PyTorch practice

Multilayer Perceptron (MLP)

Matteo Boschini, Lorenzo Bonicelli

November 25, 2022

University of Modena and Reggio Emilia

Agenda

Multilayer perceptron (MLP)

For the purpose of this lecture, we'll stick to the task of image classification.

Let's assume we have a training dataset of N images

$$x_i \in \mathbb{R}^D, i = 1, \ldots, N$$

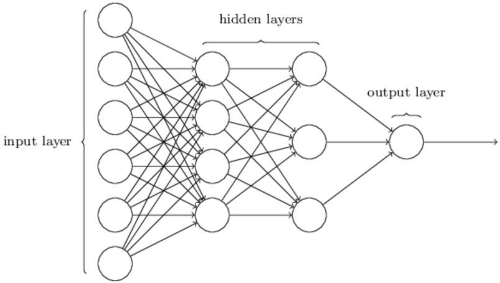
that we want to classify into K distinct classes.

Thus, training set is made by couples:

$$(x_i, y_i), \;\; where\; y_i \in \{1, \ldots, K\}$$

Our goal is to define a function $f : \mathbb{R}^D \mapsto \mathbb{R}^K$ that maps images to class scores.

When we connect an ensemble of neurons in a graph is when the magic happens and we get an actual **Multilayer perceptron (MLP)**.



Neural networks are arranged in **layers**, with one *input layer*, one *output layer* and *N hidden layers* in the middle.

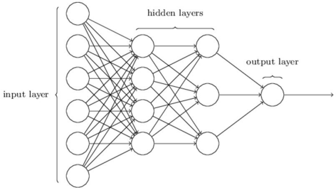
The network depicted here has a total of 47 *learnable parameters. Does this make sense to you?*

The 4-layer network previously depicted can be simply expressed as:

$$out = \phi(\mathbf{W_3}\phi(\mathbf{W_2}\phi(\mathbf{W_1}\mathbf{x}))) \tag{1}$$

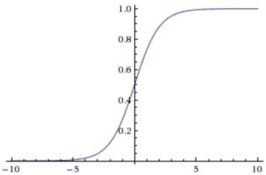
where:

- ϕ is the activation function
- $\mathbf{x} \in \mathbb{R}^6$ is the input
- $\mathbf{W_1} \in \mathbb{R}^{4x6}$ are the weights of first layer
- $\mathbf{W_2} \in \mathbb{R}^{3x4}$ are the weights of second layer
- $\mathbf{W_3} \in \mathbb{R}^{1x3}$ are the weights of third layer



Notice that to ease the notation biases have been incorporated into weight matrices W.

Activation functions are non-linear functions computed on the output of each neuron. There are a number of different activation functions you could use. In practice, the three most widely used functions have been *sigmoid*, *tanh* and *ReLu*. Nonetheless, more complex activation functions exist (*e.g.* [2, 1]).



Sigmoid nonlinearity has form $\sigma(x) = 1/(1 + e^{-x})$. Sigmoid function squashes any real-valued input into range $[0, 1]$. It has seen frequent use historically, yet now it's rarely used. It has two major drawbacks:

- saturates and kill the gradient
- output is not zero centered

Why using non-linear activations at all?

Composition of linear functions is a linear function. Without nonlinearities, neural networks would reduce to 1 layer logistic regression.

Let's say we have the following function (ϕ represent nonlinear activations):

$$f(\boldsymbol{x}) = \phi(\mathbf{W_2}\phi(\mathbf{W_1x}))$$

If we get rid of nonlinearities, this reduces to:

$$f(\boldsymbol{x}) = \mathbf{W_2W_1x} = \boldsymbol{Wx} \quad where \quad \boldsymbol{W} = \boldsymbol{W_2W_1}$$

which is clearly still linear.

Softmax Classifier generalizes Logistic Regression classifier to multi-class classification.

We first introduce the **softmax function**:

$$softmax_j(\mathbf{z}) = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

It takes a vector of arbitrary real-valued scores $\mathbf{z}$ and squashes it to a vector of values between zero and one that sum to one.

e.g.

$$\mathbf{z} = \begin{bmatrix} 1.2 \\ 5.1 \\ 2.7 \end{bmatrix} \quad softmax(\mathbf{z}) = \begin{bmatrix} 0.018 \\ 0.90 \\ 0.08 \end{bmatrix}$$

Softmax Classifier

→ In pytorch è incluso nelle loss. quindi non va fatto.

In the softmax classifier the scores of linear function mapping $f(x_i, W) = Wx_i$ are interpreted as unnormalized log probabilities. To train the classifier we optimize the **cross-entropy loss**:

$$L = -\sum_i y_i^{true} log(y_i^{pred})$$

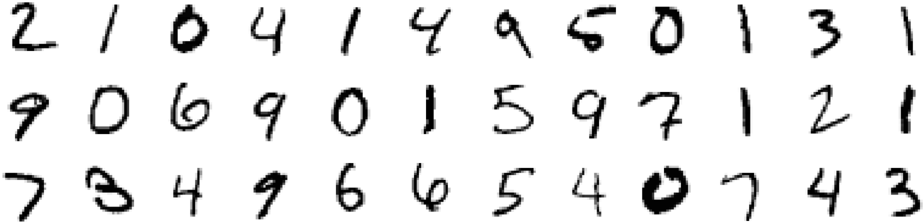
where y_i^{true} is the ground truth output distribution and y_i^{pred} is the predicted output distribution. In practice, y_i^{true} is a *one-hot* vector selecting the output of the right label.

Notice: softmax classifier has the appealing property to produce an easy-to-interpret output, that is the normalized score confidence for each class.

Goal

The **goal** of this practice is to **implement a fully-connected neural network to perform 10-class classification** on the MNIST dataset.

MNIST[3] is a database of handwritten digits consisting of 60K training images and 10K testing images. All digits have been centered in 28x28 grayscale images.



References

[1] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio.
Maxout networks.
arXiv preprint arXiv:1302.4389, 2013.

[2] K. He, X. Zhang, S. Ren, and J. Sun.
Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.
In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[3] Y. LeCun.
The mnist database of handwritten digits.
http://yann. lecun. com/exdb/mnist/, 1998.