Neural Networks for Computing

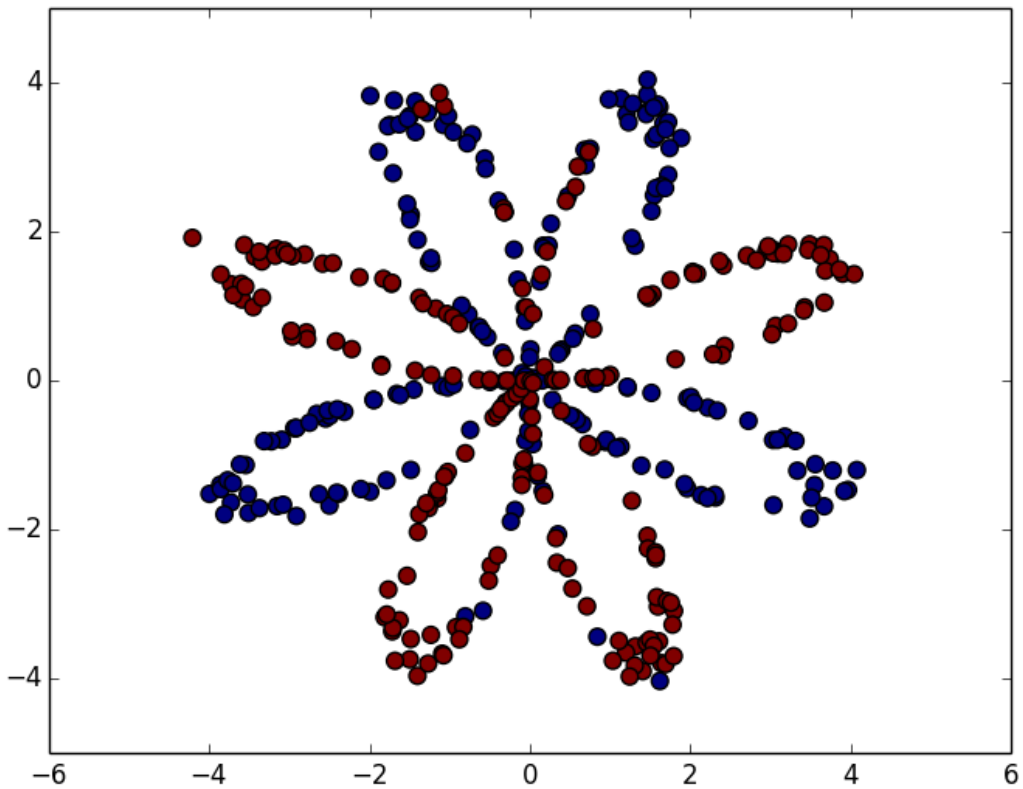
Multilayer Perceptron

Hikmat Farhat

September 28, 2020

Need for multilayers

- We have seen that using a single perceptron can learn to recognize ships
- In fact we get close to 99 accuracy on the training data and about 83 percent on the test data
- But this model is too simple to work in general because it is trying to separate the data using a plane
- In fact it has very poor performance on a 2-dimensional data set

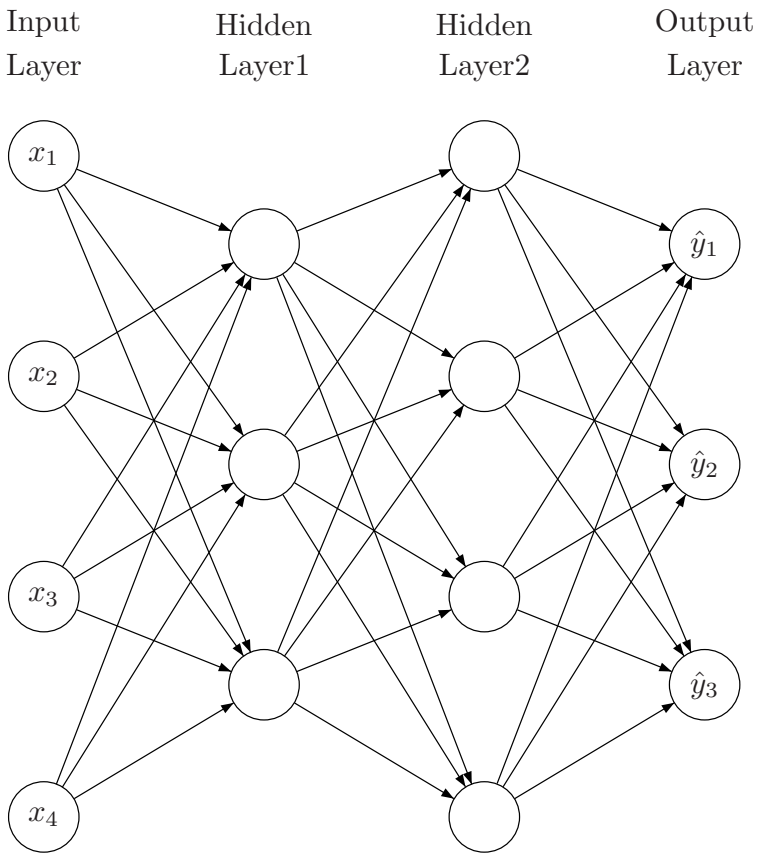

4
2
0
−2
−4
−6
−4
−2
0
2
4
6

Multilayer Perceptron

- A multilayer perceptron (MLP) is a collection of perceptrons organized in layers.
- While the connections between nodes can in principle be arbitrary, usually they are organized in layers.
- Each node feeds its output to other nodes.
- When there is a loop in the graph,i.e. a node output connects to an input node we call the network a *recurrent network*.
- When no such loop exists we call it a *feedforward network*.

Example

- The figure below shows an example of a feedforward network with two hidden layers.
- The number of layers and the number of nodes in a given layer are variables and depend on the application.



Feedforward Networks

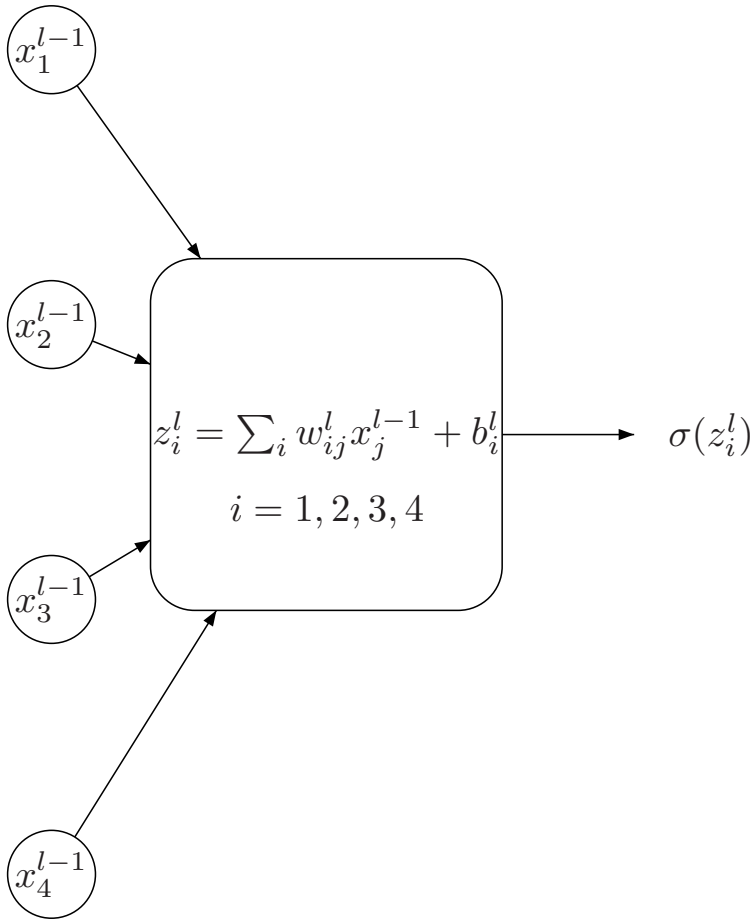
- Mathematically a feedforward network has inputs $x_1, \ldots, x_m$ and outputs $\hat{y}_1, \ldots, \hat{y}_k$.
- For the last(output) layer, L, we have $a_i^L = \hat{y}_i$.
- Each hidden or output node has a value determined by the inputs to the node and weights as we have discussed previously
- Let a_i^l be the output of node i in layer l then

$$a_i^l = \sigma\left(\sum_i w_{ij}^{l-1} a_j^{l-1} + b_i^{l-1}\right)$$

- Where w_{ij}^l and b_i^l are the weights and bias for layer l and σ is a bounded nondecreasing nonlinear function.

Computation at a Node

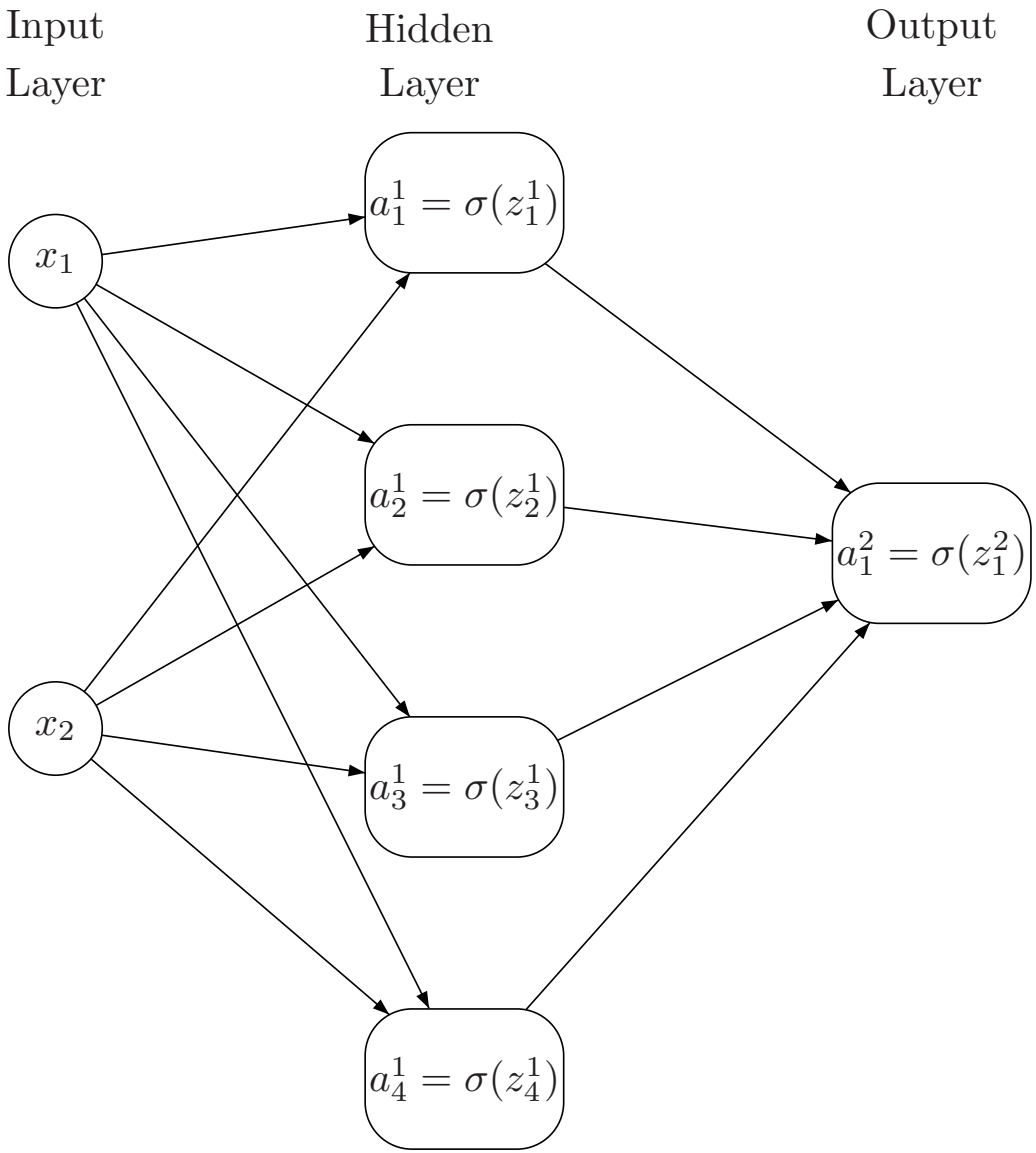
- Each node, regardless of which layer it is in has a set of inputs, say $x_1, \ldots, x_k$. In essence it is very similar to the perceptron that we have studied before.



Shallow Network Example

- We give an example with a single hidden layer (shallow).
- By convention the input layer is not counted and we say it is 2-layer network.
- The network has n_x inputs
- The hidden layer has n_h nodes
- The output layer has 1 node
- The label of every node denotes its output
- The superscript denotes the layer:e.g. a_2^1 is the output of the second node in layer 1

- Example shallow network for $n_x = 2$ and $n_h = 4$



Example

- The node computations in layer one can be written as:

$$
\begin{aligned}
z_1^1 &= w_{11}^0 * x_1 + w_{12}^0 * x_2 \\
z_2^1 &= w_{21}^0 * x_1 + w_{22}^0 * x_2 \\
z_3^1 &= w_{31}^0 * x_1 + w_{32}^0 * x_2 \\
z_4^1 &= w_{41}^0 * x_1 + w_{42}^0 * x_2
\end{aligned}
$$

- Or in vector form

$$
\begin{bmatrix} z_1^1 \\ z_2^1 \\ z_3^1 \\ z_4^1 \end{bmatrix} = \begin{bmatrix} w_{11}^0 & w_{12}^0 \\ w_{21}^0 & w_{22}^0 \\ w_{31}^0 & w_{32}^0 \\ w_{41}^0 & w_{42}^0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}
$$

- and

$$\begin{bmatrix} a_1^1 \\ a_2^1 \\ a_3^1 \\ a_4^1 \end{bmatrix} = \begin{bmatrix} \sigma(z_1^1) \\ \sigma(z_2^1) \\ \sigma(z_3^1) \\ \sigma(z_4^1) \end{bmatrix}$$

- Similarly

$$\hat{y} = a_1^2 = \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 & w_{14}^1 \end{bmatrix} \cdot \begin{bmatrix} a_1^1 \\ a_2^1 \\ a_3^1 \\ a_4^1 \end{bmatrix}$$

Computation on multiple samples

- We will write the computational equations to include m samples. All sample dependent quantities such as a, z, x have a single subscript
- We use the second subscript to denote the sample. For example
- a^i_{jk} is the output of j^{th} node of the i^{th} layer for the k^{th} sample.
- Similarly, $x_{jk} = a^0_{jk}$ is the j^{th} input in the k^{th} sample.
- Using this notation we include all the samples in a single vector operation.
- But first an example.

Example

- Using the subscript s for sample s we have, for **every** $1 \leq s \leq m$

$$\begin{bmatrix} z_{1s}^1 \\ z_{2s}^1 \\ z_{3s}^1 \\ z_{4s}^1 \end{bmatrix} = \begin{bmatrix} w_{11}^0 & w_{12}^0 \\ w_{21}^0 & w_{22}^0 \\ w_{31}^0 & w_{32}^0 \\ w_{41}^0 & w_{42}^0 \end{bmatrix} \cdot \begin{bmatrix} x_{1s} \\ x_{2s} \end{bmatrix}$$

- We can column-stack all the samples together

$$\begin{bmatrix} z_{11}^1 & \cdots & z_{1m}^1 \\ z_{21}^1 & \cdots & z_{2m}^1 \\ z_{31}^1 & \cdots & z_{3m}^1 \\ z_{41}^1 & \cdots & z_{4m}^1 \end{bmatrix} = \begin{bmatrix} w_{11}^0 & w_{12}^0 \\ w_{21}^0 & w_{22}^0 \\ w_{31}^0 & w_{32}^0 \\ w_{41}^0 & w_{42}^0 \end{bmatrix} \cdot \begin{bmatrix} x_{11} & \cdots & x_{1m} \\ x_{21} & \cdots & x_{2m} \end{bmatrix}$$

Vectorization

- In vector form we can write (also by letting $X = A^0$)

$$
\begin{aligned}
Z^1 &= W^0 \cdot A^0 + B^0 \\
A^1 &= \sigma(Z^1) \\
Z^2 &= W^1 \cdot A^1 + B^1 \\
A^2 &= \sigma(Z^2)
\end{aligned}
$$

Why non-linear activation

- The reason that we use a non-linear activation function is that if we choose a linear (e.g. identity) then no matter how many layers we have in the network it would be equivalent to the perceptron model.
- From the previous example if we set $A^i = f(Z^i) = Z^i$ we get

$$Z^1 = W^0 \cdot X + B^0$$
$$A^1 = f(Z^1) = Z^1$$
$$Z^2 = W^1 \cdot A^1 + B^1$$
$$A^2 = f(Z^2) = Z^2$$

- Replacing A^1 by its value we get

$$\begin{aligned}\hat{y} = A^2 = Z^2 &= W^1 \cdot (W^0 \cdot X + B^0) + B^1 \\ &= (W^1 \cdot W^1) \cdot X + (W^1 \cdot B^0 + B^1) \\ &= W \cdot X + B\end{aligned}$$

Gradient descent for shallow network

- To do gradient descent for the shallow network we need to compute the derivatives of the cost function with respect to the parameters
- recall that the loss is given by the cross-entropy function:

$$\mathcal{L} = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

- To do the optimization we use gradient descent where the parameters are updated in the opposite direction of the derivative of the error function

$$w = w - \alpha \frac{\partial E}{\partial w}$$
$$b = b - \alpha \frac{\partial E}{\partial b}$$

- We will compute the derivatives with respect to the loss then average over all samples.

Derivative with respect to b^1

$$\frac{\partial \mathcal{L}}{\partial b^1} = \frac{\partial \mathcal{L}}{\partial a^2}\frac{\partial a^2}{\partial b^1}$$

- From previous lecture we know that(note that $a^2 = \hat{y}$)

$$\frac{\partial \mathcal{L}}{\partial a^2} = \frac{-y}{a^2} + \frac{1-y}{1-a^2}$$

- And since

$$\begin{aligned}\frac{\partial a^2}{\partial b^1} &= \frac{\partial a^2}{\partial z^2}\frac{\partial z^2}{\partial b^1} \\ &= a^2(1-a^2)\end{aligned}$$

- Thus

$$\frac{\partial \mathcal{L}}{\partial b^1} = (a^2 - y)$$

- And the average over m samples becomes

$$db^1 = \frac{1}{m}\sum_{s=1}^{m}(a_s^2 - y_s) \tag{1}$$

Derivative with respect to w^1

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_i^1} &= \frac{\partial \mathcal{L}}{\partial a^2}\frac{\partial a^2}{\partial w_i^1} \\ &= \frac{\partial \mathcal{L}}{\partial a^2}\frac{\partial a^2}{\partial z^2}\frac{\partial z^2}{\partial w_i^1} = (a^2 - y)\frac{\partial z^2}{\partial w_i^1} \\ &= (a^2 - y)a_i^1\end{aligned}$$

- Averaging over m samples we get

$$dw_i^1 = \frac{1}{m}\sum_{s=1}^{m}(a_s^2 - y_s)a_{is}^1$$

$$dw^1 = \frac{1}{m}(A^2 - Y)\cdot {A^1}^T \tag{2}$$

Derivative with respect to w^0

$$\frac{\partial \mathcal{L}}{w_{ij}^0} = \frac{\partial \mathcal{L}}{\partial a^2}\frac{\partial a^2}{\partial z^2}\frac{\partial z^2}{\partial w_{ij}^0}$$
$$= (a^2 - y)\frac{\partial z^2}{\partial w_{ij}^0}$$

$$\frac{\partial z^2}{\partial w_{ij}^0} = \frac{\partial}{\partial w_{ij}^0}\left[\sum_k w_k^1 a_k^1 + b^1\right]$$
$$= \sum_k w_k^1 \frac{\partial \sigma(z_k^1)}{\partial w_{ij}^0}$$
$$= \sum_k w_k^1 \sigma'_k \frac{\partial z_k^1}{\partial w_{ij}^0}$$

- But

$$z_k^1 = \sum_p w_{kp}^0 x_p + b_k^0$$

$$\frac{\partial z_k^1}{\partial w_{ij}^0} = \delta_{ki} x_j$$

- replacing in the expression for $\frac{\partial z^2}{\partial w_{ij}^0}$ we get

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^0} = (a^2 - y) w_i^1 \sigma_i' x_j$$

- Averaging over m samples we get

$$dw_{ij}^{0} = \frac{1}{m}\sum_{s}(a_{s}^{2} - y_{s})w_{i}^{1}\sigma_{is}'x_{js}$$

$$dw^{0} = \frac{1}{m}\left[\left(w^{1\,T}\cdot(A^{2} - Y)\right) * \sigma'\right]\cdot X^{T}$$

Derivative with respect to b^0

$$\frac{\partial \mathcal{L}}{\partial b_i^0} = \frac{\partial \mathcal{L}}{\partial a^2}\frac{\partial a^2}{\partial b_i^0} = \frac{\partial \mathcal{L}}{\partial a^2}\frac{\partial a^2}{\partial z^2}\frac{\partial z^2}{\partial b_i^0}$$

$$= (a^2 - y)\frac{\partial}{\partial b_i^0}\left[\sum_k w_k^1 a_k^1 + b^1\right]$$

$$= (a^2 - y)\sum_k w_k^1 \frac{\partial a_k^1}{\partial b_i^0} = (a^2 - y)\sum_k w_k^1 \sigma'_k \frac{\partial z_k^1}{\partial b_i^0}$$

$$\frac{\partial \mathcal{L}}{\partial b_i^0} = (a^2 - y)\sum_k w_k^1 \sigma'_k \delta_{ki} \qquad = (a^2 - y) w_i^1 \sigma'_i$$

- Averaging over m samples we get

$$db^0 = \frac{1}{m}\sum_s (a_s^2 - y_s) {w^1}^T \sigma'$$

summary and dimensions

- Let n_h be the number of hidden nodes and m be the number of samples

$$Z^1 = W^0 \cdot X + B^0 \qquad (n_h, 2) \times (2, m) + (n_h, 1) = (n_h, m)$$
$$A^1 = \sigma(Z^1)$$
$$Z^2 = W^1 \cdot A^1 + B^1 \qquad (1, n_h) \times (n_h, m) + (1, 1) = (1, n_h)$$

summary and dimensions

- Let n_h be the number of hidden nodes then

$$db^1 = \frac{1}{m}\sum_s (a_s^2 - y_s) \qquad (1,1)$$

$$dw^1 = \frac{1}{m}(A^2 - Y)\cdot {A^1}^T \qquad (1,m)\times(m,n_h) = (1,n_h)$$

$$db^0 = \frac{1}{m}\sum_s \left[{w^1}^T \cdot (A^2 - Y)\right] * \sigma' \qquad \sum_s (n_h,1)\times(1,m) = (n_h,1)$$

$$dw^0 = \frac{1}{m}\left[\left({w^1}^T \cdot (A^2 - Y)\right) * \sigma'\right]\cdot X^T \qquad (n_h,1)\times(1,m)\times(m,2) = (n_h,2)$$

Cost Functions

- Until now the cost function was somehow arbitrarily selected.
- In this section we see how the cost function emerges naturally from assumption about distribution of the data.
- First we need some elementary concepts from probability theory

Probability Concepts

- Let X and Y be random variables that can take the values $X \in \{x_1, \ldots, x_M\}$ and $Y \in \{y_1, \ldots, y_L\}$.
- Suppose that we perform N trials, and in each trial we obtain a value for X and Y.
- Let n_{ij} be the number of times we obtained $X = x_i$ and $Y = y_j$.
- We define the *joint* probability of $X = x_i$ and $Y = y_j$ as

$$P(X = x_i, Y = y_j) = \frac{n_{ij}}{N}$$

- We can ask what is the probability that $X = x_i$ *regardless* of what the value of Y is. This is called the *marginal* probability and its basically the fraction of times that we have obtained $X = x_i$

$$P(X = x_i) = \sum_{j=1}^{L} P(X = x_i, Y = y_j) = \frac{1}{N} \sum_{j=1}^{L} n_{ij} = \frac{c_i}{N}$$

- We can also ask what is the probability that $Y = y_j$ *given(knowing)* that $X = x_i$. This is called the *conditional* probability and we write

$$P(Y = y_j | X = x_i)$$

- To obtain an expression for the above we need to compute the fraction of trials that we obtained $Y = y_i$ out of the all the trials in which $X = x_i$

$$\begin{aligned} P(Y = y_i | X = x_i) &= \frac{n_{ij}}{c_i} \\ &= \frac{n_{ij}}{N} \frac{N}{c_i} \\ &= \frac{P(X = x_i, Y = y_i)}{P(X = x_i)} \end{aligned}$$

- Or

$$P(X = x_i, Y = y_i) = P(Y = y_i | X = x_i) P(X = x_i)$$

Likelihood

- Suppose that we observed a set of data D and we assume that the data comes from some probability distribution P that depends on some parameter w.
- This means that for every w we have a different probability for the occurrence of the data.
- We fix w and we ask what is the probability of the occurrence of the set of data D given that value of w?
- The above can be written as a *conditional* probability called the *likelihood*

$$P(D|w)$$

- Naturally, among all possible values of w we would like to select the one that makes $P(D|w)$ maximum. This called the *maximum likelihood*.

Bernoulli Distribution

- Bernoulli distribution is a distribution over a single binary (0 or 1) random variable (for example or ship- nonship study). It is given by

$$P(X = 1) = \pi$$
$$P(X = 0) = (1 - \pi)$$

- In general

$$P(X = x) = \pi^x (1 - \pi)^{1-x}$$

- In our ship-nonship study we used the notation y for the random variable and $\hat{y}$ for the probability so

$$P(y) = \hat{y}^y (1 - \hat{y})^{1-y}$$

- Since our prediction $\hat{y}$ is dependent on the weights and the bias we write $\hat{y} = \hat{y}(w, b)$.
- Denote by x the image and y the label and suppose that in the learning phase we obtained the values $x_1, \ldots, x_n$ along with the labels $y_1, \ldots, y_n$.
- Assuming that the variables are iid (independent and identically distributed) from the Bernoulli distribution then for every data point we have

$$P(y_i|x_i, w, b) = \hat{y}^{y_i}(x_i, w, b)(1 - \hat{y}(x_i, w, b))^{1-y_i}$$

- Since the data point are independent then the likelihood function becomes

$$P(y_1, \ldots, y_n|x_1, \ldots, x_n, w, b) = \prod_{i=1}^{n} P(y_i|x_i, w, b)$$

Maximizing the likelihood

- Clearly for every different value of w and b we obtain a different probability.
- Our aim is to maximize the likelihood function. Since the logarithm is a monotone function we can maximize the log of the likelihood which is more convenient. Also maximizing the log is equivalent to minimizing the negative of the log so we need to minimize

$$\log P(y_1, \ldots, y_n | x_1, \ldots, x_n, w,) = \sum_{i=1}^{n} y_i \log \hat{y}_i(w, b) + (1 - y_i) \log(1 - \hat{y}_i(w, b))$$

- Which is exactly what we did in the ship-nonship example.

Gaussian Distribution

- We can do the same analysis if the data is drawn from a Gaussian distribution

$$P(y_i|x_i, w, b) = \frac{1}{2\pi\sigma^2}^{\frac{1}{2}} \exp^{-\frac{1}{2\sigma^2}(y_i - \hat{y}(x_i, w, b))^2}$$

- Again assuming the data points are iid we get for the log of the likelyhood

$$\ln P(y_1, \ldots, y_n|x_1, \ldots, x_n, w, b) = -\frac{n}{2}\ln(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \hat{y}(x_i, w, b))^2$$

- Thus minimizing the negative of the log of the likelihood is minimizing the mean square error with respect to w and b

Computing the gradient

- We need to compute $\frac{\partial E}{\partial w_{ij}^l}$ and $\frac{\partial E}{\partial b_i^l}$ for all w_{ij}^l and b_i^l
- Since b_i^l appears explicitly only in z_i^{l+1} we can write

$$\frac{\partial E}{\partial b_i^l} = \frac{\partial E}{\partial z_i^{l+1}} \frac{\partial z_i^{l+1}}{\partial b_i^l}$$

- Recall that $z_i^l = \sum_p w_{ip}^{l-1} a_p^{l-1} + b_i^{l-1}$ thus $\frac{\partial z_i^{l+1}}{\partial b_i^l} = 1$ and it follows that

$$\frac{\partial E}{\partial b_i^l} = \frac{\partial E}{\partial z_i^{l+1}} = \delta_i^{l+1} \tag{3}$$

- We have defined δ_i^l which we will we will refer to often.

Computing derivative with respect to w_{ij}^l

- For the weights we have

$$\begin{aligned}\frac{\partial E}{\partial w_{ij}^l} &= \frac{\partial E}{\partial z_i^{l+1}} \frac{\partial z_i^{l+1}}{\partial w_{ij}^l} \\ &= \delta_i^{l+1} \frac{\partial z_i^{l+1}}{\partial w_{ij}^l}\end{aligned}$$

- Since $z_i^{l+1} = \sum_p w_{ip}^l a_p^l + b_i^l$ then

$$\frac{\partial z_i^{l+1}}{\partial w_{ij}^l} = a_j^l$$

- Therefore

$$\frac{\partial E}{\partial w_{ij}^l} = \delta_i^{l+1} a_j^l \tag{4}$$

Computing δ_i^l

- To compute δ_i^l we note that z_i^l is used as an input (through a_i^l) to all nodes in level $l+1$ so

$$\begin{aligned}\delta_i^l = \frac{\partial E}{\partial z_i^l} &= \sum_k \frac{\partial E}{\partial z^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_i^l} \\ &= \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_i^l} \\ &= \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial a_i^l} \frac{\partial a_i^l}{\partial z_i^l} \\ &= \sum_k \delta_k^{l+1} w_{ki}^l \sigma'(z_i^l)\end{aligned}$$

- Where σ' is the derivative of the sigmoid function and we have used the fact that

$$z_k^{l+1} = \sum_p w_{kp}^l a_p^l + b_k^l$$

Backpropagation

- We summarize the results so far

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

$$\frac{\partial E}{\partial w_{ij}^l} = \delta_i^{l+1} a_j^l$$

$$\delta_i^l = \frac{\partial E}{\partial z_i^l} = \sum_k \delta_k^{l+1} w_{ki}^l \sigma'(z_i^l)$$

- As we can see the computation of the derivatives of E depend on the computation of δ_i^l.
- The computation of δ_i^l for layer l can be computed if we know its value δ_k^{l+1} in the next layer. This is why this computation is called *backpropagation*

Computing δ_i^L for the last layer

- If we compute δ_i^l for the last layer $l = L$ we can compute all other values. To do so we need an explicit expression of E.
- For the Bernoulli distribution we have

$$E = -\sum_k \left(y_k \log a_k^L + (1 - y_k) \log(1 - a_k^L) \right)$$

- Then

$$\begin{aligned} \delta_i^L &= \frac{\partial E}{\partial z_i^L} = \frac{\partial E}{\partial a_i^L} \frac{\partial a_i^L}{\partial z_i^L} \\ &= \left[-\frac{y_i}{a_i^L} + \frac{1 - y_i}{1 - a_i^L} \right] \sigma'(z_i^L) \end{aligned}$$

- Since $\sigma'(z_i^L) = a_i^L(1 - a_i^L)$ we get

$$\delta_i^L = (a_i^L - y_i)$$

- For a Gaussian $E = -\frac{1}{2}\sum_k (y_k - a_k^L)^2$ then

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

Vectorization

- We will summarize the results in vector form
- Since δ_i^l and a_i^l are column vectors the expressions

$$\delta_i^l = \sum_k \delta_k^{l+1} w_{ki}^l \sigma'(z_i^l)$$
$$b_i^l = \delta_i^{l+1}$$
$$dw_{ij}^l = \delta_i^{l+1} a_j^l$$

  - is written as

$$\delta^l = \left[{w^l}^T \cdot \delta^{l+1} \right] * \sigma'^l$$
$$b^l = \delta^{l+1}$$
$$dw^l = \delta^{l+1} \cdot {a^l}^T$$

Averaging over samples

- During the computation we would like to average over all samples.
- As mentioned before arrays are stacked column-wise to include the samples.
- For example if δ^l has n rows and the number of samples is m we visualize δ^l as shown below

$$\begin{bmatrix} \delta^l_{11} & \dots & \delta^l_{1m} \\ \dots & \dots & \dots \\ \delta^l_{n1} & \dots & \delta^l_{nm} \end{bmatrix}$$

- To get the average values we just sum over all the columns and divide by m. This is done in Python by specifying the axis we are summing over.

- we get

$$b^l = \begin{bmatrix} \frac{1}{m}\sum_{s=1}^{m} \delta^l_{1s} \\ \dots \\ \frac{1}{m}\sum_{s=1}^{m} \delta^l_{ns} \end{bmatrix}$$

- In Python let $dz = \delta^l$,$dzp = \delta^{l+1}$,$wp = w^{l+1}$ and $dsig = \sigma'^l$ (where the suffix p refers to the previous value in backpropagation) we write

```
dz=1/m*np.sum(np.dot(wp.T,dzp)*dsig,axis=1)
```

- When the quantity in question is a matrix there is no need to sum the average is automatically taken care of

Overaging over samples for a matrix

- The weights are usually a matrix, a result of the tensor product of the deltas and the previous outputs
- So $dw^l = \delta^l \cdot {a^l}^T$ which can be visualized as

$$\begin{bmatrix} dw^l_{11} & \dots & dw^l_{1j} \\ \dots & \dots & \dots \\ dw^l_{i1} & \dots & dw^l_{ij} \end{bmatrix} = \begin{bmatrix} \delta^l_1 \\ \dots \\ \delta^l_i \end{bmatrix} \cdot \begin{bmatrix} a^l_1 & \dots & a^l_j \end{bmatrix}$$

- To average over m samples the only extra operation is to divide by m

$$\begin{bmatrix} dw^l_{11} & \dots & dw^l_{1j} \\ \dots & \dots & \dots \\ dw^l_{i1} & \dots & dw^l_{ij} \end{bmatrix} = 1/m \begin{bmatrix} \delta^l_1 & \dots & \delta^l_{1m} \\ \dots & & \\ \delta^l_i & \dots & \delta^l_{im} \end{bmatrix} \cdot \begin{bmatrix} a^l_1 & \dots & a^l_j \end{bmatrix}$$