**Critical Thinking 3**

Viet Ngo

Colorado State University Global

CSC580: Applying Machine Learning and Neural Networks - Capstone

Isaac Gang

10/01/2024

**Critical Thinking #3**

The code for this assignment performs linear regression using TensorFlow 2, where a simple linear model is trained on noisy data. The training data is generated, the model is defined, the training process is implemented, and lastly, the plot is shown with a fitted line. The generate\_nums function generates random linear data with added noise. First, 50 points are created with np.linspace for both x and y. Then, uniform random noise is added to both to simulate real-world imperfect data. The data is normalized by subtracting the mean and dividing by the standard deviation, ensuring that the data has zero mean and unit variance, which improves numerical stability during training. The generated data, x\_train and y\_train, are converted into TensorFlow tensors using tf.convert\_to\_tensor. his allows TensorFlow operations to be applied efficiently during model training. The model parameters, W for weight and b for bias, are defined as trainable TensorFlow variables using tf.Variable. Both are initialized with small random values drawn from a normal distribution scaled by 0.01 to avoid large gradient updates in the early training stages. The linear regression model is defined with the hypothesis function y = Wx + b, where W is the slope and b is the y-intercept. The model function takes x as input and returns the predicted y values based on the current weight and bias. The cost function, which measures the difference between the predicted values and the actual target values, is defined as the mean squared error. The cost function computes the average squared differences between predictions y\_pred and true values y\_true. An optimizer is used to adjust the weight and bias to minimize the cost function. The Stochastic Gradient Descent (SGD) optimizer is selected with a learning rate of 0.001. This ensures gradual updates to the model parameters, making the training process more stable. In each training step, the model's gradients are calculated with respect to the cost function using TensorFlow's tf.GradientTape. These gradients are then used by the optimizer to update the weight and bias. A check is also added to detect if any NaN values appear in the gradients during training, which can help catch potential issues like instability or overflow in the model. The training loop runs for 1000 epochs, and after every 100 epochs, it prints the current loss, weight, and bias values. This loop calls the train\_step function repeatedly, updating the model parameters each time based on the computed gradients. The loss (MSE) is expected to decrease progressively as the model converges towards an optimal solution. Once the training loop completes, the final loss/cost, weight, and bias are printed. These values indicate how well the model has learned to fit the data. Finally, the training data and the fitted line are plotted. The training data points are displayed as blue dots, while the fitted line is plotted in red. This visualization helps confirm how well the model has captured the underlying pattern in the noisy data.

**GitHub**: https://github.com/khoiviet24/CSC580\_Applying\_Machine\_Learning\_and\_Neural\_Networks/tree/main/Module3