AI 2024
Week 4 Task

1 Training an ANN

In this task, we will explore a simple Artificial Neural Network (ANN) with one hidden layer. The network has three input features, denoted as x_1, x_2, and x_3, and one output $\hat{y}$. The architecture of the network is depicted in Figure 1.

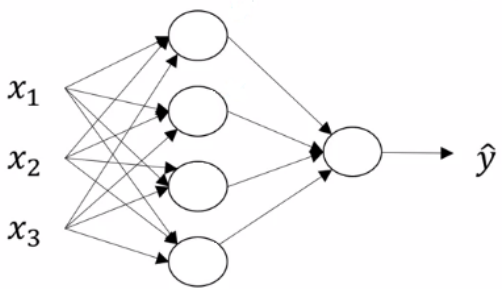

Figure 1: A simple neural network with one hidden layer and three input features.

1.1 Network Structure and Cost Function

1. **Input Layer**: Consists of three input neurons (x_1, x_2, x_3).
2. **Hidden Layer**: Contains four neurons. Each neuron in the hidden layer calculates a weighted sum of all input features, adds a bias term, and applies a tanh activation function:
$$z_j^{(1)} = \sum_{i=1}^{3} w_{ij}^{(1)} x_i + b_j^{(1)}, \quad a_j^{(1)} = \tanh(z_j^{(1)}) \quad \text{for } j = 1, 2, 3, 4.$$
3. **Output Layer**: Contains a single neuron that computes a weighted sum of all hidden layer outputs, adds a bias term, and applies a Sigmoid activation function:
$$z^{(2)} = \sum_{j=1}^{4} w_j^{(2)} a_j^{(1)} + b^{(2)}, \quad \hat{y} = \sigma(z^{(2)}) = \frac{1}{1 + e^{-z^{(2)}}}.$$
4. **Cost Function**: The cost function J is defined as the average of the binary cross-entropy loss over all training examples. For a single training example, the binary cross-entropy loss $\mathcal{L}$ is given by:
$$\mathcal{L} = -\left(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right). \tag{1}$$

Notation:

- $w_{ij}^{(l)}$: Weight connecting neuron i in layer $l-1$ to neuron j in layer l.
- $b_j^{(l)}$: Bias term for neuron j in layer l.
- $z_j^{(l)}$: Weighted input to neuron j in layer l before activation.
- $a_j^{(l)}$: Output of neuron j in layer l after applying the activation function.

1.2 Initialization of Weights and Data Generation

We will use random data points with three features and a corresponding label (0 or 1). The generated data points are:

$\mathbf{x}$	y
$(0.23, 0.45, 0.67)$	1
$(0.89, 0.12, 0.34)$	0
$(0.56, 0.78, 0.12)$	1
$(0.34, 0.88, 0.54)$	0
$(0.98, 0.76, 0.65)$	1

We initialize the weights and biases randomly as follows:

$$\mathbf{w}^{(1)} = \begin{bmatrix} 0.45 & -0.12 & 0.78 \\ 0.05 & 0.35 & -0.22 \\ -0.55 & 0.11 & 0.67 \\ 0.72 & -0.85 & 0.45 \end{bmatrix}, \quad \mathbf{b}^{(1)} = \begin{bmatrix} 0.1 \\ -0.1 \\ 0.2 \\ -0.2 \end{bmatrix},$$

$$\mathbf{w}^{(2)} = \begin{bmatrix} 0.25 \\ -0.35 \\ 0.15 \\ -0.05 \end{bmatrix}, \quad \mathbf{b}^{(2)} = 0.3.$$

1.3 Tasks

1.3.1 Gradients and Partial Derivatives

Calculate the partial derivatives required for backpropagation:

1. Derivative of the loss $\mathcal{L}$ with respect to the output of the network $\hat{y}$:

$$\frac{\partial \mathcal{L}}{\partial \hat{y}}$$

2. Derivative of the output activation (Sigmoid) with respect to its input:

$$\frac{\partial \hat{y}}{\partial z^{(2)}}$$

3. Derivative of the activation of hidden layer neurons with respect to their weighted input (tanh):

$$\frac{\partial a_j^{(1)}}{\partial z_j^{(1)}}$$

4. Derivative of the cost function J with respect to the output neuron input $z^{(2)}$ using the loss function from Equation (1):

$$\frac{\partial J}{\partial z^{(2)}}$$

5. Derivative of the loss with respect to the weights of the hidden layer:

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^{(1)}}$$

6. Derivative of the loss with respect to the bias terms in the hidden layer:

$$\frac{\partial \mathcal{L}}{\partial b_j^{(1)}}$$

7. Derivative of the loss with respect to the weights of the output layer:

$$\frac{\partial \mathcal{L}}{\partial w_j^{(2)}}$$

8. Derivative of the loss with respect to the bias term in the output layer:

$$\frac{\partial \mathcal{L}}{\partial b^{(2)}}$$

1.3.2 Forward Pass

Perform a forward pass through the network using the first data point and the weights and biases provided in the Initialization of Weights and Data Generation subsection:

1. Calculate the values $z_j^{(1)}$ for each neuron in the hidden layer.
2. Apply the tanh activation function to compute $a_j^{(1)}$.
3. Compute the output neuron input $z^{(2)}$.
4. Apply the Sigmoid activation function to compute $\hat{y}$.

1.3.3 Backpropagation and Training

Using the backpropagation algorithm:

1. Compute the gradients for each weight and bias using the partial derivatives and the loss function from Equation (1).
2. Update the weights and biases using Gradient Descent:

$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \alpha \frac{\partial \mathcal{L}}{\partial w_{ij}^{(l)}}, \quad b_j^{(l)} \leftarrow b_j^{(l)} - \alpha \frac{\partial \mathcal{L}}{\partial b_j^{(l)}}$$

where $\alpha = 0.05$ is the learning rate.

2 Implementing an Artificial Neural Network for the Titanic Dataset

In this section, you will implement a simple Artificial Neural Network (ANN) from scratch using NumPy, without using any deep learning frameworks such as TensorFlow or PyTorch. The objective is to train a model that predicts whether a passenger survived the Titanic disaster based on their characteristics. The network will have a maximum of two layers (one hidden layer and one output layer).

2.1 Steps to Implement the ANN

Follow these steps to complete the task:

2.1.1 Download the Datasets

Download the train and test datasets from the Titanic competition on Kaggle.

2.1.2 Load the Datasets

Load the downloaded datasets into your Python environment using libraries such as pandas.

2.1.3 Perform Exploratory Data Analysis (EDA)

Conduct exploratory data analysis (EDA) on the datasets to understand the features and their distributions. In particular, focus on:

- Identifying and handling missing values.
- Normalizing numerical features to ensure they are on a similar scale.
- Encoding categorical features into numerical values if necessary.

You can refer to your Week 1 task results for insights and methods on EDA.

2.1.4 Divide the Train Set into Feature Matrix and Labels

Divide the training dataset into two separate arrays:

- $\mathbf{X}_{\text{train}}$: A matrix containing all the input features for the training examples.
- $\mathbf{Y}_{\text{train}}$: A vector containing the corresponding labels (0 for not survived, 1 for survived).

Similarly, prepare the test dataset features as $\mathbf{X}_{\text{test}}$.

2.1.5 Initialize Weights and Biases

Randomly initialize the weights and biases for each layer in your neural network. Use small random values to start the optimization process.

2.1.6 Implement Forward Pass Function

Write a function to perform a forward pass through the network. This function should compute the outputs (activations) for each layer given an input feature vector.

2.1.7 Implement Backpropagation Function

Write a function to perform backpropagation. This function should compute the gradients (partial derivatives) of the loss function with respect to each weight and bias.

2.1.8 Update Parameters Function

Write a function to update the network's parameters (weights and biases) using the gradients computed in the backpropagation step. The update should use gradient descent and a specified learning rate.

2.1.9 Prediction Function

Write a function to predict if a passenger has survived or not. The function should take an input feature vector, perform a forward pass, and output a binary prediction (0 or 1).

2.1.10 Training and Evaluation Function

Combine the above functions into a single function that:

- Takes $\mathbf{X}_{\text{train}}$, $\mathbf{Y}_{\text{train}}$, $\mathbf{X}_{\text{test}}$, `num_iterations`, and `learning_rate` as inputs.
- Trains the model over the specified number of iterations.
- Logs the loss on the training set for each iteration.
- Saves the predictions for the test set in a CSV file formatted according to the Kaggle submission requirements.

  Note: Ensure that your code is vectorized and utilizes NumPy for all matrix and vector operations to optimize performance.

2.1.11 Upload Results to Kaggle

Upload your prediction CSV file to Kaggle to evaluate your model's accuracy. Record the score for reporting purposes.

2.1.12 Upload to IUTBox

Finally, upload your code, the Kaggle predictions CSV file, and the accuracy score obtained from Kaggle to IUTBox as instructed.

2.2 Optional Steps

- Experiment with different architectures (e.g., changing the number of neurons in the hidden layer) to see their impact on performance.
- Plot the loss curve to visualize how the training progresses over time.
- Test different learning rates and observe their effects on model convergence.