Introduction to Neural Networks

Part II : Learning of MLP

Web site of this course: http://pattern-recognition.weebly.com

Two Parts

Part I : Neural information processing

- Origins
- Perceptron
- Multilayer perceptron (MLP)
- Convolutional networks (CNN)

Part II : Learning of MLP

- **An example of backpropagation learning**
- **Learning algorithms**
- **Optimization and learning**

Learning of MLP Network

An example of learning

Learning algorithms

Optimization theory

Source:

http://www.existor.com/en/news-neural-networks.html

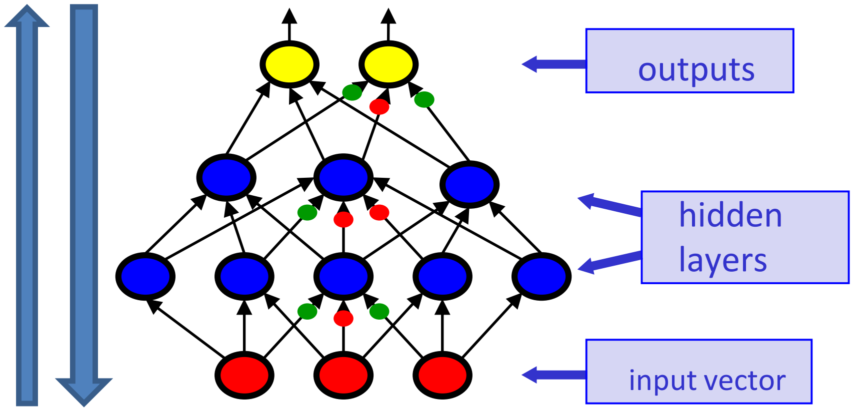
Training the MLP: Backpropagation

Testing for K-class classification problem

- For a given x with unknown class
- $x \in$ class k, $if\ y_k = max_i y_i$
- $y_i = v_i^T z = \sum_{h=1}^{H} v_{ih} z_h + v_{i0}$

That is

- A w represents a MLP
- Given a w, then we can classify a pattern x



$$w = [w_1, \cdots, w_K, v_1, \cdots, v_H]$$

A Machine Learning problem:

how to obtain the w of a MLP

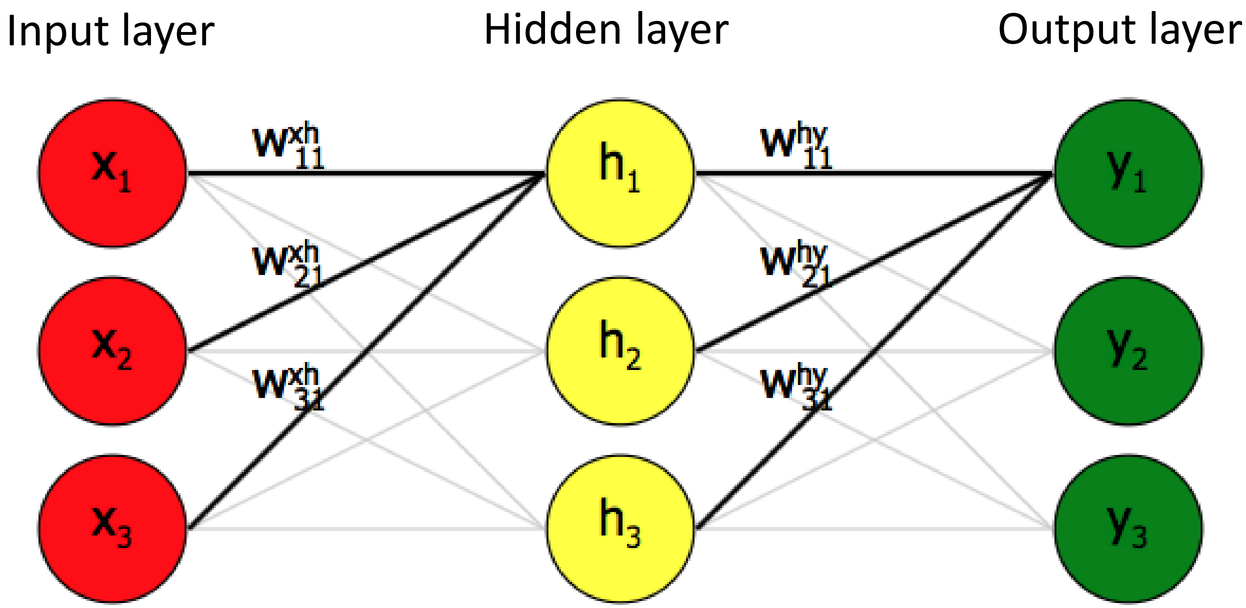
- We need a set of training patterns (x,y)
- We need a learning algorithm to learn w by (x,y)

=> Backpropagation learning algorithm B: $w=B(x,y)$

Backpropagation

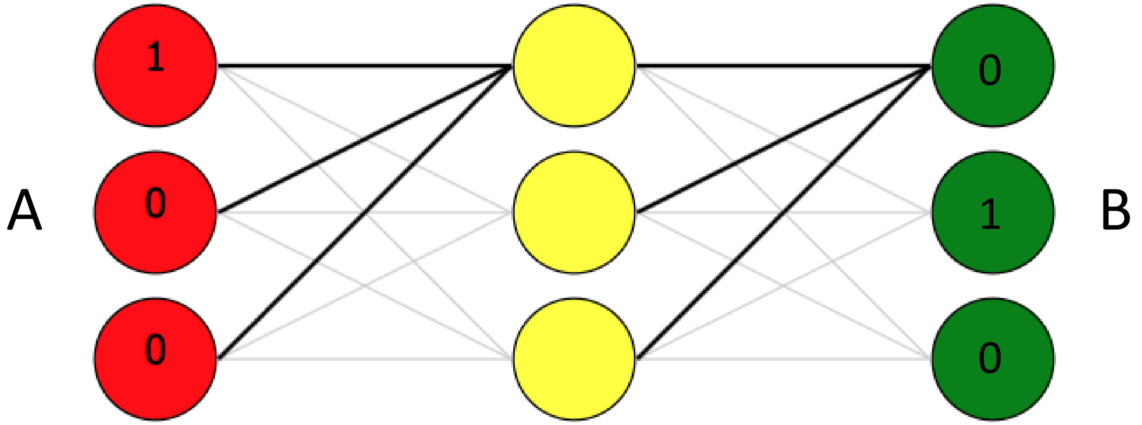
A multilayer neural network

- A three-layer network: one hidden layer
  - 9 nodes(x_i, h_j, y_k), 6 neurons(h_j, y_k)
  - 18 weights(w)



Example problem: Convert letters A,B,C

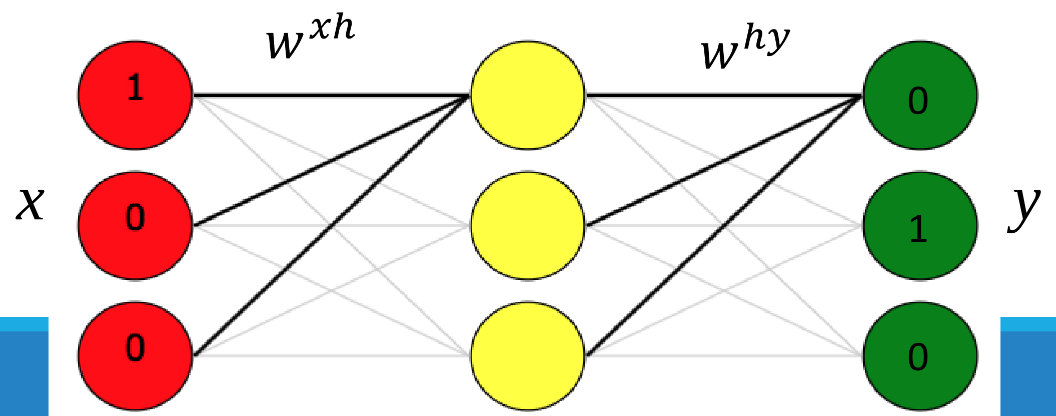
- Input: 1-of-K binary encoding
  - Letters are encoded into binary: A - 100, B - 010, C - 001
- Output
  - Convert A to B, B to C, C to A
  - 100 -> 010, 010 -> 001, 001 -> 100



Training of the network

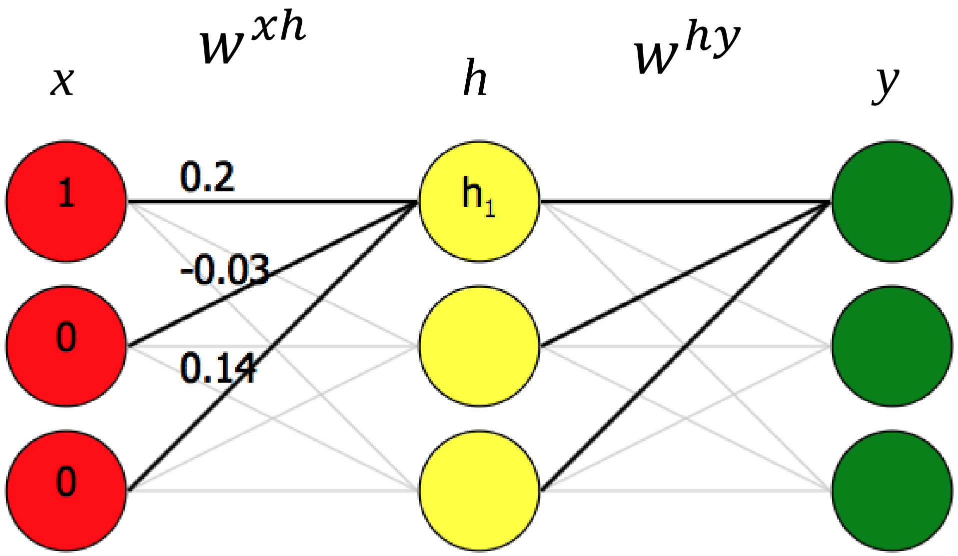
- Given a training pair (x,y)
  - x: input values,   y: desired output values
- Network training will get a weight matrix $w=(w^{xh},w^{hy})$
- Basic steps to train the network
  1. Randomly initialize the weight matrix w
  2. Forward propagation: $y'=xw$
  3. Compute the error: $E=y-y'$
  4. Compute weight change value by the error: $\Delta w=f(E)$
  5. Backpropagation: $w = w - \Delta w$
  6. Go to step 2

supervised learning



Step 1: Random starting weights

- Now we will compute the values of the first hidden node h_1 in the second layer
- The weights are usually initialised to be small random values between -1 and 1

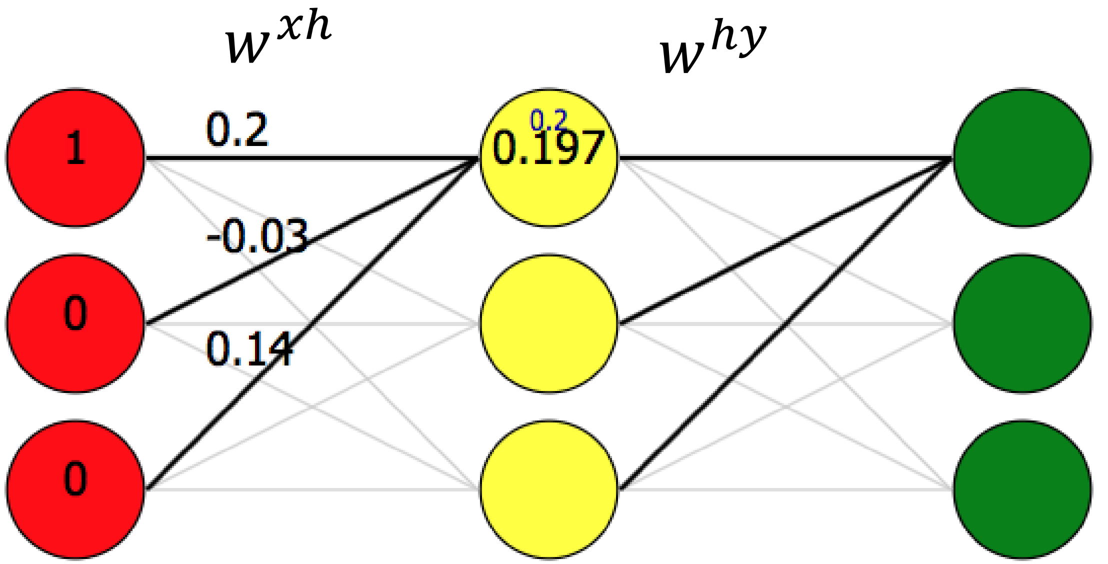


Step 2: Forward propagation
Weighted sum

- Z_{h1} represents the weighted sum of the node h_1

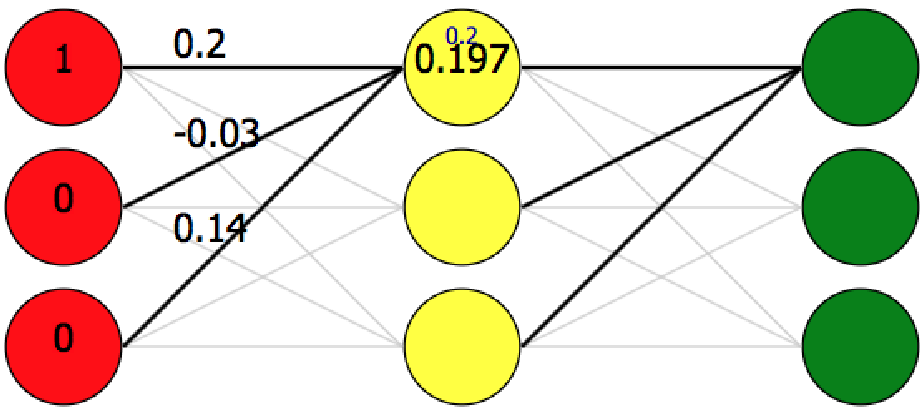
$$z_{h1} = x_1 w_{11}^{xh} + x_2 w_{21}^{xh} + x_3 w_{31}^{xh} = 1 * 0.2 + 0 * -0.03 + 0 * 0.14 = 0.2$$

$$z_{h1} = \sum_{i=1}^{3} x_i w_{i1}^{xh}$$



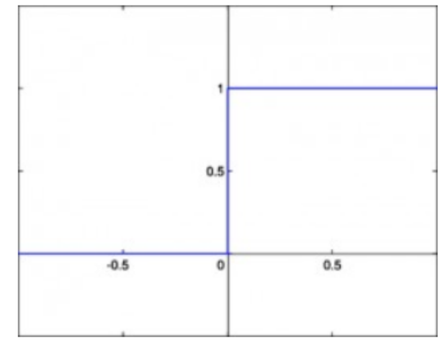
Step 2: Forward propagation
Activation of weighted sum

- Assume we use bipolar sigmoid
- $h_1 = sigmoid(z_{h1}) = sigmoid(0.2) \approx 0.197$

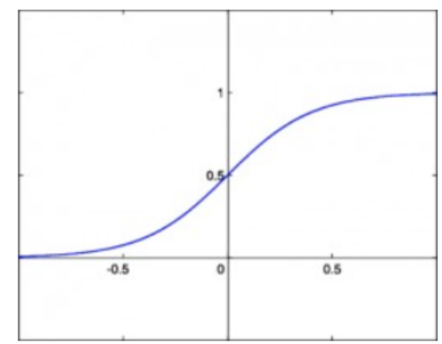


$$h_1 = f(z_{h1}) = \frac{1}{1 + e^{-z_{h1}}}$$

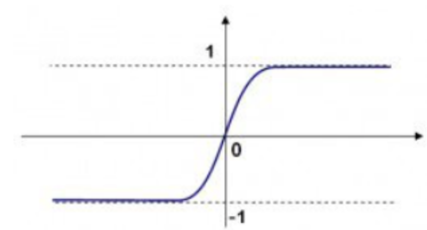
$$h_1 = sigmoid(z_{h1}) = 2*(f(z_{h1}) -0.5)$$



Binary Step Function



Binary Sigmoid Function



Bipolar Sigmoid function

Step 2: Forward propagation

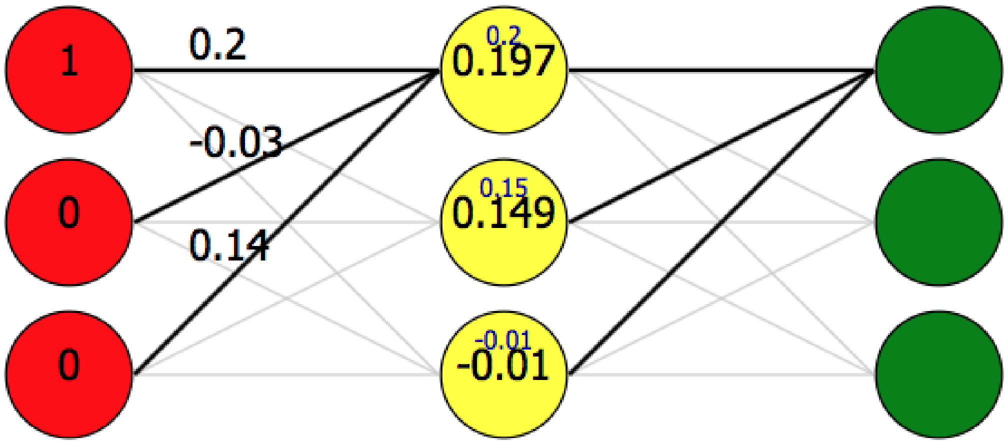
Matrix notation

$$x = [x_1 \quad x_2 \quad x_3] = [1 \quad 0 \quad 0]$$

$$h_j = sigmoid\left(z_{h_j}\right) = sigmoid\left(\sum_{i=1}^{3} x_i w_{ij}^{xh}\right) \qquad w^{xh} = \begin{bmatrix} 0.2 & 0.15 & -0.01 \\ -0.03 & -0.1 & -0.06 \\ 0.14 & -0.2 & 0.03 \end{bmatrix}$$

$$z_h = xw^{xh} = [1 \quad 0 \quad 0] \begin{bmatrix} 0.2 & 0.15 & -0.01 \\ -0.03 & -0.1 & -0.06 \\ 0.14 & -0.2 & 0.03 \end{bmatrix} = [0.2 \quad 0.15 \quad -0.01]$$

$$\begin{aligned} h &= sigmoid(z_h) \\ &= sigmoid([0.2 \quad 0.15 \quad -0.01]) \\ &= [0.197 \quad 0.149 \quad -0.01] \end{aligned}$$



Step 2: Forward propagation

Output layer

- Assume w^{hy} are the weights between hidden and output layers

$$w^{hy} = \begin{bmatrix} 0.08 & 0.11 & -0.3 \\ 0.1 & -0.15 & 0.08 \\ 0.1 & 0.1 & -0.07 \end{bmatrix}$$

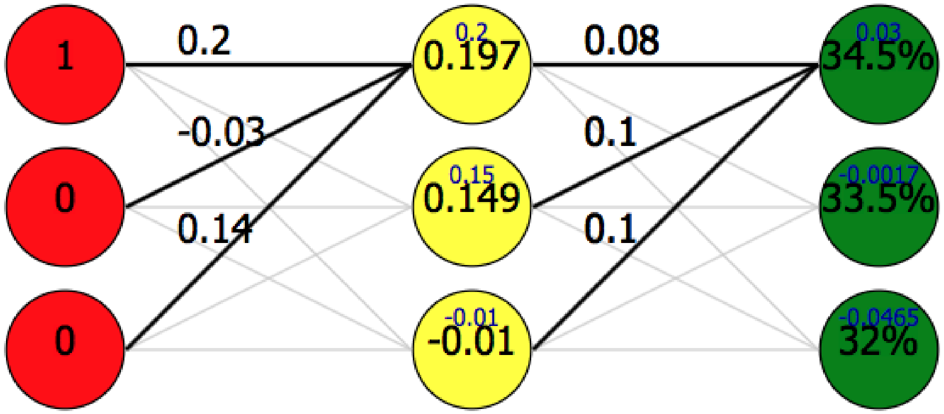
$$z_y = hw^{hy}$$

$$= \begin{bmatrix} 0.197 & 0.149 & -0.01 \end{bmatrix} \begin{bmatrix} 0.08 & 0.11 & -0.3 \\ 0.1 & -0.15 & 0.08 \\ 0.1 & 0.1 & -0.07 \end{bmatrix} = \begin{bmatrix} 0.03 & -0.0017 & -0.0465 \end{bmatrix}$$

~~$y_k = sigmoid(z_{y_k})$~~

~~$= sigmoid\left(\sum_{j=1}^{3} h_i w_{jks}^{hy}\right)$~~

We usually use softmax function for output nodes, but not sigmoid. See next slide.

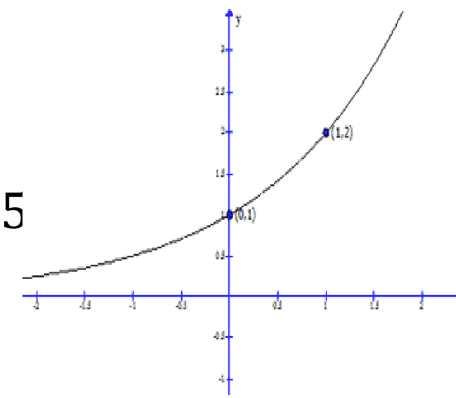


Step 2: Forward propagation

Output layer

- The softmax function $p_k = \dfrac{e^{z_{y_k}}}{\sum_{k=1}^{3} e^{z_{y_k}}}$

e^x

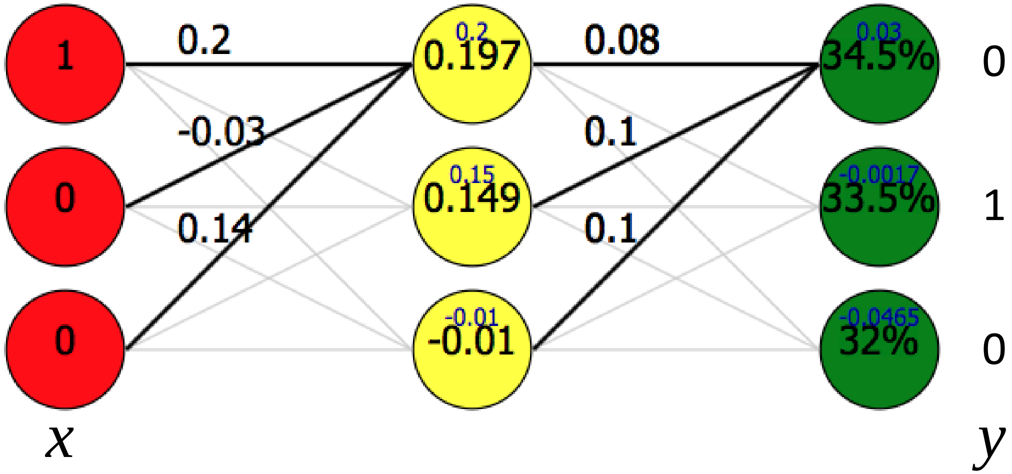


$$p = softmax(z_y) = softmax([0.03 \quad -0.0017 \quad -0.0465$$
$$= [0.345 \quad 0.335 \quad 0.32]$$

$$y' = [1\ 0\ 0]$$

$$y'_k = \begin{cases} 1, p_k \ is\ the\ \max(p_i) \\ 0, otherwise \end{cases}$$

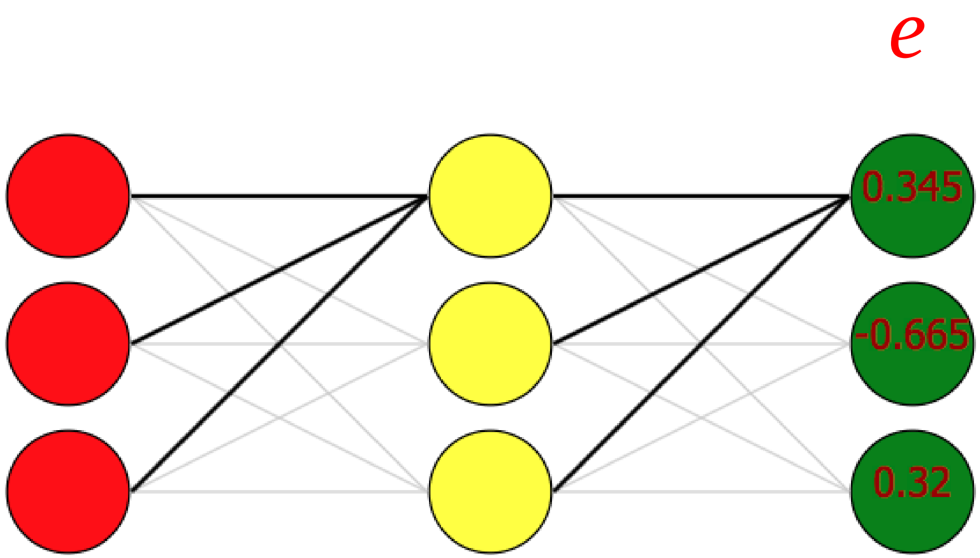
The random *w* gets a wrong output



Step 3: Computing output error

$$y = [0 \quad 1 \quad 0], p \; = [0.345 \quad 0.335 \quad 0.32]$$

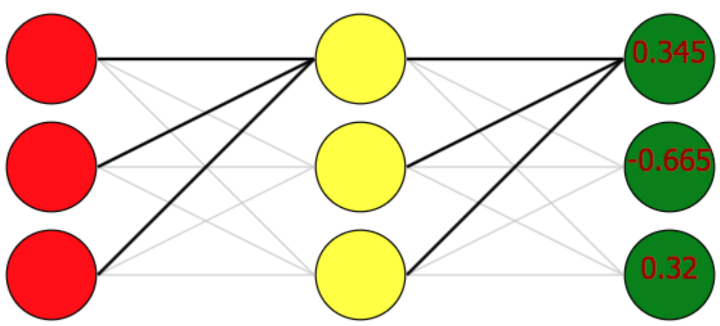
$$e = p - y = [0.345 \quad 0.335 \quad 0.32]\text{-}[0 \quad 1 \quad 0]$$
$$= [0.345 \quad -0.665 \quad 0.32]$$



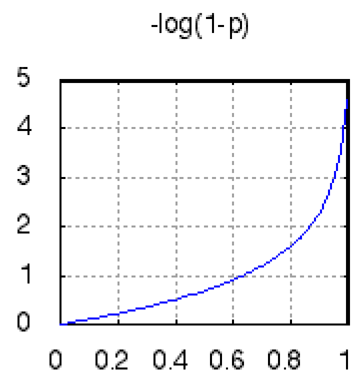
Step 3: Computing output error

Loss & cross entropy

- We need to calculate the total error for all the outputs combined. This is called the loss or cost of the network and is labelled with J.
- Three possible J
  - Absolute error $J = \sum_{k=1}^{3} |e_k| = 0.345 + 0.665 + 0.32 = 1.32$
  - Squared error $J = \sum_{k=1}^{3} e_k^2 = 0.664$
  - Cross entropy $J = -\sum_{k=1}^{3} y_k log p_k = -0 - 1 * \log(0.335) - 0 = 1.0936$

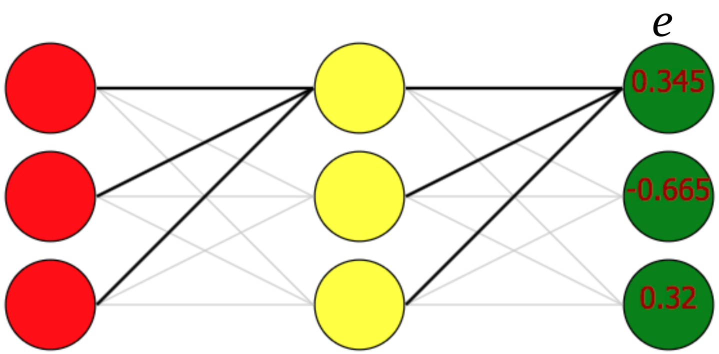


e	y	p
0.345	0	0.345
-0.665	1	0.335
0.32	0	0.32



Step 4: Adjusting weights
Intuition

- It feels like
  - The weights going into y_1 and y_3 should be lowered a bit, because their estimate was too high.
  - The weights going into y_2 should be raised because they were way too low and caused a large negative error.
  - The bigger the error, the more the weights should be changed.

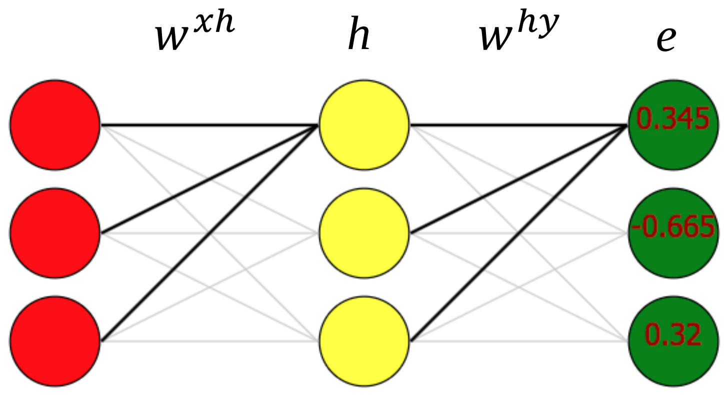


Step 4: Adjusting weights
Formula

- Mathematically the intuition is fairly easy to do.
- The error δw of the weight w
  - is proportional to the size of the thing on the other end of the connection (the activation value of the hidden node). a
  - So we can just multiply the value of the hidden node h_j times the error e_k to get δw_{jk}^{hy}

$$w_{jk}^{hy} = w_{jk}^{hy} - \delta w_{jk}^{hy}$$

$$\delta w_{jk}^{hy} \propto h_j * e_k$$



Step 4: Adjusting weights

An example

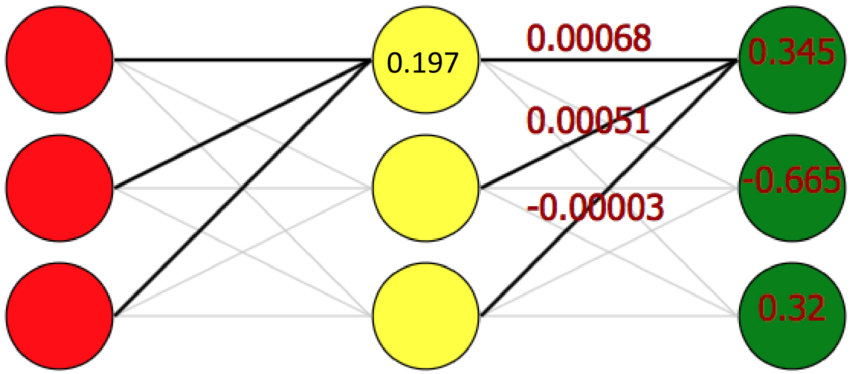
- Assume a learning rate α = 0.01

$$\delta w_{jk}^{hy} = \alpha * h_j * e_k \propto h_j * e_k$$

- For example, the adjustment on the top weight connecting the first hidden node to the first output node, δw_{11}^{hy}, could just be:

$$\delta w_{11}^{hy} = \alpha * h_j * e_k = 0.01*0.197*0.345 = 0.00068$$

$$w_{11}^{hy} = w_{11}^{hy} - \delta w_{11}^{hy}$$
$$= 0.08 - 0.00068$$
$$= 0.07932$$

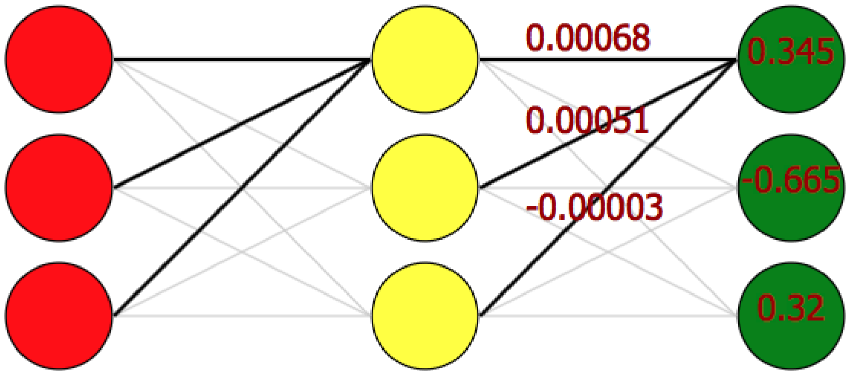


Step 4: Adjusting weights
Matrix

- We can compute all the adjustments δw^{hy} with one matrix operation. Assume a learning rate α = 0.01

$$\delta w^{hy} = \alpha h^T e = 0.01 \begin{bmatrix} 0.197 \\ 0.149 \\ -0.01 \end{bmatrix} \begin{bmatrix} 0.345 & -0.665 & 0.32 \end{bmatrix}$$

$$= \begin{bmatrix} 0.00068 & -0.00131 & 0.00063 \\ 0.00051 & -0.00099 & 0.00047 \\ -0.00003 & 0.00007 & -0.00003 \end{bmatrix}$$



Step 4: Adjusting weights

Theory

- Why the formula?

$$w_{jk}^{hy} = w_{jk}^{hy} - \delta w_{jk}^{hy}$$

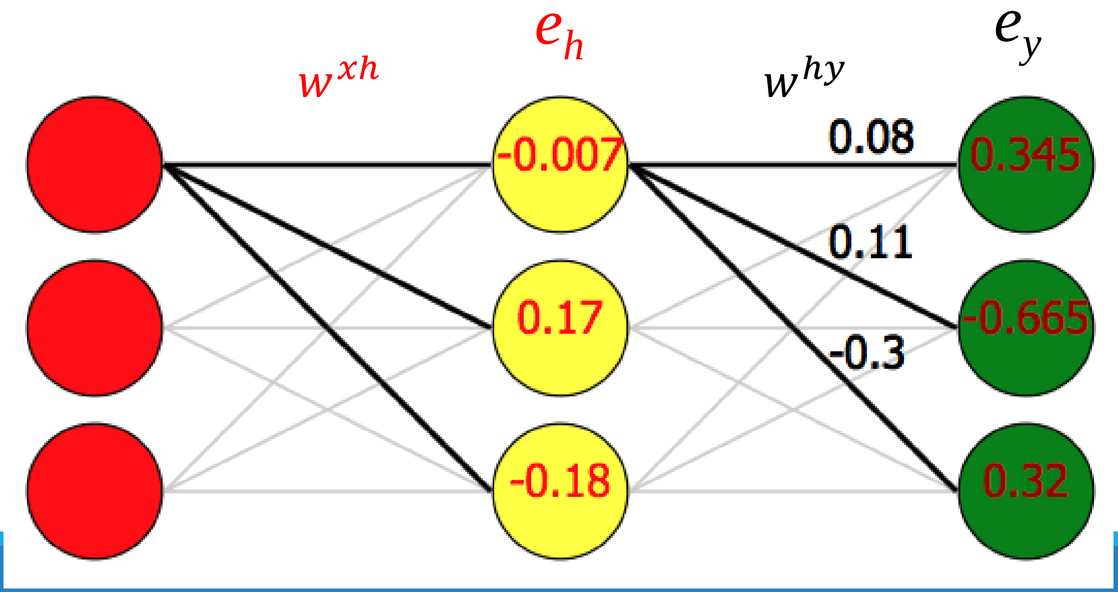
$$\delta w_{jk}^{hy} \propto h_j * e_k$$

- The theory of weights adjustment
  - Gradient descent, partial derivatives
  - The theory of optimization

Step 5: Backward propagation

Basic concept

- In Step 4 we use the error e_y to update w^{hy}
- Here we need to further update w^{xh}
  - Backpropagate the error of output layer e_y to hidden layer: the error of hidden layer e_h
  - Use the error e_h to update w^{xh}



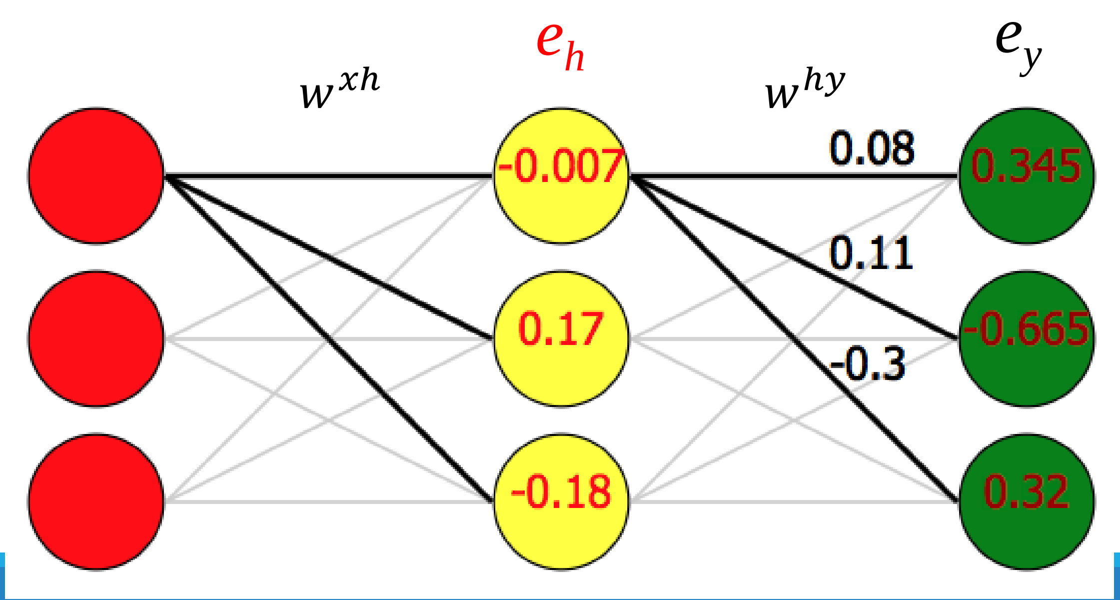
Step 5: Backward propagation

Error propagation

- Backpropagate the error of output layer e_y to hidden layer: the error of hidden layer e_h

$$e_h = e_y w^{hy}$$

$$= [0.345 \quad -0.665 \quad 0.32]\begin{bmatrix} 0.08 & 0.11 & -0.3 \\ 0.1 & -0.15 & 0.08 \\ 0.1 & 0.1 & -0.07 \end{bmatrix} = [-0.007 \quad 0.17 \quad -0.18]$$



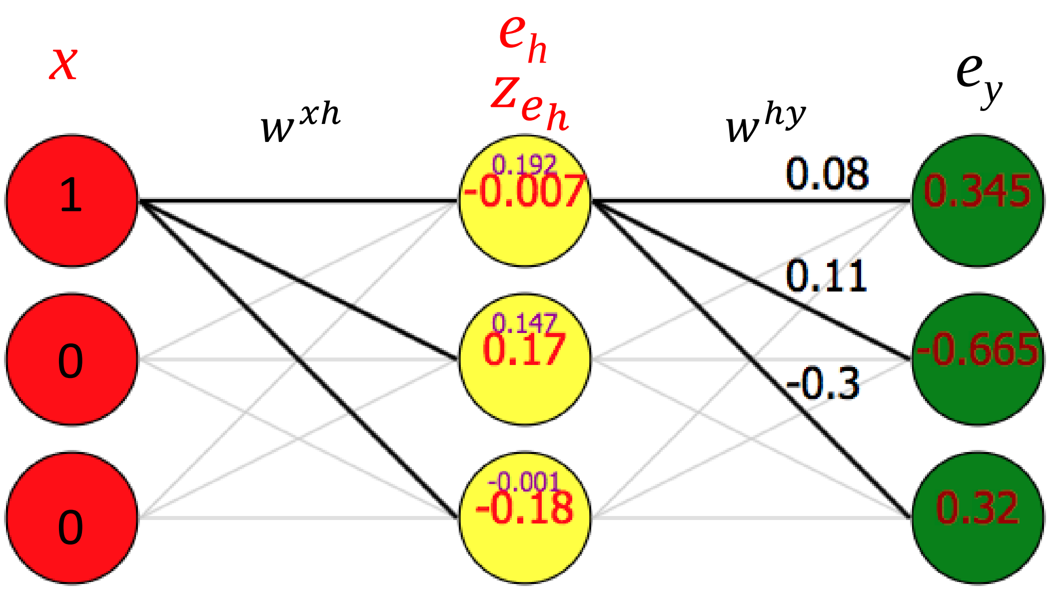
Step 5: Backward propagation

$$z_{e_h} = e_h \odot \left(1 - sigmoid^2(z_h)\right)$$

$$= [-0.007 \quad 0.17 \quad -0.18] \odot [0.961 \quad 0.978 \quad 0.999] = [0.192 \quad 0.147 \quad -0.001]$$

$$\delta w^{xh} = \alpha x^T z_{e_h} = 0.01 \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} [0.192 \quad 0.147 \quad -0.001]$$

$$= \begin{bmatrix} 0.00192 & 0.00147 & -0.00001 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

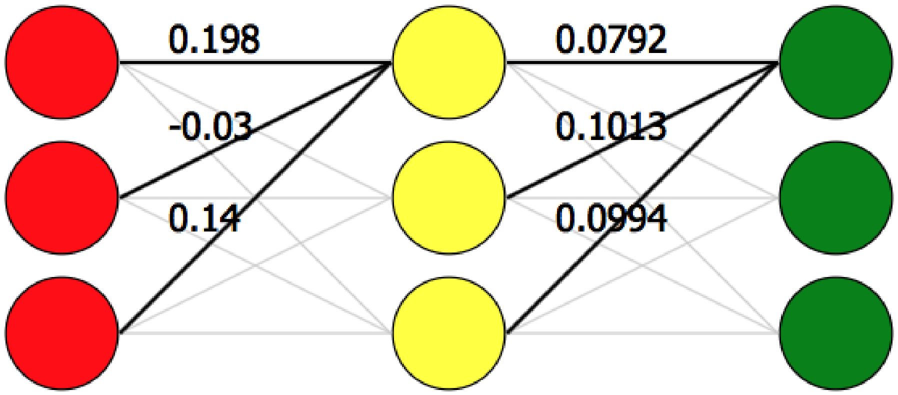


Step 5: Backward propagation

Changing weights

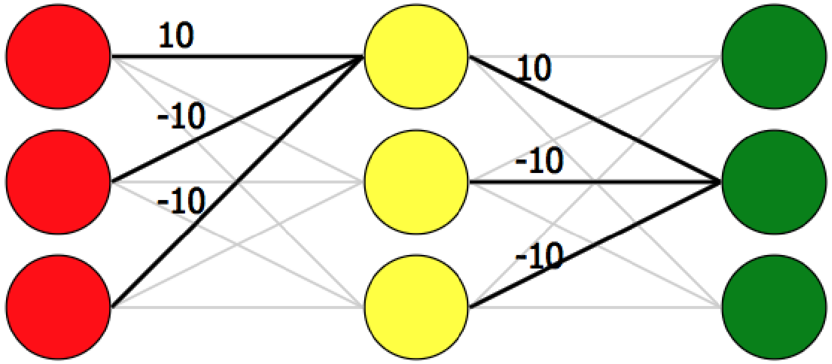
$$w^{xh} = \begin{bmatrix} 0.2 & 0.15 & -0.01 \\ -0.03 & -0.1 & -0.06 \\ 0.14 & -0.2 & 0.03 \end{bmatrix} \qquad \delta w^{xh} = \begin{bmatrix} 0.00192 & 0.00147 & -0.00001 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$w^{xh} - \delta w^{xh} = \begin{bmatrix} 0.19808 & 0.14853 & -0.00999 \\ -0.03 & -0.1 & -0.06 \\ 0.14 & -0.2 & 0.03 \end{bmatrix}$$

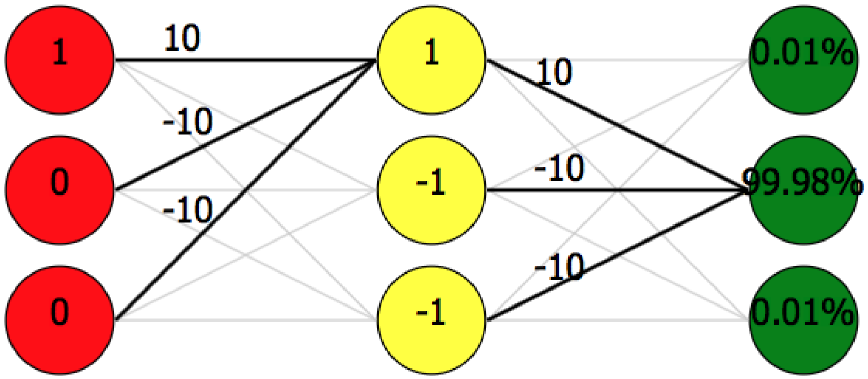


Final network

- Final training result



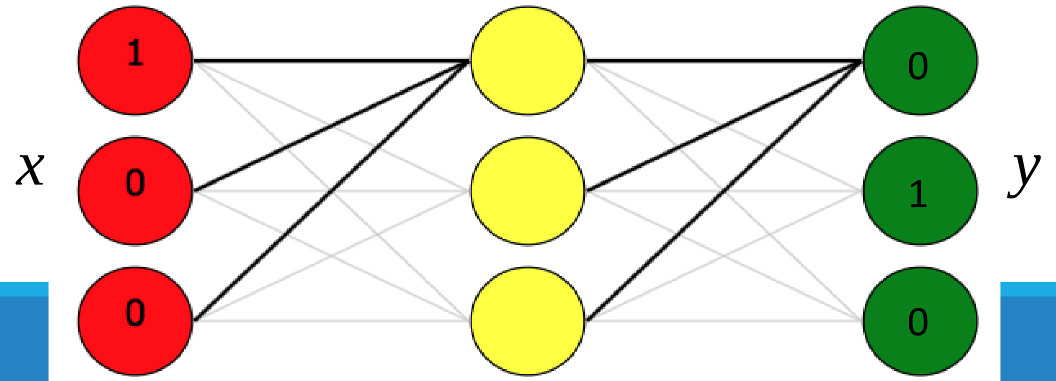
- Convert letter A to letter B
  - An input of 100
  - Hidden nodes activation values: +1, -1 and -1.
  - Output layer has weighted sums of -10, 10, -10,
    - Probabilities : 0%, 100%, 0%.
    - An output of 010.



Summary of the Single-sample Training

- Given a single training sample (x,y)
  - x: input values,   y: desired output values
- Network training will get a new weight matrix w
- Basic steps to train the network
  1. Randomly initialize the weight matrix $w=(w^{xh},w^{hy})$
  2. Forward propagation: $y'=xw$
  3. Compute the error: $E=y-y'$
  4. Compute weight change value by the error: $\Delta w=f(E)$
  5. Backpropagation: $w = w - \Delta w$
  6. Go to step 2

supervised learning



Learning of MLP Network

An example of backpropagation learning

Learning algorithms

Optimization and learning

The learning algorithm

- We just know how to train the MLP for "only one" learning sample: (x,y)
- How to train the MLP for a lot of learning samples, $\mathcal{X}=\{(x_1,y_1), (x_2,y_2), \ldots, (x_N,y_N)\}$?
  - Online learning
  - Offline(Batch) learning

Online learning vs. Batch learning

- Online
  - Randomly initialize w
  - For a $(x_i, y_i) \in \mathcal{X}$ in random order
    - Forward propagation: get error e
    - Backward propagation: get weight change Δw_i
    - Update w : $w = w - \Delta w_i$
  - Until convergence

Online learning is also called SGD(Stochastic gradient descent)

- Offline(Batch)
  - Randomly initialize w
  - While not converge
    - For all $(x_i, y_i) \in \mathcal{X}$ in sequential order
      - Forward propagation: get error e
      - Backward propagation: get weight change Δw_i
    - Average N weight changes: $\Delta w = (\sum_{i=1}^{N} \Delta w_i)/N$
    - Update w : $w = w - \Delta w$
  - Until convergence

Improving the learning algorithm

- Improving convergence
  - Momentum, adaptive learning rate
  - Improved gradient descent
- Mini-batch techniques
- Hardware acceleration
  - Parallel training, GPGPU

Parallel training of neural nets

An active topic of research.

- No clear winner yet.

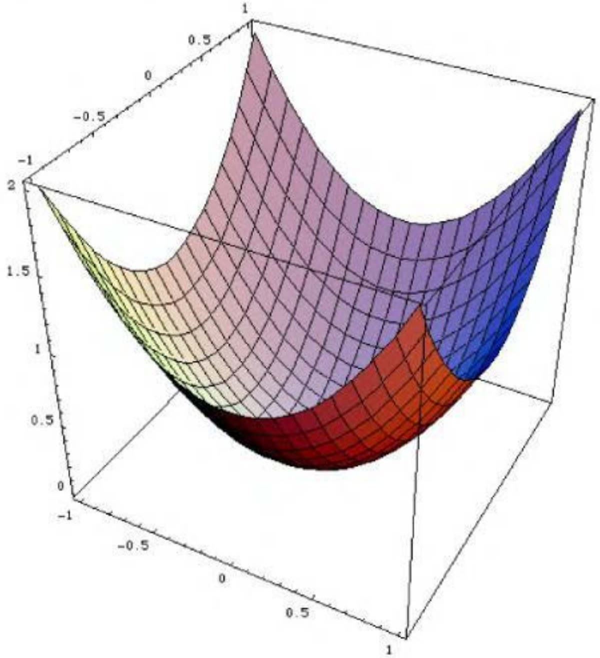
Baseline: lock-free stochastic gradient

- Assume shared memory
- Each processor access the weights through the shared memory
- Each processor runs SGD on different examples
- Read and writes to the weight memory are unsynchronized.
- Synchronization issues are just another kind noise…

Learning of MLP Network

An example of backpropagation learning
Learning algorithms
Optimization and learning

Convex



Definition

$\forall\, x, y,\ \forall\, 0 \leq \lambda \leq 1,$

$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y)$

Property

Any local minimum is a global minimum.

Conclusion

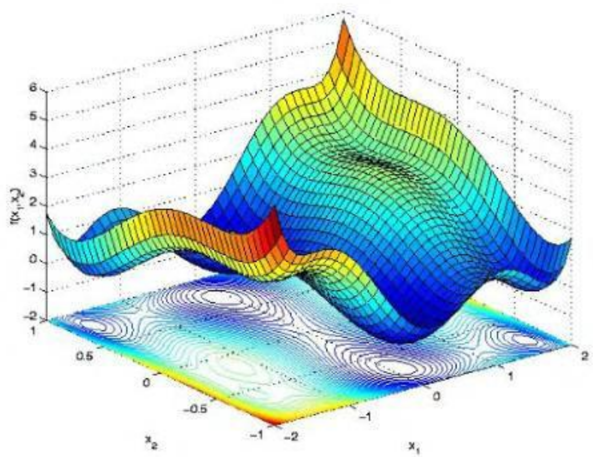
Optimization algorithms are easy to use.
They always return the same solution.

Example: Linear model with convex loss function.

– Curve fitting with mean squared error.

– Linear classification with log-loss or hinge loss.

Non-convex



Landscape

– local minima, saddle points.
– plateaux, ravines, etc.

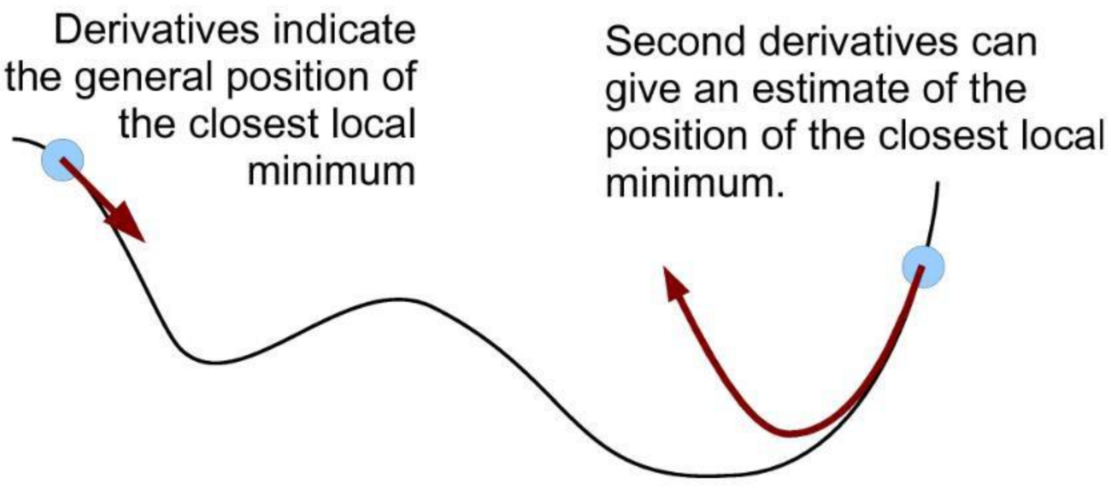
Optimization algorithms

– Usually find local minima.
– Good and bad local minima.
– Result depend on subtle details.

Examples

– Multilayer networks.
– Clustering algorithms.
– Learning features.
– Mixture models.
– Hidden Markov Models.
– Selecting features (some).

Derivatives



No such local cues without derivatives

- Derivatives may not exist.
- Derivatives may be too costly to compute.

Optimization vs. learning

Empirical cost

– Usually $f(w) = \frac{1}{n} \sum_{i=1}^{n} L(x_i, y_i, w)$

– The number n of training examples can be large (billions?)

Redundant examples

– Examples are redundant (otherwise there is nothing to learn.)

– Doubling the number of examples brings a little more information.

– Do we need it during the first optimization iterations?

Examples on-the-fly

– All examples may not be available simultaneously.

– Sometimes they come on the fly (e.g. web click stream.)

– In quantities that are too large to store or retrieve (e.g. click stream.)

Offline vs. online

Minimize $C(w) = \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} L(x_i, y_i, w)$.

Offline: process all examples together

– Example: minimization by gradient descent

$$\text{Repeat: } w \leftarrow w - \gamma \left(\lambda w + \frac{1}{n}\sum_{i=1}^{n} \frac{\partial L}{\partial w}(x_i, y_i, w) \right)$$

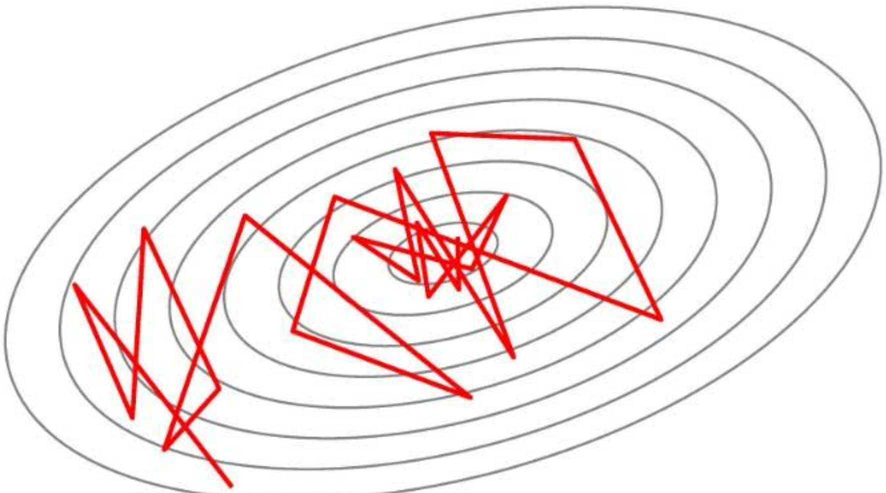
Offline: process examples one by one

– Example: minimization by stochastic gradient descent

Repeat: (a) Pick random example x_t, y_t

(b) $w \leftarrow w - \gamma_t \left(\lambda w + \frac{\partial L}{\partial w}(x_t, y_t, w) \right)$

Stochastic Gradient Descent



- Very noisy estimates of the gradient.
- Gain γ_t controls the size of the cloud.
- Decreasing gains $\gamma_t = \gamma_0(1 + \lambda\gamma_0 t)^{-1}$.
- Why is it attractive?