Lab: Optimisation

ACTL3143 & ACTL5111 Deep Learning for Actuaries

As you have learned, a neural network consists of a set of weights and biases, and the network learns by adjusting these values so that we minimise the network's loss. Mathematically, we aim to find the optimum weights and biases $(\boldsymbol{w}^*, \boldsymbol{b}^*)$:

$$(\boldsymbol{w}^*, \boldsymbol{b}^*) = \arg\min_{\boldsymbol{w},\boldsymbol{b}} \mathcal{L}(\mathcal{D}, (\boldsymbol{w}, \boldsymbol{b}))$$

where $\mathcal{D}$ denotes the training data set and $\mathcal{L}(\cdot, \cdot)$ is the user-defined loss function.

Gradient descent is the method through which we update the weights and biases. We introduce two types of gradient descent: **stochastic** and **batch**.

- **Stochastic** gradient descent updates the weights and biases once for each observation in the data set.
- **Batch** gradient descent updates the values repeatedly by averaging the gradients across all the observations.

Example: Batch Gradient Descent for Linear Regression

Notation:

- $\mathcal{L}(\mathcal{D}, (\boldsymbol{w}, \boldsymbol{b}))$ denotes the loss function.
- $\hat{y}(\boldsymbol{x}_i)$ denotes the predicted value for the ith observation $\boldsymbol{x}_i \in \mathbb{R}^{d_{\boldsymbol{x}}}$, where $d_{\boldsymbol{x}}$ represents the dimension of the input.
- N denotes the batch size.

The true model:

$$\boldsymbol{w}_{\text{True}} = \begin{pmatrix} 1.5 \\ 1.5 \end{pmatrix}, \boldsymbol{b}_{\text{True}} = \begin{pmatrix} 0.1 \\ 0.1 \\ 0.1 \end{pmatrix}. \tag{1}$$

Obtain target values $\boldsymbol{y}$:

```
import numpy as np
X = np.array([[1, 2], [3, 1], [1, 1]])
true_w = np.array([[1.5], [1.5]])
true_b = np.array([[0.1], [0.1], [0.1]])
y = X @ true_w + true_b
print(X); print(y)
```

```
[[1 2]
 [3 1]
 [1 1]]
[[4.6]
 [6.1]
 [3.1]]
```

A batch with $N = 3$ and $d_{\boldsymbol{x}} = 2$:

$$\boldsymbol{X} = \begin{pmatrix} 1 & 2 \\ 3 & 1 \\ 1 & 1 \end{pmatrix}, \boldsymbol{y} = \begin{pmatrix} 4.6 \\ 6.1 \\ 3.1 \end{pmatrix}. \tag{2}$$

Step 1: Write down $\mathcal{L}(\mathcal{D}, (\boldsymbol{w}, \boldsymbol{b}))$ and $\hat{\boldsymbol{y}}^*$

$$\mathcal{L}(\mathcal{D}, (\boldsymbol{w}, \boldsymbol{b})) = \frac{1}{N} \sum_{i=1}^{N} \left(\hat{y}(\boldsymbol{x}_i) - y_i\right)^2 = \frac{1}{N} (\hat{\boldsymbol{y}} - \boldsymbol{y})^\top (\hat{\boldsymbol{y}} - \boldsymbol{y}), \tag{3}$$

where

$$\hat{y}(\boldsymbol{x}_i) = \boldsymbol{x}_i \boldsymbol{w} + \boldsymbol{b}, \tag{4}$$

$$\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{w} + \boldsymbol{b} = \begin{pmatrix} \hat{y}(\boldsymbol{x}_1) \\ \hat{y}(\boldsymbol{x}_2) \\ \hat{y}(\boldsymbol{x}_3) \end{pmatrix}. \tag{5}$$

Step 2: Derive $\frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}}$, $\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{w}}$, and $\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{b}}$

$$\frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}} = \frac{2}{N}\sum_{i=1}^{N}\left(\hat{y}(\boldsymbol{x}_i) - y_i\right) = \frac{2}{N}(\hat{\boldsymbol{y}} - \boldsymbol{y}), \tag{6}$$

$$\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{w}} = \boldsymbol{X}, \tag{7}$$

$$\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{b}} = \boldsymbol{I}_{3\times 1} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}. \tag{8}$$

Step 3: Derive $\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}}$ and $\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}}$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}}\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{w}} = \frac{2}{N}\sum_{i=1}^{N}\boldsymbol{x}_i^{\top}\left(\hat{y}(\boldsymbol{x}_i) - y_i\right) = \frac{2}{N}\boldsymbol{X}^{\top}(\hat{\boldsymbol{y}} - \boldsymbol{y}), \tag{9}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}}\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{b}} = \frac{2}{N}\sum_{i=1}^{N}\left(\hat{y}(\boldsymbol{x}_i) - y_i\right) = \frac{2}{N}\boldsymbol{I}^{\top}(\hat{\boldsymbol{y}} - \boldsymbol{y}). \tag{10}$$

Step 4: Initialise the weights and biases. Evaluate the gradients.

$$\boldsymbol{w} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \boldsymbol{b} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \tag{11}$$

Subsequently,

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = \frac{2}{3}\underbrace{\begin{pmatrix} 1 & 3 & 1 \\ 2 & 1 & 1 \end{pmatrix}}_{\boldsymbol{X}}\left[\underbrace{\begin{pmatrix} 3 \\ 4 \\ 2 \end{pmatrix}}_{\hat{\boldsymbol{y}}} - \underbrace{\begin{pmatrix} 4.6 \\ 6.1 \\ 3.1 \end{pmatrix}}_{\boldsymbol{y}}\right] = \begin{pmatrix} -6.000 \\ -4.267 \end{pmatrix}, \tag{12}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}} = \frac{2}{3}\underbrace{\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}}_{\boldsymbol{I}}\left[\underbrace{\begin{pmatrix} 3 \\ 4 \\ 2 \end{pmatrix}}_{\hat{\boldsymbol{y}}} - \underbrace{\begin{pmatrix} 4.6 \\ 6.1 \\ 3.1 \end{pmatrix}}_{\boldsymbol{y}}\right] = -3.200. \tag{13}$$

```
#number of rows == number of observations in the batch
N = X.shape[0]
w = np.array([[1], [1]])
b = np.array([[0] * N]).reshape(N, 1)

#Gradients
y_hat = X @ w + b
dw = 2/N * X.T @ (y_hat - y)
db = 2/N * np.sum(y_hat - y)
print(dw); print(db)
```

```
[[-6.        ]
 [-4.26666667]]
-3.1999999999999993
```

Step 5: Pick a learning rate η and update the weights and biases.

$$\eta = 0.1, \tag{14}$$

$$\boldsymbol{w} = \boldsymbol{w} - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = \begin{pmatrix} 1.600 \\ 1.427 \end{pmatrix}, \tag{15}$$

$$\boldsymbol{b} = \boldsymbol{b} - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}} = \begin{pmatrix} 0.320 \\ 0.320 \\ 0.320 \end{pmatrix} \tag{16}$$

```
#specify a learning rate to update
eta = 0.1
w = w - eta * dw
b = b - eta * db
print(w); print(b)
```

```
[[1.6       ]
 [1.42666667]]
[[0.32]
 [0.32]
 [0.32]]
```

Next Step: Update until convergence.

```
#loss function
def mse(y_pred, y_true):
  return(np.mean((y_pred-y_true)**2))

def lr_gradient_descent(eta = 0.1, w = None, b = None,
                        max_iter = 100, tol = 1e-08):
  """
  Gradient descent optimization for linear regression.

  Parameters:
  eta: float - learning rate (default=0.1)
  w: numpy array of shape (d, 1) - initial weights (default=ones)
  b: numpy array of shape (N, 1) - initial bias (default=zeros)
  max_iter: int - maximum number of iterations (default=100)
  tol: float - tolerance for stopping criteria (default=1e-08)

  Returns:
  w, b - optimized weights and bias
  """
  N, d = X.shape

  if w is None:
      w = np.ones((d, 1))
  if b is None:
      b = np.zeros((N, 1))

  prev_error = np.inf
  for _ in range(max_iter):
    y_hat = X @ w + b
    error = mse(y_hat, y)
    if np.abs(error - prev_error) < tol:
      return(w, b)
      break
    prev_error = error

    dw = 2/N * X.T @ (y_hat - y)
    db = 2/N * (y_hat - y)
    w -= eta * dw
    b -= eta * db
  return(w, b)

#Default initialisation
```

```
w_updated, b_updated = lr_gradient_descent(max_iter = 1000)
print(w_updated)
print(b_updated)
```

```
[[1.51250004]
 [1.47499994]]
[[0.13766113]
 [0.08758062]
 [0.11209706]]
```

Different Learning Rates and Initialisations

See more details in `Week_2_Tutorial_Notebook`.

Exercises

1. Apply stochastic gradient descent for the example given in Section 4.3.
2. Apply batch gradient descent for logistic regression. Follow the steps and information in Section 4.3.