FPT UNIVERSITY

Basic Neural Network

Basic Neural Network

Learning Objectives:

- Build a logistic regression model structured as a shallow neural network
- Gradient calculation, and optimization implementation (gradient descent)
- Implement computationally efficient and highly vectorized versions of models
- Compute derivatives for logistic regression, using a backpropagation mindset
- Use Numpy functions and Numpy matrix/vector operations
- Implement vectorization across multiple training examples
- Explain the concept of broadcasting

Basic Neural Network

1. Binary Classification
2. Logistic Regression
3. Logistic Regression cost function
4. Gradient Descent
5. Derivatives
6. Computation Graph
7. Derivatives with a Computation Graph
8. Logistic Regression Gradient descent
9. Gradient descent on m examples
10. Vectorization
11. Vectorizing Logistic Regression
12. Vectorizing Logistic Regression's Gradient Output
13. Broadcasting in Python
14. Explanation of logistic regression cost function (Optional)

FPT UNIVERSITY

Basics of Neural Network Programming

Binary Classification

Binary Classification

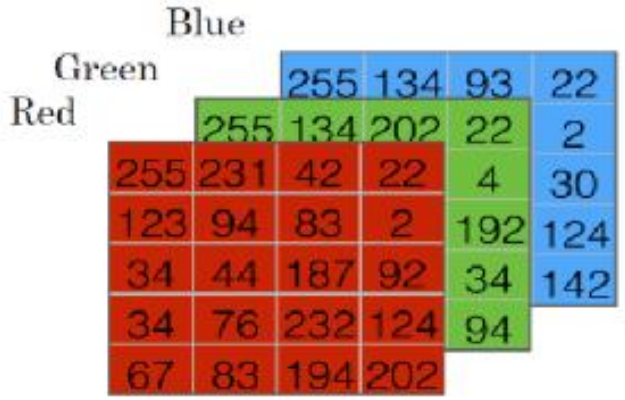
- Binary classification is a supervised learning algorithm that classify input data into one of two categories or classes. The output of the algorithm is a binary decision, such as "yes" or "no", "true" or "false", or 1 or 0.
- The goal is to learn a model that can correctly classify new input data based on patterns or features extracted from the input data.
- Examples: predict whether an email is spam or not spam, or to recognize whether an image contains a cat or not.
- Logistic regression is an algorithm used to solve the binary classification problems.
- Consider example in the next slide.

Binary Classification

$\longrightarrow$ y = 1 (cat) vs 0 (non cat)

An image is store in the computer in three separate matrices corresponding to the Red, Green, and Blue color channels of the image. The three matrices have the same size as the image, for example, the resolution of the cat image is 64 pixels X 64 pixels, the three matrices (RGB) are 64 X 64 each.

To create a feature vector, x, the pixel intensity values will be "unroll" or "reshape" for each color. The dimension of the input feature vector x is $n_x = 64\ x\ 64\ x\ 3 = 12\ 288$.



$$x = \begin{bmatrix} 255 \\ 231 \\ 42 \\ \vdots \\ 255 \\ 134 \\ 202 \\ \vdots \\ 255 \\ 134 \\ 93 \\ \vdots \end{bmatrix} \begin{matrix} \text{red} \\ \\ \\ \\ \text{green} \\ \\ \\ \\ \text{blue} \\ \\ \\ \\ \end{matrix}$$

Notation

FPT Education
FPT UNIVERSITY

1 Neural Networks Notations.

General comments:

· superscript (i) will denote the i^{th} training example while superscript [l] will denote the l^{th} layer

Sizes:

·m : number of examples in the dataset

·n_x : input size

·n_y : output size (or number of classes)

·$n_h^{[l]}$: number of hidden units of the l^{th} layer

In a for loop, it is possible to denote $n_x = n_h^{[0]}$ and $n_y = n_h{}^{[\text{number of layers } +1]}$.

·L : number of layers in the network.

Objects:

·$X \in \mathbb{R}^{n_x \times m}$ is the input matrix

·$x^{(i)} \in \mathbb{R}^{n_x}$ is the i^{th}example represented as a column vector

·$Y \in \mathbb{R}^{n_y \times m}$ is the label matrix

·$y^{(i)} \in \mathbb{R}^{n_y}$ is the output label for the i^{th} example

·$W^{[l]} \in \mathbb{R}^{\text{number of units in next layer} \times \text{number of units in the previous layer}}$ is the weight matrix,superscript [l] indicates the layer

·$b^{[l]} \in \mathbb{R}^{\text{number of units in next layer}}$ is the bias vector in the l^{th} layer

·$\hat{y} \in \mathbb{R}^{n_y}$ is the predicted output vector. It can also be denoted $a^{[L]}$ where L is the number of layers in the network.

FPT UNIVERSITY

Basics of Neural Network Programming

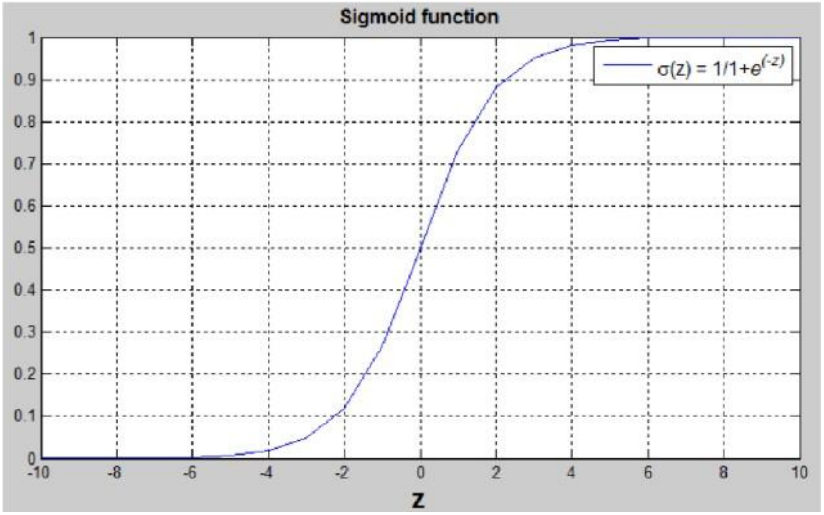
Logistic Regression

Logistic Regression

$$Given\ x\ ,\ \hat{y} = P(y = 1|x), \text{where } 0 \leq \hat{y} \leq 1$$

The parameters used in Logistic regression are:

- The input features vector: $x \in \mathbb{R}^{n_x}$, where n_x is the number of features
- The training label: $y \in 0,1$
- The weights: $w \in \mathbb{R}^{n_x}$, where n_x is the number of features
- The threshold: $b \in \mathbb{R}$
- The output: $\hat{y} = \sigma(w^T x + b)$
- Sigmoid function: s = $\sigma(w^T x + b)$ = $\sigma(z)$= $\frac{1}{1+e^{-z}}$



$(w^T x + b)$ is a linear function $(ax + b)$, but since we are looking for a probability constraint between [0,1], the sigmoid function is used. The function is bounded between [0,1] as shown in the graph above.

Some observations from the graph:

- If z is a large positive number, then $\sigma(z)$ = 1
- If z is small or large negative number, then $\sigma(z)$ = 0
- If $z = 0$, then $\sigma(z)$ = 0.5

FPT UNIVERSITY

Logistic Regression cost function

Logistic Regression cost function

FPT Education
FPT UNIVERSITY

To train the parameters w and b, we need to define a cost function.

Recap:

$\hat{y}^{(i)} = \sigma(w^T x^{(i)} + b)$, where $\sigma(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}}$

$x^{(i)}$ the i-th training example

$Given \{(x^{(1)}, y^{(1)}), \cdots, (x^{(m)}, y^{(m)})\},\ we\ want\ \hat{y}^{(i)} \approx y^{(i)}$

Loss (error) function:

The loss function measures the discrepancy between the prediction ($\hat{y}^{(i)}$) and the desired output ($y^{(i)}$). In other words, the loss function computes the error for a single training example.

$$L(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{2}(\hat{y}^{(i)} - y^{(i)})^2$$

$$L(\hat{y}^{(i)}, y^{(i)}) = -(y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

- If $y^{(i)} = 1$: $L(\hat{y}^{(i)}, y^{(i)}) = -\log(\hat{y}^{(i)})$ where $\log(\hat{y}^{(i)})$ and $\hat{y}^{(i)}$ should be close to 1
- If $y^{(i)} = 0$: $L(\hat{y}^{(i)}, y^{(i)}) = -\log(1 - \hat{y}^{(i)})$ where $\log(1 - \hat{y}^{(i)})$ and $\hat{y}^{(i)}$ should be close to 0

Cost function

The cost function is the average of the loss function of the entire training set. We are going to find the parameters $w\ and\ b$ that minimize the overall cost function.

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

FPT UNIVERSITY

Basics of Neural Network Programming

Gradient Descent

Gradient Descent

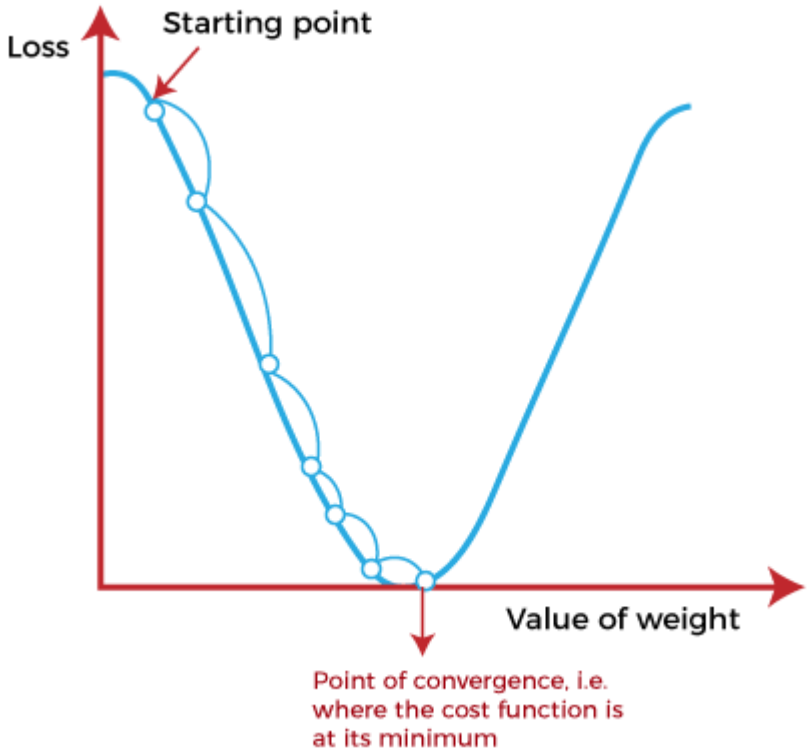
- Gradient descent is an optimization algorithm commonly used in machine learning and deep learning to minimize the error or loss function of a model.
- The goal is to find the optimal set of parameters (weights and biases) for a given model that minimizes the difference between the predicted output and the actual target values.
- The ideal is to iteratively update the model's parameters in the direction of the steepest descent of the loss function.
- Some variations of gradient descent: batch gradient descent, stochastic gradient descent, and mini-batch gradient descent.

Gradient Descent

- How the gradient descent algorithm works:

1. Initialization: Start with an initial set of parameter values randomly or using some predefined values.
2. Forward Pass: Feed the input data into the model to obtain the predicted output.
3. Loss Calculation: Calculate the difference (error) between the predicted output and the actual target values using the chosen loss function.
4. Gradient Calculation: Compute the gradient (partial derivatives) of the loss function with respect to each model parameter. This step tells us the direction and magnitude in which the parameters should be adjusted to reduce the loss.
5. Parameter Update: Update the model parameters by moving them in the opposite direction of the gradient (descending the loss function). The size of the update is controlled by a learning rate, which determines the step size in the parameter space.
6. Repeat: Repeat steps 2-5 for a predefined number of iterations or until the loss reaches an acceptable level.

Gradient Descent



$$X = X - lr * \frac{dJ(X)}{dX}$$

Where

X is input

J(X) is cost function (loss function)

lr is learning rate

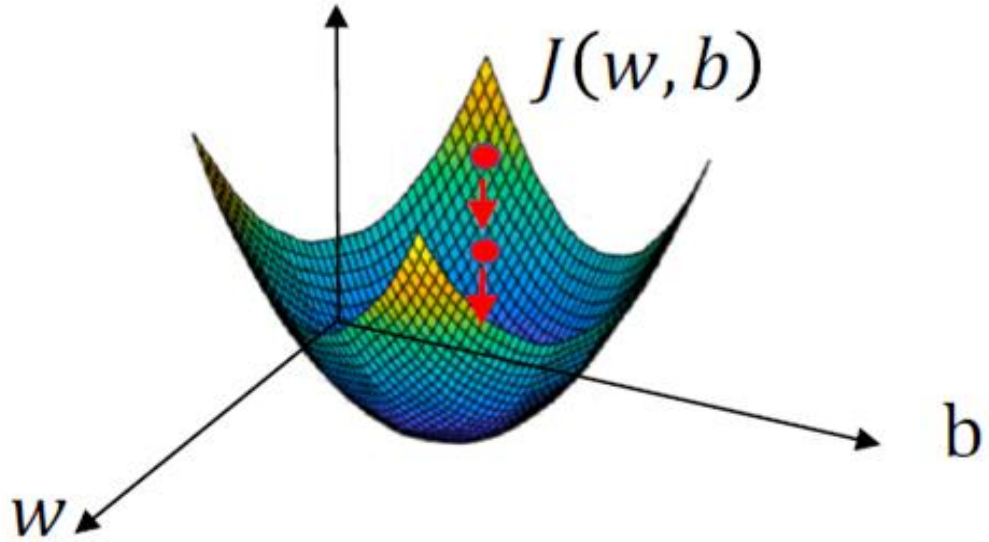
Gradient Descent

FPT Education
FPT UNIVERSITY

Recap: $\hat{y} = \sigma(w^T x + b), \ \sigma(z) = \frac{1}{1+e^{-z}}$

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

Want to find w, b that minimize $J(w, b)$



FPT UNIVERSITY

Basics of Neural Network Programming

Derivatives

Derivatives

- In mathematics, the derivative is a fundamental concept in calculus that measures how a function changes as its input changes.
- It provides the rate of change or the slope of a function at any specific point. Given a function f(x), the derivative of f with respect to x is denoted as f′(x) or $\frac{df}{dx}$
- Graphically, the derivative represents the slope of the tangent line to the curve of the function at a particular point.
  - When the derivative is positive, it indicates that the function is increasing at that point; when it is negative, it shows the function is decreasing.
  - A derivative of zero means that the function has a critical point, which could be a local minimum or maximum.

Derivatives

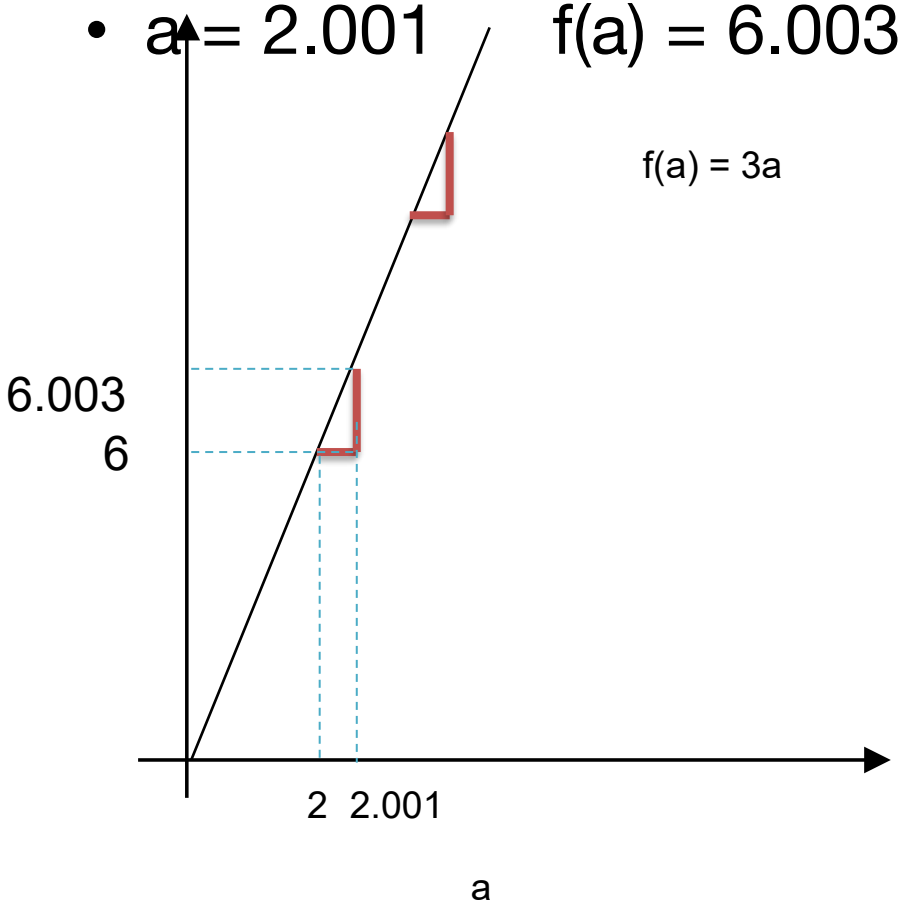
- The formal definition of the derivative of a function f(x) with respect to x is given by:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- This definition describes the change in the function f(x) as the difference in f(x) for a small change in x (given by h) approaches zero.
- Consider the example in the next slide.

Intuition about derivatives

- a = 2 f(a) = 6
- a = 2.001 f(a) = 6.003



slope derivative of f(a) at a = 2 is 3

a = 5 f(a) = 15
a = 5.001 f(a) = 15.003

slope derivative of f(a) at a = 5 is also 3

$$\frac{df(a)}{da} = 3 = \frac{d}{da} f(a)$$

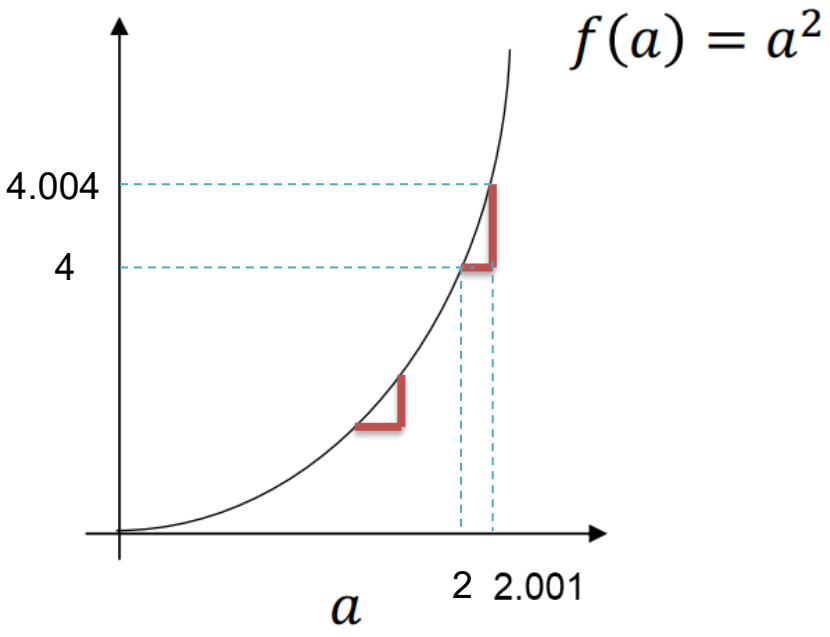
More derivatives examples

More derivatives examples

- A derivative is the slope of a function at a particular point. The slope can be different at different points on the function, depending on the shape of the function.

- For example, the function $f(a) = a^2$ has a derivative of $2a$.
  - This means that if you nudge a to the right by a small amount, say 0.001, f(a) should go up by approximately 2a times 0.001.
  - So if a = 2, then the derivative is 4, and if you nudge a to 2.001, f(a) should go up by about 0.008

Intuition about derivatives



a = 2 f(a) = 4
a = 2.001 f(a) = 4.004

slope derivative of f(a) at a = 2 is 4

a = 5 f(a) = 25
a = 5.001 f(a) = 25.010

$$\frac{d}{da} f(a) = 10 \; when \; a = 5$$

$$\frac{df(a)}{da} = \frac{d}{da} f(a) = 2a$$

More derivatives examples

- Similarly, the function $f(a) = a^3$ has a derivative of $3a^2$, and the function $f(a) = \log(a)$ has a derivative of $1/a$.
- Derivatives are defined using infinitesimally small nudges to a, rather than small but finite nudges like 0.001.
- This means that the values of f(a) given by the derivative formula may be only approximate if you use a small but finite nudge like 0.001.

More derivative examples

f(a) = a^3 $\quad \frac{df(a)}{da} = 3a^2$ a = 2 f(a) = 8
a = 2.001 f(a) = 8.012

f(a) = ln(a) $\quad \frac{df(a)}{da} = \frac{1}{a}$ a = 2 f(a) = 0.69315
a = 2.001 f(a) = 0. 69365

FPT UNIVERSITY

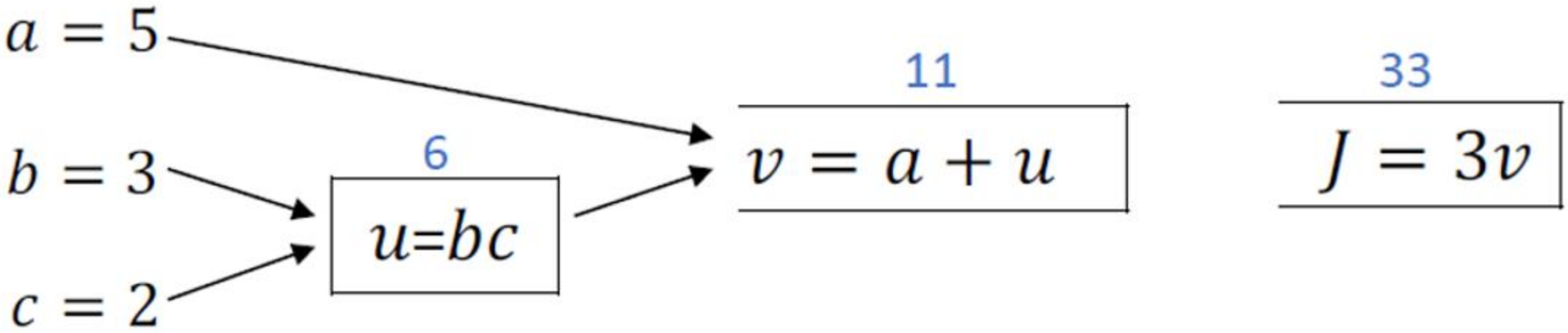
Basics of Neural Network Programming

Computation Graph

Computation Graph

- Computation graph is a way to visualize the computations of a function as a graph. The computation graph is helpful for computing derivatives or gradients of a function efficiently.
- The 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 compute derivatives. Computation graphs explain why it is organized this way.
- Example: $J = 3(a + bc)$.

Computation Graph



Computation Graph

- The computation graph is useful when there is a distinguished output variable, such as J in this case, that we want to optimize.
- The computation of J involves a left-to-right pass through the graph, which computes the value of J.
- In order to compute the derivatives of J, a right-to-left pass is needed, which computes the gradients or derivatives of the variables with respect to J.
- In summary, the computation graph provides a way to organize the computations of a function, and it helps to efficiently compute derivatives or gradients of the function.

FPT UNIVERSITY

Basics of Neural Network Programming

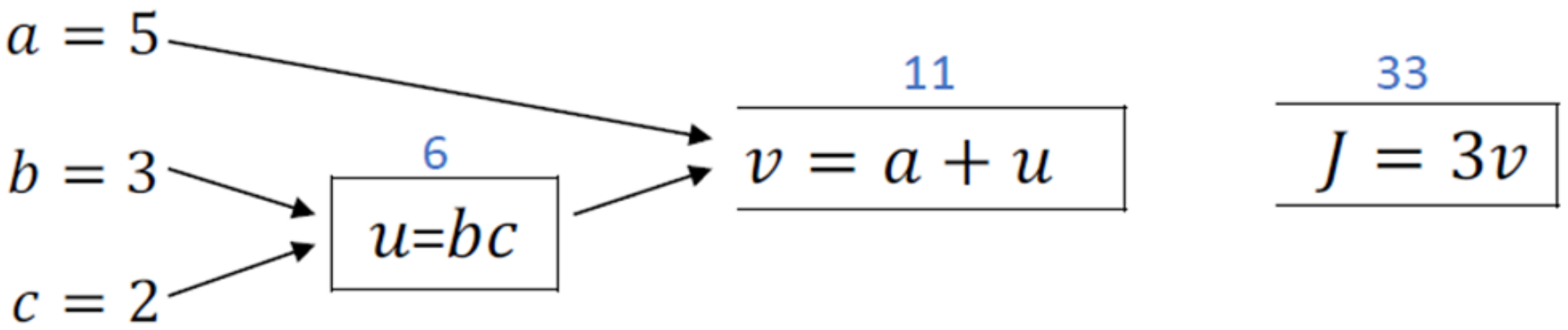
Derivatives with a Computation Graph

Derivatives with a Computation Graph

FPT Education
FPT UNIVERSITY

- Consider example of how to use a computation graph to compute a function J and then how to use it to figure out derivative calculations for J.
- The computation graph organizes the computation with a left-to-right pass and a right-to-left pass. The left-to-right pass computes the value of the final output variable, J, while the right-to-left pass computes derivatives.
- Computing the derivative of J with respect to v and shows how to use the chain rule to compute the derivatives of intermediate variables such as a, u, b, and c.

Computing derivatives



$$\frac{dj}{dv}=3\ ;\frac{dj}{da}=3=\frac{dj}{dv}*\frac{dv}{da}\ ;\frac{dv}{da}=1$$

$$\frac{dj}{du}=\frac{dj}{dv}*\frac{dv}{du}=3\ ;\quad \frac{dj}{db}=\frac{dj}{du}*\frac{du}{db}=6$$

$$\frac{dj}{dc}=\frac{dj}{du}*\frac{du}{dc}=9$$

- Given J = u + v - w, where u = a*b, v = a*c, and w = b+c, simplify J, calculate the first derivative of J.

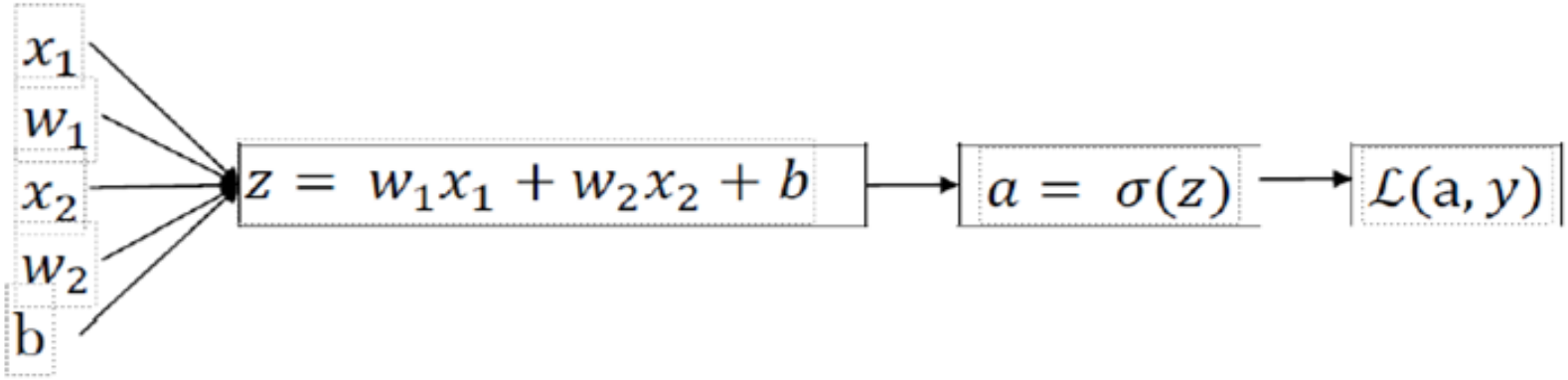
FPT UNIVERSITY

Basics of Neural Network Programming

Logistic Regression Gradient Descent

Logistic regression recap

$$z = w^T x + b$$
$$\hat{y} = a = \sigma(z)$$
$$\mathcal{L}(a, y) = -(y \log(a) + (1 - y) \log(1 - a))$$

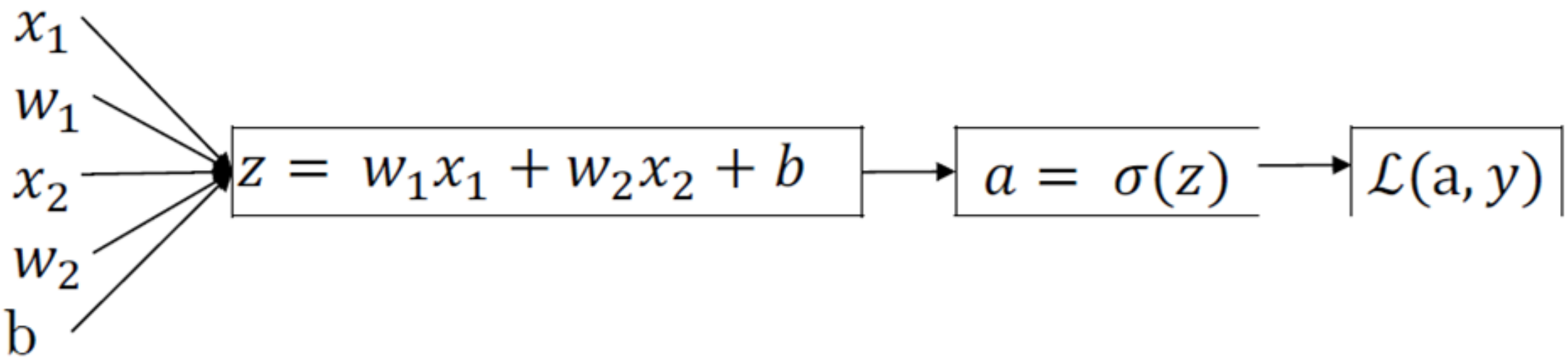


dz = df/dz = dL(a,y)/dz

= a - y

= dL/da * da/dz

Logistic regression derivatives



FPT UNIVERSITY

Gradient descent on m examples

Gradient descent on m examples

- To compute the derivatives of the cost function J with respect to each parameter w1, w2, and b, use a for loop over the m training examples (the next slide)
- There are two weaknesses with this calculation:
  - Firstly, two for loops are required, one over the m training examples and another over the n features.
  - Secondly, explicit for loops make the algorithm run less efficiently, which is problematic when working with larger datasets. Use the concept of vectorization, which will be discussed in the next section, to address these issues.

Logistic regression on m examples

FPT Education
FPT UNIVERSITY

$J = 0, dw_1 = 0, dw_2 = 0, db = 0$

for i = 1 to m:

$z^{(i)} = w^T x^i + b$

$a^{(i)} = \sigma(z^{(i)})$

$J \mathrel{+}= -[y^{(i)} \log\hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}]$

$dz^{(i)} = a^{(i)} - y^{(i)}$

$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$

$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$

$db \mathrel{+}= dz^{(i)}$

$J = J/m, dw_1 = dw_1/m, dw_2 = dw_2/m, db = db/m$

$$w_1 := w_1 - \alpha\, dw_1$$
$$w_2 := w_2 - \alpha\, dw_2$$
$$b := b - \alpha\, db$$

FPT UNIVERSITY

Basics of Neural Network Programming

Vectorization

Vectorization

- Vectorization is the process of getting rid of explicit for loops in code to make it run faster, especially in the era of deep learning where large datasets are common.
- The goal is to perform operations on vectors and matrices directly instead of using loops.
  - For example, in logistic regression, the equation
  - $Z = W^T X + b$ can be vectorized by computing
  - Z = np.dot(W, X) + b.
  - This eliminates the need for loops and runs much faster.
- In a Jupyter notebook, the np library can be used to create and manipulate arrays.

Vectorization

- A vectorization demo was performed by creating two million-dimensional arrays and using np.dot() to calculate the dot product.
- A non-vectorized version was then implemented using a for loop to compute the dot product.
- Vectorized version took around 1.5 milliseconds to run, while the non-vectorized version took about 500 milliseconds, or 300 times longer.
- It is important to remember to vectorize code whenever possible to make it run faster. Built-in functions like np.dot() make use of parallelism instructions that allow CPUs and GPUs to execute calculations much faster.

FPT UNIVERSITY

Basics of Neural Network Programming

More vectorization examples

Neural network programming guideline

Whenever possible, avoid explicit for-loops.

```
U = A*v
U = np.dot(A,v)
```

Vectors and matrix valued functions

- Say you need to apply the exponential operation on every element of a matrix/vector.

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}$$

```
u = np.zeros((n,1))
for i in range(n):
    u[i]=math.exp(v[i])
```

```
import numpy as np
u = np.exp(v)
np.log(v)
np.abs(v)
np.np.maximum(v,0)
v**2
```

FPT UNIVERSITY

Basics of Neural Network Programming

Vectorizing Logistic Regression

Vectorizing Logistic Regression

- Consider the four propagation steps of logistic regression, which include computing the activation values for each training example.
- So, if you have M training examples, you need to do this M times.

$$
\begin{aligned}
z^{(1)} &= w^T x^{(1)} + b \\
a^{(1)} &= \sigma(z^{(1)}) \\
z^{(2)} &= w^T x^{(2)} + b \\
a^{(2)} &= \sigma(z^{(2)}) \\
z^{(3)} &= w^T x^{(3)} + b \\
a^{(3)} &= \sigma(z^{(3)})
\end{aligned}
$$

Vectorizing Logistic Regression

- Use a (Nx by M) matrix.
- Use this matrix to compute Z=Z(1) Z(2) Z(3) ... Z(n) with one line of code.
- The matrix Z is a 1 by M matrix that's really a row vector.
- Using matrix multiplication $Z=W^TX+b$ can be computed to obtain Z in one step.
- The python command is : z = np.dot(w.T, X) + B
- Here W = (NX by 1), X is (NX by M) and b is (1 by M), multiplication and addition gives Z (1 by M).

FPT UNIVERSITY

Basics of Neural Network Programming

Vectorizing Logistic Regression's Gradient Output

Vectorizing Logistic regression

FPT Education
FPT UNIVERSITY

$J = 0, dw_1 = 0, dw_2 = 0, db = 0$

for i = 1 to m:

$z^{(i)} = w^T x^i + b$

$a^{(i)} = \sigma(z^{(i)})$

$J \mathrel{+}= -[y^{(i)} \log\hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}]$

$dz^{(i)} = a^{(i)} - y^{(i)}$

$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$

$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$

$db \mathrel{+}= dz^{(i)}$

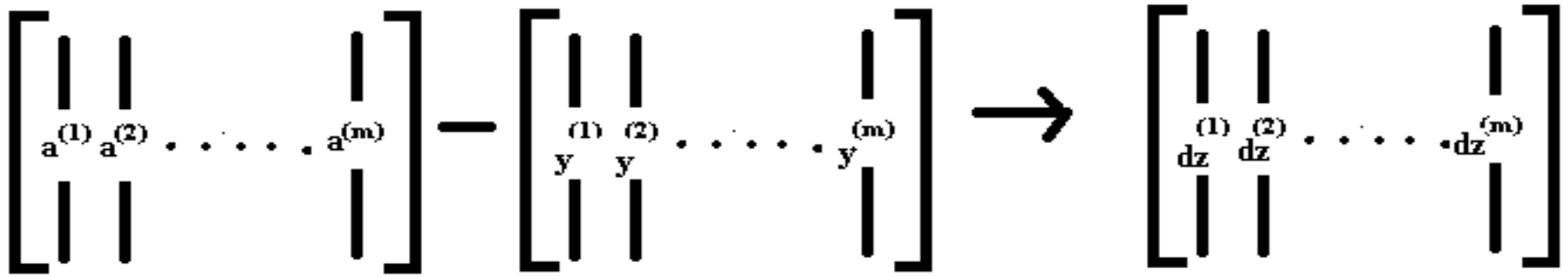
$J = J/m, dw_1 = dw_1/m, dw_2 = dw_2/m, db = db/m$

Vectorizing Logistic Regression

- Vectorization also performs the gradient computations for all M training samples. To derive a very efficient implementation of logistic regression.
- For the gradient computation , the first step is to compute dz(1) for the first example, which could be a(1)–y(1) and then dz(2) = a(2)–y(2) and so on. And so on for all M training examples.

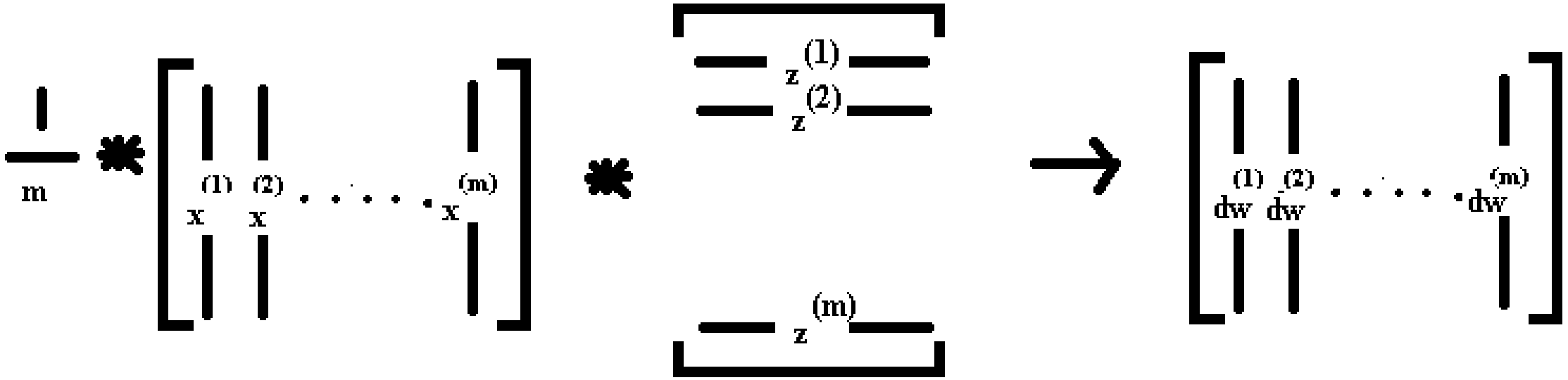
Vectorizing Logistic Regression

- Lets define a new variable , dZ is going to be dz(1) , dz(2),..., dz(m). Again, all the lowercase z variables stacked horizontally.
- So, this would be 1 by m matrix or alternatively a 'm' dimensional row vector.
- Previously , we'd already figured out how to compute A and Y as shown below and you can see for yourself that dz can be computed as just A minus Y because it's going to be equal to a(1) – y(1), a(2) – y(2), and so on.
- So, with just one line of code , you can compute all of this at the same time. By a simple matrix subtraction as shown below.



Vectorizing Logistic Regression

- To compute dw , we initialize dw to zero to a vector of zeroes.
- Then we still have to loop over examples where we have dw + = x(1) * dz(1), for the first training example and so on.



Python: dw = 1/m * np.dot(X,(Z).T)

Vectorizing Logistic Regression

FPT Education
FPT UNIVERSITY

- Similarly for the vectorize implementation of db was doing is basically summing up, all of these dz and then dividing by m. This can be done using just one line in python as: db = 1/m * np.sum(dz)
- And so the gradient descent update then would be you know W gets updated as w minus the learning rate times dw which was just computed above and B is update as B minus the learning rate times db.
- We should get rid of explicit full loops whenever you can but if you want to implement multiple adjuration as a gradient descent then you still need a full loop over the number of iterations. So, if you want to have a thousand deliberations of gradient descent, you might still need a full loop over the iteration number.
- There is an outermost full loop like that then I don't think there is any way to get rid of that full loop.
- A technique in python which makes our code easy for implementation is called broadcasting.

FPT UNIVERSITY

Basics of Neural Network Programming

Broadcasting in Python

Broadcasting in Python

- Broadcasting in Python is a technique that allows you to perform mathematical operations between arrays or matrices of different sizes without explicitly iterating over them using a for-loop.
- The principle of broadcasting is that when you perform an operation between arrays or matrices of different sizes, NumPy will automatically copy the smaller array or matrix to match the size of the larger one, and then perform the operation element-wise.
- In the example in the next slide, a three by four matrix representing the number of calories from carbohydrates, proteins, and fats in 100 grams of four different foods was used. The goal was to calculate the percentage of calories from carbs, proteins, and fats for each of the four foods.

Broadcasting example

- Calories from Carbs, Proteins, Fats in 100g of different foods:

	Apples	Beef	Eggs	Potatoes
Carb	56.0	0.0	4.4	6.8
Protein	1.2	104.0	52.0	8.0
Fat	1.8	135.0	99.0	0.9

```
cal = A.sum(axis = 0)
percentage = 100*A/(cal.reshape(1,4))
```

Broadcasting in Python

- The code used to achieve this involved two lines.
  - The first line used the sum function to sum down the columns of the matrix, and the second line divided each element of the matrix by its corresponding sum.
  - The reshape function was used to ensure that the dimensions of the arrays were compatible for the division operation.

Broadcasting example

$$\begin{bmatrix}1\\2\\3\\4\end{bmatrix} + 100$$

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix} + [100 \quad 200 \quad 300]$$

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix} + \begin{bmatrix}100\\200\end{bmatrix}$$

```
a = np.array([[1], [2], [3],[4]])

b = np.array([100])
print(a)
print(b)
print(a+b)
```

```
[[1]
 [2]
 [3]
 [4]]
[100]
[[101]
 [102]
 [103]
 [104]]
```

```
c = np.array([[1, 2, 3],[4,5,6]])
d = np.array([100,200,300])
print( c)
print( d)
print(c+d)
```

```
[[1 2 3]
 [4 5 6]]
[100 200 300]
[[101 202 303]
 [104 205 306]]
```

```
e = np.array([[1, 2, 3],[4,5,6]])
f = np.array([[100],[200]])
print(e)
print(f)
print(e+f)
```

```
[[1 2 3]
 [4 5 6]]
[[100]
 [200]]
[[101 102 103]
 [204 205 206]]
```

Broadcasting in Python

- Broadcasting in Python is a powerful tool that can greatly simplify your code and make it run faster. It can be used for a wide range of operations, including addition, subtraction, multiplication, and division.
- There are some tips and tricks that can help reduce the number of bugs in your Python code. These include using descriptive variable names, writing modular code, commenting your code, testing your code frequently, and using version control. These tips can help you write cleaner, more efficient code that is easier to debug and maintain.

FPT UNIVERSITY

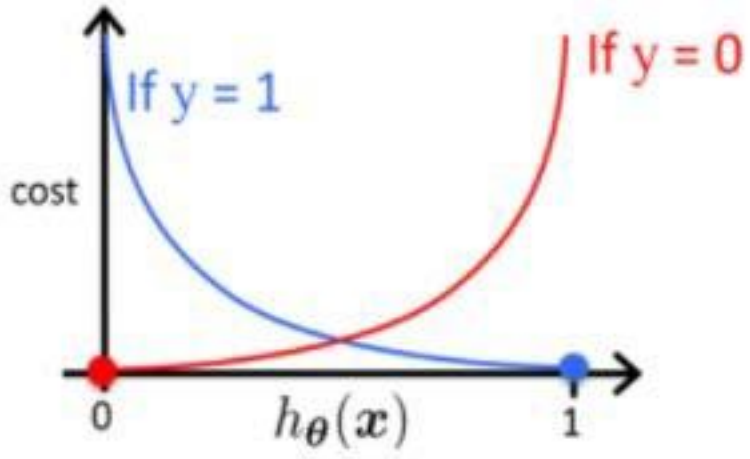
Explanation of logistic regression cost function (Optional)

Logistic regression cost function

Given the cost function of the Logistic regression as following, explain the intuition or rationale of it.

$$\text{cost}\,(h_{\boldsymbol{\theta}}(\boldsymbol{x}), y) = \begin{cases} -\log(h_{\boldsymbol{\theta}}(\boldsymbol{x})) & \text{if } y = 1 \\ -\log(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x})) & \text{if } y = 0 \end{cases}$$

Hint:



Logistic regression cost function

Instead of Mean Squared Error, we use a cost function called Cross-Entropy, also known as Log Loss. Cross-entropy loss can be divided into two separate cost functions: one for $y = 1$ and one for $y = 0$.

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$\text{Cost}(h_\theta(x), y) = -\log(h_\theta(x)) \qquad \text{if y} = 1$$

$$\text{Cost}(h_\theta(x), y) = -\log(1 - h_\theta(x)) \qquad \text{if y} = 0$$

The benefits of taking the logarithm reveal themselves when you look at the cost function graphs for y=1 and y=0. These smooth monotonic functions [7] (always increasing or always decreasing) make it easy to calculate the gradient and minimize cost. Image from Andrew Ng's slides on logistic regression [1].

Cost on m examples

$$J(\theta) = \frac{-1}{m}\sum_{i=1}^{m} y^{(i)} log\left(\sigma\left(\theta^T x^{(i)}\right)\right) + \left(1 - y^{(i)}\right) log\left(1 - \sigma\left(\theta^T x^{(i)}\right)\right)$$

$$\frac{\partial(J(\theta))}{\partial(\theta_j)} = \frac{-1}{m}\sum_{i=1}^{m}\left(y^{(i)} - \theta^T x^{(i)}\right) x_j^{(i)}$$

m = *Number of examples*
i = *An example*
j = *Feature j*

$$\theta_j := \theta_j - \alpha\frac{\partial(J(\theta))}{\partial\theta_j}$$

α = *Learning rate*

How to deal with non-linear decision boundaries

If you have a dataset with a non-linear decision boundary, it is still possible to use logistic regression.
The way it works is practically the same as polynomial regression where you add polynomial terms.

So, you would modify the hypothesis equation to be as follows:

$$\hat{p} = \sigma\left(\theta_0 x_1 + \theta_1 x_2 + \theta_2 x_1^2 + \theta_3 x_2^2 + \theta_4 x_1 x_2\right)$$

$$\hat{y} = 1 \; if \; \hat{p} \geq 0.5$$
$$0 \; if \; \hat{p} < 0.5$$

Summarization

- Logistic regression, a binary classification algorithm, can be viewed as a single-layer neural network, employing input features, weights, a bias term, and a sigmoid activation for predictions.
- For efficiency, use matrix operations and vectorized computations, minimizing loops and leveraging library functions.
- Derivatives are calculated for gradient descent.
- Vectorization processes multiple training examples concurrently, using matrices for parallel operations.
- Broadcasting, facilitating element-wise operations between arrays of different shapes, is employed in libraries like NumPy.

Question

FPT Education
FPT UNIVERSITY

1. What does a neuron compute, considering its linear function and activation (e.g., a = g(Wx + b))?
2. What is the formula for "Logistic Loss" (as seen in lecture; refer to hint if needed)?
3. How do you reshape a (32,32,3) image array img into a column vector x?
4. If a.shape=(2,3) and b.shape=(2,1), what is the shape of c = a + b due to broadcasting?
5. If a.shape=(4,3) and b.shape=(3,2), what is the result/error of c = a * b (element-wise)?
6. If you have n_x input features/example and m examples, what's the dimension of matrix X?
7. If a.shape=(12288,150) and b.shape=(150,45), what is the shape of c = np.dot(a, b)?
8. How do you vectorize c[i][j] = a[i][j] + b[j] if a.shape=(3,4) and b.shape=(3,1)?
9. If a.shape=(3,3) and b.shape=(3,1), what is c = a * b after broadcasting and element-wise product?
10. Given J = u + v - w, where u = a*b, v = a*c, and w = b+c, simplify J, calculate the first derivative of J.