Machine Learning

Backpropagation for Neural Network Learning

Yuan-Kai Wang (王 元 凱)

ykwang@fju.edu.tw

http://www.ykwang.tw

Department of Electrical Engineering

Fu Jen Catholic University

輔仁大學電機工程系

2016~2025

Goal of This Unit

- Backpropagation is the algorithm to learn the weights of MLP(multi-layer perceptron)
  - It is a variant of gradient descent
- Important concepts in this unit
  - Error and loss: squared error, cross entropy
  - Chain rule of gradients

References

- Machine Learning: An Algorithmic Perspective, S. Marsland, Chapman and Hall/CRC, 2009
  - Chapter 3 The Multi-Layer Perceptron
- Chapter 11 Multilayer Perceptrons, Introduction to Machine Learning, 2nd, E. Alpaydin, MIT Press, 2010

Contents

- An example of backpropagation learning (***single*** training sample)
- Learning algorithms (***multiple*** training samples, training set)
- Optimization and learning

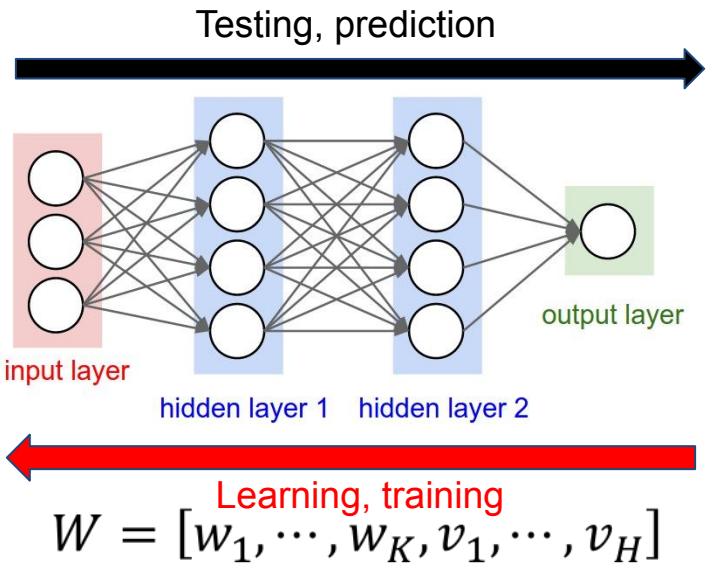
Training the MLP: Backpropagation

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: $w=B(x,y)$

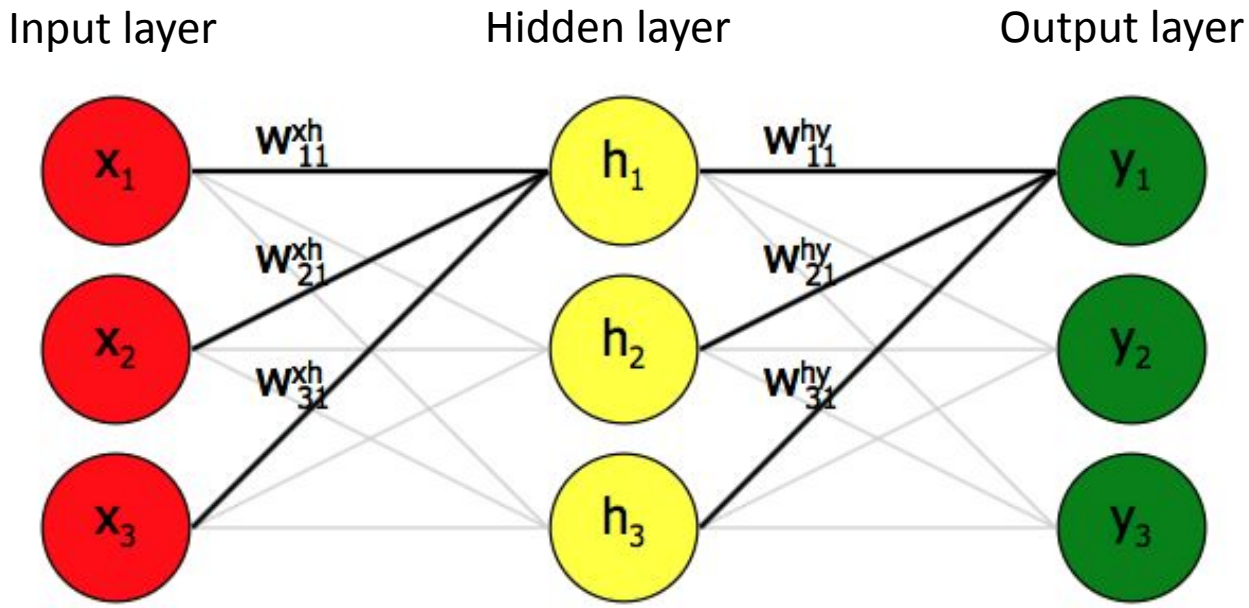
Backpropagation - Wikipedia

Source of the example: Machine Learning - Neural Networks Tutorial, Paul Tero of Existor Ltd, 2015

An Example of Learning MLP

An Example MLP

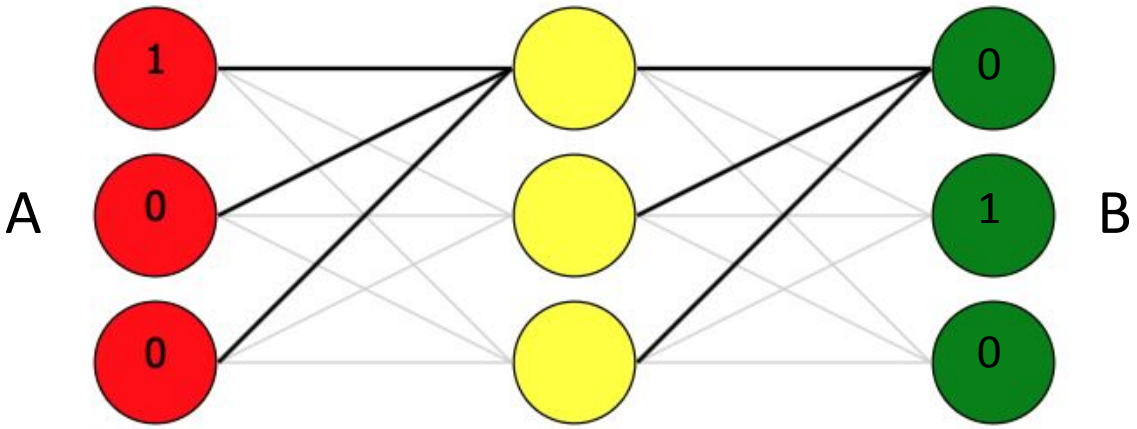
- A three-layer network: one hidden layer
  - 9 nodes (x_i, h_j, y_k), 3 neurons (h_j)
  - 18 weights: $w=(w^{xh}, w^{hy})$



Note: only 6 weights are labeled for examples.

Example Problem:
Convert letters A, B, C

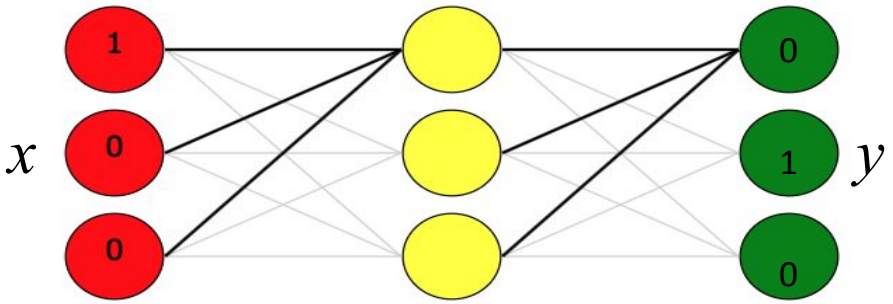
- Input: 1-of-K binary encoding
  - Letters are encoded into binary
  - A : 1 0 0, B : 0 1 0, C : 0 0 1
- Output
  - Convert A to B, B to C, C to A
  - 100 → 010, 010 → 001, 001 → 100



Training of the Network

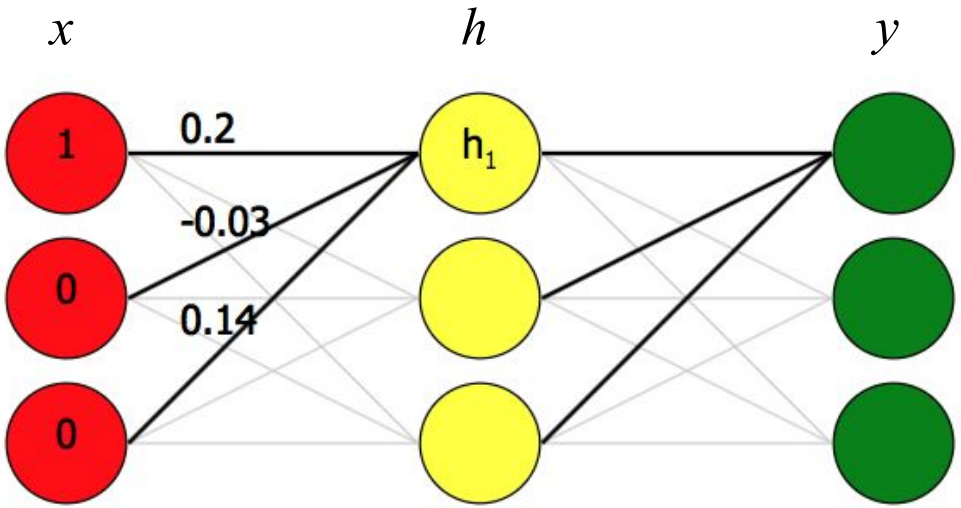
- Given a training pair (x,y)
  - x: input values, y: desired output values
- Network training will get a weight matrix $w=(w^{xh},w^{hy})$
- Basic steps to train the network
  1. Randomly initialize the weight matrix w
  2. Forward propagation: $y'=xw$
  3. Compute the error: $E=y-y'$
  4. Compute gradient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

supervised learning



Step 1: Random Starting Weights

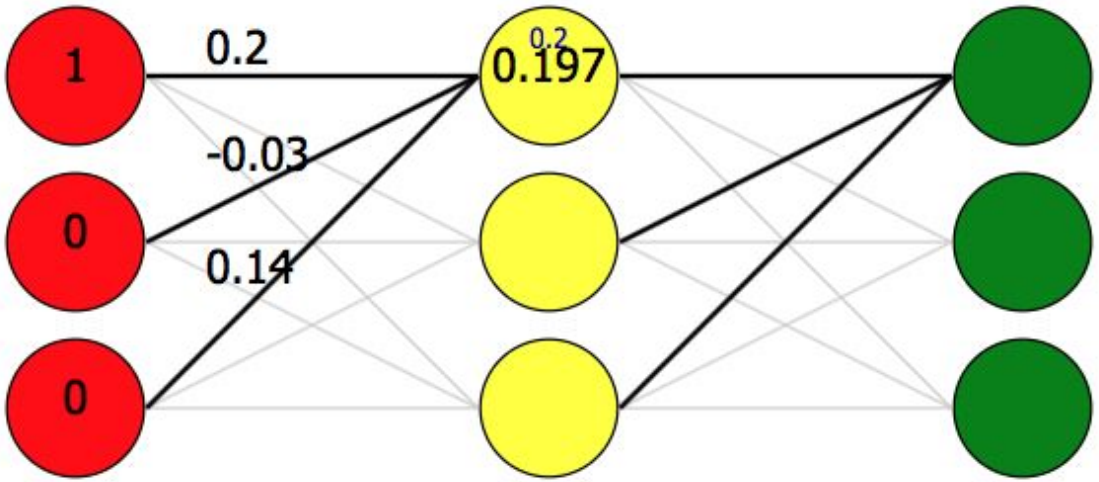
- We first compute the values of the first hidden node h_1 in the second layer
- The weights are usually initialised to be small random values between -1 and 1



Step 2: Forward Propagation
Weighted Sum

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

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

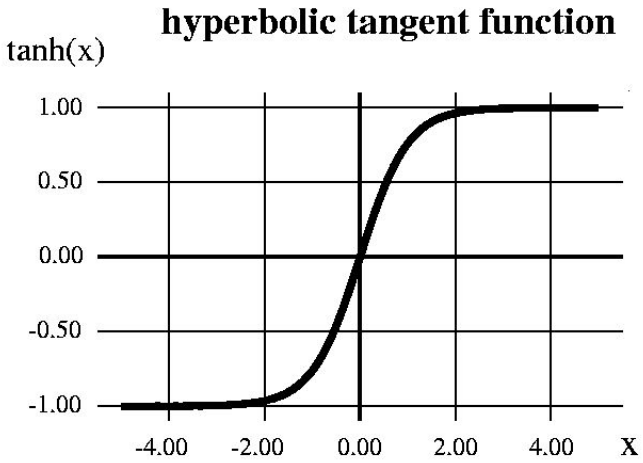


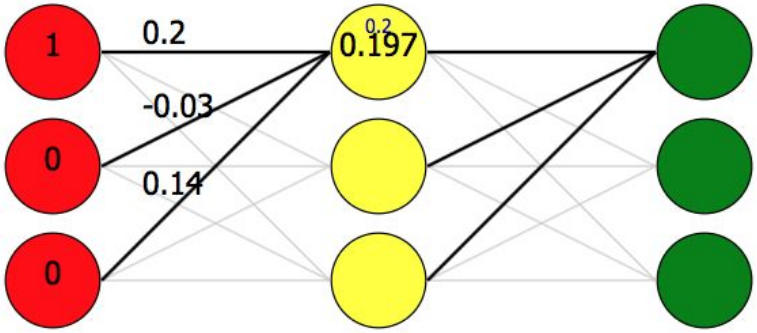
$$z_{h1} = \sum_{i=1}^{3} x_i w_{i1}^{xh}$$

Step 2: Forward Propagation
Activation of Weighted Sum

- Assume we use hyperbolic tangent

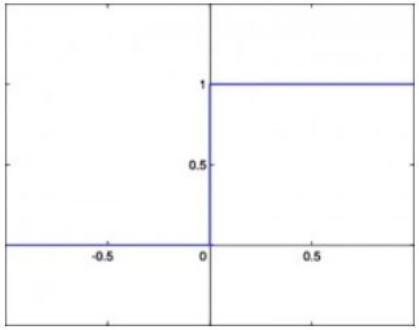
$$h_1 = \tanh\ (z_{h1})$$
$$= \tanh\ (0.2)$$
$$= 0.19738$$



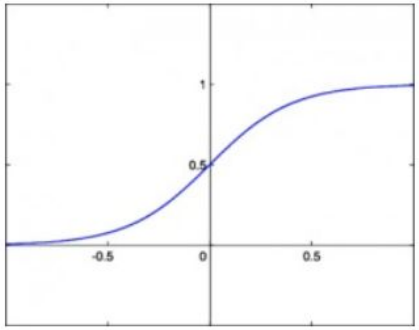


$$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$$

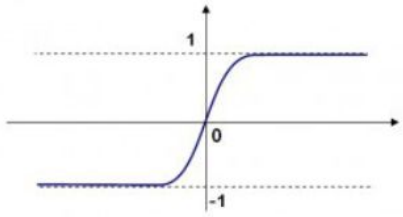
[Activation function - Wikipedia]



Binary Step Function



Binary Sigmoid Function

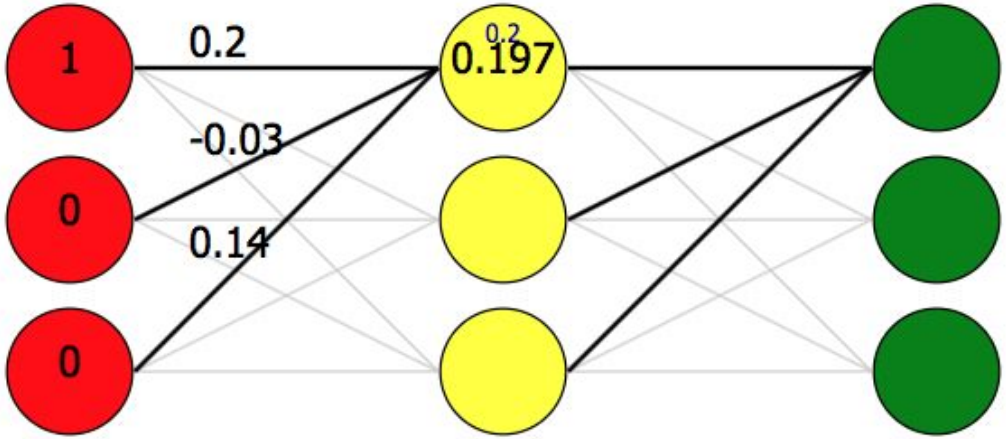


Bipolar Sigmoid function

Step 2: Forward Propagation
Matrix Notation

Formulation for a Single Neuron

$$h_j = \tanh\left(z_{hj}\right) = \tanh\left(\sum_{i=1}^{n_x} x_i w_{ij}^{xh}\right)$$



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

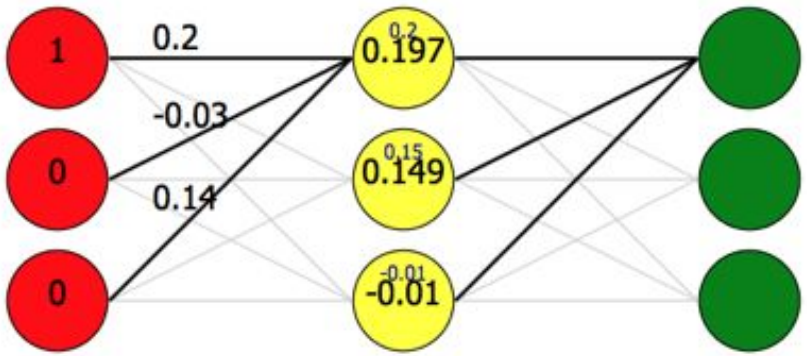
$$h_1 = \tanh\left(z_{h1}\right) = \tanh\left(0.2\right) = 0.19738$$

Step 2: Forward Propagation
Matrix Notation

Formulation for All Hidden Neurons

$$x = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

$$w^{xh} = \begin{bmatrix} 0.2 & 0.15 & -0.01 \\ -0.03 & -0.1 & -0.06 \\ 0.14 & -0.2 & -0.03 \end{bmatrix}$$



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

$$= \begin{bmatrix} 1 * 0.2 + 0 * -0.03 + 0 * 0.14 \\ 1 * 0.15 + 0 * -0.1 + 0 * -0.2 \\ 1 * -0.01 + 0 * -0.06 + 0 * -0.03 \end{bmatrix} = \begin{bmatrix} 0.2 & 0.15 & -0.01 \end{bmatrix}$$

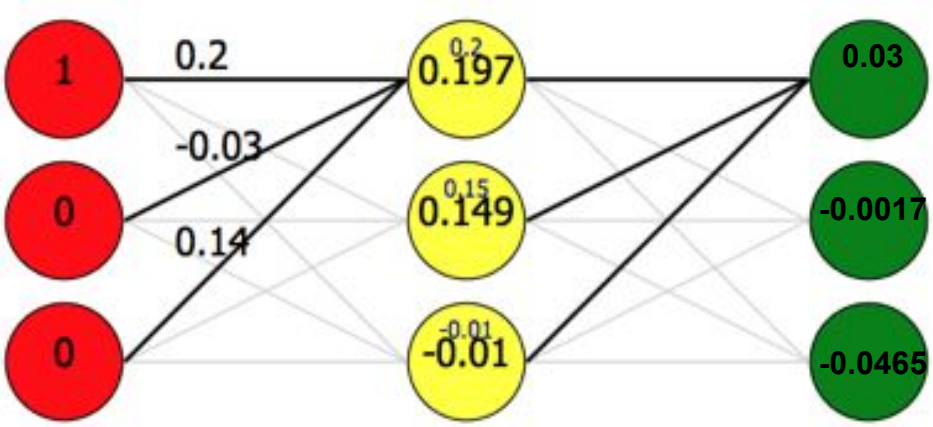
$$h = \tanh\left(xw_{xh}\right) = \tanh\left(\begin{bmatrix} 0.2 & 0.15 & -0.01 \end{bmatrix}\right) = \begin{bmatrix} 0.197 & 0.149 & -0.01 \end{bmatrix}$$

Step 2: Forward Propagation
Output Layer

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

$$w^{hy} = \begin{bmatrix} 0.08 & 0.11 & -0.3 \\ 0.1 & -0.15 & 0.08 \\ 0.1 & 0.1 & -0.07 \end{bmatrix} \qquad z_{yj} = \sum_{i=1}^{n_h} h_i w_{ij}^{hy}$$

$$z_y = hw^{hy} = [0.197 \quad 0.149 \quad -0.01] \begin{bmatrix} 0.08 & 0.11 & -0.3 \\ 0.1 & -0.15 & 0.08 \\ 0.1 & 0.1 & -0.07 \end{bmatrix} = [0.03 \quad -0.0017 \quad -0.0465]$$



real value
regression result

0

1

0

binary value
classification result

softmax function

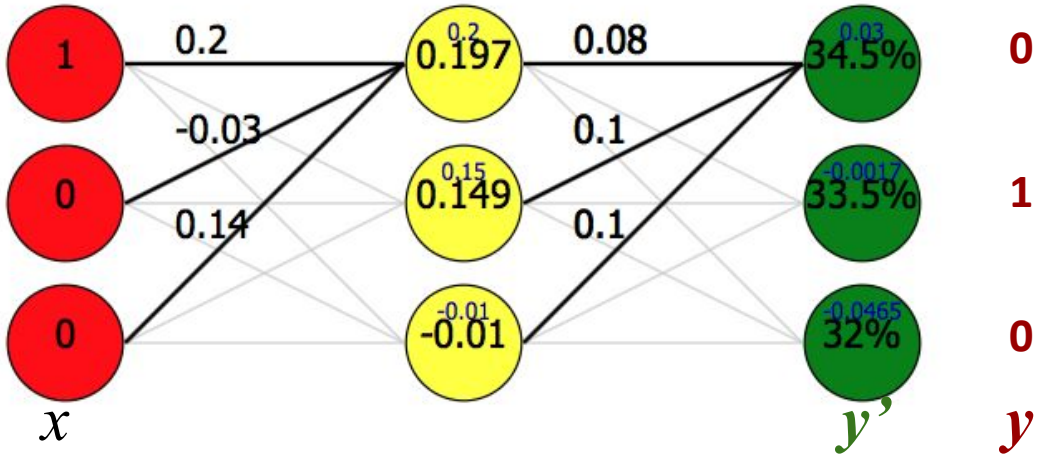
Step 2: Forward Propagation
Output Layer

- The softmax function

Softmax function - Wikipedia

$$p = \text{softmax}(z_y)$$
$$= \text{softmax}([0.03 \quad -0.0017 \quad -0.0465])$$
$$= [0.345 \quad 0.335 \quad 0.32] \quad = y'$$

$$\text{softmax}(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{n_y} e^{z_k}}$$



0
1
0
y

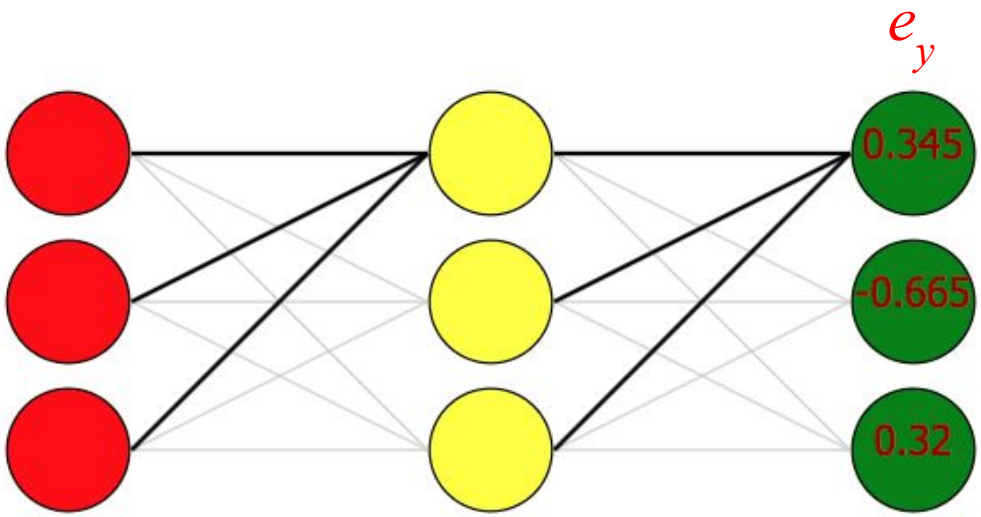
Answer A : 34.5% chance
Answer B : 33.5% change
Answer C : 32% change

The random w gets a wrong output in feed forward phase

Modify w by backward computation

Step 3: Computing Output Error

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



$$e_y = p - y = [0.345 \quad 0.335 \quad 0.32] - [0 \quad 1 \quad 0]$$
$$= [0.345 \quad -0.665 \quad 0.32]$$

Step 3: Computing Output Error
Loss & Cross Entropy

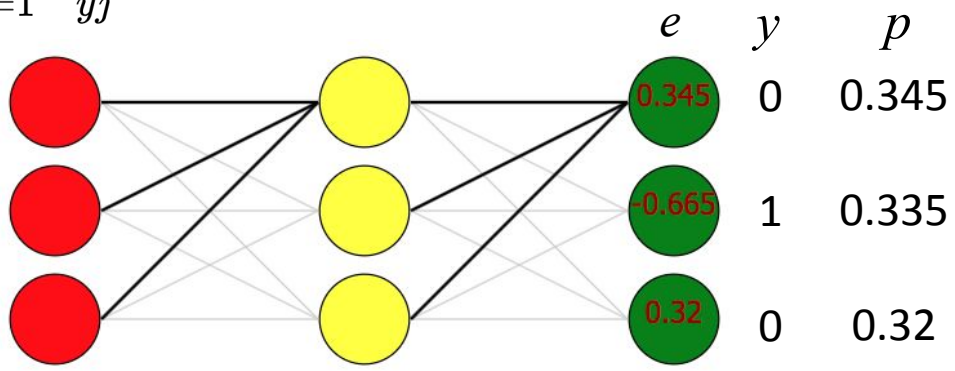
- We need to calculate the total error for all the outputs combined
- This is called the loss or cost of the network
  - The loss (cost) function is labelled with J
- Three possible J
  - Absolute error
  - Squared error
  - Cross entropy

$$J = \sum_{j=1}^{n_y} |e_{yj}| = 0.345 + 0.665 + 0.32 = 1.32$$

$$J = \sum_{j=1}^{n_y} e_{yj}^2 = 0.664$$

$$J = -\sum_{j=1}^{n_y} y_j \log p_j$$

$$J = -y_1 \log p_1 - y_2 \log p_2 - y_3 \log p_3$$
$$= -0 - 1 * \log(0.335) - 0 = 1.0936$$



Step 3: Computing Output Error
Loss & Cross Entropy

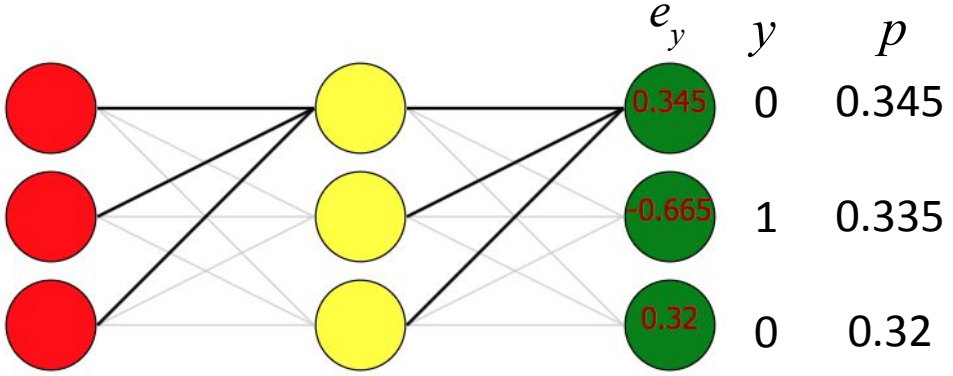
- Loss/cost function J : Cross entropy

$$J = -\sum_{j=1}^{n_y} y_j \log p_j$$

$$J = -y_1 \log p_1 - y_2 \log p_2 - y_3 \log p_3$$
$$= -0 - 1 * \log(0.335) - 0 = 1.0936$$

Output error e_y

$$e_y = p - y$$

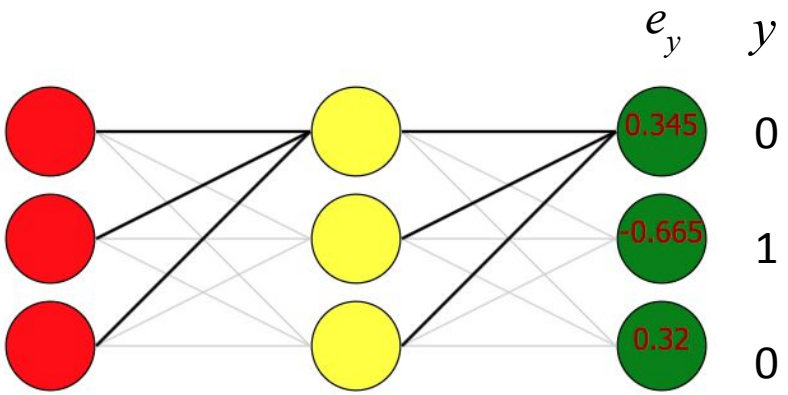


The random w gets a cross entropy value 1.0936 > 0.04

Modify w by backward computation

Step 4: Compute Gradient of Error
Intuition

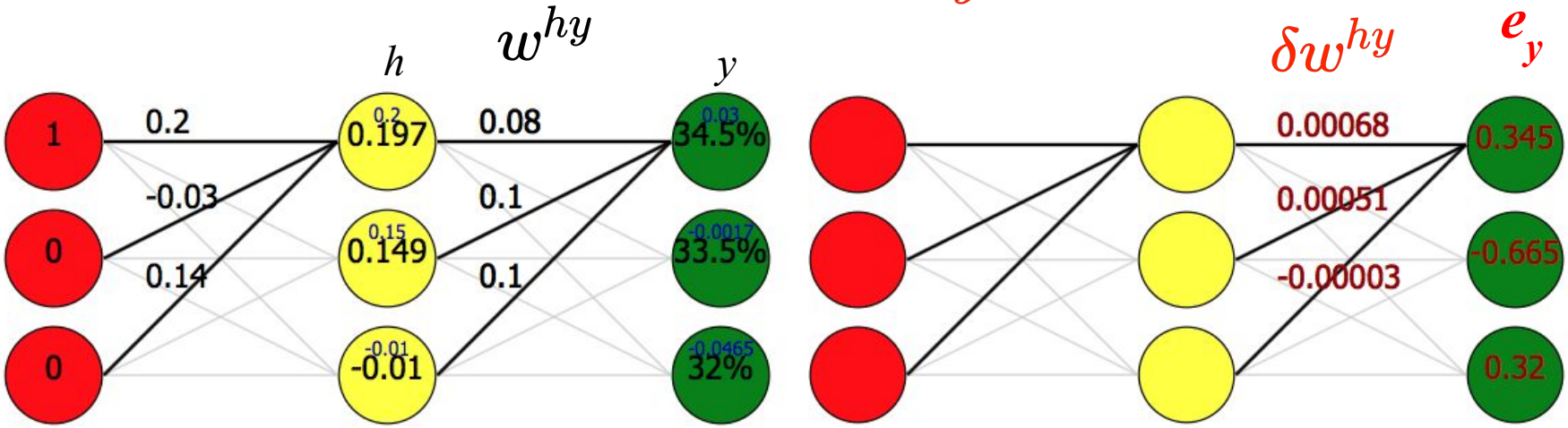
- The weights going into y_1 and y_3 should be lowered a bit, because their estimate was too high
- The weights going into y_2 should be raised because they were way too low and caused a large negative error
- The bigger the error, the more the weights should be changed



Step 4: Compute Gradient of Error
from output to hidden

$$w^{hy} = w^{hy} - \delta w^{hy}$$

$$\delta w^{hy} = h e_y$$



$$\delta w_{11}^{hy} = h_1 e_{y1}$$

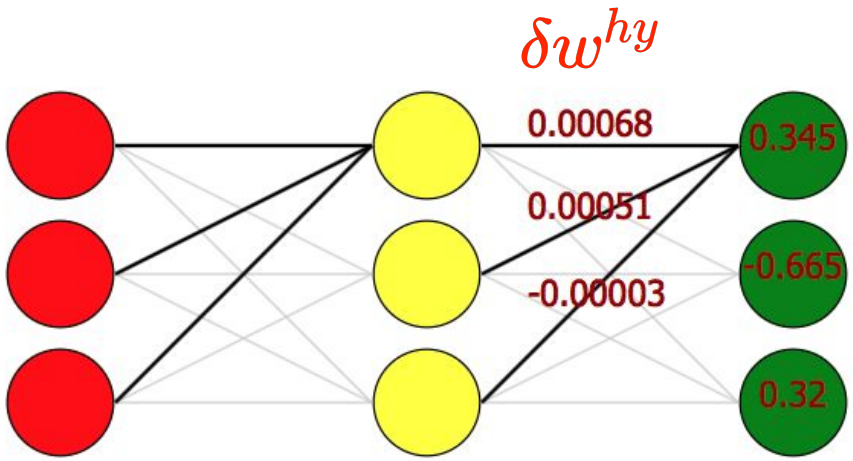
$$\delta w_{11}^{hy} = h_1 e_{y1} = 0.197 * 0.345 = 0.068$$

$$\delta w_{11}^{hy} = \alpha h_1 e_{y1} = 0.01 * 0.197 * 0.345 = 0.00068$$

Step 4: Compute Gradient of Error
Matrix operation from output to hidden

- We can compute all the adjustments with one matrix operation

$$\delta w^{hy} = \alpha h^T e_y = 0.01 \begin{bmatrix} 0.197 \\ 0.149 \\ -0.01 \end{bmatrix} \begin{bmatrix} 0.345 & -0.665 & 0.32 \end{bmatrix} = \begin{bmatrix} 0.00068 & -0.00131 & 0.00063 \\ 0.00051 & -0.00099 & 0.00047 \\ -0.00003 & 0.00007 & -0.00003 \end{bmatrix}$$



Step 4: Compute Gradient of Error
Theory

- Why the formula?

$$w^{hy} = w^{hy} - \delta w^{hy} \qquad \delta w^{hy} = h e_y$$

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

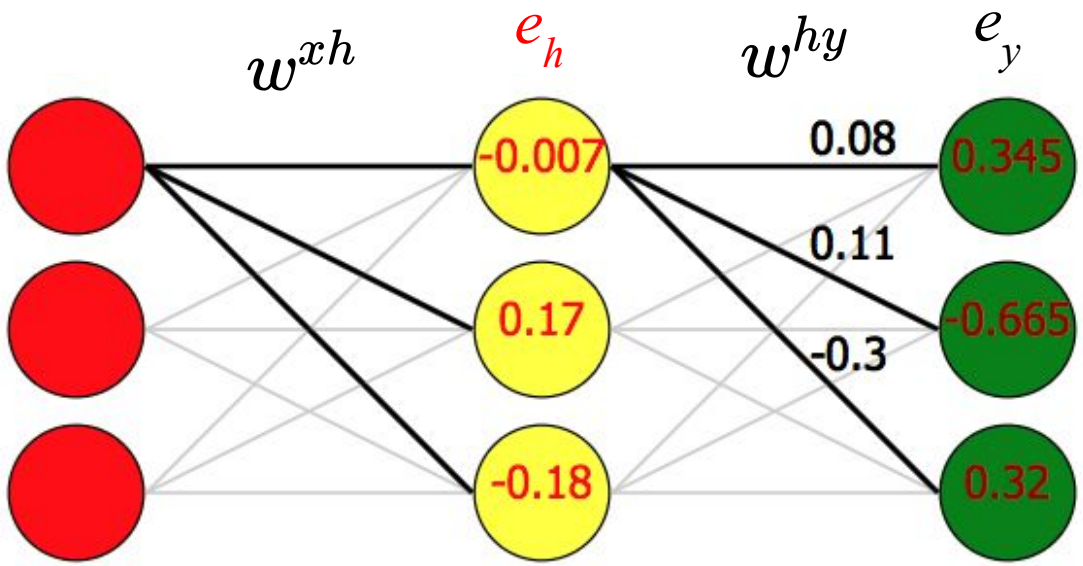
$$y = x^2, \frac{\partial y}{\partial x} = 2x$$

Machine Learning - Neural Networks Tutorial,

Paul Tero of Existor Ltd, 2015

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}

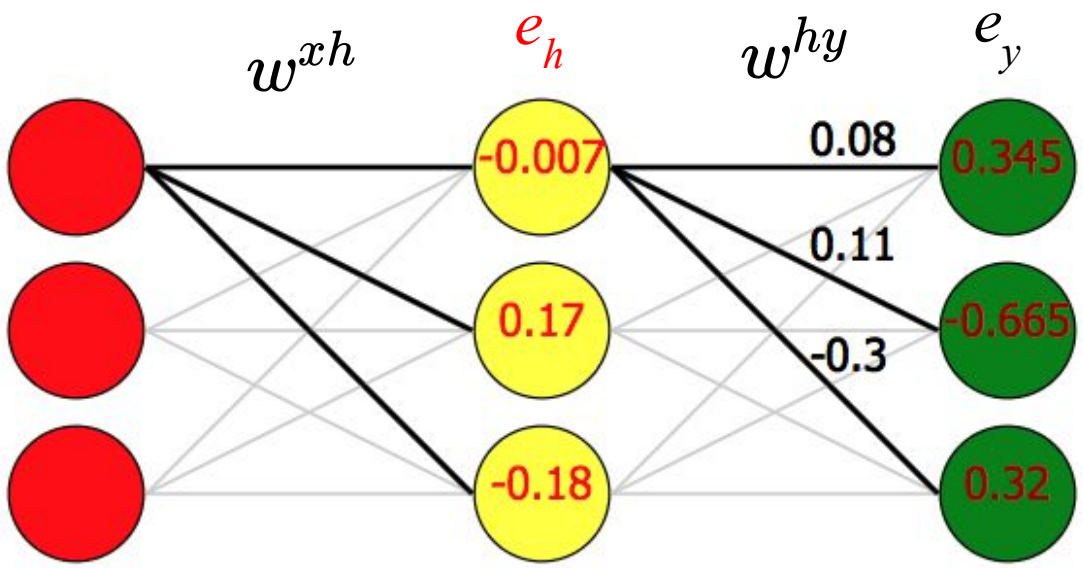


Step 5: Backward Propagation
Backpropagate Errors to Hidden Layer

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

$$= \begin{bmatrix} 0.345 & -0.665 & 0.32 \end{bmatrix} \begin{bmatrix} 0.08 & 0.11 & -0.3 \\ 0.1 & -0.15 & 0.08 \\ 0.1 & 0.1 & -0.07 \end{bmatrix}$$

$$= \begin{bmatrix} -0.007 & 0.17 & -0.18 \end{bmatrix}$$



Step 5: Backward Propagation
Backpropagate Errors to Hidden Layer

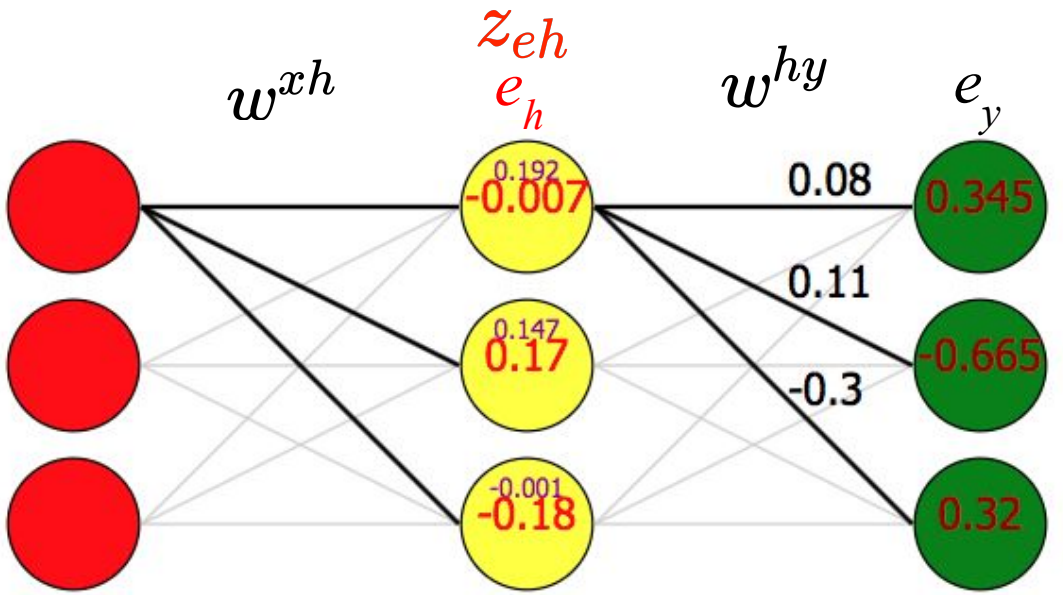
Then we undo the activation function to find the corresponding weighted sum

$$z_{eh} = e_h \odot (1 - \tanh^2(z_h)) = e_h \odot (1 - \tanh^2([0.2 \quad 0.15 \quad -0.01]))$$
$$= [-0.007 \quad 0.17 \quad -0.18] \odot [0.961 \quad 0.978 \quad 0.999] = [0.192 \quad 0.147 \quad -0.001]$$

$\odot$: elementwise multiplication

$z_{eh} = e_h \odot (1 - \tanh^2(z_h))$

derived from math

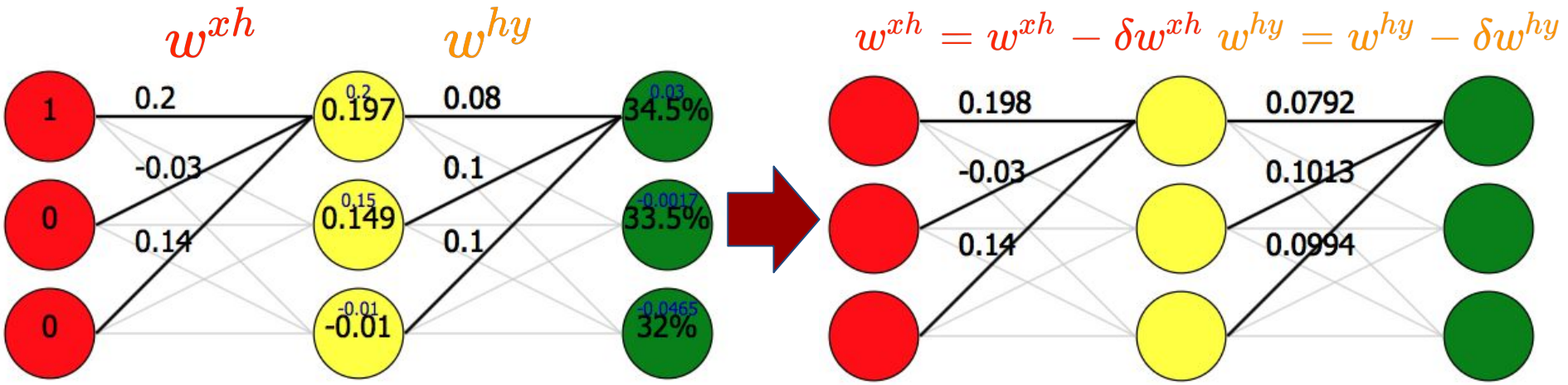


Step 5: Backward Propagation
Changing Weights

$\mathbf{Y \rightarrow H:}\ w^{hy} = w^{hy} - \delta w^{hy} \qquad \delta w^{hy} = \alpha h e_y$

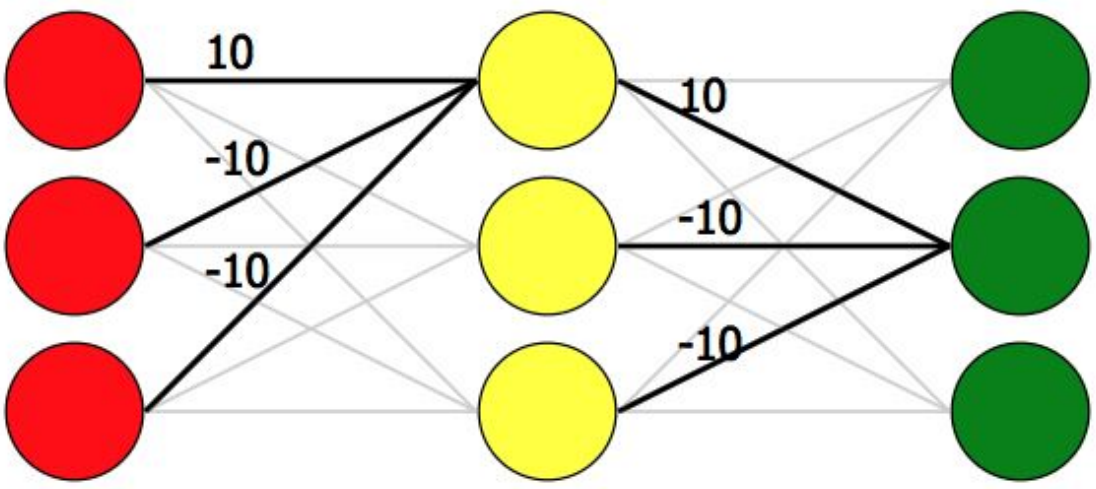
$\mathbf{H \rightarrow X:}\ w^{xh} = w^{xh} - \delta w^{xh} \qquad \delta w^{xh} = \alpha x z_{eh}$

$$\delta w^{xh} = \alpha x^T z_{eh} = 0.01 \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0.192 & 0.147 & -0.001 \end{bmatrix} = \begin{bmatrix} 0.00192 & 0.00147 & -0.00001 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$



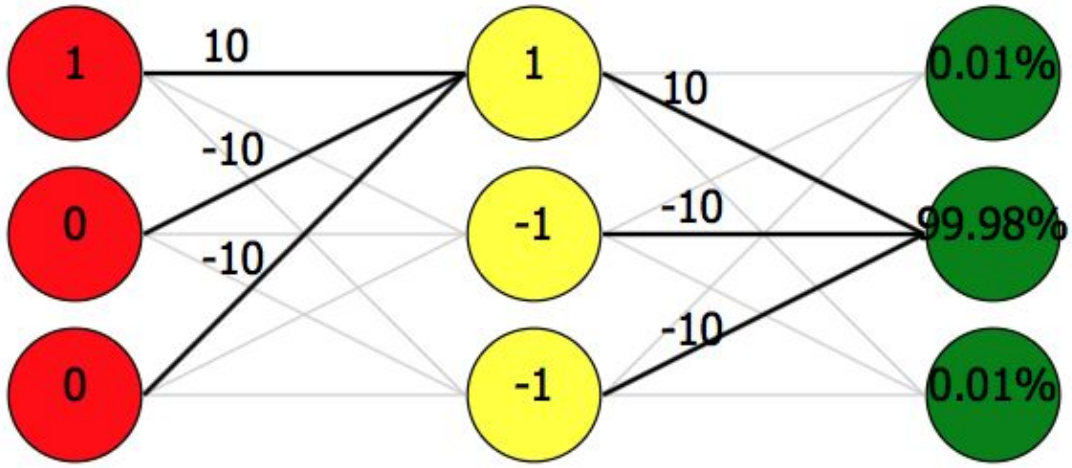
Final Network
an example

- Assume we finally get the trained MLP as below



Final Network
Convert A to 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 gradient of error: $\Delta w=f(E)$
  5. Backpropagation: $w = w - \Delta w$
  6. Go to step 2 (next iteration)

single iteration

(=single sample, if it is online gradient descent)

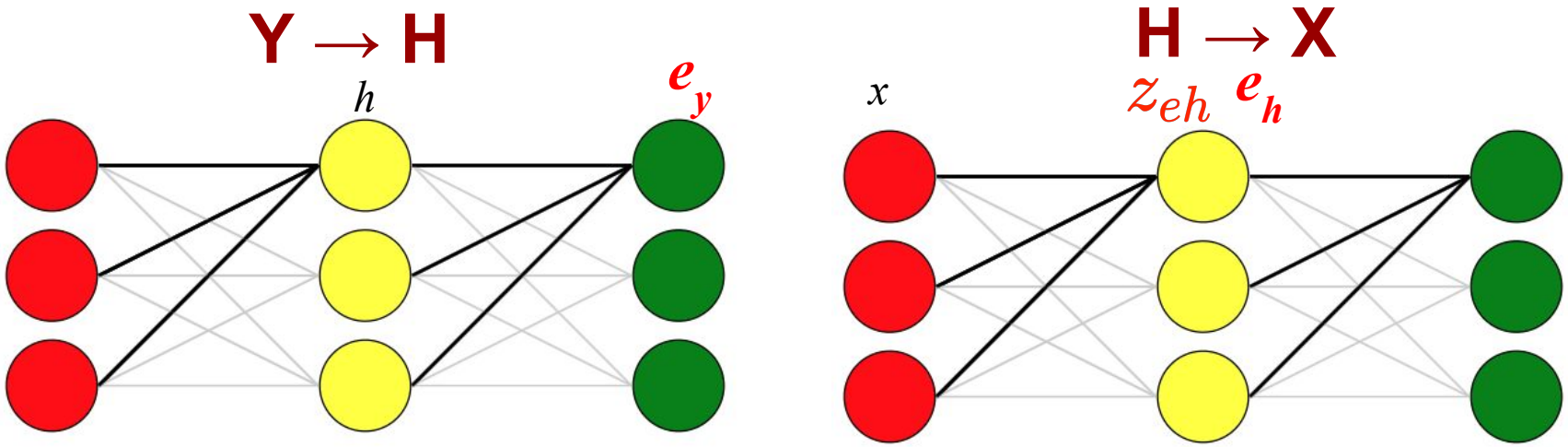
Backpropagation Summary

Y → H : $w^{hy} = w^{hy} - \delta w^{hy}$ $\quad \delta w^{hy} = \alpha h e_y$

output error $e_y = p - y$

H → X : $w^{xh} = w^{xh} - \delta w^{xh}$ $\quad \delta w^{xh} = \alpha x z_{eh}$

$e_h = e_y w^{hy}$ $\quad z_{eh} = e_h \odot (1 - \tanh^2(z_h))$



Red equations are derived from mathematics : **gradient descent** and **chain rule**

See Appendix for mathematical derivations

Learning Algorithms

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, $\mathscr{X}=\{(x_1,y_1), (x_2,y_2), \dots , (x_N,y_N)\}$?
  - Online learning
  - Offline(Batch) learning

Online Learning vs. Batch Learning

- Online
  - Randomly initialize w
  - Loop for a $(x_i, y_i) \in \mathcal{X}$ in random order
    - Forward propagation: get error $\mathcal{E}$
    - Backward propagation: get weight change Δw_i
    - Update w : $W = W - \alpha \nabla \varepsilon_i(W)$
  - Until convergence
- Batch
  - Randomly initialize w
  - Loop for all n samples $(x_i, y_i) \in \mathcal{X}$
    - Forward propagation: get error $\mathcal{E}$
    - Backward propagation: get weight change Δw_i
    - Update w : $W = W - \alpha \sum_{j=1}^{n} \nabla \varepsilon^j(W)/n$
  - Until convergence

Online learning is also called SGD(Stochastic gradient descent)

Improving the Learning Algorithm

- Improving convergence
  - Acceleration of convergence
    vs. lower error rate of learning
    (these two goals are contradictory)
  - Methods
    - Momentum
    - Adaptive learning rate α
    - Improved gradient descent (line search …)
- Mini-batch techniques
- Hardware acceleration: Parallel training, GPGPU

Parallel Training

- **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…

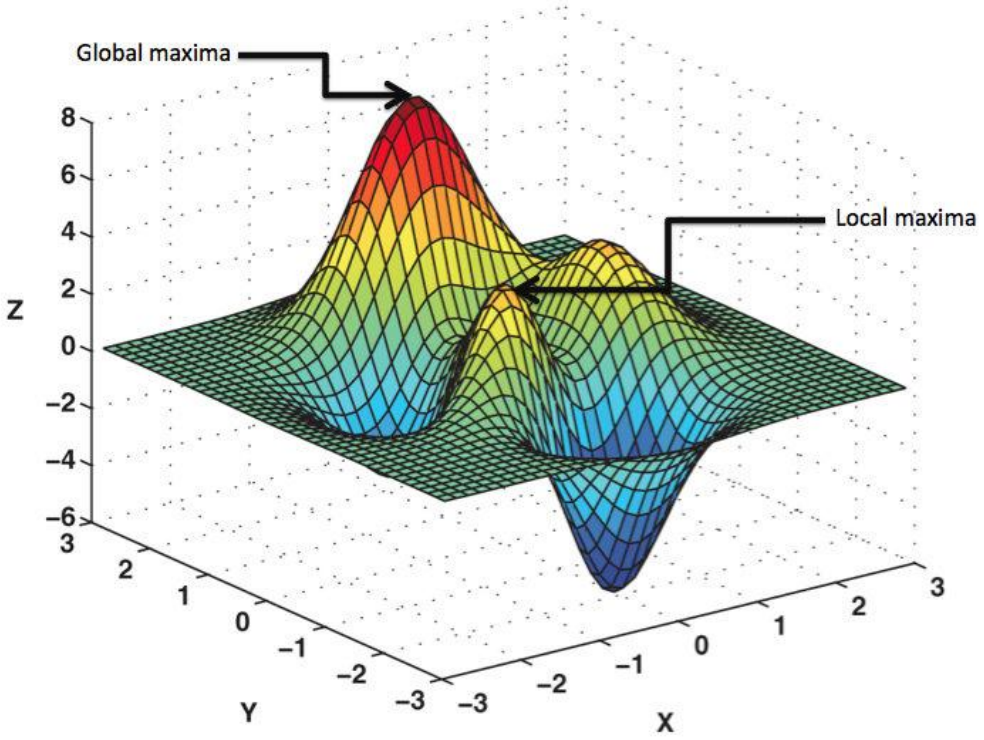
Local optima vs Global optima

Convex vs Non-convex

Offline vs Online

Optimization and Learning

What is Optima



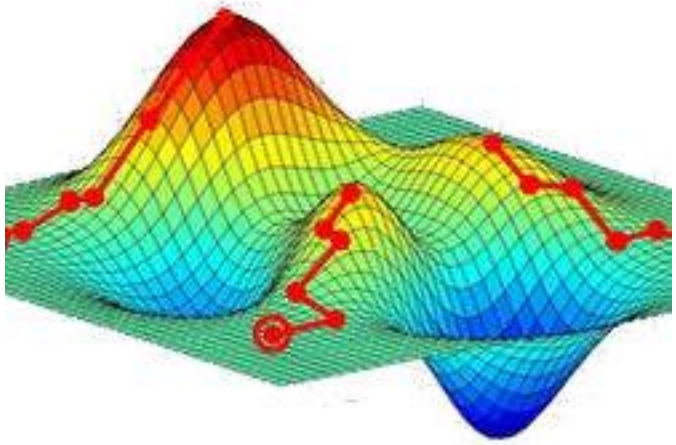
Optima can be maxima or minima

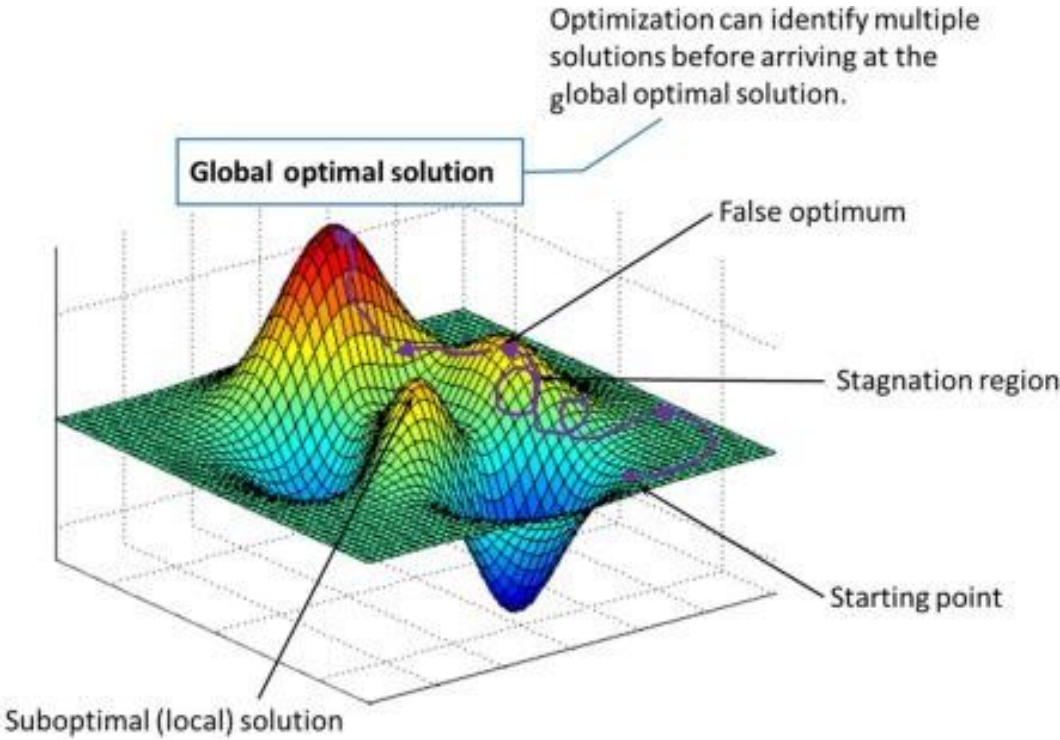
Maxima or minima? It depends on z

If z is error: minima is optima

If z is accuracy: maxima is optima

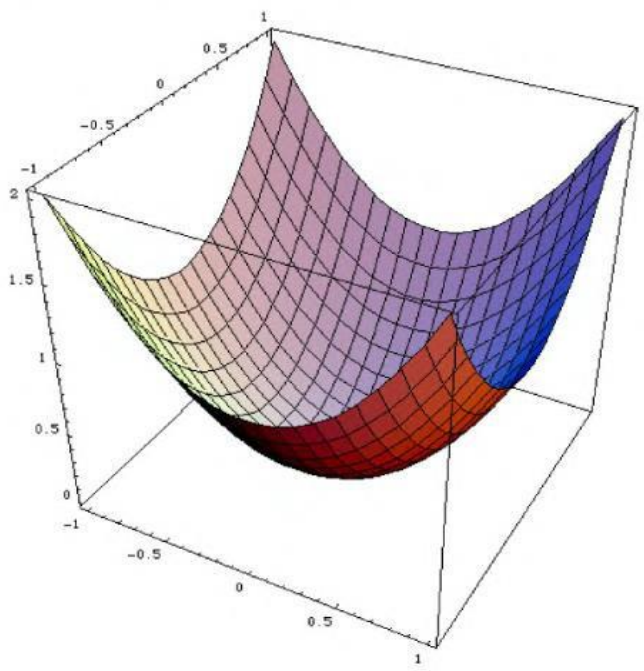
Local Optima vs Global Optima

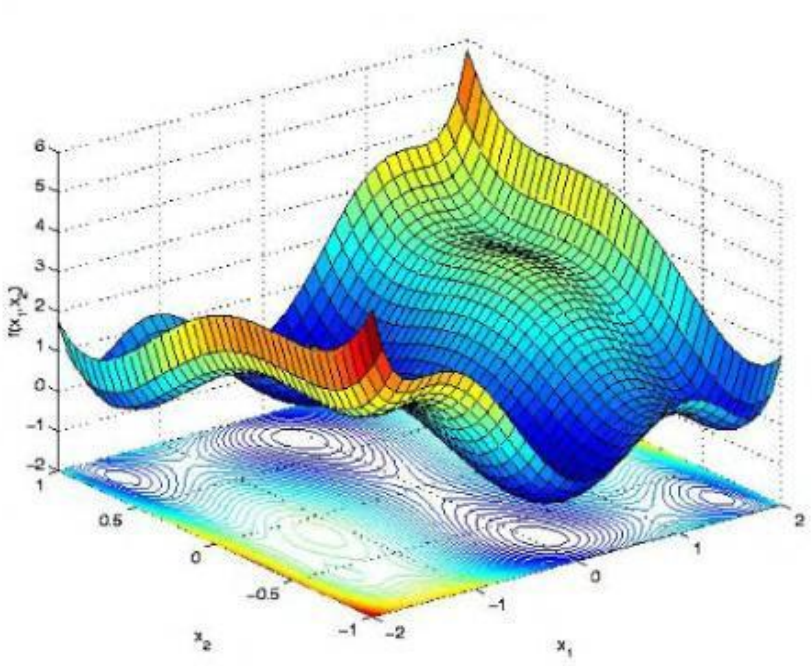




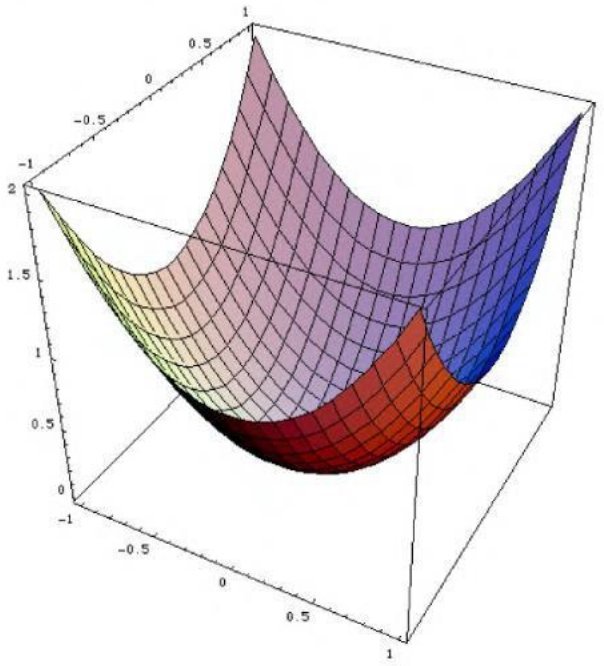
Convex vs Non-convex

Two different landscapes





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.

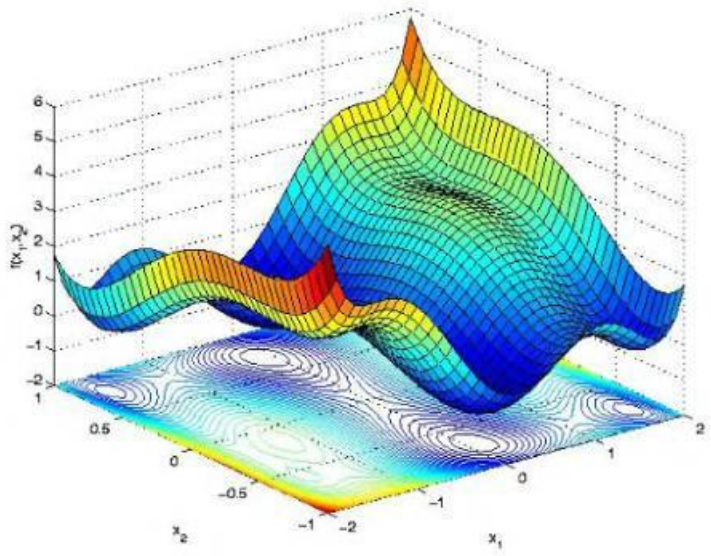
Conclusion

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.

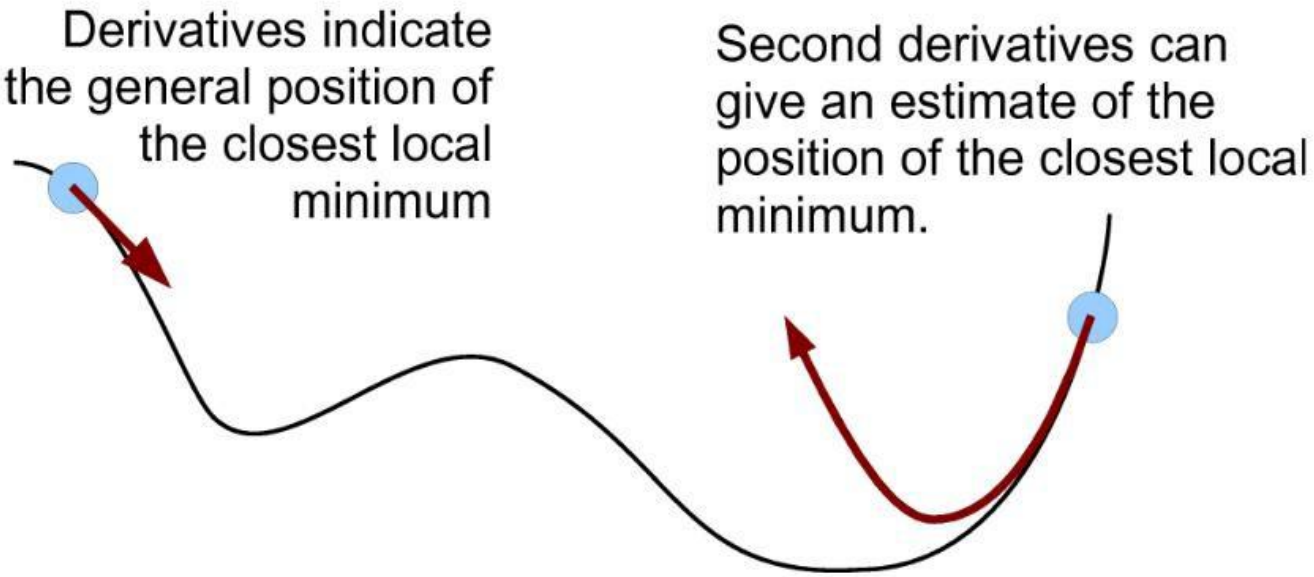
Optimization algorithms

– 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

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

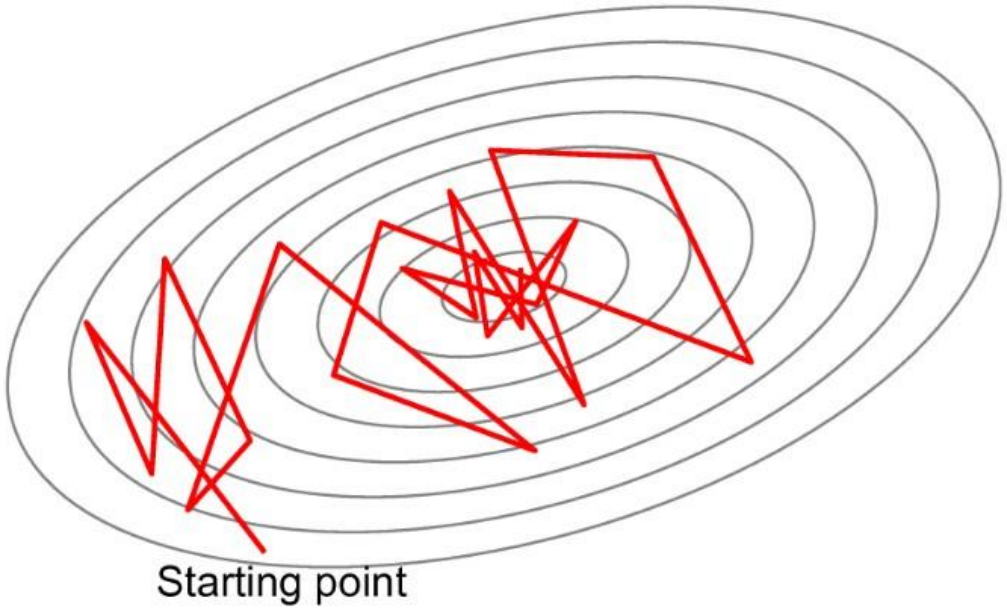
Online : process examples one by one

– 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?

Source of the example: Machine Learning - Neural Networks Tutorial, Paul Tero of Existor Ltd, 2015

Appendix: Theory of Backpropagation

Two Derivatives

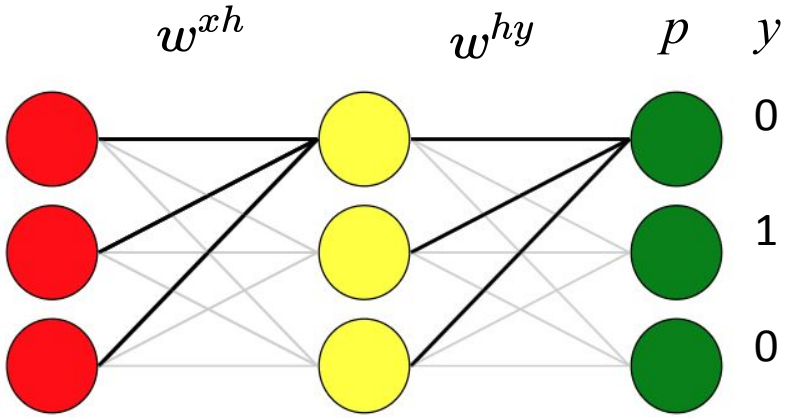
Y → H : $w^{hy} = w^{hy} - \delta w^{hy}$ $\quad \delta w^{hy} = \alpha h e_y$

output error $e_y = p - y$

H → X : $w^{xh} = w^{xh} - \delta w^{xh}$ $\quad \delta w^{xh} = \alpha x z_{eh}$

$e_h = e_y w^{hy}$ $\quad z_{eh} = e_h \odot (1 - \tanh^2(z_h))$

$$\delta w^{xh} = \frac{\partial J}{\partial w^{xh}} = x z_{eh} \qquad \delta w^{hy} = \frac{\partial J}{\partial w^{hy}} = h e_y$$

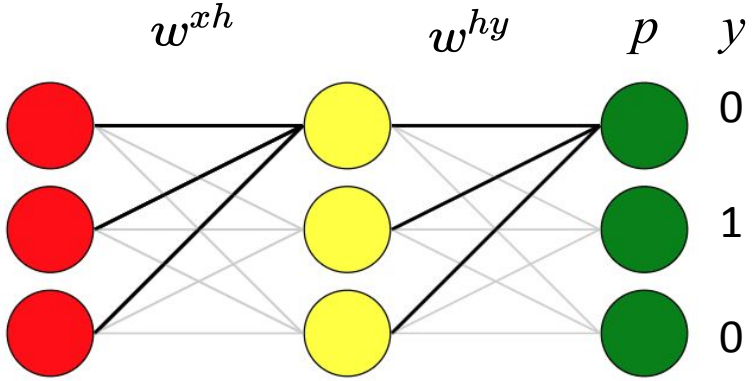


Partial Derivatives $\frac{\partial J}{\partial w_{ij}^{hy}}$

- We need to calculate how the loss J changes as each weight changes

$$\frac{\partial J}{\partial w_{ij}^{hy}} = h_i e_{yj}$$

"change in the loss J" with respect to the change in the weight w_{ij}^{hy} between h_i and y_j



$$J = -\sum_{j=1}^{n_y} y_j \log p_j \quad : \text{ cross entropy}$$

Partial Derivatives $\frac{\partial J}{\partial w_{ij}^{hy}}$

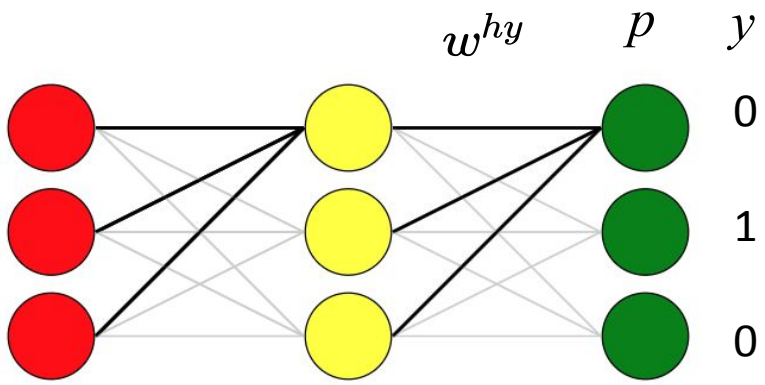
- We need to proceed according to the chain rule of calculus, deriving one step at a time
- Chain rule : chain together a bunch of different derivatives like this

$$\frac{\partial J}{\partial w_{ij}^{hy}} = \frac{\partial J}{\partial p_j} \frac{\partial p_j}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}^{hy}}$$

$$J = -\sum_{j=1}^{n_y} y_j \log p_j$$

$$p_j = \text{softmax}(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{n_y} e^{z_k}}$$

$$z_y = hw^{hy} \quad z_{yj} = \sum_{i=1}^{n_h} h_i w_{ij}^{hy}$$



Partial Derivatives $\frac{\partial p_j}{\partial z_j}$

- We start with the second one (which is softmax) because it will reveal something that is needed to compute the first one

$$\frac{\partial J}{\partial w_{ij}^{hy}} = \frac{\partial J}{\partial p_j} \frac{\partial p_j}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}^{hy}}$$

$$p_j = \text{softmax}(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{n_y} e^{z_k}}$$

- The derivative of e^z is also e^z $\partial e^z = e^z$
- It is a function which changes at the same rate as itself

Partial Derivatives $\frac{\partial p_j}{\partial z_j}$

- We break apart the top and bottom halves of the fraction and treat them separately and then apply the chain rules for things that are being multiplied together

$$p_j = \text{softmax}(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{ny} e^{z_k}}$$

$$\frac{\partial p_j}{\partial z_j} = \partial\left(e^{z_j}\frac{1}{\sum e^{z_k}}\right) = \partial\left(e^{z_j}\right)\frac{1}{\sum e^{z_k}} + e^{z_j}\,\partial\left(\frac{1}{\sum e^{z_k}}\right)$$

$$= \partial\left(e^{z_j}\right)\frac{1}{\sum e^{z_k}} - e^{z_j}\frac{1}{\left(\sum e^{z_k}\right)^2}\partial\left(\sum e^{z_k}\right)$$

$$= \partial\left(e^{z_j}\right)\frac{1}{\sum e^{z_k}} - e^{z_j}\frac{1}{\left(\sum e^{z_k}\right)^2}\partial\left(e^{z_j}\right)$$

We have to apply the chain rule again to find derivative of the sum. The sum contains the thing we are differentiating z_i along with all the other z_k which we are not interested in. Those other z_k are constants and can be removed.

Partial Derivatives $\frac{\partial p_j}{\partial z_j}$

$$\frac{\partial p_j}{\partial z_j} = \partial(e^{z_j})\frac{1}{\sum e^{z_k}} - e^{z_j}\frac{1}{(\sum e^{z_k})^2}\partial(e^{z_j})$$

$$= e^{z_j}\frac{1}{\sum e^{z_k}} - e^{z_j}\frac{1}{(\sum e^{z_k})^2}e^{z_j}$$

$$\partial e^z = e^z$$

$$= \underbrace{\frac{e^{z_j}}{\sum e^{z_k}}}_{p} - \underbrace{\frac{(e^{z_j})^2}{(\sum e^{z_k})^2}}_{p^2}$$

$$\frac{\partial p_j}{\partial z_j} = p_j - {p_j}^2$$

$j \neq q$

$$\frac{\partial p_q}{\partial z_{j\neq q}} = \partial(e^{z_j}\frac{1}{\sum e^{z_k}})$$

$$= \partial(e^{z_j})\frac{1}{\sum e^{z_k}} + e^{z_j}\partial(\frac{1}{\sum e^{z_k}})$$

$$= -e^{z_j}\frac{1}{(\sum e^{z_k})^2}\partial(\sum e^{z_k})$$

$$= -e^{z_j}\frac{1}{(\sum e^{z_k})^2}\partial(e^{z_q})$$

$$= -\frac{e^{z_j}e^{z_q}}{(\sum e^{z_k})^2} = -p_j p_q$$

Partial Derivatives $\frac{\partial J}{\partial w_{ij}^{hy}} = \frac{\partial J}{\partial p_j}\frac{\partial p_j}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}^{hy}}$

- So now we have two dervatives

$$\frac{\partial p_j}{\partial z_j} = p_j - {p_j}^2 \qquad \frac{\partial p_j}{\partial z_{q\neq j}} = -p_j p_q$$

- Now we can return to the first two terms of $\frac{\partial J}{\partial w_{ij}^{hy}} = \frac{\partial J}{\partial p_j}\frac{\partial p_j}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}^{hy}}$

$$\frac{\partial J}{\partial p_j}\frac{\partial p_j}{\partial z_j} = \partial(-\sum_{j=1}^{n_y} y_j \log p_j)\frac{\partial p_j}{\partial z_j}$$

$$= -\partial(y_j \log p_j)\frac{\partial p_j}{\partial z_j} - \partial(\sum_{j\neq q} y_q \log p_q)\frac{\partial p_{j\neq q}}{\partial z_j}$$

$$= -\frac{y_j}{p_j}(p_j - {p_j}^2) - \sum_{j\neq q}(\frac{y_q}{p_q}(-p_j p_q))$$

$$= -y_j + y_j p_j + \sum_{j\neq q} y_q p_q = p_j \sum y_j - y_j$$

$$= p_j - y_j$$

Partial Derivatives $\frac{\partial J}{\partial w_{ij}^{hy}} = \frac{\partial J}{\partial p_j} \frac{\partial p_j}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}^{hy}}$

$$\frac{\partial z_j}{\partial w_{ij}^{hy}} = h_i \qquad z_y = hw^{hy}$$

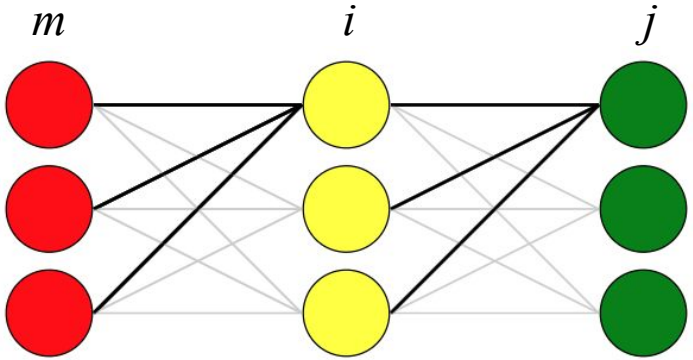
$$\frac{\partial J}{\partial w_{ij}^{hy}} = \left(\frac{\partial J}{\partial p_j} \frac{\partial p_j}{\partial z_j}\right) \frac{\partial z_j}{\partial w_{ij}^{hy}} = (p_j - y_j) h_i = \boxed{h_i e_{yj}}$$

- This result is the same as the intuition
- Note: **softmax** and **cross entropy** were specifically chosen to go together because they have such a nice friendly derivative
  - Because there may be many simpler and less computationally expensive formulas which give a nice probability distribution, but softmax is so popular because of its derivative.
- Also, **tanh** is also a very complicated function with a nice derivative, as is its alternative **sigmoid**
  - The same goes for other versions of log loss and the squared error loss. All are carefully selected for their graceful derivatives.

Partial Propagation $\frac{\partial J}{\partial w_{mi}^{xh}}$

- Backpropagate the error of output layer e_y to hidden layer: the error of hidden layer e_h
- In the following equations
  - Output nodes are indexed by j
  - Hidden nodes are indexed with i
  - Input nodes are indexed by m

$$\frac{\partial J}{\partial w_{mi}^{xh}} = \sum_{j=1}^{n_y} \left(\frac{\partial J}{\partial p_j} \frac{\partial p_j}{\partial z_{yj}} \frac{\partial z_{yj}}{\partial h_i} \frac{\partial h_i}{\partial z_{hi}} \frac{\partial z_{hi}}{\partial w_{mi}^{xh}} \right)$$



Partial Propagation $\frac{\partial J}{\partial w_{mi}^{xh}}$

$$\frac{\partial z_{yj}}{\partial h_i} = w_{ij}^{hy}$$

$$\frac{\partial J}{\partial w_{mi}^{xh}} = \sum_{j=1}^{n_y} (e_{yj} w_{ij}^{hy}) \frac{\partial h_i}{\partial z_{hi}} \frac{\partial z_{hi}}{\partial w_{mi}^{xh}}$$

$$\partial h_i = \tanh(z_{hi}) = \frac{e^{z_{hi}} - e^{-z_{hi}}}{e^{z_{hi}} + e^{-z_{hi}}} = 1 - \tanh^2(z_{hi})$$

$$\frac{\partial z_{hi}}{\partial w_{mi}^{xh}} = x_m$$

$$\frac{\partial J}{\partial w_{mi}^{xh}} = \sum_{j=1}^{n_y} (w_{ij}^{hy} e_{yj})(1 - \tanh^2(z_{hi}))x_m$$

Partial Propagation $\frac{\partial J}{\partial w_{mi}^{xh}}$

$$e_{hi} = \sum_{j=1}^{n_y} \left(w_{ij}^{hy} e_{yj} \right)$$

$$z_{ehi} = e_{hi} \left(1 - \tanh^2 \left(z_{hi} \right) \right)$$

$$\delta w^{xh} = \alpha x^T z_{ehi}$$