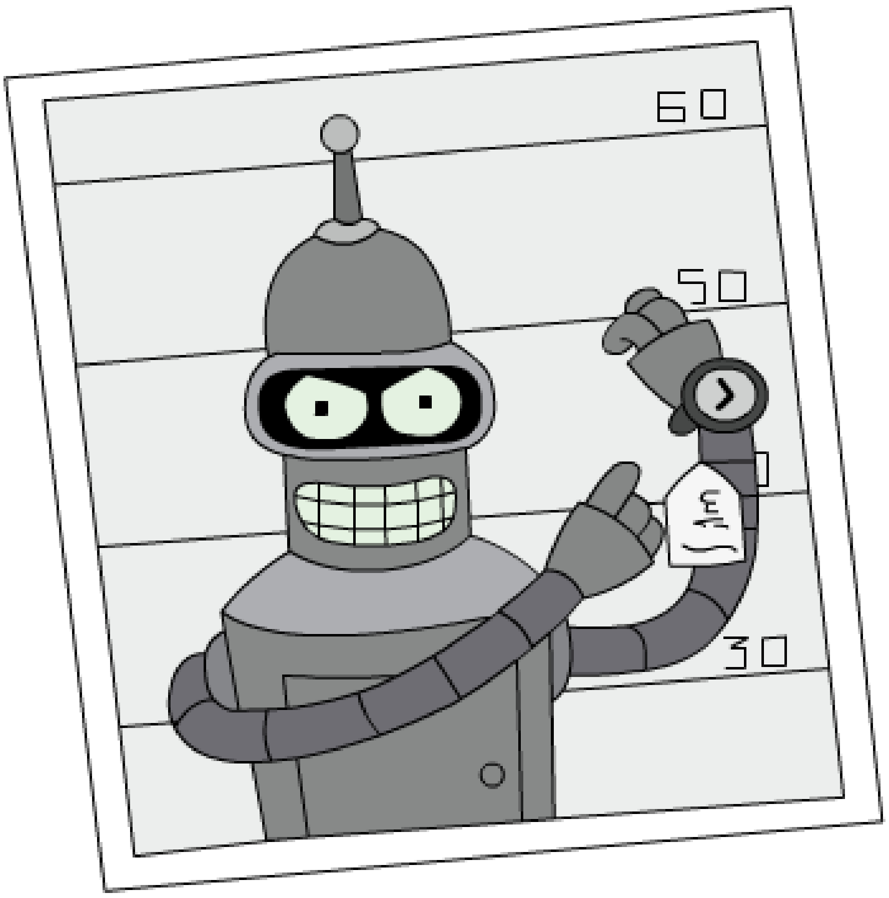
Model training with logistic regression

ECE4179/5179/6179: Neural networks and deep learning
Lab2 (Weeks 3,4)



Academic integrity Every lab submission will be screened for any collusion and/or plagiarism. Breaches of academic integrity will be investigated thoroughly and may result in a zero for the assessment along with interviews with the plagiarism officers at Monash University.

Late submission. Late submission of the lab will incur a penalty of 10% for each day late. That is with one day delay, the maximum mark you can get from the assignment is 9 out of 10, so if you score 9.5, we will (sadly) give you 9. Labs submitted with more than a week delay will not be assessed. Please apply for special consideration for late submission as soon as possible (*e.g.*, documented serious illness).

Each lab is worth 5% and there are a number of sections and tasks with their own weighting. A task is only considered complete if you can demonstrate a working program and show an understanding of the underlying concepts. Note that later tasks should reuse code from earlier tasks.

This lab is about integrating your Python knowledge and applying it to the learning material obtained from the lectures. At the end of this lab, you will be able to train a simple model, and understand the concept of training/testing of models. The following are the four sections that you should complete.

- Section 1: The sigmoid function and making predictions. This section focuses on what sigmoid function, what it is, and writing a function that makes a prediction.
- Section 2: Gradient and Cost Computation. This section focuses on applying the gradient descent function to minimise a cost function (also known as a loss function). In this case, we will be applying it to a cross entropy loss function.
- Section 3: Analysing convergence and accuracy. This section will focus on analysing plots of loss and accuracies, and understanding how well our model is performing. You will also have learnt the concept of train/test data in terms of model evaulation.
- Section 4: Using non-linear features for better classification. This section will focus on improving classification for non-linear features with a simple logistic regression model.

The learning outcomes for this lab are:

- Training and testing machine learning models
- Understanding the data you are dealing with
- Applying gradient descent to minimise a cost function
- Evaluating model performance
- Furthering your understanding of numpy and its core functionality
- Furthering your Python programming skills

Introduction.

The concept of machine learning and deep learning have been around for decades. The fundamentals of model training and being able to correctly evaluate model performance is a key requirement in order to train deep learning models in future labs. In this lab, we will still be using JupyterLab to lay out the template. After becoming familiar with the lecture and lab material corresponding to this lab, you can begin lab 2 by completing the provided template. The submission is 21st of August (Sunday) 4:30 PM AEST.

It is recommended that you go through the following videos/documents prior to attending your lab 2:

1. Begin by reading through this document. This document contains all the relevant information for lab 2
2. Watch the lab 2 video (goes through this document and the student template)
3. Watch the lab 2 key concepts video (this goes through core knowledge)

In the lab and in your own time, you will be completing the lab 2 notebook by going through this document and the provided notebook.

Please note that to achieve the provided checkpoint results (*i.e.*, for the sigmoid function), **do not change** any hyper parameters (variables) that are defined outside of the **"set_background(red)"** cells.

Section 1: The sigmoid function and making predictions (out of 10%)

This section is an introduction to sigmoid and what it means to make predictions. In this case, our definition of predictions is a soft value (*i.e.*, can be any decimal value between 0 and 1 for binary classification). When we want the actual accuracy of a model, we will use the hard value by rounding the predicted value to a 0 or 1.

- 1.1 The sigmoid function
- 1.2 Making predictions

The sigmoid function has the form of:

$$\sigma(z) = \frac{1}{1 + \exp(-z)} \; . \tag{1}$$

The sigmoid function makes an "S" shape (figure below) which bounds the values from $(-\infty, +\infty)$ to $(0, 1)$. Hence this is useful for binary classification.

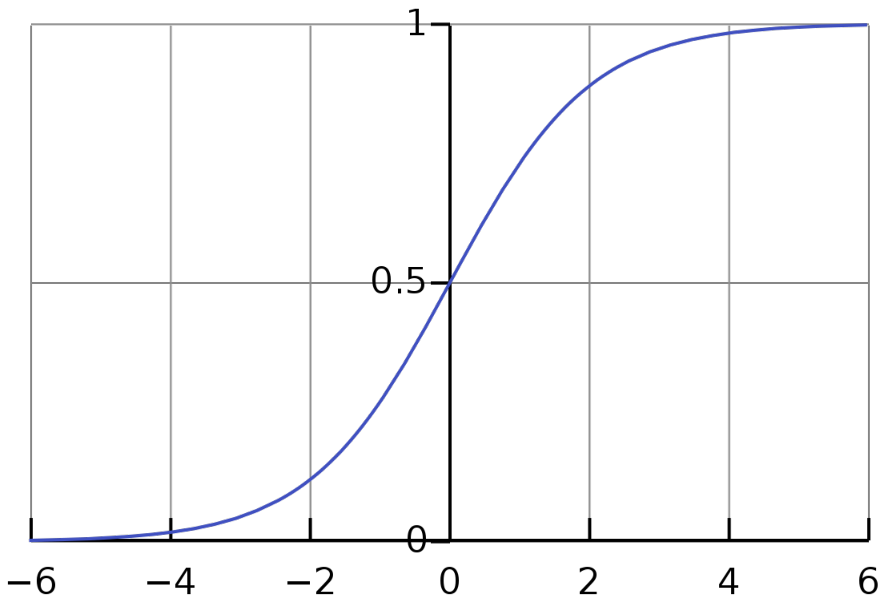


Figure 1: The Sigmoid function.

Section 2: Training a model via Gradient Descent: (out of 20%)

In this section, we are introducing the cross entropy loss that is useful in training models for binary classification problems. We will also update the model parameters via a simple gradient descent method to minimise the loss function.

- 2.1 Gradient and Cost Computation
- 2.2 Training with Gradient Descent
- 2.3 Evaluating the trained model

The cross entropy loss is defined as:

$$\mathcal{L}_{\mathrm{CE}}(\boldsymbol{w}) = -\frac{1}{m}\sum_{i=1}^{m}\left\{y_i \log\left(\underbrace{\sigma\left(\boldsymbol{w}^\top \boldsymbol{x}_i\right)}_{\hat{y}_i}\right) + (1-y_i)\log\left(1-\underbrace{\sigma\left(\boldsymbol{w}^\top \boldsymbol{x}_i\right)}_{\hat{y}_i}\right)\right\} \tag{2}$$

And the gradient of the cross entropy loss w.r.t. the weights $\boldsymbol{w}$ can be written as:

$$\nabla_{\boldsymbol{w}}\mathcal{L}_{\mathrm{CE}} = \frac{1}{m}\sum_{i=1}^{m}\left(\underbrace{\sigma\left(\boldsymbol{w}^\top \boldsymbol{x}_i\right)}_{\hat{y}_i} - y_i\right)\boldsymbol{x}_i \tag{3}$$

The gradient can be calculated via partial the derivative of Eq. (2). This gradient function allows us to minimise the loss by updating the parameters of the model ($\boldsymbol{w}$) by deducting this gradient function from $\boldsymbol{w}$. The amount that is deducted is determined by a hyper parameter (usually written as η) called learning rate. Update via gradient descent uses the formula:

$$\boldsymbol{w}_{\mathrm{new}} = \boldsymbol{w}_{\mathrm{old}} - \eta \cdot \nabla_{\boldsymbol{w}}\mathcal{L}_{\mathrm{CE}} \tag{4}$$

Training and test data has been split already for you, so you will be using **only** the train data to update the model parameters, and using the test data to evaluate the model's performance.

Section 3: Analysing convergence and accuracy (out of 20%)

In the following tasks, you will apply your knowledge of Python to a basic mathematical problem.

- 3.1 Improving the accuracy
- 3.2 Training convergence
- 3.3 Describe & Explain

In any machine learning problem, the base code that generates our model and predictions does not usually yield the optimal results. The model requires some fine-tuning so that the test accuracy is improved. Some hyper-parameters that we have provided so far are learning rate, and number of epochs. We have provided some learning rate values for you to try out, but in the future, you will have to test the values yourself. Normally, learning rate is picked between values of $(0, 1]$ and you should test logarithmically, (*i.e.*, 0.01, 0.1, 1). In this section, we provide some learning rate values that are not usually used to see the effects of high learning rates.

Section 4: Using non-linear features for better classification (out of 50%)

In the following section, you will improve your model via other avenues such as modifying the input parameters, and implementing a dynamic learning rate.

- 4.1 Load data and create train and test dataset
- 4.2 Train the model using GD for scenario 1 data
- 4.3 Decision Boundary for model trained with scenario 1 data
- 4.4 Train the model using GD for scenario 2 data
- 4.5 Decision Boundary for model trained with scenario 2 data
- 4.6 Evaluate models performance with Decaying Learning rate

Load the data file called "Lab2_task4_data.npz" and visualize dataset. It is always good to understand the data before applying models to it. Note that each sample $\boldsymbol{x}_i$ is a point in 2D Euclidean space (*i.e.*, $\boldsymbol{x}_i \in \mathbb{R}^2$) and each label is $y_i \in \{0, 1\}$.

Scenario 1: Non-transformation of input data We will naively train a logistic model without changing any of the input and model parameters. Use the training data (X_train, y_train) to train a logistic model using GD (You can reuse functions **compute_loss_and_grad** and **predict**). Plot the decision boundary and compute the accuracy of the resulting model. (Note that you have to report the accuracy of your model on the test data (X_test, y_test).

Scenario 2: Transforming input variables and using dynamic learning rate Your friend suggests that to better classify this data, you need to use nonlinear features. In particular, he suggests to map a sample $\mathbb{R}^2 \ni x = (x_1, x_2)^\top$ to something like $\mathbb{R}^m$ where $m > 2$, maybe by considering higher-order polynomials (*i.e.*, creating new features in the form x_1^2, x_2^2, $x_1 x_2$ and etc. By looking at the data distribution (4.1), what do you suggest? Use the suggested mapping and then train a new logistic model using GD (You can reuse functions **compute_loss_and_grad** and **predict**). Plot the decision boundary for above on test data. Compute the accuracy of the resulting model (Note that you have to report the accuracy of your model on the test data (X_test, y_test). Evaluate models performance with Decaying Learning rate for non-linear features and report your findings.