



Don’t forget overfitting

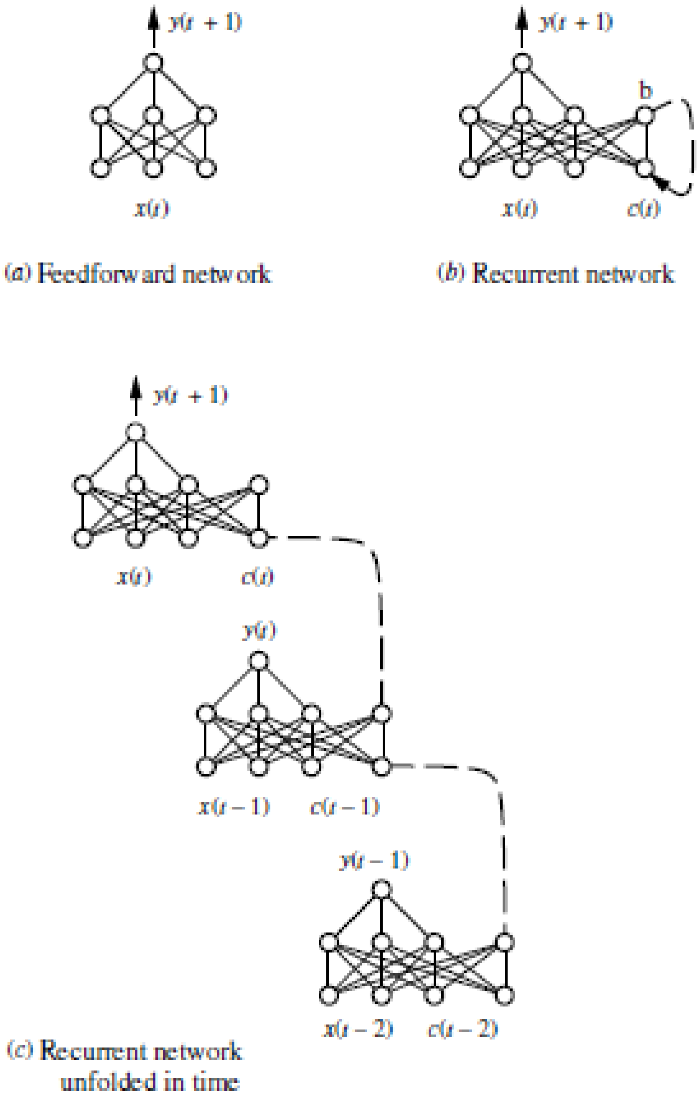


Words of wisdom for ANN

- Stochastic gradient descent is your best friend when trying to find the optimal parameter set.
  - Careful though, in contrast to linear regression, it can get stuck to a local minimum or converge very slowly
- How to initialize the weights?
  - Embrace zero: Network initially simple and progressively can become complex/non-linear
- What can be represented in a ANN ? All continuous and Boolean functions!!
  - A NN with at least 1 hidden layer can approximate every bounded continuous function (Cybenko 1989)
  - A NN with at least 2 hidden layers can approximate any function to arbitrary accuracy (universal approximation property, Cybenko 1988)
- When to add hidden units?
  - When output error doesn't decrease
  - When a small fraction of the hidden units are activated by the input or a specific training pattern cannot be learned
- When a new hidden unit is added you can retrain the entire net or the new weights only

Other Artificial Neural Networks

- Next time we will look at how to construct and train a ANN in detail.
- And it is not just FFNN..



(a) Feedforward network

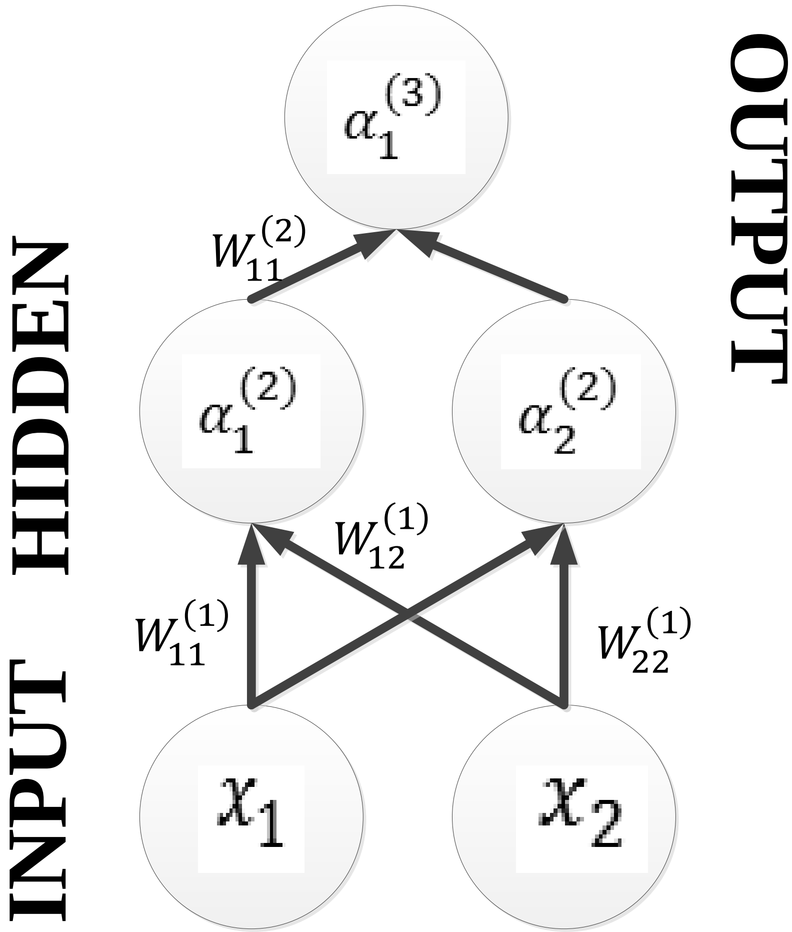
(b) Recurrent network

(c) Recurrent network unfolded in time

Some notation that we will use for next time

$$\alpha_i^{(j)} = \textit{Activation of unit i in layer j}$$

$$W^{(j)} = \textit{weight matrix from layer j to } j+1$$



$$\boldsymbol{\alpha_1^{(3)} = g(W_{10}^{(2)} x_0 + W_{11}^{(2)} \alpha_1^{(2)} + W_{12}^{(2)} \alpha_2^{(2)})}$$

$$\alpha_1^{(2)} = g(W_{10}^{(1)} x_0 + W_{11}^{(1)} x_1 + W_{12}^{(1)} x_2)$$

$$\alpha_2^{(2)} = g(W_{20}^{(1)} x_0 + W_{21}^{(1)} x_1 + W_{22}^{(1)} x_2)$$

End of Lecture 5