Assignment 5

Instructions

1. Each problem needs to be coded in python only.
2. In Q2, in-built commands cannot be used.

Assignment Questions

1. Consider the training data set with $y = ||x||^2$.
   (a) Pick $x_1, x_2, \ldots, x_{10000} \in \mathbb{R}^2$ so that x_i's are i.i.d. $\sim$ Unif$[-10, 10]^2$. Generate y_i values using the expression given above.
   (b) Consider the feed-forward neural network with single hidden layer. Choose the right number of nodes in the hidden layer, M, so as to minimize the squared error. It is sufficient to consider $M \leq 10$. Fit the training data using the command *sklearn.neural_network.MLPRegressor*. Use $ReLU$ activation function between input-hidden layers. **Print** the training error.
   (c) Generate 10,000 test vectors $x_1, x_2, \ldots, x_{10000} \in \mathbb{R}^2$ i.i.d. $\sim$Unif$[-10, 10]^2$, and test them on the trained networks. **Print** the test error. **Plot** $g(x, y) = x^2 + y^2$ in 3D for $x \in [-5, 5]$ and $y \in [-5, 5]$. **Plot** the prediction by neural networks in the same plot.
2. Consider the same training data set as in Q1, and the number of nodes chosen. Compute the weights in the neural network using adam optimizer method (given below in Figure 1). The adam optimizer needs to be coded from scratch. Generate the initial value of weights and intercepts independently from a standard Gaussian distribution. Minimize the function $\sum_{i=1}^{n}(y_i - f(x_i))^2$ with respect to α and β, where α and β are the weights in between input-hidden layers and hidden-output layers respectively.
   (a) Consider the convergence criterion to be $||\theta^{(k)} - \theta^{(k+1)}|| \leq 10^{-4}$. **Print** the number of iterations needed for convergence, and the mean squared error.
   (b) Now choose the initial values of weights as the weights computed in Q1b. Repeat the computation using adam optimizer. **Print** the number of iterations for convergence, and the mean squared error.

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Figure 1: ADAM optimizer

3. Consider the alphabet prediction problem discussed in the class.

   (a) Generate 500 alphabets from the set $\{b, c, d, f\}$ uniformly at random. Then convert each alphabet as follows:

   $$b \to ba,\ c \to ce,\ d \to di,\ f \to fo.$$

   We thus have a set of features $X_1, X_2 \ldots, X_{1000} \in \{b, c, d, f, a, e, i, o\}$. **Print** $X_1, X_2, \ldots, X_{20}$. The task is to predict the next alphabet. So fix the responses $Y_i = X_{i+1}$ with Y_{1000} generated randomly.

   (b) Construct a *recurrent neural network* using a *SimpleRNN* layer and a *Dense* layer from *tensorflow.keras.layers*. Choose the number of nodes and the activation function so as to minimize the *cross-entropy* loss function. **Print** the fraction of misclassified consonants and misclassified vowels.