A 4-Module Sequence for Applied Deep Learning

Narges Norouzi

University of California Santa Cruz & University of California, Berkeley

1 Outline

The 4-module assignment sequence is designed for an introductory undergraduate deep learning course. Modules are designed to give students hands-on experience with deep learning frameworks and concepts through working with datasets of varying types. The coverage for modules includes:

- **Module 1)** introduction to linear regression analysis by working on Facebook Metrics Dataset [2] for modeling total user interactions. Students have also been introduced to non-linear datasets and how kernel methods and weight regularization can be used for non-linear modeling patterns.
- **Module 2)** through working on the Fashion-MNIST dataset [6], students will 1) understand how to pre-process image data and conduct dimensionality reduction, 2) implement logistic regression from scratch, and 3) implement a neural network and observe the capacity of neural networks to learn a non-linearly separable decision boundary.
- **Module 3)** focuses on designing convolutional neural networks and training them on CIFAR-10 dataset [4]. The modules also introduce transfer learning.
- **Module 4)** students will work on training recurrent neural networks on the Reuters newswire classification dataset [3] and will analyze their observations.

This 4-module assignment sequence is easily adoptable by instructors and is implemented in a modular structure. The target audience of these modules is undergraduate students taking artificial intelligence or machine learning courses that cover topics such as 1) linear regression, 2) logistic regression, 3) neural networks, 4) convolutional neural networks, and 5) recurrent neural networks. The prerequisites for the course include programming, data structures, algorithms, and probability courses.

This document provides a summary of the required background knowledge for each module as well as a high-level outline of the 4-module sequence.

2 Module 1: Linear Regression

In this module, students will train a linear regression model to predict total interactions on the Facebook Metrics dataset [2]. Students will also learn how to fit a non-linear polynomial model using the linear regression framework and use weight regularization to control the polynomial degree of the model.

2.1 Question 1: Linear regression from scratch

In this question, students implement linear regression algorithm from scratch in Python. Linear regression aims to map feature vectors to a continuous value in the range $[-\infty, +\infty]$ by linearly combining the feature values.

Data is represented as a dataframe or a feature matrix. Let our feature matrix be X whose dimensions are $n \times m$, θ be a weight matrix of dimensions $m \times 1$, the bias vector b a column vector of dimension $m \times 1$. Using these we can predict $\hat{Y}$ by the following relationship:

$$\hat{Y} = X\theta + b$$

2.1.1 Data: Facebook posts metrics

The dataset in this assignment contains features describing posts from a cosmetic brand's Facebook page. The authors use the following features:

- Category,
- Page total likes: Number of people who have liked the company's page),
- Type: Type of content (Link, Photo, Status, Video),
- Post month: Month the post was published (January, February, March, ..., December),
- Post hour: Hour the post was published (0, 1, 2, 3, 4, ..., 23) ,
- Post weekday: Weekday the post was published (Sunday, Monday, ..., Saturday) ,
- Paid: If the company paid to Facebook for advertising (yes, no)

These features are used to model any of the following: 'Lifetime Post Total Reach', 'Lifetime Post Total Impressions', 'Lifetime Engaged Users', 'Lifetime Post Consumers', 'Lifetime Post Consumption', 'Lifetime Post Impressions by people who have liked your Page', 'Lifetime Post reach by people who like your Page', 'Lifetime People who have liked your Page and engaged with your post', 'comment', 'like', 'share', 'Total Interactions'.

In this assignment, we will focus on 'Total Interactions'. Our feature space will include: Category, Page total likes, Post month, Post hour, Post weekday, and Paid.

2.1.2 Training and testing the linear regression model

Students will implement Mean Squared Error (MSE) and gradient descent algorithm. Suppose our dataset consists of n records, each with d features:

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,d} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,d} \end{bmatrix}$$

One way to include a bias is to augment X with a column of ones:

$$X = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,d} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n,1} & x_{n,2} & \cdots & x_{n,d} \end{bmatrix}$$

We also have n labels corresponding to the correct classification of each of the above records, $y = [y_1, y_2, \cdots, y_n]^T$, i.e.:

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Through optimization, we will find the optimal parameter values $\theta = [\theta_0, \theta_1, \cdots, \theta_d]^T$ of the linear regression model, where θ_0 is the bias weight. To simplify our notation, let

$$\hat{y} = X\theta = \begin{bmatrix} X_{1,0}\theta_0 + X_{1,1}\theta_1 + \cdots + X_{1,d}\theta_d \\ X_{2,0}\theta_0 + X_{2,1}\theta_1 + \cdots + X_{2,d}\theta_d \\ \vdots \\ X_{n,0}\theta_0 + X_{n,1}\theta_1 + \cdots + X_{n,d}\theta_d \end{bmatrix} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_n \end{bmatrix}$$

We seek θ such that the MSE is minimized (the $1/2$ factor makes the derivation easier). Let the MSE be a function of θ, $J(\theta)$:

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

Since the above is a convex function, it has a unique minimum value. Taking the derivative with respect to θ_i, we get:

$$\begin{aligned} \frac{\partial}{\partial \theta_j} J(\theta) &= \frac{1}{2n} \sum_{i=1}^{n} \frac{\partial}{\partial \theta_j} (\hat{y}_i - y_i)^2 \\ &= \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) \frac{\partial}{\partial \theta_j} (\hat{y}_i) \end{aligned}$$

Recall the chain rule from calculus, and that each $\hat{y}_i$ is a function of the θ_i, so the above becomes:

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) x_{i,j}$$

Students will:

1. Get training and testing set by calling train_test_split()
2. Define a weight (θ) vector
3. Implement Gradient Descent using the information above
4. Record the Sum Squared Error for training and test data
5. Return the weight matrix, train errors and test errors
6. Plot the training and test errors and comment on the plot.

2.2 Question 2: Fitting non-linear data

Data may not follow a linear relationship from the independent variable X to the dependent variable y. Fitting a linear model to this would be inaccurate and yield a high loss.

If we want to model an order d polynomial relationship between X and y we can augment our initial linear model where instead of having:

$$y_i = \theta_0 + \theta_1 x_i$$

We have:

$$y_i = \theta_0 + \theta_1 x_i + \theta_2 x_i^2 + \cdots + \theta_d x_i^d$$

We can use the same linear regression algorithm we if we first augment X and add extra columns (or dimensions).

$$\mathbf{X} = \begin{bmatrix} x_1 & x_1^2 & \cdots & x_1^d \\ x_2 & x_2^2 & \cdots & x_2^d \\ \vdots & \vdots & \ddots & \vdots \\ x_n & x_n^2 & \cdots & x_n^d \end{bmatrix}$$

Then our new higher order $\hat{y}$ is computed same as before.

$$\hat{y} = X\theta = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^d \\ 1 & x_2 & x_2^2 & \cdots & x_2^d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^d \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} = \begin{bmatrix} \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \cdots + \theta_d x_1^d \\ \theta_0 + \theta_1 x_2 + \theta_2 x_2^2 + \cdots + \theta_d x_2^d \\ \vdots \\ \theta_0 + \theta_1 x_n + \theta_2 x_n^2 + \cdots + \theta_d x_n^d \end{bmatrix} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_n \end{bmatrix}$$

2.2.1 Weight regularization

When we try to fit a d-order polynomial to our data, we could end up overfitting. This happens when you try to fit a higher dimensional curve than what the distribution of our data actually exhibits. We can mitigate this by choosing an order d that matches your data closely, but often times this is not directly apparent in noisy data. Another method to avoid overfitting is regularizing, where you modify your loss to keep weights small which flattens our polynomial. This helps us avoid learning polynomials that are too complex for our data.

To add regularization we modify the original loss function J to include regularizing term and a new hyperparameter that we tune λ. This controls the amount of regularizing we impose on the weights. We use the loss computed from the validation set to tweak this parameter.

$$J(\theta) = \frac{1}{2n}\sum_{i=1}^{n}(h^{(i)} - y^{(i)})^2 + \lambda\sum_{j=1}^{d}\theta_j^2$$

Our gradient computation also changes:

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{n}\sum_{i=1}^{n}(h^{(i)} - y^{(i)})x_{i,j} + 2\lambda\theta_j$$

We apply this gradient the same way as before in our gradient descent algorithm:

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

3 Module 2: Logistic Regression and Neural Network

In this module, students will use the Fashion-MNIST dataset [6], which is a cool little dataset with gray scale 28×28 images of articles of clothing. Using this dataset, students will observe linearly separable and non-linearly separable datasets and will train logistic regression and neural network for each type of decision boundary.

3.1 Question 1: Data pre-processing

The images in the dataset are valued from $[0, 255]$. This is the normal range for images and we need to normalize the data.

In order to normalize the data to $[0, 1]$, we use the equation:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

We can assume that $x_{min} = 0$ and $x_{max} = 255$, this is a safe assumption since we are working with image data. This means that for image data, if we want to normalize to $[0, 1]$ the equation simplifies to:

$$img_{norm} = \frac{img}{255}$$

Students will then need to reshape every single image in the dataset as a vector of length $28 \times 28 = 784$. The data is then transformed into a 2D data using PCA algorithm.

3.2 Question 2: Linearly-separable data

Some of the data is easily to separate with a line, this concept is called linear separability. Students will plot samples from Ankle Boot and Trouser class. and will observe that it's easy to distinguish between a shoe and pants with a linearly-separable decision boundary.

3.3 Question 3: Non-linearly separable data

Students will then plot the Pullover and Coat classes which are basically mixed together. We can't easily draw a line between them.

3.4 Question 4: Logistic regression

Recall that each image is $28 \times 28 \times 1$ matrix which we flatten to a 784-dimensional row vector. Image i looks like:

$$\mathbf{x}_i = \begin{bmatrix} x_1 & x_2 & \cdots & x_{784} \end{bmatrix}$$

The dataset $\mathbf{X}$ is then the collection of all these n images.

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,784} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,784} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,784} \end{bmatrix}$$

Logistic regression uses a bias term, we can easily incorporate this by adding a column of ones at the beginning.

$$\mathbf{X} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,784} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,784} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n,1} & x_{n,2} & \cdots & x_{n,784} \end{bmatrix}$$

For each image we have a corresponding label y_i. This is the class "Coat" and "Pullover". Each of which is mapped to a unique number.

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

We will try to find the optimal parameter values $\theta = [\theta_0, \theta_1, \cdots, \theta_{784}]^T$ of our logistic regression model, where θ_0 is the bias weight. To simplify our notation, let

$$Z = X\theta = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,784} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,784} \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & x_{n,1} & x_{n,2} & \cdots & x_{n,784} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_{784} \end{bmatrix}$$

$$= \begin{bmatrix} \theta_0 + x_{1,1}\theta_1 + \cdots + x_{1,784}\theta_{748} \\ \theta_0 + x_{2,1}\theta_1 + \cdots + x_{2,784}\theta_{784} \\ \vdots \\ \theta_0 + X_{n,1}\theta_1 + \cdots + x_{n,784}\theta_{784} \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}$$

Since each z_i is in the range $(-\infty, \infty)$ and our labels are $[0, 1]$ we pass z_i into the sigmoid function.

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

Now we can make predictions using h_i by simply rounding it to 0 or 1.

$$pred_i = round(h_i)$$

In order to train our logistic regression model we need to find the parameter vector θ that minimizes our cost function J, we will be Binary Cross Entropy (BCE).

$$\mathrm{J} = -\frac{1}{n}\sum_{i=1}^{n} \left[y_i \log(h_i) + (1 - y_i) \log(1 - h_i) \right]$$

Where y_i is your true label and $h_i = \sigma(z_i)$.

In order to know whether we want to increment or decrement our weights to minimize the loss, we calculate its partial derivative with respect to weights, we call this the gradient, the matrix form of the gradient is:

$$\nabla \mathrm{J} = \frac{1}{n} X^T (H - Y)$$

with $X = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,784} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,784} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n,1} & x_{n,2} & \cdots & x_{n,784} \end{bmatrix}$, $\mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$, and $\mathbf{H} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \end{bmatrix}$

We then use this gradient in an iterative algorithm called gradient descent where every iteration we change the weights like so:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla \mathrm{J}$$

Where t is the iteration number, α is your learning rate, $\theta^{(t)}$ is your current weight column vector, and $\theta^{(t+1)}$ is your updated weight column vector.

3.5 Question 5: Neural network

Since Pullovers vs. Coats demonstrate a non-linear decision boundary, a logistic regression cannot accurately classify between the two classes. Instead, in this question, students will build a neural network with Keras.

4 Module 3: Convolutional Neural Network

In this module, students will build a classifier model that is able to distinguish between 10 different classes of images - airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks in CIFAR-10 [4].

4.1 Question 1: Convolutional neural network baseline

Students will follow these steps to build a baseline CNN model:

1. Explore the data
2. Build a small Convolutional Neural Network to solve the classification problem
3. Evaluate training and validation accuracy

4.2 Question 2: Transfer learning

There is a huge shortcut possible in training neural networks for recognition tasks, called transfer learning. The idea is to start with a fully-trained image recognition neural network, off-the-shelf with trained weights. We can repurpose the trained network for a particular recognition task, making use of the days of training that were needed to find those weights. What learned by the neural network in its early layers are useful features in recognizing various features in images. Keras even has pre-trained models built in for this purpose.

Some Keras Pretrained Models include: Xception, VGG16, VGG19, ResNet, ResNetV2, ResNeXt, InceptionV3, InceptionResNetV2, MobileNet, MobileNetV2, DenseNet, and NASNet.

Usually one uses the layers of the pre-trained model up to some point, and then creates some fully-connected layers to learn the desired recognition task. The earlier layers are "frozen", and only the later layers need to be trained. In this module, VGG16 [5] is used, which was trained to recognize 1000 objects in ImageNet [1].

5 Module 4: Recurrent Neural Network

In this module, students will train a Recurrent Neural Network (RNN) for a Natural Language Processing (NLP) task.

5.1 Question 1: Recurrent neural network for topic classification

In this question, Reuters newswire [3] is used for a news topic classification task. The dataset includes text paired with 46 topics as labels. Students will create a model using an RNN layer (Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU), unidirectional or bidirectional) to achieve at least 60% validation accuracy in 10 epochs or less.

5.2 Question 2: Comparison between deep recurrent structures

Using the topic classification task in question 1, students are asked to answer the following questions:

1. What are the major differences between LSTM or GRU, unidirectional or bidirectional?
2. What are the major advantages of each?
3. What are the major disadvantages of each?

5.3 Acknowledgement

I would like to thank all my Teaching Assistants who contributed to the conception and implementation of the 4-module sequence. I would also like to thank my students that provided feedback over the years to improve the quality of these assignments.

References

[1] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[2] Kaggle. Facebook Metrics. `https://www.kaggle.com/datasets/masoodanzar/facebook-metrics`, 2020. [Online; accessed 27-November-2022].

[3] Keras. Reuters Newswire Sataset. `https://keras.io/api/datasets/reuters/`. [Online; accessed 27-November-2022].

[4] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[5] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[6] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.