CS5242 August 2023, Assignment 2

Due week 8, 10 October 2023, 1800hours

Question 1 - Optimization (2.5 Marks)

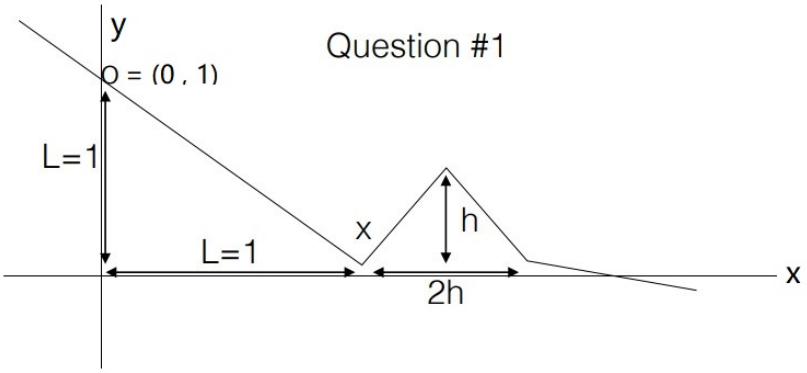

Figure 1: Illustration for question 1.

Figure 1 shows the plot of a 1D loss function. Apply gradient descend to optimize the loss function starting from point O. There is a bump at a distance $L = 1$ away with the bump dimension $h \times 2h$.

Question 1.1

Explain what happens when you apply the standard gradient descend algorithm. Let the learning rate of the gradient descend be $\eta = 0.3$. Given $h = 0.5$.

$$x \leftarrow x - \eta \frac{\partial L}{\partial x} \tag{1}$$

Question 1.2

Now, instead of gradient descend, apply adam optimisation with parameters given in figure 2. What is the max height 'h' of the bump in which the adam optimiser will escape the local min at 'x'? Use $\alpha = 0.3$ and $\epsilon = 0$ in your calculations. Round 'h' to 2 decimal places.

Question 1.3

Plot the following values with respect to t in a single plot and explain your observations. Use learning rate = 0.3

$$g_t, m_t, v_t, \hat{m}_t, \hat{v}_t, x \tag{2}$$

Question 1.4

Perform the Adam optimizer starting at $x = 2$ for the following function and plot $g_t, m_t, v_t, \hat{m}_t, \hat{v}_t, x$ w.r.t t. Explain your observations. Use learning rate = 0.3 for 30 iterations.

$$\text{ReLU}(x), \text{Sigmoid}(x), |x| \tag{3}$$

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Figure 2: Adam algorithm

Question 2 - Autoencoder (2.5 Marks)

Question 2.1

Make an autoencoder for the MNIST data set using the following architecture.

1. Input $28 \times 28 \times 1$.
2. Flatten 784
3. Fully connected $784 \rightarrow 196$ with ReLU.
4. Fully connected $196 \rightarrow 49$ with ReLU.
5. Fully connected $49 \rightarrow \lambda$ with ReLU.
6. Fully connected $\lambda \rightarrow 49$ with ReLU.
7. Fully connected $49 \rightarrow 196$ with ReLU.
8. Fully connected $196 \rightarrow 784$ with ReLU.
9. Reshape to $28 \times 28 \times 1$.

Keeping all the parameters the same (example: epochs, batch size, learning rate etc,.), train the autoencoder on the the whole of MNIST training dataset (60k images) using this architecture for $\lambda = 2$ and $\lambda = 32$. Use L1 loss function and adam optimizer.

- Explain what you notice about the results from these two scenarios?
- For both the scenarios, submit the training loss vs epochs, sample input image and the reconstructed output image. (1 image with 6 subimages)
- Submit the code *exxxxxxx_q2.1.py* or google colab link.

Question 2.2

Instead of fully connected layers, design the autoencoder with convolutional layers following the figure 3. Similar to question 2.1, use activation function ReLu, L1 loss function and adam optimizer and train the autoencoder on the MNIST training dataset for $\lambda = 2$ and $\lambda = 32$.

- Explain what you notice about the results from these two scenarios?

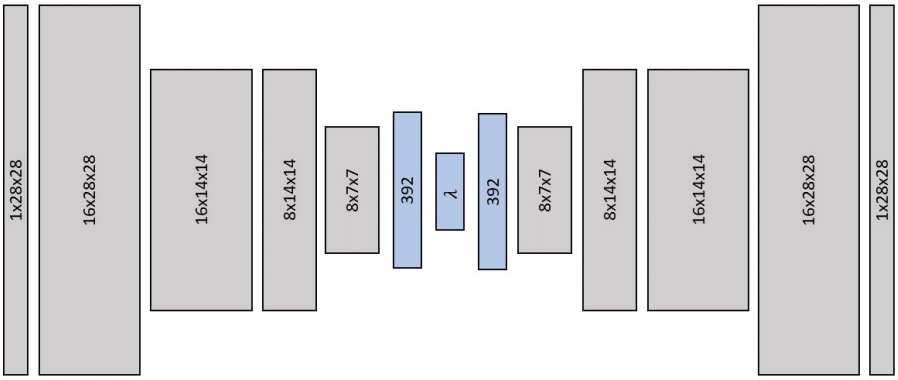

Figure 3: CNN artitechture

- For both the scenarios, submit the training loss vs epochs, sample input image and the reconstructed output image. (1 image with 6 subimages)
- Submit the code *exxxxxxx_q2.2.py* or google colab link.

Question 3: Common features - (2.5 Marks)

If an autoencoder is used to reconstruct MNIST images, we say that the latent variables contain the instance specific information and the weights and bias contain common features among instances in the data set. Then, how can we extract common features from a dataset?

Take the model trained in question 2.2 (both $\lambda = 2$ and $\lambda = 32$. Pass the whole MNIST training data set through the trained model. Record all the latent features $z_1, z_2, \cdots, z_{60000}$ and then take its mean value.

$$\mu = \frac{1}{60,000} \sum_{1}^{60,000} z_i \tag{4}$$

This mean value μ must contain common features (together with all weights and bias) of the data set and is the same value for all instances. Feed μ into the decoder and observe its output.

- Explain what you notice about the results?
- For both ($\lambda = 2$ and $\lambda = 32$), submit the reconstructed output image. (1 image with 2 subimages)
- Submit the code *exxxxxxx_q3.py* or google colab link.

Question 4: Understanding pytorch - (2.5 Marks)

Given: 32×1 pixel input (*pixel_input.pt*) and 3×1 filters ($filter.pt$). Tip: Use `torch.load` and `torch.save`.

Implement the following using pytorch to generate *torch_xxx_out* tensors. Re-implement again, this time without using any deep learning library to generate *my_xxx_out* tensors. Compare the output tensors and ensure that they match. Submit the code *exxxxxxx_q4.py* or the google colab link.

1. `torch.nn.MaxPool1d(kernel_size=2, stride=1, padding=0, dilation=1, return_indices=False, ceil_mode=False)`
2. `torch.nn.AvgPool1d(kernel_size=2, stride=1, padding=0, ceil_mode=False, count_include_pad=True)`
3. `torch.nn.functional.conv1d(input, filter, bias = None, stride = 1, padding = 0, dilation = 1, groups = 1)`
4. `torch.nn.Sigmoid()`

5. `torch.nn.BatchNorm1d(1, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True, device=None, dtype=None)`
6. `torch.nn.Linear(in_features, out_features, bias=True)`

Question 5: CIFAR10 - (2.5 Marks)

Question 5.1

Make a neural network with a few layers of convolution and fully connected layers. Construct your network from scratch. Do not download code and use, e.g. do not use pytorch to call Resnet. You may use low level pytorch function such as `nn.Linear` and `nn.Conv2d`. Train your network on the full CIFAR training dataset to achieve testing accuracy of $> 90\%$ on the CIFAR testing dataset.

- Were you able to achive more than ¿90% testing accuracy? Why was it difficult? Explain briefly what you did to achieve ¿90% testing accuracy?
- Submit the 1) Training loss vs Epochs 2) Training accuracy vs Epochs 3) Testing loss vs Epochs and 4) Testing acuracy vs Epochs
- Submit the code *exxxxxxx_q5.1.py* or google colab link.

Question 5.2

Now train only on a subset of the CIFAR10 training dataset. Use the below subset conditions. Keep the full CIFAR10 testing set intact. Repeat the above experiment and tune the hyper-parameters as best as you can.

1. 1 instance per class, a total of 10 train images only.
2. 10 instances per class
3. 100 instances per class
4. 1000 instances per class
5. full train data set

- Plot the (Training accuracy - testing accuracy) vs number of training data used.
- Submit the code *exxxxxxx_q5.2.py* or google colab link.

Question 6: Receptive field - (2.5 Marks)

The receptive field is defined as the region in the input space that a particular CNN's feature is looking at (i.e. be affected by). Compute size of receptive fields with respect to the input image across the layers. Fill in the empty cells in the table 1

Table 1: Fill in receptive field size of a single pixel at the output with respect to the input image.

conv2d(k,s)	conv2d with kernel k x k and stride s x s
maxpool2d(k,s)	maxpool with kernel k x k and stride s x s

S.No	layer1	layer2	layer3	layer4	layer5	receptive field
1	conv2d(3,1)	conv2d(3,1)	conv2d(1,1)	maxpool2d(2,2)	conv2d(2,2)	8 x 8
2	conv2d(3,1)	maxpool2d(2,2)	conv2d(2,2)	maxpool2d(2,2)	conv2d(3,1)	–
3	conv2d(5,1)	maxpool2d(3,1)	conv2d(4,1)	maxpool2d(2,2)	conv2d(3,1)	–
4	conv2d(7,2)	conv2d(5,1)	conv2d(3,2)	conv2d(3,1)	conv2d(1,1)	–
5	conv2d(4,1)	conv2d(3,2)	maxpool2d(2,2)	conv2d(3,1)	maxpool2d(2,2)	–
6	conv2d(7,2)	conv2d(5,2)	maxpool2d(2,2)	conv2d(5,2)	conv2d(7,2)	–
7	conv2d(3,3)	maxpool2d(2,2)	conv2d(5,3)	maxpool2d(3,3)	conv2d(5,1)	–
8	conv2d(4,2)	maxpool2d(2,2)	conv2d(3,1)	conv2d(1,1)	conv2d(2,1)	–
9	conv2d(4,2)	maxpool2d(2,2)	conv2d(3,1)	conv2d(1,1)	conv2d(2,1)	–
10	conv2d(11,2)	conv2d(7,2)	maxpool2d(2,2)	conv2d(5,2)	conv2d(3,1)	–