M01S02- Feed Forward in Neural Networks

Introduction to Neural Networks

1. Introduction
2. Flow of Information in Neural Networks - Between 2 Layers
3. Information Flow - Image Recognition
4. Comprehension - Count of Pixels
5. Learning the Dimensions Weight Matrices
6. Feedforward Algorithm
7. Vectorized Feedforward Implementation
8. Understanding Vectorized Feedforward Implementation
9. Summary
10. Graded Questions

Introduction

Previous session covered the architecture of neural networks and their inspiration from the brain as also the working of an artificial neuron, the hyperparameters and parameters of neural networks and various simplifying assumptions.

This session covers how information flows in a neural network from the input layer to the output layer. The information flow in this direction is often called **feedforward**.

In this Session:

- Information flow between two layers.
- Information flow in the entire network.
- Inference in neural networks
- Basic image recognition with Neural network

Prerequisites

There are no prerequisites for this session other than knowledge of the matrix multiplication and previous courses on Statistics and ML.

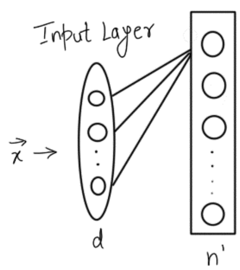
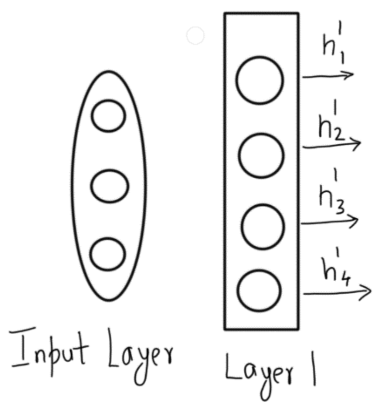
Flow of Information in Neural Networks - Between 2 Layers

Previous session covered the structure, topology, hyperparameters and the simplifying assumptions of neural networks. This segment covers how the information flows from one layer to the adjacent one in a neural network.

In artificial neural networks, the output from one layer is used as input to the next layer. Such networks are called **feedforward neural networks**. This means there are no loops in the network - information is always fed forward, never fed back. Let's start off with understanding the feedforward mechanism between two layers. For simplicity, the professor has taken the input and the first layer to demonstrate how information flows between any two layers.

How information flows from one layer to another

We have the two layers as follows:

 Input layer	The weight matrix between layer 0 (the input layer) and the first hidden layer is denoted by W^1. The dot product between W^1 and the input x_i along with the bias **b**, $W^1.x_i$ + b, acts as the input to layer-1. The activation function is applied to this input to compute the output of the first layer.
Let's take a concrete example - consider that the input is a vector of length 3 $\boldsymbol{x_i} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$. The dimension of $\boldsymbol{x_i}$ is (3,1).	There are four neurons in layer-1 with their outputs represented as follows:  4 neuron hidden layer

<table>
<tr>
<td>

$$\text{Hence, } h^1 = \begin{bmatrix} h_1^1 \\ h_2^1 \\ h_3^1 \\ h_4^1 \end{bmatrix}$$

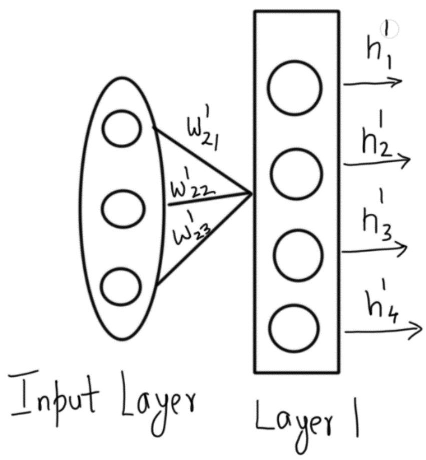
</td>
<td>

Weight matrix will be of dimensions 4 x 3

$$W^1 = \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 \\ w_{31}^1 & w_{32}^1 & w_{33}^1 \\ w_{41}^1 & w_{42}^1 & w_{43}^1 \end{bmatrix}$$

</td>
</tr>
<tr>
<td>

The individual elements of the weight matrix are shown below for the 2nd neuron:



Weight elements

</td>
<td>

On taking the dot product, we get:

$$W^1 . x_i = \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 \\ w_{31}^1 & w_{32}^1 & w_{33}^1 \\ w_{41}^1 & w_{42}^1 & w_{43}^1 \end{bmatrix} . \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} w_{11}^1 x_1 + w_{12}^1 x_2 + w_{13}^1 x_3 \\ w_{21}^1 x_1 + w_{22}^1 x_2 + w_{23}^1 x_3 \\ w_{31}^1 x_1 + w_{32}^1 x_2 + w_{33}^1 x_3 \\ w_{41}^1 x_1 + w_{42}^1 x_2 + w_{43}^1 x_3 \end{bmatrix}$$

We add the bias to the dot product. Bias vector is of the same dimension as $W^1.x_i$, i.e. 4 x 1.

$$b^1 = \begin{bmatrix} b_1^1 \\ b_2^1 \\ b_3^1 \\ b_4^1 \end{bmatrix}.$$

</td>
</tr>
</table>

Hence, on adding the bias, we get:

$$W^1 . x_i + b = \begin{bmatrix} w_{11}^1 x_1 + w_{12}^1 x_2 + w_{13}^1 x_3 \\ w_{21}^1 x_1 + w_{22}^1 x_2 + w_{23}^1 x_3 \\ w_{31}^1 x_1 + w_{32}^1 x_2 + w_{33}^1 x_3 \\ w_{41}^1 x_1 + w_{42}^1 x_2 + w_{43}^1 x_3 \end{bmatrix} + \begin{bmatrix} b_1^1 \\ b_2^1 \\ b_3^1 \\ b_4^1 \end{bmatrix} = \begin{bmatrix} w_{11}^1 x_1 + w_{12}^1 x_2 + w_{13}^1 x_3 + b_1^1 \\ w_{21}^1 x_1 + w_{22}^1 x_2 + w_{23}^1 x_3 + b_2^1 \\ w_{31}^1 x_1 + w_{32}^1 x_2 + w_{33}^1 x_3 + b_3^1 \\ w_{41}^1 x_1 + w_{42}^1 x_2 + w_{43}^1 x_3 + b_4^1 \end{bmatrix}$$

The last step is to apply the activation function. Activation function is applied to each element of the vector. Thus, the output of layer-1 is:

$$h^1 = \begin{bmatrix} h^1_1 \\ h^1_2 \\ h^1_3 \\ h^1_4 \end{bmatrix} = \sigma(W^1 . x_i + b) = \begin{bmatrix} \sigma(w^1_{11}x_1 + w^1_{12}x_2 + w^1_{13}x_3 + b^1_1) \\ \sigma(w^1_{21}x_1 + w^1_{22}x_2 + w^1_{23}x_3 + b^1_2) \\ \sigma(w^1_{31}x_1 + w^1_{32}x_2 + w^1_{33}x_3 + b^1_3) \\ \sigma(w^1_{41}x_1 + w^1_{42}x_2 + w^1_{43}x_3 + b^1_4) \end{bmatrix}$$

σ(x) is a vector function, i.e. it is applied element-wise to a vector.

This completes the forward propagation of a single data point through one layer of the network. To summarize, the procedure to compute the output of the i^{th} neuron in the layer l is:

- Multiply the i^{th} row of the weight matrix with the output of layer l-1 to get the weighted sum of inputs
- Convert the weighted sum to cumulative sum by adding the i^{th} bias term of the bias vector
- Apply the activation function, σ(x) to the cumulative input to get the output of the i^{th} neuron in the layer l

With this premise, let's study feedforward in a small neural network in the next segment.

Information Flow - Image Recognition

Let's now study the feedforward algorithm using a small example. Take the example of a 2-pixel x 2-pixel greyscale image. We will discuss a simple network whose task is to compute an amplified count of the number of grey (or 'on') pixels in the image.

Objective of the network is to calculate the amplified count (or number) of 'on' pixels in the 2 x 2 image. Outputs of hidden layers in large, real networks are not usually interpretable. this example only to get an intuitive understanding of the feedforward process.

The first hidden layer in the network counts the number of grey pixels in the image - the first and the second neurons count the number of grey pixels in row-1 and row-2 respectively. Since the input is a 2 x 2 image, and the first hidden layer has two neurons, the weight matrix associated with it is of dimensions 2 x 4:

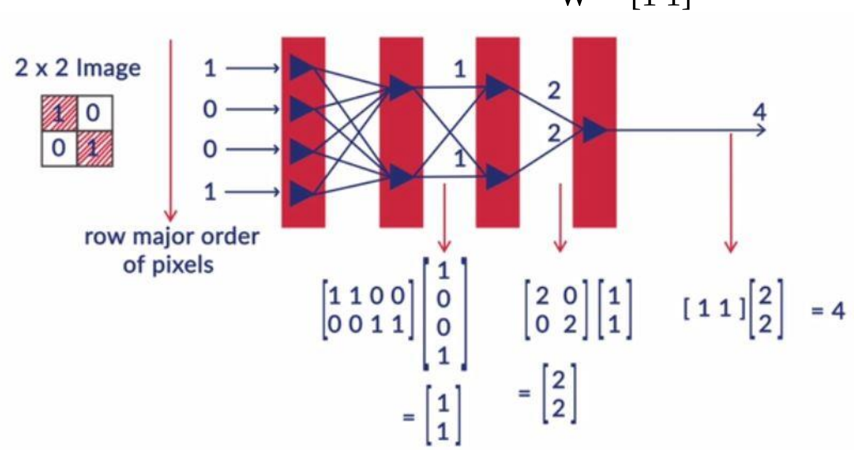
$$W^1 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

The second hidden layer simply amplifies the count by a factor of 2. The weight matrix corresponding to this operation is:

$$W^2 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

Finally, the output layer adds the two elements in the input and reports the amplified count of the grey pixels:

$W^3 = [1\ 1]$



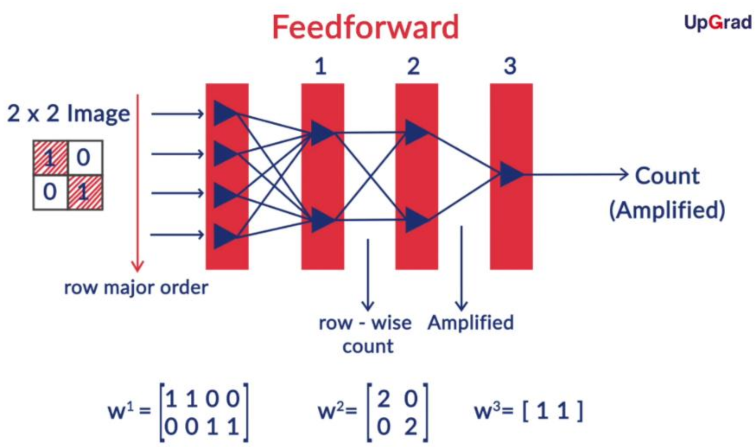
Counting Dark Pixels in a 2X2 image

We have assumed that all the biases are 0 and that we have used the trivial identity activation function which is a passthrough function.

Comprehension – Count of pixels

The figure below shows an artificial neural network which calculates the count of the number of pixels which are ‘on’, i.e. have a value of 1. It further amplifies the output by a factor of 2; so if 2 pixels are on, the output is 4, if 3 pixels are on the output is 6 and so on.

We’ll call the input layer as layer 0 or simply the input layer. The other three layers are numbered 1, 2 and 3 (3 is the output layer).



Pixel Count

The weight matrices of the 2 hidden layers and the third output layer are shown above. The first and the second neurons of the (hidden) layer 1 represent the number of ‘on’ pixels in row 1 and 2 respectively. The second (hidden) layer amplifies the output of layer 1 by a factor of 2 and the third (output) layer sums up the amplified counts.

The biases are all 0 and the activation function is $\sigma(x) = x$.

Let's make one minor modification in our network - let’s represent the input with 4 pixels as a vector with pixels **counted clockwise**. Thus, the **input** shown in the figure is $\begin{bmatrix}1\\0\\1\\0\end{bmatrix}$ (and NOT $\begin{bmatrix}1\\0\\0\\1\end{bmatrix}$ which is called the row-major order).

Inputs to the Network

If the output of the network is 6, then the possible inputs are (mark all that apply):

- [x] $\begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}$

  Feedback : The on pixels are represented by number 1. It amplifies the output by a factor of 2. So if 3 pixels are on, then output is 6.

- [x] $\begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$

  Feedback : The on pixels are represented by number 1. It amplifies the output by a factor of 2. So if 3 pixels are on, then output is 6.

- [ ] $\begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$

- [x] $\begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}$

  Feedback : The on pixels are represented by number 1. It amplifies the output by a factor of 2. So if 3 pixels are on, then output is 6.

Output of a Neuron

For the input $\begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, the output of the first neuron of hidden layer 1 is:

- ( ) 0
- ( ) 1
- (•) 2

  Feedback : *This neuron represents the number of on pixels in row 1, which is 2.*
- ( ) 3

Weights in Neural networks

Now, you want to modify the weights of layer-1 so that the first and the second neurons in the hidden layer 1 represent the number of 'on' pixels in the first and second **column** of the input image respectively. This should be true for all possible inputs into the network. What should be W^1? (note that the input vector is created by counting the pixels clockwise)

- ( ) $\begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}$
- (•) $\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$

  Feedback : *The first neuron's output will be the first row of w1 multiplied by the input vector. If the input vector is [1 0 0 1], i.e both pixels in the first column are on, then the output of neuron 1 should be 2. For [1 0 0 0], the output should be 1. In general, for the first neuron, the output should sum up the first and the fourth element of the input. Thus row 1 should be [1 0 0 1]. Similarly, for the second neuron, the output should sum up the second and the third element of the input. So the second row should be [0 1 1 0].*
- ( ) $\begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$
- ( ) $\begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$

Computational Complexity

Let's assume that each elementary algebraic operation, such as multiplication of two numbers, applying the activation function on a scalar f(x), etc. takes 0.10 microseconds (on a certain OS in python). Note that addition is NOT included as an operation. For the network discussed above, how much time would it take to compute the output from the input given all the parameters?

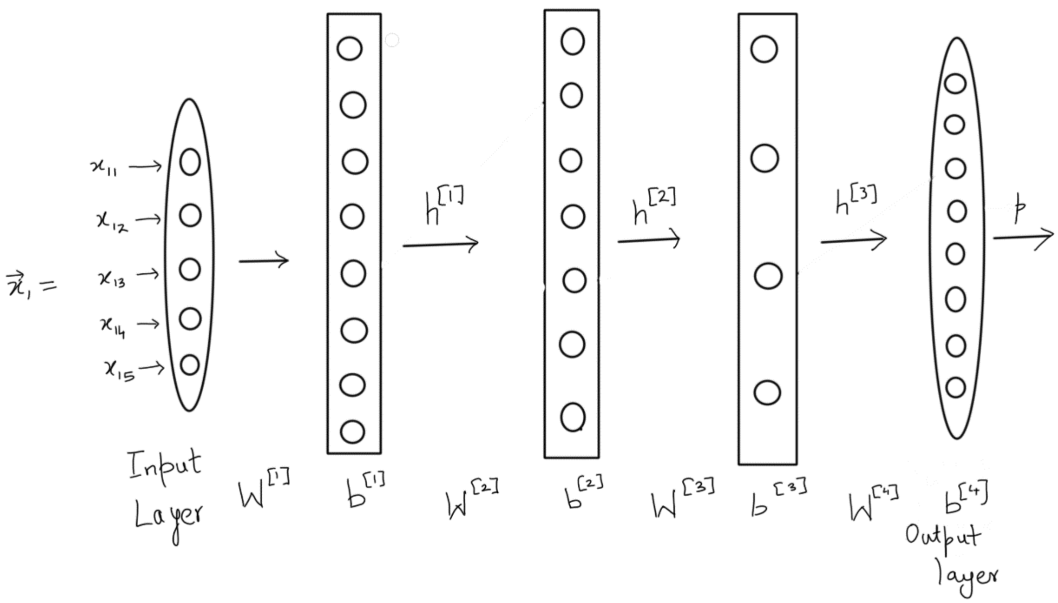
Hint: There are 3 weight matrices W^1, W^2 and W^3 of sizes 4x2, 2x2 and 2x1. The activation function is applied to each of the three layers' input to compute h^1, h^2 and h^3.

- (•) 1.9 microseconds ✓

  Feedback : *For the weight matrix W^1: It is a (2x4)(4x1) product, thus has 8 multiplications (on for each row). The activation function is then applied to the 2 neurons. Thus, 10 x 0.1 ms is the time taken till layer 1's output. For layer 2, W^2, it is a 2x2 product + activation function applied on 2 scalars. So far, we have 8 + 2 + 4 + 2 = 16 operations. For layer 3, its a (2x1)(1x2) matrix product + 1 activation operation. Thus, we have 16 + 3 = 19 operations which will take 1.9 microseconds.*
- ( ) 1.3 microseconds
- ( ) 19 microseconds
- ( ) 2.1 microseconds

Learning the Dimensions Weight Matrices

Practice computing the dimensions of the weight matrices and outputs of various layers of this network. Number of learnable parameters in a neural network are the weights and biases.



Dimension Check

Input vector

What is the dimension of the input vector x_1?

- (5,1)

  Feedback :
  There are 5 neurons in the input layer. Hence (5,1). By default, a vector is assumed to be a column vector.

- (1,5)

Weight Matrices

What are the dimensions of the weight matrices W^1, W^2, W^3 and W^4?

- (5,8), (8,7), (7,4) and (4,8)
- (8,5), (7,8), (4,7) and (8,4)

  Feedback :
  The dimensions of W^l are defined as (number of neurons in layer l, number of neurons in layer l-1)

Output vectors

What are the dimensions of the output vectors of the hidden layers h^1 and h^2?

- (8,1) and (7,1)

  Feedback :
  The dimension of the output vector for a layer l is (number of neurons in the layer, 1)

- (1,8) and (1,7)

Bias vectors

The dimension of the bias vector is the same as the output vector for a layer l for a single input vector. True or False.

- True

  Feedback :
  For a single data input, both have the same dimension.

- False

Number of Learnable Parameters in the Network

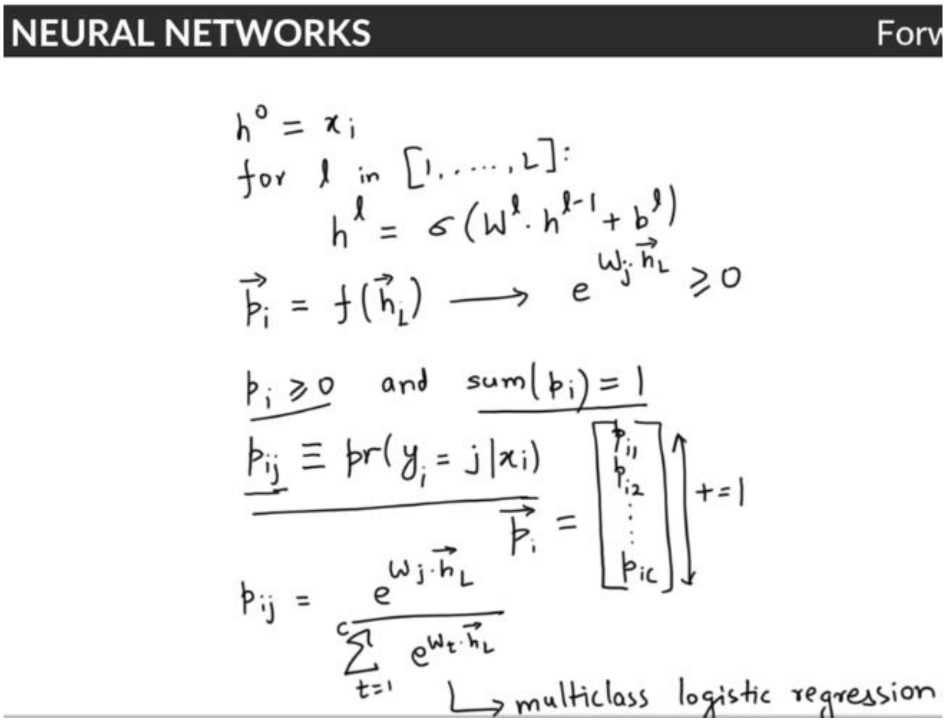
What is the number of learnable parameters in this network? Note that the learnable parameters are weights and biases.

- 156
- **183**

  Feedback :
  The weights have 40, 56, 28, 32 individual parameters(the weight elements). The layers have 8, 7, 4, 8 biases respectively.
- 176

Feedforward Algorithm

Let's write the **pseudocode for a feedforward pass** through the network for a single data point x_i. This will help implement your own neural network in NumPy.



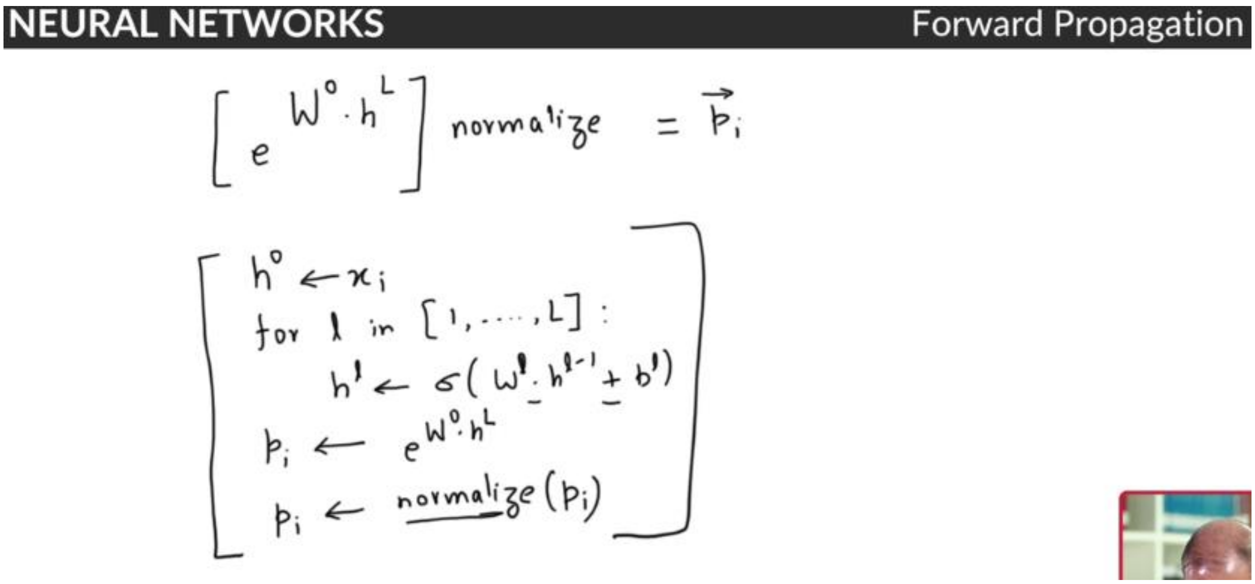
Pseudocode of the feedforward algorithm is as follows. h_0 has been initialised with x_i, the input to the network:

1. $h^0 = x_i$
2. for l in $[1, 2, \ldots\ldots., L]$:
    1. $h^l = \sigma(W^l . h^{l-1} + b^l)$
3. $p = f(h^L)$

The last layer (the output layer) is different from the rest, and it is important how we define the output layer. Here, since we have a multiclass classification problem (the MNIST digits between 0-9), we have used the **softmax output.**

$$p_i = \begin{bmatrix} p_{i1} \\ p_{i1} \\ . \\ . \\ p_{ic} \end{bmatrix} \text{ where } p_{ij} = \frac{e^{w_j.h^L}}{\sum_{t=1}^{c} e^{w_t.h^L}} \text{ for } j = [1, 2, \dots, c] \text{ \& } c = \text{number of classes}$$

To have a better understanding of the softmax function and understand what the $w_j.h^L$ computes, let's just focus on the output layer.



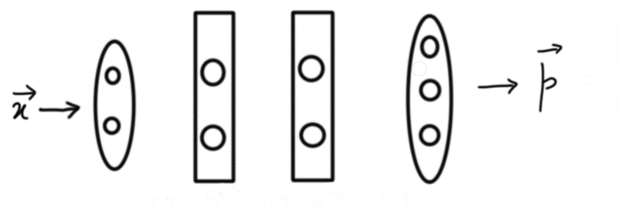
Output of the last layer is computed. Calculating $p_{ij} = \frac{e^{w_j.h^L}}{\sum_{t=1}^{c} e^{w_t.h^L}}$ is often called **normalizing** the vector $\mathbf{p}_i$.

Hence, the complete feedforward algorithm becomes:

1. $h^0 = x_i$
2. for l in [1,2,.......,L]:
   1. $h^l = \sigma(W^l.h^{l-1} + b^l)$
3. $p_i = e^{W^o.h^L}$
4. p_i = normalize(p_i)

W^0 (the weights of the output layer) can also be written as W^{L+1}.

Let's now try to understand how decision making happens in the softmax layer using the same example network we had used in the previous segment:



Softmax

We have the last weight matrix W^3 as W^0. The output layer classifies the input into one of the three labels: 1, 2 or 3. The first neuron outputs the probability for label 1, the second neuron outputs the probability for label 2 and hence the third neuron outputs the probability for label 3.

Dimension W o

What is the dimension of W^o?

- (2,3)
- (3,2)

  Feedback :
  Dimension = (number of neurons in layer l, number of neurons in layer l-1) for a weight matrix W^l

Weight matrix calculation

Consider $W^o = \begin{bmatrix} 3 & 4 \\ 1 & 9 \\ 6 & 2 \end{bmatrix}$ and $h^2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ and bias is 0. What will be $W^o.h^2$?

- $\begin{bmatrix} 11 \\ 19 \\ 10 \end{bmatrix}$

  Feedback :
  It is a simple matrix multiplication.
- $\begin{bmatrix} 19 \\ 11 \\ 10 \end{bmatrix}$

Softmax Claculation

Consider $W^o = \begin{bmatrix} 3 & 4 \\ 1 & 9 \\ 6 & 2 \end{bmatrix}$ and $h^2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ with the bias = 0. What will be the softmax output vector p, i.e. the output of the 3^{rd} layer? In other words, what is normalized(p) (up to 5 decimal places)?

- $\begin{bmatrix} 0.00034 \\ 0.99954 \\ 0.00012 \end{bmatrix}$

  Feedback :
  The probability order should be 3<1<2 for the class labels.
- $\begin{bmatrix} 0.00012 \\ 0.99954 \\ 0.00034 \end{bmatrix}$

Predicted label

What is the predicted label?

- 1
- 2

  Feedback :
  As the highest probability is for the neuron representing label 2, it is the predicted label.
- 3

Until now, we have been doing feed forward for one single data point at a time (i.e. a single image, in case of the MNIST dataset). But the training data may have millions of data points. For e.g., the MNISt dataset itself has about 60,000 images in the training set.

Vectorized Feedforward Implementation

In the previous segment, we wrote pseudocode for doing feedforward for a single data point x_i at a time. Of course, training data has multiple data points, and we need to perform feedforward computation for all of them.

A bad way to do that would be to write a 'for loop' iterating through all the data points. There must be a more efficient way of doing it.

Let's now study how to do feed forward for an **entire batch of data points** in one go using **vectorized computation techniques.**

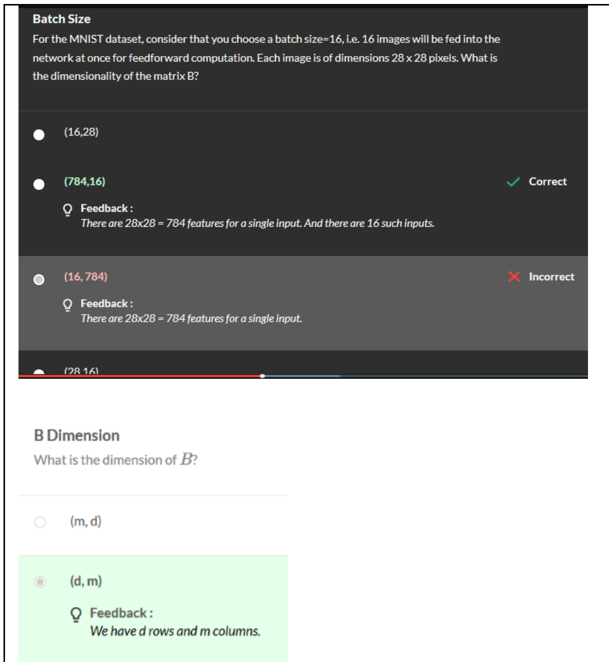
Vectorized implementation means to perform the computation (here, feedforward) for multiple data points using matrices. This will be much quicker than looping through one data point at a time.

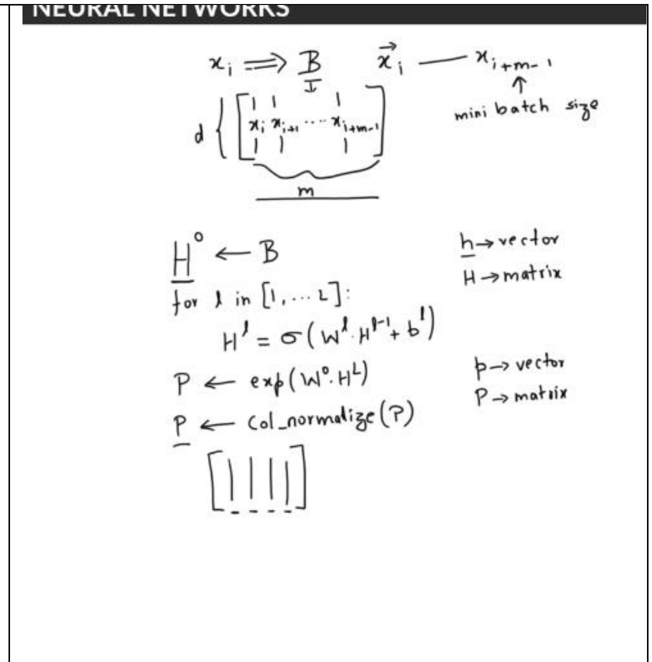
Before we move to the vectorized implementation, let's write the feedforward pseudocode for a set of m data points using a 'for loop':

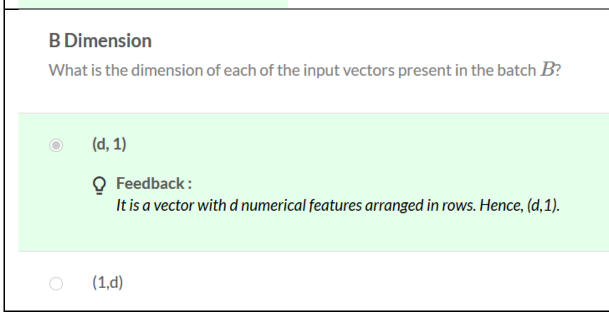
1. for i in $[1, 2, \ldots, m]$:
    1. $h^0 = x_i$
    2. for l in $[1, 2, \ldots, L]$:
        1. $h_i^l = \sigma(W^l . h_i^{l-1} + b^l)$
    3. $p_i = f(h^L)$

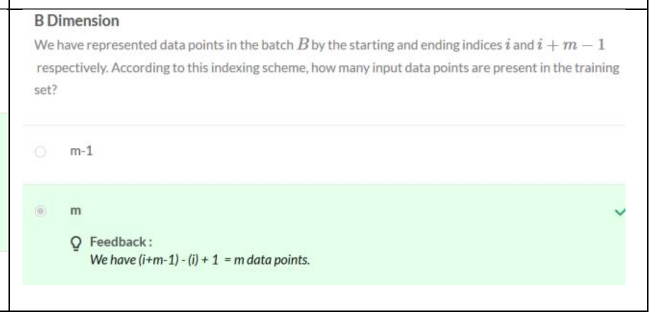
We require two nested 'for loops'. This will become computationally quite expensive if we have a large dataset (which is often the case with neural networks).

Now let's understand how doing the same using **matrices** can be much efficient.









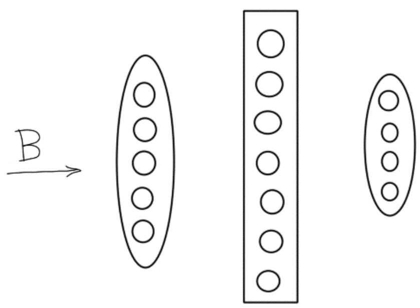
Thus, the feedforward algorithm for a batch B is as follows:

1. $H^0 = B$
2. for l in $[1, 2, \ldots\ldots, L]$:
    1. $H^l = \sigma(W^l . H^{l-1} + b^l)$
3. P = normalize(exp($W^o . H^L + b^o$))

This is very similar to the algorithm for a single data point with some notational changes. Specifically, we have used the uppercase notations H^l and P instead of h^l and p respectively to denote an entire batch of data points. In other words, H^l and P are matrices whose i^{th} column represents the h^l and p vectors respectively of the i^{th} data point. The number of columns in these 'batch matrices' is equal to the number of data points in the batch m.

$$H^l = \begin{bmatrix} | & | & | & | & | \\ h_i^l & h_{i+1}^l & . & . & h_{i+m-1}^l \\ | & | & | & | & | \end{bmatrix} \text{ and } P = \begin{bmatrix} | & | & | & | & | \\ p_i & p_{i+1} & . & . & p_{i+m-1} \\ | & | & | & | & | \end{bmatrix}$$

You are given a simple network as shown below:



Question Image

You pass a batch B consisting of 50 data points through this network. Each data point is represented by five features.

P Dimension

What is the dimension of the network output matrix P?

- (5,50)
- (4,50)

  Feedback :
  There are 4 neurons in the output layer. Hence, for every input data point, there will be an output vector of shape (4,1). Since there are 50 such data points, the shape of the matrix P is (4,50).
- (7,50)
- (16,50)

H 1 Dimension

What is the dimension of the output matrix out of the first hidden layer, that is H^1?

- (7,50)

  Feedback :
  There are 7 neurons in the hidden layer 1. Hence, for every input data point, there will be an output vector of shape (7,1). Since there are 50 such data points, the shape of the matrix H^1 is (7,50).
- (5,50)
- (12, 50)
- (4,50)

B Dimension

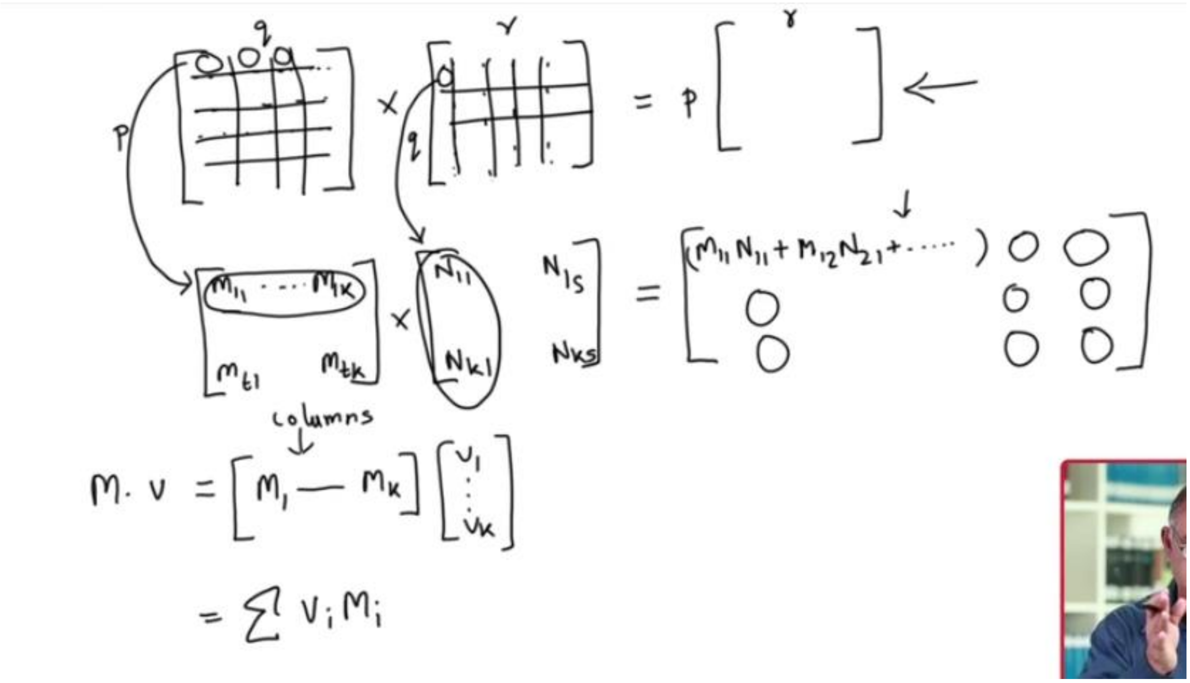
What is the dimension of the input batch B?

- (7,50)
- (5,50)

  Feedback :
  There are 50 input data points and there are 5 neurons for each input data point. Hence, dimension = (5,50)
- (4,50)
- (16,50)

Understanding Vectorized Feedforward Implementation

Previous segment covered how multiple data points can be fed forward as a batch. In this segment, we will try to make sense of the matrix multiplications mentioned in the feedforward algorithm. Let's go through some nice properties and tricks of matrix multiplication in this lecture:



$$M.\underline{H} = M.[\underline{H_1 \cdots\cdots H_k}] = [\underline{M.H_1} \cdots\cdots \underline{M.H_k}]$$

Let's understand the block matrix multiplication using some examples. It would be convenient to use NumPy to do the following matrix calculations. To compute matrix multiplication of A and B, write *numpy.dot(A, B)* in python. Hence, this product is often referred to as the dot product of matrices.

Consider the multiplication of the following two matrices:

$$A = \begin{array}{c} \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{bmatrix} 5 & 6 & 7 \\ 9 & 12 & 13 \\ 10 & 11 & 12 \end{bmatrix} \end{array} \text{ and } B = \begin{bmatrix} 1 & 2 & 3 \\ 12 & 9 & 5 \\ 10 & 11 & 12 \end{bmatrix}$$

Now consider block matrices of A in such a way that $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ where,

$$a_{11} = \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix}, a_{12} = \begin{bmatrix} 3 \\ 7 \end{bmatrix}, a_{21} = \begin{bmatrix} 9 & 12 \\ 10 & 11 \end{bmatrix}, a_{22} = \begin{bmatrix} 13 \\ 12 \end{bmatrix}$$

are block matrices. Now, since the block matrices of A are to be multiplied with the block matrices of B, the number of columns in the former should match the number of rows in the latter.

We represent $B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$ where $b_{11}, b_{12}, b_{21}, b_{22}$ are the component block matrices of B.

Dimensions

What should be the number of rows in the block matrices b_{11}, b_{21} of B where $B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$?

- 2, 1

  Feedback :
  The number of columns in the block matrices of A should match the number of rows in the block matrix representation of B

- 1,2

Now let the individual block matrices of $B = \begin{bmatrix} 1 & 2 & 3 \\ 12 & 9 & 5 \\ 10 & 11 & 12 \end{bmatrix}$ be as follows:

$$b_{11} = \begin{bmatrix} 1 & 2 \\ 12 & 9 \end{bmatrix}, b_{12} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}, b_{21} = \begin{bmatrix} 10 & 11 \end{bmatrix}, b_{22} = \begin{bmatrix} 12 \end{bmatrix}$$

We already know the component matrices of A:

$$a_{11} = \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix}, a_{12} = \begin{bmatrix} 3 \\ 7 \end{bmatrix}, a_{21} = \begin{bmatrix} 9 & 12 \\ 10 & 11 \end{bmatrix}, a_{22} = \begin{bmatrix} 13 \\ 12 \end{bmatrix}$$

Hence, $A.B = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} . \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$

Block Matrix Calculation

What is $a_{11}b_{11} + a_{12}b_{21}$?

$$\begin{bmatrix} 5 & 53 \\ 147 & 11 \end{bmatrix}$$

$$\begin{bmatrix} 55 & 53 \\ 147 & 141 \end{bmatrix}$$

Feedback :
Simple matrix multiplication. Try it using numpy.

Block Matrix Calculation

What is $a_{11}b_{12} + a_{12}b_{22}$?

$$\begin{bmatrix} 49 \\ 129 \end{bmatrix}$$

Feedback :
Simple matrix multiplication. Try it using numpy.

$$\begin{bmatrix} 4 \\ 12 \end{bmatrix}$$

Dot Product

What is the dot product of A and B?

$$\begin{bmatrix} 55 & 53 & 49 \\ 147 & 141 & 129 \\ 283 & 269 & 243 \\ 262 & 251 & 229 \end{bmatrix}$$

Feedback :
Simple matrix multiplication. Try it using numpy.

$$\begin{bmatrix} 55 & 53 & 49 \\ 147 & 11 & 129 \\ 83 & 26 & 23 \\ 22 & 251 & 229 \end{bmatrix}$$

Now, let's use the block matrix multiplication techniques described above in vectorized feedforward implementation.

$$M.\underline{H} = M.[\underline{H_1 \cdots H_K}] = [\underline{M.H_1} \cdots \underline{M.H_K}]$$

$$H^l = \sigma(\underline{W^l.H^{l-1}} + b^l)$$

matrix

$$W^l.[H_1^{l-1} \cdots H_{|B|}^{l-1}] + b^l \leftarrow \text{vector}$$

$$= [\underline{W^l.H_1^{l-1} \cdots W^l.H_{|B|}^{l-1}}] \oplus b^l$$

broadcast

$$= [W^l.H_1^{l-1}+b^l \cdots W^l.H_{|B|}^{l-1}+b^l] \quad [\underbrace{b^l\; b^l \cdots b^l}_{|B| \text{ copies}}]$$

We already know that the output of layer l (for a batch of m points) is:

$$H^l = \sigma(W^l.H^{l-1} + b^l)$$

We know that in $W^l.H^{l-1}$ both W^l and H^{l-1} are matrices. Using the above-described block matrix multiplication, we break down H^{l-1} into blocks of column vectors with each column vector representing one data point in the batch:

$$W^l.H^{l-1} = W^l.[H_i^{l-1} \quad H_{i+1}^{l-1} \quad . \quad . \quad H_{i+m-1}^{l-1}] = W^l.[h_i^{l-1} \quad h_{i+1}^{l-1} \quad . \quad . \quad h_{i+m-1}^{l-1}]$$

Thus,

$$W^l.H^{l-1} = [W^l.h_i^{l-1} \quad W^l.h_{i+1}^{l-1} \quad . \quad . \quad W^l.h_{i+m-1}^{l-1}]$$

Note that, $W^l.H^{l-1}$ is a matrix, and here, adding a vector b^l to the matrix $W^l.H^{l-1}$ refers to adding the vector each column of the matrix. In vectorized code, such as in Numpy, this addition is done through a process known as **broadcasting**.

Broadcasting of a vector b^l basically means creating a matrix with as many columns as in $W^l.H^{l-1}$ – each column being a copy of the vector b^l. Hence,

$$W^l.H^{l-1} + b^l = \left[W^l.h_i^{l-1} \quad W^l.h_{i+1}^{l-1} \quad . \quad . \quad W^l.h_{i+m-1}^{l-1} \right] + [b^l \quad b^l \quad . \quad . \quad b^l]$$

$$W^l.H^{l-1} + b^l = \left[W^l.h_i^{l-1} + b^l \quad W^l.h_{i+1}^{l-1} + b^l \quad . \quad . \quad W^l.h_{i+m-1}^{l-1} + b^l \right]$$

It is now easy to see how **parallelised computation** is possible (which is what modern GPUs specialise in doing).

Computation in Neural Networks

Neural network computations can be parallelized because:

- ○ Computation for each layer can be done independently of the others
- ◉ **Products of matrices and vectors can be easily parallelized**

  Feedback :
  Neural network computations essentially boil down to matrix-vector products.
- ○ Neurons in one layer are not connected to each other

Summary

How the information flows from the input layer to the output layer in Artificial Neural Networks (feedforward) using a simple image recognition problem as an example.

How to specify the dimensions and representations of the weight matrices, the biases, inputs and outputs of the layers, etc. of the various layers.

How **feedforward** can be done in a **vectorized form** and how it can be become efficient by using **parallelization.** The feedforward algorithm is summarised for a batch input:

1. $H^0 = B$
2. for l in $[1, 2, \ldots\ldots, L]$:
    1. $H^l = \sigma(W^l . H^{l-1} + b^l)$
3. P = normalize(exp($W^o . H^L + b^o$))