Faculty of Engineering

Ain Shams University

2021-2022

Neural Networks and Their Applications

CSE 616

PG2015

Assignment #: 2

Title: ***CNNs and Training Optimizations***

Made by:

2100966 حمدي أسامه محمد السيد فرج

Mechatronics PG Student

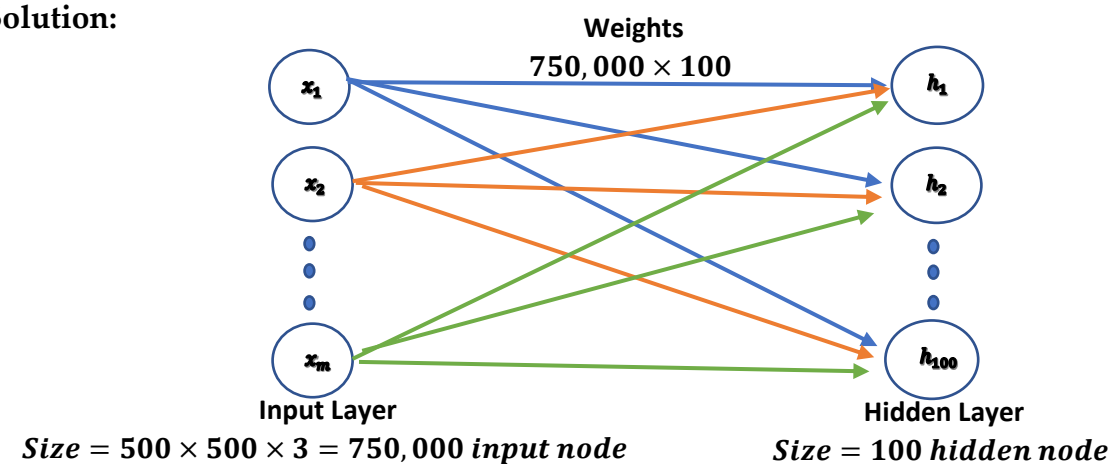
Neural Networks – CSE 616

Assignment 2 Solution

Made by: Hamdy Osama (2100966)

1- Consider an input image of shape 500x500x3. The image is flattened and a fully connected layer with 100 hidden units is used. What is the shape of the weight matrix of this layer (without the bias)? What is the shape of the bias?

Solution:



$$\text{Weight Matrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & \dots & w_{1,750000} \\ w_{2,1} & w_{2,2} & w_{2,3} & \dots & w_{2,750000} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ w_{100,1} & w_{100,2} & w_{100,3} & \dots & w_{100,750000} \end{bmatrix}_{100\times750000}$$

Weights connected to the first node of the hidden layer

Weights connected to the N node of the hidden layer

75000000 weights in this layer

The bias has a shape of (100×1)as each hidden node has its bias.

2- You run this image in a convolutional layer with 10 filters of kernel size 5x5. How many parameters does this layer have?

Solution:

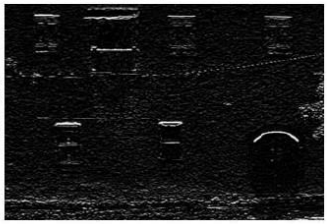
Number of Parameters in one filter (Note that the filter should be $\times 3$ to be compatible on RGB input image.) $= 5 \times 5 \times 3 = 75$

Number of Parameters in all 10 filters $= 75 \times 10 = 750$

Don't forget the bias (each filter has its bias parameter) $\rightarrow 750 + 10$

$=$ 760 Parameters in this layer

3- The top gray image has run through different types of filters and the results are shown in the following images. What type of convolutional filter was used to get each of the resulting images. Explain briefly and include the values of these filters. The filters have a shape of (3,3).



Solution:

$$Filter(1)\ used = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \qquad Filter(2)\ used = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

The filter captures the gradient change in the image. Filter (1) is used to capture vertical lines as the change in values of the filter parameters is horizontal. Filter (2) is used to capture horizontal lines because of its gradient direction that changes vertically. Other filters can be used with different parameters, but all should have the changing gradient in directions like the one proposed here in this solution.

4- In the Adam optimizer. Show what will happen the numbers of steps to compute the exponential moving averages gets large.

Solution:

Adam Optimizer uses a combination of the gradient descent with momentum algorithm and the RMSP algorithm. Momentum algorithm is used to accelerate the gradient descent algorithm by taking into consideration the exponentially weighted average of the gradients. RMSprop is an adaptive learning algorithm that tries to improve AdaGrad. Instead of taking the cumulative sum of squared gradients like in AdaGrad, it takes the exponential moving average. As the exponential moving average gets large, the number of steps to reach gradient minimum point decrease as each step is larger (higher learning rate). This is useful in cases where we get stuck in local minimum with poor results, so we sling shot ourselves out of this local minimum and descent further. As steps of training increase, the learning rate decreases as we fine tune to reach optimal point.

5- Given a batch of size *m,* and assume that a batch normalization layer takes an input $z=(z(1),\ldots,z(m))$ as an input. Write down the output equation(s) of this layer. Give two reasons for using the batch normalization layer.

Solution:

To calculate batch normalization, we get the mean of the batch, the variance of the batch, and then we normalize the batch. After that, we can let the network learn parameters to squash the range if it wants to.

$$\text{mean of batch} \rightarrow \ \mu_B = \frac{1}{m}\sum_{i=1}^{m} z_i$$

$$\text{variance of batch} \rightarrow \ \sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(z_i - \ \mu_B)^2$$

$$\text{Normalizing every input } (\hat{z}_i) \rightarrow \ \hat{z}_i = \frac{z_i - \ \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad \text{where } (i: 1 \rightarrow m)$$

note that ϵ is a small positive constant ($\approx 10^{-8}$)used to prevent dividing by zero

Scaling and shifting the output of the BN Layer$(y_i) \rightarrow y_i = \gamma\hat{z}_i + \beta$ where $(i: 1 \rightarrow m)$

where γ and β are parameters to be learned in the training of the network

Batch normalization is used to have faster learning rate; thus, it reduces the effect of diminishing gradient values of weights. Also, Batch normalization has a regularizing effect while the training phase

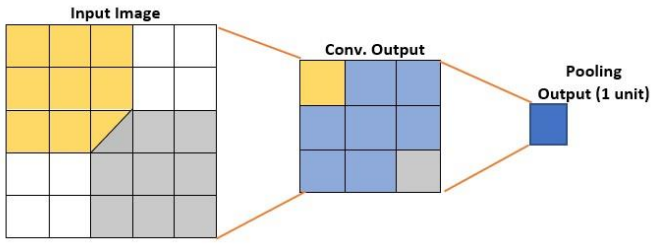
6- Suppose you have a convolutional network with the following architecture: The input is an RGB image of size 256 x 256. The first layer is a convolution layer with 32 feature maps and filters of size 3x3. It uses a stride of 1, so it has the same width and height as the original image. _ The next layer is a pooling layer with a stride of 2 (so it reduces the size of each dimension by a factor of 2) and pooling groups of size 3 x 3. Determine the size of the receptive _eld for a single unit in the pooling layer. (i.e., determine the size of the region of the input image which influences the activation of that unit.) You may assume the receptive field lies entirely within the image.

Solution: s_i = stride of layer i ;;; r_i = receptive field of layer i ;;; $k_i = filter\ size\ of\ layer\ i$

$r_{i-1} = s_i \times r_i + (k_i - s_i)$

$r_{input} = s_1 \times r_1 + (k_1 - s_1)$

$r_{input} = 1 \times 3 + (3 - 1) = 5$



The size of the input image influencing the activation of the pooling layer is 5×5 in each channel of RGB

7- If an input data block in a convolutional network has dimension $C \times H \times W = 96 \times 128 \times 128$, (96 channels, spatial dim 128x128) and we apply a convolutional filter to it of dimension $D \times C \times HF \times WF = 128 \times 96 \times 7 \times 7$, (i.e. a block of D=128 filters) with stride 2 and pad 3,what is the dimension of the output data block?

Solution:

$$\text{Output per channel} = \frac{(\mathrm{N} + 2 \times \mathrm{P} - \mathrm{F})}{\mathrm{S}} + 1 \;\rightarrow\; \text{where } \mathrm{N} = 128, \mathrm{P} = 3, \mathrm{F} = 7, \text{and } \mathrm{S} = 2$$

$$\text{Output per channel} = \frac{(128 + 2 \times 3 - 7)}{2} + 1 = 64.5$$

This stride is REJECTED because it results in output not a whole number of pixels

$$\text{By using a stride S of 1} \rightarrow \text{Output per channel} = \frac{(128 + 2 \times 3 - 7)}{1} + 1 = 128$$

$$\text{Dimension of output block} = \mathrm{D} \times 128 \times 128 = 128 \times 128 \times 128$$

8- What is inverted dropout and what is its advantage?

Solution:

Inverted dropout is a variant of the original dropout technique. Just like traditional dropout, inverted dropout randomly keeps some weights and sets others to zero. This is known as the "keep probability" p. The one difference is that, during the training of a neural network, inverted dropout scales the activations by the inverse of the keep probability $q=1-p$. This prevents network's activations from getting too large and does not require any changes to the network during evaluation. In contrast, traditional dropout requires scaling to be implemented during the test phase. Dropout in general is used as a regularization technique to prevent overfitting.

9- Explain briefly why fully connected neural networks do not work well for image classification.

Solution:

Neural networks used in image classification should be shift invariant to features present in the image. To use Fully connected neural networks, we need to train the network on all images containing the feature in all possible scaling and positions, which is not possible to make because it is computationally intensive. Instead, use CNN as a feature extractor.

10- Compute the convolution of the following two arrays: (4 1 -1 3) * (-2 1). Your answer should be an array of length 5. Show your detailed work.

Solution:

$$\text{Solving the following} \rightarrow [4 \quad 1 \quad -1 \quad 3] \star [-2 \quad 1]$$

$$\text{Flipping the kernel and then cross correlation} \rightarrow [4 \quad 1 \quad -1 \quad 3] \otimes [1 \quad -2]$$

$$\text{To get an answer to be an array of length of 5, we use padding by zero}$$

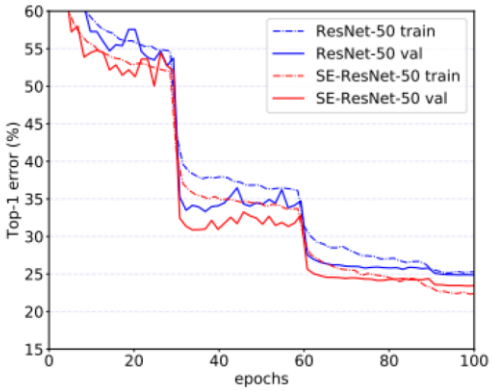
$$[0 \quad 4 \quad 1 \quad -1 \quad 3 \quad 0] \otimes [1 \quad -2]$$

$$[(0 \times 1) + (4 \times -2) \quad (4 \times 1) + (1 \times -2) \quad (1 \times 1) + (-1 \times -2) \quad (-1 \times 1) + (3 \times -2) \quad (3 \times 1) + (0 \times -2)]$$

$$[0 \quad 4 \quad 1 \quad -1 \quad 3 \quad 0] \otimes [1 \quad -2] = [-8 \quad 2 \quad 3 \quad -7 \quad 3]$$

11- Describe what setting change in epochs 25 and 60 could have produced this training curve. Be brief.

Solution:



In this training approach, the learning rate is a hyperparameter that is a changing variable. When the learning rate decay is used, the error decreases in a step manner as the learning rate is decreased to fine tune the neural network weight parameters to reach minimal error. Learning rate decay is used at epochs 25 and 60.

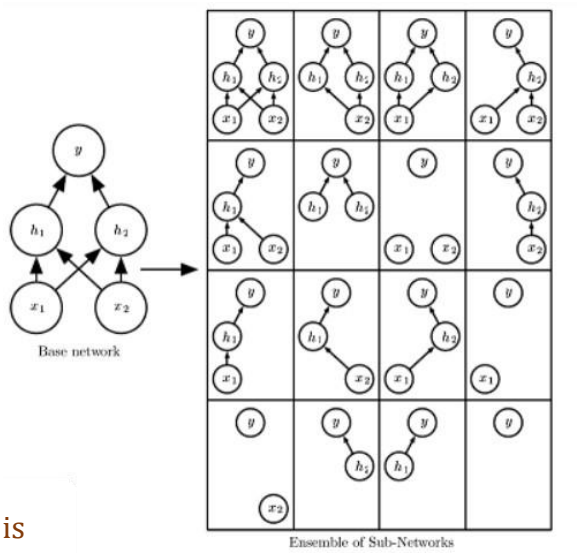
12- Why are convolutional layers more commonly used than fully connected layers for image processing?

Solution:

Because shift invariant networks are needed in image processing to extract features, Convolutional layers are used as scanning networks that look for the pattern they are trained to classify as they move through subsets of neurons of the previous layer. Convolutional layers use filters with receptive fields that scan the input image with the same parameters used by the filters in each scanned area of the image. Compared to fully connected neural networks, CNNs are generally faster to train because of their sparse weight matrix as the network is a shared parameters model (by using filters). Training the CNN uses conventional gradient descent and back propagation, but it is used to learn filter parameters instead of weights connecting all input neurons to hidden layer neurons as in conventional neural networks.

13- Dropout layers implement different forward functions at train and test time. Explain what they do. Let p be the probability that node value is *retained.*

Solution:



The dropout layer is used as a regularization technique to reduce overfitting. Overfitting occurs when the network performs well on the training data but performs poorly on the testing data as the neural network had high capacity (the network is more complex than needed). Dropout is used to randomly remove neurons from the layer during training so that the network can reach good accuracies using different neurons and not become dependent on few nodes of that layer. Dropout can be performed on all hidden and input layers of the network but is not made on the output layer.

During testing, Dropout doesn't perform the random omission of neurons as done in training because all neurons are already trained on extracting specific information from the coming data. Instead, the dropout layer scales the output of the neurons of a layer so that the output is not excessively larger due to the effect of using all available neurons. For example, if P is the probability that a node value is retained in training, the dropout layer will scale the output of that node by P.

14- Explain how standard momentum and Nesterov accelerated gradient differ. How does their performance (convergence as a function of time) differ on convex optimization problems? Use sketches as an aid.

Solution:

The standard momentum provides another term that is responsible in changing the value of the weights during training along with the gradient of weights. This term is known as the velocity of the changing gradient, and this velocity is increasing in each iteration because of the momentum of the changing gradient. A friction coefficient is found that slows the increase of velocity.

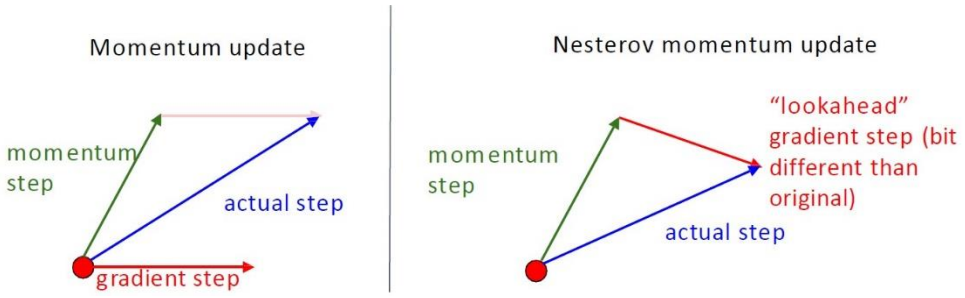
$v_{t+1} = \rho v_t + \nabla f(x_t)$ $\quad v \rightarrow velocity;\ \rho \rightarrow friction\ coefficient;\ \nabla f(x_t) \rightarrow gradient\ of\ weights$

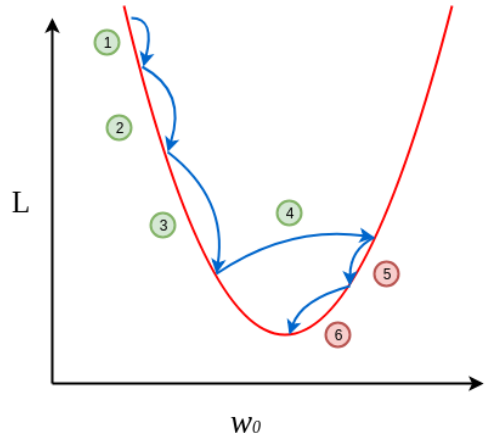
$x_{t+1} = x_t - \alpha v_{t+1}$ $\quad x_{t+1} \rightarrow weight\ after\ update\ ;\ x_t \rightarrow weight\ before\ update\ ;\ \alpha \rightarrow learning\ rate$

The Nesterov accelerated gradient provides a solution like momentum as it contains a velocity term with its friction coefficient, but the gradient term is different here in that it takes in consideration the change in the velocity term too, not only the change in weights as in the standard momentum method. Also, the learning rate affects the gradient term only not the velocity term, and the gradient term is being subtracted from the velocity term. This change in signs and gradient calculation allows the Nesterov method to reach the optimized point faster as the gradient has a look-ahead effect in which it bypasses steps in which the change in weight values will be negative like the error gradient as seen in the convex optimization illustration seen below.

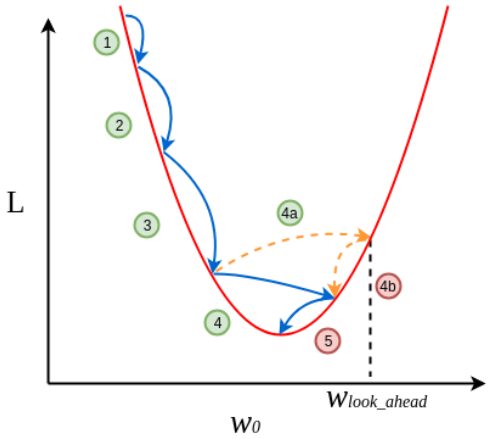
$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t)$$

$$x_{t+1} = x_t + v_{t+1}$$





(a) Momentum-Based Gradient Descent



(b) Nesterov Accelerated Gradient Descent

$$\bigcirc \Longrightarrow \frac{\partial L}{\partial w_0} = \frac{Negative(-)}{Positive(+)} \qquad \bigcirc \Longrightarrow \frac{\partial L}{\partial w_0} = \frac{Negative(-)}{Negative(-)}$$

15- How does the actual learning rate of ADAGRAD change with the number of steps?

Solution:

In conventional stochastic gradient descent, the value of the learning rate is used to be the same for each weight. But in Adagrad (Adaptive Gradient) Optimizer the core idea is that each weight has a different learning rate. As the number of steps during training increase, the learning rate decreases because we are near the optimum point, and we are fine tuning to reach the optimum point; thus, we need smaller steps, and this is caused by the smaller learning rate.

$$x_{t+1} = x_t - \alpha' \nabla f(x_t) \quad \textit{where } \alpha' \textit{ is the adaptive learning rate}$$

$$\alpha' = \frac{\alpha}{\sqrt{(g)^2 + \epsilon}} \quad \textit{where } \alpha \rightarrow \textit{learning rate}; \ \epsilon \rightarrow \textit{small constant prevent dividing by } 0$$

$$g = \sum_{i=1}^{t} \nabla f(x_i)$$

As seen from the equations above, the value of "g" is increasing as more iterations of updating weights come because its value is being incremented each iteration. This process leads to gradually decreasing the learning rate as we come near the end of the training process and will enable us to achieve faster convergence.