ETH zürich

Computer Vision & Geometry Group

263-5902-00L
Computer Vision HS 2023
Exam Preparation — Deep Learning

Zuoyue Li

Oct 2023

Outline

1. CNN Arithmetic
2. Numerical Calculations
3. Derivation

Outline

1. CNN Arithmetic
2. Numerical Calculations
3. Derivation

Array shape

(**Exam HS 2022**) Below is an architecture with standard convolution and max-pooling layers. What are the output shapes (channel-first, *i.e.*, channels followed by height and width) after each layer? We have provided the output of an intermediate layer, please fill in your answers below for the other layers. You do not have to consider the dimension of batch size. (**5 pt.**)
Notation: input channels C_{in}, output channels C_{out}, kernel size K, stride S, and padding P.

Layer	Output shape
Conv2d($C_{in} = 3, C_{out} = 64, K = 7, S = 2, P = 3$)	*Answer:* $(64, 112, 112)$
ReLU()	*Answer:* $(64, 112, 112)$
MaxPool2d($K = 2, S = 2, P = 0$)	$(64, 56, 56)$
Conv2d($C_{in} = 64, C_{out} = 192, K = 3, S = 1, P = 1$)	*Answer:* $(192, 56, 56)$
ReLU()	*Answer:* $(192, 56, 56)$
MaxPool2d($K = 2, S = 2, P = 0$)	*Answer:* $(192, 28, 28)$

Additional question: what is the input shape? *Answer:* $(3, 224, 224)$
Reference: https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

Number of (trainable) parameters

Formulas

Fully connected layer (w/o bias): $N = C_{out} \times C_{in}$
Fully connected layer (w/ bias): $N = C_{out} \times (C_{in} + 1)$
2D convolutional layer (w/o bias): $N = C_{out} \times K \times K \times C_{in}$
2D convolutional layer (w/ bias): $N = C_{out} \times (K \times K \times C_{in} + 1)$

Example: Conv2d($C_{in} = 3, C_{out} = 64, K = 3, S = 1, P = 1$) with bias? *Answer:* $1'792$
Use PyTorch we can get:

```
>>> import torch
>>> conv = torch.nn.Conv2d(3, 64, 3, 1, 1)
>>> for name, param in conv.named_parameters():
...     print(name, param.shape)
...
weight torch.Size([64, 3, 3, 3])
bias torch.Size([64])
```

Additional question: Conv**3**d($C_{in} = 3, C_{out} = 64, K = 3, S = 1, P = 1$) with bias? *Answer:* $5'248$

Outline

1. CNN Arithmetic

2. Numerical Calculations

3. Derivation

Output values of an operation

(**Exam HS 2021**) Consider the following 4×4 single-channel image, what is the output of a 2D average pooling layer AvgPool2d($K = 2, S = 2, P = 0$) performed on the below image? (**3 pt.**)

$$\begin{bmatrix} 1 & 4 & 3 & 2 \\ 7 & 8 & 1 & 2 \\ 1 & 0 & 7 & 3 \\ 2 & 1 & 5 & 1 \end{bmatrix}$$

Answer:

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

Additional question: how about MaxPool2d($K = 2, \mathbf{S} = \mathbf{1}, P = 0$)?

Answer:

$$\begin{bmatrix} 8 & 8 & 3 \\ 8 & 8 & 7 \\ 2 & 7 & 7 \end{bmatrix}$$

Backpropagation

(**Exam HS 2022**) Consider a single-layer network with input $\mathbf{x} \in \mathbb{R}^2$, and output $\mathbf{y} \in \mathbb{R}^2$ without any activation or bias, *i.e.*, $\mathbf{y} = \mathbf{A}\mathbf{x}$, with the layer parameters $\mathbf{A} \in \mathbb{R}^{2\times 2}$ initialized with the following. One GT data pair with the target $\mathbf{t} \in \mathbb{R}^2$ is given below and the loss L is also defined as follows.

$$\mathbf{A} = \begin{bmatrix} 3 & 2 \\ 2 & 1 \end{bmatrix}, \mathbf{x} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}, \mathbf{t} = \begin{bmatrix} 15 \\ 6 \end{bmatrix}, L = \frac{1}{2}\sum_{i=1}^{2}(y_i - t_i)^2$$

(1) Perform a forward pass and calculate the loss. (**4 pt.**)

Forward pass:

$$\mathbf{y} = \mathbf{A}\mathbf{x} = \begin{bmatrix} 3 & 2 \\ 2 & 1 \end{bmatrix}\begin{bmatrix} 2 \\ 4 \end{bmatrix} = \begin{bmatrix} 14 \\ 8 \end{bmatrix}$$

Compute loss:

$$L = \frac{1}{2}[(14-15)^2 + (8-6)^2] = \frac{5}{2}$$

Backpropagation

(2) Perform one gradient descent step with a learning rate of $\eta = 1$. Make sure to include all steps of your derivation. **(8 pt.)**

Rewrite loss function:

$$L = \frac{1}{2}[(y_1 - t_1)^2 + (y_2 - t_2)^2] = \frac{1}{2}[(a_{11}x_1 + a_{12}x_2 - t_1)^2 + (a_{21}x_1 + a_{21}x_2 - t_2)^2]$$

Compute gradient:

$$\frac{\partial L}{\partial \mathbf{A}} = \begin{bmatrix} \frac{\partial L}{\partial a_{11}} & \frac{\partial L}{\partial a_{12}} \\ \frac{\partial L}{\partial a_{21}} & \frac{\partial L}{\partial a_{22}} \end{bmatrix} = \begin{bmatrix} (y_1 - t_1)x_1 & (y_1 - t_1)x_2 \\ (y_2 - t_2)x_1 & (y_2 - t_2)x_2 \end{bmatrix} = (\mathbf{y} - \mathbf{t})\mathbf{x}^\top = \begin{bmatrix} -2 & -4 \\ 4 & 8 \end{bmatrix}$$

Perform gradient descent:

$$\mathbf{A}' = \mathbf{A} - \eta \frac{\partial L}{\partial \mathbf{A}} = \begin{bmatrix} 3 & 2 \\ 2 & 1 \end{bmatrix} - 1 \times \begin{bmatrix} -2 & -4 \\ 4 & 8 \end{bmatrix} = \begin{bmatrix} 5 & 6 \\ -2 & -7 \end{bmatrix}$$

Backpropagation

Use PyTorch to verify the answers:

```
>>> A = torch.Tensor([[3, 2],[2, 1]])
>>> A.requires_grad = True
>>> x = torch.Tensor([2, 4])
>>> t = torch.Tensor([15, 6])
>>>
>>> # Build optimizer & forward pass
>>> optimizer = torch.optim.SGD([A], lr=1)
>>> y = A.matmul(x)
>>> L = torch.sum((y - t) ** 2) / 2
>>>
>>> # See loss and backward
>>> print("Loss", L.item())
Loss 2.5
>>> L.backward()
>>>
```

```
>>> # A gradient
>>> print(A.grad)
tensor([[-2., -4.],
        [ 4.,  8.]])
>>>
>>> # A before gradient descent
>>> print(A)
tensor([[3., 2.],
        [2., 1.]], requires_grad=True)
>>>
>>> optimizer.step()
>>> # A after gradient descent
>>> print(A)
tensor([[ 5.,  6.],
        [-2., -7.]], requires_grad=True)
```

Outline

Softmax

The softmax function $\sigma(\cdot)$ takes as input a vector $\mathbf{x}$ of n real numbers and normalizes it into a probability distribution consisting of n probabilities proportional to the exponentials of the input numbers.[1] $\sigma(\cdot) : \mathbb{R}^n \mapsto (0,1)^n$.

Definition

$$\sigma(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}, i = 1, \cdots, n$$

The softmax function is commonly used in the multi-class classification task for obtaining the probability belonging to each class. Prove the equation below, where $\mathbf{x} \in \mathbb{R}^n$ is a vector and c is an arbitrary constant.

$$\sigma(\mathbf{x} + c) = \sigma(\mathbf{x})$$

Proof:

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

[1] https://en.wikipedia.org/wiki/Softmax_function

Backpropagation of a linear layer

Consider a linear layer with input $\mathbf{x} \in \mathbb{R}^n$, and output $\mathbf{y} \in \mathbb{R}^m$ without any activation or bias, *i.e.*, $\mathbf{y} = \mathbf{A}\mathbf{x}$, with the layer parameters $\mathbf{A} \in \mathbb{R}^{m \times n}$. Suppose there is a defined loss L and the gradients are back-propagated to $\mathbf{y}$ from further layers, *i.e.*, $\frac{\partial L}{\partial \mathbf{y}}$ is known and given.

Derive the gradients of the input $\frac{\partial L}{\partial \mathbf{x}}$ (will be passed to the downstream layers) and the layer weight $\frac{\partial L}{\partial \mathbf{A}}$ (for gradient descent) in terms of $\mathbf{x}$, $\mathbf{y}$, $\frac{\partial L}{\partial \mathbf{y}}$ and $\mathbf{A}$ (not necessarily all used).

Gradients of the input:

$$\frac{\partial L}{\partial \mathbf{x}} = {}^{2}\sum_{i=1}^{m} \frac{\partial L}{\partial y_i}\frac{\partial y_i}{\partial \mathbf{x}} = \sum_{i=1}^{m} \frac{\partial L}{\partial y_i}\frac{\partial (\mathbf{A}_i \cdot \mathbf{x})}{\partial \mathbf{x}} = \sum_{i=1}^{m} \frac{\partial L}{\partial y_i}\mathbf{A}_i^\top = \mathbf{A}^\top \frac{\partial L}{\partial \mathbf{y}}$$

[2] Total derivative.

Backpropagation of a linear layer

Gradients of the layer weight:

$$\frac{\partial L}{\partial \mathbf{A}} = \sum_{i=1}^{m} \frac{\partial L}{\partial y_i} \frac{\partial y_i}{\partial \mathbf{A}} = \sum_{i=1}^{m} \frac{\partial L}{\partial y_i} \frac{\partial (\mathbf{A}_i \cdot \mathbf{x})}{\partial \mathbf{A}} = \sum_{i=1}^{m} \frac{\partial L}{\partial y_i} \frac{\partial (a_{i1}x_1 + \cdots + a_{ij}x_j + \cdots + a_{in}x_n)}{\partial \mathbf{A}} \tag{1}$$

$$= \sum_{i=1}^{m} \frac{\partial L}{\partial y_i} \begin{bmatrix} 0 & \cdots & 0 & \cdots & 0 \\ & & \vdots & & \\ & & \mathbf{x}^\top & (i\text{-th row}) & \\ & & \vdots & & \\ 0 & \cdots & 0 & \cdots & 0 \end{bmatrix} = \begin{bmatrix} \frac{\partial L}{\partial y_1}\mathbf{x}^\top \\ \vdots \\ \frac{\partial L}{\partial y_i}\mathbf{x}^\top \\ \vdots \\ \frac{\partial L}{\partial y_m}\mathbf{x}^\top \end{bmatrix} \tag{2}$$

$$= \frac{\partial L}{\partial \mathbf{y}}\mathbf{x}^\top = \frac{\partial L}{\partial \mathbf{y}} \otimes \mathbf{x} \text{ (outer product)} \tag{3}$$

Backpropagation of a linear layer

Recap the loss function defined on the page 7:

$$L = \frac{1}{2}\sum_i (y_i - t_i)^2 = \frac{1}{2}||\mathbf{y} - \mathbf{t}||_2^2 = \frac{1}{2}(\mathbf{y} - \mathbf{t})^\top (\mathbf{y} - \mathbf{t})$$

Thus the gradient $\frac{\partial L}{\partial \mathbf{y}}$ in this question is:

$$\frac{\partial L}{\partial \mathbf{y}} = \mathbf{y} - \mathbf{t}$$

Use the conclusion we got from the last page, we have:

$$\frac{\partial L}{\partial \mathbf{A}} = \frac{\partial L}{\partial \mathbf{y}}\mathbf{x}^\top = (\mathbf{y} - \mathbf{t})\mathbf{x}^\top$$

which is the same as what we have seen on the page 8.

ETH zürich

Zuoyue Li
PhD student
Computer Vision & Geometry group
li.zuoyue@inf.ethz.ch

ETH Zürich
CAB G 85.2
Universitätstrasse 6
8006 Zürich