# CS 230: Deep Learning

Hargen Zheng

December 21, 2023

# Contents

# Part I

# Neural Networks and Deep Learning

# Chapter 1

# Introduction

Why it matters?

- AI is the new Electricity.
- Electricity had once transformed countless industries: transportation, manufacturing, healthcare, communications, and more.
- AI will now bring about an equally big transformation.

Courses in this sequence (Specialization):

1. Neural Networks and Deep Learning
2. Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization
3. Structuring your Machine Learning project
4. Convolutional Neural Networks (CNNs)
5. Natural Language Processing: Building sequence models (RNNs, LSTM)

## 1.1 What is a Neural Network?

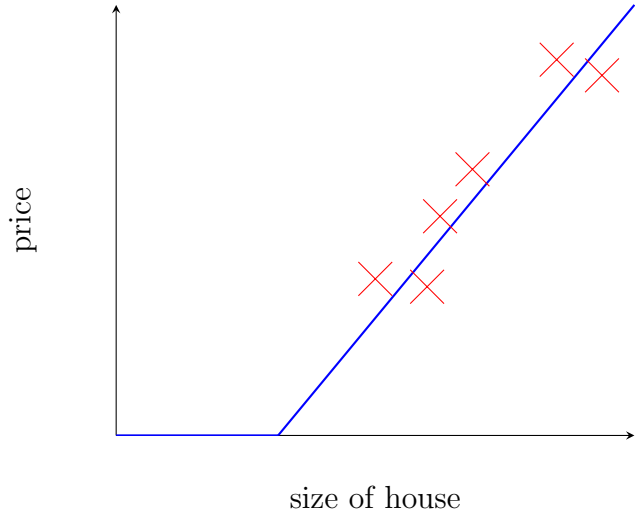


Figure 1.1: Housing Price Prediction

In the neural network literature, this function appears by a lot. This function is called a ReLU function, which stands for rectified linear units.

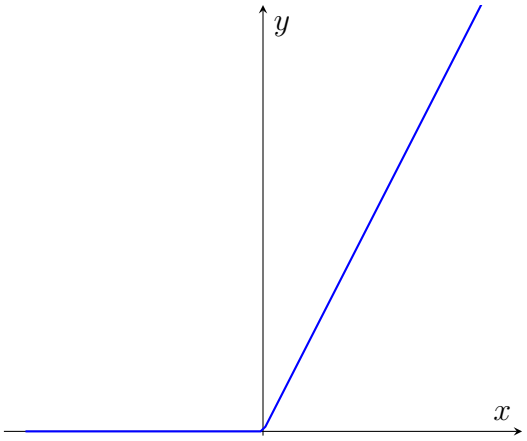


Figure 1.2: ReLU function: $\max\{0, x\}$

With the size of houses in square feet or square meters and the price of the house, we can fit a function to predict the price of a house as a function of its size.

This is the simplest neural network. We have the size of a house as input $x$, which goes into a node (a single "neuron"), and outputs the price $y$.

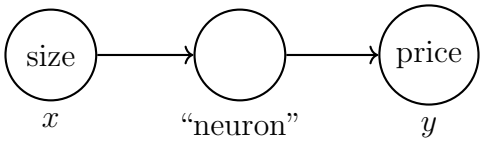


Figure 1.3: Simple Neural Network of Housing Example

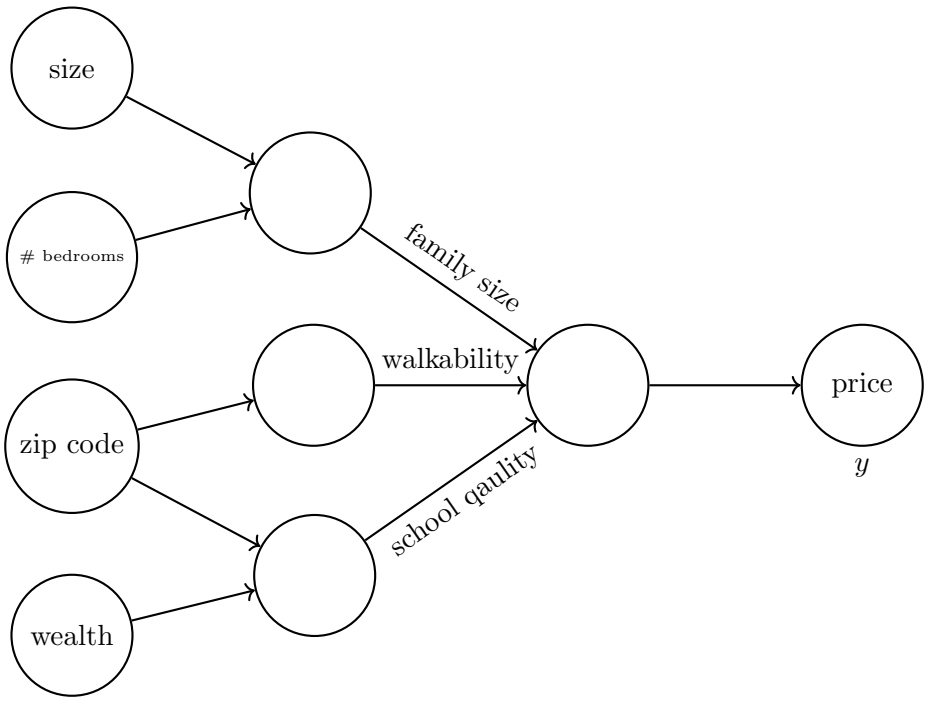


Figure 1.4: Slightly Larger Neural Network

Suppose that, instead of predicting the price of a house just from the size, we also have other features, such as the number of bedrooms, zip code, and wealth. The number of bedrooms determines whether or not a house can fit one's family size; Zip code tells walkability; and zip code and wealth tells how good the school quality is. Each of the circle could be ReLU or other nonlinear function.

We need to give the neural network the input $x$ and the output $y$ for a number of examples in the training set and, for all the things in the middle, neural network will figure out by itself.

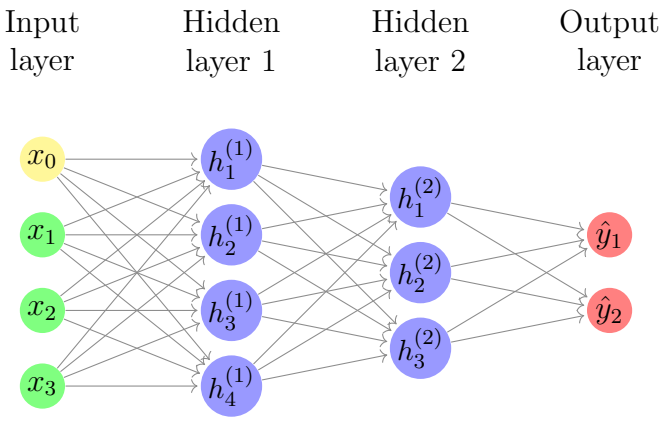


Figure 1.5: General Picture of Neural Network

## 1.2 Supervised Learning with Neural Networks

Supervised Learning Examples.

| Input($x$) | Output($y$) | Application | Type of NN |
|---|---|---|---|
| Home features | Price | Real Estate | Standard |
| Ad, user info | Click on ad? (0/1) | Online Advertising | Standard |
| Image | Object $(1, \ldots, 1000)$ | Photo tagging | CNN |
| Audio | Text transcript | Speech recognition | RNN |
| English | Chinese | Machine translation | RNN |
| Image, Radar info | Position of other cars | Autonomous driving | Custom |

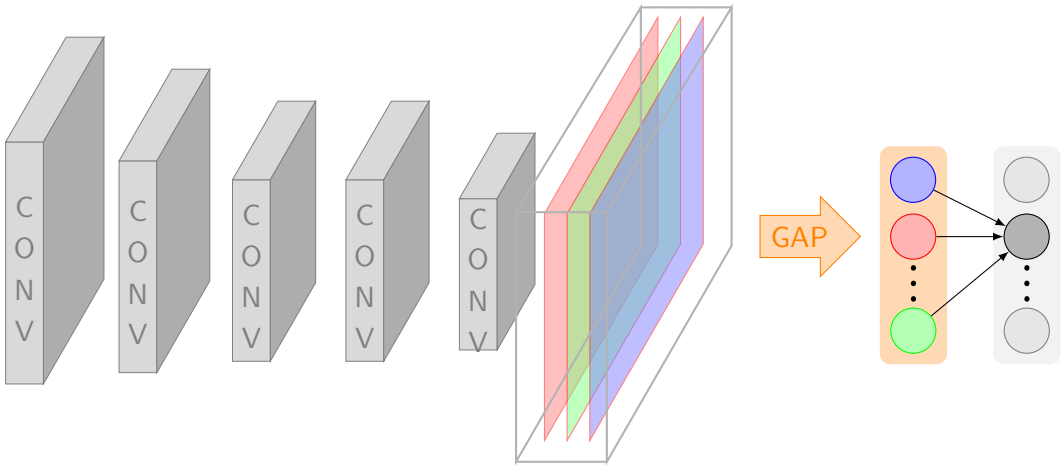


Figure 1.6: Convolutional Neural Network

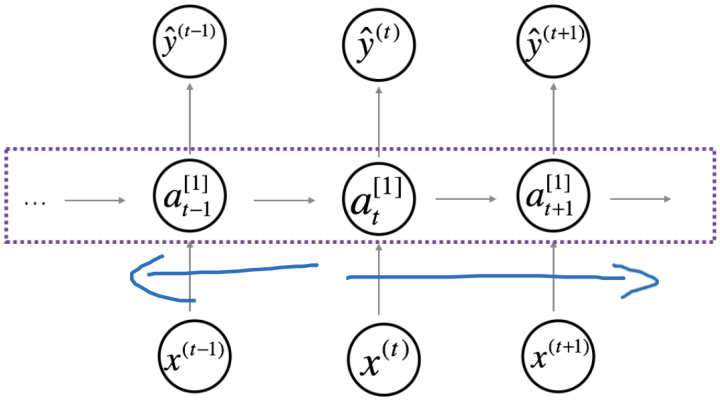


Figure 1.7: Recurrent Neural Network

There are two types of data: Structured Data and Unstructured Data. We could have the following examples for each.

| Size | #bedrooms | ··· | Price(1000$s) |
|---|---|---|---|
| 2104 | 3 | | 400 |
| 1600 | 3 | | 330 |
| 2400 | 3 | | 369 |
| ⋮ | ⋮ | | ⋮ |
| 3000 | 4 | | 540 |

| User Age | Ad ID | ··· | Click |
|---|---|---|---|
| 41 | 93242 | | 1 |
| 80 | 93287 | | 0 |
| 18 | 87312 | | 1 |
| ⋮ | ⋮ | | ⋮ |
| 27 | 71244 | | 1 |

Table 1.1: Structured Data

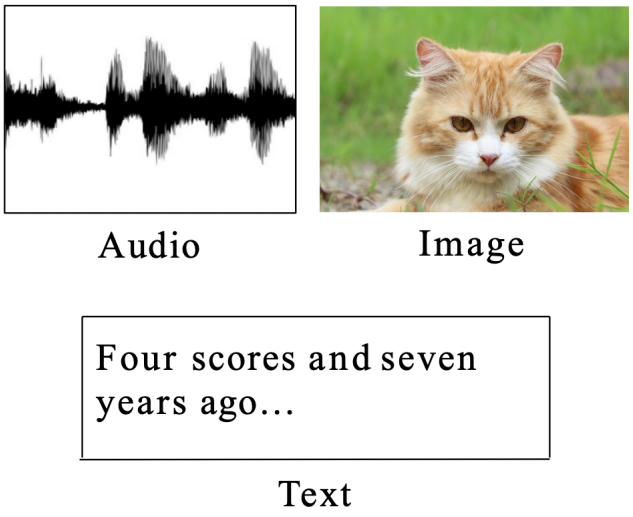


Figure 1.8: Unstructured Data

Historically, it has been much harder for computers to make sense of unstructured data compared to structured data. Thanks to the neural networks, computers are better at interpreting unstructured data as well, compared to just a few years ago.

## 1.3 Why is Deep Learning Taking off?

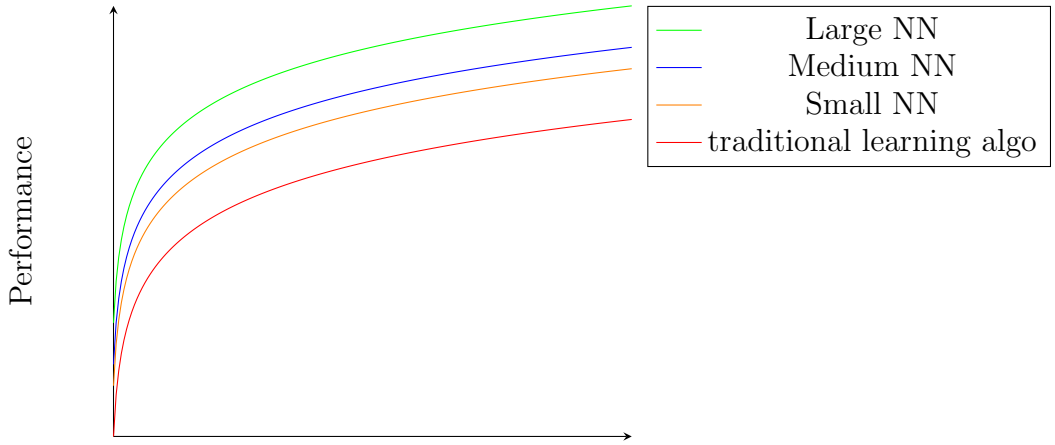


Figure 1.9: Scale drives deep learning progress

In the regime of small number of training sets, the relative ordering of the algorithms is not very well defined. If you don't have a lot of training data, it is often up to your skill at hand engineering features that determines performance. When we have large training sets – large labeled data regime in the right, we more consistently see large neural networks dominating the other approaches.

# Chapter 2

# Neural Networks Basics

## 2.1 Logistic Regression as a Neural Network

When implementing a neural network, you usually want to process entire training set without using an explicit for-loop. Also, when organizing the computation of a neural network, usually you have what's called a forward pass or forward propagation step, followed by a backward pass or backward propagation step.

### 2.1.1 Notation

Each training set will be comprised of $m$ training examples:

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}.$$

We denote $m_{train}$ to be the number of training examples in the training set, and $m_{test}$ to be the number of test examples. The $i$th training example is represented by a pair $(x^{(i)}, y^{(i)})$, where $x^{(i)} \in \mathbb{R}^{n_x}$ and $y^{(i)} \in \{0, 1\}$. To put all of the training examples in a more compact notation, we define matrix $X$ as

$$X = \begin{bmatrix} | & | & & | \\ \mathbf{x}^{(1)} & \mathbf{x}^{(2)} & \cdots & \mathbf{x}^{(m)} \\ | & | & & | \end{bmatrix}$$

where $X \in \mathbb{R}^{n_x \times m}$. Similarly, we define $Y$ as

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & \cdots & y^{(m)} \end{bmatrix}$$

where $Y \in \mathbb{R}^{1 \times m}$.

### 2.1.2 Binary Classification


Figure 2.1: Cat Image

You might have an input of an image and want to output a label to recognize this image as either being a cat, in which case you output 1, or non-cat, in which case you output 0. We use $y$ to denote the output label.

To store an image, computer stores three separate matrices corresponding to the red, green, and blue color channels of this image. If the input image is 64 pixels by 64 pixels, then you would have 3 $64 \times 64$ matrices corresponding to the red, green, and blue pixel intensity values for the image. We could define a feature vector $x$ as follows:

$$x = \begin{bmatrix} | & | & | \\ red & green & blue \\ | & | & | \end{bmatrix}.$$

Suppose each matrix has dimension $64 \times 64$, then

$$n_x = 64 \times 64 \times 3 = 12288.$$

### 2.1.3 Logistic Regression

Given an input feature $x \in \mathbb{R}^{n_x}$, we want to find a prediction $\hat{y}$, where

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

Define the weights $w \in \mathbb{R}^{n_x}$ and bias term $b \in \mathbb{R}$ in the logistic regression. It turns out, when implementing the neural networks, it is better to keep parameters $w$ and $b$ separate, instead of putting them together as $\theta \in \mathbb{R}^{n_x+1}$.

In linear regression, we would use $\hat{y} = w^T x + b$. However, this is not good for binary classification because the prediction could be much bigger than 1 or even negative, which does not make sense for probability. Therefore, we apply the sigmoid function, where

$$\hat{y} = \sigma(w^T x + b) = \sigma(z).$$

The formula of the sigmoid function is given as follows:

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

If $z$ is very large, then $e^{-z}$ will be close to zero, so $\sigma(z) \approx 1$. Conversely, if $z$ is very small or a very large negative number, then $\sigma(z) \approx 0$.

The graph of the sigmoid function is given below:

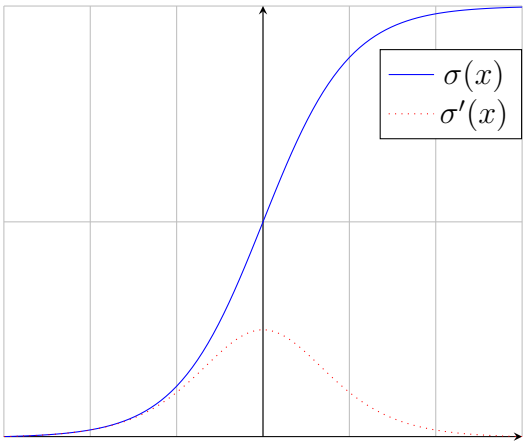


Figure 2.2: Sigmoid Function

### 2.1.4 Logistic Regression Cost Function

Logistic regression is a supervised learning algorithm, where given training examples $\{(x^{(1)}, y^{(1)}, \ldots, (x^{(m)}, y^{(m)})\}$, we want to find parameters $w$ and $b$ such that $\hat{y}^{(i)} \approx y^{(i)}$.

MSE is not used as the loss function because when learning parameters, the optimization problem becomes non-convex, where we end up with multiple local optima, so gradient descent may not find a global optimum. The loss function is defined to measure how good our output $\hat{y}$ is when the true label is $y$. Instead, the loss (error) function for logistic regression could be represented by

$$\mathcal{L}(\hat{y}, y) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})).$$

The intuition behind is that when considering the squared error cost function, we want the squared error to be as small as possible. Similarly, we want to loss function for logistic regression to be as small as possible.

If $y = 1$, then the loss function ends up with

$$\mathcal{L}(\hat{y}, 1) = -\log(\hat{y}).$$

In this case, we want $\log(\hat{y})$ to be large $\Leftrightarrow$ want $\hat{y}$ to be large.

If $y = 0$, then

$$\mathcal{L}(\hat{y}, 0) = -\log(1 - \hat{y}).$$

In this case, we want $\log(1 - \hat{y})$ to be large $\Leftrightarrow$ want $\hat{y}$ to be small.

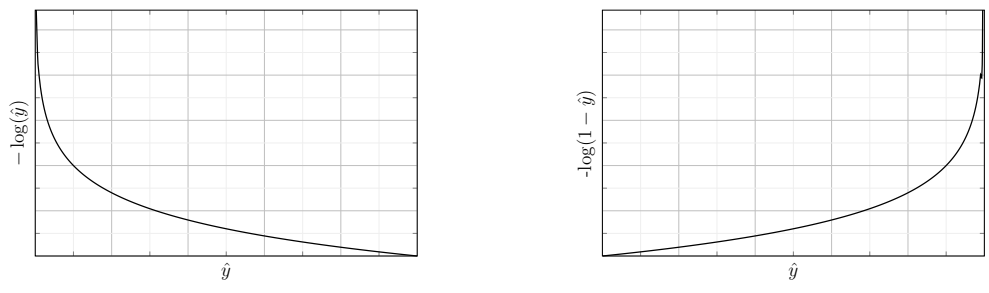


Figure 2.3: A binary classifier's loss function based on if $y = 1$ or $y = 0$.

The loss function is defined with respect to a single training example. The cost function measures how well the model is doing on the entire training set, which is given as

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

### 2.1.5 Gradient Descent

We want to find $w, b$ that minimize $J(w,b)$. Our cost function $J(w,b)$ is a single big bowl, which is a convex function, which is one of the huge reasons why we use this particular cost function $J$ for logistic regression.

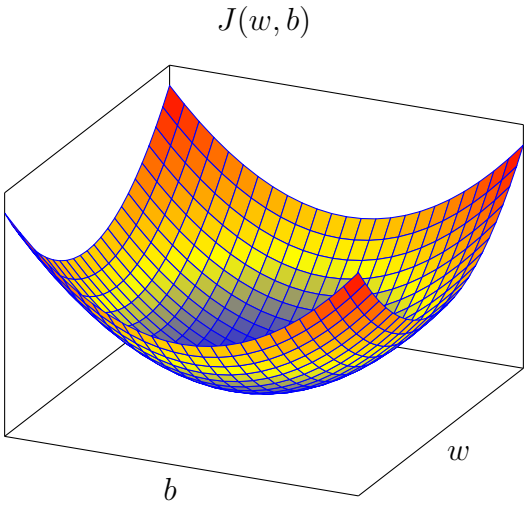


Figure 2.4: Convex Cost Function

For logistic regression, almost any initialization method works. Though random initialization works, people don't usually do that for logistic regression. Instead, we generally initialize the weights to 0. Gradient descent takes a step in the steepest downhill direction. After iterations, we converge to the global optimum or get close to the global optimum. In one dimension, for the parameter $w$, we can define the gradient descent algorithm as follows

$$\begin{aligned} w &:= w - \alpha\frac{\partial J(w,b)}{\partial w} \\ b &:= b - \alpha\frac{\partial J(w,b)}{\partial b} \end{aligned}$$

where $\alpha$ is the learning rate that control the size of steps and $\frac{\partial J(w)}{\partial w}, \frac{\partial J(w,b)}{\partial b}$ are the changes we make for the parameter $w, b$.

### 2.1.6 Computation Graph

The computations of neural network are organized in terms of a forward pass step, in which we compute the output of the neural network, followed by a backward propagation step, which we use to compute the gradients. The computation graph explains why it is organized this way.

Suppose we want the compute the function

$$J(a, b, c) = 3(a + bc).$$

We can break down the computation into three steps:

1. $u = bc$
2. $v = a + u$
3. $J = 3v$

This could be represented by the computation graph below:

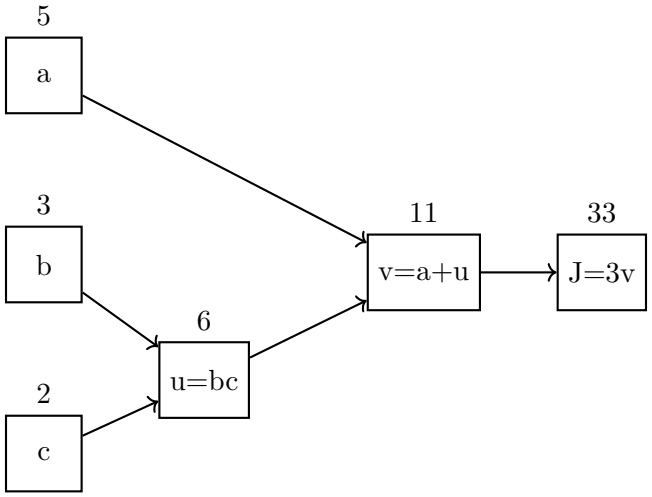


Figure 2.5: Computation Graph for $J(a, b, c) = 3(a + bc)$

Suppose $a = 5, b = 3, c = 2$, then $J(a, b, c) = 3(a + bc) = 3(5 + 3 \times 2) = 33$.

### 2.1.7 Derivatives with a Computation Graph

Consider the computation graph 2.5. Suppose we want to compute the derivative of $J$ with respect to $v$. Since we know $J = 3v$, we can take the derivative as follows:

$$\frac{\partial J}{\partial v} = \frac{\partial}{\partial v}(3v) = 3.$$

Suppose then we want to compute the derivative of $J$ with respect to $a$. We can apply the chain rule as follows:

$$\frac{\partial J}{\partial a} = \frac{\partial J}{\partial v} \cdot \frac{\partial v}{\partial a} = 3 \cdot \frac{\partial}{\partial a}(a + u) = 3 \cdot 1 = 3.$$

For brevity, we could use "$da$" to represent $\frac{\partial J}{\partial a}$. Similarly, we can use "$dv$" to represent $\frac{\partial J}{\partial v}$.

To find the derivative of $J$ with respect to $u$, we could do similar computation as $\frac{\partial J}{\partial a}$ as follows:

$$\frac{\partial J}{\partial u} = \frac{\partial J}{\partial v} \cdot \frac{\partial v}{\partial u} = 3 \cdot 1 = 3.$$

$\frac{\partial J}{\partial b}$ could be compute as follows:

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial u} \cdot \frac{\partial u}{\partial b} = 3 \cdot c = 3 \cdot 2 = 6.$$

Similarly, we can compute

$$\frac{\partial J}{\partial c} = \frac{\partial J}{\partial u} \cdot \frac{\partial u}{\partial c} = 3 \cdot b = 3 \cdot 3 = 9.$$

The most efficient way to compute all these derivatives is through a right-to-left computation following the direction of the arrows below:

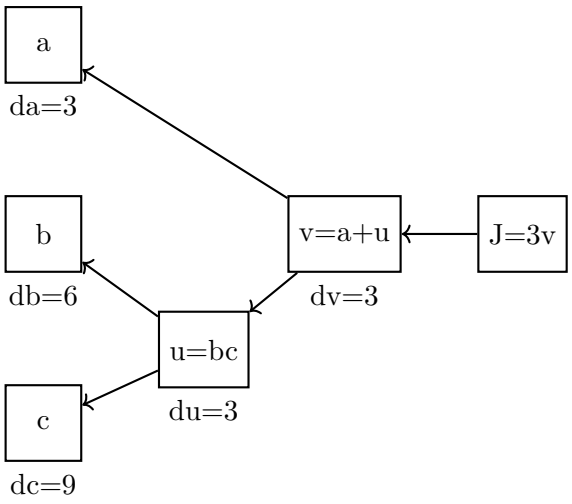


Figure 2.6: Backward Propagation for $J(a, b, c) = 3(a + bc)$

### 2.1.8 Logistic Regression Gradient Descent

This section will cover how to compute derivatives to implement gradient descent for logistic regression.

To recap, logistic regression is set up as follows:

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

This can be represented by the computation graph below:

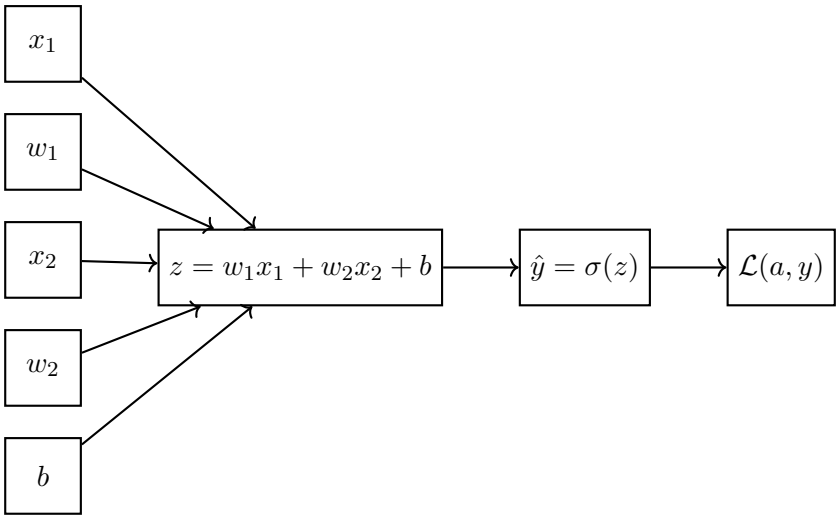


Figure 2.7: Computation Graph for Logistic Regression

In this case, we can compute "$da$" as follows:

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

We can give both terms the same denominator and clean up as follows:

$$\frac{\partial \mathcal{L}(a,y)}{\partial a} = \frac{-y(1-a)}{a(1-a)} + \frac{a(1-y)}{a(1-a)} = \frac{-y + ay + a - ay}{a(1-a)} = \frac{a-y}{a(1-a)}.$$

Then, we can go backwards and compute $\frac{\partial \mathcal{L}(a,y)}{\partial z}$. We know that the derivative of a sigmoid function has the form $\frac{\partial}{\partial z}\sigma(z) = \sigma(z)(1-\sigma(z))$. In this case $\sigma(z) = a$, as we have defined, so $\frac{\partial a}{\partial z} = a(1-a)$. Therefore,

$$dz = \frac{\partial \mathcal{L}(a,y)}{\partial z} = \frac{\partial \mathcal{L}}{\partial a} \cdot \frac{\partial a}{\partial z} = \frac{a-y}{a(1-a)} \times a(1-a) = a - y.$$

Finally, we do the backward propagation for the input layer as follows:

$$\begin{aligned}
\frac{\partial \mathcal{L}(a,y)}{\partial w_1} &= x_1 \cdot \frac{\partial \mathcal{L}(a,y)}{\partial z} = x_1 \cdot (a-y) \\
\frac{\partial \mathcal{L}(a,y)}{\partial w_2} &= x_2 \cdot \frac{\partial \mathcal{L}(a,y)}{\partial z} = x_2 \cdot (a-y) \\
\frac{\partial \mathcal{L}(a,y)}{\partial x_1} &= w_1 \cdot \frac{\partial \mathcal{L}(a,y)}{\partial z} = w_1 \cdot (a-y) \\
\frac{\partial \mathcal{L}(a,y)}{\partial x_2} &= w_2 \cdot \frac{\partial \mathcal{L}(a,y)}{\partial z} = w_2 \cdot (a-y) \\
\frac{\partial \mathcal{L}(a,y)}{\partial b} &= 1 \cdot \frac{\partial \mathcal{L}(a,y)}{\partial z} = (a-y)
\end{aligned}$$

### 2.1.9 Gradient Descent on $m$ examples

Recall that the cost function for $m$ training examples is given by

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

where $a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b)$.

The cost function is the average of loss values for each training example. It turns out that derivative works similarly and we can compute the derivative of $J(w, b)$ with respect to $w_1$ as follows:

$$\frac{\partial}{\partial w_1} J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial w_i} \mathcal{L}(a^{(i)}, y^{(i)}) = \frac{1}{m} \sum_{i=1}^{m} x_1 \cdot (a^{(i)} - y^{(i)}).$$

Now, let's formalize the algorithm for the logistic regression gradient descent.

**Algorithm 1** Gradient Descent for Logistic Regression

**Initialize** $J = 0; dw_1 = 0; dw_2 = 0; db = 0$
**for** $i = 1$ to $m$ **do**
  $z^{(i)} \leftarrow w^T x^{(i)} + b$
  $a^{(i)} \leftarrow \sigma(z^{(i)})$
  $J \leftarrow J + [-(y^{(i)} \log(a^{(i)} + (1 - y^{(i)}) \log(1 - a^{(i)}))]$
  $dz^{(i)} \leftarrow a^{(i)} - y^{(i)}$ ▷ Superscript refers to one training example
  $dw_1 \leftarrow dw_1 + x_1^{(i)} dz^{(i)}$ ▷ Suppose $n = 2$
  $dw_2 \leftarrow dw_2 + x_2^{(i)} dz^{(i)}$ ▷ No superscript i as it's accumulative
  $db \leftarrow db + dz^{(i)}$
**end for**
$J \leftarrow J/m$
$dw_1 \leftarrow dw_1/m; dw_2 \leftarrow dw_2/m; db \leftarrow db/m$

To implement logistic regression this way, we have two for-loops. The first for-loop is to iterate through all training examples. The second for-loop is to iterate through all the features. In the example above, we only have features $w^{(1)}, w^{(2)}$. If we have $n$ features instead, then we need to iterate through all of them.

However, when implementing deep learning algorithms, the explicit for-loop makes the algorithms less efficient. The vectorization technique allows us to get rid of the explicit for-loops, which allows us to scale to larger datasets.

## 2.2 Python and Vectorization

### 2.2.1 Vectorization

To be continued...