Neural Networks

This project implements a neural network framework from first principles, beginning with linear regression and extending to more complex architectures.

Linear Regression

Neural networks begin with linear regression. Let $x \in \mathbb{R}^d$ be our input vector with dimension d, $W \in \mathbb{R}^{p \times d}$ a weight matrix mapping to p outputs, and $b \in \mathbb{R}^p$ a bias vector. The linear transformation computes:

$$\hat{y} = \boldsymbol{W}\boldsymbol{x} + b$$

This computation forms the core of our linear layer. Learning occurs through gradient descent, which requires computing how changes in weights affect the loss. Let $\mathcal{L}$ be our loss function, mean square error. The weight gradients are:

$$\nabla_{\boldsymbol{W}} \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \boldsymbol{W}}$$

For a linear layer, $\frac{\partial \hat{y}}{\partial \boldsymbol{W}_{ij}} = x_j$ for output $\hat{y}_i$, which in matrix form is $\frac{\partial \hat{y}}{\partial \boldsymbol{W}} = x^T$.

For MSE, defined as:

$$\mathcal{L}(y, \hat{y}) = \frac{1}{n} \sum_{i=0}^{n} (y_i - \hat{y}_i)^2$$

This gradient is:

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = -\frac{2}{n}(y - \hat{y})$$

Our implementation seperates these concerns into distinct modules:

- A `Linear` module handling forward computation and gradient calculations ;
- A `MSELoss` module computing the loss and its gradient ;
- The base `Module` class defining interfaces for all layers, given by the handout.

The training process iteratively applies gradient descent:

Gradient descent for linear regression

Input: $\boldsymbol{x}$ examples, y labels, η, $\mathcal{E}$
Ouput: $\boldsymbol{W}, b$
$\boldsymbol{W} \xleftarrow{⚀⚀} \mathbb{R}^{p \times d}, b \xleftarrow{⚀⚀} \mathbb{R}^p$
For $e \in \{1, ..., \mathcal{E}\}$
  $\hat{y} \leftarrow \boldsymbol{W}\boldsymbol{x} + b$
  $\mathcal{L} \leftarrow \frac{1}{n} \sum (y_i - \hat{y}_i)^2$
  $\nabla_{\boldsymbol{W}} \mathcal{L} \leftarrow -\frac{2}{n}(y - \hat{y})\boldsymbol{x}^T$
  $\nabla_b \mathcal{L} \leftarrow \sum -\frac{2}{n}(y - \hat{y})$
  $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \nabla_{\boldsymbol{W}} \mathcal{L}$
  $b \leftarrow b - \eta \nabla_b \mathcal{L}$

Results analysis. Fig. 1 shows our linear regression results. The loss curve displays the expected pattern: rapid initial decrease followed by gradual convergence. Within 200 epochs, most improvement occurs, with minimal gains thereafter. The bottom plots contrast the best and worst models. Both capture linear relationships but with different slopes. The data points cluster tightly around both lines, showing both models learned useful patterns performance differences. This variability stems from random initialization and dataset generation.

These results indicate our implementation correctly applies the gradient descent algorithm. The smooth convergence indicates proper gradient computation and parameter updates.

Linear and non-linear classification

Neural networks can also be used for classification tasks. We can adapt our linear regression model to classify data points into k classes. The

output layer is a softmax layer, which normalizes the output to a probability distribution over the classes. The loss function is categorical cross-entropy, defined as:

$$\mathcal{L}(y, \hat{y}) = -\frac{1}{n} \sum_{i=0}^{n} \sum_{j=0}^{k} y_i j \log(\hat{y}_j)$$

where $y_i j$ is 1 if the i-th example belongs to class j, and 0 otherwise. The gradient of the loss with respect to the output is:

$$\nabla_{\hat{y}} \mathcal{L} = \hat{y} - y$$

The training process is similar to linear regression, but we use the softmax layer and categorical cross-entropy loss. The algorithm is as follows:

Gradient descent for classification

Input: $\boldsymbol{x}$ examples, y labels, η, $\mathcal{E}$
Ouput: $\boldsymbol{W}, b$
$\boldsymbol{W} \xleftarrow{\square\square} \mathbb{R}^{p \times d}, b \xleftarrow{\square\square} \mathbb{R}^{p}$
For $e \in \{1, ..., \mathcal{E}\}$
  $\hat{y} \leftarrow \text{softmax}(\boldsymbol{W}\boldsymbol{x} + b)$
  $\mathcal{L} \leftarrow -\frac{1}{n} \sum (y_i \log(\hat{y}_i))$
  $\nabla_{\boldsymbol{W}} \mathcal{L} \leftarrow \nabla_{\hat{y}} \mathcal{L} \boldsymbol{x}^T$
  $\nabla_b \mathcal{L} \leftarrow \sum \nabla_{\hat{y}} \mathcal{L}$
  $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \nabla_{\boldsymbol{W}} \mathcal{L}$
  $b \leftarrow b - \eta \nabla_b \mathcal{L}$

Results analysis.

Fig. 2 shows the results of our linear classification task. The model successfully separates the two classes, with a clear decision boundary. The loss curve indicates that the model converges quickly, with most improvement occurring in the first few epochs. The bottom plot shows the best and worst models, which both capture the underlying structure of the data. Fig. 3 shows the results of our non-linear classification task on the XOR problem. The model successfully separates the two classes, with a clear decision boundary. Although the loss curve converge a little bit slowly, it is still pretty fast, with most improvement occurring in the first few epochs. The bottom plot shows the best and worst models, which both capture the underlying structure of the data.

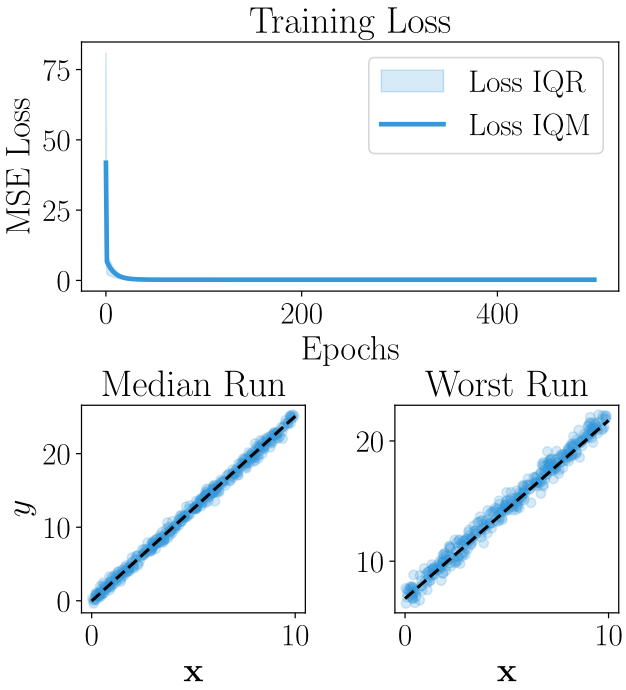

Fig. 1. – This figure displays training results from 100 linear regression trials ($n = 100$) with 200 samples per run ($\sigma = 200$), learning rate 0.01 ($\eta = 0.01$), and 1000 epochs ($\mathcal{E} = 1000$). The top plot shows the interquartile mean loss and Q1-Q3 range during training, while the bottom plots contrast the best and worst performing models from the ensemble.

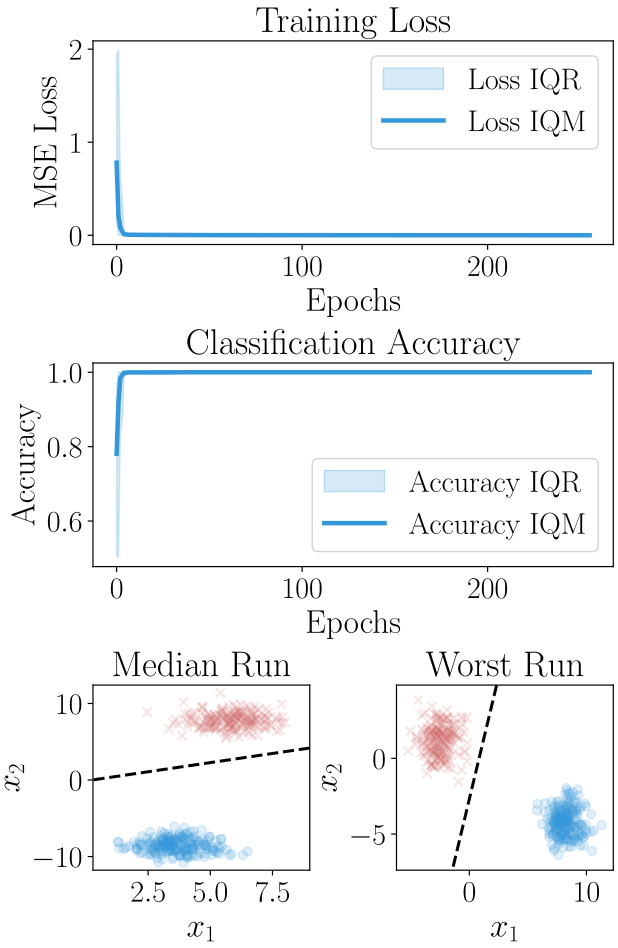

Fig. 2. – This figure displays training results from 100 linear classification trials ($n = 100$) with 200 samples per run ($\sigma = 200$), learning rate 0.01 ($\eta = 0.01$), and 1000 epochs ($\mathcal{E} = 1000$). The top plot shows the interquartile mean loss and Q1-Q3 range during training, while the bottom plots contrast the best and worst performing models from the ensemble.

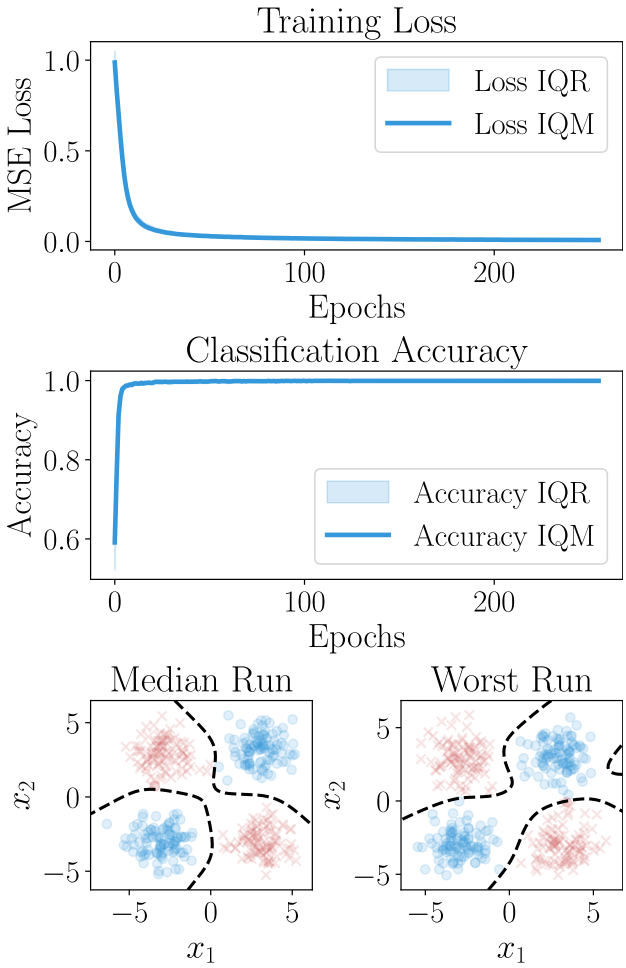

Fig. 3. – This figure displays training results from 100 non-linear classification trials ($n = 100$) with 200 samples per run ($\sigma = 200$), learning rate 0.01 ($\eta = 0.01$), and 1000 epochs ($\mathcal{E} = 1000$). The top plot shows the interquartile mean loss and Q1-Q3 range during training, while the bottom plots contrast the best and worst performing models from the ensemble.