COMP36212 Assignment EX3. Gradient-Based Optimisation

Vladislav Yotkov

1 Problem Statement

In this assignment we explore the optimisation methods of an Artificial Neural Network (ANN) for image classification. The task is to train a multi-layer perceptron, applied to classify images of handwritten digits (0-9) from the MNIST dataset (Lecun et al. 1998), aiming to achieve maximum classification accuracy. The process involves the computation of a prediction from a sample input (i.e., an image), the comparison of the network prediction with the true label to formulate an objective function for optimisation (cross-entropy loss, Eq. 1), and the use of target optimisation approaches to minimise the objective function with respect to the network parameters.

$$L = -\sum_{k}^{N} \hat{y}_k \log(P_k) \tag{1}$$

1.1 Dataset

The MNIST dataset (Lecun et al. 1998) is a collection of 70,000 images of handwritten digits (0-9), each of which is a 28x28 pixel image, with pixel intensity values ranging from 0-255. For input into the ANN, it is divided into 60,000 training and 10,000 testing images, while each sample is flattened into an array of 784 elements, which is further normalised to the range $0 \leq x_i \leq 1$.

1.2 Artificial Neural Network (ANN)

The ANN is a multi-layer perceptron, consisting of five layers with a total of 784 neurons in the input layer, three hidden layers containing 300, 100, and 100 neurons respectively, and an output layer of 10 neurons (Figure 1) for a total of $276,200$ trainable parameters.

The activation function used for the hidden layers is the Rectified Linear Unit (ReLU, Eq. 2), while the output layer uses the softmax function (Eq. 3), which normalises the output logits to a probability distribution over the 10 classes.

$$ReLU(x) = \max(0, x) \tag{2}$$

$$softmax(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} \tag{3}$$

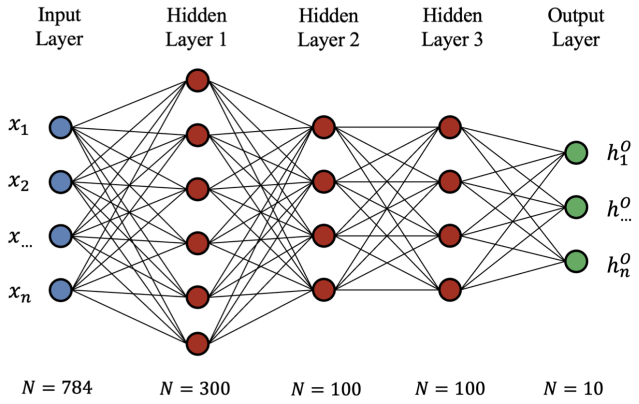


Figure 1: Artificial Neural Network (ANN) architecture, taken from the assignment specification.

2 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent is one of the most significant and widely used optimisation algorithms used in machine learning. It is a variant of the Gradient Descent (GD) algorithm, which is used to find the local minimum of a function by iteratively moving in the direction of the negative gradient. However, the main difference is that SGD uses a random sample of the training data in order to compute the gradient, unlike the GD algorithm which is applied over the entire dataset - leading to high computational costs. In our work, we will be exploring the **on-line** (i.e., stochastic) and **mini-batch SGD** (where $m \ll N$, such that m is the size of the mini-batch, and N is the size of the training dataset) variants of the algorithm. We can define more formally this optimisation method in Algorithm 1, while the update rule is given by Eq. 4.

$$w = w - \frac{\eta}{m} \sum_{i=1}^{m} \nabla L_i(x_i) \tag{4}$$

We must further note a few key benefits of using an SGD over general GD:

1. On-line (i.e., stochastic) learning requires a smaller step size (i.e., learning rate) to counteract inherent noise, resulting in smoother but slower adaptation (Wilson and Martinez 2003). Despite the increased

Algorithm 1 Mini-batch Stochastic Gradient Descent (SGD)

Require: Training dataset D
Require: Learning rate η
Require: Mini-batch size m
Ensure: Optimized model parameters w
1: Initialize model parameters w
2: **while** stopping criteria not met **do**
3: Sample an m-sized mini-batch from D: $\{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$
4: Compute gradient: $\nabla L = \frac{1}{m} \sum_{i=1}^{m} \nabla L_i(x_i, y_i)$
5: Update model weights: $w = w - \eta \nabla L$
6: **end while**
7: **return** Optimized model parameters w

runtime, small batches offer a beneficial regularization effect and over time, noise averages out, driving the weights towards the true gradient (Goodfellow, Bengio, and Courville 2016).

2. Large batch sizes exhibit a degradation in quality (Mishkin, Sergievskiy, and Matas 2017) and a low generalisation capability and are prone to getting stuck in local minima due to their convergence to sharp minimizers of the training function (Keskar et al. 2017).

2.1 Learning Rate Experiments

We experimented with different hyperparameters for the SGD algorithm, in order to find the optimal configuration for our ANN classifier:

1.**Learning rate**: The learning rate η is the step size of the gradient descent algorithm, and is the most important hyper-parameter (Goodfellow, Bengio, and Courville 2016). Setting it to larger values risks rapid changes and overshooting the minimum, while converging to a suboptimal solution. On the other hand, a smaller learning rate requires more training epochs to converge, and is more likely to get stuck in a local minimum. In our work on SGD, we experimented with the following values $\eta = \{0.1, 0.01, 0.001\}$.

2.**Batch size**: The batch size m is the number of training samples used to compute the gradient at each iteration. The mini-batch SGD is said to follow the gradient of the true generalization error (Goodfellow, Bengio, and Courville 2016) by computing an unbiased estimate (implying data is sampled randomly). When $m = 1$, the algorithm is called **on-line** (i.e., stochastic) learning, whereas for $m = N$ it is called **batch** learning. In our work, we experimented with the following batch size values $m = \{1, 10, 100\}$.

From the experiments on both learning rates and batch sizes we reach the following conclusions:

1. **Instability for** $lr = 0.1, m = 1$: From Fig. 3a it becomes clear that a combination of a large learning rate and a small batch size leads to a very unstable optimization process, with a nan loss value and a constant test accuracy of 0.098 (i.e., random guessing) as visible in Table **??**. These results are expected due to the erratic stochastic updates, and the high variance in the gradient estimates (Goodfellow, Bengio, and Courville 2016). In contrast, just by decreasing the learning rate to $\eta = 0.01$ or by increasing the batch size to $m = 10$ we can achieve a stable and an accurate solution ($\geq 70\%$).
2. **Slow convergence for large** m: While a larger batch size leads to a more stable learning because of the averaging effect, it can suffer from a lack of generalization caused by the convergence to sharp minimizers and the inability to escape them post factum (Keskar et al. 2017). This is evident for $(lr = 0.001, m = 10)$ and $(lr = 0.001, m = 100)$ from Fig. 3b and 3c, respectively, where the test accuracy convergence slows down over all 10 epochs.

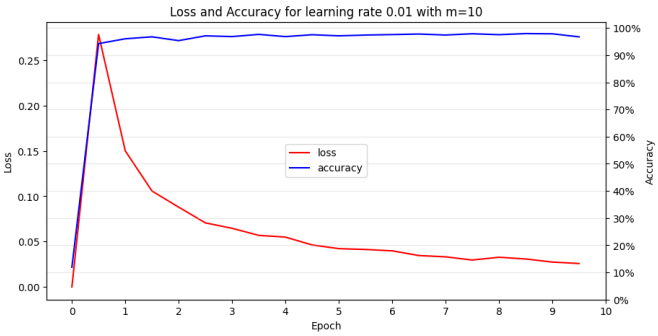


Figure 2: Loss and test accuracy for lr=0.1 and $m = 10$.

Mean Loss	Test Acc	lr	Batch Size	Time (s)
nan	0.098	0.1	1	3533s
0.025	0.977	0.1	10	3268s
0.017	0.976	0.1	100	3268s
0.027	0.977	0.01	1	3557s
0.019	0.976	0.01	10	3338s
0.17	0.950	0.01	100	3301s
0.019	**0.978**	0.001	1	3553s
0.16	0.952	0.001	10	3332s
0.53	0.872	0.001	100	3288s

Table 1: SGD performance: Loss and Accuracy

2.2 Analytical Validation

To validate the correctness of the provided analytical gradient calculation, we approximate the derivative using three different finite-difference methods (Ford 2015):

1. Forward difference: $\frac{f(x+h)-f(x)}{h}$
2. Backward difference: $\frac{f(x)-f(x-h)}{h}$
3. Central difference: $\frac{f(x+h)-f(x-h)}{2h}$

where we set the step size h to 10^{-8}.

Due to the significant computational cost of the all these methods (each takes ~1s per sample), we only compute the numerical gradients for a single sample at the end of the first training epoch and compare them to the analytical ones. We carry out our experiments using a learning rate of $\eta = 0.1$ and a mini-batch size of $m = 10$. The results shown in Figure 4 and Figure 5 indicate that the analytical and numerical gradients differ by less than $3 \times 10^{-4}\%$ (central difference) on average, which goes to show that the analytical gradient calculation is correct. As expected the central difference method provides a better approximation and a lower truncation error than the forward and backward difference ones due to the fact that it is second-order accurate, while the others are first-order accurate. This is clearly visible from the lower mean relative error and the narrower distribution of the central difference method.

3 Improving Convergence

4 Adaptive Learning

Conclusion

Ford, William. 2015. "Chapter 12 - Linear System Applications." In *Numerical Linear Algebra with Applications*, edited by William Ford, 241–62. Boston: Academic Press. https://doi.org/https://doi.org/10.1016/B978-0-12-394435-1.00012-0.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

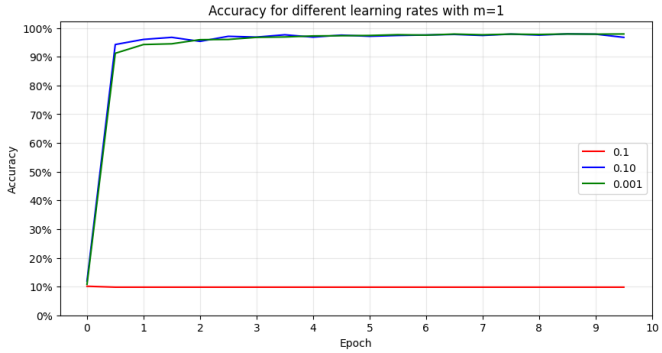
Keskar, Nitish Shirish, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2017. "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima." https://arxiv.org/abs/1609.04836.

Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner. 1998. "Gradient-Based Learning Applied to Document Recognition." *Proceedings of the IEEE* 86 (11): 2278–2324. https://doi.org/10.1109/5.726791.

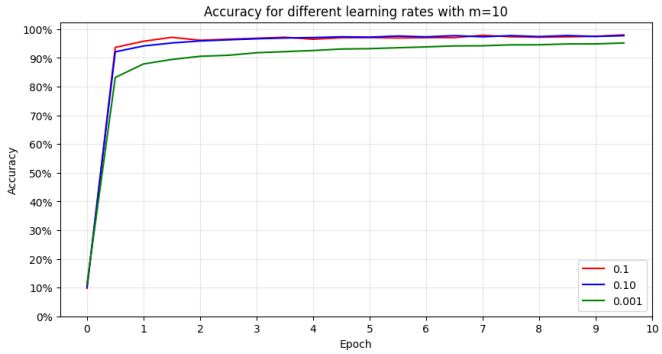
Mishkin, Dmytro, Nikolay Sergievskiy, and Jiri Matas. 2017. "Systematic Evaluation of Convolution Neural Network Advances on the Imagenet." *Computer Vision and Image Understanding* 161 (August): 11–19. https://doi.org/10.1016/j.cviu.2017.05.007.

Wilson, D.Randall, and Tony R. Martinez. 2003. "The General Inefficiency of Batch Training for Gradient Descent Learning." *Neural Networks* 16 (10): 1429–51. https://doi.org/https://doi.org/10.1016/S0893-6080(03)00138-2.

(a) Batch size (m) = 1



(b) Batch size (m) = 10



(c) Batch size (m) = 100

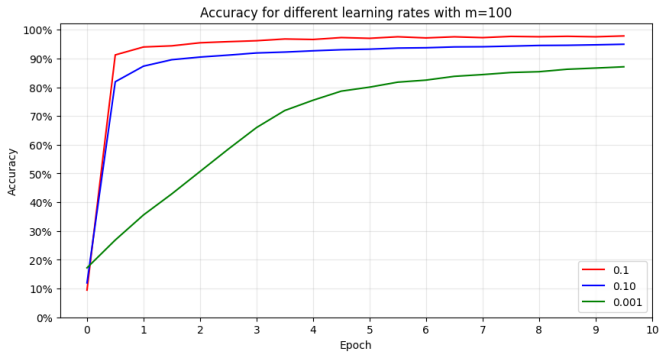


Figure 3: Test accuracy for different learning rates and batch sizes.

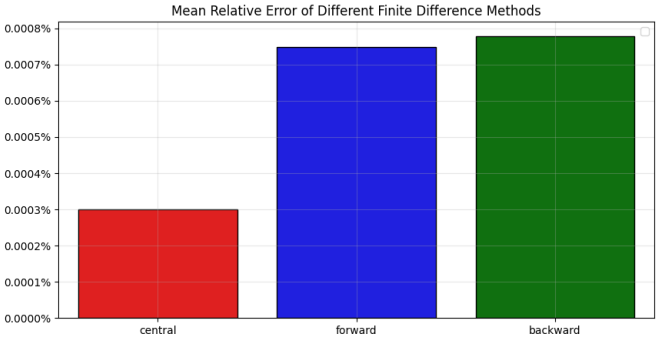


Figure 4: Mean relative error of the finite-difference methods.

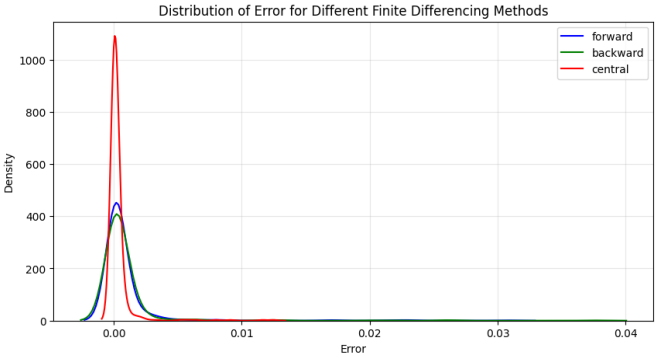


Figure 5: Relative error distribution of the finite-difference methods.