Chapter - 1: Vectors, Matrices and Tensors for Deep Learning

Vectors from Data Matrix

Consider a patient's data that has 4 patients and 3 features - HeartRate (HR), Blood Pressure (BP) and Temperature (Temp). The data can be visualized as:

	HR	BP	Temp
P-1	76	126	38.0
P-2	74	120	38.0
P-3	72	118	37.5
P-4	78	136	37.0

A vector is a column wise listing of numbers. The first vector HeratRate (HR) for all patients is denoted by X_1 and it is given as:

$$X_1 = \begin{bmatrix} 76 \\ 74 \\ 72 \\ 78 \end{bmatrix}$$

In similar manner, we can define feature vectors X_2 and X_3 for Blood Pressure (BP) and Temperature (Temp) respectively.

The first patient vector across all features is denoted by $X^{(1)}$ and it is given as:

$$X^{(1)} = \begin{bmatrix} 76 \\ 126 \\ 38 \end{bmatrix}$$

In similar manner, we can define the vectors $X^{(2)}$, $X^{(3)}$ and $X^{(4)}$ for the second, third and fourth patient respectively. In short, the subscript is used for denoting features and the superscript is used for denoting sample vectors. One thing to keep in mind is Vectors are always written in column-wise fashions.

Attributes of vectors

The attributes of a vector are:

- The elements (entries/coefficients/components) of a vector are the values that comprise it.
- The size (dimension) of a vector is the number of elements it contains.
- The vector of size n is called a n-vector
- The i^{th} element of a vector X is denoted by x_i

Consider the first sample vector $X^{(1)}$ is explicitly denoted as:

$$X^{(1)} = \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{bmatrix} = \begin{bmatrix} 76 \\ 126 \\ 38 \end{bmatrix}$$

Vector operations

Vector addition, subtraction and scalar multiplication works elementwise on a vector. For example, if X and Y are two word count vectors for two documents, then

- $(X + Y)$ is a word count vector of a new document created by combining the original two vectors.

- $(X - Y)$ gives the number of times more each word appears in the first than the second document.

Similarly, if Z is a vector representing an audio signal, the scalar vector product $2Z$ is the same audio signal perceived twice as loud.

Concept of Matrix

A matrix is a tabular arrangement of numbers written between rectangular brackets. For the 4 patient's 3 feature data, we have a (4×3) matrix. We also have a (8×8) gray scale image matrix, denoting 64 pixels; 0 for black and 1 for white.

For a (4×3) matrix, if is expressed as:

- 3 vectors (for each patient) repeating 4 times.
- 4 patient vectors repeating across 3 features.
- So, a $(m \times n)$ matrix means n feature vectors repeating m times.

Attributes of Matrix

The attributes of a matrix are:

- The size (dimension) of a matrix is the number of rows and columns. For example, the patient data matrix has 4 rows and 3 columns.
- A matrix having m rows and n columns is denoted as $(m \times n)$ matrix. For example, (4×3) patient matrix.
- The elements (entries/components) of a matrix X are the values that comprise it.
- To denote an element of a matrix X, we use the symbol $x_{i,j}$ that means the element in the i^{th} row and j^{th} column. For example, in patient data matrix, $x_{1,2}$ is 126

Matrix Operations

Matrix operations are performed elementwise. The transpose of a matrix X is denoted by X^T and it is obtained by switching the rows and columns of the matrix. So, a matrix X of shape (4×3) becomes of shape (3×4) after transpose operation.

Different expressions for Matrix

A matrix can be represented in many ways. As a vector is column wise entries, so to make a row vector usable, sometimes operations like transpose are used. For example,

- The vector $X^{(1)}$ is given by a (3×1) format, commonly known as column vector.

  $$X^{(1)} = \begin{bmatrix} 76 \\ 126 \\ 38 \end{bmatrix}$$

  However, the same matrix $X^{(1)}$ can be denoted by following notations, which is a (1×3) matrix, commonly known as row vector.

  $$X^{(1)^T} = \begin{bmatrix} 76 & 126 & 38 \end{bmatrix}$$

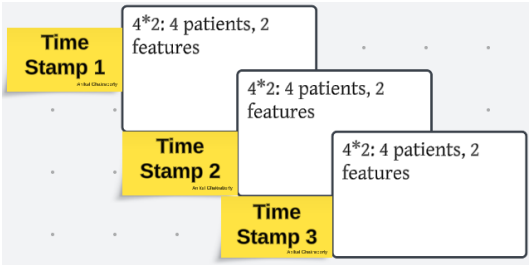
- Using the above approach, we can say that, a matrix X can be represented in terms of its columns and rows. For example, suppose X is the patient's data matrix, and we define $X^{(i)}$'s as sample vectors. Then, the matrix X can be represented in terms of sample vectors as following:

  $$X = \begin{bmatrix} X^{(1)^T} & X^{(2)^T} & X^{(3)^T} & X^{(4)^T} \end{bmatrix}$$

Tensors in Deep Learning

A tensor is typically a 3-dimensional (or more) arrangement of numbers. For example, a 3-channel RGB color image can be represented as a 3D tensor. The image tensor has shape $(337 \times 600 \times 3)$ - which means a (600×3) matrix that is repeated 337 times.

Suppose, we measured for 3 patients at 4 hourly time stamps 2 features (HeartRate HR, Blood Pressure BP). Then this tensor will have the shape of $(3 \times 4 \times 2)$ i.e. a (4×2) matrix repeating 3 times. The tensor will look like:



Chapter - 2: Norm, Dot product and Angle between two vectors

2 and 3 dimensional vectors can be easily visualized by using graphical tools. The norm of a vector is the length of the hypotenuse of the vector.

Norm of a vector

Norm denotes the geometric length of a vector. This definition should satisfy a set of certain intuitive properties. For example, when a vector is multiplied by a scalar, the resulting vector should have a length that is equal to the scaled (magnitude of the scalar) length of the original vector.

Such length-of-vector measurement definitions are called norms. There are two types of norms. They are l_1 and l_2 norm respectively. Consider a vector X, given as:

$$X = \begin{bmatrix} 1 \\ -2 \\ 3 \end{bmatrix}$$

Then the l_1 norm is given as: $||X||_1 = |1| + |-2| + |3| = 6$

Then the l_2 norm is given as: $||X||_2 = \sqrt{1^2 + (-2)^2 + 3^2} = \sqrt{14}$

Dot product of two vectors

The dot product of two vectors of the same size is an operation that returns a scalar value through an elementwise multiplication and addition. Suppose, X and Y are two n-vectors, defined as:

$$X = x_1 + x_2 + \ldots + x_n \text{ and } Y = y_1 + y_2 + \ldots + y_n$$

Then the dot product of X and Y is denoted as $X.Y$ is defined as: $X.Y = x_1y_1 + x_2y_2 + \ldots + x_ny_n$

Mathematically, it is precisely denoted as: $X.Y = \sum_{i=1}^{n} x_i y_i$

There is a relation between dot product and norm of a vector. For the vector X defined above, the dot product of X with itself is denoted as $X.X$ and it is nothing but the squared l_2 norm of X

$$X.X = x_1^2 + x_2^2 + \ldots + x_n^2 = ||X||_2^2$$

Application

- If a vector A denotes prices of n items and another vector B denotes the quantities of every item, then the vector $A.B$ will denote the total cost of buying those n items.
- If $X^{(1)}$ is the first patient vector and W is a vector of weights applied to three features then $W.X^{(1)}$ gives a weighted raw score for the 1st sample.

Angle between two vectors

The concept of angle between two n sized vectors X and Y, denoted by θ comes from Cauchy-Schwarz inequality. The angle between two vectors is defined as:

$$\cos\theta = \frac{X.Y}{||X||_2||Y||_2} \implies \theta = \cos^{-1}\left\{\frac{X.Y}{||X||_2||Y||_2}\right\}$$

The quantity that is inside the second brackets, always lie in the range $[-1, 1]$

There are some facts about angle between two vectors:

- The angle between two vectors vary between 0 and π radian.
- When the dot product of X and Y is 0, it is clear from above equation that the angle between X and Y is $\frac{\pi}{2}$ radian or 90 degrees. In this case, X and Y are called perpendicular/orthogonal vectors.
- If angle between X and Y is obtuse, then the dot product will be negative. For acute angle between X and Y, the dot product will be positive.

Calculation dissimilarity between vectors

How different two vectors X and Y are called the dissimilarity between two vectors. It can be calculated in two ways. They are -

1. Calculate distance dissimilarity as the norm of the different vectors, given as $||X - Y||_2$. Higher the norm value, higher the dissimilarity.
2. Calculate angular dissimilarity as $(1 - cos\,\theta)$; where θ is the angle between X and Y. If the value of the expression, $(1 - cos\,\theta)$ is 0, then there is no difference, if the expression value is 1, then the difference increases. As the expression value increases, the dissimilarity also increases. This approach is the best choice when the number of components in vectors are huge making it more applicable for NLP algorithms.
3. It is a good choice to standardize the vectors before calculating distance dissimilarity. It is helpful for avoiding some similar case situations.

Chapter - 3: Matrix Vector Product, Matrix-Matrix Product, Hadamard Product

Matrix Vector Product

The idea of a dot product between two vectors can be easily extended to define a matrix vector product. If A is a (2×3) matrix and X is a (3×1) vector, then the result will be a (2×1) vector. A matrix vector product can be seen as sequence of dot products between the rows of matrix A and vector X. The matrix vector product can also be seen as a linear combination of the columns of matrix A using the elements of the vector X as multipliers.

$$A = \begin{bmatrix} 1 & 2 & 4 \\ 2 & -1 & 3 \end{bmatrix}; X = \begin{bmatrix} 4 \\ 2 \\ -2 \end{bmatrix}$$

$$AX = \begin{bmatrix} 1 & 2 & 4 \\ 2 & -1 & 3 \end{bmatrix} \begin{bmatrix} 4 \\ 2 \\ -2 \end{bmatrix} = 4 \begin{bmatrix} 1 \\ 2 \end{bmatrix} + 2 \begin{bmatrix} 2 \\ -1 \end{bmatrix} + (-2) \begin{bmatrix} 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Application

Suppose, a matrix P of shape $(m \times n)$ gives the prices of n goods from m suppliers and q is an n-vector of quantities of the n goods. Then the result $C = Pq$ is an m-vector that gives the total purchase cost from each m supplier.

Matrix Matrix Product

It is a natural and repeated extension of a matrix-vector product. Suppose, A & B be two matrices of shapes (2×3) and (3×2) respectively. Then the matrix matrix product, $A \times B$ is of shape (2×2)

$$A = \begin{bmatrix} 1 & 2 & 4 \\ 2 & -1 & 3 \end{bmatrix}; B = \begin{bmatrix} 4 & -1 \\ 2 & 0 \\ -2 & 3 \end{bmatrix}$$

$$|A \times B| = 2 \times 2$$

$$A \times B = \begin{bmatrix} R_1C_1 & R_1C_2 \\ R_2C_1 & R_2C_2 \end{bmatrix}$$

$$R_1, R_2 : \text{Rows of } A$$

$$C_1, C_2 : \text{Columns of } B$$

In General, if A is a $(m \times n)$ matrix and B is a $(n \times p)$ matrix, then the matrix product AB of order $(m \times p)$ is given as:

$$AB_{m \times p} = \begin{bmatrix} a^{(1)}b_1 & a^{(1)}b_2 & \dots & a^{(1)}b_p \\ a^{(2)}b_1 & a^{(2)}b_2 & \dots & a^{(2)}b_p \\ \vdots & \vdots & \vdots & \vdots \\ a^{(m)}b_1 & a^{(m)}b_2 & \dots & a^{(m)}b_p \end{bmatrix} = [Ab_1 \quad Ab_2 \quad Ab_3]$$

Any entry of the matrix AB is denoted by $ab_{i,j}$ which is defined as:

$$ab_{i,j} = (i^{th}\ row\ of\ A) \times (j^{th}\ column\ of\ B)$$

Hadamard Product

The Hadamard product, denoted as $\odot$ of two matrices A and B of same sizes $(m \times n)$ is an elementwise product resulting in a matrix of the same size.

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}; B = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{bmatrix}$$

$$A \circ B = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & \dots & a_{1n}b_{1n} \\ a_{21}b_{21} & a_{22}b_{22} & \dots & a_{2n}b_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1}b_{m1} & a_{m2}b_{m2} & \dots & a_{mn}b_{mn} \end{bmatrix}$$

Chapter - 4: Weights, Bias, Raw Scores and Loss Function

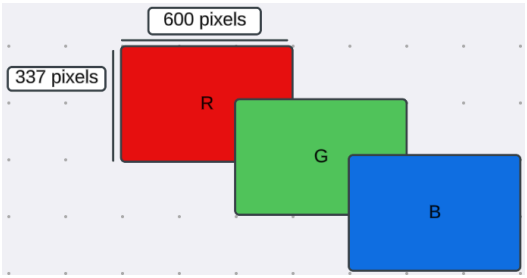
Classification problem in Practice

Classifying a sample into one of the known categories (or classes) is a common challenge across different domains. It may be either for simple label data classification, or for image classification. However, for Computer Vision related tasks, this classification problem is the most popular thing. So, in this section, the whole concept will be discussed using RGB images.

The main aim of this problem is to classify an image into its three possible categories - Lion/Tiger/Leopard. All the images that are selected for this purpose is color image, internally represented as a 3-dimensional tensor of shape $(337 \times 600 \times 3)$ - the tensor of integers that lies in the range 0 to 255.

Linear Classifier: Basic Idea

Consider the example of classifying a sample into a Lion/Tiger/Leopard image. Each image is 600 pixels in length and 337 pixels in width. This whole image is repeated 3 times for 3 channels - Red, Green and Blue. This can be easily visualized as:



A training image that can be seen as a vector X with $(337 \times 600 \times 3) = 606600$ numbers. Now, the next step is to calculate 3 class scores as:

- Weights matrix W, having shape (3×606600)
- The feature vector X of shape (606600×1)
- The bias term b , represented as bias matrix of shape (3×1)

The expression, $[WX + b]$ results in a (3×1) matrix. This can be used to assess how good the choices of W and b are. Here, W and b are the parameters of this model.

Weights, Bias and Raw Scores

Using the training sample (say Lion image), we devise a computational approach for calculating the optimal weights matrix W and bias vector b. Consider three output categories - Lion/Tiger/Leopard. Also consider 4 features for each category. Then the weights matrix W will be of shape (3×4). Consider the first image vector as X, which is of shape (4×1). Consider the bias term for each output category. So, the bias vector will contain three components, indicating that it has shape (3×1). Now, generate this terms arbitrarily as:

$$W = \begin{bmatrix} 0.1 & -0.1 & 0 & 0.5 \\ 2.3 & 0.8 & 1.2 & 0.5 \\ 0 & -1 & 0.5 & 1.0 \end{bmatrix}; X = \begin{bmatrix} 26 \\ 100 \\ 90 \\ 80 \end{bmatrix}; b = \begin{bmatrix} -1.0 \\ -1.5 \\ 1.0 \end{bmatrix}$$

Using the matrix and vectors defined above, we define another vector, called as raw scores vector, which is denoted by Z and it is calculated as following:

$$Z = WX + b = \begin{bmatrix} 32.6 \\ 287.8 \\ 25 \end{bmatrix} = \begin{bmatrix} \text{Raw score for Lion category} \\ \text{Raw score for Tiger category} \\ \text{Raw score for Leopard category} \end{bmatrix}$$

The current set of weight and bias values lead to a maximum raw score of 287.8 for the incorrect class (as Lion is training image and Tiger is predicted class). Now the main question is how can we quantify unhappiness? This leads to the basic idea of loss function.

Loss Function - Intuition

Given that, we know the true output class for a set of training samples, we can quantify the unhappiness for a particular set of weights W and bias b values using raw scores for three training samples as - suppose we take three samples. Also suppose, for the first sample, the actual category is Lion, for the second sample the actual category is Tiger and for the third sample the actual category is Leopard. Then the raw scores with happiness can be given as:

Raw Scores	Category		
	Sample – 1: Lion is Ac	Sample – 2: Tiger is Ac	Sample – 3: Leopard is Ac
Lion	**5.6**	-1.8	2.0
Tiger	6.4	**10.2**	5.4
Leopard	-4.6	3.5	**-8.6**
Happiness	Little Unhappy	Satisfied	Very Unhappy

If the incorrect class raw scores are greater than correct class raw scores then it contributes to loss. So, from the above table, the loss value can be calculated for each sample as:

$$\text{Loss for Sample - 1: } L_1 = \{\max(0, 6.4 - 5.6) + \max(0, -4.6 - 5.6)\} = 0.8$$

$$\text{Loss for Sample - 2: } L_2 = \{\max(0, -1.8 - 10.2) + \max(0, 3.5 - 10.2)\} = 0$$

$$\text{Loss for Sample - 3: } L_3 = \{\max(0, 2 + 8.6) + \max(0, 5.4 + 8.6)\} = 24.6$$

$$\text{The average training loss is calculated as: } \frac{0.8 + 0 + 24.6}{3} = 8.5$$

Hinge Loss Function

Suppose, there are n training samples (containing both independent and dependent variable), denoted as (X^i, y^i); where i stands for the number of samples. Here X^i is the sample vector and y^i is the correct class label. We choose an initial weights matrix W along with bias vector b. Then the i^{th} sample's raw score vector, denoted by z^i is defined as:

$$z^i = WX^i + b$$

The multiclass SVM hinge loss function for the i^{th} sample is given as:

$$L_i = \sum_{j \neq y^i} \max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)} + 1\right)$$

$$\text{Where, } z_j^{(i)} = i^{th} \text{ sample } j^{th} \text{ category raw score, } j \text{ is not the correct class}$$

$$z_{y^{(i)}}^{(i)} = \text{Correct class raw scoe for the } i^{th} \text{ sample}$$

$$j \text{ is called the offset}$$

$$\text{So, } L_i = \sum_{j \neq y^i} \max\left(0, \left[WX^{(i)} + b\right]_j - \left[WX^{(i)} + b\right]_{y^{(i)}} + 1\right)$$

$$\text{The average training data loss is defined as: } \frac{1}{n}\sum_{i=1}^{n} L_i$$

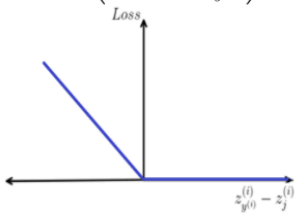
$$\text{Where, } \sum_{i=1}^{n} L_i \text{ is called training loss}$$

Graphical representation of Loss Function

Visualizing different loss functions considering contribution from one incorrect class: There are three types of loss functions that are most popular in such scenarios. They are - Perceptron Loss, Hinge Loss and Squared Hinge Loss.

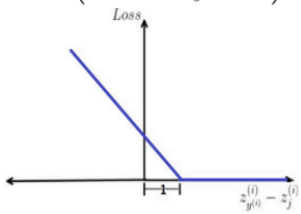
Perceptron Loss

$$\max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)}\right)$$



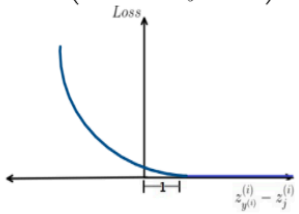
Hinge Loss

$$\max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)} + 1\right)$$



Squared Hinge Loss

$$\max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)} + 1\right)^2$$



Softmax Loss Function

It is possible to turn the raw scores vectors into a vector of probabilities. Firstly the raw scores are raise to the power of exponential function e and then those huge numbers are normalized. With an example this is given as:

Raw Score	Lion	Exp (Raw Scores)	Probability Conversion – Normalization
Lion	5.6	$e^{5.6}$	$\frac{e^{5.6}}{e^{5.6}+e^{6.4}+e^{-4.6}} = 0.31$
Tiger	6.4	$e^{6.4}$	$\frac{e^{6.4}}{e^{5.6}+e^{6.4}+e^{-4.6}} = 0.69$
Leopard	-4.6	$e^{-4.6}$	$\frac{e^{-4.6}}{e^{5.6}+e^{6.4}+e^{-4.6}} = 0.00$

Formally, the softMax function takes a vector as input and outputs a same sized vector of probabilities through exponentiation and normalization. The Lion probability is not 1, rather it is now 0.31.

We use the natural logarithm function to determine the loss function value for the SoftMax classifier problem. Suppose a training sample has raw score vector $z = WX + b$ and it belongs to the correct class y. Then $SoftMax(z)$ gives the class probability vector. So, $[SoftMax(z)]_y$ is the correct class probability and we are happy until its value is closer to 1. If the value is closer to 0, then we are not happy with the choices of W and b. The softmax loss function for the sample is defined as: $softMax_{loss} = -\ log([SoftMax(z)]_y)$.

So, if we are happy about our choices of W and b, then the value of $[SoftMax(z)]_y$ will be 1 which suggests that the loss will be $-\ log(1) = 0$. If the value of $[SoftMax(z)]_y$ is close to 0, then we are not happy and hence the loss is $-\ log(0) = \infty$

So, in short the loss function value is the negative of the logarithm of the correct class's probability.

Chapter - 5: Optimization Using Gradient Descent Method

Optimization Setup

Given a training sample, the goal is to find optimal values for the weights and biases that can minimize the average training loss. Consider a function of weights as $L(w) = w^2 + 3$. The main aim is to determine how can we tweak the input w from its current value so that the current value of $L(w)$ decreases. This can be done by moving the value of w to its left (to decrease) or to its right (to increase).

The next question that arises is can we quantify the sensitivity of the output $L(w)$ with respect to a small change in the input w? From here the concept of sensitivity arises.

Sensitivity

The sensitivity of the functional output L with respect to a small change in the functional input w is given as:

$$\frac{Change\ in\ output}{Change\ in\ Input} = \frac{\Delta L}{\Delta w}$$

Now, when we move the w value in the right direction (positive direction), both the ΔL and Δw values are positive. Hence, the sensitivity is also positive. Similarly, when we move the w value in the left direction (negative direction), both the ΔL and Δw values are negative. Hence, the sensitivity is also positive in this case.

Positive sensitivity means the change in input is directly proportional to the change in output. They both increase or decrease in the same direction. Negative sensitivity means the change in input is inverse proportional to the change in output. If input increases, output decreases and vice-versa. Numerically, sensitivity value can never be 0. However, it can be close to 0 sometimes.

Gradient for Lower Dimension

The sensitivity of the function $L(w)$ w.r.t. Its input w can be functionally represented using the gradient denoted by the symbol $\nabla_w(L)$. This is simply the derivative of the function $L(w)$ w.r.t. w.For example, for the function $L(w) = w^2 + 3$, $\nabla_w(L)$ will be $2w$ that comes as 4 when $w = 2$. So, we can say that at $w = 2, \nabla L = 4 \times \nabla w$. Higher gradient value implies higher sensitivity.

Chain rule for lower dimension

The chain rule is used for calculating gradients in a hierarchical way by using intermediate variables. Consider the function $L(w) = \frac{1}{1+e^{-w}}$. There are two ways or methods of calculating the derivative of this function.

First Method: By using basic mathematical operations:

$$L(w) = \frac{1}{1+e^{-w}} = (1+e^{-w})^{-1}$$

$$\Rightarrow \nabla_w(L) = (-1).(1+e^{-w})^{-2}.(-e^{-w})$$

$$\Rightarrow \nabla_w(L) = \frac{e^{-w}}{(1+e^{-w})^2} = \frac{1}{1+e^{-w}}.\frac{e^{-w}}{1+e^{-w}} = \frac{1}{1+e^{-w}}.\left\{1-\frac{1}{1+e^{-w}}\right\}$$

$$\Rightarrow \nabla_w(L) = L(w) \times \{1 - L(w)\}$$

Second Method: The second method deals with substitution, which is a way of applying chain rule:

Computation Graph: $L \rightarrow z \rightarrow w$

Given, $L(w) = \frac{1}{1+e^{-w}}$

Let, $z = 1 + e^{-w}$, then $L(z) = \frac{1}{z}$

Now by chain rule, we can write that:

$$\nabla_w(L) = \nabla_z(L) \times \nabla_w(z) = \left(-\frac{1}{z^2}\right) \times (-e^{-w})$$

Substitution the z value back in final expression we get:

$\nabla_w(L) = L(w) \times \{1 - L(w)\}$

Chain rule kind of reduces the length of calculation. It is specifically useful for higher dimension gradient determination tasks. By observing the resultant gradient, it can be say that the value of $\nabla_w(L)$ will be 0 when either $L(w) = 0$ or either $L(w) = 1$

Gradient for Higher dimension

The gradient for higher dimensions can be calculated as: input shape × output shape. This is given by the following image as:

Function	I/O shapes	Grad Shape	Gradient
$L(w) = w^T w$ w is a $(n \times 1)$ vector	Input: n Output: 1	$(n \times 1)$	$\nabla_w(L) = 2w$
$L(w) = X^T w$ X is a $(n \times 1)$ vector	Input: n Output: 1	$(n \times 1)$	$\nabla_w(L) = X$
$L(w) = (w_1 - 2)^2 + (w_2 + 3)^2$	Input: 2 Output: 1	(2×1)	$\nabla_w(L) = \begin{bmatrix}\nabla_{w_1}(L) \\ \nabla_{w_2}(L)\end{bmatrix}$ $\nabla_w(L) = \begin{bmatrix}2(w_1 - 2) \\ 2(w_2 + 3)\end{bmatrix}$
$L(z) = Wz$ W is a $(p \times k)$ matrix	Input: $(k \times 1)$ Output: $(p \times 1)$	$(k \times p)$	$\nabla_z(L) = W^T$

Chain rule for higher dimension

Calculate the gradient of the function, given as:

$$L(W) = \frac{1}{1 + e^{-W^T X}}$$

By using chain rule, some values can be substituted in the main expression as:

$$L(W) = \frac{1}{1 + e^{-W^T X}} : \begin{cases} L(z_1) = \dfrac{1}{z_1} \\ z_1(z_2) = 1 + e^{z_2} \\ z_2(W) = -W^T X \end{cases}$$

The Computation Graph becomes: $L \rightarrow z_1 \rightarrow z_2 \rightarrow W$

$$\text{So, } \nabla_W(L) = \nabla_W(z_2) \times \nabla_{z_2}(z_1) \times \nabla_{z_1}(L)$$

$$\Rightarrow \nabla_W(L) = \nabla_W\left(-W^T X\right) \times \nabla_{z_2}\left(1 + e^{z_2}\right) \times \nabla_{z_1}\left(\frac{1}{z_1}\right)$$

$$\Rightarrow \nabla_W(L) = (-X) \times e^{z_2} \times \left(-\frac{1}{z_1^2}\right)$$

$$\Rightarrow \nabla_W(L) = \frac{e^{-W^T X}}{\left(1 + e^{-W^T X}\right)^2} . X \text{ [Substituting the values of } z_1 \text{ and } z_2]$$

Here, $\nabla_W(L)$ is a $(n \times 1)$ vector

Gradient Descent Method

The gradient (considered as a vector) is the direction of steepest ascent. It means that the negative of the gradient is the direction of steepest descent. Consider the function, $L(w) = w^2 + 3$ at $w = 2$. The gradient at $w = 2$ is denoted as $\nabla_w(L)|_{w=2} = [4]$. So, we move in the negative direction of the gradient i.e. in the direction of $-\nabla_w(L)$ which is -4. Now the question is how much should we move. We move by a specific small amount known as the learning rate α. We keep repeating this process iteratively until the gradient is close to zero or there is no significant change in w values. Thus we can approach to the point w at which $L(w)$ has the smallest value. The iteration process looks like:

$$w_{new} = w_{old} - \alpha \times \nabla_w(L)_{w_{old}}$$

The same update works for higher dimensions, given as:

$$\text{Suppose, } L(w) = (w_1 - 2)^2 + (w_2 + 3)^2$$

$$\text{Then the gradient update iteration expression is given as:}$$

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}_{new} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}_{old} - \alpha \begin{bmatrix} \nabla_{w_1} L(w)_{w_{old}} \\ \nabla_{w_2} L(w)_{w_{old}} \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}_{new} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}_{old} - \alpha \begin{bmatrix} 2(w_1 - 2) \\ 2(w_2 + 3) \end{bmatrix}_{w_{old}}$$

Chapter - 6: SoftMax and Logistic Regression Classifier

Bias Trick

In calculating the raw score of a sample $z = WX + b$, it is possible to absorb the bias values into the weights matrix W. Suppose, we initiate the weights matrix W, a sample vector X and the bias vector b for a classification problem having three possible output categories.

$$W = \begin{bmatrix} 0.1 & -0.1 & 0 & 0.5 \\ 2.3 & 0.8 & 1.2 & 0.5 \\ 0 & -1 & 0.5 & 1.0 \end{bmatrix}; X = \begin{bmatrix} 26 \\ 100 \\ 90 \\ 80 \end{bmatrix}; b = \begin{bmatrix} 1.0 \\ -1.5 \\ 1.0 \end{bmatrix}$$

After absorbtion of the bias values in the weights matrix, the weights matrix W and the image vector X becomes:

$$W = \begin{bmatrix} 0.1 & -0.1 & 0 & 0.5 & 1.0 \\ 2.3 & 0.8 & 1.2 & 0.5 & -1.5 \\ 0 & -1 & 0.5 & 1.0 & 1.0 \end{bmatrix}; X = \begin{bmatrix} 26 \\ 100 \\ 90 \\ 80 \\ 1 \end{bmatrix}$$ – This approach is useful for shape matching purpose

Some things to note:

- The last column of the weights matrix hold the bias values
- The bias feature with value 1 gets appended to the sample image vector.
- This approach helps in simplifying gradient calculations for computing optimal weights and bias values without considering the bias separately.

SoftMax Classifier Setup

Suppose we have n samples with p features: $X^{(1)}$, $X^{(2)}$, ..., $X^{(n)}$ with labels $y^{(1)}$, $y^{(2)}$, ..., $y^{(n)}$ that belongs to k classes. So, in normal situation, the weights matrix W is of shape $(k \times p)$. After performing the bias trick, one column gets added in the weights matrix and hence the shape changes to $(k \times \{p+1\})$. After performing the bias trick, the data matrix X is of shape $(\{p+1\} \times n)$. Note that the columns of the data matrix correspond to the samples with the last row equal to ones.

The raw scores for all samples can be computed as a $(k \times n)$ matrix given as:

$$Z = WX = W\begin{bmatrix} X^{(1)} & X^{(2)} & \dots & X^{(n)} \end{bmatrix}$$
$$\Rightarrow Z = \begin{bmatrix} WX^{(1)} & WX^{(2)} & \dots & WX^{(n)} \end{bmatrix} = \begin{bmatrix} Z^{(1)} & Z^{(2)} & \dots & Z^{(n)} \end{bmatrix}$$

The SoftMax loss for the i^{th} sample is given as: $-\ log([SoftMax(Z^{(i)})]_{y^{(i)}})$

SoftMax Classifier Gradient

The average training softmax loss is given as:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n} L_i = \frac{1}{n}\sum_{i=1}^{n}\left[-\log\left(\frac{e^{W_{y^{(i)}}^T X^{(i)}}}{\sum_{j=1}^{k} e^{W_j^T X^{(i)}}}\right)\right]$$

The gradient $\nabla_w(L)$ has shape $(k \times \{p+1\})$. Solving big equations, we can come to the little expression for the gradient of the loss function. This is given as:

$$\nabla_w(L) = \frac{1}{n} P_{adj} X^T$$

The gradient descent updation formula for this SoftMax classifier gradient can be expressed as:

$$w_{new} = w_{old} - \alpha \times (\nabla_w(L))_{w_{old}}$$
$$\Rightarrow w_{new} = w_{old} - \alpha \times \left(\frac{1}{n} P_{adj} X^T\right)$$

Logistic Regression Classifier Setup

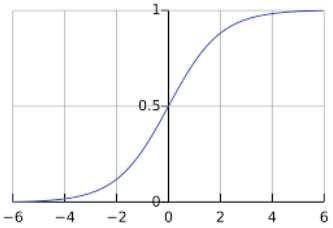
Logistic regression is a classification problem, particularly useful for binary classification. In such problems, there are two output classes labeled as 0 and 1. Just like the softmax classifier, we calculate raw scores for each sample. The raw score for the i^{th} sample is $z^{(i)} = W^T X^{(i)}$ (assuming bias trick): the matrix W has only one row W^T.

The raw score $z^{(i)}$ is used to calculate the predicted probability that the i^{th} sample belongs to its correct class, which in turn is used to define the loss for the i^{th} sample.

Now, the question is, how to get the predicted probability? The answer is by observing the sigmoid function. The sigmoid function, denoted as $\sigma(z)$ is given as:

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

Graphically, this is represented as:



The X axis is denoted by z and the Y axis is the value of $\sigma(z)$. At $z = 0$, $\sigma(z) = 0.5$, as $z \to \infty$, $\sigma(z) \to 1$ and as $z \to -\infty$, $\sigma(z) \to 0$

The predicted probability that the i^{th} sample belongs to its correct class $y^{(i)}$ is denoted as $\hat{y}^{(i)}$. The sigmoid function is applied to the raw score $z^{(i)}$ and it is used to predict the probabilities as:

$$\hat{y}^{(i)} = \begin{cases} \sigma\left(z^{(i)}\right) = \sigma\left(W^T X^{(i)}\right) & \text{if correct class } y^{(i)} = 1 \\ 1 - \sigma\left(z^{(i)}\right) = 1 - \sigma\left(W^T X^{(i)}\right) & \text{if correct class } y^{(i)} = 0 \end{cases}$$

So, in compact expression, we can write that:

$$\hat{y}^{(i)} = \left\{\sigma\left(W^T X^{(i)}\right)\right\}^{y^{(i)}} \left\{1 - \sigma\left(W^T X^{(i)}\right)\right\}^{1-y^{(i)}}$$

Logistic Regression Classifier Loss Function and Gradient

The loss of the logistic regression classifier for the i^{th} sample is the negative logarithm of the predicted probability that the i^{th} sample belongs to its correct class $y^{(i)}$. Mathematically, this is given as:

$$L_i(w) = -\log\left(\hat{y}^{(i)}\right) = -\log\left[\left\{\sigma\left(W^T X^{(i)}\right)\right\}^{y^{(i)}} \left\{1 - \sigma\left(W^T X^{(i)}\right)\right\}^{1-y^{(i)}}\right]$$

The average training loss is given as: $L(w) = \frac{1}{n}\sum_{i=1}^{n} L_i(w)$

The gradient of the training loss is given as:

$$\nabla_w(L) = \frac{1}{n}\sum_{i=1}^{n} \nabla_w(L_i) = \frac{1}{n} X\left[\sigma\left(X^T W\right) - y\right]$$

The shape of weights matrix W, is $|W| = (p+1) \times 1$

The shape of data matrix X, $|X| = (p+1) \times n$; So, $|X^T| = n \times (p+1)$

So, combining all, we have $Z = W^T X$ and $|Z| = n \times 1$

Following intermediate variables and computation graph can be used to derive the i^{th} samples logistic regression loss function's gradient by applying chain rule on it.

We have: $L_i = -\log\left[\left\{\sigma\left(W^T X^{(i)}\right)\right\}^{y^{(i)}} \left\{1 - \sigma\left(W^T X^{(i)}\right)\right\}^{1-y^{(i)}}\right]$

Let, $Z^{(i)} = W^T X^{(i)}$ and $a^{(i)} = \sigma\left(Z^{(i)}\right)$

Then, the transformed equation becomes: $L_i = -\log\left[\left\{a^{(i)}\right\}^{y^{(i)}} \left\{1 - a^{(i)}\right\}^{1-y^{(i)}}\right]$

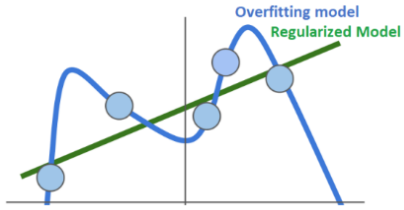
By applying chain rule, we can write that:

$$\nabla_W(L) = \nabla_W\left(Z^{(i)}\right) \times \nabla_{Z^{(i)}}\left(a^{(i)}\right) \times \nabla_{a^{(i)}}(L_i)$$

Chapter - 7: Regularization and Batch Processing

Why Regularization?

Regularization is an approach to prevent the model from doing too well on training data. Models that overfit the training data typically perform poorly on test data. Model overfitting happens when some features of the training sample drives the learning process or the model learns noise in the data. Regularization is particularly helpful for high dimensional data sets (more features than samples). The result of regularization can be shown as:



Regularization Approach

Regularization is achieved by adding to the average training data loss constraints on the weights. Note that regularization is not applied on the bias term, as the bias term doesn't contribute to any kind of learning. There are three types of regularization namely: Lasso or L1 regularization and Ridge or L2 regularization and Elastic-net regularization, which is a combination of both Lasso and Ridge regularization. Mathematically, they are defined as:

$$L = \frac{1}{n}\sum_{i=1}^{n} L_i(w) + \begin{cases} \lambda \sum_{r,s} |w_{rs}| & \text{Lasso or } L_1 \text{ Regularization} \\ \lambda \sum_{r,s} |w_{rs}|^2 & \text{Ridge or } L_2 \text{ Regularization} \\ \lambda \left[\alpha \sum_{r,s} |w_{rs}| + (1-\alpha) \sum_{r,s} |w_{rs}|^2\right] & \text{Elastic-net Regularization} \end{cases}$$

In the above expression, $\lambda > 0$ is called regularization strength and it is a hyper-parameter that has to be tuned. All regularization approaches tend to drive the weight matrix values close to zero.

However, use of Lasso or Ridge regularization is case specific. Lasso (L1) regularization typically results in a smaller subset of non-zero weights than Ridge (L2) regularization. Lasso (L1) regularization is used when we know which features matter the most by previous experience. On the other hand, when we don't have any previous experience or knowledge on which feature subset matters the most, we use Ridge (L2) regularization.

Regularization Loss and Gradient

Consider the Ridge (L2) regularization term (also called as regularization loss) for a weights matrix W corresponding to k classes and p features. Mathematically, this is given as:

$$L_{reg}(W) = \sum_{r,s} ||w_{rs}||^2$$

$$\Rightarrow L_{reg}(W) = ||w_1||^2 + ||w_2||^2 + \ldots + ||w_k||^2$$

$$\Rightarrow L_{reg}(W) = w_1^T w_1 + w_2^T w_2 + \ldots + w_k^T w_k$$

The input shape is $(k \times p)$ and output shape is 1. The shape of the gradient is thus $(k \times p)$, given as:

$$\nabla_W(L) = \begin{bmatrix} (\nabla_{w_1}(L_{reg}))^T \\ (\nabla_2(L_{reg}))^T \\ \vdots \\ (\nabla_k(L_{reg}))^T \end{bmatrix} = 2\begin{bmatrix} w_1^T \\ w_2^T \\ \vdots \\ w_k^T \end{bmatrix} = 2W$$

Batch Processing

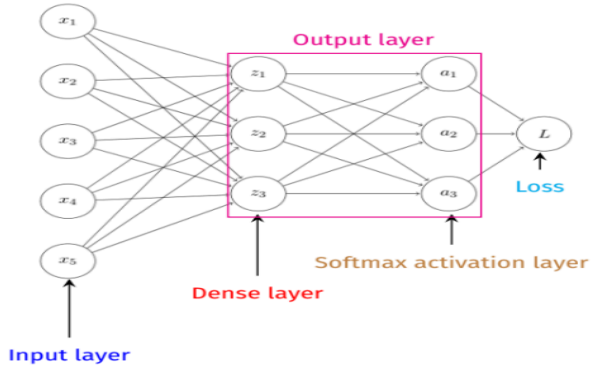
The average training data loss, calculated as $\frac{1}{n}\sum_{i=1}^{n} L_i$ has contributions from all the training samples. If training samples are images, it could be computationally expensive to calculate the loss for all training samples and then average training loss. Another way to do the same is called Batch Processing, where the training samples are randomly shuffled and split into batches of a specific batch size (e.g. 16, 32, 64 etc) and then the weights are trained using a smaller set of samples in training batches.

The weights will be updated more frequently (initially inaccurately) using the gradient descent as a small number of batch training samples. An Epoch is when the weights have been updated using all training samples through batches; the model has seen all the training samples once.

Chapter - 8: One Hidden Layer Neural Network: Architecture, Notation and Activation Function

SoftMax classifier as a zero hidden layer Neural Network

A layered visualization applying the SoftMax classifier to a sample X with 5 features x_1, x_2, x_3, x_4, x_5 and correct output label y from 3 possible output labels is given as:



The loss function is given as L. The bias feature is excluded for clarity purposes.

Input Layer: The sample vector X, that has five features x_i's for $i = 1, 2, 3, 4, 5$

Dense Layer: Owns the (3×5) weights matrix W and calculates raw scores as $z = WX$. So, the dense layer learns the matrix W

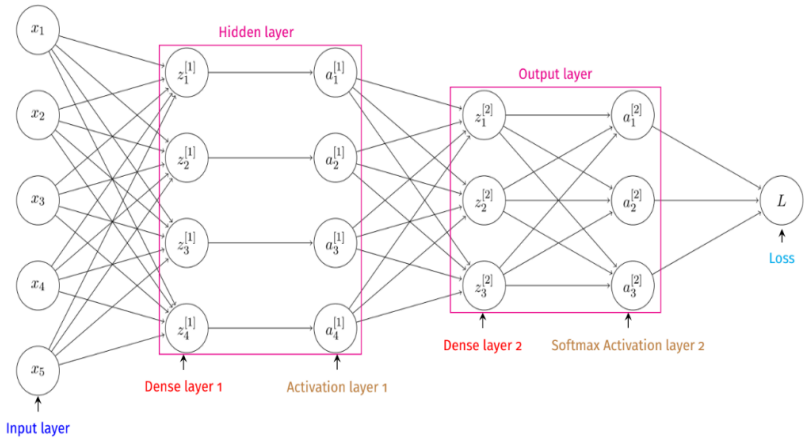
SoftMax Activation Layer: Activates the raw scores vectors as $a = SoftMax(z)$

Loss: The loss function L is calculated as $L = -log([a]_y)$

Here, a_i is the predicted probability that the object will belong to correct class i. The Dense Layer and SoftMax Activation layer are jointly called as Output layer for Zero hidden layer Neural Network. The name 'Zero Hidden Layer' means after the input layer, there is only the output layer.

Single Hidden Layer Neural Network: Architecture

The architecture of Single Hidden Layer Neural Network is given as:



The superscript $[1]$ in $z_1^{[1]}$ and $a_1^{[1]}$ denotes the dense layer 1 and activation layer 1 of Dense Layer 1. Typically, the superscript denotes the layer index. In the hidden layer, point wise activation happens. So one-one mapping. In the output layer SoftMax activation happens. So, one-to-many mapping. The dense layer 1 understands the features by the weights matrix. Activation layer 1 is a type of representation for the input layer.

Single Hidden Layer Neural Network: Notation

In the single hidden layer neural network architecture given above, there are 5 input features (5 nodes) and in the hidden layer 1, there are 4 nodes.

Input Layer: It is of shape (5×1)

Dense Layer 1: owns a (4×5) weights matrix $W^{[1]}$. It creates componentwise activated raw scores vector $Z^{[1]}$ and it is of shape (4×1). So, the raw score vector $Z^{[1]}$ for Dense Layer 1 is given as:

$$Z^{[1]} = \begin{bmatrix} z_1^{[1]} \\ z_1^{[2]} \\ z_1^{[3]} \\ z_1^{[4]} \end{bmatrix}$$

Activation Layer 1: Activates the raw score vector $Z^{[1]}$ by a activation function g, which is not SoftMax activation. So, similarly as $Z^{[1]}$, the activation layer 1 vector $a^{[1]}$ is also of shape (4×1). Mathematically, $a^{[1]}$ is given as:

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_1^{[2]} \\ a_1^{[3]} \\ a_1^{[4]} \end{bmatrix} = \begin{bmatrix} g\left\{z_1^{[1]}\right\} \\ g\left\{z_1^{[2]}\right\} \\ g\left\{z_1^{[3]}\right\} \\ g\left\{z_1^{[4]}\right\} \end{bmatrix}$$

Dense Layer 2 for Dense Layer 2, there are 4 nodes in the previous activation layer 1 and 3 nodes in the dense layer 2. So, the Dense Layer 2 owns a weights matrix $W^{[2]}$, which is of shape (3×4). It creates componentwise activated raw scores vector $Z^{[2]}$ and it is of shape (3×1). So, the raw score vector $Z^{[2]}$ for Dense Layer 2 is given as:

$$Z^{[2]} = \begin{bmatrix} z_1^{[2]} \\ z_2^{[2]} \\ z_3^{[2]} \end{bmatrix}$$

Activation Layer 2/ SoftMax Activation Layer: Activates the raw score vector $Z^{[2]}$ by SoftMax activation function. So, similarly as $Z^{[2]}$, the activation layer 2 vector $a^{[2]}$ is also of shape (3×1). Mathematically, $a^{[2]}$ is given as:

$$a^{[2]} = \begin{bmatrix} a_1^{[2]} \\ a_2^{[2]} \\ a_3^{[2]} \end{bmatrix} = SoftMax\left(Z^{[2]}\right) = SoftMax\left(\begin{bmatrix} z_1^{[2]} \\ z_2^{[2]} \\ z_3^{[2]} \end{bmatrix}\right)$$

Loss Function: The loss function L is calculated as, $L = -\ log\{(a^{[2]})_y\}$

Note that, the SoftMax activation function is only used in the layer that is previous to the loss function calculation.

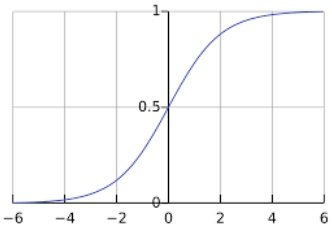
Activation Function: Need and Types

Calculating the raw scores at each dense layer can be related to weighting the features. Activating those raw scores can be related to how each neuron (nodes) raw score fires it so that relevant information is related and pushed further by the neuron.

The simplest activation function is given as $g(z) = z$ (linear/identity activation function); which simply pushes the input forward. However, the identity activation function doesn't lead to non-linear learning that can capture potential non-linear relationships between the input and output. So, non-linear learning is achieved by using non-linear activation functions. Some examples of non-linear activation functions includes -

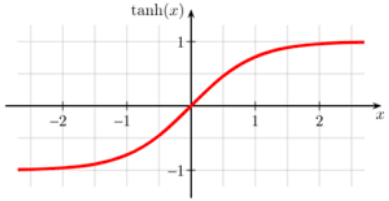
Sigmoid Activation Function: It activates the output between 0 and 1. It almost shows linear behavior for small inputs around 0. Mathematically, this is given as:

$$g(z) = \frac{1}{1+e^{-z}}$$



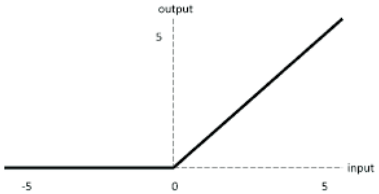
Hyperbolic Tangent Activation Function: It activates the input between -1 and 1. It is 0 centered. Behaves linearly for small values around 0. Mathematically, this is given as:

$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$



ReLU Activation Function: ReLU stands for Rectified Linear Unit. In this function, the negative values are mapped to 0 and non-negative values are transmitted as much. This is a very simple and effective activation function. Mathematically, this is given as:

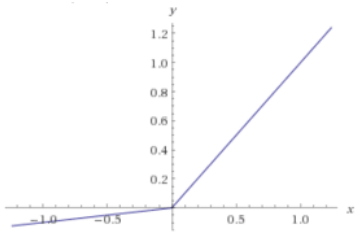
$$g(z) = max(0, z)$$



Leaky ReLU Activation Function: Stands for Leaky Rectified Linear Unit. This is also simple and effective, more effective than ReLU activation function. Mathematically, this is defined as:

$$g(z) = max(0.01z, z)$$

So, for negative z values ($z < 0$), the functional value is $0.01z$. For positive z values ($z > 0$), the functional value is z. The function is undefined at $z = 0$



Gradients of Activation Function

The gradients of those non-linear activation functions are given as:

Gradient of Sigmoid Activation Function: The gradient of Sigmoid Activation function is given as:

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

$$\Rightarrow \nabla_z \{g(z)\} = (-1)\left(1 + e^{-z}\right)^{-2}\left(-e^{-z}\right)$$

$$\Rightarrow \nabla_z \{g(z)\} = \frac{e^{-z}}{\left(1 + e^{-z}\right)^2} = \frac{1}{1 + e^{-z}} \cdot \frac{e^{-z}}{1 + e^{-z}} = g(z)\{1 - g(z)\}$$

When $z \to \infty$, $g(z) \to 1 \Rightarrow \nabla_z \{g(z)\} = 0$

When $z \to -\infty$, $g(z) \to 0 \Rightarrow \nabla_z \{g(z)\} = 0$

So, for higher magnitude of input raw scores, the gradient is 0, hence the gradient vanishes.

Gradient of Hyperbolic Tangent Activation Function: The gradient of Hyperbolic Tangent Activation function is given as:

$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$\Rightarrow \nabla_z \{g(z)\} = 1 - \left(\frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}\right)^2 = 1 - [g(z)]^2$$

When $z \to \infty$, $g(z) \to 1 \Rightarrow \nabla_z \{g(z)\} \to 0$

When $z \to -\infty$, $g(z) \to -1 \Rightarrow \nabla_z \{g(z)\} \to 0$

So, for higher magnitude of input raw scores, the gradient is 0, hence the gradient vanishes.

Gradient of ReLU Activation Function: The gradient of the ReLU activation function is given as:

$$g(z) = \max(0, z)$$

$$\Rightarrow \nabla_z \{g(z)\} = \begin{cases} 0 & if\ z < 0 \\ 1 & if\ z > 1 \end{cases}$$

The gradient vanishes when the input value is less than 0. However, the gradient is still undefined for $z = 0$

Gradient of Leaky ReLU Activation Function: The gradient for Leaky ReLU Activation function is given as:

$$g(z) = \max(0.01z,\ z)$$

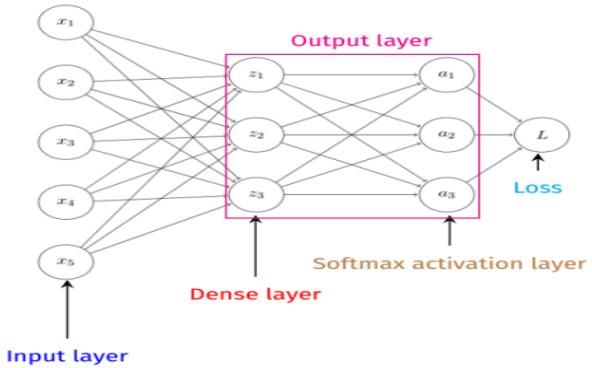
$$\Rightarrow \nabla_z \{g(z)\} = \begin{cases} 0.01 & if\ z < 0 \\ 1 & if\ z > 1 \end{cases}$$

The gradient doesn't vanish for any value of input (either positive or negative). The learning never stops. However, the gradient is still undefined at $z = 0$

Chapter - 9: Forward and Backward propagation using matrix based approach

Forward Propagation for Zero Hidden Layer Neural Network

The neural network architecture for a SoftMax classifier of a sample X with 5 features x_1, x_2, x_3, x_4, x_5 and the correct output label y from 3 possible output labels (ignoring the bias feature) is given as:



Input Layer: The input is a sample vector X having five features.

Dense Layer: It owns a weights matrix W of shape (3×5) and calculates the raw score vector Z of shape (3×1) as $Z = WX$.

SoftMax Activation Layer: In this step, the raw scores are SoftMax activated and stored in the vector a opf shape (3×1)

Loss: The loss L is calculated as $L = -log([a]_y)$

Categorical Cross Entropy (CCE) Loss for Classification problem

The predicted probability vector that the sample belongs to each one of the output categories is given as a new name $\hat{y} = a$, where a is the vector of predicted probabilities. We do one hot encoding for the output label, this is denoted as $\bar{y}$. For example, if the sample belongs to second class, then the one hot encoded output vector is defined as:

$$\bar{y} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

This results in the following representation for the softmax loss for the sample which is also referred to as the Categorical Cross Entropy (CCE) loss. Mathematically this is defined as:

$$\text{Loss}\,(L) = -\log\left([a]_y\right)$$

$$\Rightarrow \text{Loss}\,(L) = L\left(\bar{y}, \hat{y}\right) = \sum_{k=1}^{3} -\bar{y}_k \log\left(\hat{y}_k\right)$$

Example: Suppose a vector $\hat{y}$ and the actual one hot encoded output labels are given as:

$$\hat{y} = \begin{bmatrix} 0.85 \\ 0.05 \\ 0.15 \end{bmatrix};\ \bar{y} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

The Loss (L) is defined as:

$$L = -\left[0.\log\left(0.85\right) + 1.\log\left(0.05\right) + 0.\log\left(0.15\right)\right]$$

$$\Rightarrow L = -\log\left(0.05\right)$$

The more $\hat{y}$ and $\bar{y}$ are close to each other, the resulting loss will decrease more and vice-versa.

Gradient Calculation using Backward Propagation for Zero hidden layer Neural Network

The gradient of the loss (for a sample) with respect to the weights can be derived by using the following computation graph and chain rule:

$$\text{Computation Graph: } L \rightarrow \hat{y}/a \rightarrow Z \rightarrow W$$

By applying chain rule, we can derive the gradient of the loss function L w.r.t the weights matrix W as:

$$\nabla_W\left(L\right) = \nabla_W\left(Z\right) \times \nabla_Z\left(\hat{y}\right) \times \nabla_{\hat{y}}\left(L\right)$$

$$\Rightarrow \nabla_W\left(L\right) = \nabla_W\left(Z\right) \times \nabla_Z\left(a\right) \times \nabla_{\hat{y}}\left(L\right)$$

$$\Rightarrow \nabla_W\left(L\right) = \nabla_W\left(WX\right) \times \nabla_Z\left(SoftMax\left(Z\right)\right) \times \nabla_{\hat{y}}\left(\sum_k -y_k \log\left(\hat{y}_k\right)\right)$$

$$\downarrow \qquad\qquad \downarrow \qquad\qquad \downarrow$$

$$1 \qquad\qquad 2 \qquad\qquad 3$$

Shape Determination of the Gradient: We know, shape of a gradient can be calculated as input shape × output shape. In a gradient notation, the input is written in the lower part of the nabla symbol. The output is written on the side of the nabla symbol. For example, in the term $\nabla_W(WX)$, W is the input and WX is the output.

- **Shape of first gradient term:** For this above Neural Network architecture, the shape of the weights matrix W is (3×5). The sample vector X is of order (5×1). So, the raw score vector WX is of order (3×1). As, the input shape is (3×5) and output shape is (3×1), so the gradient $\nabla_W(WX)$ will be of shape $(3 \times 5 \times 3)$ - which is a tensor - a (5×3) matrix repeating 3 times.
- **Shape of second gradient term:** Since, the raw score vector Z is of shape (3×1), so the SoftMax activated vector a is also of shape (3×1). Since, the input and output shape for the second gradient term are same as (3×1), so the shape of the second gradient term will be (3×3). As differentiating a (3×1) vector with another (3×1) vector will result in a (3×3) matrix.
- **Shape of third gradient term:** For the last gradient term, the shape of the vector $\hat{y}$ is (3×1). The loss function value is a scalar quantity. So, the shape of the last gradient term is (3×1)

So, combining all these three gradient terms, we conclude that the shape of the gradient of the loss function L w.r.t the weights matrix W is (3×5). So, $\left|\nabla_W(L)\right| = (3 \times 5)$

Matrix Based Gradient Visualization: Before going in this process, remember that the weights matrix W is of shape (3×5). There are 3 possible output categories. So, the value of k in the CCE loss will be 3.

- **Visualization of third gradient term:**

$$\text{Loss } (L) = \sum_{k=1}^{3} -\bar{y}_k \log\left(\hat{y}_k\right) = -\left[y_1 \log\left(\hat{y}_1\right) + y_2 \log\left(\hat{y}_2\right) + y_3 \log\left(\hat{y}_3\right)\right]$$

So, the gradient of the loss function w.r.t the output class label $\hat{y}$ is defined as:

$$\nabla_{\hat{y}}(L) = \begin{bmatrix} \nabla_{\hat{y}_1}(L) \\ \nabla_{\hat{y}_2}(L) \\ \nabla_{\hat{y}_3}(L) \end{bmatrix} = \begin{bmatrix} -\dfrac{y_1}{\hat{y}_1} \\ -\dfrac{y_2}{\hat{y}_2} \\ -\dfrac{y_3}{\hat{y}_3} \end{bmatrix}_{3\times 1}$$

While doing the derivative of the Loss function L w.r.t. The output class label y_1, only the first term of L is treated as variable, rest terms are treated as constant.

- **Visualization of the second gradient term:**

$$a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} SoftMax(z_1) \\ SoftMax(z_2) \\ SoftMax(z_3) \end{bmatrix} = \begin{bmatrix} \dfrac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}} \\ \dfrac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}} \\ \dfrac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}} \end{bmatrix}$$

$$\nabla_Z\left(SoftMax(Z)\right) = \nabla_Z(a) = \nabla_Z\left(\begin{bmatrix} \dfrac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}} \\ \dfrac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}} \\ \dfrac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}} \end{bmatrix}\right)$$

$$\nabla_Z(a) = \begin{bmatrix} \nabla_Z\left(\dfrac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_Z\left(\dfrac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_Z\left(\dfrac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) \end{bmatrix}$$

$$\nabla_Z(a) = \begin{bmatrix} \nabla_{z_1}\left(\dfrac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_1}\left(\dfrac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_1}\left(\dfrac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) \\ \nabla_{z_2}\left(\dfrac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_2}\left(\dfrac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_2}\left(\dfrac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) \\ \nabla_{z_3}\left(\dfrac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_3}\left(\dfrac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_3}\left(\dfrac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) \end{bmatrix}$$

Now suppose, $a_1 = \dfrac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}$, $a_2 = \dfrac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}$ and $a_3 = \dfrac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}$

So, $\nabla_Z(a)$ becomes

$$\nabla_Z(a) = \begin{bmatrix} a_1(1-a_1) & -a_2 a_1 & -a_3 a_1 \\ -a_1 a_2 & a_2(1-a_2) & -a_3 a_2 \\ -a_1 a_3 & -a_2 a_3 & a_3(1-a_3) \end{bmatrix}$$

- **Visualization of first gradient term:** The visualization for the term $\nabla_W(WX)$ is given as:

$$\nabla_W(WX) = \nabla_W\left(\begin{bmatrix} w_1^T \\ w_2^T \\ w_3^T \end{bmatrix} X\right) = \nabla_W\left(\begin{bmatrix} w_1^T X \\ w_2^T X \\ w_3^T X \end{bmatrix}\right)$$

As we know, the shape of this gradient is $(3 \times 5 \times 3)$ or a (5×3) matrix repeating 3 times, the gradient can be visualized as:

$$\nabla_{w_1}\left(w_1^T X\right) \quad \nabla_{w_1}\left(w_2^T X\right) \quad \nabla_{w_1}\left(w_3^T X\right) \quad \rightarrow (5 \times 3)$$

$$\nabla_{w_2}\left(w_1^T X\right) \quad \nabla_{w_2}\left(w_2^T X\right) \quad \nabla_{w_2}\left(w_3^T X\right) \quad \rightarrow (5 \times 3)$$

$$\nabla_{w_3}\left(w_1^T X\right) \quad \nabla_{w_3}\left(w_2^T X\right) \quad \nabla_{w_3}\left(w_3^T X\right) \quad \rightarrow (5 \times 3)$$

Each term is a 5 vector. We know that

$$\nabla_{w_i}\left(w_j^T X\right) = \begin{cases} X & if\, i = j \\ 0 & if\, i \neq j \end{cases}$$

Then, $\nabla_W(WX)$ can be re-written as:

$$\nabla_W(WX) = \begin{matrix} X & 0 & 0 \\ 0 & X & 0 \\ 0 & 0 & X \end{matrix}$$

So, putting all the gradient values in the chain rule expression, we can write that

Computation Graph: $L \rightarrow \hat{y}/a \rightarrow Z \rightarrow W$

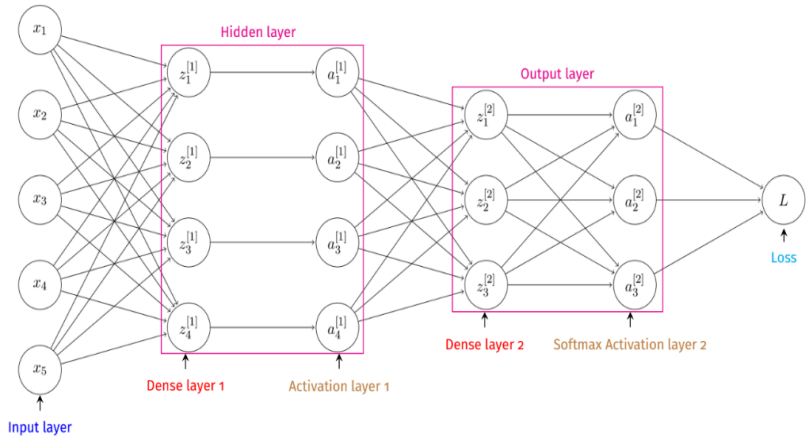
Chain Rule: $\nabla_W(L) = \nabla_W(WX) \times \nabla_Z\left(SoftMax(Z)\right) \times \nabla_{\hat{y}}\left(\sum_k -y_k \log\left(\hat{y_k}\right)\right)$

$$\Rightarrow \nabla_W(L) = \begin{matrix} X & 0 & 0 \\ 0 & X & 0 \\ 0 & 0 & X \end{matrix} \times \begin{bmatrix} a_1(1-a_1) & -a_2 a_1 & -a_3 a_1 \\ -a_1 a_2 & a_2(1-a_2) & -a_3 a_2 \\ -a_1 a_3 & -a_2 a_3 & a_3(1-a_3) \end{bmatrix} \times \begin{bmatrix} -\dfrac{y_1}{\hat{y}_1} \\ -\dfrac{y_2}{\hat{y}_2} \\ -\dfrac{y_3}{\hat{y}_3} \end{bmatrix}$$

When we multiply a 3D tensor X with a vector V, then we result we get is VX^T

Forward Propagation for Single Hidden Layer Neural Network

A neural network architecture for Single Hidden Layer Neural Network having a sample vector X (having five features) and output class labels y (possible 3 categories) can be given as:



Input Layer: Given by the sample vector X, also denoted as 0^{th} activation layer $a^{[0]}$. So, $X = a^{[0]}$

Dense Layer 1: Owns a weights matrix $W^{[1]}$, of shape (4×5). This weights matrix is converted into raw score vector as $Z^{[1]} = W^{[1]}X$ or $Z^{[1]} = W^{[1]}a^{[0]}$. So, $Z^{[1]}$ is of shape (4×1)

Activation Layer 1: The raw scores vector $Z^{[1]}$ is activated by using an activation function g (other than SoftMax activation function), which results in a activated raw scores vector $a^{[1]}$ of shape (4×1), given as $a^{[1]} = g(Z^{[1]})$

Hidden Layer 1: Both the Dense Layer 1 and Activation Layer 1 together creates a hidden layer.

Dense Layer 2: It owns a weights matrix $W^{[2]}$ of shape (3×4). This weights matrix is converted into raw score vector as $Z^{[2]} = W^{[2]}a^{[1]}$. So, $Z^{[2]}$ is of shape (3×1)

SoftMax Activation Layer/Activation Laye 2: The raw scores vector $Z^{[2]}$ is activated by using SoftMax function which results in a activated raw scores vector $a^{[2]}$ of shape (3×1), given as $a^{[2]} = SoftMax(Z^{[2]})$

Output Layer: The Dense Layer 2 and SoftMax Activation Layer or Activation Layer 2 together creates the output layer.

Loss: The loss is calculated as $L = -\ log[(a^{[2]})_y]$; where y is the correct class label.

Gradient Calculation using Backward Propagation for Single hidden layer Neural Network

The computation graph is defined as:

$$L \rightarrow \hat{y} = a^{[2]} \rightarrow Z^{[2]} \rightarrow a^{[1]} \rightarrow Z^{[1]} \rightarrow a^{[0]}$$
$$\downarrow \qquad\qquad \downarrow$$
$$W^{[2]} \qquad\qquad W^{[1]}$$

By applying the chain rule, we have:

$$\nabla_{W^{[2]}}(L) = \nabla_{W^{[2]}}\left(Z^{[2]}\right) \times \underbrace{\nabla_{Z^{[2]}}(\hat{y}) \times \nabla_{\hat{y}}(L)}$$

$$\nabla_{W^{[1]}}(L) = \nabla_{W^{[1]}}\left(Z^{[1]}\right) \times \nabla_{Z^{[1]}}\left(a^{[1]}\right) \times \nabla_{a^{[1]}}\left(Z^{[2]}\right) \times \underbrace{\nabla_{Z^{[2]}}(\hat{y}) \times \nabla_{\hat{y}}(L)}$$

The highlighted term appears in both the gradient calculations. In this example, the shape of weights matrix $W^{[1]}$ is (4×5) and the sample vector X having 5 features is of shape (5×1). So, the raw scores vector $Z^{[1]}$ is defined as $Z^{[1]} = W^{[1]}X$ and shape of $Z^{[1]}$ is (4×1). After applying the activation function say g (other than SoftMax activation function), the activated raw scores vector $a^{[1]}$ is of shape (4×1), same as $Z^{[1]}$.

Now, for the gradient term, $\nabla_{Z^{[1]}}(a^{[1]})$, the input $Z^{[1]}$ and output $a^{[1]}$ both are of shape (4×1). So, The resulting gradient will be a matrix of shape (4×4).

Mathematically this gradient is defined as:

$$\nabla_{Z^{[1]}}\left(a^{[1]}\right) = \left[\nabla_{Z^{[1]}}\left(a_1^{[1]}\right) \quad \nabla_{Z^{[1]}}\left(a_2^{[1]}\right) \quad \nabla_{Z^{[1]}}\left(a_3^{[1]}\right) \quad \nabla_{Z^{[1]}}\left(a_4^{[1]}\right)\right] \rightarrow \text{Each term is a 4 vector}$$

$$\Rightarrow \nabla_{Z^{[1]}}\left(a^{[1]}\right) = \left[\nabla_{Z^{[1]}}\left\{g\left(z_1^{[1]}\right)\right\} \quad \nabla_{Z^{[1]}}\left\{g\left(z_2^{[1]}\right)\right\} \quad \nabla_{Z^{[1]}}\left\{g\left(z_3^{[1]}\right)\right\} \quad \nabla_{Z^{[1]}}\left\{g\left(z_4^{[1]}\right)\right\}\right]$$

$$\Rightarrow \nabla_{Z^{[1]}}\left(a^{[1]}\right) = \begin{bmatrix} \nabla_{z_1^{[1]}}\left\{g\left(z_1^{[1]}\right)\right\} & \nabla_{z_1^{[1]}}\left\{g\left(z_2^{[1]}\right)\right\} & \nabla_{z_1^{[1]}}\left\{g\left(z_3^{[1]}\right)\right\} & \nabla_{z_1^{[1]}}\left\{g\left(z_4^{[1]}\right)\right\} \\ \nabla_{z_2^{[1]}}\left\{g\left(z_1^{[1]}\right)\right\} & \nabla_{z_2^{[1]}}\left\{g\left(z_2^{[1]}\right)\right\} & \nabla_{z_2^{[1]}}\left\{g\left(z_3^{[1]}\right)\right\} & \nabla_{z_2^{[1]}}\left\{g\left(z_4^{[1]}\right)\right\} \\ \nabla_{z_3^{[1]}}\left\{g\left(z_1^{[1]}\right)\right\} & \nabla_{z_3^{[1]}}\left\{g\left(z_2^{[1]}\right)\right\} & \nabla_{z_3^{[1]}}\left\{g\left(z_3^{[1]}\right)\right\} & \nabla_{z_3^{[1]}}\left\{g\left(z_4^{[1]}\right)\right\} \\ \nabla_{z_4^{[1]}}\left\{g\left(z_1^{[1]}\right)\right\} & \nabla_{z_4^{[1]}}\left\{g\left(z_2^{[1]}\right)\right\} & \nabla_{z_4^{[1]}}\left\{g\left(z_3^{[1]}\right)\right\} & \nabla_{z_4^{[1]}}\left\{g\left(z_4^{[1]}\right)\right\} \end{bmatrix}_{4\times4}$$

The diagonal entries will be non-zero. Other non-diagonal entries will be 0

$$\Rightarrow \nabla_{Z^{[1]}}\left(a^{[1]}\right) = \begin{bmatrix} \nabla_{z_1^{[1]}}\left\{g\left(z_1^{[1]}\right)\right\} & 0 & 0 & 0 \\ 0 & \nabla_{z_2^{[1]}}\left\{g\left(z_2^{[1]}\right)\right\} & 0 & 0 \\ 0 & 0 & \nabla_{z_3^{[1]}}\left\{g\left(z_3^{[1]}\right)\right\} & 0 \\ 0 & 0 & 0 & \nabla_{z_4^{[1]}}\left\{g\left(z_4^{[1]}\right)\right\} \end{bmatrix}$$

If we consider the activation function as the sigmoid function, then a large (+ve) or (-ve) value of raw score will result in a zero value of raw score and will result in a zero matrix for the above gradient. Consequently, the learning process stops and $\nabla_{W^{[1]}}(L)$ becomes a zero matrix. This phenomenon is called the vanishing gradient problem.

However, the second weights matrix $W^{[2]}$ is updated as soon as $\nabla_{W^{[2]}}(L)$ is available.

Importance of Dense Layer in Forward and Backward Propagation

In forward and backward propagation, Dense layer (specifically Dense Layer 2) does the following jobs:

Forward Propagation

$a^{[1]}$ (Input) $\rightarrow$ Dense Layer 2 $\rightarrow Z^{[2]} = W^{[2]}a^{[1]}$ (Output)

Backward Propagation

$\nabla_{W^{[2]}}(L)$ (grad w.r.t weight) $\leftarrow$ Dense Layer 2 $\leftarrow \nabla_{Z^{[2]}}(L)$ (gad w.r.t output)

$\nabla_{a^{[1]}}(L)$ (grad w.r.t input) $\leftarrow$

Parameters and Hyper-parameters of a model

Model Parameters: Model parameters are internal to the model that are learned from the data typically in an iterative manner that starts with random initial guess. Weights and bias values of a Neural Network model are examples of model parameters.

Model Hyper-parameters: Model hyper-parameters are external to the model and are not learned from data, but rather used to estimate the model parameters. Examples of model hyper-parameters include number of layers, nodes per layer, learning rate, regularization strength etc. They are typically tuned when constructing models specific to a data set.

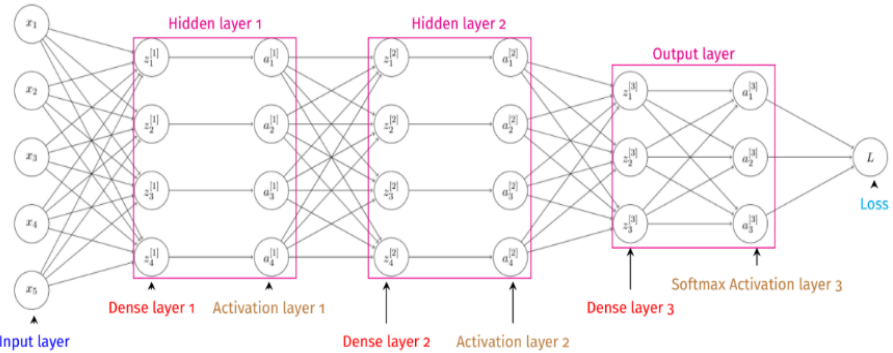
Chapter - 10: L-Layered NN: Architecture, Notation and Building Blocks

Introduction

Establishing mathematical relationships between different various functions is also possible by a single hidden layer NN. In this case, the number of neurons/nodes will be very high, which is not theoretically feasible. So, we use a much deeper architectural concept. So, in order to achieve this, we increase the number of hidden layers each with a manageable number of nodes.

Deep L-Layered Neural Network: Architecture

Consider a sample vector X having five features x_1, x_2, x_3, x_4, x_5. Also suppose that the correct output class labels as a vector y that can belong to 3 possible output categories. Then the architecture for a 2 hidden layer Neural Network is given as:



The activation of the output layer (SoftMax Activation) is fully connected only in the deep learning network.

Deep L-Layered Neural Network: Notation, Building Blocks

In a L-layered deep NN, there are total $(L + 1)$ layers - from input layer to output layer. The input layer has layer index 0. The hidden layers have layer index $1, 2, 3, \ldots, L - 1$. The output layer has layer index L. For a L-layered deep NN, L always denotes the output layer. For example, in the above NN architecture, 3-layered deep NN is shown. Hence the value of L is 3. So, there are total $(3 + 1) = 4$ layers. On the other hand, for a L-layered deep NN, there are $(L - 1)$ hidden layers. In the above 3-layered deep NN architecture, there are $(3 - 1) = 2$ hidden layers.

The term $n^{[l]}$ denotes the number of neurons/nodes in the layer index l. For the above 3-layered deep NN architecture, the layer indexes are 0 (input layer),1 (hidden layer 1),2 (hidden layer 2) and 3 (output layer). So, from the notation meaning, we can say that $n^{[0]} = 5$, $n^{[1]} = 4$, $n^{[2]} = 4$ & $n^{[3]} = 3$

The raw scores vector for hidden layer l is denoted by $Z^{[l]}$ and it is of shape $n^{[l]} \times 1$. For example, for the Dense Layer 1 (associated with hidden layer 1), the raw scores vector is denoted as $Z^{[1]}$ and it has 4 components as shown in the architecture, given as $z_1^{[1]}$, $z_2^{[1]}$, $z_3^{[1]}$ & $z_4^{[1]}$. So, the shape of $Z^{[1]}$ is (4×1), same as $n^{[1]} \times 1$

The activated raw scores vector for hidden layer l is denoted by $a^{[l]}$ and it is of shape $n^{[l]} \times 1$. For example, for the Activation Layer 1 (associated with hidden layer 1), the raw scores vector is denoted as $a^{[1]}$ and it has 4 components as shown in the architecture, given as $a_1^{[1]}$, $a_2^{[1]}$, $a_3^{[1]}$ & $a_4^{[1]}$. So, the shape of $a^{[1]}$ is (4×1), same as $n^{[1]} \times 1$

The weights matrix associated with dense layer l, is denoted as $W^{[l]}$ and it is of shape $n^{[l]} \times n^{[l-1]}$ for $l = 0, 1, 2, 3, \ldots, L$. For example, for the dense layer 1 in the 3-layered deep NN architecture, the weights matrix associated with dense layer 1 is denoted as $W^{[1]}$ and it is of shape (4×5), same as $(n^{[1]} \times n^{[0]})$

The activation function associated with layer index l is denoted as $g^{[l]}$. For the first hidden layer (Activation layer 1), it is denoted as $g^{[1]}$. It activates the raw scores vector of that layer $Z^{[1]}$ and creates activated raw scores vector $a^{[1]}$ as: $a^{[1]} = g^{[1]}(Z^{[1]})$ and $Z^{[1]} = W^{[1]}a^{[0]}$

For a batch of samples of size b, and for the layer l with $l = 1, 2, 3, \ldots L$, the raw scores matrix $Z^{[l]}$ and the activated raw scores matrix $A^{[l]}$ is denoted as:

$$\bar{Z}^{[l]} = \begin{bmatrix} Z^{[l](0)} & Z^{[l](1)} & \ldots & Z^{[l](b-1)} \end{bmatrix}$$

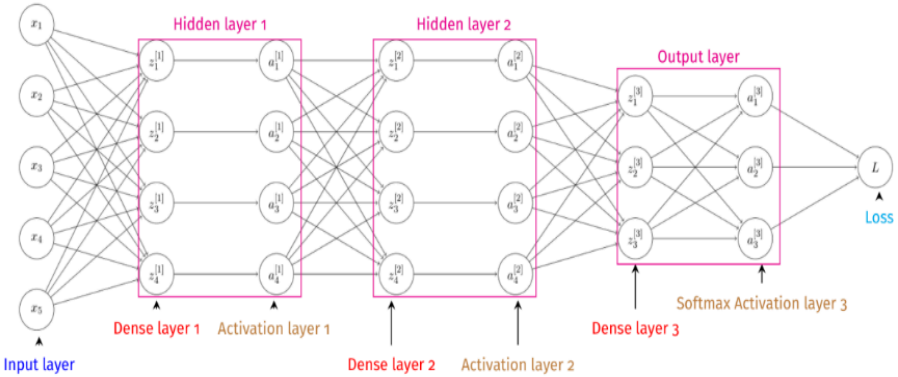
$$A^{[l]} = \begin{bmatrix} a^{[l](0)} & a^{[l](1)} & \ldots & a^{[l](b-1)} \end{bmatrix}$$

So, $Z^{[1](0)}$ is the first hidden layer's raw score vector for the 0^{th} sample. Each of the components of $\bar{Z}^{[l]}$ has size $n^{[l]}$ and there are b columns. So, shape of $\bar{Z}^{[l]}$ is $(n^{[l]} \times b)$. Similarly, the shape of the matrix $A^{[l]}$ will be same as $\bar{Z}^{[l]}$ which is $(n^{[l]} \times b)$. For example, if batch size b is 16 and we consider the first hidden layer, then $\bar{Z}^{[l]}$ and $A^{[l]}$ both have shape (4×16)

Chapter - 11: Forward & Backward propagation in Deep NN using Matrix

Introduction

Consider applying a 3-layered deep neural network to a sample X having five features and correct output class labels given as y from 3 possible output categories. Ignore the bias feature for clear understanding. The architecture is same as before, given as:



Forward Propagation

Input layer: Having layer index 0 is a sample denoted by vector X, having five features. Denoted as $a^{[0]}$

Hidden Layer 1: Having layer index 1, constituting two layers - Dense Layer 1 and Activation Layer 1

- **Dense Layer 1:** Owns a weights matrix $W^{[1]}$ of shape (4×5). It creates the raw scores vector $Z^{[1]}$ by the expression $Z^{[1]} = W^{[1]}a^{[0]}$. So, the raw scores vector $Z^{[1]}$ is of shape (4×1)
- **Activation Layer 1:** It elementwise activates the raw scores vector $Z^{[1]}$ by an activation function say $g^{[1]}$, other than SoftMax activation function and creates a activated raw scores vector $a^{[1]}$ as $a^{[1]} = g^{[1]}(Z^{[1]})$. So, the activated raw scores vector $a^{[1]}$ is of shape (4×1)

Hidden Layer 2: Having layer index 2, constituting two layers - Dense Layer 2 and Activation Layer 2

- **Dense Layer 2:** Owns a weights matrix $W^{[2]}$ of shape (4×4). It creates the raw scores vector $Z^{[2]}$ by the expression $Z^{[2]} = W^{[2]}a^{[1]}$. So, the raw scores vector $Z^{[2]}$ is of shape (4×1)
- **Activation Layer 2:** It elementwise activates the raw scores vector $Z^{[2]}$ by an activation function say $g^{[2]}$, other than SoftMax activation function and creates a activated raw scores vector $a^{[2]}$ as $a^{[2]} = g^{[2]}(Z^{[2]})$. So, the activated raw scores vector $a^{[2]}$ is of shape (4×1)

Output Layer: Having layer index 3, constituting two sub layers - Dense Layer 3 and SoftMax Activation Layer

- **Dense Layer 1:** Owns a weights matrix $W^{[3]}$ of shape (3×4). It creates the raw scores vector $Z^{[3]}$ by the expression $Z^{[3]} = W^{[3]}a^{[2]}$. So, the raw scores vector $Z^{[3]}$ is of shape (3×1)
- **SoftMax Activation Layer 1:** It activates the raw scores vector $Z^{[3]}$ by an activation function say $g^{[3]}$which is SoftMax activation function and creates a activated raw scores vector $a^{[3]}$ as $a^{[3]} = g^{[3]}(Z^{[3]})$. So, the activated raw scores vector $a^{[3]}$ is of shape (3×1). This can be written as $a^{[3]} = SoftMax(Z^{[3]})$. Often the vector $a^{[3]}$ is denoted as $\hat{y}$

Loss Calculation: The CCE loss is denoted by L and it is given as $L = -\sum_{k=1}^{3} y_k log(a^{[3]}{}_k)$; where y_k is one hot encoded values of correct class labels and $a^{[3]}$ is a vector of predicted probabilities and $a^{[3]}{}_k$ is the predicted probability that the sample belongs to category k

Bias Accounting in Deep L-Layered Neural Network

In the 3-layered deep neural network architecture considered above, the number of nodes in each layer with layer index $l = 0, 1, 2, 3$ is given as $n^{[0]} = 5,\ n^{[1]} = 4,\ n^{[2]} = 4\ \&\ n^{[3]} = 3$. (These are excluded of the bias term)
The bias feature 1 is appended to the activated scores prior to feeding as input to the next dense layer. So, in short the bias features are appended in the activation layers. Thus mathematically we get,

Hidden Layer 1

$$a_B^{[0]} = \begin{bmatrix} a^{[0]} \\ 1 \end{bmatrix};\ \left|a_B^{[0]}\right| = (6 \times 1);\ \text{Where } a^{[0]} \text{ is of shape } (5 \times 1)$$

$$\text{So, } Z^{[1]} = W_B^{[1]}a_B^{[0]};\ \left|W_B^{[1]}\right| = n^{[1]} \times \left(n^{[0]} + 1\right) = (4 \times 6);\ W_B^{[1]} = W^{[1]}\ \text{(For simplicity)}$$

$$\Rightarrow \left|Z^{[1]}\right| = (4 \times 1)$$

Hidden Layer 2

$$a_B^{[1]} = \begin{bmatrix} a^{[1]} \\ 1 \end{bmatrix};\ \left|a_B^{[1]}\right| = (5 \times 1);\ \text{Where } a^{[1]} \text{ is of shape } (4 \times 1)$$

$$\text{So, } Z^{[2]} = W_B^{[2]}a_B^{[1]};\ \left|W_B^{[2]}\right| = n^{[2]} \times \left(n^{[1]} + 1\right) = (4 \times 5);\ W_B^{[2]} = W^{[2]}\ \text{(For simplicity)}$$

$$\Rightarrow \left|Z^{[2]}\right| = (4 \times 1)$$

Output Layer

$$a_B^{[2]} = \begin{bmatrix} a^{[2]} \\ 1 \end{bmatrix};\ \left|a_B^{[2]}\right| = (5 \times 1);\ \text{Where } a^{[2]} \text{ is of shape } (4 \times 1)$$

$$\text{So, } Z^{[3]} = W^{[3]}a_B^{[2]};\ \left|W_B^{[3]}\right| = n^{[3]} \times \left(n^{[2]} + 1\right) = (3 \times 5);\ W_B^{[3]} = W^{[3]}\ \text{(For simplicity)}$$

$$\Rightarrow \left|Z^{[3]}\right| = (3 \times 1)$$

Minibatch forward propagation

Suppose, we have a minibatch comprising b samples represented as the $(5 \times b)$ matrix (5 features), with one hot encoded true class labels as $(3 \times b)$ matrix (3 possible output categories) represented as:

$$\bar{X} = \begin{bmatrix} X^{(1)} & X^{(2)} & \dots & X^{(b)} \end{bmatrix}_{5 \times b};\ \bar{Y} = \begin{bmatrix} y^{(1)} & y^{(2)} & \dots & y^{(b)} \end{bmatrix}_{3 \times b}$$

Hidden layer 1: we calculate the raw scores vector and activated raw scores vector considering b samples and bias feature.

- **Dense Layer 1:** Here, each element of $\bar{Z}^{[1]}$ is a 4 vector and there are b samples. So, shape of $\bar{Z}^{[1]}$ is $(4 \times b)$. The term $A_B^{[0]}$ is the activated raw scores for all sample in the 0^{th} layer and bias feature is added.

Each element of $A_B^{[0]}$ is a 6 vector and there are b samples. So, $A_B^{[0]}$ is of shape $(6 \times b)$. As stated earlier $W^{[1]}$ is of shape (4×6)

$$\underbrace{\begin{bmatrix} Z^{[1](1)} & Z^{[1](2)} & \dots & Z^{[1](b)} \end{bmatrix}}_{\bar{Z}^{[1]}} = \begin{bmatrix} W^{[1]}a_B^{[0](1)} & W^{[1]}a_B^{[0](2)} & \dots & W^{[1]}a_B^{[0](b)} \end{bmatrix}$$

$$= W^{[1]} \underbrace{\begin{bmatrix} a_B^{[0](1)} & a_B^{[0](2)} & \dots & a_B^{[0](b)} \end{bmatrix}}_{A_B^{[0]}}$$

So, for Dense Layer 1 of Hidden Layer 1, the raw scores vector $\bar{Z}^{[1]}$ can be calculated as:

$$\bar{Z}^{[1]} = W^{[1]}A_B^{[0]}$$

- **Activation Layer 1:** As mentioned earlier, $\bar{Z}^{[1]}$ is of shape $(4 \times b)$ and the activation function say $g^{[1]}$ is a function applied column by column and further element by element. So, at the end the shape of $A^{[1]}$ becomes same as $\bar{Z}^{[1]}$ which is $(4 \times b)$

$$\underbrace{\begin{bmatrix} a^{[1](1)} & a^{[1](2)} & \dots & a^{[1](b)} \end{bmatrix}}_{A^{[1]}} = \begin{bmatrix} g^{[1]}\left(Z^{[1](1)}\right) & g^{[1]}\left(Z^{[1](2)}\right) & \dots & g^{[1]}\left(Z^{[1](b)}\right) \end{bmatrix}$$

$$= g^{[1]} \underbrace{\begin{bmatrix} Z^{[1](1)} & Z^{[1](2)} & \dots & Z^{[1](b)} \end{bmatrix}}_{\bar{Z}^{[1]}}$$

So, for Activation Layer 1 of Hidden Layer 1, the activated raw scores matrix is:

$$A^{[1]} = g^{[1]}\left(\bar{Z}^{[1]}\right)$$

Hidden Layer 2: For the hidden layer 2, in similar manner of Hidden layer 1 we calculate the raw scores matrix and activated raw scores matrix $\bar{Z}^{[2]}$ and $A^{[2]}$ as

$$\bar{Z}^{[2]} = W^{[2]}A_B^{[1]}\ ;\ A^{[2]} = g^{[2]}\left(\bar{Z}^{[2]}\right)$$

Where, $A_B^{[1]}$ is a $(5 \times b)$ matrix, $W^{[2]}$ is a (4×5) matrix. So, $\bar{Z}^{[2]}$ is a $(4 \times b)$ matrix, $g^{[2]}$ is an activation function, other than SoftMax activation function and $A^{[2]}$ is also a $(4 \times b)$ matrix.

Output Layer: For the output layer, the raw scores matrix $\bar{Z}^{[3]}$ and the SoftMax activated raw scores matrix $A^{[3]}$ is given as:

$$\bar{Z}^{[3]} = W^{[3]}A_B^{[2]}\ ;\ A^{[3]} = SoftMax\left(\bar{Z}^{[3]}\right)$$

Where, $A_B^{[2]}$ is a matrix of shape $(5 \times b)$, $W^{[3]}$ is a (3×5) matrix. So, $\bar{Z}^{[3]}$ is a $(3 \times b)$ matrix which is SoftMax activated and hence $A^{[3]}$ is also a $(3 \times b)$ matrix.

Each column of $A^{[3]}$ tells the predicted probability of a particular sample. Now, we set $\hat{Y} = A^{[3]}$, the predicted probabilities matrix for the minibatch. The CCE loss, denoted by L_i for the i^{th} sample is defined as:

$$L_i = \sum_{k=1}^{3} -y_k^i \log\left(\hat{y}_k^{(i)}\right)$$

This leads to the average Categorical Cross Entropy loss calculation for minibatch sample as:

$$L = \frac{1}{b}[L_1 + L_2 + \dots + L_b]$$

$$\Rightarrow L = \frac{1}{b}\left[\sum_{k=1}^{3}\left\{-y_k^{(1)}\log\left(\hat{y}_k^{(1)}\right)\right\} + \sum_{k=1}^{3}\left\{-y_k^{(2)}\log\left(\hat{y}_k^{(2)}\right)\right\} + \dots + \sum_{k=1}^{3}\left\{-y_k^{(b)}\log\left(\hat{y}_k^{(b)}\right)\right\}\right]$$

The loss calculation involves forward propagation and loss matrix manipulation. It is a time taking process, so batch size must be chosen appropriately.

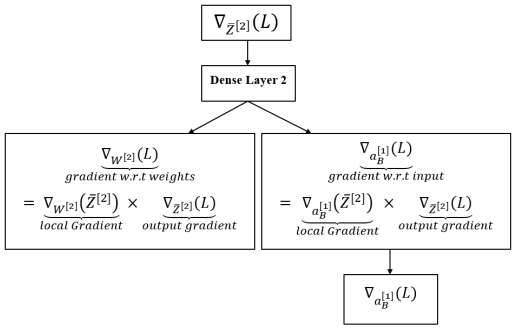
Gradient calculation using Backward Propagation

The computation graph for the gradient calculation using backward propagation for a 3-layered deep neural network is given as:

$$\begin{array}{l} L \rightarrow \hat{y}\left(= a^{[3]}\right) \rightarrow Z^{[3]} \rightarrow W^{[3]} \\ \qquad\qquad \downarrow \\ \qquad\qquad a_B^{[2]} \rightarrow a^{[2]} \rightarrow Z^{[2]} \rightarrow W^{[2]} \\ \qquad\qquad\qquad\qquad \downarrow \\ \qquad\qquad\qquad\qquad a_B^{[1]} \rightarrow a^{[1]} \rightarrow Z^{[1]} \rightarrow W^{[1]} \\ \qquad\qquad\qquad\qquad\qquad\qquad \downarrow \\ \qquad\qquad\qquad\qquad\qquad\qquad a_B^{[0]} \end{array}$$

During the backward propagation, a **dense layer** looks at the gradient flowing backward from its input side and does two things -

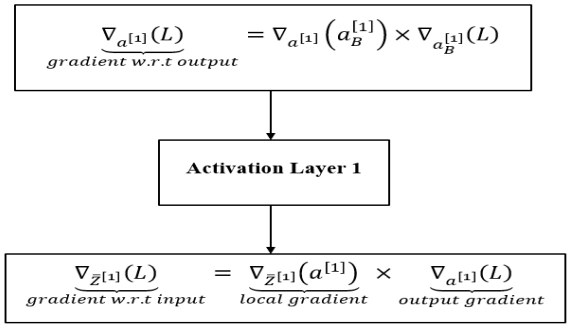
1. It calculates the local gradient w.r.t its weights and multiplies that with the output gradient so that the weights can be updated.
2. It calculates the local gradient w.r.t its input and multiplies that with the output gradient so that the resulting gradient flowing backward from its input side is returned.



This result then becomes the gradient flowing backward from the outside of the previous activation layer.
During the backward propagation, an **activation layer** does the following things -

1. It clips the gradient flowing backward sent by a subsequent dense layer such that the last row of the gradient is removed.
2. The resulting gradient is then sent flowing backward on the output side for the activation layer.
3. After that it calculates its local gradient which is multiplied by the clipped output gradient so that the resulting gradient flowing backward from its input side is returned.
4. This then becomes the gradient flowing backward from the output side of the previous dense layer.

The mechanism of the activation layer, is depicted by using a flowchart as:



Minibatch Backward Propagation

Suppose, we have a minibatch comprising b samples; the gradient of the average loss w.r.t a particular set of weights is simply the average of the gradient of each samples loss w.r.t those weights; for example, consider the gradient of the average loss w.r.t the weight matrix $W^{[2]}$, focusing on the gradient flowing through the hidden layer 2 is given as:

$$L = \frac{1}{b}[L_1 + L_2 + \ldots + L_b]$$

$$\text{So, } \nabla_{W^{[2]}}(L) = \frac{1}{b}\{\nabla_{W^{[2]}}(L_1) + \nabla_{W^{[2]}}(L_2) + \ldots + \nabla_{W^{[2]}}(L_b)\}$$

Now, applying the chain rule, we can easily find the gradients in the above expression as:

$$\nabla_{W^{[2]}}(L_1) = \nabla_{W^{[2]}}\left(\bar{Z}^{[2](1)}\right) \times \nabla_{\bar{Z}^{[2](1)}}\left(a^{[2](1)}\right) \times \nabla_{a^{[2](1)}}(L_1)$$

$$\nabla_{W^{[2]}}(L_2) = \nabla_{W^{[2]}}\left(\bar{Z}^{[2](2)}\right) \times \nabla_{\bar{Z}^{[2](2)}}\left(a^{[2](2)}\right) \times \nabla_{a^{[2](2)}}(L_2)$$

$$\vdots$$

$$\nabla_{W^{[2]}}(L_b) = \nabla_{W^{[2]}}\left(\bar{Z}^{[2](b)}\right) \times \nabla_{\bar{Z}^{[2](b)}}\left(a^{[2](b)}\right) \times \nabla_{a^{[2](b)}}(L_b)$$

Chapter - 12: Overfitting in Deep NN: Loss Based, Dropout Regularization Approaches

Introduction

Deep learning models should do well not only on the train data (low training loss), but also on unseen test data (low test loss). Deep learning models by construction are highly non-linear due to the presence of multiple neurons and layers with their associated weights and non-linear activation functions. More nodes in a layer increases the complexity of the model. By complexity, we mean the non-linear relationships. More layers increases the capability of the model. Capability means the ability to combine different variables.
Complex and high capacity models like these, when overtrained on the train data, start learning the noise in the data and tend to learn from specific features. One way to address this overfitting or overtraining issue is to have more training data, say by augmenting the training dataset. Weights getting disproportionately bigger is the root cause of overfitting; more nodes or layers, more likely this is to happen.

Loss Based Regularization Approach

Recall loss based regularization which is achieved by adding to the average training data loss constraints on the weights (not to the bias values). For a deep neural network, regularization is applied to the weights owned by each dense layer. Consider a 3 layered deep neural network with L2 regularization.

Dense Layer 1: The loss loss based regularized loss function is given as:

$$Loss = L\left(W^{[1]}\right) + \lambda\left(\left\|w^{[1]}_{1,\,1:5}\right\|^2 + \left\|w^{[1]}_{2,\,1:5}\right\|^2 + \left\|w^{[1]}_{3,\,1:5}\right\|^2 + \left\|w^{[1]}_{4,\,1:5}\right\|^2\right)$$

Here $L\left(W^{[1]}\right)$ is called training data loss and $\left|W^{[1]}\right| = (4 \times 6)$

The term $(1:5)$ in subscript means consider first five columns, exclude bias term

We not only want to minimize the training loss related to dense layer 1, we want to make the weights uniformly small through the regularization strength λ, which is a hyper-parameter.

Dense Layer 2: The loss loss based regularized loss function is given as:

$$Loss = L\left(W^{[2]}\right) + \lambda\left(\left\|w^{[2]}_{1,\,1:4}\right\|^2 + \left\|w^{[2]}_{2,\,1:4}\right\|^2 + \left\|w^{[2]}_{3,\,1:4}\right\|^2 + \left\|w^{[2]}_{4,\,1:4}\right\|^2\right)$$

Here $L\left(W^{[2]}\right)$ is called training data loss and $\left|W^{[2]}\right| = (4 \times 5)$

The term $(1:4)$ in subscript means consider first four columns, exclude bias term

There are some certain differences between L1 and L2 regularization. In L1 regularization, most of the weights are converted to zero. So there are very small numbers of non zero weights. Specifically used when we have previous knowledge which factors impact the most. On the other hand in L2 regularization, all the weights are uniformly shrink to zero. Specifically used when there is no previous knowledge about which factors are more important.

Densen Layer 3: The loss loss based regularized loss function is given as:

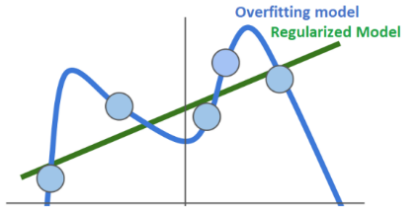
$$Loss = L\left(W^{[3]}\right) + \lambda\left(\left\|w^{[3]}_{1,\,1:4}\right\|^2 + \left\|w^{[3]}_{2,\,1:4}\right\|^2 + \left\|w^{[3]}_{3,\,1:4}\right\|^2\right)$$

Here $L\left(W^{[3]}\right)$ is called training data loss and $\left|W^{[3]}\right| = (3 \times 5)$

The term $(1:4)$ in subscript means consider first four columns, exclude bias term

Why does the Loss Based Regularization process work?

Consider a model having a overfitting trend line and a regularized trend line depicted as:



The straight line has high bias (difference between actual and predicted values) and low variance, that generalizes well on unseen test data. L2 Regularization shrinks the weights uniformly close to zero and hence keeps the model simple.

Consider the hyperbolic tangent activation function given as:

$$g(Z) = \frac{e^{Z} - e^{-Z}}{e^{Z} + e^{-Z}};\ \text{As } Z \approx 0,\ g(Z) \text{ behaves as linear activation function}$$

We know that $Z^{[l]} = W^{[l]}a^{[l-1]}$. So small weights keep the raw scores small thus keeping the activations in the linear zone, thus it keeps the model simple.

Idea of Dropout Regularization

Dropout is another approach, used is regularization. In this process, during training the model using a particular batch of samples in each hidden layer, nodes are randomly eliminated. So. dropout probability of a hidden layer 1 is 0.5 means with 50% probability, i am going to drop a node. In general, no nodes are dropped from the output layer.

The dropped out nodes do not contribute to the training process (forward & backward propagation). However, there is a change in the shape of weights matrices in corresponding layers.

	Before	After	
$W^{[1]}$:	4×6	2×6	Bias feature included, 50% nodes are dropped
$W^{[2]}$:	4×5	2×3	
$W^{[3]}$:	3×5	3×3	

Dropout Regularization: Practical Details

Inverted Dropout: A popular dropout regularization technique which follows the below process given as:

- Consider layer index l and activated scores matrix $A^{[l]}$ for a training batch of size b; $\left|A^{[l]}\right| = n^{[l]} \times b$
- We generate a random matrix $D^{[l]}$ of the same shape as $A^{[l]}$ such that its elements are uniform random numbers from 0 to 1.
- Using a probability p of retaining a node in layer l (which is one minus probability of dropout), reset the elements of $D^{[l]}$ as $D^{[l]} \leq p$, which results in a matrix full of zeros and ones.
- During forward propagation, replace $A^{[l]}$ by $A^{[l]} \odot D^{[l]}$ (Hadamard product), thus zeroing out $(1 - p)\%$ of elements in each column of $A^{[l]}$

- Replace $A^{[l]}$ by $A^{[l]}/p$ so that the next layers raw scores vector $Z^{[l+1]} = W^{[l+1]}A^{[l]}$ do not go down in expected value because of dropping nodes.

Important Note: Dropout is not applied at test time (only applied in training data). This process is sample specific for example, for the first sample we drop the first and third node, for the second sample we can drop the second and fourth sample.

Why does the Dropout Regularization process work?

Dropout ensures that a neuron in a particular layer learns its weights by not relying on a particular set of features as different nodes in the previous layer are dropped out randomly for each sample in a training batch. This has an effect similar to that of shrinking weights towards zero by L2 regularization. In a deep neural network with multiple layers, layers with higher number of nodes are typically associated with a higher dropout probability that ones with smaller number of nodes.

Chapter - 13: Random Initialization, Batch Normalization

Why Initializing weights are important?

The training loss in a deep neural network is a function of the weights associated with the dense layer. Initializing those weights appropriately can have a significant impact on how fast the learning happens. Initializing the same weights for all neurons in a particular dense layer will result in symmetric learning and will prevent the neurons from learning different things.

Initializing the weights too small may lead to slow learning. Initializing the weights too big may lead to divergence.

For a single sample, the local gradient of the dense layer l, where output is $Z^{[l]}$ and input is $a^{[l-1]}$ is given as:

$$\nabla_{a^{[l-1]}}\left(Z^{[l]}\right) = \nabla_{a^{[l-1]}}\left(W^{[l]}a^{[l-1]}\right);\ \left|a^{[l-1]}\right| = n^{[l-1]};\ \left|W^{[l]}\right| = n^{[l]} \times n^{[l-1]}$$

$$\text{So, } \left|W^{[l]}a^{[l-1]}\right| = n^{[l]} \text{ and } \nabla_{a^{[l-1]}}\left(Z^{[l]}\right) = W^{[l]^T}$$

For a single sample, the local gradient of the activation layer having index $(l - 1)$, having input $Z^{[l-1]}$ and output $a^{[l-1]}$ is given by the expression:

$$\nabla_{Z^{[l-1]}}\left(a^{[l-1]}\right)$$

Which for ReLU activation is a diagonal matrix whose entries are given as an indicator function I as:

$$I\left(Z^{[l-1]} > 0\right)$$

The above terms get progressively multiplied during backward propagation for calculating gradients of loss w.r.t weights. Weights that are activated with small values result in vanishing gradients, weights activated with large values results in exploding gradients. A good initialization of weights should result in -

1. Zero mean of activations
2. Constant variance of activation across layers.

Both of them address the vanishing and exploding gradient problem.

Weight Initializing Techniques

For a dense layer with index l, the weights matrix $W^{[l]}$ is of shape $(n^{[l]} \times n^{[l-1]})$. Different initialization techniques are used to initialize these weights randomly. They are given as -

Xavier Initialization for Tanh Activation:

$$\underbrace{W^{[l]}_{:,1:n^{[l-1]}-1} \sim N\left(\mu = 0,\, \sigma^2 = \frac{2}{n^{[l-1]+n^{[l]}}}\right)}_{\text{Weights}} ;\; \underbrace{W^{[l]}_{:,\,n^{[l-1]}} = 0}_{\text{Bias}}$$

It is a good practice to let the Bias value very small positive number

He Initialization for ReLU Activation:

$$\underbrace{W^{[l]}_{:,1:n^{[l-1]}-1} \sim N\left(\mu = 0,\, \sigma^2 = \frac{2}{n^{[l-1]}}\right)}_{\text{Weights}} ;\; \underbrace{W^{[l]}_{:,\,n^{[l-1]}} = 0}_{\text{Bias}}$$

It is a good practice to let the Bias value very small positive number

Normalizing Activations in a Deep NN

Standardization of an input feature is the process of subtracting the feature mean across all samples and dividing them by standard deviation of that feature. This process results in both positive and negative values (above/below average), independent of units used to measure that feature. It typically speeds up the learning of weights associated with the next dense layer. In deep learning, standardization is referred to as normalization. The same idea can be applied to speed up the learning of the weights associated with deeper dense layers in a deep neural network.

Batch Normalization: Practical details

Consider the hidden layer having index l of a deep learning neural network. Weights to be learned from a batch of samples of size b. The complete Mathematical calculation is defined as:

Raw Scores matrix is given as: $\bar{Z}^{[l]} = \begin{bmatrix} Z^{[l](1)} & Z^{[l](2)} & \dots & Z^{[l](b)} \end{bmatrix}$

Mean raw scores in batch is calculated as: $\mu = \frac{1}{b}\sum_{i=1}^{b} Z^{[l](i)}$

Standard deviation of raw scores is calculated as: $\sigma^2 = \frac{1}{b}\sum_{i=1}^{b}\left(Z^{[l](i)} - \mu\right)^2$

Standardized raw scores for each sample in batch is calculated as: $Z^{[l](i)}_{norm} = \frac{Z^{[l](i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$

(ϵ is a constant term, used to give the expression a numerical stability)

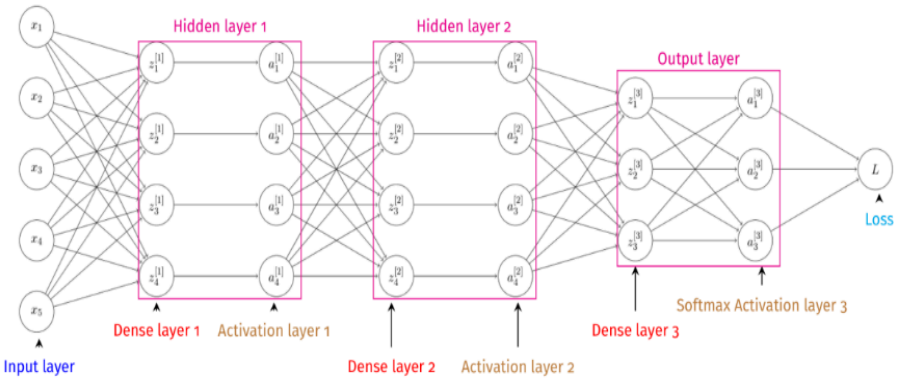
Batch norm updated aw scores for each sample in the batch is calculated as: $\tilde{Z}^{[l](i)} = \gamma Z^{[l](i)}_{norm} + \beta$

(β, γ are parameters like weight which extends the range of mean and variance beyond $(0, 1)$)

Chapter - 14: Foundations of Convolutional Neural Networks (CNNs)

Transition from Fully Connected NN to CNNs

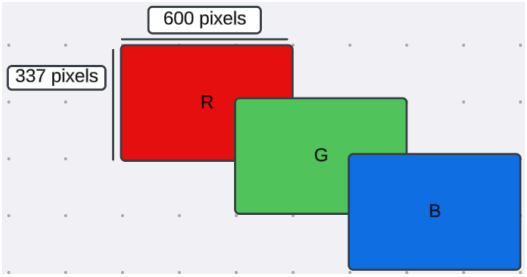
A 3-layered deep neural network architecture for a sample having 5 features is depicted as:



Note that a 3-layered deep neural network has 2 hidden layers. Each neuron in a hidden layer own a set of weights that it applies on all the features from the previous layer and calculates a single activation function score that will add in learning its weights.

When the number of features in the input layer becomes huge (for healthcare data/image data), then a deep L-layered neural network model will tend to overfit and model parameters will be very big real quick. In such situations CNNs are mostly used.

Consider image data for training and testing purposes. Each image can be represented as a 3D tensor having shape $(337 \times 600 \times 3)$, where 3 represents the color channels. 337 and 600 pixels are the length and width of the image. This can be depicted as:



The image tensor can be defined as a (600×3) matrix that repeated 337 times - total 606600 features. So, as evident we need a lot of weights to build the model and as a result the model will surely overfit.

Each neuron in a hidden layer owns a smaller set of weights that it repeatedly applies to local regions of the input volume and calculates an array of activation scores corresponding to each local region that will aid in learning its weights. There are some important things that need to be remembered. They are -

1. The CNN neurons don't look at all features, it will look at one small set of features that are locally together.

2. It will not calculate one raw score. It will calculate an array or a map of raw scores.
3. It uses the same weights for all local regions in input volume.

CNN Neuron vs. Fully Connected Neuron

A fully connected NN applies its weights to all the features from the previous layer. This approach doesn't scale well for images and it tends to overfit since there are a large number of weight parameters. On the contrary, a CNN mimics the efficient way the visual system of human works. A CNN neuron senses visual signals in a local area of the input volume and responds only to visual stimuli in that local area. The spatial extent of how much of the local area the CNN neuron looks at is called the receptive field of the neuron. CNN applies the same weights to multiple local areas of the input volume thereby drastically reducing the number of weight parameters and creates an array of visual stimuli responses called activation map.

Convolution with Filters

Each neuron calculates its activation map for a local region of the input volume through an operation called convolution; although, strictly speaking the operation is called cross-correlation. The number of neurons in the output volume is a hyperparameter also referred to as the number of filters or depth. Each filter corresponds to a set of weights and bias values such that the filter is strided across the input volume using a specific stride size, which is also a hyperparameter. We will also have to pad the input volume with zeros around the border; the size of this zero-padding is also a hyperparameter.

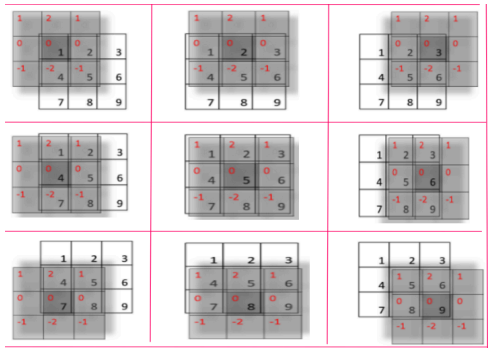
We will now see a simple example of convolving an input volume of shape (3×3) that is zero-padded with a (3×3)filter using stride 1; the entries of the filter can be seen as the weights owned by the neuron, and we are neglecting bias for simplicity. The shaded zero in the filter means it is the center

3×3 input volume convolved with a 3×3 filter

1	2	3
4	5	6
7	8	9

-1	-2	-1
0	0	0
1	2	1

Mechanism: Convolution is a dot product operation. For each entry in the input volume, starting with the first element 1, it is overlapped with the center of the filter, centered at 0 and then the dot product is calculated. The part that lies outside the overlapped region is padded with 0, so no result comes from the outside of the padded region. This process is then continued for all the elements of the input volume. Given as:



The result we get after this convolution operation is given as:

-13	-20	-17
-18	-24	-18
13	20	17

We can do that by overlapping the center of the filter with the first element of each part 9 times. The output of the convolution operation has the same shape of input volume. The resulting (3×3). The matrix is called a raw scores map calculated by the neuron which when activated results in its activation map. The output shape in any direction is given as:

$$\text{Shape} = \frac{\text{Input shape - Filter shape} + 2 \times \text{Padding}}{\text{Stride}} + 1$$

Stacking the results of k such neurons gives the output volume of shape $(3 \times 3 \times k)$. In fully connected NN, each neuron's raw scores are vectors. In CNN, each neuron's raw scores are matrices.

Convolution Layers

Just like a dense layer except that it accepts a volume as input, there are 4 hyper-parameters that are associated with the CNNs. They are -

1. Number of filters/neurons
2. Filter size
3. Filter stride
4. Amount of zero padding

Each filter has with itself associated with the shape calculated as:

$shape = (filter\ size \times filter\ size \times input\ volume\ depth\ weights) + One\ Bias\ term$

We have a sequence of convolution using filters along the depth direction. In the output volume, the 2D slice at depth d is the result of convolving the input volume with the d^{th} filter and add that filter's bias. Just like multiple dense layers in a fully connected NN, a CNN has multiple convolution layers with each layer learning a particular representation of the input volume in a hierarchical manner.

Pooling Layers

In CNN, it is essential to progressively reduce the spatial size of the representation. This ensures reduction in the number of parameters and prevents overfitting. Pooling layers operate independently on every depth slice of the input and reduce its size spatially using maximum operation. For example, a single input depth slice using a pooling filter of shape (2×2) applied with stride 2 can be represented as:

1	1	2	4
5	**6**	7	**8**
3	2	1	0
1	2	3	**4**

Max pool with (2*2) filter & stride 2 →

6	8
3	4

The effect is to down sample every depth slice in the input by 2 along both width and height while keeping the depth dimension unchanged.

Forward and Backward Propagation in CNNs

Forward propagation through a CNN layer i.e. getting its input from an input volume can be seen as a sequence of filter convolutions over input volume along its depth followed by a summation to generate the output volume of raw scores map. The output volume of raw scores maps are pointwise activated by the layers activation functions resulting in the volume of activation maps. The volume of activation maps acts as the input volume to the next convolution layer. The last few layers of a deep CNN for multiclass classification problems are typically fully connected layers followed by SoftMax activation.

Backward propagation through a convolution layer also involved a sequence of convolutions but with spatially-flipped filters. So, the shape of the gradient will depend on the batch size, height, width of the image pixel and the depth of the pixel.