



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

DEEP NEURAL NETWORK

MODULE # 2 : DEEP FEEDFORWARD NEURAL NETWORK

DNN Team, BITS Pilani

The author of this deck, Prof. Seetha Parameswaran,
is gratefully acknowledging the authors
who made their course materials freely available online.

TABLE OF CONTENTS

- 1 SINGLE NEURON FOR REGRESSION
- 2 SINGLE NEURON FOR BINARY CLASSIFICATION
- 3 MULTI-CLASS CLASSIFICATION
- 4 DEEP FEEDFORWARD NEURAL NETWORK
- 5 ACTIVATION FUNCTIONS
- 6 COMPUTATIONAL GRAPH
- 7 TRAINING DNN

LINEAR REGRESSION EXAMPLE

- Suppose that we wish to estimate the prices of houses (in dollars) based on their area (in square feet) and age (in years).
- The linearity assumption just says that the target (price) can be expressed as a weighted sum of the features (area and age):

$$\text{price} = w_{\text{area}} * \text{area} + w_{\text{age}} * \text{age} + b$$

$$y = m_1 x_1 + m_2 x_2 + \dots + m_n x_n + b$$

w_{area} and w_{age} are called weights, and b is called a bias.

d features

- The weights determine the influence of each feature on our prediction and the bias just says what value the predicted price should take when all of the features take value 0.

- The dataset comprises of training dataset and testing dataset. The split of DL is generally 99:1, as we are dealing with millions of examples.
- Each row is called an **example (or data point, data instance, sample)**.
- The thing we are trying to predict is called a **label (or target)**.
- The independent variables upon which the predictions are based are called **features (or covariates)**.

m — number of training examples

i — i^{th} example

$\rightarrow x^{(i)} = [x_1^{(1)}, x_1^{(2)}, \dots, x_1^{(m)}]$ — features of i^{th} example

$\leftarrow y^{(i)} = [y_1^{(1)}, y_1^{(2)}, \dots, y_1^{(m)}]$ — label of i^{th} example

AFFINE TRANSFORMATIONS AND LINEAR MODELS

- The equation of the form

$$\hat{y} = w_1 x_1 + \dots w_d x_d + b$$

$$\hat{\mathbf{y}} = \mathbf{w}^\top \mathbf{x} + b \quad \mathbf{x} \in \mathcal{R}^d \quad \mathbf{w} \in \mathcal{R}^d$$

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + b \quad \mathbf{X} \in \mathcal{R}^{m \times d} \quad \hat{\mathbf{y}} \in \mathcal{R}^m$$

linear
scalar valued -

is an **affine transformation** of input features, which is characterized by a linear transformation of features via weighted sum, combined with a translation via the added bias.

- Models whose output prediction is determined by the affine transformation of input features are **linear models**.
- The affine transformation is specified by the chosen weights (w) and bias (b).

LOSS FUNCTION

- Loss function is a quality measure for some given model or a measure of fitness.
- The loss function quantifies the distance between the real and predicted value of the target.
- The loss will usually be a non-negative number where smaller values are better and perfect predictions incur a loss of 0.
- The most popular loss function in regression problems is the **squared error**.

SQUARED ERROR LOSS FUNCTION

- The most popular loss function in regression problems is the squared error.
- For each example,

$$l^{(i)}(\mathbf{w}, b) = \frac{1}{2} (\hat{y}^{(i)} - y^{(i)})^2$$

- For the entire dataset of m examples, average (or equivalently, sum) the losses

$$\mathcal{L}(\mathbf{w}, b) = \frac{1}{m} \sum_{i=1}^m l^{(i)}(\mathbf{w}, b) = \frac{1}{m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})^2$$

- When training the model, find parameters $(\mathbf{w}_{opt}, b_{opt})$ that minimize the total loss across all training examples.

$$(\mathbf{w}_{opt}, b_{opt}) = \arg \min_{\mathbf{w}, b} \mathcal{L}(\mathbf{w}, b)$$

MINIBATCH STOCHASTIC GRADIENT DESCENT (SGD)

- Apply Gradient descent algorithm on a random minibatch of examples every time we need to compute the update.

- In each iteration,
 - ▶ Step 1: randomly sample a minibatch B consisting of a fixed number of training examples.

- ▶ Step 2: compute the derivative (gradient) of the average loss on the minibatch with regard to the model parameters.

- ▶ Step 3: multiply the gradient by a predetermined positive value η and subtract the resulting term from the current parameter values.

$\eta = 0.01$
 10 million
 $B = 2$
 $= 1024$
 1000 batches
 $\theta \leftarrow \theta - \eta \cdot g$
 1024

$$\begin{aligned}
 \mathbf{w} &\leftarrow \mathbf{w} - \frac{\eta}{|B|} \sum_{i \in B} \partial_{\mathbf{w}} \text{loss}^{(i)}(\mathbf{w}, b) \\
 b &\leftarrow b - \frac{\eta}{|B|} \sum_{i \in B} \partial_b \text{loss}^{(i)}(\mathbf{w}, b)
 \end{aligned}$$

$\frac{\partial \text{error}}{\partial \theta}$

TRAINING USING SGD ALGORITHM

- Initialize parameters (\mathbf{w}, b)
- Repeat until done
 - ▶ Compute gradient

$$\mathbf{g} \leftarrow \partial_{(\mathbf{w}, b)} \frac{1}{|B|} \sum_{i \in B} \text{loss}(\mathbf{x}^{(i)}, y^{(i)}, \mathbf{w}, b)$$

- ▶ Update parameters

$$(\mathbf{w}, b) \leftarrow (\mathbf{w}, b) - \eta \mathbf{g}$$

PS: The number of epochs and the learning rate are both hyperparameters. Setting hyperparameters requires some adjustment by trial and error.

PREDICTION

$y \rightarrow$ vector

one value

$$1. \text{ FP} \rightarrow \hat{y}^{(i)} = w^T x^{(i)} + b$$

$$2. \text{ Cost} \rightarrow L(w, b) = \frac{1}{m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2$$

- Estimating targets given features is commonly called **prediction or inference**.
- Given the learned model, values of target can be predicted, for any set of features.

$$3. \theta \leftarrow \theta - \eta \frac{\partial L(w, b)}{\partial \theta}$$

vector \leftarrow

One layer regression

TABLE OF CONTENTS

- 1 SINGLE NEURON FOR REGRESSION
- 2 SINGLE NEURON FOR BINARY CLASSIFICATION
- 3 MULTI-CLASS CLASSIFICATION
- 4 DEEP FEEDFORWARD NEURAL NETWORK
- 5 ACTIVATION FUNCTIONS
- 6 COMPUTATIONAL GRAPH
- 7 TRAINING DNN

BINARY CLASSIFICATION – TRAINING EXAMPLES

Single training example = $\{(x, y)\}$ where $x \in \mathcal{R}^d$ and $y \in \{0, 1\}$

training examples = m

m training examples = $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}) \dots (x^{(m)}, y^{(m)})\}$

y discrete
0 or 1

$$\mathbf{X} = \begin{bmatrix} | & | & \dots & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & & | \end{bmatrix} \quad \text{where } \mathbf{X} \in \mathcal{R}^{d \times m}$$

$$\mathbf{Y} = [y^{(1)} \quad y^{(2)} \quad \dots \quad y^{(m)}] \quad \text{where } \mathbf{Y} \in \mathcal{R}^{1 \times m}$$

FORWARD PROPAGATION

(1) FP (2)
$$z = w^T x + b$$

$$a = \sigma(z)$$

Given X find $\hat{y} = P(y = 1 | X)$

Input $X \in \mathcal{R}^{d \times m}$

Parameters $w \in \mathcal{R}^d$

$b \in \mathcal{R}$

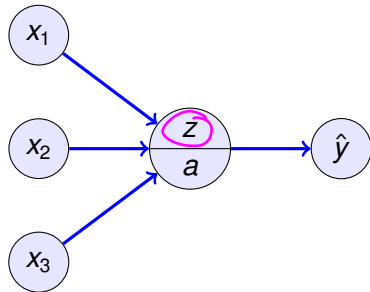
Activation $z = w^T x + b$

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

$$= \begin{cases} 1 & \text{if } z \text{ is large positive} \\ 0 & \text{if } z \text{ is large negative} \end{cases}$$

real
0 to 1

Output $\hat{y} = \sigma(w^T x + b)$ $0 \leq \hat{y} \leq 1$





COST FUNCTION FOR BINARY CLASSIFICATION

log 0 = large value

$\hat{y}(0,1)$

loss

$$\text{Loss function } \text{loss}(\hat{y}, y) = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$$

$0 + 1 \cdot \log 1 = 0$

If $y = 1$ $\text{loss}(\hat{y}, y) = -\log \hat{y}$

$\text{loss} \approx 0 \implies \log \hat{y} \approx \text{large} \implies \hat{y} \approx \text{large}$

$1 \log 1 + 0 = 0$

If $y = 0$ $\text{loss}(\hat{y}, y) = -\log(1 - \hat{y})$

$\text{loss} \approx 0 \implies \log(1 - \hat{y}) \approx \text{small} \implies \hat{y} \approx \text{small}$

$0 + 1 \log 0 = \text{large}$

$1 \log 0 + 0 = \text{large}$

Cost function $\mathcal{L}(w, b) = \frac{1}{m} \sum_{i=1}^m \text{loss}(\hat{y}^{(i)}, y^{(i)})$

$$= \frac{1}{m} \sum_{i=1}^m -[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

y < 0
y > 1
0, 1
correct

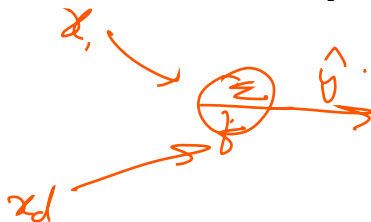
HOW TO LEARN PARAMETERS?

To Learn parameters use Gradient Descent Algorithm

$$\hat{y} = \sigma(\mathbf{w}^\top \mathbf{x} + b) \quad \text{where } \sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\mathcal{L}(\mathbf{w}, b) = \frac{1}{m} \sum_{i=1}^m - [y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

Find $\mathbf{w}_{opt}, b_{opt}$ that minimize $\mathcal{L}(\mathbf{w}, b)$.



GRADIENT DESCENT

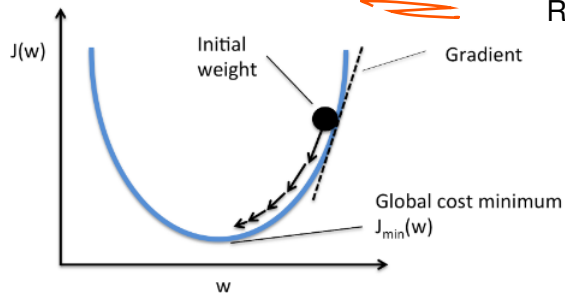


$$\frac{\partial \mathcal{L}(\mathbf{w}, b)}{\partial \mathbf{w}}$$

$$\frac{\partial \mathcal{L}(\mathbf{w}, b)}{\partial b}$$



Find $\mathbf{w}_{opt}, b_{opt}$ that minimize $\mathcal{L}(\mathbf{w}, b)$



Repeat {

$$w \leftarrow w - \eta \frac{\partial \mathcal{L}(\mathbf{w}, b)}{\partial w}$$

$$b \leftarrow b - \eta \frac{\partial \mathcal{L}(\mathbf{w}, b)}{\partial b}$$

}

$$\theta \leftarrow \theta - \eta g$$

TRAINING USING SGD ALGORITHM

- Initialize parameters (\mathbf{w}, b)
- Repeat until done
 - ▶ Compute gradient

$$\mathbf{g} \leftarrow \partial_{(\mathbf{w}, b)} \frac{1}{|B|} \sum_{i \in B} \text{loss}(\mathbf{x}^{(i)}, y^{(i)}, \mathbf{w}, b)$$

- ▶ Update parameters

$$(\mathbf{w}, b) \leftarrow (\mathbf{w}, b) - \eta \mathbf{g}$$

PS: The number of epochs and the learning rate are both hyperparameters. Setting hyperparameters requires some adjustment by trial and error.

EXERCISE - MSE LOSS



Consider the neural network with two inputs x_1 and x_2 and the initial weights are $w_0 = 0.5$, $w_1 = 0.8$, $w_2 = 0.3$. Draw the network, compute the output, mean squared loss function and weight updation when the input is $(1, 0)$, the learning rate is 0.01 and target output is 1. Assume any other relevant information.

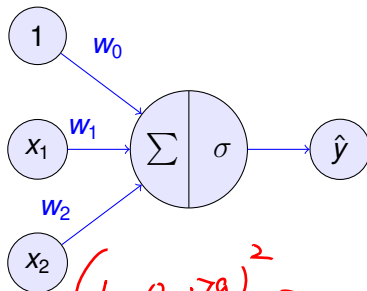
$$Z = w_0 + w_1 x_1 + w_2 x_2$$

$$= 1.7$$

$$a = \sigma(1.7)$$

$$= 0.79$$

$$mse = \frac{1}{2} \sum (y - \hat{y})^2$$



$$Loss = (y - \hat{y})^2$$

$$\frac{\partial Loss}{\partial w} = +2(y - \hat{y}) * (-1)$$

innovate

achieve

lead

$$\hat{y} = (1.3) = 0.79$$

$$g = \frac{\partial Loss}{\partial \theta}$$

$$= -2(y - \hat{y})$$

$$w \leftarrow w - \eta g$$

EXERCISE - MSE LOSS SOLUTION

$$\hat{y} = \sigma(w_0 + w_1x_1 + w_2x_2) = \sigma(0.5 + 0.8 * 1 + 0.3 * 0) = \sigma(1.3) = 0.7858$$

$$MSE = \frac{1}{2m} \sum_{i=1}^m (y - \hat{y})^2 = \frac{1}{2} (1 - 0.7858)^2 = \frac{0.04588}{2} = 0.02294$$

$$\mathbf{g} = \frac{2}{2m} \sum_{i=1}^m (y - \hat{y}) = (1 - 0.7858) = 0.2142$$

$$\text{new } w \leftarrow \text{old } w - \eta * \mathbf{g}$$

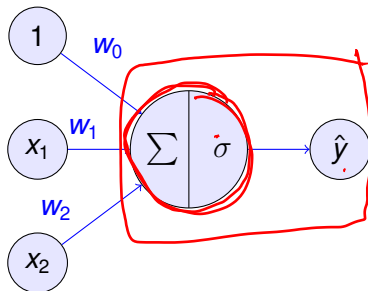
$$w_0 = 0.5 - 0.01 * 0.2142 = 0.4978$$

$$w_1 = 0.8 - 0.01 * 0.2142 = 0.7978$$

$$w_2 = 0.3 - 0.01 * 0.2142 = 0.2978$$

EXERCISE - BCE

Consider the neural network with two inputs x_1 and x_2 and the initial weights are $w_0 = 0.5$, $w_1 = 0.8$, $w_2 = 0.3$. Draw the network, compute the output, binary cross entropy loss function and weight updation when the input is $(1, 0)$, the learning rate is 0.01 and target output is 1. Assume any other relevant information.



EXERCISE - BCE SOLUTION

$$\hat{y} = \sigma(w_0 + w_1 x_1 + w_2 x_2) = \sigma(0.5 + 0.8 * 1 + 0.3 * 0) = \sigma(1.3) = 0.7858$$

$$\begin{aligned} Loss &= \frac{1}{m} \sum_{i=1}^m - [y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})] \\ &= -[1 \ln 0.7858 + (1 - 1) \ln(1 - 0.7858)] = 0.24 \end{aligned}$$

$$\mathbf{g} = \sum_{i=1}^m (y - \hat{y})z = (1 - 0.7858) * 1.3 = 0.278$$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta * \mathbf{g}$$

$$w_0 = 0.5 - 0.01 * 0.278 = 0.497$$

$$w_1 = 0.8 - 0.01 * 0.278 = 0.797$$

$$w_2 = 0.3 - 0.01 * 0.278 = 0.297$$

SINGLE-LAYER NEURAL NETWORK

- ~~Linear regression~~ is a single-layer neural network.
 - ▶ Number of inputs (or feature dimensionality) in the input layer is d . The inputs are x_1, \dots, x_d .
 - ▶ Number of outputs in the output layer is 1. The output is \hat{y} .
 - ▶ Number of layers for the neural network is 1. (conventionally we do not consider the input layer when counting layers.)
 - ▶ Every input is connected to every output, This transformation is a fully-connected layer or dense layer.

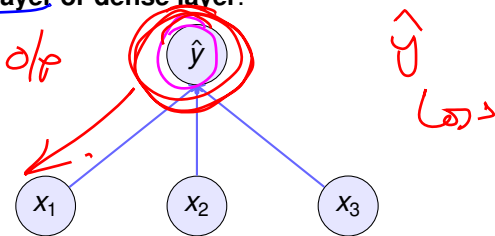


TABLE OF CONTENTS



$$B = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

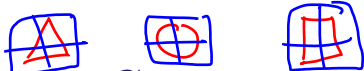
- 1 SINGLE NEURON FOR REGRESSION
- 2 SINGLE NEURON FOR BINARY CLASSIFICATION
- 3 MULTI-CLASS CLASSIFICATION
- 4 DEEP FEEDFORWARD NEURAL NETWORK
- 5 ACTIVATION FUNCTIONS
- 6 COMPUTATIONAL GRAPH
- 7 TRAINING DNN

$$Z = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

$$\hat{Y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

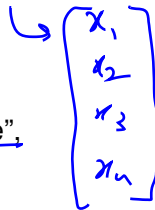
$$W = \begin{bmatrix} 3 \times 4 \end{bmatrix}$$

$$Z = WX + B$$
$$\hat{Y} = \text{softmax}(Z)$$



MULTI-CLASS CLASSIFICATION EXAMPLE

- Each input consists of a 2×2 grayscale image.
- Represent each pixel value with a single scalar, giving four features x_1, x_2, x_3, x_4 .
- Assume that each image belongs to one among the categories "square", "triangle", and "circle".
- How to represent the labels?



- ▶ Use label encoding.

0, 1, 2

$y \in 1, 2, 3$, where the integers represent circle, square, triangle respectively.

- ▶ Use one-hot encoding.

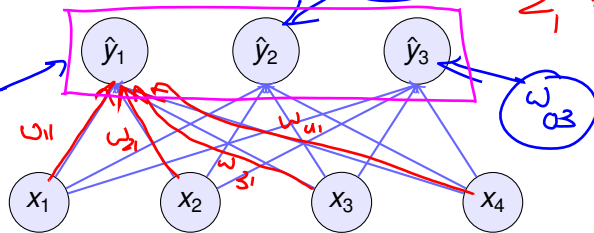
$y \in (1, 0, 0), (0, 1, 0), (0, 0, 1)$. y would be a three-dimensional vector, with $(1, 0, 0)$ corresponding to "circle", $(0, 1, 0)$ to "square", and $(0, 0, 1)$ to "triangle".

SINGLE-LAYER NEURAL NETWORK

$\langle X, Y \rangle$

- A model with multiple outputs, one per class. Each output will correspond to its own affine function.
- 4 features and 3 possible output categories
- Every input is connected to every output, This transformation is a **fully-connected layer or dense layer**.

class 1 $P(\hat{y}_i = 1 | x)$



$$z_i = w_{1,i}x_1 + w_{2,i}x_2 + w_{3,i}x_3 + w_{4,i}x_4 + w_{0,i}$$

softmax

$$\hat{y}_i = \text{softmax}(z_i)$$

ARCHITECTURE

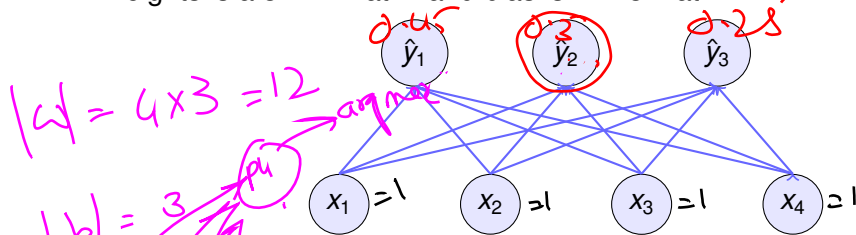
$$\checkmark \beta = \begin{bmatrix} 1 \\ 0.6 \\ 0.4 \end{bmatrix} \quad \text{arg max } \hat{y} = \begin{bmatrix} 0.45 \\ 0.30 \\ 0.25 \end{bmatrix}$$



0 to 1.

- 12 scalars to represent the weights and 3 scalars to represent the biases.
- Compute three logits, \hat{y}_1 , \hat{y}_2 , and \hat{y}_3 , for each input.
- Weights is a 3×4 matrix and bias is 1×3 matrix.

assume bias as 0



$$\hat{y}_i = \frac{e^{z_i}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

norm

$$z_1 = 2.72$$

$$\begin{aligned} z_1 &= x_1 w_{11} + x_2 w_{12} + x_3 w_{13} + x_4 w_{14} + b_1 = 1 & \hat{y}_1 &= \text{softmax}(z_1) \\ z_2 &= x_1 w_{21} + x_2 w_{22} + x_3 w_{23} + x_4 w_{24} + b_2 = 0.6 & \hat{y}_2 &= \text{softmax}(z_2) \\ z_3 &= x_1 w_{31} + x_2 w_{32} + x_3 w_{33} + x_4 w_{34} + b_3 = 0.4 & \hat{y}_3 &= \text{softmax}(z_3) \end{aligned}$$

denom = 6.03

SOFTMAX OPERATION

- Interpret the outputs of our model as probabilities.
- Any output \hat{y}_j is interpreted as the probability that a given item belongs to class j . Then choose the class with the largest output value as our prediction $\text{argmax}_j \hat{y}_j$.
- If \hat{y}_1 , \hat{y}_2 , and \hat{y}_3 are 0.1, 0.8, and 0.1, respectively, then predict category 2.
- The softmax function transforms the outputs such that they become non-negative and sum to 1, while requiring that the model remains differentiable.

$$\hat{y} = \text{softmax}(Z) \quad \text{where} \quad \hat{y}_j = \frac{\exp(z_j)}{\sum_k \exp(z_k)}$$

- First exponentiate each logit (ensuring non-negativity) and then divide by their sum (ensuring that they sum to 1).
- Softmax is a non-linear function.

LOG-LIKELIHOOD LOSS / CROSS-ENTROPY LOSS

- The softmax function gives a vector \hat{y} , which can be interpreted as estimated conditional probabilities of each class given any input x ,

$$\hat{y}_j = P(y = \text{class}_j | x)$$

- Compare the estimates with reality by checking how probable the actual classes are according to our model, given the features:

$$\begin{matrix} P(\hat{y}_1 | x) \\ P(\hat{y}_2 | x) \end{matrix}$$

$$P(Y | X) = \prod_{i=1}^m P(y^{(i)} | x^{(i)})$$

→ conditional

$$P(Y | X)$$

argmax $\hat{y} = \begin{bmatrix} 0.45 \\ 0.30 \\ 0.25 \end{bmatrix}$ ✓

$x \rightarrow \text{class 2}$

class 2 $[0 \ 1 \ 0]^T$



LOG-LIKELIHOOD LOSS / CROSS-ENTROPY LOSS

$$P(y|x) = \frac{e^{z_i}}{\sum e^{z_i}}$$

- Maximize $P(Y | X)$ = Minimize the negative log-likelihood

m classes

$$-\log P(Y | X) = \sum_{i=1}^m -\log P(y^{(i)} | x^{(i)}) = \sum_{i=1}^m \text{loss} \left[\underbrace{P(y^{(i)}, \hat{y}^{(i)})}_{\text{joint}} \right]$$

conditional

$$\text{loss } P(y^{(i)}, \hat{y}^{(i)}) = - \sum_{j=1}^m y_j \log \hat{y}_j$$

$$\text{loss} = - \left(y_1 \log \hat{y}_1 + y_2 \log \hat{y}_2 + y_3 \log \hat{y}_3 \right)$$

0 log 0.45 + 1 log 0.3 + 0 log 0.25

SOFTMAX EXAMPLE

- z_1 = Un-normalized log probabilities
- e^{z_1} = Un-normalized probabilities
- $\hat{y}_i = \text{softmax}(z_1)$ = normalized probabilities
- $L_i = \text{loss} = y_i \log \hat{y}_i$

class	z_1	e^{z_1}	$\text{softmax}(z_1)$	L_1
cat	3.2	24.5	0.13	0.89
car	5.1	164.0	0.87	0.06
dog	-1.7	0.18	0.00	∞

TABLE OF CONTENTS

- 1 SINGLE NEURON FOR REGRESSION
- 2 SINGLE NEURON FOR BINARY CLASSIFICATION
- 3 MULTI-CLASS CLASSIFICATION
- 4 DEEP FEEDFORWARD NEURAL NETWORK**
- 5 ACTIVATION FUNCTIONS
- 6 COMPUTATIONAL GRAPH
- 7 TRAINING DNN

DEEP FEEDFORWARD NEURAL NETWORK

- A neural network with more number of hidden layers is considered as a **deep neural network**. There are no feedback connections.
- Convolutional networks are examples of feedforward neural networks.

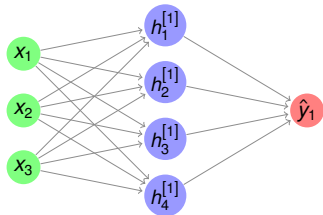


FIGURE: Shallow Neural Network

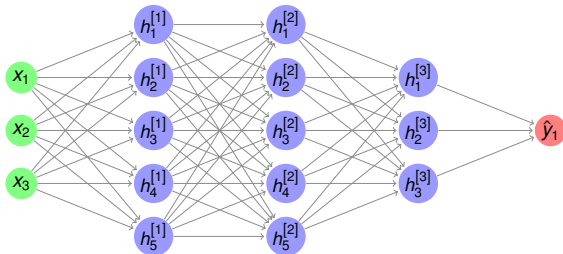


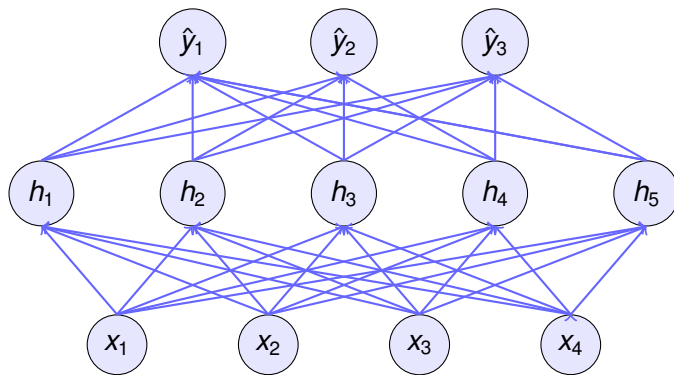
FIGURE: Deep Neural Network



DEEP FEEDFORWARD NEURAL NETWORK (DNN)

- With deep neural networks, use the data to jointly learn both a representation via hidden layers and a linear predictor that acts upon that representation.
- Add many hidden layers by stacking many fully-connected layers on top of each other. Each layer feeds into the layer above it, until we generate outputs.
- The first $(L - 1)$ layers learns the representation and the final layer is the linear predictor.

DNN ARCHITECTURE



- DNN has 4 inputs, 3 outputs, and its hidden layer contains 5 hidden units.
- Number of layers in this DNN is 2.

DNN ARCHITECTURE

- The layers are fully connected.
- Every input influences every neuron in the hidden layer.
- Each of the hidden neurons in turn influences every neuron in the output layer.
- The outputs of the hidden layer are called as **hidden representations** or hidden-layer variable or a hidden variable.

DNN ARCHITECTURE

HIDDEN LAYERS – intermediate layers, desired output for each of these layers are not shown.

DEPTH – number of layers.

WIDTH – dimensionality of the hidden layers

DNN – GENERAL STRATEGY

- Design a DNN architecture of the network. Architecture depends on the problem / application / complexity of the decision boundary.
- Choose the activation functions that will be used to compute the hidden layer activations. The activation function is the same for all neurons in a layer. Activation function can change between layers.
- Choose the cost function. It depends on whether regression, binary classification or multi-class classification.
- Choose the optimizer algorithm, depends on the complexity and variance of the input data.
- Train the feedforward network. Learning in deep neural networks requires computing the gradients using the back-propagation algorithm.
- Evaluate the performance of the network.

NON-LINEARITY IN DNN

- Input is $X \in \mathcal{R}^{m \times d}$ with m examples where each example has d features.
- Hidden layer has h hidden units $H \in \mathcal{R}^{m \times h}$.
- Hidden layer weights $W^{(1)} \in \mathcal{R}^{d \times h}$ and biases $b^{(1)} \in \mathcal{R}^{1 \times h}$
- Output layer weights $W^{(2)} \in \mathcal{R}^{h \times q}$ and biases $b^{(2)} \in \mathcal{R}^{1 \times q}$
- Output is $\hat{Y} \in \mathcal{R}^{m \times q}$
- A non-linear activation function σ has to be applied to each hidden unit following the affine transformation. The outputs of activation functions are called **activations**.

$$H = \sigma(XW^{(1)} + b^{(1)})$$

$$\hat{Y} = \text{softmax}(XW^{(2)} + b^{(2)})$$

TABLE OF CONTENTS

- ① SINGLE NEURON FOR REGRESSION
- ② SINGLE NEURON FOR BINARY CLASSIFICATION
- ③ MULTI-CLASS CLASSIFICATION
- ④ DEEP FEEDFORWARD NEURAL NETWORK
- ⑤ **ACTIVATION FUNCTIONS**
- ⑥ COMPUTATIONAL GRAPH
- ⑦ TRAINING DNN

ACTIVATION FUNCTIONS

- Activation function of a neuron defines the output of that neuron given an input or set of inputs.
- Introduces non-linearity to a neuron.
- A non-activated neuron will act as a linear regression with limited learning.
- Multilayered deep neural networks learn meaningful features from data.
- Artificial neural networks are designed as universal function approximators, they must have the ability to calculate and learn any non-linear function.

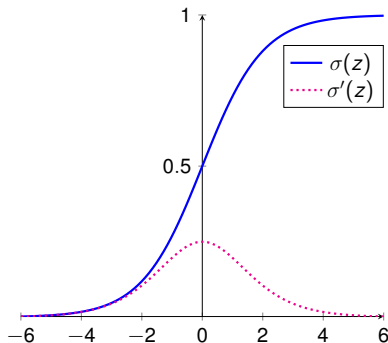
SIGMOID (LOGISTIC) ACTIVATION FUNCTION

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

Derivative: $\sigma'(z) = \sigma(z)(1 - \sigma(z))$

Range : $(0, 1)$

- Non-Linear function
- Small changes in z will be large in $f(z)$. This means it is a good classifier.
- Leads to vanishing gradient. So the learning is minimal.

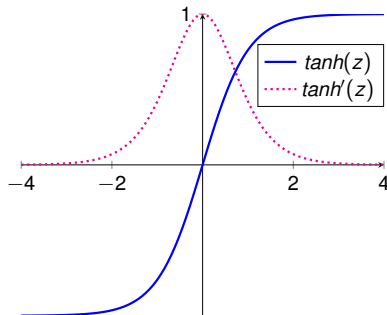


TANH ACTIVATION FUNCTION

Function: $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

Derivative: $\tanh'(z) = (1 - \tanh^2(z))$

Range : $(-1, 1)$



TANH ACTIVATION FUNCTIONS

- More efficient because it has a wider range for faster learning and grading.
- The tanh activation usually works better than sigmoid activation function for hidden units because the mean of its output is closer to zero, and so it centers the data better for the next layer.
- Issues with tanh
 - ▶ computationally expensive
 - ▶ lead to vanishing gradients

If we initialize the weights to relative large values, this will cause the inputs of the tanh to also be very large, thus causing gradients to be close to zero. The optimization algorithm will thus become slow.

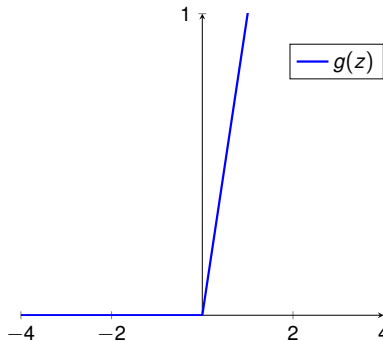
RELU ACTIVATION FUNCTION

Rectified Linear Unit

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

Derivative: $g'(z) = \begin{cases} 0 & \text{for } z \leq 0 \\ 1 & \text{for } z > 0 \end{cases}$

Range : $[0, \infty]$



ReLU ACTIVATION FUNCTIONS

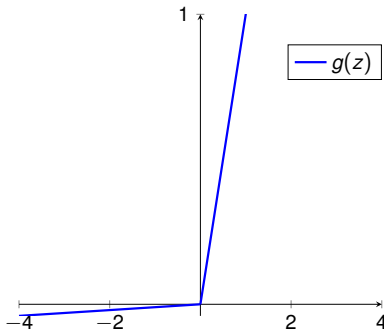
- ReLU activation function is not differentiable at origin.
- If we combine two ReLU units, we can recover a piece-wise linear approximation of the Sigmoid function.
- ReLU activation can lead to exploding gradient.
- In zero value region, learning is not happening.
- Fast Learning
- Fewer vanishing gradient problems
- Sparse activation
- Efficient computation
- Scale invariant (max operation)
- Non Zero centered
- Non differentiable at Zero

LEAKY RELU ACTIVATION FUNCTION

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

Derivative: $g'(z) = \begin{cases} 0.01 & \text{for } z < 0 \\ 1 & \text{for } z \geq 0 \end{cases}$

Range : $[-\infty, \infty]$



PARAMETRIC RELU (PRELU) ACTIVATION FUNCTION

Function: $g(z) = \max(az, z)$

Derivative: $g'(z) = \begin{cases} a & \text{for } z < 0 \\ 1 & \text{for } z \geq 0 \end{cases}$

Range : $[-\infty, \infty]$

ELU (EXPONENTIAL LINEAR UNITS)









$$\text{Function: } g(z) = \begin{cases} z & \text{for } z < 0 \\ a(e^z - 1) & \text{for } z \geq 0 \end{cases}$$

$$\text{Range : } [-\infty, \infty]$$

COMPARING ACTIVATION FUNCTIONS

Activation Function	Sigmoid	Tanh	ReLU
Linearity	Non Linear	Non Linear	Non Linear
Activation Function	$\sigma(x) = \frac{1}{1+e^{-x}}$	$\tanh(z) = \frac{(e^z - e^{-z})}{(e^z + e^{-z})}$	$g(x) = \max(0, x)$
Derivative	$\sigma(z)(1 - \sigma(z))$	$(1 - \tanh^2(z))$	1 if $x > 0$ else 0
Symmetric Function	No	Yes	No
Range	$[0, 1]$	$[-1, 1]$	$[0, \infty]$
Vanishing Gradient	Yes	Yes	No
Exploding Gradient	No	No	Yes
Zero Centered	No	Yes	No

COMPARING ACTIVATION FUNCTIONS

ACTIVATION FUNCTION	PLOT	EQUATION	DERIVATIVE	RANGE
Linear		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$
Binary Step		$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$\{0, 1\}$
Sigmoid		$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	$(0, 1)$
Hyperbolic Tangent(tanh)		$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$
Rectified Linear Unit(ReLU)		$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$[0, \infty)$
Softplus		$f(x) = \ln(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$	$(0, 1)$
Leaky ReLU		$f(x) = \begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$(-1, 1)$
Exponential Linear Unit(ELU)		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$	$f'(x) = \begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$	$[0, \infty)$

ACTIVATION FUNCTIONS PLOTS

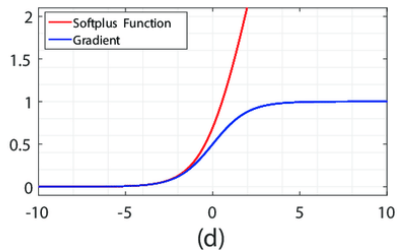
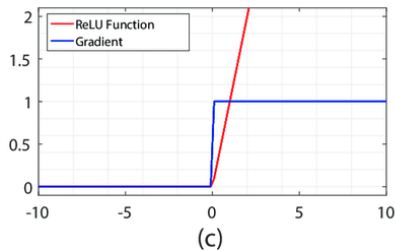
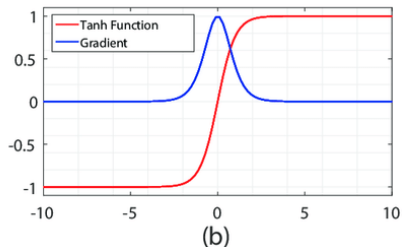
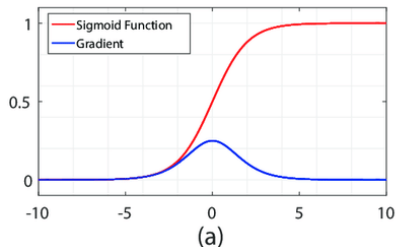


TABLE OF CONTENTS

- 1 SINGLE NEURON FOR REGRESSION
- 2 SINGLE NEURON FOR BINARY CLASSIFICATION
- 3 MULTI-CLASS CLASSIFICATION
- 4 DEEP FEEDFORWARD NEURAL NETWORK
- 5 ACTIVATION FUNCTIONS
- 6 COMPUTATIONAL GRAPH
- 7 TRAINING DNN

COMPUTATION GRAPHS

- Computations of a neural network are organized in terms of
 - ▶ a forward pass or a forward propagation step, in which we compute the output of the neural network,
 - ▶ followed by a backward pass or back propagation step, which we use to compute gradients or derivatives.
- The computation graph organizes a computation with this blue arrow, left-to-right computation.
- The backward red arrow, right-to-left shows computation of the derivatives.

COMPUTATION GRAPHS

- Each node in the graph indicate a variable.
- An operation is a simple function of one or more variable.
- An operation is defined such that it returns only a single output variable.
- If a variable y is computed by applying an operation to a variable x , then we draw a directed edge from x to y .

COMPUTATION GRAPHS EXAMPLE

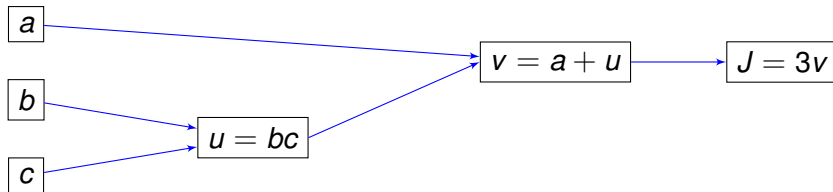
Forward Pass:

$$\text{Let } J(a, b, c) = 3(a + bc)$$

$$u = bc$$

$$v = a + u$$

$$\text{Then } J = 3v$$



COMPUTATION GRAPHS EXAMPLE

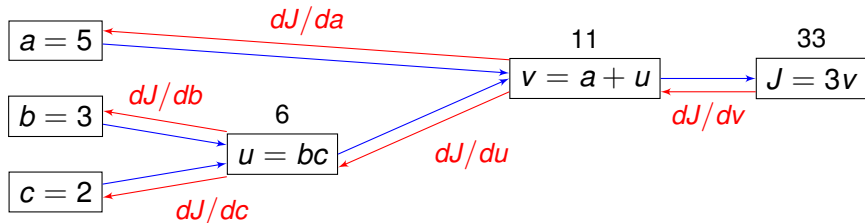
Backward Pass / Back propagation of gradients:

$$\text{Let } J(a, b, c) = 3(a + bc)$$

$$u = bc$$

$$v = a + u$$

$$\text{Then } J = 3v$$

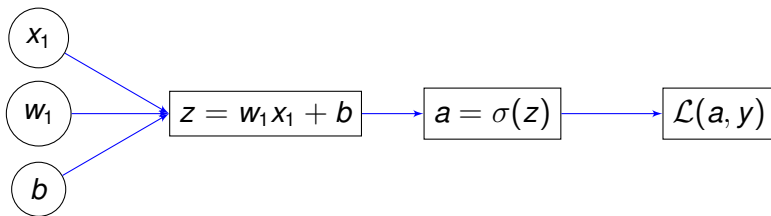


COMPUTATION GRAPH – BINARY CLASSIFICATION

$$z = w^T X + b$$

$$a = \hat{y} = \sigma(z)$$

$$\mathcal{L}(a, y) = -[y \log a + (1 - y) \log(1 - a)]$$



COMPUTATION GRAPH – BINARY CLASSIFICATION

$$z = w^T X + b$$

$$a = \hat{y} = \sigma(z)$$

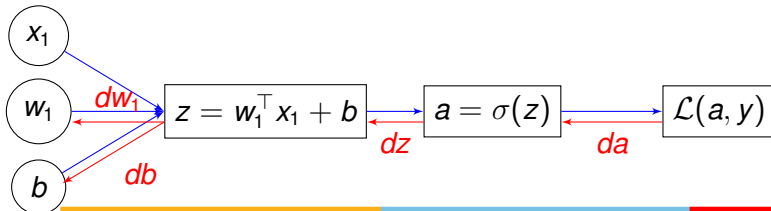
$$\mathcal{L}(a, y) = -[y \log a + (1 - y) \log(1 - a)]$$

$$da = \frac{d\mathcal{L}(a, y)}{da} = \frac{-y}{a} + \frac{1 + y}{1 + a}$$

$$dz = \frac{d\mathcal{L}(a, y)}{dz} = a - y$$

$$dw_1 = \frac{d\mathcal{L}(a, y)}{dw_1} = x_1 dz$$

$$db = dz$$



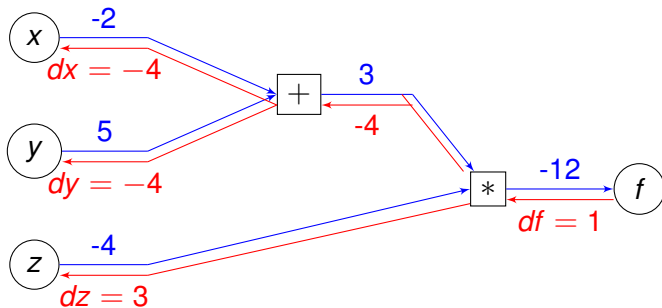
EXERCISE

Draw the computational graph for the equation

$$f(x, y, z) = (x + y)z$$

Assume that $x = -2$, $y = 5$ and $z = -4$. Using the computation graph show the computation of the gradients also.

EXERCISE - SOLUTION



EXERCISE

Draw the computational graph for the equation

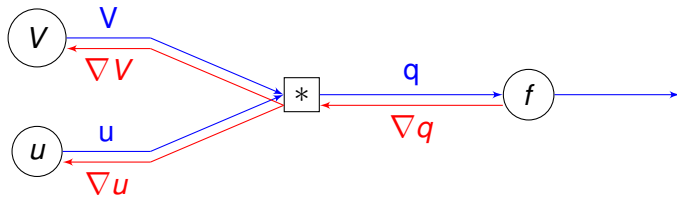
$$f(u, V) = || V \cdot u ||^2 = \sum_{i=1}^n (V \cdot u)_i^2$$

Using the computation graph show the computation of the gradients also. Assume that

$$V = \begin{bmatrix} 0.1 & 0.5 \\ -0.3 & 0.8 \end{bmatrix}$$

$$u = \begin{bmatrix} 0.2 \\ 0.4 \end{bmatrix}$$

EXERCISE - SOLUTION



EXERCISE - SOLUTION

$$V = \begin{bmatrix} 0.1 & 0.5 \\ -0.3 & 0.8 \end{bmatrix}$$

$$u = \begin{bmatrix} 0.2 \\ 0.4 \end{bmatrix}$$

$$q = V \cdot u = \begin{bmatrix} 0.1 * 0.2 + 0.5 * 0.4 \\ -0.3 * 0.2 + 0.8 * 0.4 \end{bmatrix} = \begin{bmatrix} 0.22 \\ 0.26 \end{bmatrix}$$

$$f = \sum_{i=1} q_i^2 = 0.22^2 + 0.26^2 = 0.116$$

EXERCISE - SOLUTION

$$\nabla q = \nabla_q f = 2q = 2 \begin{bmatrix} 0.22 \\ 0.26 \end{bmatrix} = \begin{bmatrix} 0.44 \\ 0.52 \end{bmatrix}$$

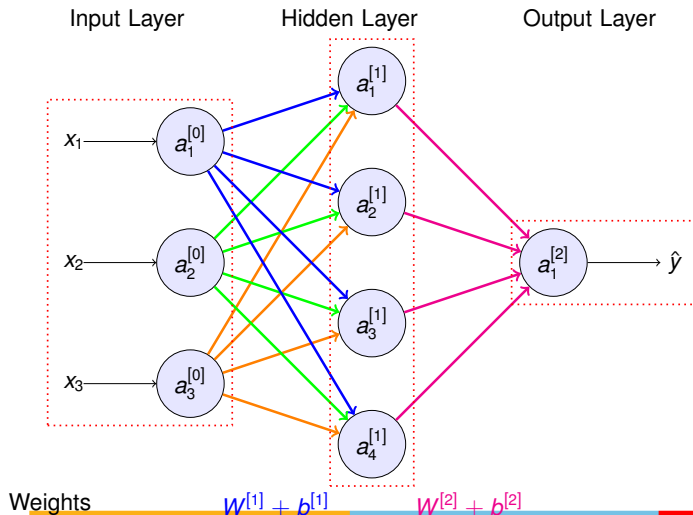
$$\nabla V = \nabla_V f = 2q \cdot u^\top = 2 \begin{bmatrix} 0.22 \\ 0.26 \end{bmatrix} \begin{bmatrix} 0.2 & 0.4 \end{bmatrix} = \begin{bmatrix} 0.088 & 0.176 \\ 0.105 & 0.208 \end{bmatrix}$$

$$\nabla u = \nabla_u f = 2V^\top \cdot q = 2 \begin{bmatrix} 0.1 & -0.3 \\ 0.5 & 0.8 \end{bmatrix} \begin{bmatrix} 0.22 \\ 0.26 \end{bmatrix} = \begin{bmatrix} -0.112 \\ 0.636 \end{bmatrix}$$

TABLE OF CONTENTS

- 1 SINGLE NEURON FOR REGRESSION
- 2 SINGLE NEURON FOR BINARY CLASSIFICATION
- 3 MULTI-CLASS CLASSIFICATION
- 4 DEEP FEEDFORWARD NEURAL NETWORK
- 5 ACTIVATION FUNCTIONS
- 6 COMPUTATIONAL GRAPH
- 7 TRAINING DNN

TWO LAYER NEURAL NETWORK ARCHITECTURE



COMPUTING THE ACTIVATIONS

For Hidden layer

$$z_1^{[1]} = w_1^{[1]} a^{[0]} + b^{[1]}$$

$$a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_2^{[1]} = w_2^{[1]} a^{[0]} + b^{[1]}$$

$$a_2^{[1]} = \sigma(z_2^{[1]})$$

$$z_3^{[1]} = w_3^{[1]} a^{[0]} + b^{[1]}$$

$$a_3^{[1]} = \sigma(z_3^{[1]})$$

$$z_4^{[1]} = w_4^{[1]} a^{[0]} + b^{[1]}$$

$$a_4^{[1]} = \sigma(z_4^{[1]})$$

In the matrix form

$$\begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix} = \underbrace{\begin{bmatrix} \dots & w_1^{[1]T} & \dots \\ \dots & w_2^{[1]T} & \dots \\ \dots & w_3^{[1]T} & \dots \\ \dots & w_4^{[1]T} & \dots \end{bmatrix}}_{4 \times 3} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{3 \times 1} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}}_{4 \times 1}$$

$$z^{[1]} = \begin{bmatrix} w_1^{[1]T} a^{[0]} + b_1^{[1]} \\ w_2^{[1]T} a^{[0]} + b_2^{[1]} \\ w_3^{[1]T} a^{[0]} + b_3^{[1]} \\ w_4^{[1]T} a^{[0]} + b_4^{[1]} \end{bmatrix} \leftarrow 4 \times 1 \text{ matrix}$$

$$a^{[1]} = \sigma(z^{[1]}) \leftarrow 4 \times 1 \text{ matrix}$$

ACTIVATIONS FOR TRAINING EXAMPLES

Vectorizing for one training example

$$a^{[0]} = X$$

$$Z^{[1]} = W^{[1]} a^{[0]} + b^{[1]}$$

$$a^{[1]} = \sigma(Z^{[1]})$$

$$Z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$

$$a^{[2]} = \sigma(Z^{[2]})$$

$$\hat{y} = a^{[2]}$$

Vectorizing for all training examples

$$\text{Example 1 : } X^{(1)} \longrightarrow a^{[2](1)} = \hat{y}^{(1)}$$

$$\text{Example 2 : } X^{(2)} \longrightarrow a^{2} = \hat{y}^{(2)}$$

$$\text{Example 3 : } X^{(2)} \longrightarrow a^{[2](3)} = \hat{y}^{(3)}$$

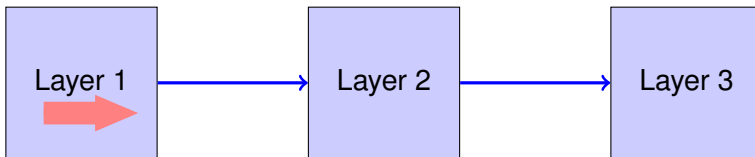
$$\vdots$$

$$\vdots$$

$$\text{Example m : } X^{(m)} \longrightarrow a^{[2](m)} = \hat{y}^{(m)}$$

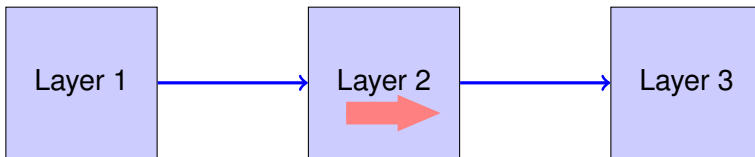
NEURAL NETWORK TRAINING - FORWARD PASS

Step 1: Compute Loss on mini-batch [Forward pass]



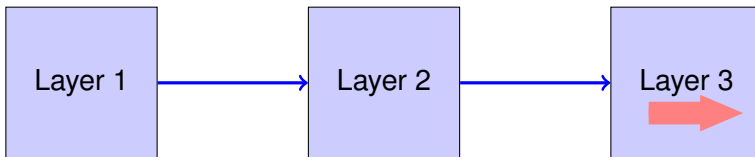
NEURAL NETWORK TRAINING - FORWARD PASS

Step 1: Compute Loss on mini-batch [Forward pass]



NEURAL NETWORK TRAINING - FORWARD PASS

Step 1: Compute Loss on mini-batch [Forward pass]



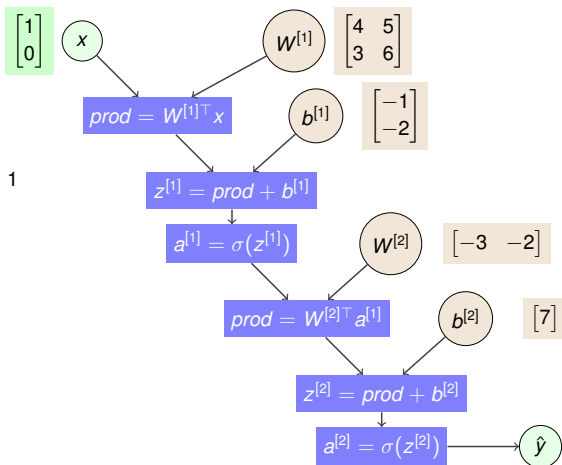
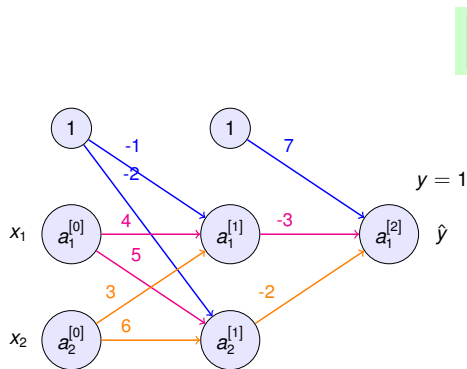
FORWARD PROPAGATION ALGORITHM

Algorithm 1: FORWARD PROPAGATION

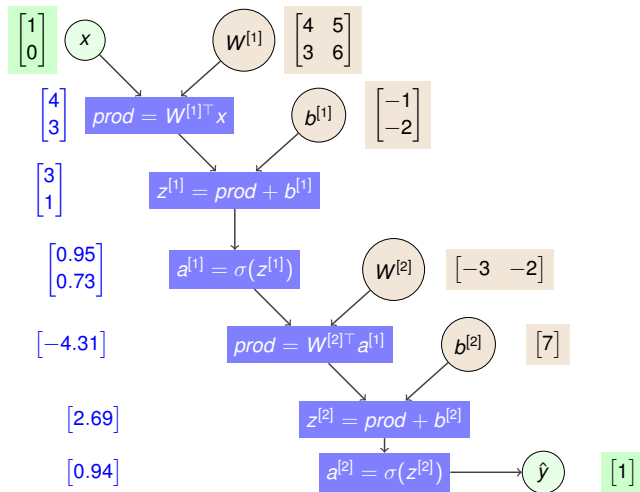
- 1 Initialize the weights and bias randomly.
- 2 $a^{[0]} = X^{(i)}$
- 3 $Z^{[1]} = W^{[1]}a^{[0]} + b^{[1]}$
- 4 $a^{[1]} = \sigma(Z^{[1]})$
- 5 $Z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$
- 6 $a^{[2]} = \sigma(Z^{[2]})$
- 7 $\hat{y} = \begin{cases} 1 & \text{if } a^{[2]} > 0.5 \\ 0 & \text{otherwise} \end{cases}$

Note: All training examples are considered in the vectorized form.

EXAMPLE NEURAL NETWORK



COMPUTATION GRAPH FOR FORWARD PASS



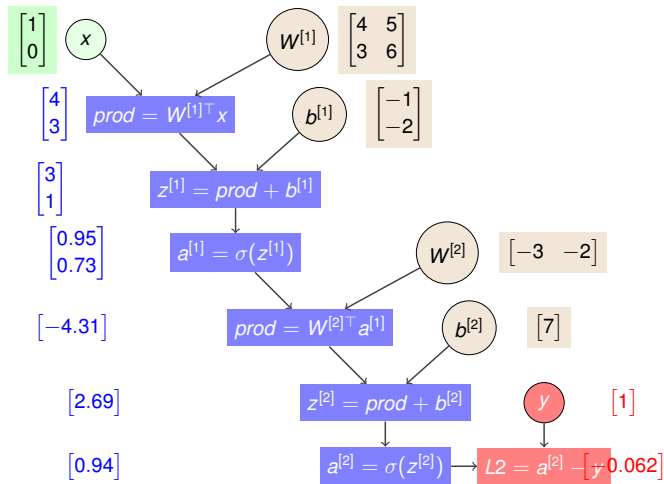
COST FUNCTION

- The difference between the actual observation y and the computed activation \hat{y} gives the error or the cost function.
- For binary classification,

$$\mathcal{L} = \frac{1}{m} \sum_{i=0}^m \text{loss}(\hat{y}^{(i)}, y^{(i)})$$

$$\mathcal{L} = -\frac{1}{m} \sum_{i=0}^m (y^{(i)} \log(a^{[2](i)}) + (1 - y^{(i)}) \log(1 - a^{[2](i)}))$$

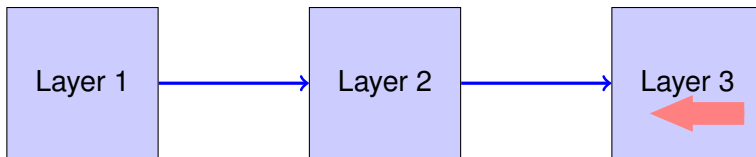
COMPUTATION GRAPH FOR COST FUNCTION



NEURAL NETWORK TRAINING - BACKWARD PASS

Step 1: Compute Loss on mini-batch [Forward pass]

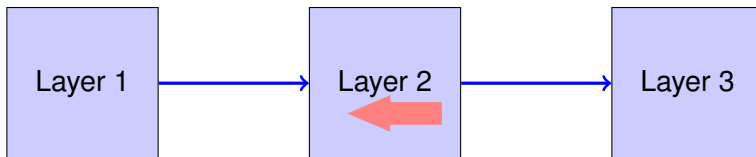
Step 2: Compute gradients wrt parameters [Backward pass]



NEURAL NETWORK TRAINING - BACKWARD PASS

Step 1: Compute Loss on mini-batch [Forward pass]

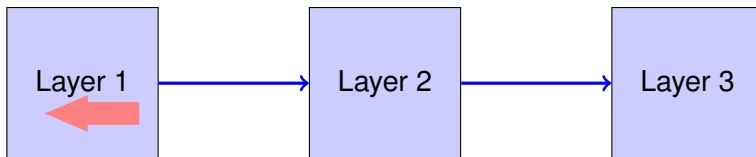
Step 2: Compute gradients wrt parameters [Backward pass]



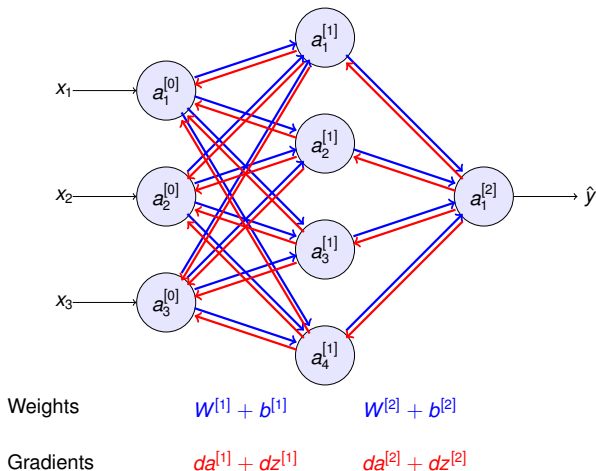
NEURAL NETWORK TRAINING - BACKWARD PASS

Step 1: Compute Loss on mini-batch [Forward pass]

Step 2: Compute gradients wrt parameters [Backward pass]



BACK PROPAGATION OF COST FUNCTION



COMPUTING THE GRADIENTS

For all training examples

$$dZ^{[2]} = A^{[2]} - Y$$

$$dW^{[2]} = \frac{1}{m} (dZ^{[2]} \cdot A^{[1]T})$$

$$db^{[2]} = \frac{1}{m} \sum dZ^{[2]}$$

$$dZ^{[1]} = (W^{[2]T} dZ^{[2]}) * \sigma'(Z^{[1]})$$

$$dW^{[1]} = \frac{1}{m} (dZ^{[1]} \cdot X^T)$$

$$db^{[1]} = \frac{1}{m} \sum dZ^{[1]}$$

For one training example

$$dz^{[2]} = a^{[2]} - y$$

$$dw^{[2]} = dz^{[2]} \cdot a^{[1]T}$$

$$db^{[2]} = dz^{[2]}$$

$$dz^{[1]} = (w^{[2]T} dz^{[2]}) * \sigma'(z^{[1]})$$

$$dw^{[1]} = dz^{[1]} \cdot x^T$$

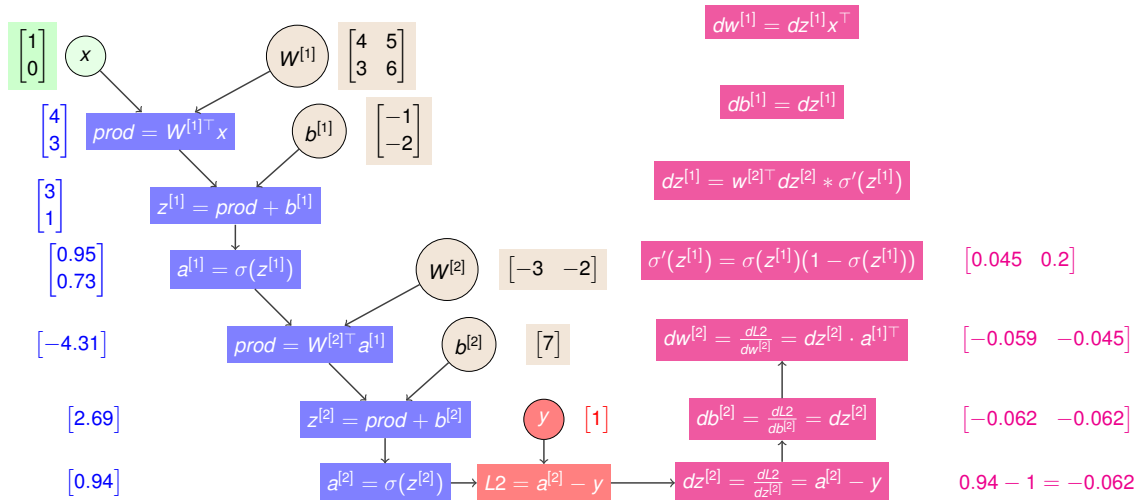
$$db^{[1]} = dz^{[1]}$$

BACKWARD PROPAGATION ALGORITHM

Algorithm 2: BACKWARD PROPAGATION

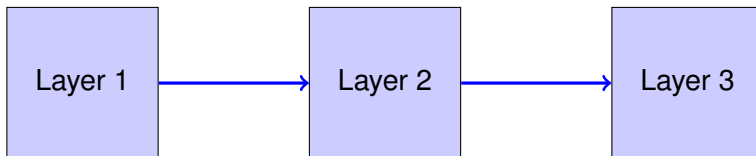
- 1 $dZ^{[2]} = A^{[2]} - Y$
 - 2 $dW^{[2]} = \frac{1}{m} (dZ^{[2]} \cdot A^{[1]T})$
 - 3 $db^{[2]} = \frac{1}{m} \sum dZ^{[2]}$
 - 4 $dZ^{[1]} = (W^{[2]T} dZ^{[2]}) * \sigma'(Z^{[1]})$
 - 5 $dW^{[1]} = \frac{1}{m} (dZ^{[1]} \cdot X^T)$
 - 6 $db^{[1]} = \frac{1}{m} \sum dZ^{[1]}$
-

COMPUTATION GRAPH FOR BACKWARD PASS



NEURAL NETWORK TRAINING - BACKWARD PASS

- Step 1: Compute Loss on mini-batch [Forward pass]
- Step 2: Compute gradients wrt parameters [Backward pass]
- Step 3: Use gradients to update parameters



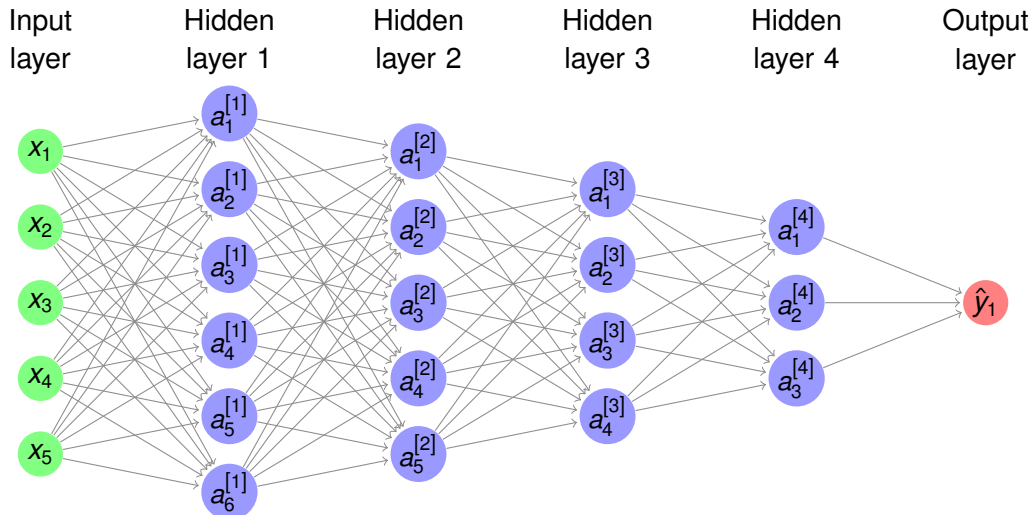
$$\theta \leftarrow \theta - \eta \frac{d\mathcal{L}}{d\theta}$$

UPDATE THE WEIGHTS AND BIAS

Algorithm 3: WEIGHT UPDATION

- 1 $W^{[1]} = W^{[1]} - \text{learning rate} * dW^{[1]}$
 - 2 $b^{[1]} = b^{[1]} - \text{learning rate} * db^{[1]}$
 - 3 $W^{[2]} = W^{[2]} - \text{learning rate} * dW^{[2]}$
 - 4 $b^{[2]} = b^{[2]} - \text{learning rate} * db^{[2]}$
-

DNN ARCHITECTURE



Requires

- ➊ **Forward Pass** through each layer to compute the output.
- ➋ Compute the deviation or error between the desired output and computed output in the forward pass (first step). This morphs into **objective function**, as we want to minimize this deviation or error.
- ➌ The deviation has to be send back through each layer to compute the delta or change in the parameter values. This is achieved using **back propagation algorithm**.
- ➍ **Update** the parameters.

DNN ARCHITECTURE

Notations

- L : Number of layers
- $n^{[l]}$: Number of units in layer l
- $w^{[l]}$: Weights for layer l
- $b^{[l]}$: Bias for layer l
- $z^{[l]}$: Hypothesis for layer l
- $g^{[l]}$: Activation Function used for layer l
- $a^{[l]}$: Activation for layer l

Matrix Dimensions

- $W^{[l]} : n^{[l]} \times n^{[l-1]}$
- $b^{[l]} : n^{[l]} \times 1$
- $z^{[l]} : n^{[l]} \times 1$
- $a^{[l]} : n^{[l]} \times 1$
- $dW^{[l]} : n^{[l]} \times n^{[l-1]}$
- $db^{[l]} : n^{[l]} \times 1$
- $dz^{[l]} : n^{[l]} \times 1$
- $da^{[l]} : n^{[l]} \times 1$

FORWARD PROPAGATION

- The inputs \mathbf{X} provide the initial information that then propagates up to the hidden units at each layer and finally produces \hat{y} . This is called forward propagation.
- Information flows forward through the network.
- During training, it produces a scalar cost $\mathcal{L}(\theta)$.

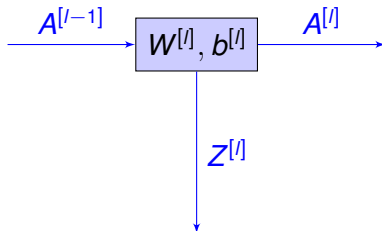
FORWARD PROPAGATION ALGORITHM

Algorithm 4: Forward Propagation

```

1 for  $l$  in range (1,  $L$ ) do
2    $Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}$ 
3    $A^{[l]} = g^{[l]}(Z^{[l]})$ 
4  $\hat{y} = A^{[L]}$ 
5  $J = L(\hat{y}, y) + \lambda\Omega(\theta)$ 

```



BACKWARD PROPAGATION

- Back-propagation algorithm or backprop, allows the information from the cost to then flow backwards through the network, in order to compute the gradient $\nabla_{\theta} J(\theta)$.
- Back-propagation refers only to the method for computing the gradient.

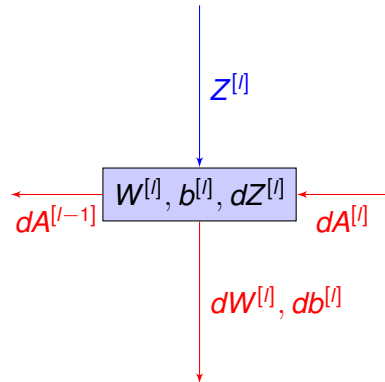
BACKWARD PROPAGATION ALGORITHM

Algorithm 5: Backward Propagation

```

1  $da^{[L]} = \frac{-y}{a} + \frac{1-y}{1-a}$ 
2 for  $l$  in range  $(1, L)$  do
3    $dZ^{[l]} = dA^{[l]} * g'^{[l]}(Z^{[l]})$ 
4    $dW^{[l]} = dZ^{[l]} \cdot A^{[l-1]T}$ 
5    $db^{[l]} = dZ^{[l]}$ 
6    $dA^{[l-1]} = W^{[l]T} \cdot dZ^{[l]}$ 

```



UPDATE THE WEIGHTS AND BIAS

Algorithm 6: Weight Updation

```
1 for  $l$  in range (1,  $L$ ) do  
2    $W^{[l]} = W^{[l]} - \eta * dW^{[l]}$   
3    $b^{[l]} = b^{[l]} - \eta * db^{[l]}$ 
```

OUTPUT NEURONS

- The choice of how to represent the output determines the form of the cross-entropy function.
- The feedforward network provides a set of hidden features $h = f(x; \theta)$.
- The role of the output layer is to provide some additional transformation.
- Types of output transformation
 - 1 Linear
 - 2 Sigmoid
 - 3 Softmax

LOSS FUNCTIONS

- Loss function compares the actual output from data set, in case of supervised learning, to the predicted output of the output layer.
- Based on the task, the loss functions differ.

For Regression	For Classification
Mean Squared Error (MSE)	Binary Cross Entropy (Sigmoid)
Mean Absolute Error (MAE)	Categorical Cross Entropy (Softmax)
Huber loss (Smooth MAE)	KL Divergence (Relative Entropy)
Log Hyperbolic Cosine (log cosh)	Exponential Loss
Quantile Loss	Hinge Loss
Cosine Similarity	

Further Reading

- 1 Dive into Deep Learning (T1)



Thank You!