16 October, 2016

Neural Networks

Course 3: Gradient Descent and Backpropagation

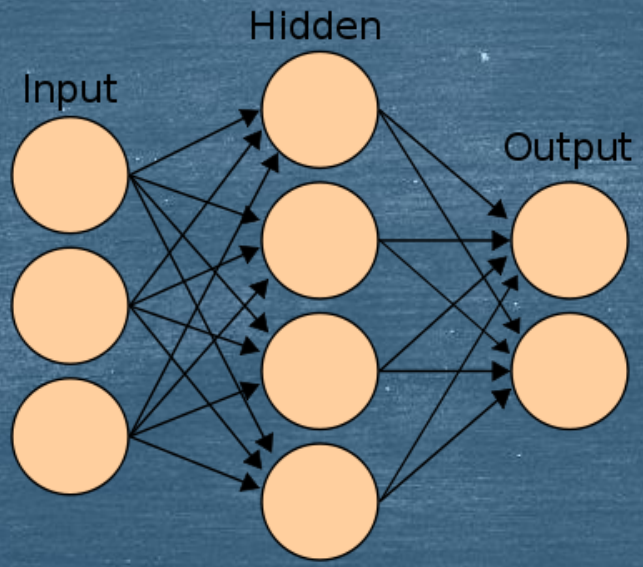
Overview

- Feed Forward Network Architecture
- Gradient Descent
- Backpropagation
- A network to read digits
- Conclusions

Feed Forward Network Architecture

Feed Forward Network Architecture

- Composed of at least 3 layers:
  - The first one is called the input layer
  - The last one is the output layer
  - The ones in the middle are called hidden layers. The hidden layer is composed of hidden units.
  - Each unit is symmetrically connected with all the units in the layer above (there are no loops)



Feed Forward Network Architecture

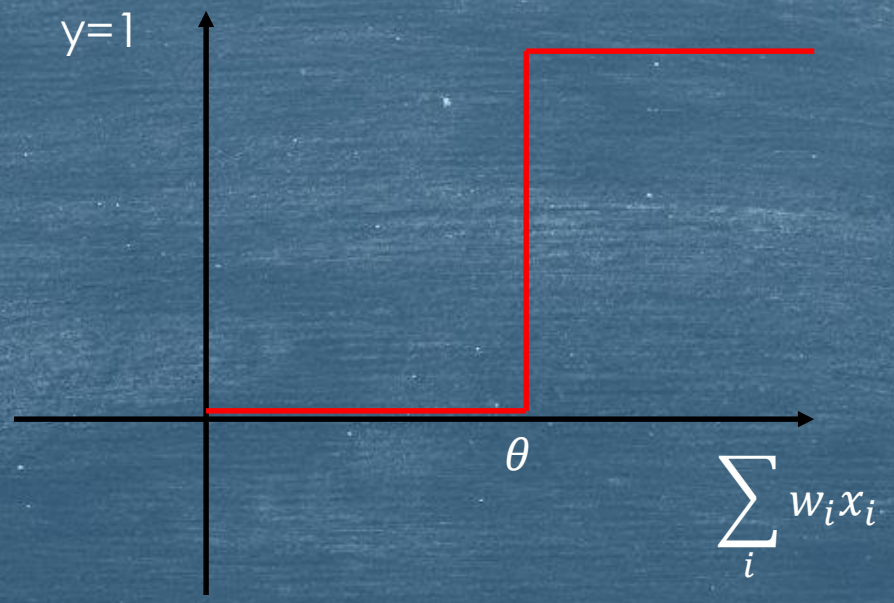
- The neuron in the hidden layer and in the output layer are non-linear neurons. Most of the times are logistic neurons (sigmoid), but can also be tanh, rectified linear units or based on other non-linear function

- This type of network is also called Multi Layer Perceptron (MLP), but this is confusing, since the perceptron is rarely used in this kind of architecture

- Why not use perceptron?

Feed Forward Network Architecture

- Why not use the perceptron
  - Learning is performed by slightly modifying weights and observing the output. However, since the perceptron only outputs 0 or 1 it is possible to update the weights and see no change at all



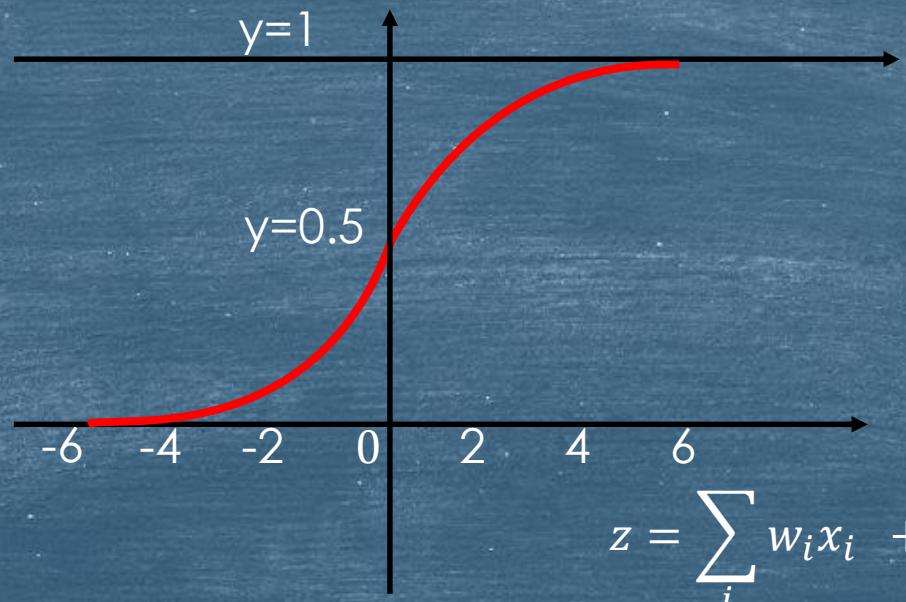
Feed Forward Network Architecture

- A smoother version of the perceptron will be used: the logistic function (sigmoid)

$$y = \sigma(z) = \frac{1}{1+e^{-z}},$$

where z will is the net input computed the neuron:

$$z = \sum_i w_i x_i + b$$



Gradient Descent

Gradient Descent

- Adjust the weights and biases such that to minimize a cost function
- A cost function is a mathematical function that assigns a value (cost) to how bad a sample is classified
- A common used function (that we will also use) is the mean squared error

$$C(w,b) = \frac{1}{2n}\sum_x ||t - y||^2$$

$$C(w,b) = \frac{1}{2n}\sum_x (t - y)^2$$

w=all weights in the network
t = target output vector for input x
$||v||$ = length of the vector

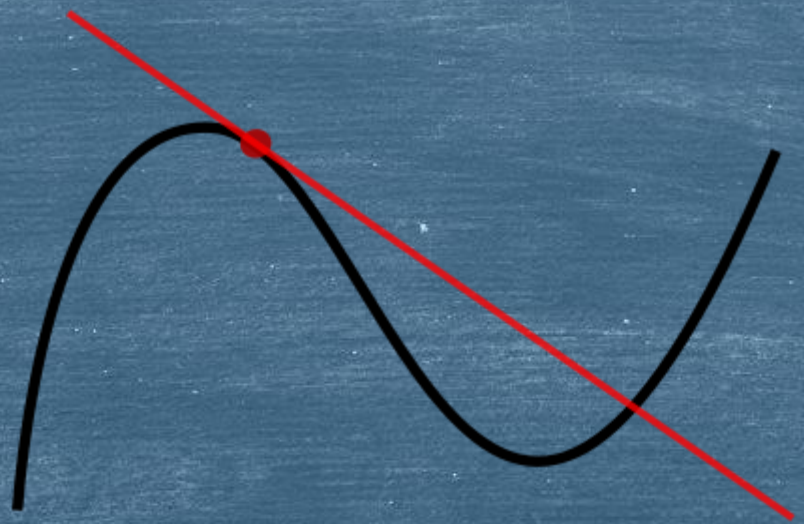
b = all biases in the network
y = output when input is x. y(x)

Gradient Descent

- Why use a cost function? (why not just count the correct outputs)
  - A small update in the weights might not result in a change in the number of correctly classified samples
- Why use Mean Square Error ?
  - We can interpret the formula as being very similar to the Euclidian distance, thus this can be interpreted as minimizing the distance between the target and the output
  - The formula also resembles the one for the variance (which computes how far the elements are from the mean). So, for example in regression, this would be equivalent to reducing how far the elements go from the mean (the target hyperplane)
  - It is continuous and it is easy differentiable (which will be useful later)

Gradient Descent

- What is a gradient?
  - A gradient is just a fancy word for derivate ☺
  - The gradient (first derivative) determines the slope of the tangent of the graph of the function. (points in the direction of the greatest rate of increase)



Gradient Descent

- A function with multiple variables have multiple derivatives:

  f(x,y,z) has 3 derivatives, $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

  $\frac{\partial f}{\partial x}$ means partial derivative and is obtained by differentiating f with respect to x and considering all the other variables (y, z) as constants

- If x changes with Δx, y with Δy and z with Δz then:

  $\Delta f \approx \frac{\partial f}{\partial x}\Delta x + \frac{\partial f}{\partial y}\Delta \mathrm{y} + \frac{\partial f}{\partial z}\Delta \mathrm{z}$

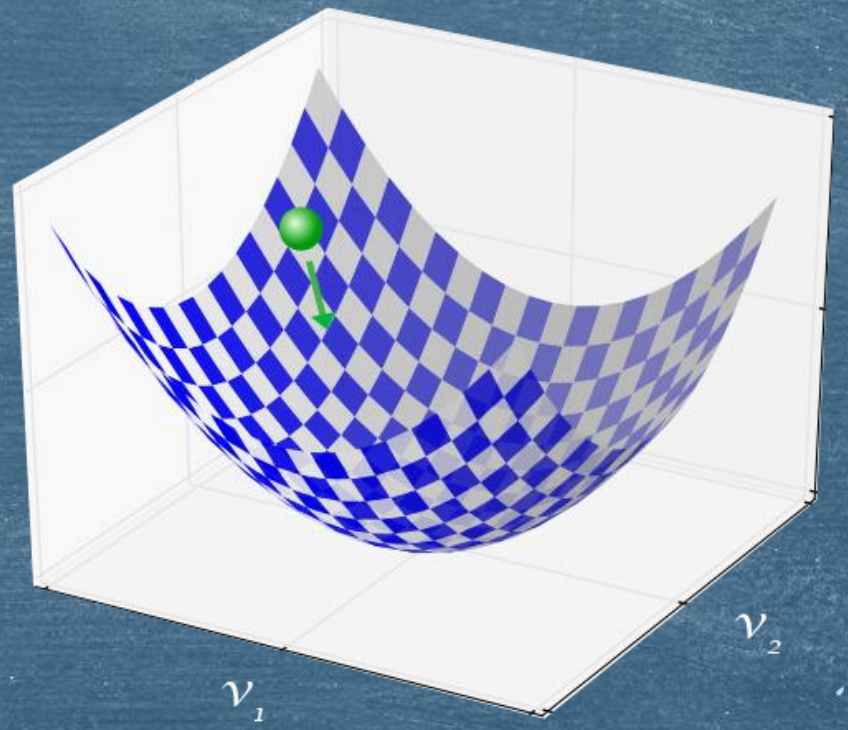
Gradient Descent

- Minimizing the Cost function:

  Suppose the Cost functions has only two variables (v_1 and v_2) and its geometric representation is a quadratic bowl.

  If we move in the direction v_1 by $\Delta v1$ and in the direction v_2 by $\Delta v2$ then

$$\Delta C(v1, v2) \approx \frac{\partial C}{\partial v1}\Delta v1 + \frac{\partial C}{\partial v2}\Delta v2$$



Gradient Descent

- Minimizing the Cost function:

$$\nabla C = \left(\frac{\partial C}{\partial v1}, \frac{\partial C}{\partial v2}\right)$$

and

$$\Delta v = (\Delta v1, \Delta v2)$$

then

$$\Delta C(v1, v2) \approx \nabla C \cdot \Delta \mathrm{v}$$

Since we want to always move downwards (minimizing C function) we want ΔC to be negative. So

$$\Delta \mathrm{v} = -\eta \nabla C$$

Where η is a small number, called learning rate

Gradient Descent

- Minimizing the Cost function:
  - So, by adjusting v1 and v2 such that $\Delta v = -\eta \nabla C$ will always lead to a smaller value for C.
  - Repeating the above step multiple times drives us to a local minimum
  - The learning rate must be small enough to not jump over the minimum
  - Even though we have used just two variables, the same principle can be used for any derivable Cost function of many variables

Gradient Descent

- Example:
  - Performing gradient descent for the Adaline perceptron.

    Adaline Perceptron:

    The activation function (identity function for Adaline):

    $$y = z = wx + b$$

    We will use the mean square error as the cost functions

    $$C(w, b) = \frac{1}{2n} \sum_x (t - y)^2$$

Gradient Descent

- Example:
  - Performing gradient descent for the Adaline perceptron.
    1. Compute the gradient ∇C

$$\nabla C = \left(\frac{dC}{dw}, \frac{dC}{db}\right)$$

Remember the chain rule:

C = a function of the variable y (output)

y = a function of the variable (w)

$$\frac{1}{2n}\sum_x (t-y)^2$$

$$y = wx + b$$

$$\frac{\partial C}{\partial w} = \frac{\partial C}{\partial y} \cdot \frac{\partial y}{\partial w}$$

Gradient Descent

- Example:
  - Performing gradient descent for the Adaline perceptron.
    1. Compute the gradient ∇C

$$C' = \left(\frac{1}{2n}\sum_x (t-y)^2\right)' = \frac{1}{2n} 2 \sum_x (t-y)' = \frac{1}{n}\sum_x (t-y)(-y)'$$

$$\frac{dy}{dw} = \frac{d(wx+b)}{dw} = x$$

$$\frac{dy}{db} = \frac{d(wx+b)}{db} = 1$$

$$\frac{dC}{dw} = -\frac{1}{n}\sum_x (t-y)x$$

$$\frac{dC}{db} = -\frac{1}{n}\sum_x (t-y)$$

Gradient Descent

- Example:
  - Performing gradient descent for the Adaline perceptron.

    2. Choose a learning rate η

    3. For a fixed number of iterations:

adjust w: $w = w - \eta \frac{dC}{dw} = w + \frac{\eta}{n} \sum_x (t - y) x$

adjust b: $b = b - \eta \frac{dC}{db} = b + \frac{\eta}{n} \sum_x \eta (t - y)$

  - This is the same update rule we used for the Adaline in the previous course. The formula looks different since now we are averaging over all samples

Gradient Descent

- We will usually be performing stochastic gradient descent (mini-batch) instead gradient descent.

- The idea is very simple:
  - Choose a subset of smaller size that approximates the ∇C. Use that instead of the real ∇C.
  - Update the weights using the previously computed value
  - Choose another subset from the training data and repeat the previous steps until all the samples have been processed

This speeds up the learning process by a factor of $\frac{size\ of\ training\ data}{size\ of\ minibatch}$

Backpropagation

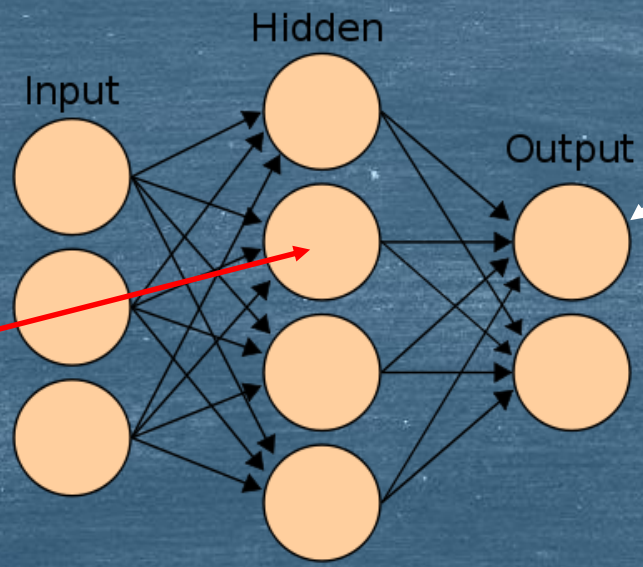
- Using gradient descent we can optimize a perceptron. However, can we train a network with multiple neurons?
  - Yes, but not yet ! ☺
  - Why?

Backpropagation

- Using gradient descent we can optimize a perceptron. However, can we train a network with multiple layers?
  - Yes, but not yet ! ☺
  - Why?

We don't know how much of the error depends on the previous layers.

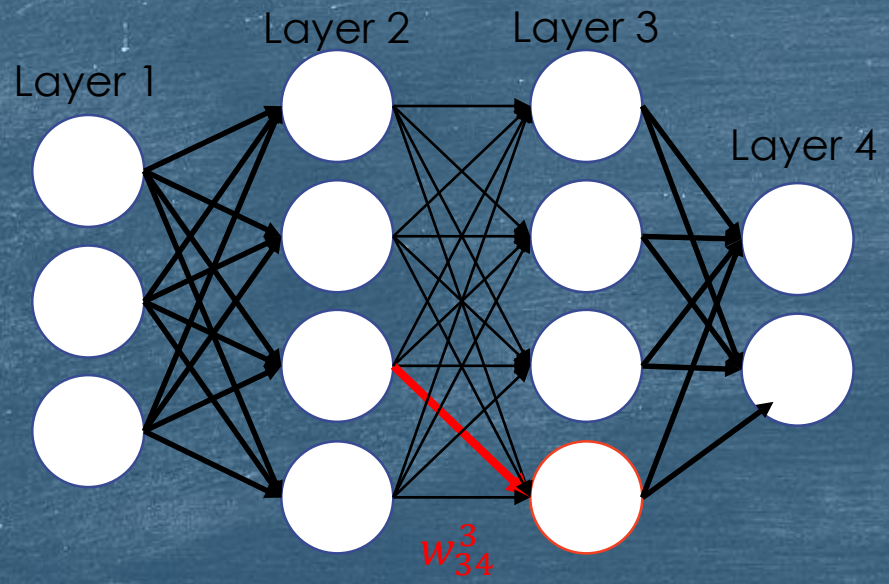
We need to backpropagate the error

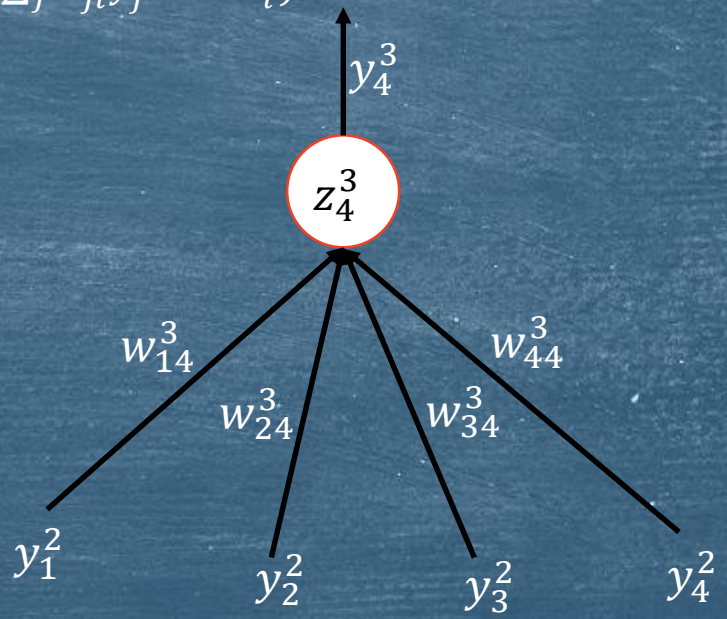


We know the error at the last layers, so we can update the immediate weights that affect that error

Backpropagation

- Some notations:
  - w_{ij}^{l}= the weight from the neuron i from the l – 1 layer to the neuron j in the l layer
  - L = the last layer
  - y_i^l = the activation of neuron i from the l layer. $y_i^l = \sigma(z_i^l)$
  - b_i^l = the bias of neuron i from the l layer
  - z_i^l= the net input for the neuron i from the l layer $(\sum_j w_{ji}^l y_j^{l-1} + b_i^l)$

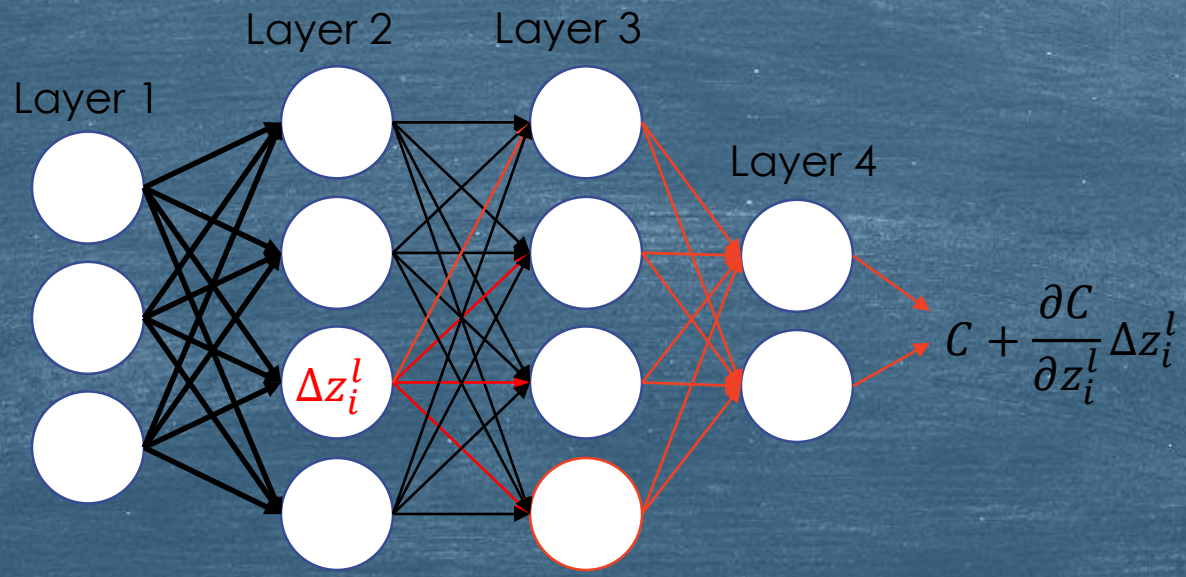




Backpropagation

- We can adjust the error by adjusting the bias and the weights of each neuron i from each layer l. This will change the net input from z_i^l to $(z_i^l + \Delta z_i^l)$ and the activation from $\sigma(z_i^l)$ to $\sigma(z_i^l + \Delta z_i^l)$

- In this case, the cost function will change by $\Delta C = \frac{\partial C}{\partial z_i^l} \Delta z_i^l$



Backpropagation

- We can minimize C by making $\Delta C = \frac{\partial C}{\partial z_i^l} \Delta z_i^l$ negative:
  - Δz_i^l must have an opposite sign to $\frac{\partial C}{\partial z_i^l}$

Considering that Δz_i^l is a small number (since we want to make small changes), the amount by how the error is minimized depends on how large is $\left|\frac{\partial C}{\partial z_i^l}\right|$

If $\left|\frac{\partial C}{\partial z_i^l}\right|$ is close to zero, then the cost can not be further reduced

Thus, we can consider that $\frac{\partial C}{\partial z_i^l}$ represents a measure of the error for the neuron i in the layer l

We will use $\delta_i^l = \frac{\partial C}{\partial z_i^l}$ to represent the error .

Note, that we could have also represented the error in respect to the output y_i^l but the current variant results in using less formulas

Backpropagation

- How backpropagation algorithm works:
- We will compute the error for each neuron at the last layer: δ_i^L

  For each layer l , starting from the last one to the first:

  For each neuron i in the layer l

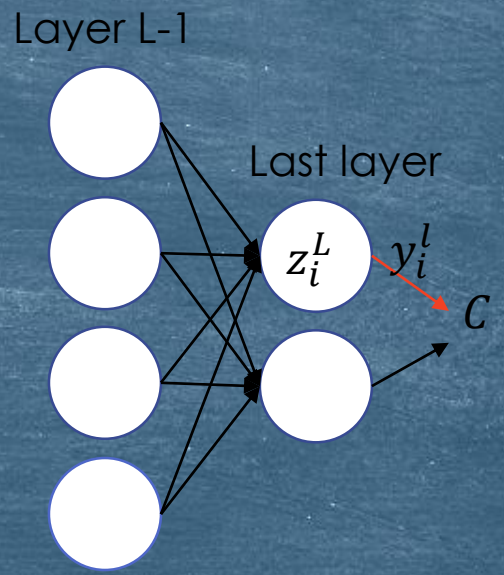
  - We will compute the error : δ_i^l
  - Using this value we will compute $\frac{\partial C}{\partial b_i^l}$ and $\frac{\partial C}{\partial w_{ij}^l}$
  - We will back-propagate the error to the neurons in the previous layer and will repeat the above steps

Backpropagation

- Compute how the cost function depends on the error from the last layer

$$\frac{\partial C}{\partial z_i^L} = \frac{\partial C}{\partial y_i^L} \cdot \frac{\partial y_i^L}{\partial z_i^L} = \frac{\partial C}{\partial y_i^L} \cdot \sigma'(z_i^L)$$

$$\boxed{\delta_i^L = \frac{\partial C}{\partial z_i^L} = \frac{\partial C}{\partial y_i^L} \cdot \sigma'(z_i^L)}$$

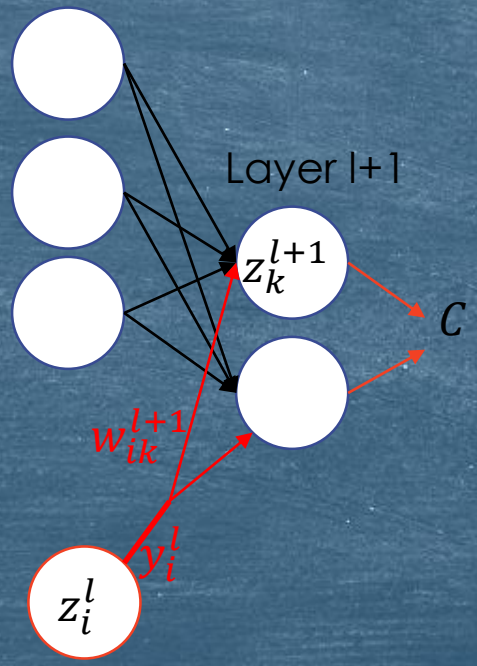


Backpropagation

- Backpropagate the error (write the error in respect to the error in the next layer)

$$\frac{\partial C}{\partial z_i^l} = \frac{\partial C}{\partial y_i^l} \cdot \frac{\partial y_i^l}{\partial z_i^l} = \frac{\partial y_i^l}{\partial z_i^l} \cdot \sum_k \frac{\partial C}{\partial z_k^{l+1}} \cdot \frac{\partial z_k^{l+1}}{\partial y_i^l} = \sigma'(z_i^i) \sum_k \delta_k^{l+1} \cdot w_{ik}^{l+1}$$

$$\boxed{\delta_i^l = \frac{\partial C}{\partial z_i^l} = \sigma'(z_i^l) \sum_k \delta_i^{l+1} \cdot w_{ik}^{l+1}}$$

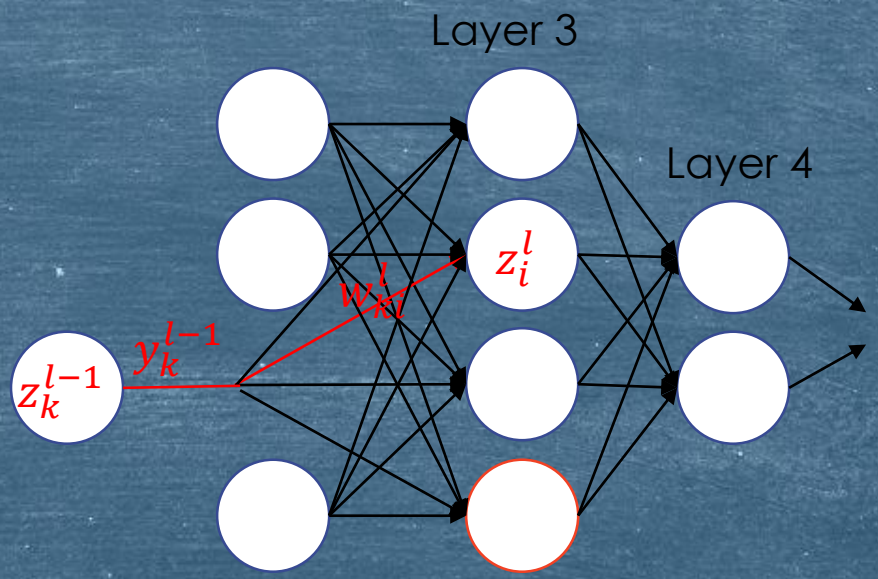


Backpropagation

- Compute how the cost function depends on a weight

$$\frac{\partial C}{\partial w_{ki}^l} = \frac{\partial C}{\partial z_i^l} \cdot \frac{\partial z_i^l}{\partial w_{ki}^l} = \delta_i^l \cdot y_k^{l-1}$$

$$\boxed{\frac{\partial C}{\partial w_{ki}^l} = \delta_i^l \cdot y_k^{l-1}}$$

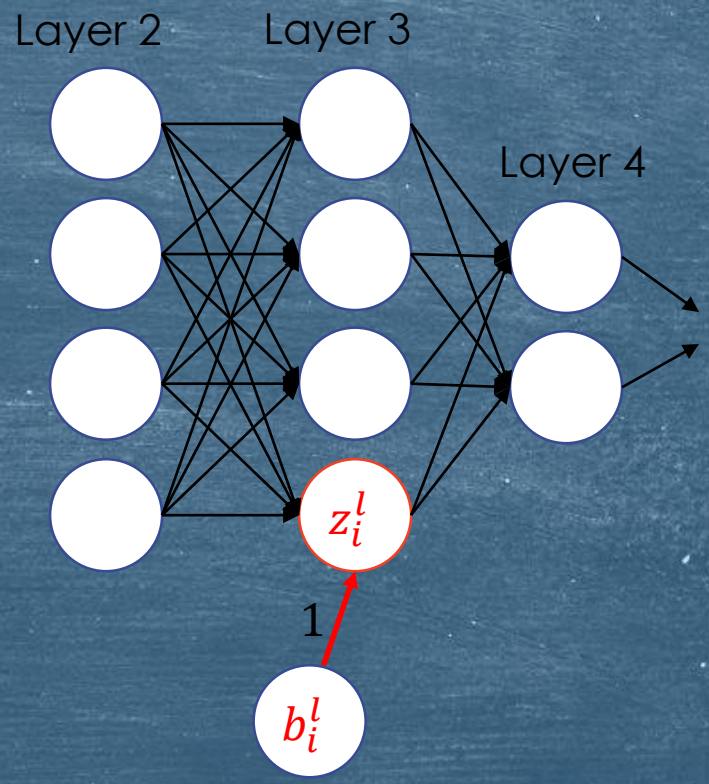


Backpropagation

- Compute how the cost function depends on a bias

$$\frac{\partial C}{\partial b_i^l} = \frac{\partial C}{\partial z_i^l} \cdot \frac{\partial z_i^l}{\partial b_i^l} = \delta_i^l \cdot$$

$$\boxed{\frac{\partial C}{\partial b_i^l} = \delta_i^l}$$



Backpropagation

- Doing the math for the σ'

$$\frac{d\sigma'}{dz} = \left(\frac{1}{1+e^{-z}}\right)' = ((1+e^{-z})^{-1})' = -1(1+e^{-z})'(1+e^{-z})^{-2} =$$

$$= -\frac{1}{(1+e^{-z})^2}(1+e^{-z})' = -\frac{(-z)'e^{-z}}{(1+e^{-z})^2} = \frac{e^{-z}}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}} \cdot \frac{e^{-z}}{1+e^{-z}} =$$

$$= \sigma(z) \cdot \left(\frac{1+e^{-z}}{1+e^{-z}} - \frac{1}{1+e^{-z}}\right) = \sigma(z) \cdot \big(1-\sigma(z)\big)$$

$$\sigma'(z_i^l) = y_i^l(1-y_i^l)$$

Backpropagation

- Doing the math for δ_i^L

$$\frac{\partial C}{\partial y_i^L} \cdot \sigma'(z_i^L) = y_i^L(1 - y_i^L) \cdot \frac{d\left(\sum_j \frac{1}{2}(t_j - y_j^L)^2\right)}{dy_i^L} = y_i^L(1 - y_i^L)(t_i - y_i^L)' =$$
$$= y_i^L(1 - y_i^L)(t_i - y_i^L)(-1) = y_i^L(1 - y_i^L)(y_i^L - t_i)$$

$$\boxed{\delta_i^L = y_i^L(1 - y_i^L)(y_i^L - t_i)}$$

Backpropagation

- Putting it all together

0. Compute the error for the final layer:

$$\delta_i^L = y_i^L(1 - y_i^L)(y_i^L - t_i)$$

Process the layer below:

1. Compute the error for the previous layer:

$$\delta_i^l = y_i^l(1 - y_i^l)\sum_k \delta_k^{l+1} \cdot w_{ik}^{l+1}$$

2. Compute the gradient for the weights in the current layer:

$$\frac{\partial C}{\partial w_{ij}} = \delta_i^l y^{l-1}$$

3. Compute the gradient for the biases in the current layers:

$$\frac{\partial C}{\partial b_i} = \delta_i^l$$

Repeat until we reach the input layer

A network to read digits

A network to read digits

- We will train a feed forward network, using the backpropagation algorithm that can recognize handwritten digits

- The dataset can be downloaded from here: (: http://deeplearning.net/data/mnist/mnist.pkl.gz)

A network to read digits

- Each image is 28x28 in size and is represented as a vector of 784 pixels, each pixel having an intensity.

- We will use a network of 3 layers:
  - 784 for the input layer
  - 36 for the hidden layer
  - 10 for the output layer

  Each neuron from the output layer will activate for a certain digit. The outputted digit of the network will be given by the output neuron that has the highest confidence. (the largest outputted activation)

A network to read digits

- Training info:
  - The weights were initialized with random values generated according to the normal standard distribution
  - The learning rate used is $\eta = 3.0$
  - Learning is performed using SGD with minibatch size of 10
  - Training is performed for 30 iterations
  - Training data consists of 50000 images.
  - Test data consists of 10000 images
- Results:
  - ANN identifies approx. 95% of the tested images
  - A network made of perceptron (10), detects approx. 83%

(watch demo)

Bibliography

http://neuralnetworksanddeeplearning.com/

Chris Bishop, “Neural Network for Pattern Recognition”

http://sebastianraschka.com/Articles/2015_singlelayer_neurons.html

http://deeplearning.net/

Questions & Discussion