Course: DD2424 - Assignment 3

In this assignment you will train and test a three layer network with multiple outputs to classify images from the CIFAR-10 dataset. The first layer will be a convolution layer applied with a stride equal to the width of the filter. In the parlance of computer vision this corresponds to a *patchify* layer, see figure 1, and this is the first layer that is applied in the Vision Transformer and other architectures such as MLPMixer and ConvMixer. You will train the network using mini-batch gradient descent applied to a cost function computing the cross-entropy loss of the classifier applied to the labelled training data and an L_2 regularization term on the weight matrix.

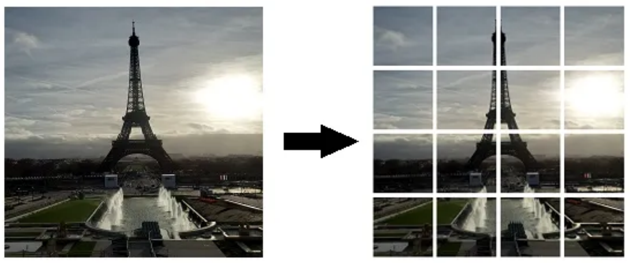

Figure 1: To *patchify* an input image, split it into a regular grid of non-overlapping sub-regions. For Vision Transformers the pixel data in each patch is flattened into a vector, transformed with an affine transformation and this output vector becomes an input to a Transformer network. The *patchify* operation is just a convolution applied with stride equal to the width of the filter.

The overall structure of your code for this assignment should mimic that from the previous assignments. You will have slightly different parameters than before and you will have to change the functions that **1**) evaluate the network (the forward pass) and **2**) compute the gradients (the backward pass). There is some work to do to get an efficient implementation of the first convolutional layer to allow reasonable training times without a GPU! But the reward will be an increased performance from assignment 2 (don't get your expectations too high though).

Background 1: *Network with an initial patchify layer*

Each input image X in its original form is a 3D array of size $32 \times 32 \times 3$.

The network function you will apply to the input X is:

$$H_i = \max(0, X * F_i) \quad \text{for } i = 1, \ldots, n_f \tag{1}$$

$$\mathbf{h} = \begin{pmatrix} \text{vec}(H_1) \\ \text{vec}(H_2) \\ \vdots \\ \text{vec}(H_{n_f}) \end{pmatrix} \tag{2}$$

$$\mathbf{x}_1 = \max(0, W_1 \, \mathbf{h} + \mathbf{b}_1) \tag{3}$$

$$\mathbf{s} = W_2 \, \mathbf{x}_1 + \mathbf{b}_2 \tag{4}$$

$$\mathbf{p} = \text{SoftMax}(\mathbf{s}) \tag{5}$$

where

- the input image X has size $32 \times 32 \times 3$
- Each filter F_i has size $f \times f \times 3$ and is applied with stride f and no zero-padding. The possible values for $f \in \{2, 4, 8, 16\}$. We denote the set of layer 1 filters as $\mathbf{F} = \{F_1, \ldots, F_{n_f}\}$.
- The output of each convolution H_i has size $32/f \times 32/f \times 1$.
- The H_i's are each flattened with the vec$(\cdot)$ operation and concatenated so $\mathbf{h}$ will have size $n_f \, n_p \times 1$ where $n_p = (32/f)^2$ is the number of the sub-patches to which the filter is applied.
- Two successive fully connected layers are then applied where W_1 has size $d \times d_0$ with $d_0 = n_f \, n_p$ and W_2 has size $K \times d$. The last weight matrix implies $\implies$ $\mathbf{s}$ has size $K \times 1$.
- SOFTMAX is defined as

$$\textsc{softmax}(\mathbf{s}) = \frac{\exp(\mathbf{s})}{\mathbf{1}^T \exp(\mathbf{s})} \tag{6}$$

- The predicted class corresponds to the label with the highest probability:

$$k^* = \arg \max_{1 \leq k \leq K} \{p_1, \ldots, p_K\} \tag{7}$$

In the lecture notes I have the convention that the operator vec$(\cdot)$ makes a vector by traversing the elements of the matrix row by row from left to right. This simple example illustrates:

$$H = \begin{pmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{pmatrix} \quad \implies \quad \text{vec}(H) = \begin{pmatrix} H_{11} \\ H_{12} \\ H_{21} \\ H_{22} \end{pmatrix}$$

I have omitted all bias terms in the network for the sake of clarity and simplicity of implementation.

Background 2: *Writing the convolution as a matrix multiplication*

To make the back-propagation algorithm transparent and relatively efficient (as we will be running this on CPU not a GPU and to take computational advantage of fast matrix operations) we will set up the convolutions as matrix multiplications.

Let's consider this simple example where X is the 4×4 input matrix and F is $2 \times 2 \times 2$ filter

$$X = \begin{pmatrix} X_{11} & X_{12} & X_{13} & X_{14} \\ X_{21} & X_{22} & X_{23} & X_{24} \\ X_{31} & X_{32} & X_{33} & X_{34} \\ X_{41} & X_{42} & X_{43} & X_{44} \end{pmatrix}, \quad F = \begin{pmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{pmatrix}. \tag{8}$$

When we convolve X with F (stride 2 and no zero-padding) the output vector has size 2×2. We can write this convolution as a matrix convolution:

$$H = X * F \qquad \Longrightarrow \qquad \mathbf{h} = M_X \, \mathrm{vec}(F) \tag{9}$$

where M_X is the matrix of size 4×4

$$M_X = \begin{pmatrix} X_{11} & X_{12} & X_{21} & X_{22} \\ X_{13} & X_{14} & X_{23} & X_{24} \\ X_{31} & X_{32} & X_{41} & X_{42} \\ X_{33} & X_{34} & X_{43} & X_{44} \end{pmatrix} \tag{10}$$

The rows of M_X correspond to the all sub-blocks of X which are involved with a dot-product with F when the convolution with stride f and no zero-padding is applied.

Multiple Channels The input image has multiple channels corresponding to the red, green and blue channels of an image. We will extend the tutorial example to the case where X is a 3D array. Let the input matrix X have size $4 \times 4 \times 2$ and the filter F has size $2 \times 2 \times 2$. Then:

$$X(:,:,1) = \begin{pmatrix} X_{111} & X_{121} & X_{131} & X_{141} \\ X_{211} & X_{221} & X_{231} & X_{241} \\ X_{311} & X_{321} & X_{331} & X_{341} \\ X_{411} & X_{421} & X_{431} & X_{441} \end{pmatrix}, \quad X(:,:,2) = \begin{pmatrix} X_{112} & X_{122} & X_{132} & X_{142} \\ X_{212} & X_{222} & X_{232} & X_{242} \\ X_{312} & X_{322} & X_{332} & X_{342} \\ X_{412} & X_{422} & X_{432} & X_{442} \end{pmatrix} \tag{11}$$

and the definition of F is extended from equation (8) similarly. Now when we convolve X with F (stride 2 and no zero-padding) the output vector still has size 2×2. Once again we can write this convolution as a matrix convolution but in this case:

$$M_X = \begin{pmatrix} X_{111} & X_{121} & X_{211} & X_{221} & X_{112} & X_{122} & X_{212} & X_{222} \\ X_{131} & X_{141} & X_{231} & X_{241} & X_{132} & X_{142} & X_{232} & X_{242} \\ X_{311} & X_{321} & X_{411} & X_{421} & X_{312} & X_{322} & X_{412} & X_{422} \\ X_{331} & X_{341} & X_{431} & X_{441} & X_{332} & X_{342} & X_{432} & X_{442} \end{pmatrix} \tag{12}$$

and

$$\text{vec}(F) = \begin{pmatrix} F_{111} & F_{121} & F_{211} & F_{221} & F_{112} & F_{122} & F_{212} & F_{222} \end{pmatrix}^T \tag{13}$$

Note here we have flattened F channel by channel. You can, of course, flatten F in any order you like but you have to ensure that M_X is defined to be consistent with this ordering!

Propagating the gradient to F To back-prop the gradient back to filter F, as we have written the convolution as a matrix multiplication, you can

$$\frac{\partial L}{\partial \text{vec}(F)} = M_X^T \, \mathbf{g} \tag{14}$$

where $\mathbf{g}$ is the vector of size 4×1 containing the gradient of the loss relative to the vector $\mathbf{h}$. Here we have applied the convention the gradient of the loss w.r.t. to a vector is a column vector. Apologies for the slight inconsistencies w.r.t. the lecture notes. If there is a batch of input images then

$$\frac{\partial L}{\partial \text{vec}(F)} = \frac{1}{|\mathcal{B}|} \sum_{(X,y) \in \mathcal{B}} M_X^T \, \mathbf{g}_y \tag{15}$$

where L is the average cross-entropy loss for the batch.

Background 3: *Apply multiple convolution filters*

Returning to consider our simple example, we now review the case when we apply 3 filters F_1, F_2 and F_3 to each patch. These three filters can be applied efficiently to the region patches defined in M_X by concatenating the flattened versions of each F_i into an 8×3 matrix F_{all} that is

$$H = M_X \, F_{\text{all}} \tag{16}$$

where H is 4×3 and

$$F_{\text{all}} = \left[\text{vec}(F_1), \text{vec}(F_2), \text{vec}(F_3)\right] \tag{17}$$

$$\frac{\partial L}{\partial F_{\text{all}}} = M_X^T G \tag{18}$$

where G is the 4×3 matrix constructed by concatenating the $\mathbf{g}_i$'s

$$G = \left[\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3\right] \tag{19}$$

and M_X is the same 4×8 matrix defined in equation (12).

Background 4: *Equations for the back-propagation algorithm*

As per usual let $L(\cdot, \cdot)$ represent the loss for the mini-batch of data $\mathcal{B}$

$$L(\mathcal{B}, \Theta) = \frac{1}{|\mathcal{B}|} \sum_{(\mathbf{x}, y) \in \mathcal{B}} l_{\text{cross}}(y, f_{\text{network}}(X, \Theta)) \tag{20}$$

where $l_{\text{cross}}(\cdot)$ is the usual cross-entropy loss and f_{network} represents the function corresponding to our network. To learn the parameters $\Theta = \{\mathbf{F}_{\text{all}}, W_1, W_2\}$ we need to compute the gradient of the batch loss w.r.t. each parameter.

Given the probabilistic outputs of the network for all the images in the mini-batch, then the gradient of the batch loss w.r.t. W_1 and W_2 is computed as in Assignment 2. The novel quantity to be computed is the gradient w.r.t. $\mathbf{F}_{\text{all}}$. Given the expression of the gradient for one input in equation (18), the gradient for the batch is:

$$\frac{\partial L(\mathcal{B})}{\partial F_{\text{all}}} = \frac{1}{|\mathcal{B}|} \sum_{(X,y) \in \mathcal{B}} \frac{\partial\, l_{\text{cross}}(y, f_{\text{network}}(X, \Theta))}{\partial\, F_{\text{all}}} = \frac{1}{|\mathcal{B}|} \sum_{(X,y) \in \mathcal{B}} M_X^T G_y \tag{21}$$

where we have used the subscript y for G_y to denote the 2D array $(n_p \times n_f)$ of gradients, as defined in equation (19), for a particular input. In code you will store the "M_X" representations of each input in a 3D array M of size $n_p \times 3f^2 \times n$ and the gradients for the H nodes for each input in another 3D array G of size $n_p \times n_f \times n$. Then

$$F_{\text{all}}^{\text{grad}} = \frac{1}{n} \sum_{i=1}^{n} M(:, :, i)^T G(:, :, i) \tag{22}$$

where $F_{\text{all}}^{\text{grad}}$ has size $3f^2 \times n_f$ and contains the gradient of the loss w.r.t. all the conlution filters.

Background 5: *Label smoothing another form of regularization*

As you begin to fit larger models you need to apply more regularization to allow longer training runs without over-fitting. These longer runs are required to train the best possible version of your network! In this training scenario a simple and complimentary approach to other regularization techniques such as data-augmentation, weight decay, is *label smoothing*. Label smoothing works as follows. Assume you have a labelled training example (X, y) where $y \in \{1, \ldots, K\}$. Let $\mathbf{y}$ represent the one-hot encoding of label y. This is the usual target output for cross-entropy training. However, this target vector does not necessarily need to be a one-hot encoded vector. One can instead spread some of the probability from the ground truth class to the other classes. Let ϵ be small number in $[0, 1)$. The ith entry of the label smoothed target vector $\mathbf{y}_{\text{smooth}}$ is defined by

$$y_{\text{smooth},i} = \begin{cases} 1 - \epsilon & \text{if } i = y \\ \epsilon/(K-1) & \text{otherwise} \end{cases} \tag{23}$$

Typically we set $\epsilon = .1$. Once you have defined $\mathbf{y}_{\text{smooth}}$ then you only have to make a minimal change in your training algorithm. In the backward-pass to compute the gradients when you propagate the gradient through cross-entropy loss and softmax operations you should make this replacement:

$$-(\mathbf{y} - \mathbf{p}) \qquad \Longrightarrow \qquad -(\mathbf{y}_{\text{smooth}} - \mathbf{p}) \tag{24}$$

Thus label smoothing also has the benefit that it has minimal computational overhead per update iteration. Though, of course, similar to over regularization techniques if applied then longer training is required as you have made the training task harder.

Background 6: *Cyclical learning rates with increasing step sizes*

In Assignment 2 you trained with cyclical learning rates. For most of the experiments you will use this optimizer again for this assignment. However, you will apply a small *upgrade* to this algorithm for longer training runs to help with efficiency, that is we will have shorter cycles at the start of training. The upgrade is that the number of update steps per cycle is doubled after each cycle. Let $n_{i,s}$ be the number of steps for the ith cycle then

$$n_{i+1,s} = 2\, n_{i,s} \tag{25}$$

This schedule of the learning rate is an approximation to the cosine with warm restarts schedule of [Loshchilov and Hutter, 2017]. For an even more accurate approximation you should also decay $\eta_{\max}$ for each new cycle but in the basic assignment we will not do this.

Exercise 1: *Write code to implement the convolution efficiently*