

Mini Deep-Learning Framework

Menghe JIN
menghe.jin@epfl.ch

Yueqing SHEN
yueqing.shen@epfl.ch

Jiaan ZHU
jiaan.zhu@epfl.ch

Abstract—Though there have been various tools for deep-learning structures, the exploration of this field always starts with manually implementing some simple neural networks. In the project, a simple neural network is constructed with PyTorch tensor operations with components like fully-connected layers, several activation functions, batch normalization and their corresponding back propagation.

I. INTRODUCTION

The goal of this project is to manually implement a simple neural network with some key components, such as fully-connected layer, activation function, batch normalization, etc. In this report, the implementation of these key components and the math behind is shown in Section II and the structure of the Python class is described in Section III. Section IV shows some empirical results and Section V concludes the report.

II. COMPONENTS

In this report, we denote the row vectors as bold lower-case letters (e.g. \mathbf{b}), matrices as upper-case letters (e.g. W), multi-dimensional arrays as bold upper-case letters (e.g. \mathbf{W}), activation functions as $\delta(\cdot)$ and constants as lower-case Greek letters (e.g. γ). For each of the parameters and results, a superscript will be added to show on which layer it is in, like $W^{[l]}$, while another superscript is added to infer the sample index, like $\mathbf{x}^{[l],(k)}$. For all subscripts, they are indicating the position of element in the array/matrix/vector (e.g. $W_{ij}^{[l]}$, $\mathbf{x}_i^{[l],(k)}$).

A. Fully-connected layer

For fully-connected layers, we use the same dimensions as our code where the input matrix $X^{[l-1]} \in \mathbb{R}^{N \times D^{[l-1]}}$ with each row corresponding to each sample, parameters $W^{[l]} \in \mathbb{R}^{D^{[l-1]} \times D^{[l]}}$ and $\mathbf{b}^{[l]} \in \mathbb{R}^{1 \times D^{[l]}}$. Then we have

$$\mathbf{Y}^{[l]} = X^{[l-1]}W^{[l]} + \mathbf{b}^{[l]}, \quad \mathbf{Y}^{[l]} \in \mathbb{R}^{N \times D^{[l]}}$$

For each sample represented in a row vector (\mathbf{x} or \mathbf{y}), it is

$$\mathbf{y}^{[l],(k)} = \mathbf{x}^{[l-1],(k)} \cdot W^{[l]} + \mathbf{b}^{[l]}$$

For simpler representation in the derivation of gradient, temporarily rewrite the above formula as

$$\mathbf{y} = \mathbf{x}W + \mathbf{b}$$

and we have

$$\mathbf{y}_j = \sum_{i=1}^{D^{[l-1]}} \mathbf{x}_i W_{ij} + \mathbf{b}_j, \quad j = 1, \dots, D^{[l]}$$

To get the gradient in the back propagation, we are looking for $\nabla_{\mathbf{x}}\mathcal{L}$, $\nabla_W\mathcal{L}$ and $\nabla_{\mathbf{b}}\mathcal{L}$ and by chain rule,

$$\nabla_{\mathbf{x}}\mathcal{L} = \left[\sum_{j=1}^{D^l} \frac{\partial \mathcal{L}}{\partial \mathbf{y}_j} \frac{\partial \mathbf{y}_j}{\partial \mathbf{x}_i} \right] = \left[\sum_{j=1}^{D^l} \frac{\partial \mathcal{L}}{\partial \mathbf{y}_j} W_{ij} \right] = \nabla_{\mathbf{y}}\mathcal{L} \cdot W^T$$

$$\nabla_W\mathcal{L} = \left[\frac{\partial \mathcal{L}}{\partial \mathbf{y}_j} \frac{\partial \mathbf{y}_j}{\partial W_{ij}} \right] = \left[\frac{\partial \mathcal{L}}{\partial \mathbf{y}_j} \mathbf{x}_i \right] = \mathbf{x}^T \cdot \nabla_{\mathbf{y}}\mathcal{L}$$

$$\nabla_{\mathbf{b}}\mathcal{L} = \left[\frac{\partial \mathcal{L}}{\partial \mathbf{y}_j} \frac{\partial \mathbf{y}_j}{\partial \mathbf{b}_j} \right] = \left[\frac{\partial \mathcal{L}}{\partial \mathbf{y}_j} \right] = \nabla_{\mathbf{y}}\mathcal{L}$$

B. Batch normalization

To perform batch normalization, again temporarily ignore the layer superscripts, in the forward pass of training, we have

$$\mu_j = \frac{1}{N} \sum_{k=1}^N Y_{kj}, \quad j = 1, \dots, D^{[l]}$$

$$\sigma_j^2 = \frac{1}{N} \sum_{k=1}^N (Y_{kj} - \mu_j)^2, \quad j = 1, \dots, D^{[l]}$$

$$\bar{Y}_{kj} = \frac{Y_{kj} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}, \quad k = 1, \dots, N, \quad j = 1, \dots, D^{[l]}$$

$$\hat{Y}_{kj} = \gamma_j \cdot \bar{Y}_{kj} + \beta_j, \quad j = 1, \dots, D^{[l]}$$

where N is the number of samples; μ_j and σ_j^2 are the mean and variance of characteristics at index j ; γ and β are to be learned. In the forward pass of testing, however, the normalized data is not calculated from the sample mean and variance, but the running mean and variance which has been updated at each iteration of training, with

$$\hat{\mu}_j = \rho \hat{\mu}_j + (1 - \rho) \mu_j$$

$$\hat{\sigma}_j = \rho \hat{\sigma}_j + (1 - \rho) \sigma_j$$

In back propagation of batch normalization,

$$\begin{aligned} \nabla_{\bar{\mathbf{Y}}}\mathcal{L} &= \left[\frac{\partial \mathcal{L}}{\partial \bar{Y}_{kj}} \right] = \left[\frac{\partial \mathcal{L}}{\partial \hat{Y}_{kj}} \frac{\partial \hat{Y}_{kj}}{\partial \bar{Y}_{kj}} \right] = \left[\frac{\partial \mathcal{L}}{\partial \hat{Y}_{kj}} \gamma_j \right] \\ &= (\gamma)^T \otimes \nabla_{\hat{\mathbf{Y}}}\mathcal{L} \quad (\text{with broadcasting}) \end{aligned}$$

$$\begin{aligned} \nabla_{\gamma}\mathcal{L} &= \left[\sum_{k=1}^N \frac{\partial \mathcal{L}}{\partial \hat{Y}_{kj}} \frac{\partial \hat{Y}_{kj}}{\partial \gamma_j} \right] = \left[\sum_{k=1}^N \frac{\partial \mathcal{L}}{\partial \hat{Y}_{kj}} \bar{Y}_{kj} \right] \\ &= \sum_{k=1}^N (\nabla_{\hat{\mathbf{Y}}}\mathcal{L} \otimes \bar{\mathbf{Y}})_k \end{aligned}$$

$$\begin{aligned}\nabla_{\beta}\mathcal{L} &= \left[\sum_{k=1}^N \frac{\partial \mathcal{L}}{\partial \hat{Y}_{kj}} \frac{\partial \hat{Y}_{kj}}{\partial \beta_j} \right] = \left[\sum_{k=1}^N \frac{\partial \mathcal{L}}{\partial \hat{Y}_{kj}} \right] \\ &= \sum_{k=1}^N (\nabla_{\hat{Y}} \mathcal{L})_k\end{aligned}$$

where \otimes stands for element-wise multiplication. And for the gradient with respect to Y ,

$$\begin{aligned}\nabla_{\sigma^2}\mathcal{L} &= [\partial \mathcal{L} / \partial \sigma_j^2] \\ &= \left[-\frac{1}{2} \sum_{k=1}^N \frac{\partial \mathcal{L}}{\partial \hat{Y}_{kj}} (Y_{kj} - \mu_j) (\sigma_j^2 + \epsilon)^{-3/2} \right] \\ &= -\frac{1}{2} \sum_{k=1}^N (\nabla_{\hat{Y}} \mathcal{L} \otimes (Y - \mu))_k \otimes (\sigma^2 + \epsilon)^{-3/2}\end{aligned}$$

$$\begin{aligned}\nabla_{\mu}\mathcal{L} &= [\partial \mathcal{L} / \partial \mu_j] = \left[-\sum_{k=1}^N \frac{\partial \mathcal{L}}{\partial \hat{Y}_{kj}} (\sigma_j^2 + \epsilon)^{-1/2} \right] \\ &\quad + \left[\frac{\partial \mathcal{L}}{\partial \sigma_j^2} \cdot \frac{1}{N} \cdot \sum_{k=1}^N (-2) \cdot (Y_{kj} - \mu_j) \right] \\ &= -\sum_{k=1}^N (\nabla_{\hat{Y}} \mathcal{L})_k \otimes (\sigma^2 + \epsilon)^{-1/2} \\ &\quad - 2 \cdot \nabla_{\sigma^2} \mathcal{L} \otimes \left(\frac{1}{N} (Y - \mu) \right)\end{aligned}$$

$$\begin{aligned}\nabla_Y \mathcal{L} &= [\partial \mathcal{L} / \partial Y_{kj}] = \left[\frac{\partial \mathcal{L}}{\partial \hat{Y}_{kj}} \cdot (\sigma_j^2 + \epsilon)^{-1/2} \right] \\ &\quad + \left[\frac{\partial \mathcal{L}}{\partial \sigma_j^2} \cdot \frac{2(Y_{kj} - \mu_j)}{N} + \frac{\partial \mathcal{L}}{\partial \mu_j} \cdot \frac{1}{N} \right] \\ &= \nabla_{\hat{Y}} \mathcal{L} \otimes (\sigma^2 + \epsilon)^{-1/2} \\ &\quad + \frac{2}{N} \nabla_{\sigma^2} \mathcal{L} \otimes (Y - \mu) + \frac{1}{N} \nabla_{\mu} \mathcal{L}\end{aligned}$$

C. Activation functions

We have implemented ReLU, sigmoid and tanh as activation functions ($Z = \delta(\hat{Y})$) and their gradients are implemented as: For ReLU,

$$\nabla_{\hat{Y}_{kj}} \mathcal{L} = \begin{cases} \nabla_{Z_{kj}} \mathcal{L}, & \text{for } Y_{kj} \geq 0 \\ 0, & \text{for } Y_{kj} < 0 \end{cases}$$

For tanh,

$$\nabla_{\hat{Y}_{kj}} \mathcal{L} = \nabla_{Z_{kj}} \mathcal{L} \cdot (1 - Z_{kj}^2)$$

and for sigmoid,

$$\nabla_{\hat{Y}_{kj}} \mathcal{L} = \nabla_{Z_{kj}} \mathcal{L} \cdot Z_{kj} \cdot (1 - Z_{kj})$$

III. STRUCTURE OF MODEL

We have implemented two Python classes, one being the inheritance of the other. The parent class, `Module`, can be configured as a single `Module` or a sequence of `Modules`. A single `Module` includes a fully-connected/2D-convolutional

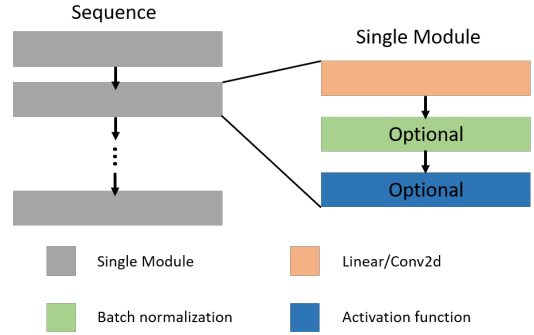


Fig. 1: Illustration of Module class

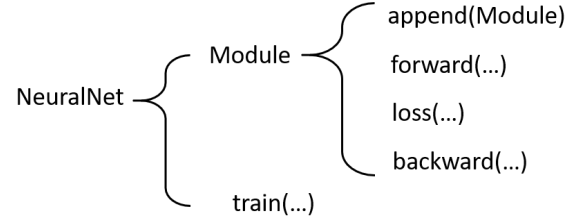


Fig. 2: Structure of NeuralNet class

layer, an optional batch normalization layer and an optional activation function, which should all be specified in the constructor of a `Module` instance together with the shape of the weights. The 'Sequential' type `Module` contains a list of single `Modules` in the order of appending them by a public method named `append(Module)`, as illustrated in Fig.1. `Module` class has several other public methods, `forward`, `loss`, `backward` and `update_params`. These methods takes a mini-batch of the whole dataset, computing the loss and its gradient, performing back propagation and gradient descent on the parameters.

`Module` class has several other public methods, `forward`, `loss`, `backward` and `update_params`. These methods takes a mini-batch of the whole dataset, computing the mean squared error as the loss and its gradient, performing back propagation and gradient descent on the parameters. The child class, `NeuralNet`, takes extra parameters for the training process, such as the learning rate, the regularization term, the number of epoches and the mini-batch size. The stochastic gradient descent (SGD) is implemented in the method `train` and is performed by randomly picking a mini-batch of samples for forward and backward pass instead of a single sample. The structure of the two classes is illustrated in Fig.2.

IV. EXPERIMENTS AND RESULTS

The training and testing dataset is generated with 1000 samples uniformly distributed in $[0, 1]^2$, each with a label 0 if outside the disk centered at $(0.5, 0.5)$ of radius $1/\sqrt{2\pi}$ and 1 inside. A sanity check is made by scattering the data points and the disk, as shown in Fig.3.

The neural network in the trial is a sequence of 4 linear `Modules` with ReLU activation, first 3 of which adopts batch

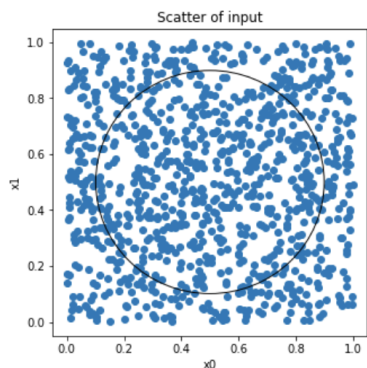


Fig. 3: Validation of the generated dataset

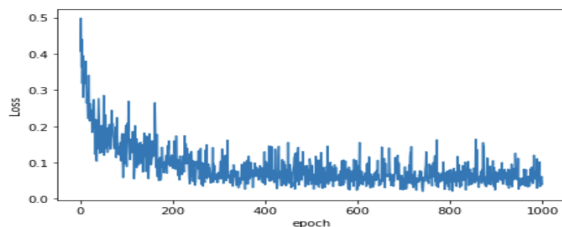


Fig. 4: Losses over training

normalization. The shape of weights in each layer is 2×100 , 100×50 , 50×20 and 20×1 respectively. Training has been performed over 1000 epoches, with learning rate of $2e^{-3}$, regularization of 0.05 and mini-batch size of 64. Fig.4 is an example of the loss change over the training and the accuracy on the testing data for this set of losses is around 0.977. Most training proceeds as above but there are occasionally some less satisfying training results, as shown in Fig.5 and the corresponding accuracy is 0.74.

V. CONCLUSION

In this project we have implemented some basic components of a neural network. The model is capable of training on the data with expected performances. This project may become a starting point of construction other components and structures, such as max-pooling and dropout, and some optimization of algorithm.

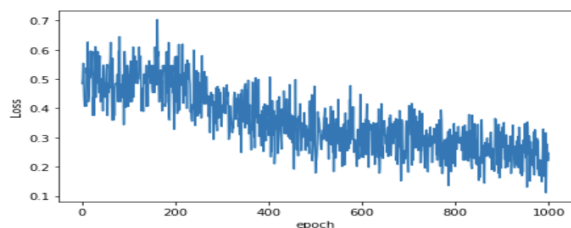


Fig. 5: Less satisfying losses over training