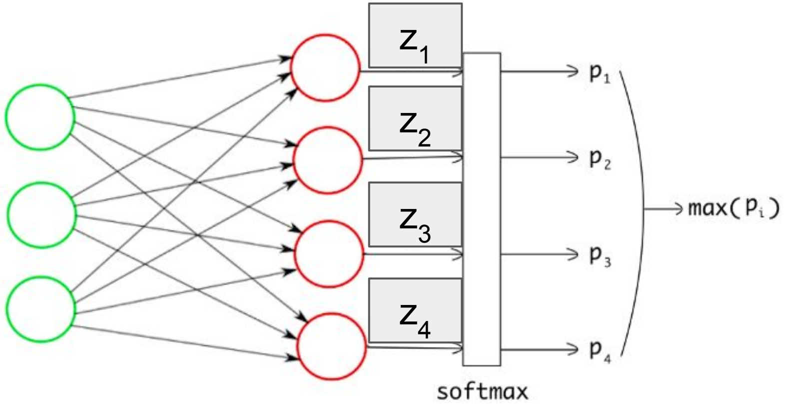
EE655: Computer Vision & Deep Learning

Lecture 08

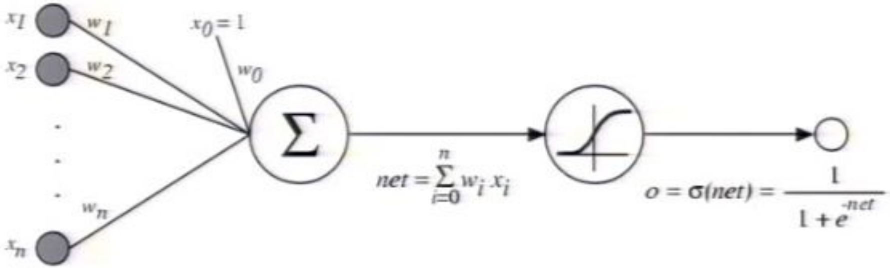
Koteswar Rao Jerripothula, PhD
Department of Electrical Engineering
IIT Kanpur

Multi-class Classification v/s Multi-label Classification



$$\boxed{p_i} = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Scores from the last layer are passed through a **softmax** layer. The softmax layer converts the score into **probability** values.



We use the **sigmoid** activation function in the final layer. Sigmoid converts each score of the final node between 0 to 1 independent of what the other scores are.

Loss Functions

Multi-class Classification: Categorical Cross-entropy Loss

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

- M - number of classes (dog, cat, fish)
- log - the natural log
- y - binary indicator (0 or 1) if class label c is the correct classification for observation o
- p - predicted probability observation o is of class c

Multi-label Classification: Binary Cross-entropy Loss

$$\text{Loss} = -\frac{1}{\substack{\text{output} \\ \text{size}}} \sum_{i=1}^{\substack{\text{output} \\ \text{size}}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i)$$

where $\hat{y}_i$ is the i-th scalar value in the model output, y_i is the corresponding target value, and `output size` is the number of scalar values in the model output.

Calculate the output probabilities if following are passed to the softmax layer

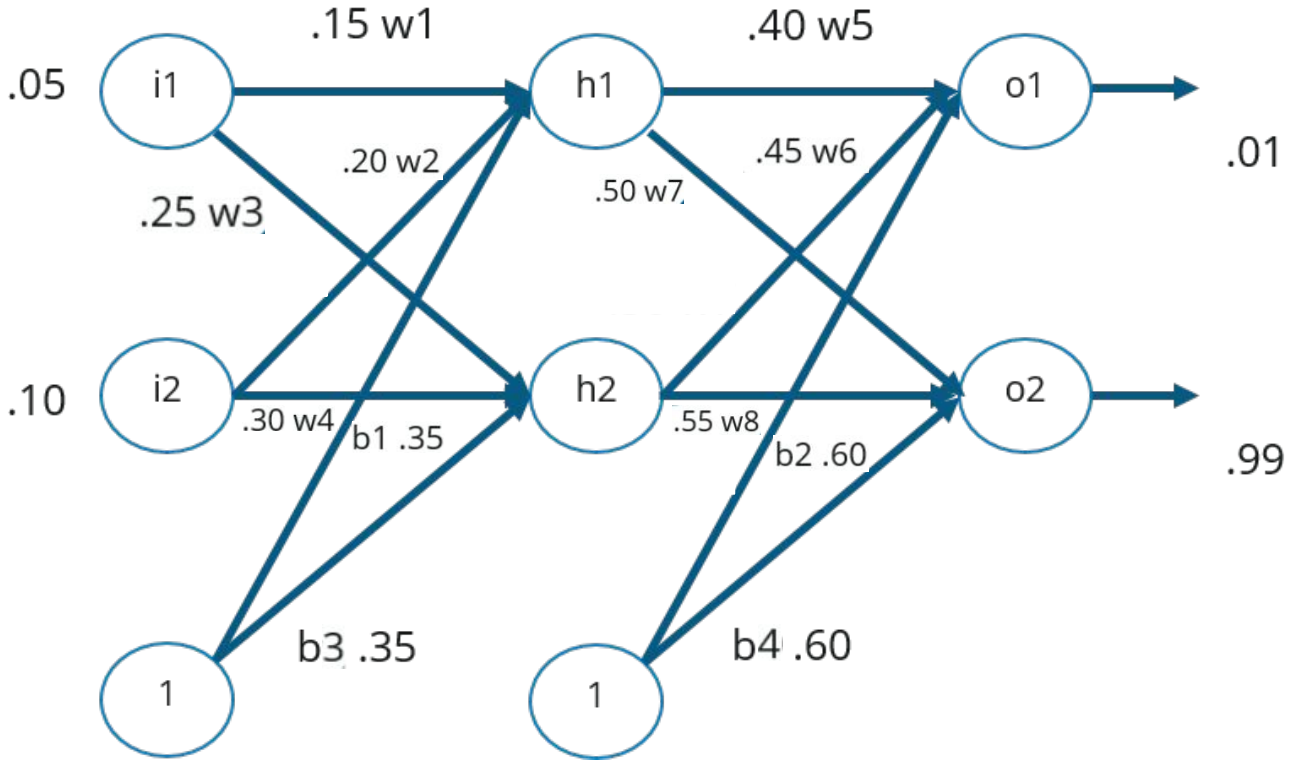
$$\begin{bmatrix} 8 \\ 5 \\ 0 \end{bmatrix}$$

Now, compute the categorical entropy loss if the target vector is the following:

[1 0 0]

Let's see how weights can be updated

A regression problem



Forward Propagation through Hidden Layer

Net Input For h1:

net h1 = w1*i1 + w2*i2 + b1*1 → net h1 = 0.15*0.05 + 0.2*0.1 + 0.35*1 = 0.3775

Output Of h1:

out h1 = $1/1+e^{-\text{net h1}}$ → $1/1+e^{.3775}$ = 0.593269992

Output Of h2:

out h2 = 0.596884378

Forward Propagation through Output layers

Output For o1:

net o1 = w5*out h1 + w6*out h2 + b2*1 → 0.4*0.593269992 + 0.45*0.596884378 + 0.6*1 = 1.105905967

Out o1 = $1/1+e^{-net\ o1}$ → $1/1+e^{-1.105905967}$ = 0.75136507

Output For o2:

Out o2 = 0.772928465

Error Computation

Error For o1:

$E_{o1} = \Sigma 1/2(target - output)^2$ → $½ (0.01 - 0.75136507)^2 = 0.274811083$

Error For o2:

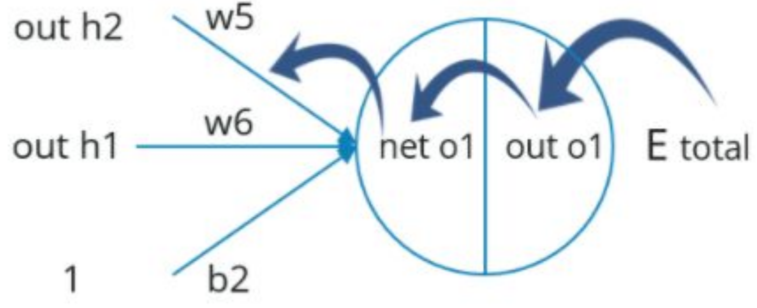
$E_{o2} = 0.023560026$

Total Error:

$E_{total} = E_{o1} + E_{o2}$ → $0.274811083 + 0.023560026 = 0.298371109$

Chain Rule

$$\frac{\delta Etotal}{\delta w5} = \frac{\delta Etotal}{\delta out\ o1} * \frac{\delta out\ o1}{\delta net\ o1} * \frac{\delta net\ o1}{\delta w5}$$



Back-propagation leverages chain-rule to compute gradients gradually from backwards.

$$E_{total} = 1/2(\text{target o1} - \text{out o1})^2 + 1/2(\text{target o2} - \text{out o2})^2$$

$$\frac{\delta Etotal}{\delta out\ o1} = -(\text{target o1} - \text{out o1}) = -(0.01 - 0.75136507) = 0.74136507$$

$$\text{out o1} = 1/1{+}e^{-neto1}$$

$$\frac{\delta out\ o1}{\delta net\ o1} = \text{out o1 } (1 - \text{out o1}) = 0.75136507\ (1 - 0.75136507) = 0.186815602$$

$$\text{net o1} = \text{w5} * \text{out h1} + \text{w6} * \text{out h2} + \text{b2} * 1$$

$$\frac{\delta net\ o1}{\delta w5} = 1 * \text{out h1 } w5^{(1-1)} + 0 + 0 = 0.593269992$$

$$\frac{\delta Etotal}{\delta w5} = \frac{\delta Etotal}{\delta out\ o1} * \frac{\delta out\ o1}{\delta net\ o1} * \frac{\delta net\ o1}{\delta w5} \rightarrow 0.082167041$$

Gradient Descent: a way to update the weights, requiring gradients

$$w5^{+} = w5 - n\frac{\delta Etotal}{\delta w5} \rightarrow w5^{+} = 0.4 - 0.5 * 0.082167041$$

Updated w5 $\rightarrow$ 0.35891648

Like this all the weights can be updated

Back Propagation

In Fully Connected Layers

Notations

$n^{[l]}$ - The number of neurons in the layer l.

$w^{[l]}$ - The weight matrix associated with the layers l and l-1, of the size ($n^{[l]}$ x $n^{[l-1]}$).
$w_{jk}^{[l]}$ refers to the weight associated with the neuron j in the layer l and the neuron k in the layer l-1.

$b^{[l]}$ - The vector of biases associated with the layer l, of the size ($n^{[l]}$ x 1).

$a^{[l]}$ - The vector of activations of the neurons in the layer l, of the size ($n^{[l]}$ x 1).

$z^{[l]}$ - The vector of the weighted output of the neurons in the layer l, of the size ($n^{[l]}$ x 1).

$g^{[l]}$ - The activation function applied to the output of the neurons in the layer l, $a^{[l]} = g^{[l]}(z^{[l]})$.

Computations happening in a layer

$$z^{[l]} = w^{[l]}.a^{[l-1]} + b^{[l]}$$

$$\mathrm{a}^{[l]} = g^{[l]}(z^{[l]})$$

Cost and how to use it to update the weights

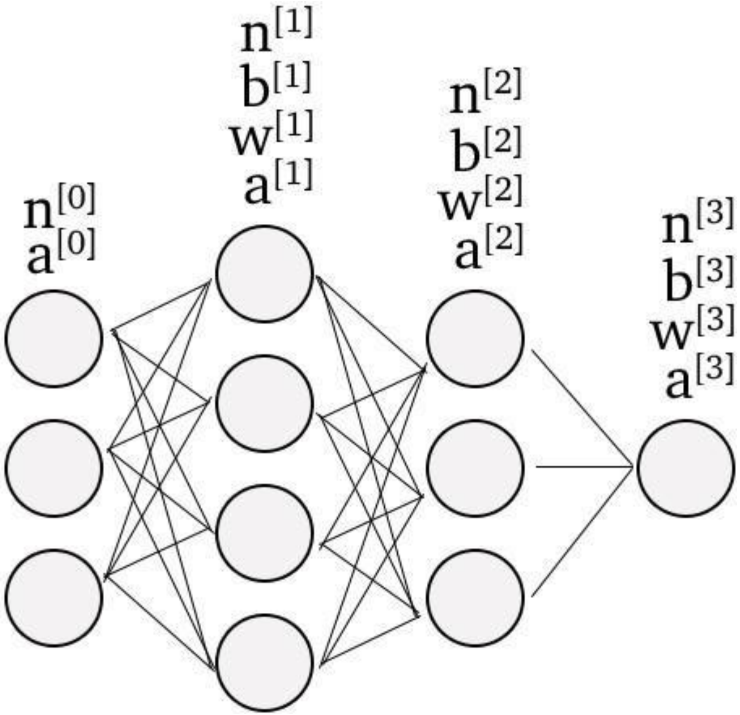
$$C = -(y\log \hat{y} + (1 - y)\log(1 - \hat{y}))$$

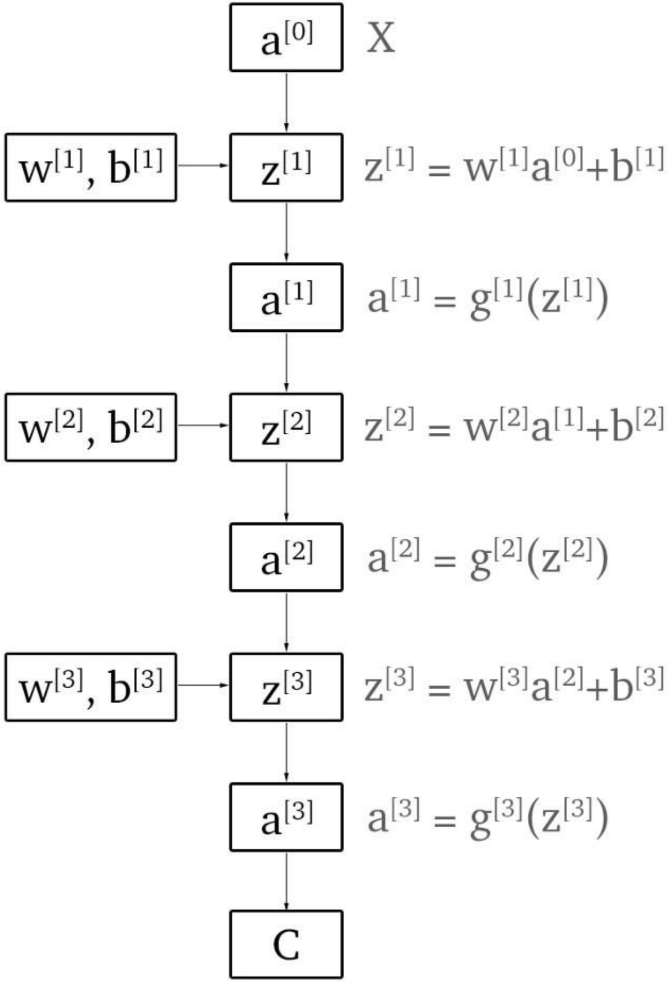
Gradient Descent Equations

$$w^{[l]} = w^{[l]} - \alpha \frac{\partial C}{\partial w^{[l]}}$$

$$b^{[l]} = b^{[l]} - \alpha \frac{\partial C}{\partial b^{[l]}}$$

Forward pass





$$C = -(y\log a^{[3]} + (1 - y)\log(1 - a^{[3]}))$$

For any layer ‘l’

dz/dwl can be computed easily because of weighted sum relationship. Similarly, dz/dbl

However, there is no direct relationship between C and zl to compute dC/dzl

How to compute it then?

$$\frac{\partial C}{\partial w^{[l]}} = \frac{\partial C}{\partial z^{[l]}} \cdot \frac{\partial z^{[l]}}{\partial w^{[l]}}$$

$$\frac{\partial C}{\partial b^{[l]}} = \frac{\partial C}{\partial z^{[l]}} \cdot \frac{\partial z^{[l]}}{\partial b^{[l]}}$$

Let’s open up dC/dzl a bit more using the chain rule

The idea is to progressively use the previous layer’s dC/dzl while coming backwards from the last layer, whose dC/dzl can be easily computed.

$$\frac{\partial C}{\partial z^{[l]}} = \frac{\partial C}{\partial z^{[l+1]}} \cdot \frac{\partial z^{[l+1]}}{\partial a^{[l]}} \cdot \frac{\partial a^{[l]}}{\partial z^{[l]}}$$

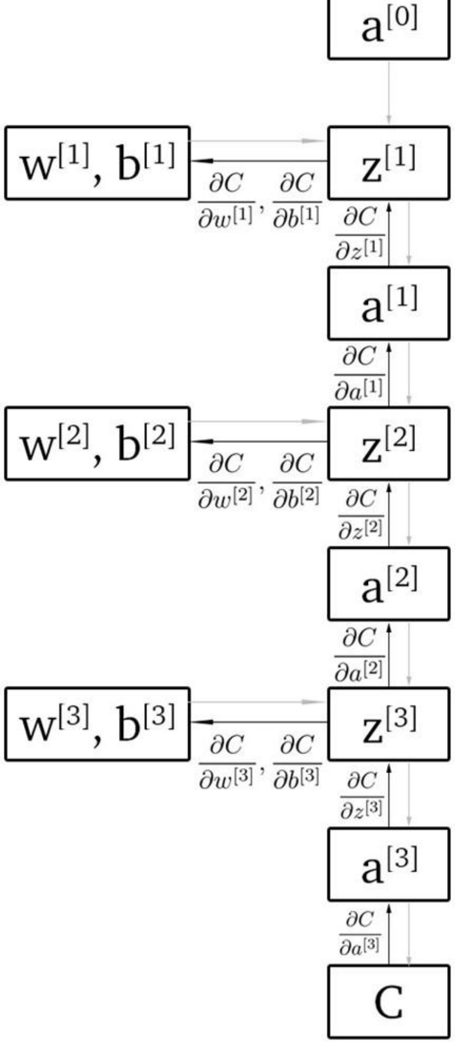
Other terms can be computed because of weighted sum and activation function relationships

Backward Propagation

At any layer, we need dC/dz to compute dC/dw & dC/db

There is relationship between dC/dz of current layer and dC/dz of previous layer while moving backwards.

Since last layer's dC/dz can be computed, we can get the dC/dz of other layers too by exploiting the relationship between the subsequent layers just mentioned.



Computation of other terms

$$z^{[l+1]} = w^{[l+1]}.a^{[l]} + b^{[l+1]}$$

$$\frac{\partial z^{[l+1]}}{\partial a^{[l]}} = \frac{\partial}{\partial a^{[l]}}(w^{[l+1]}.a^{[l]} + b^{[l+1]})$$

$$= w^{[l+1]}$$

$$\therefore \frac{\partial z^{[l+1]}}{\partial a^{[l]}} = w^{[l+1]}$$

also,

$$\frac{\partial a^{[l]}}{\partial z^{[l]}} = g^{[l]\prime}(z^{[l]})$$

$$\frac{\partial a^{[l]}}{\partial z^{[l]}} = \sigma'(z^{[l]})$$

Assuming sigmoid activation

Final Formula

$$\boxed{\frac{\partial C}{\partial z^{[l]}} = (w^{[l+1]^T} . \frac{\partial C}{\partial z^{[l+1]}}) .* \sigma'(z^{[l]})}$$

. (dot) denotes matrix multiplication
.* denotes element-wise multiplication

We need to store all z's in addition to w's and b's