Deep Learning, Mini-projects II

Noise2Noise model: image denoising network trained without clean references

Writing our own modules

Authors:
Steven Brown, Guillaume Briand and
Paulin de Schoulepnikoff

Professor:
Prof. François Fleuret

May 26, 2022

1 Introduction

The goal of this second miniproject is to build our framework for denoising images without using autograd and torch.nn modules. In order to achieve this, a framework with convolution and upsampling modules, ReLU and sigmoid activations has been implemented with forward and backward passes. Our framework also propose the Mean squared error loss as well as Stochastic gradient descent in order to minimise the loss.

2 Design and implementation

Here the aim is to design modules and an optimizer in a similar to the Pytorch implementation. With some modules inheriting from other classes.

2.1 Module class

Module class is the parent class of the following classes : Convolution layer, Upsampling, the activation functions (ReLU and sigmoid), MSE loss function and the Sequential. It helps structure the modules. By default the module class has a forward pass, backward pass and param method. The param method returns a list containing pairs of the module's parameters and gradients. This will be useful for accessing and updating the module's parameters.

For our project we will consider 4 dimensional tensor inputs. This is the standard for the tensor representation of sequences of images. The forward and backward pass will therefore be implemented according to this type of input but also accept single images (3 dimensonal tensor input).

2.2 Convolution layer

In order to compute the forward and backward passes of the convolution layer, 2 special operation from Pytorch are used : `fold` and `unfold`. With these operators, forward and backward passes can be evaluated as linear operations, avoiding loops. For the forward pass The gradient of the weights can be seen as a convolution operation between the input and the loss gradient from previous layer.
The weights and the bias are initialised with a Xavier normal distribution, close to the one Pytorch use for their initialization.

2.3 Upsampling layer

The Upsampling layer is usually implemented with the transposed convolution, but in this mini-project it has been chosen to use a combination of Nearest neighbour upsampling (or NNUpsampling) and the convolution layer from section 2.2.
Nearest neighbour upsampling takes as an input a scale factor (by default 2) and return a tensor with an image scaled up by this factor. The nearest neighbour mode is used, as shown on figure 1.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 3 & 3 & 4 & 4 \\ 3 & 3 & 4 & 4 \end{bmatrix} \tag{1}$$

For the backward pass, we take as input the gradient of the loss and return the sum over the nearest neighbour with the scale factor used. Figure 2 shows how it has been implemented.

$$\begin{bmatrix} 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 3 & 3 & 4 & 4 \\ 3 & 3 & 4 & 4 \end{bmatrix} \rightarrow \begin{bmatrix} 4 & 8 \\ 12 & 16 \end{bmatrix} \tag{2}$$

The implementation of the forward pass for the Upsampling simply consists of the forward pass of NNUpsampling followed by the forward of the 2d convolution. The 2d convolution uses as input the output of the NNUpsampling's forward. For the backward, it is the opposite.

Since only the convolution has parameters, the only parameters for the upsampling layer will be the weight and the bias.

2.4 Activation functions : ReLU and Sigmoid

The activation functions ReLU and Sigmoid have the standard implementation with no parameters, the forward and backward passes to compute the gradient with respect to input.

2.5 Loss function : Mean squared error

2.6 Optimizer : Stochastic gradient descent

2.7 Container : Sequential

3 Results

References

[1] Sutskever, Ilya and Martens, James and Dahl, George and Hinton, Geoffrey, *On the importance of initialization and momentum in deep learning*, PMLR, 2013, https://proceedings.mlr.press/v28/sutskever13.html

[2] Fangyu Zou and Li Shen and Zequn Jie and Ju Sun and Wei Liu, *Weighted AdaGrad with Unified Momentum*, 2019, arXiv:1808.03408

[3] Diederik P. Kingma, University of Amsterdam, OpenAI , Jimmy Lei Ba, University of Toronto, *adam: a method for stochastic optimization*, Published as a conference paper at ICLR 2015, 30 Jan 2017 , arXiv:1412.6980v9

[4] Glorot, A. Bordes, and Y. Bengio, *Deep sparse rectifier neural networks*, In International Conference on Artificial Intelligence and Statistics (AISTATS), 2011.

[5] Krizhevsky, I. Sutskever, and G. Hinton, *Imagenet classification with deep convolutional neural networks*, Neural Information Processing Systems (NIPS), 2012.

[6] L. Maas, A. Y. Hannun, and A. Y. Ng, *Rectifier nonlinearities improve neural network acoustic models*, ICML Workshop on Deep Learning for Audio, Speech and Language Processing, 2013.

[7] He, X. Zhang, S. Ren, and J. Sun, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*, CoRR, abs/1502.01852, 2015.

[8] Xu, N. Wang, T. Chen, and M. Li, *Empirical evaluation of rectified activations in convolutional network*, CoRR, abs/1505.00853, 2015.

[9] Clevert, T. Unterthiner, and S. Hochreiter, *Fast and accurate deep network learning by exponential linear units (elus)*, CoRR, abs/1511.07289, 2015.

[10] simonyan and zisser 2014

[11] Glorot and Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks*, International Conference on Artificial Intelligence and Statistics (AISTATS), 2010.

[12] Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, *Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies*, pages 237–243. IEEE Press, 2001.

[13] Balduzzi, M. Frean, L. Leary, J. Lewis, K. Wan-Duo Ma, and B. McWilliams, *The shattered gradients problem: If resnets are the answer, then what is the question?*, CoRR, abs/1702.08591, 2017

A Appendix : supplementary figures