Deep Learning in Biomedical Informatics

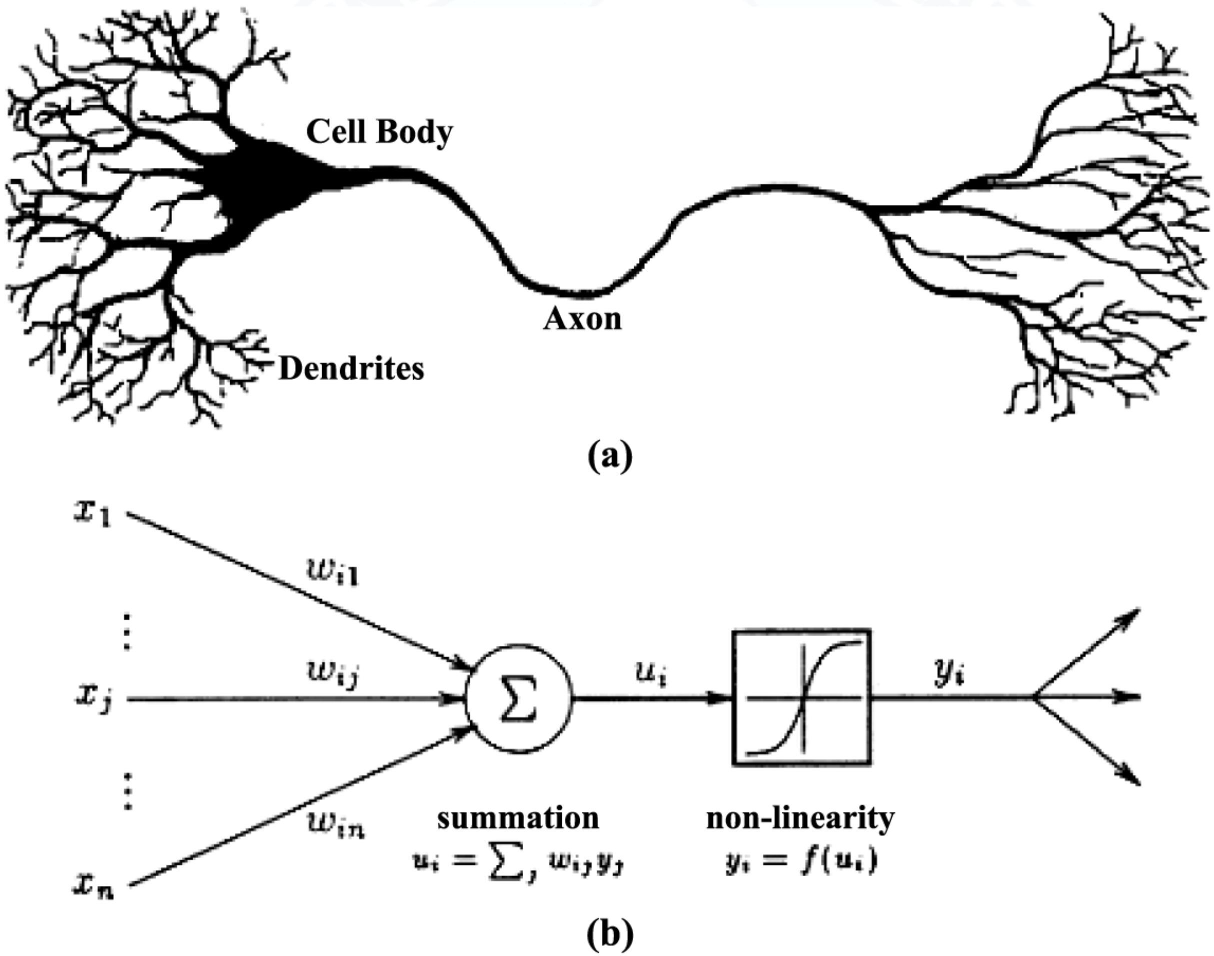
Dr. Fayyaz ul Amir Afsar Minhas
PIEAS Biomedical Informatics Research Lab
Department of Computer and Information Sciences
Pakistan Institute of Engineering & Applied Sciences
PO Nilore, Islamabad, Pakistan
http://faculty.pieas.edu.pk/fayyaz/

Lecture Plan

- What are Neural Networks?
- What is Deep Learning?
- Why go Deep?
- What are the different models for deep learning
  - CNN
  - Transfer Learning
  - Representation Learning (Auto-encoders)
  - RNN/LSTM
  - ResNet
  - GANs
- Non-Neural Deep Learning
  - Multilayer Kernel Machines
- Applications

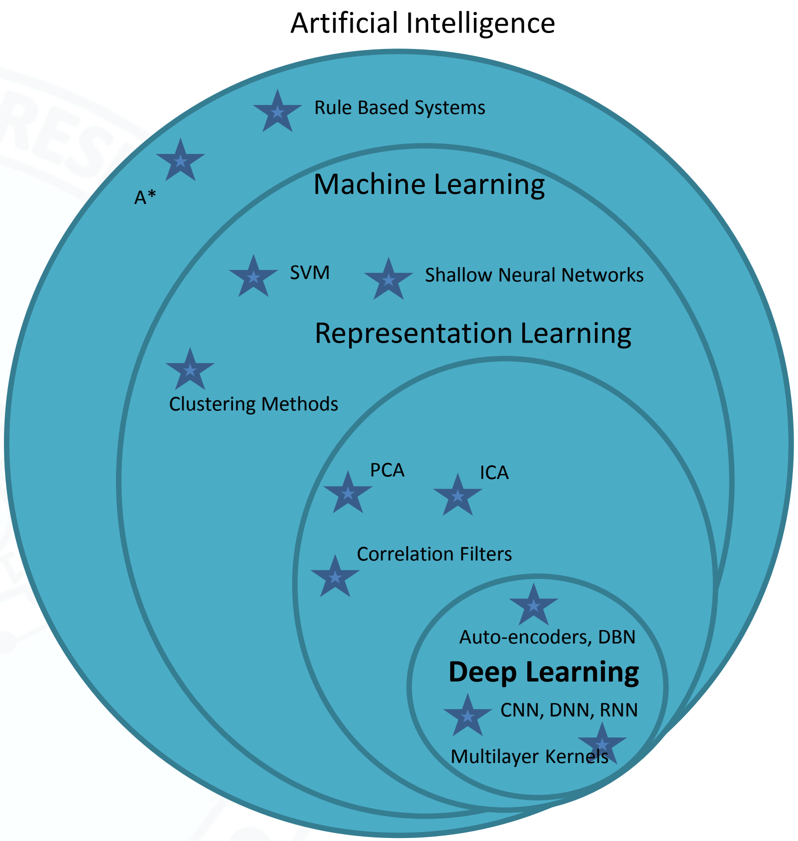
Neural Networks

- An abstraction of the biological neuron



Deep Learning

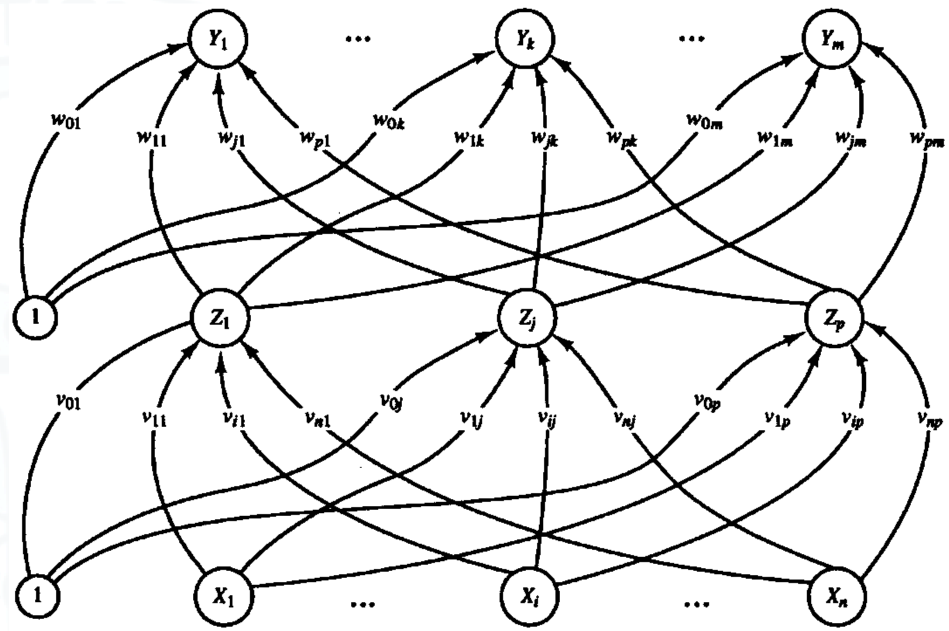
- Traditional machine learning focuses on feature engineering
- Deep learning is a branch of machine learning
  - That uses a cascade of many layers of non-linear units for feature extraction and transformation
  - Based on "automatic" learning of multiple levels of features or representations of the data
- Re-branding of neural networks!
  - Massive growth in efficient algorithms for solving AI challenges!
- Many Applications in Biomedical Informatics



AlphaGo

Multilayer Perceptron

- Consists of multiple layers of neurons
- Layers of units other than the input and output are called hidden units
- Unidirectional weight connections and biases
- Activation functions
  - Use of activation functions
    - Sigmoidal activations
    - Nonlinear Operation: Ability to solve practical problems
    - Differentiable: Makes theoretical assessment easier
    - Derivative can be expressed in terms of functions themselves: Computational Efficiency
  - Activation function is the same for all neurons in the same layer
  - Input layer just passes on the signal without processing (linear operation)



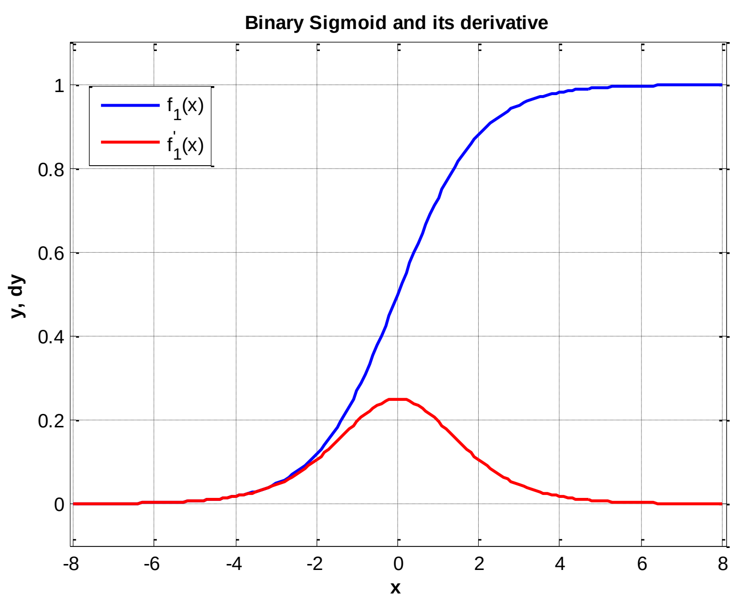
$$z_j = f\left(z\_in_j\right)$$

$$z\_in_j = \sum_{i=0}^{n} x_i v_{ij}, \qquad x_0 = 1, \qquad j = 1 \ldots p$$

$$y_k = f\left(y\_in_k\right)$$

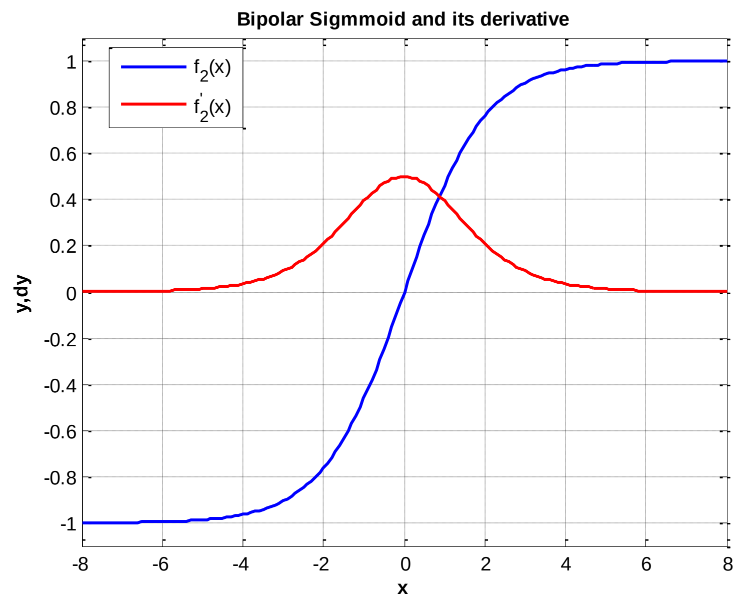
$$y\_in_k = \sum_{j=0}^{p} z_j w_{jk}, \qquad z_0 = 1, \qquad k = 1 \ldots m$$

Architecture: Activation functions



$$f_1(x) = \frac{1}{1 + \exp(-x)}$$

$$f'_1(x) = f_1(x)[1 - f_1(x)]$$

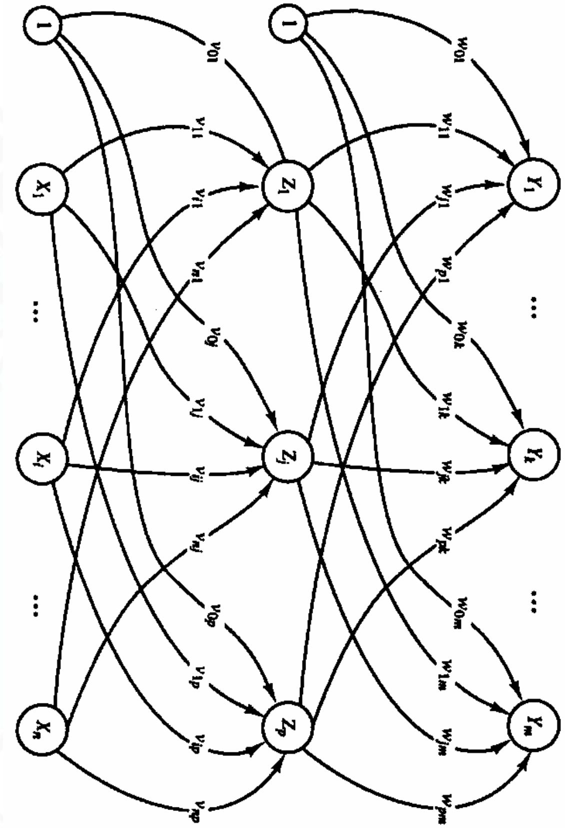


$$f_2(x) = \frac{2}{1 + \exp(-x)} - 1.$$

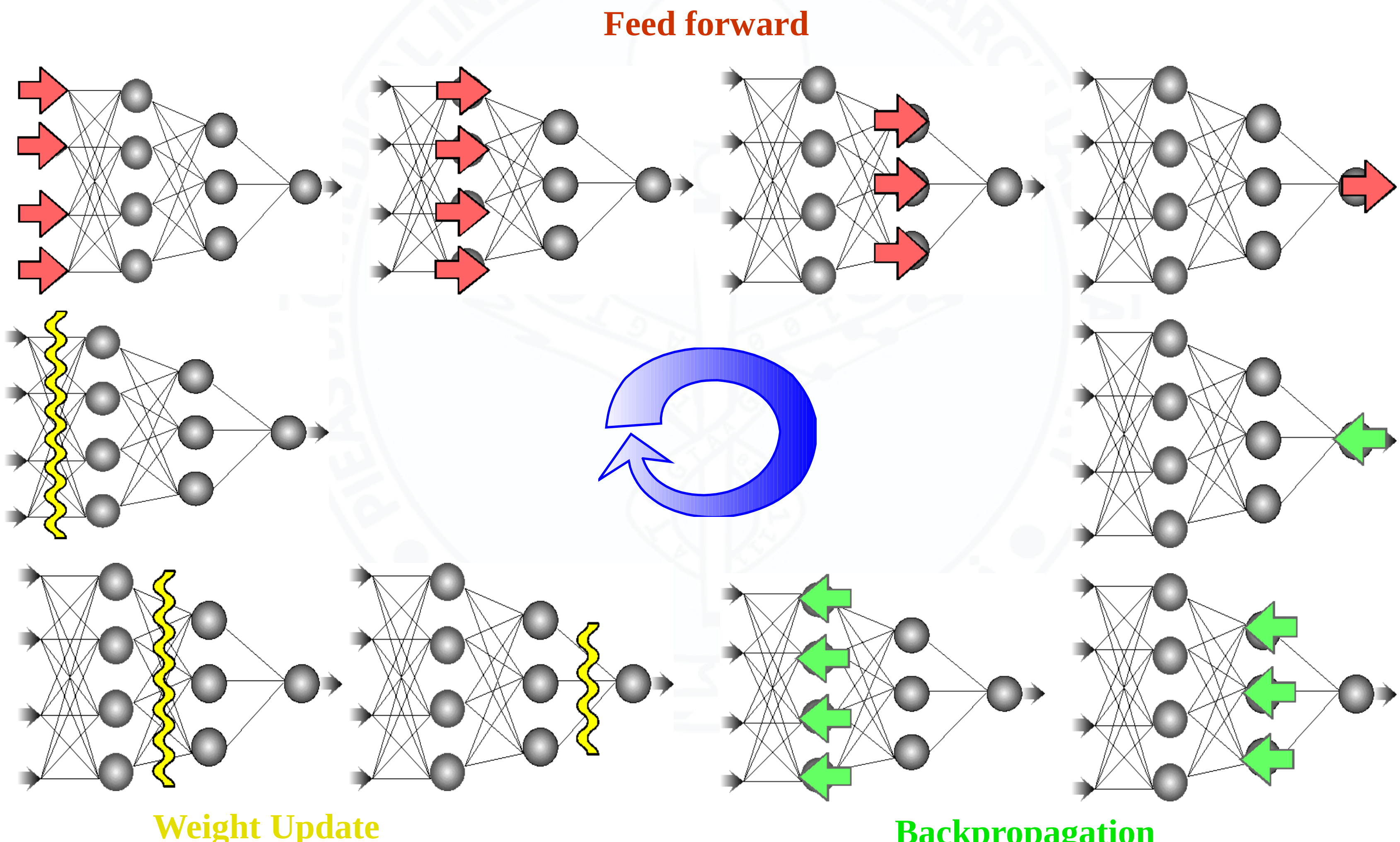
$$f'_2(x) = \frac{1}{2}[1 + f_2(x)][1 - f_2(x)].$$

Training

- During training we are presented with input patterns and their targets
- At the output layer we can compute the error between the targets and actual output and use it to compute weight updates through the Delta Rule
- But the Error cannot be calculated at the hidden input as their targets are not known
- Therefore we propagate the error at the output units to the hidden units to find the required weight changes (Backpropagation)
- 3 Stages
  - Feed-forward of the input training pattern
  - Calculation and Backpropagation of the associated error
  - Weight Adjustment
- Based on minimization of SSE (Sum of Square Errors)



Backpropagation training cycle



Proof for the Learning Rule

$$E = .5 \sum_k [t_k - y_k]^2.$$

By use of the chain rule, we have

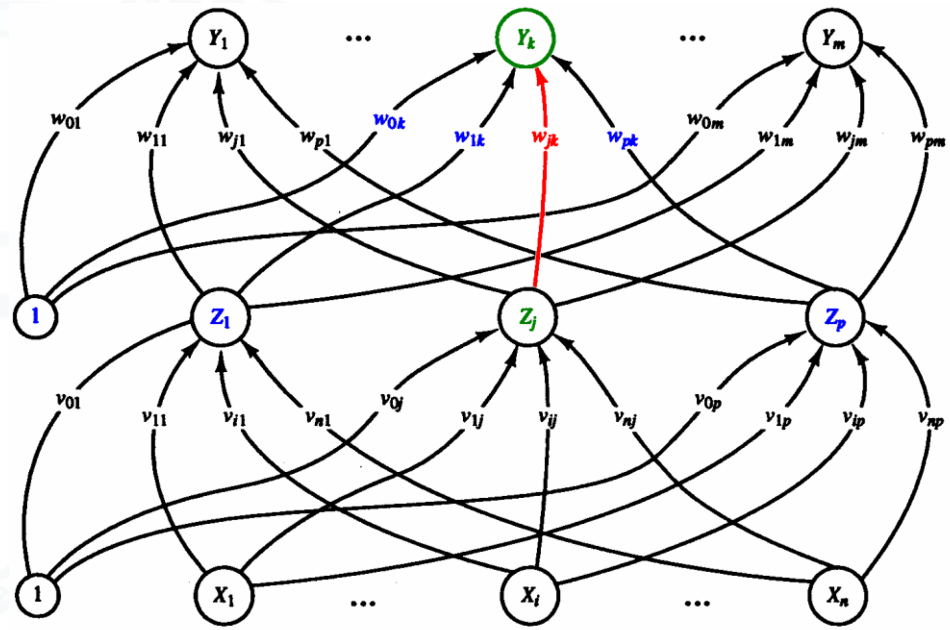
$$\frac{\partial E}{\partial w_{JK}} = \frac{\partial}{\partial w_{JK}} .5 \sum_k [t_k - y_k]^2$$

$$= \frac{\partial}{\partial w_{JK}} .5[t_K - f(y\_in_K)]^2$$

$$= -[t_K - y_K] \frac{\partial}{\partial w_{JK}} f(y\_in_K)$$

$$= -[t_K - y_K] f'(y\_in_K) \frac{\partial}{\partial w_{JK}} (y\_in_K)$$

$$= -[t_K - y_K] f'(y\_in_K) z_J.$$



Change in w_{jk} affects only Y_k

It is convenient to define δ_K:

$$\delta_K = [t_K - y_K] f'(y\_in_K).$$

$$\Delta w_{jk} = -\alpha \frac{\partial E}{\partial w_{jk}}$$

Use of Gradient Descent Minimization

$$= \alpha[t_k - y_k] f'(y\_in_k) z_j$$

$$= \alpha \delta_k z_j;$$

Proof for the Learning Rule…

For weights on connections to the hidden unit Z_J:

$$\frac{\partial E}{\partial v_{IJ}} = -\sum_k [t_k - y_k] \frac{\partial}{\partial v_{IJ}} y_k$$

$$= -\sum_k [t_k - y_k] f'(y\_in_k) \frac{\partial}{\partial v_{IJ}} y\_in_k$$

$$= -\sum_k \delta_k \frac{\partial}{\partial v_{IJ}} y\_in_k$$

$$= -\sum_k \delta_k w_{Jk} \frac{\partial}{\partial v_{IJ}} z_J$$

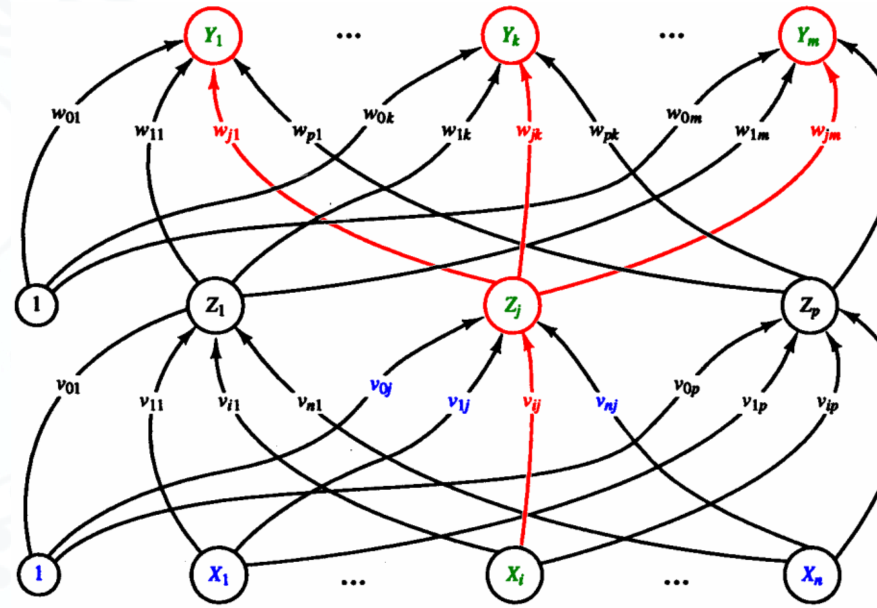
$$= -\sum_k \delta_k w_{Jk} f'(z\_in_J)[x_I].$$

Define:

$$\Delta v_{ij} = -\alpha \frac{\partial E}{\partial v_{ij}}$$

$$= \alpha f'(z\_in_j) x_i \sum_k \delta_k w_{jk},$$

$$= \alpha \delta_j x_i.$$



Change in v_{ij} affects all $Y_{1..m}$

Change in v_{ij} affects only z_j

Use of Gradient Descent Minimization

Training Algorithm

Step 0. Initialize weights.
(Set to small random values).

Step 1. While stopping condition is false, do Steps 2–9.

Step 2. For each training pair, do Steps 3–8.

Feedforward:

Step 3. Each input unit (X_i, $i = 1, \ldots, n$) receives input signal x_i and broadcasts this signal to all units in the layer above (the hidden units).

Step 4. Each hidden unit (Z_j, $j = 1, \ldots, p$) sums its weighted input signals,

$$z\_in_j = v_{0j} + \sum_{i=1}^{n} x_i v_{ij},$$

applies its activation function to compute its output signal,

$$z_j = f(z\_in_j),$$

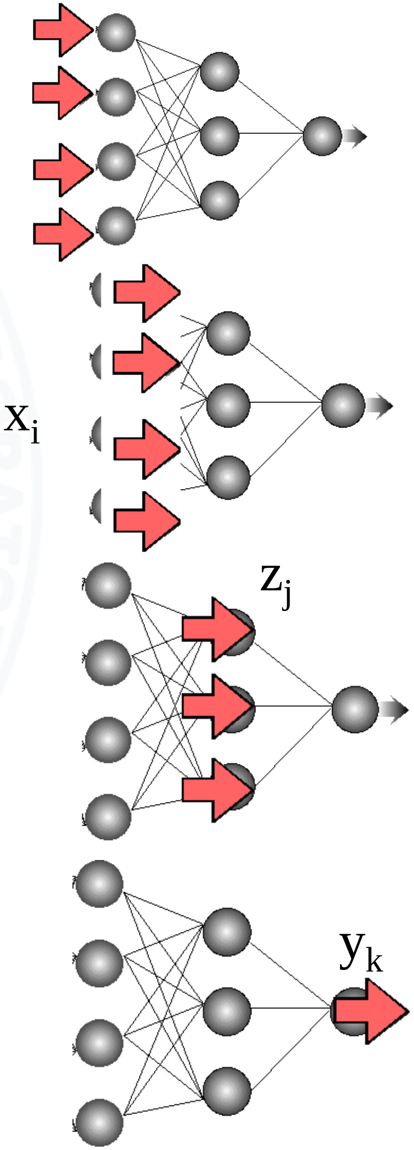
and sends this signal to all units in the layer above (output units).

Step 5. Each output unit (Y_k, $k = 1, \ldots, m$) sums its weighted input signals,

$$y\_in_k = w_{0k} + \sum_{j=1}^{p} z_j w_{jk}$$

and applies its activation function to compute its output signal,

$$y_k = f(y\_in_k).$$



Training Algorithm…

Backpropagation of error:

Step 6. Each output unit ($Y_k, k = 1, \ldots, m$) receives a target pattern corresponding to the input training pattern, computes its error information term,

$$\delta_k = (t_k - y_k)f'(y\_in_k),$$

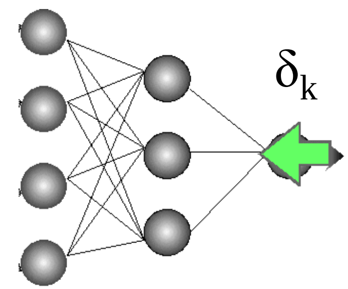
calculates its weight correction term (used to update w_{jk} later),

$$\Delta w_{jk} = \alpha \delta_k z_j,$$

calculates its bias correction term (used to update w_{0k} later),

$$\Delta w_{0k} = \alpha \delta_k,$$

and sends δ_k to units in the layer below.



Training Algorithm…

Step 7. Each hidden unit ($Z_j, j = 1, \ldots, p$) sums its delta inputs (from units in the layer above),

$$\delta\_in_j = \sum_{k=1}^{m} \delta_k w_{jk},$$

multiplies by the derivative of its activation function to calculate its error information term,

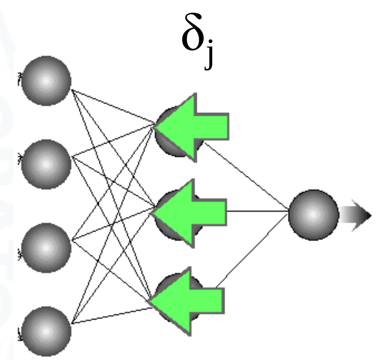
$$\delta_j = \delta\_in_j \, f'(z\_in_j),$$

calculates its weight correction term (used to update v_{ij} later),

$$\Delta v_{ij} = \alpha \delta_j x_i,$$

and calculates its bias correction term (used to update v_{0j} later),

$$\Delta v_{0j} = \alpha \delta_j.$$



Training Algorithm…

Update weights and biases:

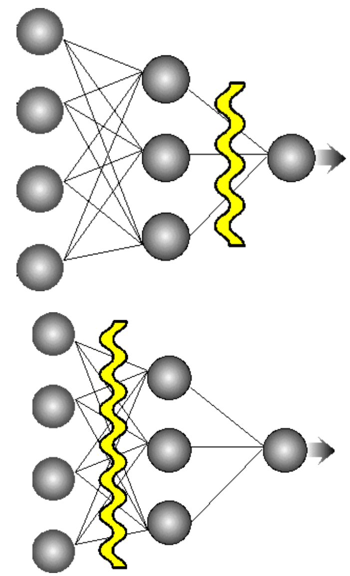
Step 8. Each output unit (Y_k, $k = 1, \ldots, m$) updates its bias and weights ($j = 0, \ldots, p$):

$$w_{jk}(\text{new}) = w_{jk}(\text{old}) + \Delta w_{jk}.$$

Each hidden unit (Z_j, $j = 1, \ldots, p$) updates its bias and weights ($i = 0, \ldots, n$):

$$v_{ij}(\text{new}) = v_{ij}(\text{old}) + \Delta v_{ij}.$$

Step 9. Test stopping condition.



Optimization in minibatches

- We can do a full scale optimization across all examples or take a few examples at a time to determine the gradients
  - Mini-batches

Things to note

- A large number of derivatives will be computed
  - For every input
  - For every weight at every layer
- The update is dependent upon
    - The activation function value
    - The input
    - The target
    - The current weight value
    - The value of the derivative of the activation function of the current layer
    - The value of the derivative of the activation function of the following layers
    - The derivatives are multiplied
    - The error value

Parameter Selection

- A MLP has a large number of parameters
  - Number of Neurons in Each Layer
  - Number of Layers
  - Activation Function for each neuron: ReLU, logsig…
  - Layer Connectivity: Dense, Dropout…
  - Objective function
    - Loss Function: MSE, Entropy, Hinge loss, …
    - Regularization: L1, L2…
  - Optimization Method
    - SGD, ADAM, RMSProp, LM …
  - Parameters for the Optimization method
    - Weight initialization
    - Momentum, weight decay, etc.