Task 1 Stochastic Gradient Descent for Linear Models

This task needs to be implemented entirely using TensorFlow/PyTorch, without using NumPy.

- Implement a polynomial function polynomial_fun, that takes two input arguments, a weight vector $\mathbf{w}$ of size $M + 1$ and an input scalar variable x, and returns the function value y. [3]

$$y = \sum_{m=0}^{M} w_m x^m$$

- Using the linear algebra modules in TensorFlow/PyTorch, implement a least square solver for fitting the polynomial functions, fit_polynomial_ls, which takes N pairs of x and target values t as input, with an additional input argument to specify the polynomial degree M, and returns the optimum weight vector $\hat{\mathbf{w}}$ in least-square sense, i.e. $\|t - y\|^2$ is minimised. [5]
- Using relevant functions/modules in TensorFlow/PyTorch, implement a stochastic minibatch gradient descent algorithm for fitting the polynomial functions, fit_polynomial_sgd, which has the same input arguments as fit_polynomial_ls does, with additional two input arguments, learning rate and minibatch size. This function also returns the optimum weight vector $\hat{\mathbf{w}}$. During training, the function should report the loss periodically using printed messages. [5]
- Implement a task script "task.py", under folder "task1", performing the following:
    - Use polynomial_fun ($M = 3$, $\mathbf{w} = [1,2,3,4]^{\mathrm{T}}$) to generate a training set and a test set, in the form of respectively sampled 100 and 50 pairs $x, x \epsilon [-20, 20]$ and observed t. The observed t values are obtained by adding Gaussian noise (standard deviation being 0.2) to y. [3]
    - Use fit_polynomial_ls ($M = 4$) to compute the optimum weight vector $\hat{\mathbf{w}}$ using the training set. In turn, compute the predicted target values $\hat{y}$ for all x in both the training and test sets. [2]
    - Report, using printed messages, the mean (and standard deviation) in difference a) between the observed training data and the underlying "true" polynomial curve; and b) between the "LS-predicted" values and the underlying "true" polynomial curve. [3]
    - Use fit_polynomial_sgd ($M = 4$) to optimise the weight vector $\hat{\mathbf{w}}$ using the training set. In turn, compute the predicted target values $\hat{y}$ for all x in both the training and test sets. [2]
    - Report, using printed messages, the mean (and standard deviation) in difference between the "SGD-predicted" values and the underlying "true" polynomial curve. [2]
    - Compare the accuracy of your implementation using the two methods with ground-truth on test set and report the root-mean-square-errors (RMSEs) in both $\mathbf{w}$ and y using printed messages. [3]
    - Compare the speed of the two methods and report time spent in fitting/training (in seconds) using printed messages. [2]

Task 2 A Regularised DenseNet

For the purpose of this task, the dataset is simply split into two, training and test sets, as in the tutorial.

- Adapt the Image Classification tutorial to implement a new network DenseNet3, with the following:
    - Contain a member function dense_block, implementing a specific form of DenseNet architecture, each contains 4 convolutional layers. [3]

    - Design and implement the new network architecture to use 3 of these dense blocks. [4]
    - Summarise and print your network architecture, e.g. using built-in summary function. [1]
- Implement a data augmentation function cutout, using the Cutout algorithm.
    - Use square masks with variable size and location. [2]
    - Add an additional parameter s, such that the mask size can be uniformly sampled from [0, s]. [3]
    - Location should be sampled uniformly in the image space. N.B. care needs to be taken around the boundaries, so the sampled mask maintains its size. [3]
    - Visualise your implementation, by saving to a PNG file "cutout.png", a montage of 16 images with randomly augmented images that are about to be fed into network training. [3]
    - Add Cutout into the network training. [3]
- Implement a task script "task.py", under folder "task2", completing the following:
    - Train the new DenseNet classification network with Cutout data augmentation. [3]
    - Run a training with 10 epochs and save the trained model. [3]
    - Submit your trained model within the task folder. [2]
    - Report the test set performance in terms of classification accuracy versus the epochs. [2]
    - Visualise your results, by saving to a PNG file "result.png", a montage of 36 test images with captions indicating the ground-truth and the predicted classes for each. [3]

Task 3 Ablation using Cross-Validation

Again, using the Image Classification tutorial, this task investigates the impact of one of the following three modifications to the original network, using cross-validation. To evaluate a modification, an ablation study can be used by comparing the performance before and after removing the modification.

- Difference between training with and without the Cutout data augmentation algorithm implemented in Task 2.
- Difference between using SGD with momentum (as in "train_pt.py") and Adam optimiser (as in "train_tf.py").
- Difference between using ReLU and leaky ReLU (with a negative slope alpha=0.1), as activation functions throughout the network.
- Indicate your choice. [1]

- Implement a task script "task.py", under folder "task3", completing the following:
    - Split the data into development set and holdout test set. [1]
    - Implement a 3-fold cross-validation scheme, using the development set. [5]
    - Print data set summary every time the random split is done. [2]
    - Design at least one metric, other than the loss, on validation set, for monitoring during training. [5]
    - Run the cross-validation scheme for each of the two networks or training strategies (with and without the one modification). [6]
    - Report a summary of loss values, speed, metric on training and validation. [4]
    - Train two further models using the entire development set and save the trained models. [4]
    - Submit these two trained models within the task folder. [2]
    - Report a summary of loss values and metrics on the holdout test set. Compare the results with those obtained during cross-validation. [5]