Training and Evaluating an MLP using SGD for Binary Classification

George Michailidis

gmichail@ucla.edu

STAT 102B

Binary Classification Problem

- Given dataset $\{(x_i, y_i)\}_{i=1}^N$, where $x_i \in \mathbb{R}^p$, $y_i \in \{0, 1\}$
- Goal: Train a model $f(x; \theta)$ to predict $\mathbb{P}(y_i = 1 | x_i)$
  Since it is a binary classification problem, $\mathbb{P}(y_i = 0 | x_i) = 1 - \mathbb{P}(y_i = 1 | x_i)$
- Compare the following two models:
  - Logistic regression
  - Multilayer Perceptron (MLP)

Binary Classification Evaluation Basics - I

Compare on a test data set of size N_{test}

- True data labels: $y_i \in \{0, 1\}, \; i = 1, \cdots, N_{\text{test}}$
- Predicted data labels: $\hat{\hat{y}}_i \in \{0, 1\}, \; i = 1, \cdots, N_{\text{test}}$

However, both logistic regression and MLP predict

$$\hat{y}_i = \mathbb{P}(y_i = 1 | x_i)$$

To obtain the required binary $\hat{\hat{y}}_i$, we employ a threshold $t \in (0, 1)$;

$$\hat{\hat{y}}_i = \begin{cases} 1 & \text{if } \hat{y}_i > t \\ 0 & \text{otherwise} \end{cases}$$

The common default is $t = 0.5$

Binary Classification Evaluation Basics - II

Confusion Matrix

		Predicted	
		Positive ($\hat{\hat{y}}_i = 1$)	Negative ($\hat{\hat{y}}_i = 0$)
Actual	Positive ($y_i = 1$)	True Positive (TP)	False Negative (FN)
	Negative ($y_i = 0$)	False Positive (FP)	True Negative (TN)

Different metrics are computed from the confusion matrix

Evaluation Metrics - I

Accuracy

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Interpretation: Proportion of total predictions that are correct

Limitation: Can be misleading for imbalanced classes; e.g., if 95% of observations are negative, a model that always predicts negative will still be 95% accurate

Metrics - II

Precision and Recall

- **Precision:**
$$\text{Precision} = \frac{TP}{TP + FP}$$
Measures how many predicted positives are actually positive
- **Recall (Sensitivity or TPR):**
$$\text{Recall} = \frac{TP}{TP + FN}$$
Measures how many actual positives were correctly predicted

Metrics - III

F1 Score

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

- Harmonic mean of precision and recall
- Useful when there's a class imbalance and a balance between precision and recall is desired

Metrics - IV

False Positive Rate (FPR) and True Positive Rate (TPR)

- **True Positive Rate (TPR) / Recall:**

$$TPR = \frac{TP}{TP + FN}$$

- **False Positive Rate (FPR):**

$$FPR = \frac{FP}{FP + TN}$$

Metrics - IV

Specificity

$$\text{Specificity} = \frac{TN}{TN + FP}$$

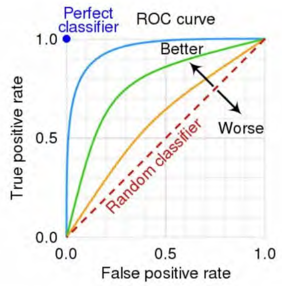
- Also called the True Negative Rate (TNR)
- Measures how well the model identifies negative instances

Which metric to use and when?

- **Imbalanced Classes:** Avoid accuracy; use precision, recall, or F1
- **Medical Tests:** High recall (few false negatives) is often prioritized
- **Spam Detection:** High precision (few false positives) may be prioritized

Receiver Operating Characteristic (ROC) Curve

- Plots **TPR** vs. **FPR** at various classification thresholds $t \in (0,1)$
- Shows the tradeoff between sensitivity and specificity
- A good classifier pushes the curve towards the top-left corner
  This task obviously depends on how "hard" the classification problem is



AUC - Area Under the ROC Curve

- AUC measures the entire two-dimensional area underneath the entire ROC curve
- **Range:** 0.5 (random guessing) to 1 (perfect classifier)
- **Interpretation:** The AUC is the probability that a randomly chosen positive instance is ranked higher (i.e., receives a higher score) than a randomly chosen negative instance

More on the interpretation of AUC - I

Setup of the "experiment":

- Take one random sample from the positive class
- Take one random sample from the negative class
- Use the classification model to assign scores/probabilities to both samples

Question: What is the probability that the positive sample (class 1) gets a higher score than the negative sample (class 0)?

Answer: This probability equals the AUC!

More on the interpretation of AUC - II

The AUC can be mathematically written as:

$$\text{AUC} = \mathbb{P}(s(x^+) > s(x^-)),$$

where $s(\cdot)$ is the score assigned by the classification model, x^+ denotes a positive sample/instance and x^- a negative sample/instance

This interpretation holds under the assumption that the classifier outputs a real-valued score; i.e, the probability of predicting the positive class based on the covariate information

Illustration of MLP on a Binary Classification Problem - I

- Sample size $n = 10000$, split into
  - training set $n_{\text{training}} = 6000$
  - validation set $n_{\text{val}} = 2000$
  - test set $n_{\text{training}} = 2000$
- $p = 100$ covariates, with average correlation between them > 0.4
- Response $y_i, i = 1, \cdots, n$ generated according to a logistic regression model with c% label noise
- Competing models
  - Logistic regression
    uses the training data set to estimate the model parameters and the test one to evaluate its performance
  - MLP
    uses the training data set to estimate the model parameters (refer to Lecture 4.1), the validation data set to tune the mini-batch size and the hidden layer dimension and the test data set to evaluate the performance of the best MLP model

Illustration of MLP on a Binary Classification Problem - II

Details on generating the response $y_i \in \{0, 1\}$

It is generated according to a logistic regression model:

$$\mathbb{P}(y_i = 1 \mid x_i) = \sigma(x_i^\top \beta) = \frac{1}{1 + \exp(-x_i^\top \beta)}, \tag{1}$$

where $x_i \in \mathbb{R}^p$ is the covariate vector for the i-th observation and $\beta \in \mathbb{R}^p$ is the true coefficient vector

To simulate label noise, we flip the class labels for a random subset of the observations

Specifically, for a given noise level $c \in [0, 1]$, we randomly select $\lfloor cn \rfloor$ indices $\mathcal{I}_{\text{flip}} \subset \{1, \ldots, n\}$, and modify the corresponding labels as follows:

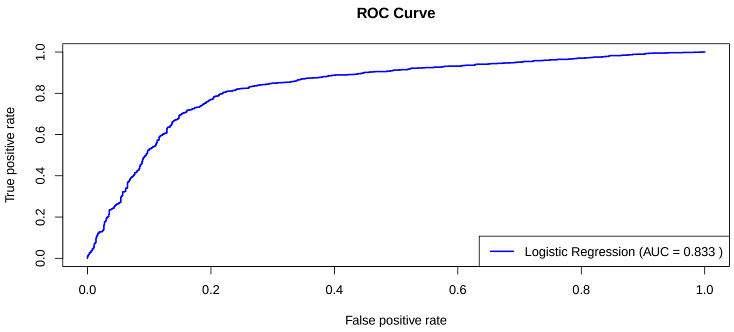
$$y_i \leftarrow 1 - y_i, \quad \text{for all } i \in \mathcal{I}_{\text{flip}}. \tag{2}$$

The higher the proportion $c\%$ the harder the classification task becomes

The gold standard: logistic regression

Given that the data are generated according to a logistic regression model, this model will set the gold standard for performance

Label noise proportion $c = 0.15$ (i.e., 15% of the labels are randomly flipped)



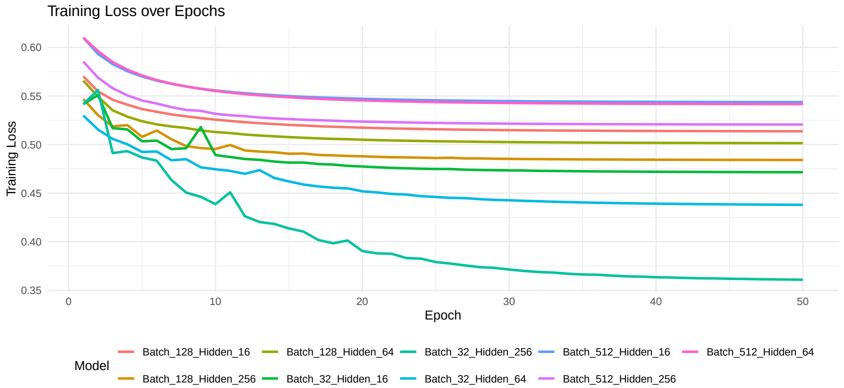
Hyperparameters for MLP

- Epochs 50
- Decreasing step size schedule

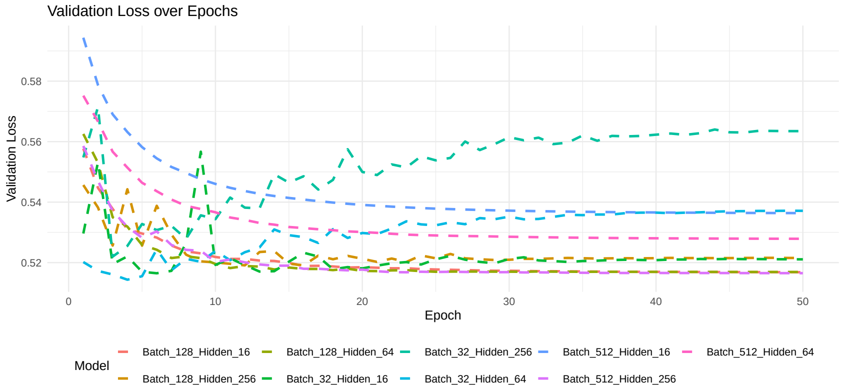
$$\eta_0 = 1, \eta_e = \eta_0 d^{1-\text{ep}},\ d = 0.7,\ \text{ep} \in \{1, \cdots, 50\}$$

  The step size is kept fixed for all backward pass mini-batch gradient updates within an epoch
- hidden dimension $q \in \{32, 128, 512\}$
- mini-batch size $s \in \{16, 64, 256\}$

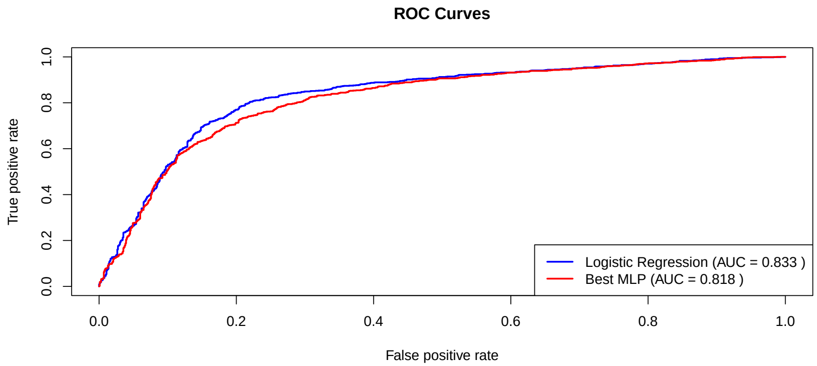
MLP Performance - I



MLP Performance - II



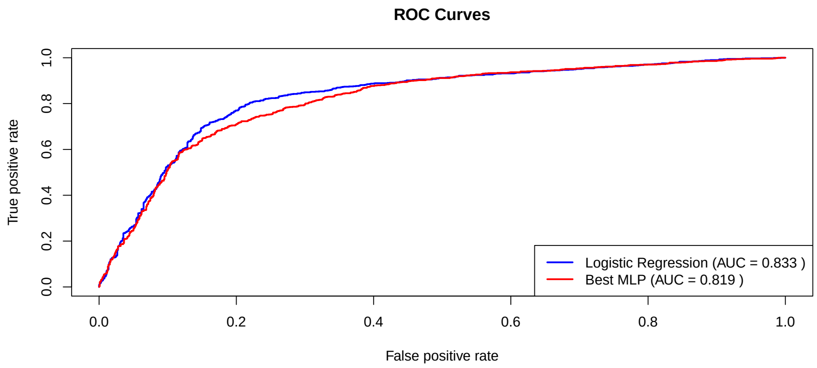
MLP Performance - III



Remark: different schedules for the step size could improve the performance of the best MLP by a small margin and make its AUC equal to that of logistic regression

Performance of full GD

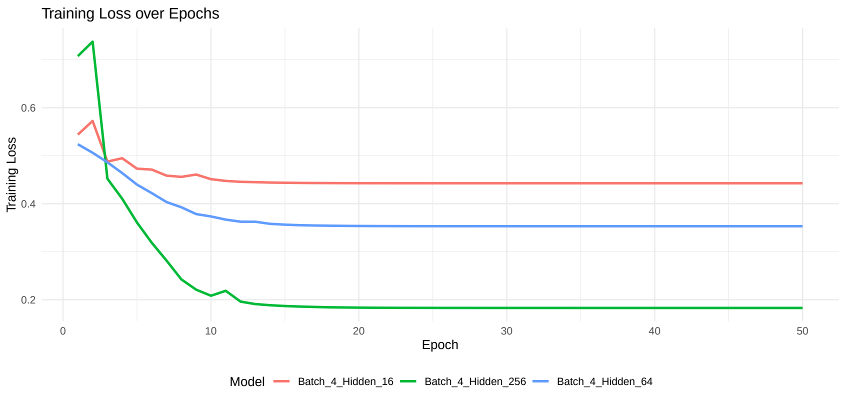
Set $s = n_{\text{training}}$



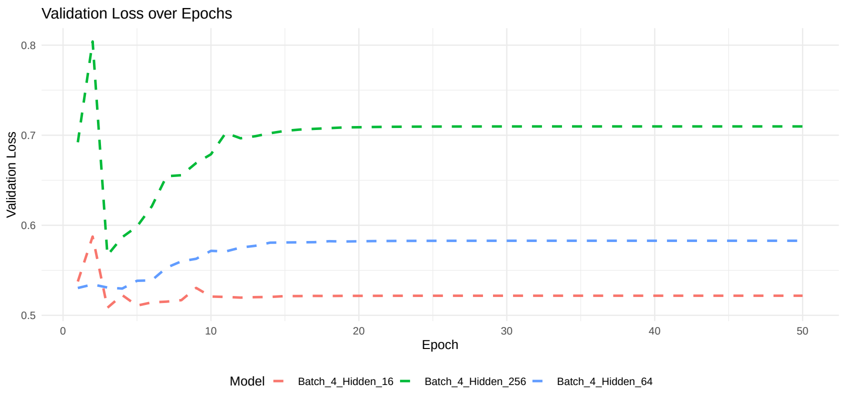
It can be seen that a full GD achieves the same performance as SGD with significantly smaller mini-batches and is much more expensive to run

Performance of SGD with small mini-batch - I

Set $s = 4, \eta_0 = 0.1, d = 0.7$

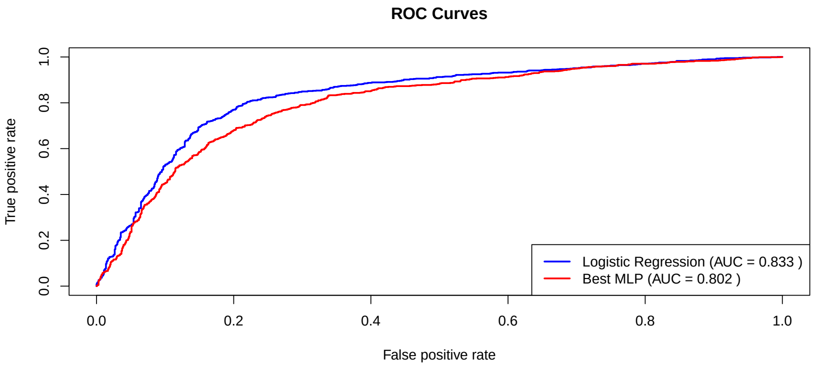


Performance of SGD with small mini-batch - II



Remark: The behavior of the training and validation losses indicates that for large hidden dimension q and small mini-batch s, the MLP is overfitting the training data

Performance of SGD with small mini-batch - III

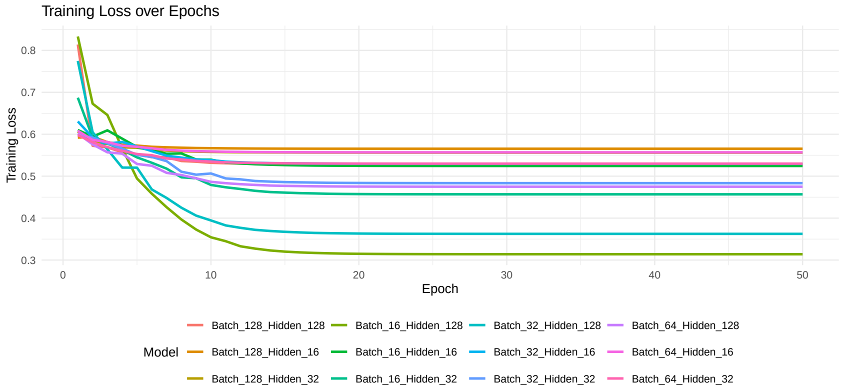


A different data generation mechanism - I

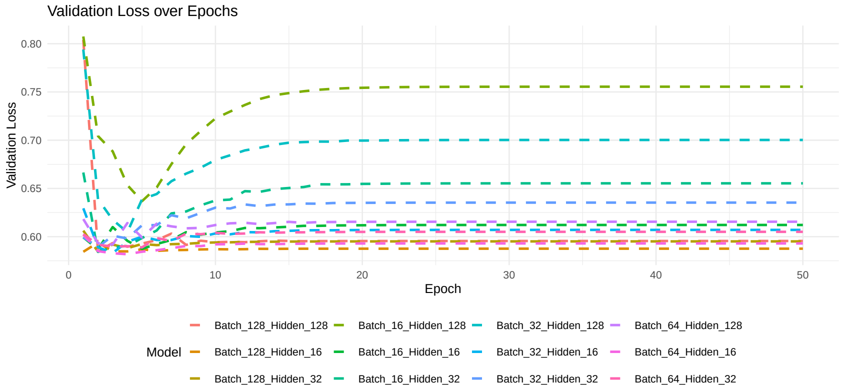
In the previous experiment, the data set was generated by a logistic regression model and with careful tuning of the hyperparamters the MLP matched the performance of logistic regression

Next, the data are generated by a different mechanism (that of classical Linear Discriminant Analysis - essentially the $y = 0$ and $y = 1$ data are distributed according to multivariate normal distributions with different means and common covariance matrix) and the performance of logistic regression is compared to the performance of the MLP

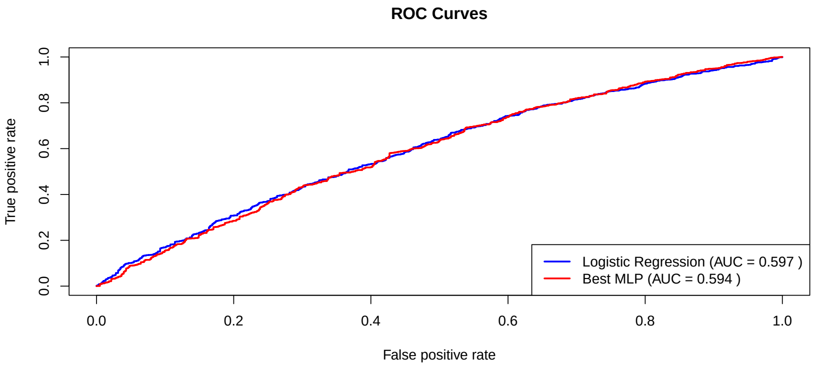
MLP Performance - I



MLP Performance - II



MLP Performance - III



Scaling up and automating neural network model training - I

The presentation thus far illustrated the components required to train a single layer MLP for a binary classification problem from scratch; namely,

- Select an appropriate loss function
- Set up the neural network architecture and the model parameters; i.e., hidden layer, activation functions
- Breaking the training process into a forward and backward pass
- Calculate the gradient of the loss function with respect to the model parameters based on the backward pass
- Select an optimization algorithm (e.g., SGD)
- Tune neural network hyperparameters based on validation data set

Scaling up and automating neural network model training - II

It can be seen that extending the MLP model from a single hidden layer to a multi-layer architecture becomes cumbersome

To that end, advanced libraries/packages have been developed in `Python` and `R`

PyTorch: Deep Learning in `Python`

- **PyTorch** is an open-source deep learning framework developed by Facebook's AI Research lab
- It is widely used for:
  - Building and training neural networks
  - Automatic differentiation with `autograd`
  - High performance through GPU acceleration (via CUDA)
- Key Components:
  - `torch.Tensor`: The primary data structure (similar to NumPy arrays, with GPU support)
  - `torch.nn`: Module for building neural networks
  - `torch.optim`: Gradient-based optimization algorithms (e.g., SGD, ADAM, ADAM-W)
  - `torch.autograd`: Autodiff engine for backpropagation
- Highly modular, flexible, and Pythonic—favored in both research and production

torch for R: Modern Deep Learning in R

- **torch** is an R package that brings PyTorch's functionality to R users
- Developed and maintained by the RStudio AI team, providing:
  - Native access to tensors, autograd, and neural networks
  - Training models on CPUs and GPUs directly from R
  - Compatibility with R ecosystem (e.g., `ggplot2`, `dplyr`, etc.)
- Key Functions:
  - `torch_tensor()`: Create tensors from R arrays/vectors
  - `nn_module()`: Define neural network models
  - `optimizer_sgd()`, `nnf_relu()`, `nnf_sigmoid()`, etc.: Functional API
- Enables seamless model training, evaluation, and deployment within R

The need for Automatic Differentiation

A critical and challenging component in training an MLP (or any other neural network for that matter) is the backward pass that requires computing the gradient of the loss with respect to the various model parameters

The logic of how to do these computations is straightforward (repeated application of the chain rule), but in any but the simplest neural network architectures tedious

What is Automatic Differentiation (AD)?

- A technique to numerically evaluate the derivative of a function specified by a computer program
- Key idea: Apply the chain rule repeatedly to elementary operations
- Differs from:
  - Symbolic differentiation (unwieldy expressions)
  - Numerical differentiation (suffers from round-off errors)
- Two main modes:
  - Forward mode: Propagates derivatives forward through computation
  - Reverse mode: Propagates derivatives backward (backpropagation)

AD will be presented in detail in the next Lecture

Next, the results of a single and a 2-layer MLP are shown for two data generated mechanisms for the binary classification problem

1. Data generated according to a logistic regression model
2. Data generated according to a nonlinear mechanism

Data generated based on a logistic regression model - I

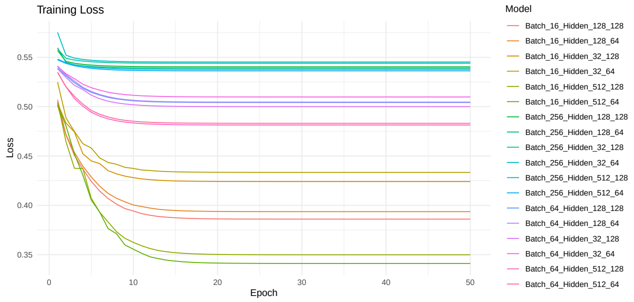
The results for a single layer MLP based on `PyTorch/Torch`, using the hyperparamters in slide 16 are identical to those obtained with the handcrafted code

For a 2-layer MLP the following combinations of hidden dimensions were examined:

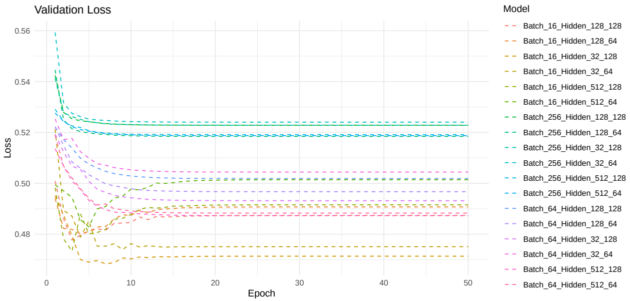
- $q_1 \in \{32, 128, 512\}$
- $q_2 \in \{64, 128\}$

All other hyperparameters were not altered

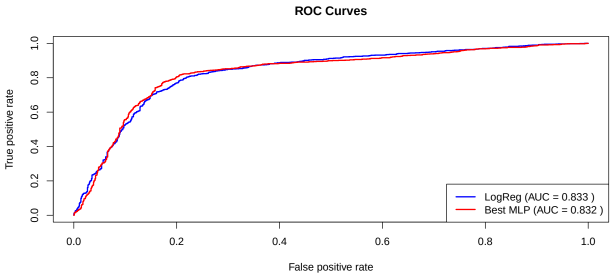
2-layer MLP Performance - I



2-layer MLP Performance - II



2-layer MLP Performance - III

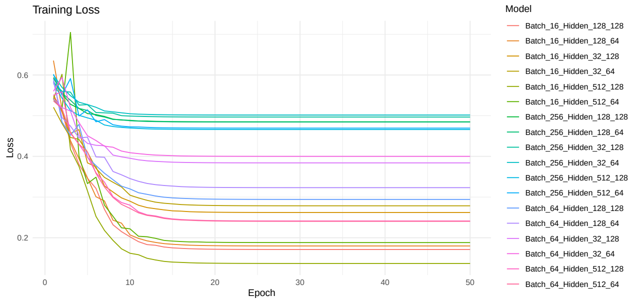


A Nonlinear Data Generating Mechanism

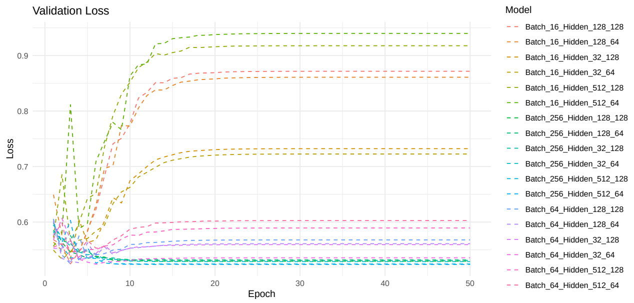
In this case, the data set is generated so that 90% of the $p = 100$ predictors impact $\mathbb{P}(y_1 = 1|X_i)$ in a linear manner and the remaining 10% in a nonlinear manner

Further, a 15% label noise is applied and the remaining hyperparamters are as in the previous 2-layer example

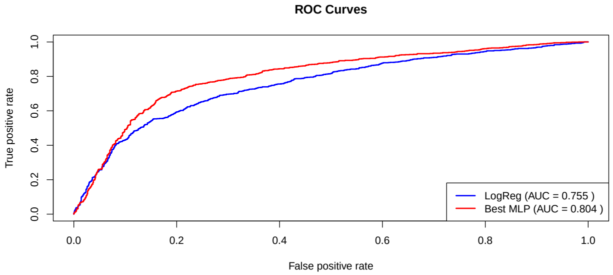
2-layer MLP Performance on nonlinear data - I



2-layer MLP Performance on nonlinear data - II



2-layer MLP Performance on nonlinear data - III


ROC Curves
True positive rate
False positive rate
LogReg (AUC = 0.755)
Best MLP (AUC = 0.804)