Computer Vision & Geometry Group
Songyou Peng

ETH zürich

Computer Vision

Exam Preparation – Deep Learning

1 Exam questions from last year

a) Let us consider a **linear layer** with a 3-dimensional input $\mathbf{x} \in \mathbb{R}^3$ and 10-dimensional output $\mathbf{y} \in \mathbb{R}^{10}$. Write the formula relating the output to the input using the weights matrix $\mathbf{W}$ and the bias vector $\mathbf{b}$. What is the size of the weights matrix? What is the dimension of the bias vector?

> The linear equation: $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$.
>
> $\mathbf{W}$ is a 10×3 matrix.
>
> $\mathbf{b}$ is a 10-dimensional vector.

b) Provide the mathematical definition/equation of Sigmoid and ReLU activation functions with respect to the input $\mathbf{x} \in \mathbb{R}$. Plot the rough shape of both these functions between -10 and 10 and give their asymptotic values.

> $\text{Sigmoid}(\mathbf{x}) = \frac{1}{1+e^{-\mathbf{x}}}$
>
> $\text{ReLU}(\mathbf{x}) = \max(0, \mathbf{x})$
>
> Draw their corresponding plot

c) Consider the following 4×4 image: $I = \begin{bmatrix} 1 & 4 & 3 & 2 \\ 7 & 8 & 1 & 2 \\ 1 & 0 & 7 & 3 \\ 2 & 1 & 5 & 1 \end{bmatrix}$. What is the output of an average pooling layer with stride 2, kernel size 2×2, and no padding?

> $\begin{bmatrix} 5 & 2 \\ 1 & 4 \end{bmatrix}$

d) What is the main advantage of convolution layers compared to MLPs?

> **Advantage**: efficiency in terms of parameters

2 Extra Exercise

a) Analytic Gradient Calculation

i) $\text{ReLU}(\mathbf{x}) = \max(0, \mathbf{x})$, $\mathbf{x} \in \mathbb{R}$

$$\frac{\partial \text{ReLU}(\mathbf{x})}{\partial \mathbf{x}} = \begin{cases} 1 & \mathbf{x} > 0 \\ 0 & \mathbf{x} \leq 0 \end{cases}$$

ii) $\text{LeakyReLU}(\mathbf{x}) = \begin{cases} \mathbf{x} & \mathbf{x} > 0 \\ c\mathbf{x} & \mathbf{x} \leq 0 \end{cases}$, where $\mathbf{x} \in \mathbb{R}$ and c is a constant.

$$\frac{\partial \text{LeakyReLU}(\mathbf{x})}{\partial \mathbf{x}} = \begin{cases} 1 & \mathbf{x} > 0 \\ c & \mathbf{x} \leq 0 \end{cases}$$

iii) $\text{Tanh}(\mathbf{x}) = \frac{2}{1+e^{-2\mathbf{x}}} - 1$, $\mathbf{x} \in \mathbb{R}$

$\text{Tanh}(\mathbf{x}) = \frac{2}{1+e^{-2\mathbf{x}}} - 1 = \frac{e^{\mathbf{x}}-e^{-\mathbf{x}}}{e^{\mathbf{x}}+e^{-\mathbf{x}}}$. So,

$$\begin{aligned} \frac{\partial \text{Tanh}(\mathbf{x})}{\partial \mathbf{x}} &= \frac{(e^{\mathbf{x}} + e^{-\mathbf{x}})(e^{\mathbf{x}} + e^{-\mathbf{x}}) - (e^{\mathbf{x}} - e^{-\mathbf{x}})(e^{\mathbf{x}} - e^{-\mathbf{x}})}{(e^{\mathbf{x}} + e^{-\mathbf{x}})^2} \\ &= \frac{(e^{\mathbf{x}} + e^{-\mathbf{x}})^2 - (e^{\mathbf{x}} - e^{-\mathbf{x}})^2}{(e^{\mathbf{x}} + e^{-\mathbf{x}})^2} \\ &= 1 - \frac{(e^{\mathbf{x}} - e^{-\mathbf{x}})^2}{(e^{\mathbf{x}} + e^{-\mathbf{x}})^2} \\ &= 1 - \text{Tanh}^2(\mathbf{x}) \end{aligned}$$

iv) Given the predicted probability $\mathbf{p} \in \mathbb{R}^n$ and ground-truth label $\mathbf{y} \in \mathbb{R}^n$, calculate the gradient of cross entropy loss $H(\mathbf{y}, \mathbf{p}) = -\sum_{i=1}^{n} y_i \log p_i$ wrt. every element of $\mathbf{p}$, i.e. $\frac{\partial H}{\partial p_i}(\mathbf{y}, \mathbf{p})$.

$$\frac{\partial H}{\partial p_i}(\mathbf{y}, \mathbf{p}) = \frac{-\sum_{i=1}^{n} y_i \log p_i}{\partial p_i} = -\frac{y_i}{p_i}$$

b) Prove the equation: $\text{Softmax}(\mathbf{x}) = \text{Softmax}(\mathbf{x} + c)$, where c is a constant.

$$\text{Softmax}(\mathbf{x} + c) = \frac{e^{\mathbf{x}+c}}{\sum_{i=1}^{n} e^{x_i+c}} = \frac{e^{\mathbf{x}} \cdot e^{c}}{e^{c} \cdot \sum_{i=1}^{n} e^{x_i}} = \frac{e^{\mathbf{x}}}{\sum_{i=1}^{n} e^{x_i}} = \text{Softmax}(\mathbf{x})$$

c) CNN Arithmetics. We aim to calculate several properties for different configurations of convolutional and max-pooling layers. For the layers, we use the same notation as PyTorch - input channels C_{in}, output channels C_{out} , kernel size K, stride S, and padding P. We use square kernels and equal stride/padding; hence, we only need to specify scalars.

Consider the following layers as examples:

- $\text{Conv}(C_{in} = 3, C_{\text{out}} = 16, K = 5, S = 2, P = 1)$: a convolutional layer with 3 input channels and 16 output channels, kernel size 5, stride 2, and padding 1.
- $\text{MaxPool}(K = 2, S = 1)$: max-pooling with a (square) kernel size of 2 with stride 1

i) We pass an image of size $C = 3, W = 320, H = 320$ through the network below. Fill in values for the ?'s so that the architecture is valid. Then report the tensor size (C_{out}, H, W) after each layer when the image is passed through the network.

Layer
Conv(?, ?, 3, 1, 1)
ReLU
MaxPool(2, 2)
Conv(64, ?, 3, 1, 1)
ReLU
MaxPool(2, 2)
Conv(128, 256, 3, 1, 1)
ReLU
MaxPool(2, 2)
Conv(?, 512, 3, 1, 1)
ReLU
MaxPool(2, 2)

Layer	Output Shape
Conv(3, 64, 3, 1, 1)	(64, 320, 320)
ReLU	(64, 320, 320)
MaxPool(2, 2)	(64, 160, 160)
Conv(64, 128, 3, 1, 1)	(128, 160, 160)
ReLU	(128, 160, 160)
MaxPool(2, 2)	(128, 80, 80)
Conv(128, 256, 3, 1, 1)	(256, 80, 80)
ReLU	(256, 80, 80)
MaxPool(2, 2)	(256, 40, 40)
Conv(256, 512, 3, 1, 1)	(512, 40, 40)
ReLU	(512, 40, 40)
MaxPool(2, 2)	(512, 20, 20)

ii) The network architecture above is the (simplified) convolutional part of a network called VGG16 by (Simonyan et al., 2014). As mentioned in the lecture, this stack of convolutional and max-pooling layers is often followed by a few fully-connected layers. The first fully-connected layer of a VGG16 is a FC($C_{in} = 25088, C_{out} = 4096$). Calculate the number of trainable parameters for the first and the second convolutional layers and the first fully-connected layer.

The formulas to calculate the number of parameters:
Fully connected layer: $\#params = C_{out} \times (C_{in} + 1)$
Convolutional layer: $\#params = C_{out} \times (K \times K \times C_{in} + 1)$

A Conv$(3, 64, 3, 1, 1)$: $\#params = 1,792$
B Conv$(64, 128, 3, 1, 1)$: $\#params = 73,856$
C FC$(25088, 4096)$: $\#params = 102,764,544$

The advantage of weight sharing becomes apparent.

d) Multiple Choice.

Determine if the following statements are true or false by marking the correct answer with a tick.

Statement	True	False
AlexNet was the first neural network to win the ImageNet challenge (ILSVRC).	✓	
Assume $p_{model}(y|\mathbf{x},\mathbf{w}) = \mathcal{N}(y|\mathbf{w}^\top\mathbf{x},\sigma)$. Maximum likelihood estimation of the parameters $\mathbf{w}$ is a least squares problem and can be solved in closed form.	✓	
Logistic regression is used to solve regression problems.		✓
The backpropagation algorithm computes the output of all compute nodes in the forward pass and all model parameter updates in the backward pass.	✓	

e) Backpropagation. Consider an 1-layer neural network $\mathbf{y} = g(\mathbf{Ax})$ with input $\mathbf{x}$, output $\mathbf{y}$, network weight $\mathbf{A}$ and output function g. Let's first assume the input and the network weights are

$$\mathbf{A} = \begin{pmatrix} 3.0 & 2.0 \end{pmatrix} \in \mathbb{R}^{1\times 2} \qquad \mathbf{x} = \begin{pmatrix} 2.0 & 1.0 & -1.0 \\ 4.0 & -2.0 & 0.0 \end{pmatrix} \in \mathbb{R}^{2\times 3}$$

where $\mathbf{x}$ are 2D points with a **minibatch** size of 3.

i) Assume the function g is linear, i.e. $g(\mathbf{Ax}) = \mathbf{Ax}$ without the bias term. Given the target output $\mathbf{t}$ and the loss function $\mathcal{L}$, perform weight update. Assume a learning rate of 1. Please provide the loss *before* and *after* the weight update.

$$\mathbf{t} = \begin{pmatrix} 15 & 3 & 1 \end{pmatrix} \in \mathbb{R}^{1\times 3} \qquad \mathcal{L} = \frac{1}{2}\sum_j (y_j - t_j)^2$$

Hint: First calculate forward pass and obtain the loss. Next, you can derive gradients of loss wrt. to every element of network weight $\frac{\partial \mathcal{L}}{\partial a_i}$. Make sure to include all steps of your derivation. Once the network weight is updated, calculate forward pass again to acquire the loss.

Set $\mathbf{A} = \begin{pmatrix} a_1 & a_2 \end{pmatrix}$, $\mathbf{y} = \mathbf{Ax} = \begin{pmatrix} 2a_1 + 4a_2 & a_1 - 2a_2 & -a_1 \end{pmatrix}$. Therefore, the loss is:

$$\mathcal{L} = \frac{1}{2}(2a_1 + 4a_2 - 15)^2 + \frac{1}{2}(a_1 - 2a_2 - 3)^2 + \frac{1}{2}(-a_1 - 1)^2$$

At time step t, the loss is:

$$\mathcal{L}^t = \frac{1}{2}(1 + 16 + 16) = 16.5$$

The derivative w.r.t. $\mathbf{A}$:

$$\frac{\partial \mathcal{L}^t}{\partial a_1} = 2\cdot(2a_1 + 4a_2 - 15) + (a_1 - 2a_2 - 3) - (-a_1 - 1) = -2$$
$$\frac{\partial \mathcal{L}^t}{\partial a_2} = 4\cdot(2a_1 + 4a_2 - 15) - 2\cdot(a_1 - 2a_2 - 3) = 4$$

Using the gradient descent algorithm, we can obtain:

$$a_1^{t+1} = a_1^t - \eta \cdot \frac{1}{3}\frac{\partial \mathcal{L}^t}{\partial a_1} = 3 + \frac{2}{3} = 3.667$$
$$a_2^{t+1} = a_2^t - \eta \cdot \frac{1}{3}\frac{\partial \mathcal{L}^t}{\partial a_2} = 2 - \frac{4}{3} = 0.667$$

The update $\mathbf{A}^{t+1} = \begin{pmatrix} 3.667 & 0.667 \end{pmatrix}$ and new $\mathcal{L}^{t+1}$ becomes:

$$\mathcal{L}^{t+1} = 23.611$$

Alternatively, you can also consider the chain rule to calculate the gradient.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{A}} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}}\frac{\partial \mathbf{y}}{\partial \mathbf{A}}$$

where

$$\frac{\partial \mathbf{y}}{\partial \mathbf{A}} = \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial y_1} & \frac{\partial \mathcal{L}}{\partial y_2} & \frac{\partial \mathcal{L}}{\partial y_3} \end{pmatrix} = \begin{pmatrix} y_1 - t_1 & y_2 - t_2 & y_3 - t_3 \end{pmatrix} = \begin{pmatrix} -1 & -4 & -4 \end{pmatrix}$$
$$\frac{\partial \mathcal{L}}{\partial \mathbf{y}} = \begin{pmatrix} \frac{\partial y_1}{a_1} & \frac{\partial y_1}{a_2} \\ \frac{\partial y_2}{a_1} & \frac{\partial y_2}{a_2} \\ \frac{\partial y_3}{a_1} & \frac{\partial y_3}{a_2} \end{pmatrix} = \begin{pmatrix} 2 & 4 \\ 1 & -2 \\ -1 & 0 \end{pmatrix}$$

so

$$\frac{\partial \mathcal{L}}{\partial \mathbf{A}} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}}\frac{\partial \mathbf{y}}{\partial \mathbf{A}} = \begin{pmatrix} -1 & -4 & -4 \end{pmatrix} \begin{pmatrix} 2 & 4 \\ 1 & -2 \\ -1 & 0 \end{pmatrix} = \begin{pmatrix} -2 & 4 \end{pmatrix}$$

The rest is the same as before.

ii) For the loss that you calculated before and after the weight update in **i)**, does the loss decrease, or in other word, is the new prediction closer to the target $\mathbf{t}$? If not, what could be the reason?

The loss increases because the learning rate is too high.