Labs
Machine Learning Course
Fall 2023

EPFL
School of Computer and Communication Sciences
Nicolas Flammarion & Martin Jaggi
www.epfl.ch/labs/mlo/machine-learning-cs-433

Problem Set 08, Nov 06, 2023 (Neural Network Training and Convolutional Networks)

Goals. The goals of this exercise are to:

- Analyze the computational cost and memory needed for forward and backward passes.
- Explore the standard variance-preserving parameter initialization scheme.
- Explore some algorithmic properties of Adam and SGD with momentum.
- Analyze the receptive field of convolutional networks.
- Define and train convolutional and residual networks in PyTorch.

Problem 1 (Computation and Memory Requirements of Neural Networks):

Let's consider a fully connected neural network with L layers in total, all of width K. The input is a mini-batch of size $n \times K$. For this exercise we will only consider the matrix multiplications, ignoring biases and activation functions.

- How many multiplications are needed in a forward pass (inference)?
- How many multiplications are needed in a forward + backward pass (training)?
- How much memory is needed for an inference forward pass? The memory needed is the sum of the memory needed for activations and weights. Assume activations are deleted as soon as they are no longer required.
- How much memory is needed for a training forward + backward pass? Note that during training we additionally use the forward activations for the computation of the derivatives.

Problem 2 (Variance Preserving Weight Initialization for ReLUs):

When training neural networks it is desirable to keep the variance of activations roughly constant across layers. Let's assume we have $y = \text{ReLU}(\mathbf{x}^\top \mathbf{w})$ for $\mathbf{x}, \mathbf{w} \in \mathbb{R}^d$. Further assume that all elements $\{w_i\}_{i=1}^d$ and $\{x_i\}_{i=1}^d$ are independent with $w_i \sim \mathcal{N}(0, \sigma)$ and $x_i \sim \mathcal{N}(0, 1)$.

Derive $\text{Var}[y]$ as a function of d and σ i.e. how should we set σ to have $\text{Var}[y] = 1$.

Hint: Remember the law of total expecation i.e. $\mathsf{E}_A[A] = \mathsf{E}_B[\mathsf{E}_A[A|B]]$

Problem 3 (Adam and SGDM):

SGD with momentum (SGDM) and Adam are two very commonly used optimizers in deep learning. Both are example of first order optimization methods that update the weights based on their gradients after some processing. The two algorithms are given below. Note that both of these algorithms act on each scalar parameter independently, and do not consider whether a parameter is a part of a larger vector/matrix/tensor.

Adam:

$$m_w^{(t+1)} \leftarrow \beta_1 m_w^{(t)} + (1-\beta_1)\nabla_w L^{(t)} \tag{1}$$
$$v_w^{(t+1)} \leftarrow \beta_2 v_w^{(t)} + (1-\beta_2)(\nabla_w L^{(t)})^2 \tag{2}$$
$$\hat{m}_w = \frac{m_w^{(t+1)}}{1-\beta_1^{t+1}} \tag{3}$$
$$\hat{v}_w = \frac{v_w^{(t+1)}}{1-\beta_2^{t+1}} \tag{4}$$
$$w^{(t+1)} \leftarrow w^{(t)} - \eta\frac{\hat{m}_w}{\sqrt{\hat{v}_w}+\epsilon} \tag{5}$$

SGDM:

$$m_w^{(t+1)} \leftarrow \beta m_w^{(t)} + \nabla_w L^{(t)} \tag{6}$$
$$w^{(t+1)} \leftarrow w^{(t)} - \eta m_w^{(t+1)} \tag{7}$$

For both algorithms, $L^{(t)}$ is the loss for time t (typically this is the loss for a mini-batch of samples), and $w^{(t)}$ represents the value of the parameter at step t. The algorithm shows an update for a single parameter but all model parameters are updated in the same way for each timestep t. Both optimizers use an exponential moving average of the gradient called momentum (represented by $m_w^{(t)}$). Adam additionally uses an exponential moving average of the square gradient ($v_w^{(t)}$) and also computes a "bias correction" for m and v given by $\hat{m}$ and $\hat{v}$. In both cases we consider the intial state to be zero, i.e. $m_w^{(0)} = v_w^{(0)} = 0$. The hyperparameters and their possible values are $\eta \geq 0, 0 < \beta_1 < 1, 0 < \beta_2 < 1, \epsilon \geq 0$ for Adam and $\eta \geq 0, 0 < \beta < 1$ for SGDM.

1. How many values does each optimizer need to store for a given parameter to perform the next update? This factor determines the memory usage of the optimizer.
2. Let's assume the gradient is a constant $\nabla_w L^{(t)} = c > 0$ for all $t \geq 0$ and $\epsilon = 0$. Compute the value of $w, m_w, v_w, \hat{m}_w, \hat{v}_w$ for timestep t and both optimizers (where applicable). Assume $w^{(0)} = 0$ for this question. How does w depend on c in each case?

Problem 4 (Receptive Field of Convolutions):

Convolutions can occur in one or more dimensions. In class you learned about 2D convolutions but both 1D and 3D convolutions are used in certain areas as well (for signals of a corresponding dimension). 1D convolutions are easy to visualize and many insights about them generalize to higher dimensions. You can view a 1D convolution as a special case of 2D convolution where the height of the input and filter is equal to 1.

In this exercise we will explore how the size of the output depends on the input size and parameters of a convolution in 1D. We will then use this to analyze the receptive field of a convolutional networks. The receptive field of a given activation is the area of the input that can affect its value. This is important to keep in mind when working with convolutional networks since the receptive field must be sufficiently large for certain features to be learned. For example, if your convolutional network was looking for a certain phrase in an audio signal, the receptive field of the later neurons should be sufficiently long to cover the length of the phrase.

The output size of a 1D convolution is depends on the dimensions of the input as well as the kernel size K, the padding P and the stride S. Padding is applied to both sides of the input signal and adds P values to each side, typically zeros (but various other forms of padding also exist). After adding a given amount of padding, a convolution only computes outputs where the filter can be fully "overlayed" on the padded input signal. A convolution with stride S only computes every S-th element of the output (starting with the first valid position on the edge). In modern networks we often use strided convolutions instead of adding pooling layers.

- Let's assume we have a 1D convolution with input X of width W_{in}, a kernel size of K, padding P and stride S. What is the size W_{out} of the output Y?
- Given an output size W_{out} for the convolution above, what is the minimum size of the input, W_{in}?
- Given a sequence of L convolutions with kernel sizes $K^{(1)}, \ldots K^{(L)}$, padding $P^{(1)}, \ldots P^{(L)}$, and strides $S^{(1)}, \ldots S^{(L)}$, what is the receptive field of an output element of the last convolution? You can assume that the input is larger than the receptive field (otherwise the definition is unclear).

Hint: Does padding affect the receptive field? Start with an output width of 1 and work your way backwards using the results of the previous parts. You don't have to simplify the resulting recurrence relation.

Problem 5 (Convolutional and Residual Networks in PyTorch):

The accompaning Jupyter Notebook has two PyTorch coding excercises. We recommend running the notebook on **Google Colab** which provides you with a free GPU and does not require installing any packages.

1. Open the colab link for the lab 8:
   `https://colab.research.google.com/github/epfml/ML_course/blob/master/labs/ex08/template/ex08.ipynb`
2. To save your progress, click on *"File > Save a Copy in Drive"* to get your own copy of the notebook.
3. Click 'connect' on top right to make the notebook executable (or 'open in playground').
4. Work your way through the introduction and exercises.

Alternatively you can download the notebook from GitHub and install PyTorch locally, see the instructions on **pytorch.org**.