AIML 425 T2/2022 - ASSIGNMENT 1

Corvin Idler - ID 300598312

1. INTRODUCTION

This report describes experiments to train a neural network on the task of projecting points from a trivariate Gaussian Normal distribution onto the surface of a unit sphere. The experiments were executed in Google Colab with the following specs:

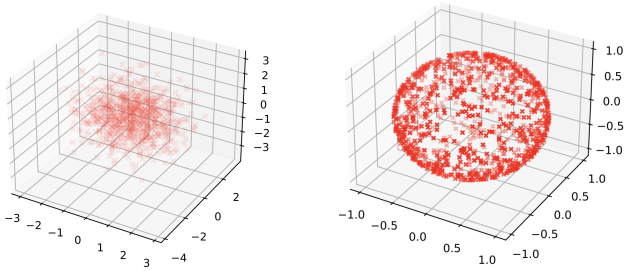
- GPU: NVIDIA Tesla T4 16GB
- NVIDIA-SMI driver version: 460.32.03
- CUDA Version: 11.2
- CPU: 2x Intel(R) Xeon(R) CPU @ 2.20GHz
- (Py)Torch [1] version 1.11.0+cu113

The jupyter-notebook underpinning this report can be found on GitHub: `https://github.com/econdatatech/AIML425/blob/main/AIML425_Assignment_1.ipynb`.

2. THEORY / CONCEPT

2.1. Data generation

The trivariate normal distribution of a 3-dimensional random vector $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with the following pdf $f_{\mathbf{X}}(x_1, x_2, x_3) = \frac{\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)}{\sqrt{(2\pi)^3|\boldsymbol{\Sigma}|}}$ is fully specified by a 3-dimensional mean vector $\boldsymbol{\mu}$ as well as a 3×3 covariance matrix $\boldsymbol{\Sigma}$. The assignment's instructions specified $\boldsymbol{\mu} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$ and $\boldsymbol{\Sigma} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$. 10000 samples were drawn from the above distribution and split into training, validation and test set with a ratio of 70%:15%:15%. The first 1000 samples of the training data set are visualized in the 3D plot in Figure 1 (a). To project these points onto a unit sphere (assumed to sit at the origin of the coordinate system) the length of the random sample point vectors has to be scaled to 1 (the radius of the unit sphere), so the vectors and the sphere intersect. This is achieved by dividing each vector $\mathbf{x}$ by it's own length to create a unit vector $\hat{\mathbf{x}} = \frac{\mathbf{x}}{\|\mathbf{x}\|} = \frac{\mathbf{x}}{\sqrt{x_1^2+x_2^2+x_3^2}}$. The result of this projection is shown in Figure 1 (b). The code for the full data generation can be found in Listing 1 in the Annex.



(a) 1000 samples from trivariate Normal distribution

(b) Projection of 1000 samples to the unit sphere

Fig. 1: Visualisation of the training data

2.2. Neural Network architecture

As per the assignment instructions a 4-layered fully connected network with the Rectified Linear Unit (ReLU) activation function in the hidden layers was specified. The network has 243 trainable parameters after accounting for biases (omitted in Figure 2 for readability). The code for the network generation can be found in Listing 2 in the Annex.

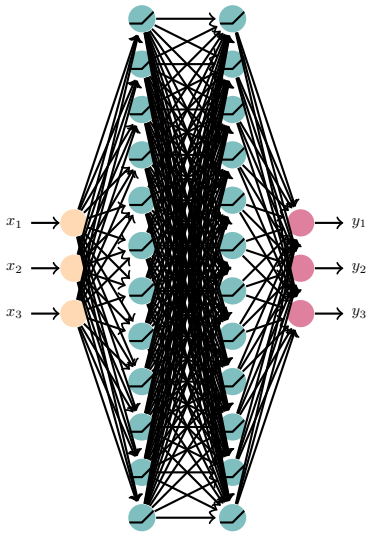


Fig. 2: 4-layer fully connected network with Relu activation function

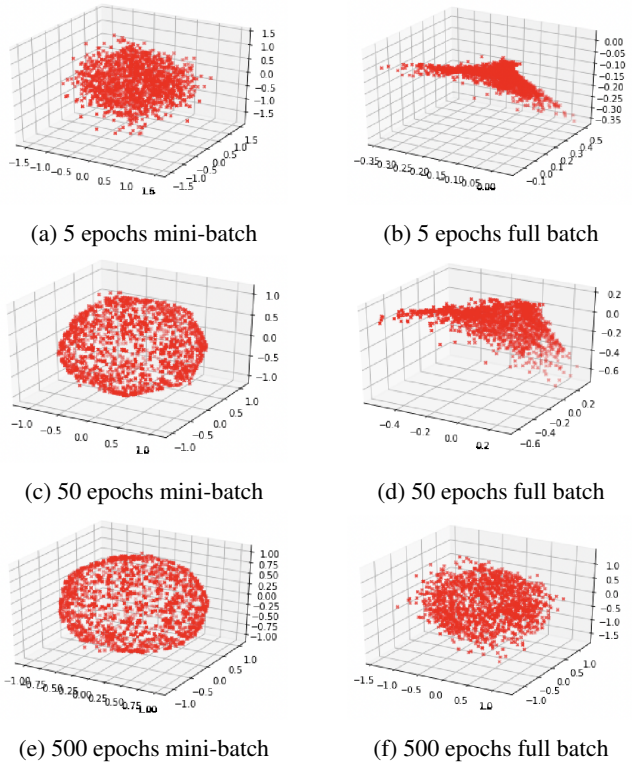
2.3. Optimizer and Loss function

As an optimizer I chose Adam [2] with Pytorch default parameters and for the loss function the mean squared error between predicted points and the ground truth. As the loss function is calculated for each mini-batch, the average is also calculated only over each mini-batch and not the whole training or validation or test set. The best one can calculate for measuring the loss in an epoch is a weighted average across mini-batches.

3. RESULTS / CONCLUSION

3.1. Training and validation performance

I experimented with several mini-batch sizes as well as number of epochs, but for space reasons will report only on the results of two experiments, namely a mini-batch size of 100 vs. a full batch approach. In both cases I used 2000 epochs, as previous experiments with 5000 epochs showed that the loss function flattens out after this point and there is little merit in running further epochs. In line with current literature, the smaller mini-batch size results into a faster conversion behaviour as well as a better overall accuracy as can be seen in Figures 3 and 4. The loss curves in Figure 4 show the training as well as the validation error. Both are virtually identical and a difference can be best spotted for the full batch case with the light blue line being the validation error. We can see from the graph that the mini-batch approach converges extremely fast to a stable loss, while the full batch approach takes over 2000 epochs to reach a similarly small loss. These findings are as well be visually confirmed by Figure 3. We can see in sub-figure (c) that the mini-batch approach reaches very good visual results already after 50 epochs. The full batch approach in return seems to end up in a visually "pathological" state after the first couple of epochs with a projection of the point cloud onto an almost planar surface (sub-figure (b)). It then takes a large number of epochs before the point cloud resembles a spherical shape again (around epoch 500, sub-figure (f)).



(a) 5 epochs mini-batch

(b) 5 epochs full batch

(c) 50 epochs mini-batch

(d) 50 epochs full batch

(e) 500 epochs mini-batch

(f) 500 epochs full batch

Fig. 3: Visualisation of training progress measured on the validation set for mini-batch vs. full batch

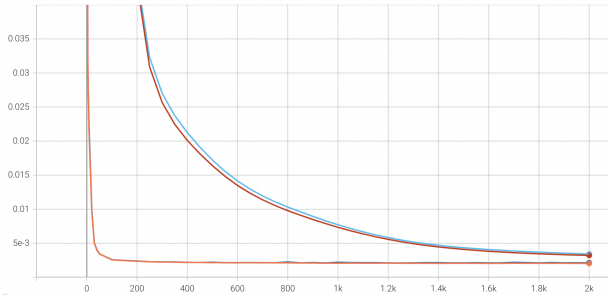


Fig. 4: Error of mini-batch (lower curve) and full batch (upper curve) experiments as a function of epochs.

3.2. Test error performance

To assess the test performance more intuitively than with a scalar MSE, I decided to focus on the length of the point vectors projected by the neural network. Figure 5 plots the histogram of the vector lengths before and after the projection. In a perfect world one would see one single blue spike at 1.0. The rather narrow distribution around 1.0 indicates that the model performs quite well on the test set. That said, this approach of performance evaluation only assesses if the point vectors intersect with the unit sphere, but theoretically allows that to happen at the wrong point on the sphere compared to where the original vector should mathematically intersect. This aspect would indeed be better covered by the MSE.

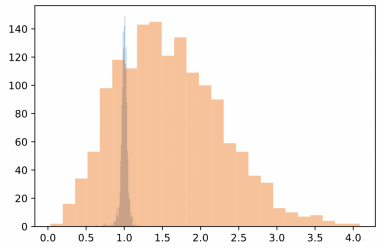


Fig. 5: Histogram of vector lengths (pre- (orange) vs. post-training (blue)

Annex

4. REFERENCES

[1] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., pp. 8024–8035. Curran Associates, Inc., 2019.

[2] Diederik P. Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun, Eds., 2015.


```python
class Gauss2UnitCircleDataset(data.Dataset):
    def __init__(self, size):
        """
        Inputs:
        size = No. of data points to generate
        """
        super().__init__()
        self.size = size
        self.generate_3D_Gauss2UnitCircle()

    def generate_3D_Gauss2UnitCircle(self):
        means=torch.zeros(3)
        cov=torch.eye(3)
        dis = MultivariateNormal(means, cov)
        X=dis.sample((self.size,))
        y = X.div(torch.linalg.norm(X,dim=1)
                  .reshape((self.size,1)))

        self.X = X
        self.y = y

    def __len__(self):
        return self.size

    def __getitem__(self, idx):
        X = self.X[idx]
        y = self.y[idx]
        return X, y

dataset = Gauss2UnitCircleDataset(size=1000)
```

Listing 1: Code for data generation

```python
class SimpleRegression(nn.Module):

    def __init__(self):
        super().__init__()
        self.hidden1 = nn.Linear(3, 12)
        self.hidden2 = nn.Linear(12, 12)
        self.output = nn.Linear(12, 3)

    def forward(self, x):
        x = F.relu(self.hidden1(x))
        x = F.relu(self.hidden2(x))
        x = self.output(x)

        return x

model = SimpleRegression()
```

Listing 2: Code for network generation