**Case Study 6 (Supplemental)**

Given the following paper: <https://arxiv.org/pdf/1402.4735.pdf>

And supplemental publication: <https://www.nature.com/articles/ncomms5308>

**Build a replica Neural Network with the paper’s architecture using TensorFlow.**

If possible begin to train on the data located here: <https://archive.ics.uci.edu/ml/datasets/HIGGS>.

How close can you get to the original results? To facilitate quicker training, you may increase the batch size temporarily (this has a small impact on final result, but can speed up your calculations significantly). **You do not need to train a final result using the paper’s parameters, only the code for your model is required in your final submission.**

**Include in your report:**

1. Your replica neural network based on the paper’s architecture
2. Any recommendations or modifications you’d make to the approach taken by the researchers (i.e., state any proposed changes along with the expected improvements or impact) - What are standard practices now versus when this paper was written?
3. How would you quantify if your result duplicated the paper’s (hint: what evaluation metric did the researchers use)?

***Relevant Paper Excerpts***

*We have published a dataset containing* ***11 million simulated collision events*** *for benchmarking machine learning classification algorithms on this task, which can be found in the UCI Machine Learning Repository at* ***archive.ics.uci.edu/ml/datasets/HIGGS.***

We explored the use of deep neural networks as a practical tool for applications in high-energy physics. Hyperparameters were chosen using a subset of the HIGGS data consisting of 2.6 million training examples and 100,000 validation examples. Due to computational costs, this optimization was not thorough, but included combinations of the pre-training methods, network architectures, initial learning rates, and regularization methods shown in Supplementary Table 1.

We selected a five-layer neural network with 300 hidden units in each layer, a learning rate of 0.05, and a weight decay coefficient of 1 × 10−5.

We produced Receiver Operating Characteristic (ROC) curves to illustrate the performance of the classifiers. Our primary metric for comparison is the area under the ROC curve (AUC), with larger AUC values indicating higher classification accuracy across a range of threshold choices.

An additional boost in performance is obtained by using the dropout training algorithm, in which we stochastically drop neurons in the top hidden layer with 50% probability during training.

In training the neural networks, the following hyperparameters were predetermined without optimization. Hidden units all used the tanh activation function. Weights were initialized from a normal distribution with zero mean and standard deviation 0.1 in the first layer, 0.001 in the output layer, and 0.05 all other hidden layers. Gradient computations were made on mini-batches of size 100. A momentum term increased linearly over the first 200 epochs from 0.9 to 0.99, at which point it remained constant. The learning rate decayed by a factor of 1.0000002 every batch update until it reached a minimum of 10−6,

Additional (helpful) links:

* Calculating the number of parameters in networks: <https://towardsdatascience.com/counting-no-of-parameters-in-deep-learning-models-by-hand-8f1716241889>
* Understanding binary cross-entropy / log-loss: <https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a>
* Epoch vs. Batch vs. Iteration: <https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9>
* Illustrative comparison of the biological vs. artificial neural network (and a wealth of other important terms): <https://cs231n.github.io/neural-networks-1/#arch>
* A brief intro into dropout: <https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5>
* A lengthy overview of optimizers: <https://ruder.io/optimizing-gradient-descent/>