**Introduction**

Overview/Explanation of task

**Data Preparation and Exploratory Analysis**

For example, though the Large Hadron Collider produces approximately 10^11 collisions per hour, approximately 300 of these collisions result in a Higgs boson, on average. Therefore, good data analysis depends on distinguishing collisions which produce particles of interest (signal) from those producing other particles (background).

\*\*The data has been produced using Monte Carlo simulations. The first 21 features (columns 2-22) are kinematic properties measured by the particle detectors in the accelerator. The last seven features are functions of the first 21 features; these are high-level features derived by physicists to help discriminate between the two classes.

Published dataset containing 11 million simulated collision events for benchmarking ML classification algorithms, found on UCI Machine Learning Repository

Data sets were nearly balanced with 53% positive examples in Higgs data

**Replicating the Original Model**

[Define Neural Network?]

Hyperparameters chosen with subset of Higgs data consisting of 2.6M training examples and 100,000 validation examples. Computational costs did not allow for thorough optimization but included combinations of pre-training methods, network architectures, initial learning rates, regularization methods. Hyperparameter optimization was performed using the full set of Higgs features…classifiers were tested on 500,000 simulated examples generated from same Monte Carlo procedures as training sets.

Optimal Model from Paper

* Five-layer Neural Network with 300 hidden units in each layer
* **Learning Rate** of 0.05
* **Weight Decay** Coefficient 1 x 10-5
* Predetermined without Optimization
  + Hidden units all used **tahn activation function**
  + Weights initialized from normal distribution with 0 mean and standard deviation 0.1 in first layer, 0.001 in output layer, and 0.05in all other hidden layers
  + **Mini-batches** of size 100 to compute gradients
  + **Momentum** increased linearly over first 200 epochs from 0.9 to 0.99, then remained constant
  + **Learning Rate** decayed by 1.0000002 every batch update until it reached a minimum of 10-6
* Training ended when **momentum** reached maximum value and minimum error on validation set of 500,000 examples had not decreased by more than a factor of 0.00001 over 10 epochs. Early stopping prevented overfitting and each NN trained over 200-1000 epochs.
* When training with **dropout,** increased learning rate decay to 1.0000003 and ended training when momentum reached maximum value and error on validation set had not decreased for 40 epochs
* Inputs standardized over entire train/test set with mean 0 and standard deviation of 1, except for features with values strictly greater than 0 (scaled for mean value of 1)
* 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.

Original model built with PyLearn2; [BRIEF EXPLAIN WHAT IT IS-NW]

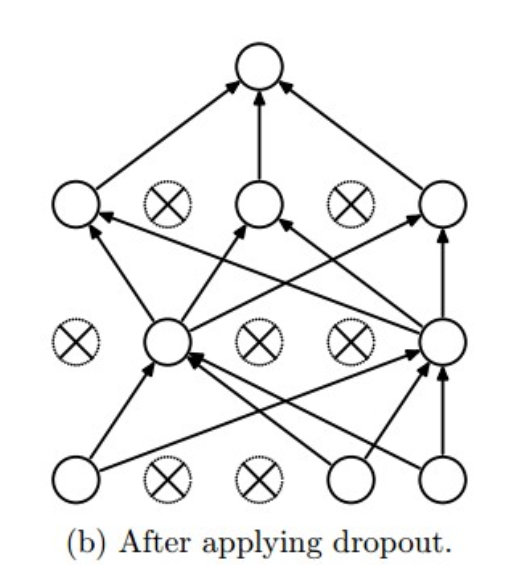
**Parameters (subsection)**

**Learning Rate** – a configurable hyperparameter that controls how quickly/slowly a neural network learns a problem, more specifically it controls how much to change the weights to correct for error during each iteration; a large learning rate allows model to train faster but a cost, where smaller learning rates may yield a better model, requiring more training epochs and smaller batch sizes.

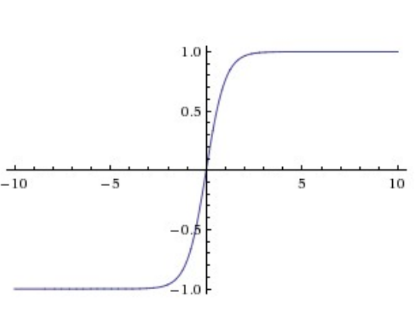
* Math: gradient descent algorithm multiples learning rate by gradient, for example value of 0.1 will update weight 10% of the amount it could be updated
* Range: 0.0 to 1.0, traditional default 0.1 or 0.01

**Learning Rate Decay** – how learning rate changes over training epochs; learning rate decay can be designed where large weight changes happen at the beginning of the process and smaller, fine-tune changes toward the end; another strategy is to decay over a fixed number of training epochs at a small, constant value.

**Drop Out** – form of regularization to minimize overfitting; technique that randomly removes/inactivates neurons at each training step, which forces remaining neurons to be more independent because they learn rated not in conjunction/cooperation with neighboring neuron; roughly doubles the number of iterations required for convergence but training time for each epoch is less.



**Activation Functions** – aka transfer functions; functions that take a weighted sum of all inputs from previous layer and generates an output value for the next layer; for each node, it defines the output of the node given an input or set of inputs.

* Tanh activation – a non-linear activation function that outputs values between -1.0 and 1.0 and the center falls around 0; limitations are that it can have limited sensitivity and is prone to saturation in larger, more layered networks due to vanishing gradients.

**Momentum** – improves the speed of optimization in concert with step size by helping SGD algorithm navigate in relevant/optimal directions; in other words, it adds inertia to the algorithm update process to continue moving in the optimal direction; best to begin with smaller momentum and then increase after passing through larger gradients – momentum can cause learning process to miss or oscillate around the minima.

* Math: adds a fraction of the direction of the previous step to a current step
* Range: 0.0 to 1.0, traditional default 0.9, 0.99 or 0.5

**Types of Gradient Descent**

* Batch Gradient Descent – batch size is set to total number of examples in the training dataset
* Stochastic Gradient Descent – batch size is set to one
* Minibatch Gradient Descent – batch size is set to more than one and less than the total number of examples in the training dataset.

**Batch/batch size** – the number of data points/observations used in one iteration (one gradient update) of model training and dictates the number of training observations to be “learned” before updating internal parameters; generally, a larger batch involves more training examples, thus yielding a more stable learning process and accurate estimate. For example, a batch size of 32 means that 32 samples from the training dataset will be used to estimate error gradient before the model weights are updated. *NOTE: batch size and number of batches are different.*

**Epoch** – represents the number of completes passes through the training dataset during the learning process, where the learning algorithm loops through a fixed number of epochs and within each, updates the network for each row in the training data; one epoch means that each sample in the training dataset has updated internal parameters; calculated as N / batch size training iterations, where N is the total number of examples.

“You can think of a for-loop over the number of epochs where each loop proceeds over the training dataset. Within this for-loop is another nested for-loop that iterates over each batch of samples, where one batch has the specified “batch size” number of samples.”

Iteration – number of batches needed to complete on epoch

Dataset with 200 samples

Batch size = 5

Epochs = 1000

* Dataset will be divided into 40 batches, each with 5 samples, model weights will update after each batch of 5 samples
* One epoch will involve 40 batches/40 updates to model
* 1000 epochs, model will be exposed/passed through whole data 1000 times, total of 40,000 batches during entire training process

**Comparison (subsection)**

Produced Receiver Operating Characteristic (ROC) curves to illustrate performance, metric for comparison was area under the ROC curve (AUC), with larger values indicating higher classification accuracy across a range of threshold choices. Directly connected to classification accuracy; standard in ML, correlated to other metrics,

**Recommendations**

* Try different activation functions, consider ReLU, others?
* Try different training/test split, 99% train vs 1% test – we would suggest more traditional 80% train vs 20% test
* Try Adam --

Sigmoid activation – S shape, logistic function, take any input and produce result between 0 and 1, cannot be used with many layers due to vanishing gradients, nonlinear activation function, large negatives become 0/large positives become 1, drawbacks – sigmoids saturate, kill gradients; sigmoid outputs are not 0-centered,

Softmax activation –

ReLU activation – rectified linear activation function, piecewise linear function that outputs the input directly if positive (otherwise, output 0). Default activation, easy train, better performance, activation is threshold at 0, can accelerate SGD, implemented by simply thresholding matrix of activations at 0, can be fragile where weights could update in a way for neuron to not activate again (Leaky ReLU attempts to fix dying problem)

**Conclusion**

**REMAINING/MISC NOTES:**

Weight Initialization – do not make all initialization 0

Weight decay -

Backpropagation Algorithm – method for training weights, supervised learning method for multilayer feed-forward networks, calculate error for each output neuron to get error signal (input) to propagate backward through network

Forward-propagation – calculate output from neural network by propagating an input signal through each layer until the output layer outputs its values, generate predictions during training that will need to be correct and also used to make predictions on new data

Weight Regularization – (large network weights may be indicative of instability, where small changes in input lead to large changes in output…can be sign of overfitting). Method to keep weights small, reduce overfitting, improve model generalization, vector norm of weights is calculated per layer, rather than for whole network, can use both L1 and L2

* Calculate the sum of the absolute values of the weights, called L1. Encourages weights to 0.0, resulting in more sparse weights (weights with more 0.0 values)
* Calculate the sum of the squared values of the weights, called L2. More nuanced, penalizing larger weights more, but results in less sparse weights, more traditional, referred to as “weight decay”

\*\*more layers and more nodes tends to overfit training data

**References**

<https://developers.google.com/machine-learning/glossary>

<https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/>

<https://machinelearningmastery.com/what-is-deep-learning/>

<https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/>

<https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/>

<https://en.wikipedia.org/wiki/Activation_function>

<https://machinelearningmastery.com/weight-regularization-to-reduce-overfitting-of-deep-learning-models/>

<https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/>

<https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/>

<https://cs231n.github.io/neural-networks-2/>

<https://arxiv.org/pdf/1206.5533.pdf>

<http://jmlr.org/papers/volume15/srivastava14a.old/srivastava14a.pdf>

<https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9>

<https://cs231n.github.io/neural-networks-1/#arch>

<https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5>

<https://ruder.io/optimizing-gradient-descent/>