Feed Forward Neural Nets

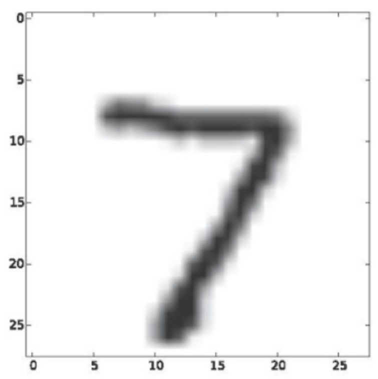
CHAPTER – 1 (Introduction to Deep Learning)

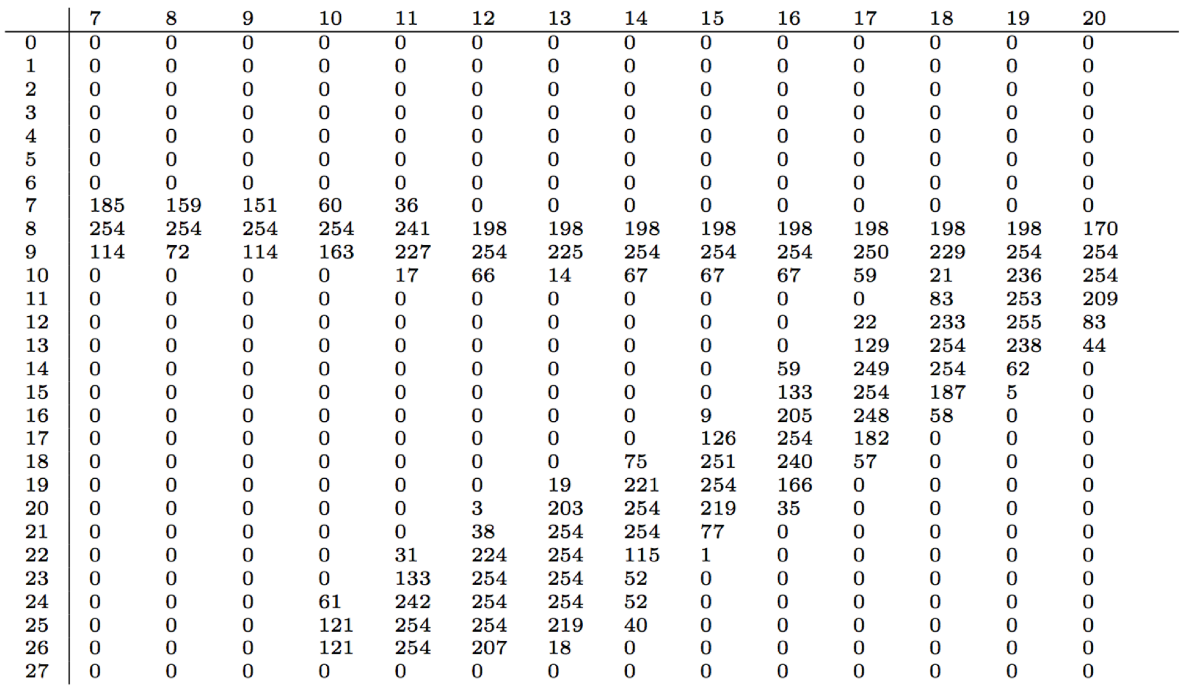
Topics

- Perceptrons
- Cross-entropy Loss functions for Neural Net
- Derivatives and Stochastic Gradient Decent
- Writing our program
- Matrix representation of Neural Net

Identifying Digits

- Mnist Dataset (National Institute of Standards and Technology)
    - Hand written images scanned and presented as 28x28 integer data.



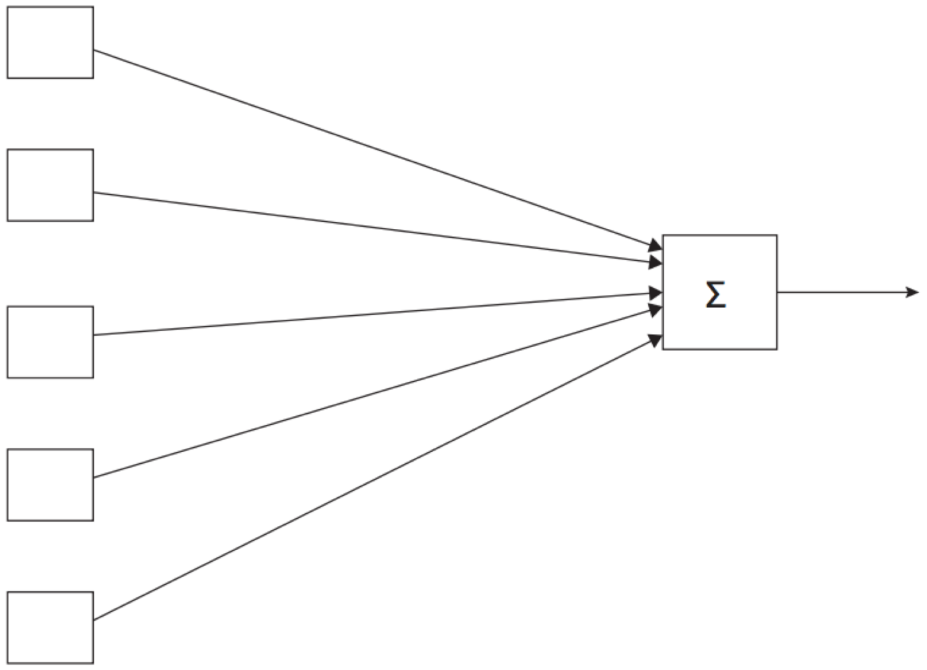


	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	185	159	151	60	36	0	0	0	0	0	0	0	0	0
8	254	254	254	254	241	198	198	198	198	198	198	198	198	170
9	114	72	114	163	227	254	225	254	254	254	250	229	254	254
10	0	0	0	0	17	66	14	67	67	67	59	21	236	254
11	0	0	0	0	0	0	0	0	0	0	0	83	253	209
12	0	0	0	0	0	0	0	0	0	0	22	233	255	83
13	0	0	0	0	0	0	0	0	0	0	129	254	238	44
14	0	0	0	0	0	0	0	0	0	59	249	254	62	0
15	0	0	0	0	0	0	0	0	0	133	254	187	5	0
16	0	0	0	0	0	0	0	0	9	205	248	58	0	0
17	0	0	0	0	0	0	0	0	126	254	182	0	0	0
18	0	0	0	0	0	0	0	75	251	240	57	0	0	0
19	0	0	0	0	0	0	19	221	254	166	0	0	0	0
20	0	0	0	0	0	3	203	254	219	35	0	0	0	0
21	0	0	0	0	0	38	254	254	77	0	0	0	0	0
22	0	0	0	0	31	224	254	115	1	0	0	0	0	0
23	0	0	0	0	133	254	254	52	0	0	0	0	0	0
24	0	0	0	61	242	254	254	52	0	0	0	0	0	0
25	0	0	0	121	254	254	219	40	0	0	0	0	0	0
26	0	0	0	121	254	207	18	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 1.1: An Mnist discretized version of an image

Perceptrons

- Schematic diagram of perceptrons.



Perceptrons

- Binary classification machine learning scheme.
    - In our case of identifying numbers from image, if a given image is of number ‘0’ or not.
    - Simple computational models of neurons.

- A single neuron has
    - Many inputs (dendrites)
    - A cell body
    - A single output (the axon)

Perceptrons

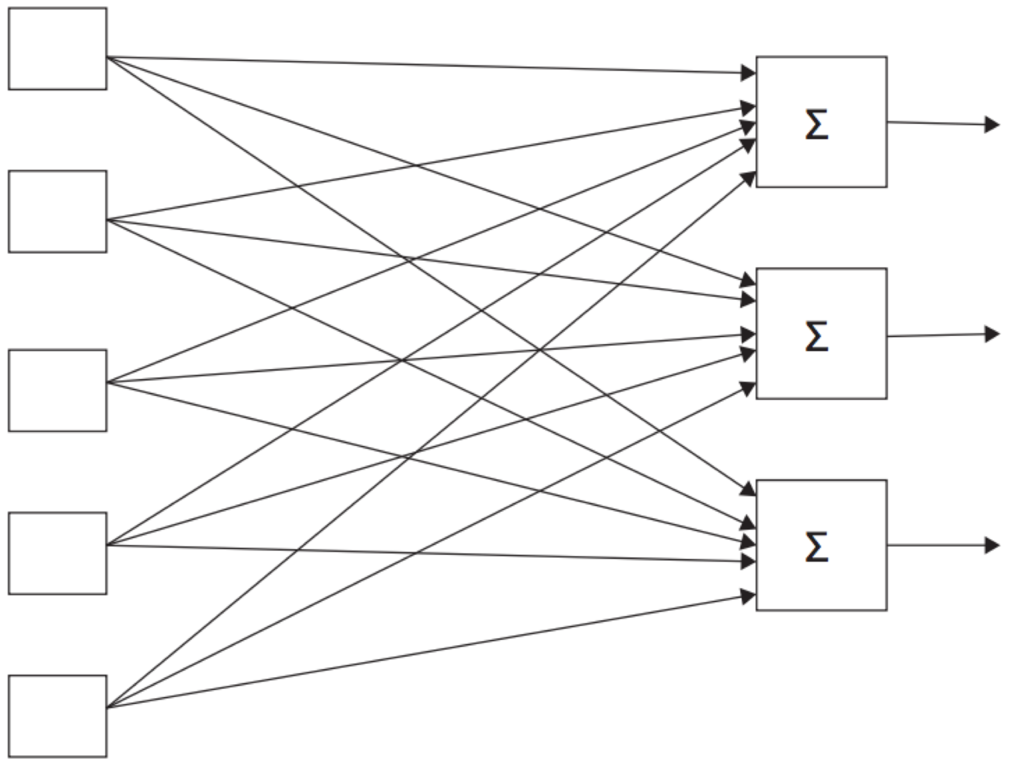
- A perceptron consists of
  - A vector of weights **w** = [$w_1 \ldots w_m$], one for each input
  - A distinguished weight **b**, which is bias
- **w** and **b** are the parameters of the perceptrons. Φ is used to denote the parameters which can be defined as Φ = {w ∪ b}
- Below is the perceptron function,

$$f_\Phi(\mathbf{x}) = \begin{cases} 1 & \text{if } b + \sum_{i=1}^{l} x_i w_i > 0 \\ 0 & \text{otherwise} \end{cases}$$

Perceptrons Algorithm

```
1. set $b$ and all of the $\mathbf{w}$'s to 0.
2. for $N$ iterations, or until he weights do not change
    (a) for each training example $\mathbf{x^k}$ with answer $a^k$
        i. if $a^k - f(\mathbf{x}^k) = 0$ continue
        ii. else for all weights $w_i$, $\Delta w_i = (a^k - f(\mathbf{x}^k))x_i$
```

Multiple Perceptrons for multiple classes



Cross-entropy Loss functions for Neural Net

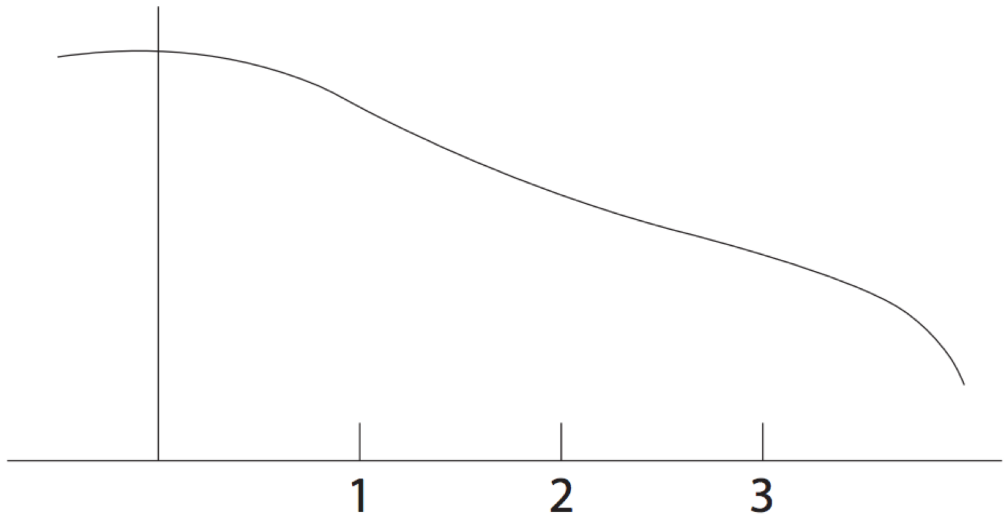
- A loss function (*L*) is a function indicating from an outcome to how bad the outcome is.
  - Loss function in perceptrons would have value 0 if the result is correct otherwise it is 1.

$$\Delta\phi_i = -\mathcal{L}\frac{\partial L}{\partial \phi_i}$$

- Here $\mathcal{L}$ denotes learning rate. The partial derivative indicates the change in loss function respect to the change in the parameters.

Loss as a function of ϕ_1

- Goal is to minimize the loss function value.
- Avoid large weights.

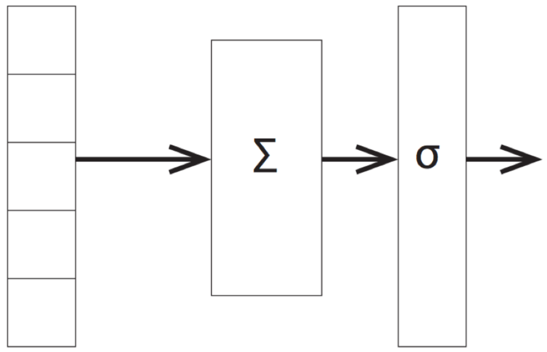


Softmax

- The layers mentioned above produce both positive and negative numbers. To convert this output to a probability distribution, softmax can be used.

$$\sigma(\mathbf{x})_j = \frac{e^{x_j}}{\sum_i e^{x_i}}$$

- Extension of simple network using softmax.



Cross-entropy Loss Function

- In the information theory, there is a property of probability distributions called cross-entropy.

$$X(\Phi, x) = -\ln p_{\Phi}(a_x)$$

- Equation above computes an estimate of cross-entropy.
- For a loss function, it should increase if model gets worse and should improve if model is progressing correctly.
  - The negative sign in front makes the number smaller if probability gets higher.
  - Log of a number increases or decreases the number.
  - This proves that X indeed behaves as a loss function.

Cross-entropy loss function, perceptron and softmax probability distribution.

- Cross-entropy loss function: $X(\Phi, x) = -\ln p(a)$
- Softmax probability distribution: $p(a) = \sigma_a(\mathbf{l}) = \frac{e^{l_a}}{\sum_i e^{l_i}}$
- Perceptron: $l_j = b_j + \mathbf{x} \cdot \mathbf{w_j}$

Stochastic Gradient Decent

- Process for going from input to the loss is called forward pass of the learning algorithm.
    - Forward pass computes the values to be used in backward pass also known as weight adjustment pass.
- It follows the gradient of the loss function and the system is working towards minimizing the loss.
    - Hence the name “Gradient Decent”

Derivatives of Gradient Descent

- From previous slide, the change in bias value directly reflects change in the loss function through logit l.

$$\frac{\partial X(\Phi)}{\partial b_j} = \frac{\partial l_i}{\partial b_j} \frac{\partial X(\Phi)}{\partial l_j}$$

- For first partial derivative, replacing l_j with it's equation we get,

$$\frac{\partial l_i}{\partial b_j} = \frac{\partial}{\partial b_j}(b_j + \sum_i x_i w_{j,i}) = 1$$

- For the second term, chain rule can again be applied,

$$\frac{\partial X(\Phi)}{\partial l_j} = \frac{\partial p_a}{\partial l_j} \frac{\partial X(\phi)}{\partial p_c}$$

Derivatives of Gradient Descent – cont'd

- For the second term in the last equation,

$$\frac{\partial X(\phi)}{\partial p_a} = \frac{\partial}{\partial p_a}(-\ln p_a) = -\frac{1}{p_a}$$

  - This above equation says that because X is dependent on the probability of the correct answer, logit only affects X by changing probability.

- That leaves the first term,

$$\frac{\partial p_a}{\partial l_j} = \frac{\partial \sigma_a(\mathbf{l})}{\partial l_j} = \begin{cases} (1-p_j)p_a & a = j \\ -p_j p_a & a \neq j \end{cases}$$

$$\boxed{\begin{aligned} \frac{\partial X(\Phi)}{\partial l_j} &= -\frac{1}{p_a}\begin{cases} (1-p_j)p_a & a = j \\ -p_j p_a & a \neq j \end{cases} \\ &= \begin{cases} -(1-p_j) & a = j \\ p_j & a \neq j \end{cases} \end{aligned}}$$

Derivatives of Gradient Descent – cont’d

- To reiterate, if all these values are put in,

$$\frac{\partial X(\Phi)}{\partial b_j} = \frac{\partial l_i}{\partial b_j}\frac{\partial X(\Phi)}{\partial l_j}$$

$$\boxed{\Delta b_j = \mathcal{L}\begin{cases}(1-p_j) & a=j\\ -p_j & a\neq j\end{cases}}$$

  - Change in bias can be calculated with respect to the learning curve and the probability of the given input. Here both options are listed where input is same as answer and it is not same as answer.

Derivatives of Gradient Descent – cont'd

- Change in loss can also be determined with respect to the change in weight which gives a way to determine the change in weight for every iteration.

$$\frac{\partial X(\Phi)}{\partial w_{j,i}} = \frac{\partial}{\partial w_{j,i}}(b_j + (w_{j,1}x_1 + \ldots + w_{j,i}x_i + \ldots)) = x_i$$

  - Which similarly results in,

$$\Delta w_{j,i} = -\mathcal{L} x_i \frac{\partial X(\Phi)}{\partial l_j}$$

Pseudo code for simple feed-forward digital recognition

```
1. for j from 0 to 9 set $b_j$ randomly (but close to zero)

2. for j from 0 to 9 and for i from 0 to 783 set $w_{j,i}$ similarly

3. until development accuracy stops increasing

   (a) for each training example $k$ in batches of $m$ examples
         i. do the forward pass using Equations 1.7 1.8, and 1.9
        ii. do the backward pass using Equations 1.20, 1.17, and 1.12
       iii. every $m$ examples, modify all $\Phi$'s with the summed updates
   (b) compute the accuracy of the model by running the forward pass
       on all examples in the development corpus

4. output the $\Phi$ from the iteration *before* the decrease in development
   accuracy.
```

Matrix representation of NN

- Basic matrix operations example:
  - X = Y + Z ($x_{i,j} = y_{i,j} + z_{i,j}$)
  - X = YZ (dot product between rows of Y and columns of Z)
  - Matrix transposition which exchanges matrix's rows and columns.

$$x_{i,j} = \sum_{k=1}^{k=m} y_{i,k} z_{k,j}$$

- Example of matrix operation:
  - L = X * W + B

$$\begin{aligned} \begin{pmatrix} 1 & 2 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} + \begin{pmatrix} 7 & 8 & 9 \end{pmatrix} &= \begin{pmatrix} 9 & 12 & 15 \end{pmatrix} + \begin{pmatrix} 7 & 8 & 9 \end{pmatrix} \\ &= \begin{pmatrix} 16 & 20 & 24 \end{pmatrix} \end{aligned}$$

Matrix form of weight update

- Backward pass can be defined as,

$$\nabla_1 X(\Phi) = \left(\frac{\partial X(\Phi)}{\partial l_1} \cdots \frac{\partial X(\Phi)}{\partial l_m} \right)$$

- Below upside down triangle is the symbol for the gradient. the input layer feeds into the layer of linear units to produce the logits, and then following the loss derivatives back to the changes in the parameters.

$$\mathbf{\Delta W} = -\mathcal{L}\mathbf{X}^T \nabla_1 X(\Phi)$$