Assignment 2 Report

CS5242 Neural Network and Deep Learning

Meng-Jiun Chiou

October 18, 2017

1 Introduction

Convolutional neural network has been a popular approach for computer vision problem since the breakthrough in 2012 [1]. To realize how convolutional neural network became successful, we have to fully understand the architecture of it. This assignment let students building a convolutional neural network from scratch. Students are required to complete both the forward and backward passes of the network, including convolutional layers, non linearity layer, pooling layer, dropout layer and fully connected layer. Finally, training the convolutional neural network on the CIFAR dataset which is limited to only two classes.

The modules used in the network are discussed in section two, then we present a couple of different modules used in our neural network. We also introduce the architecture used in the experimenton the dataset with different hyperparameters in section three, followed by the result and discussion in section four.

2 Modules

2.1 Forward Pass of Convolution Layers

The convolution layers do the convolution process between the filters (kernel) and receptive field on the image. We denote the height and the width of filters are l_h and l_w, vertical and horizontal padding sizes are p_h and p_w, vertical and horizontal kernel sizes are k_h and k_w, vertical and horizontal strides are s_h and s_w, the number of kernels is f, then we can calculate the output shapes as

$$(f, o_h, o_w) = (f,\ \lfloor \frac{(l_h + p_h - k_h)}{s_h} \rfloor + 1,\ \lfloor \frac{(l_w + p_w - k_w)}{s_w} \rfloor + 1)$$

When implementing 2D convolution, filters and image patches should be converted into vector form to speed up the multiplication. After the transformation, multiple kernels with multiple channels should have the shape of $(f,\ c * k_h * k_w)$ and the image patch has the shape of $(c * k_h * k_w,\ o_h * o_w)$.

What the function *conv_forward* does are converting the filters and image patches into vector form, performing dot product on them, adding the biases, finally reshaping into the correct form. That is, it computes the class scores

$$scores = W * X + b$$

where W denotes weights, X denotes input data and b denotes biases.

2.2 Backward Pass of Convolution Layers

Let $\delta^{(l+1)}$ be the error term for the $(l+1)$-st layer in the network with a cost function $J(W, b; x, y)$, if l-th layer is a convolutional and subsampling layer, then the propogated error is

$$\delta_k^l = ((W_k^{(l)})^T \delta_k^{(l+1)}) \cdot f'(z_k^{(l)})$$

where k is the index of the filter and $f'(z_k^{(l)})$ is the derivative of the activation function. Also, we also want to calculate the derivative of error w.r.t. weights

$$\bigtriangledown_{W_k^{(l)}} J(W, b; x, y) = \sum_{i=1}^{m} (a_i^(l)) * rot90(\delta_k^{(l+1)}, 2)$$

and the derivative of error w.r.t. biases

$$\bigtriangledown_{b_k^{(l)}} J(W, b; x, y) = \sum_{a,b} (\delta_k^{(l+1)})_{a,b}.$$

2.3 Forward Pass of ReLU Layers

Rectified Linear Unit (ReLU) is a very common choice of activation functions. It has a very easy form

$$f(x) = max(0, x)$$

ReLU was observed to be able to accelerate the convergence of stochastic gradient descent compared [1] to sigmoid or tanh functions because of its linearity and non-saturating form. It can also be implemented easily by thresholding a matrix at zero.

2.4 Backward Pass of ReLU Layers

Derivative of ReLU is also easy to calculate. It is a mask of x that is 1 if x is greater or equal to 0, 0 in other conditions. In backward pass, we just put on the mask to backpropagae the errors.

2.5 Forward Pass of Max Pooling Layers

In max pooling layers, we downsample the feature map by choosing the maximum value in the receptive field. Similar to the forward pass of convolutional layer, we need to calculate the output shape

$$(f, o_h, o_w) = (f, \lfloor \frac{(l_h - p_h)}{s_h} \rfloor + 1, \lfloor \frac{(l_w - p_w)}{s_w} \rfloor + 1)$$

and taking the maximum from the patch of the input.

2.6 Backward Pass of Max Pooling Layers

The backward pass of max pooling is put the gradient w.r.t. x back to the place where the maximum value was. Therefore we form another mask which is 1 if the position is the maximum value and 0 if not.

2.7 Forward and Backward Pass of Dropout Layers

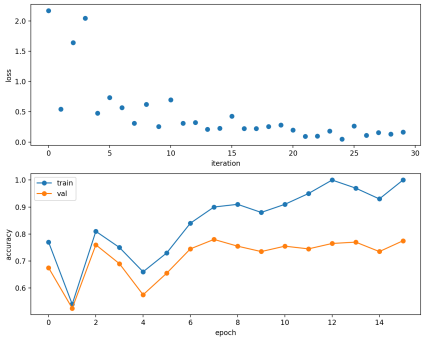
Dropout is a technique that prevents overfitting and provides a way of approximately combining many different neural network architectures efficiently. It works by randomly setting some neurons off (zero-out) to add variance to the network. The probability of zero-out is usually set to 0.5 after convolutional layers or 0.8 after fully connected layers [2].
Similar, the backward pass of dropout simply applying mask to the derivative. Note that we do not apply dropout when testing in both forward and backward passes.

3 Architecture

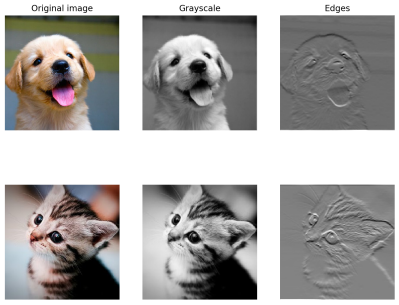
The final architecture of the convolutional neural network is:

$$Conv - ReLU - (Dropout) - 2 * 2\, maxpool - FCN - (Dropout) - ReLU - FCN - Softmax$$

We check the effect of dropout layer by comparing the performance with/without it.

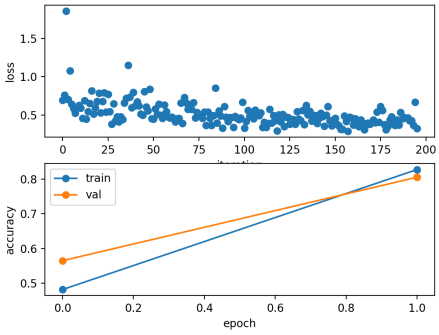


(a) Loss and accuracy when overfitting a small dataset

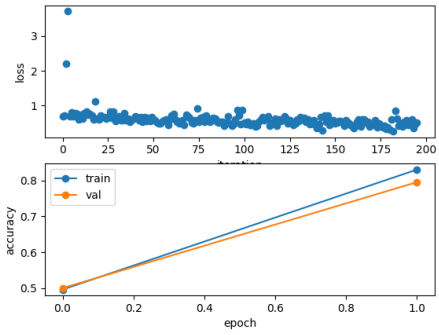


(b) Using convolutional layer to perform grayscale and image detection.

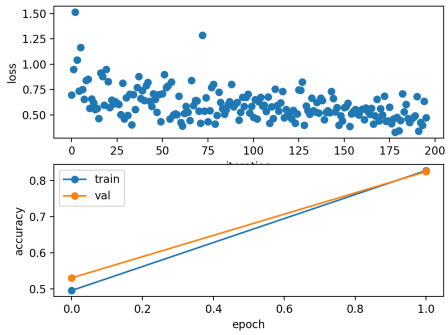
Figure 1: Results output from sample code



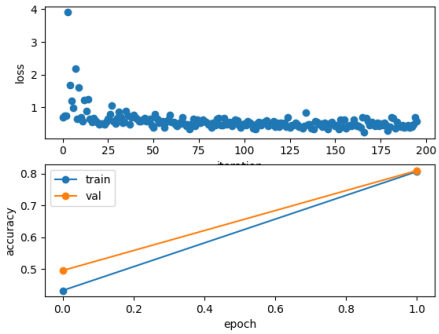
(a) Default setting (without dropout).



(b) Two dropout layers are added into the network and the dropout rates are 0.5 and 0.8, respectively.



(c) Two dropout layers are added into the network and the dropout rates are 0.7 and 0.5, respectively.



(d) Two dropout layers are added into the network and the dropout rates are 0.5 and 0.8, respectively. In addition, learning rate is set to a smaller value 10^{-4}.

Figure 2: Training Results of the three-layer convolutional Neural Network

4 Result

As shows in figure 1a, when we train our network on a small dataset, it results to bigger difference between high training accuracy and low validation accuracy. Figure 1b shows the convolutional layer can also be used

to perform some technique of image processing, including grayscale and edge detection.

Firstly, we follow the original setting to train our three-layer convolutional neural network. The final training and validation accuracy 82.67 and 80.5 respectively, which is a acceptable result.

We then add two dropout layers to the network, one is behind the first convolutional layer and ReLU layer and the other is behind the affine layer. We follow [2] to set the dropout rate, 0.5 for the first and 0.8 for the second dropout layer. The result is showed in figure 2b. The training and validation accuracy are 82.8 and 82.5, higher than the result without the dropout layers. The small depth of the network is guessed to be the reason why accuracies were not increased too much.

To find the effect of dropout layer, We then choose other dropout rate, which are 0.7 and 0.5, respectively. Figure 2c shows the result. Training and validation accuracy are 83 and 79.5, which is lower even than default setting. Also, the loss is vibrating apparently. We speculate these are caused by inappropriate setting of dropout rate.

2d shows the change the learning rate to 10^{-4} has almost no change to the performance, except for slower convergence (decrease speed of loss).

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[2] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting.," *Journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.