Mathematical Foundations of Deep Neural Networks, M1407.001200
E. Ryu
Spring 2024

Homework 4
Due 5pm, Monday, April 1, 2024

Problem 1: *Finite difference with convolution.* Given an image $X \in \mathbb{R}^{m \times n}$, we wish to compute the x- and y-direction derivatives. This is commonly done in computer vision for the purpose of "edge detection". Specifically, define $Y \in \mathbb{R}^{2 \times m \times n}$ with

$$Y_{1,i,j} = X_{i+1,j} - X_{i,j}$$

and

$$Y_{2,i,j} = X_{i,j+1} - X_{i,j}$$

for $i = 1, \dots, m$ and $j = 1, \dots, n$. We define $X_{m+1,:} = 0$ and $X_{:,n+1} = 0$, i.e., we define the out-of-bounds elements to have 0 value. How can we represent the mapping $X \mapsto Y$ as a convolution with a 3×3 filter and zero padding of 1? More specifically, what should the filter $w \in \mathbb{R}^{2 \times 3 \times 3}$ be?

Solution. Let $w = [w_1, w_2] \in \mathbb{R}^{2 \times 3 \times 3}$, where $w_1, w_2 \in \mathbb{R}^{3 \times 3}$. Then

$$\begin{aligned} Y_{1,i,j} &= [w_1 \circledast X]_{i,j} \\ &= w_{1,1,1} X_{i-1,j-1} + w_{1,1,2} X_{i-1,j} + w_{1,1,3} X_{i-1,j+1} \\ &\quad + w_{1,2,1} X_{i,j-1} + w_{1,2,2} X_{i,j} + w_{1,2,3} X_{i,j+1} \\ &\quad + w_{1,3,1} X_{i+1,j-1} + w_{1,3,2} X_{i+1,j} + w_{1,3,3} X_{i+1,j+1} \\ &= X_{i+1,j} - X_{i,j}. \end{aligned}$$

So we have

$$\begin{gathered} w_{1,1,1} = w_{1,1,2} = w_{1,1,3} = w_{1,2,1} = w_{1,2,3} = w_{1,3,1} = w_{1,3,3} = 0, \\ w_{1,2,2} = -1, \\ w_{1,3,2} = 1. \end{gathered}$$

By characterizing w_2 likewise, we get

$$w = \left[\begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \right].$$

■

Problem 2: *Average pooling as convolution.* Given an input tensor $X \in \mathbb{R}^{C \times m \times n}$, the AvgPool2d operation with kernel size k outputs $Y \in \mathbb{R}^{C \times (m/k) \times (n/k)}$ with

$$Y_{c,i,j} = \frac{1}{k^2} \sum_{a=1}^{k} \sum_{b=1}^{k} X_{c,k(i-1)+a,k(j-1)+b}$$

for $i = 1, \dots, (m/k)$ and $j = 1, \dots, (n/k)$. For the sake of simplicity, assume m and n are divisible by k. How can we represent the AvgPool2d operation $X \mapsto Y$ as a convolution?

Solution. Let $w \in \mathbb{R}^{C\times k\times k}$ be a convolution filter representation of the AvgPool2d with no padding and stride=k. Denote $w = [w_1, \cdots, w_C]$ where $w_1, \cdots, w_C \in \mathbb{R}^{k\times k}$. Then

$$\begin{aligned} Y_{c,i,j} =& \sum_{a=1}^{k}\sum_{b=1}^{k} w_{c,a,b} * X_{c,k(i-1)+a,k(j-1)+b} \\ =& \frac{1}{k^2}\sum_{a=1}^{k}\sum_{b=1}^{k} X_{c,k(i-1)+a,k(j-1)+b} \end{aligned}$$

Similar logic of **Problem 1**, Since all $X_{\cdot,\cdot,\cdot}$'s are arbitrary, all $w_{c,a,b} = \dfrac{1}{k^2}$. Hence $w = \frac{1}{k^2}\mathbf{1}$. (All elements of $\mathbf{1}$ are ones. And $\mathbf{1} \in \mathbb{R}^{C\times k\times k}$) ■

Problem 3: *RGB to greyscale mapping with* 1×1 *convolution.* The standard conversion from an RGB pixel to greyscale value (the luminance "Y value") is

$$Y = 0.299R + 0.587G + 0.114B.$$

This conversion produces visually superior results when judged by human test subjects, compared to a uniform averaging with 1/3 weights. Specifically, given $X \in \mathbb{R}^{3\times m\times n}$, define $Y \in \mathbb{R}^{m\times n}$ with

$$Y_{i,j} = 0.299X_{1,i,j} + 0.587X_{2,i,j} + 0.114X_{3,i,j}$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, n$. How can we represent the mapping $X \mapsto Y$ as a convolution with a 1×1 filter? More specifically, what should the filter $w \in \mathbb{R}^{3\times 1\times 1}$ be?

Remark. Of course, the answer is

$$w_{1,1,1} = 0.299, \qquad w_{2,1,1} = 0.587, \qquad w_{3,1,1} = 0.114.$$

The intention of this problem is to have you think about what a 1×1 convolution is.

Remark. Using 1×1 convolutions to linearly combine channel information for each spatial location is a common technique that we will revisit when constructing "bottleneck" layers.

Solution. Since

$$\begin{aligned} Y_{i,j} = [w * X]_{i.j} =& w_{1,1,1} \times X_{1,i,j} + w_{2,1,1} \times X_{2,i,j} + w_{3,1,1} \times X_{3,i,j} \\ =& 0.299X_{1,i,j} + 0.587X_{2,i,j} + 0.114X_{3,i,j} \end{aligned}$$

Since $X_{i,j}$'s are arbitrary, we have

$$w_{1,1,1} = 0.299, \qquad w_{2,1,1} = 0.587, \qquad w_{3,1,1} = 0.114.$$

■

Problem 4: Let $\sigma\colon \mathbb{R} \to \mathbb{R}$ a nondecreasing activation function, and let $\rho\colon \mathbb{R}^{m\times n} \to \mathbb{R}^{k\times \ell}$ be a max pool operation. Show that

$$\sigma(\rho(X)) = \rho(\sigma(X))$$

for all $X \in \mathbb{R}^{m\times n}$, i.e., show that σ and ρ commute.

Solution. Since each component of max pool is dependent on finite set, it is sufficient to prove that for finite set $S = \{x_1, \cdots, x_n\}$, $\sigma(\max(S)) = \max(\sigma(S))$. Without loss of generality, we can assume $x_1 \leq x_2 \leq \cdots \leq x_n$. Then, $\rho(S) = \{\sigma(x_1), \sigma(x_2), \cdots, \sigma(x_n)\}$ and since σ is a nondecreasing function, $\sigma(x_1) \leq \cdots \leq \sigma(x_n)$.

In sum,

$$\sigma(\max(S)) = \sigma(x_n) = \max(\sigma(S)),$$

which indicates σ and ρ commute. ■

Problem 5: *Non-CE loss function.* Consider the setup of predicting labels of the 4 and 9 classes of the MNIST dataset. As we did for logistic regression, choose the model

$$f_{a,b}(x) = \begin{bmatrix} 1/(1+e^{a^\intercal x+b}) \\ 1/(1+e^{-(a^\intercal x+b)}) \end{bmatrix},$$

but minimize the sum-of-squares loss instead of the KL-divergence. In other words, instead of solving

$$\underset{a\in\mathbb{R}^p,\, b\in\mathbb{R}}{\text{minimize}} \quad \sum_{i=1}^{N} D_{\mathrm{KL}}(\mathcal{P}(Y_i) \| f_{a,b}(X_i))$$

solve

$$\underset{a\in\mathbb{R}^p,\, b\in\mathbb{R}}{\text{minimize}} \quad \frac{1}{N}\sum_{i=1}^{N} \|\mathcal{P}(Y_i) - f_{a,b}(X_i)\|^2,$$

where

$$\mathcal{P}(y) = \begin{cases} \begin{bmatrix} 1 & 0 \end{bmatrix}^\intercal & \text{if } y = -1 \\ \begin{bmatrix} 0 & 1 \end{bmatrix}^\intercal & \text{if } y = 1. \end{cases}$$

How does the performance compare to minimizing the KL divergence?

Hint. Let $\sigma(z) = 1/(1+e^{-z})$ be the sigmoid function, which is implemented as `torch.sigmoid` in PyTorch. Consider

$$\ell(z, y) = \frac{1}{2}(1-y)\left((1-\sigma(-z))^2 + (\sigma(z))^2\right) + \frac{1}{2}(1+y)\left((\sigma(-z))^2 + (1-\sigma(z))^2\right).$$

Remark. When defining loss functions in PyTorch, you want to avoid using if-statements until you understand backprop sufficiently well.

Remark. Throughout the deep learning literature, the cross-entropy (CE) loss is most commonly used for image classification. However, there does not seem to be sufficient evidence, experimental nor theoretical, to justify this default choice, and some recent experimental investigations indicate that different losses can outperform the CE loss [2, 3].

Solution. See `mseloss_sol.py`. In the example of classifying 4 and 9 in MNIST dataset, using sum-of-squares loss seems to lead to better results. ■

Problem 6: *Backprop for MLP.* In this problem, we take a closer look at the gradient computation of multi-layer perceptrons. Let $\sigma\colon \mathbb{R} \to \mathbb{R}$ be a differentiable activation function and consider the following multi-layer perceptron

$$\begin{aligned} y_L &= A_L y_{L-1} + b_L \\ y_{L-1} &= \sigma(A_{L-1} y_{L-2} + b_{L-1}) \\ &\vdots \\ y_2 &= \sigma(A_2 y_1 + b_2) \\ y_1 &= \sigma(A_1 x + b_1), \end{aligned}$$

where $x \in \mathbb{R}^{n_0}$, $A_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$, $b_\ell \in \mathbb{R}^{n_\ell}$, and $n_L = 1$. (To clarify, σ is applied element-wise.) For notational convenience, define $y_0 = x$.

(a) Show

$$\begin{aligned} \frac{\partial y_L}{\partial b_L} &= 1, \qquad \frac{\partial y_L}{\partial y_{L-1}} = A_L, \\ \frac{\partial y_\ell}{\partial b_\ell} &= \operatorname{diag}\left(\sigma'(A_\ell y_{\ell-1} + b_\ell)\right), \qquad \text{for } \ell = 1, \dots, L-1 \\ \frac{\partial y_\ell}{\partial y_{\ell-1}} &= \operatorname{diag}\left(\sigma'(A_\ell y_{\ell-1} + b_\ell)\right) A_\ell, \qquad \text{for } \ell = 2, \dots, L-1, \end{aligned}$$

where $\frac{\partial y_\ell}{\partial b_\ell} \in \mathbb{R}^{n_\ell \times n_\ell}$ and $\frac{\partial y_\ell}{\partial y_{\ell-1}} \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ are Jacobian matrices. (For any $v \in \mathbb{R}^k$, we define $\operatorname{diag}(v)$ to be the $k \times k$ diagonal matrix with $v_1, \dots, v_k$ as its diagonal entries.)

(b) Since y_ℓ is a vector and A_ℓ is a matrix, writing $\frac{\partial y_\ell}{\partial A_\ell}$ would not make sense. However, $y_L \in \mathbb{R}$ is a scalar, so we define $\frac{\partial y_L}{\partial A_\ell} \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ with

$$\left(\frac{\partial y_L}{\partial A_\ell}\right)_{ij} = \frac{\partial y_L}{\partial (A_\ell)_{ij}}$$

for $i = 1, \dots, n_\ell$ and $j = 1, \dots, n_{\ell-1}$. Show

$$\begin{aligned} \frac{\partial y_L}{\partial A_L} &= y_{L-1}^\intercal \\ \frac{\partial y_L}{\partial A_\ell} &= \operatorname{diag}\left(\sigma'(A_\ell y_{\ell-1} + b_\ell)\right) \left(\frac{\partial y_L}{\partial y_\ell}\right)^\intercal y_{\ell-1}^\intercal, \qquad \text{for } \ell = 1 \dots, L-1. \end{aligned}$$

Hint. For part (a), first compute

$$\left(\frac{\partial y_\ell}{\partial b_\ell}\right)_{ij} = \frac{\partial (y_\ell)_i}{\partial (b_\ell)_j}, \qquad \left(\frac{\partial y_\ell}{\partial y_{\ell-1}}\right)_{ij} = \frac{\partial (y_\ell)_i}{\partial (y_{\ell-1})_j}.$$

For part (b), use the chain rule to first compute

$$\frac{\partial y_L}{\partial (A_\ell)_{ij}} = \frac{\partial y_L}{\partial y_\ell} \frac{\partial y_\ell}{\partial (A_\ell)_{ij}}.$$

To clarify, $\frac{\partial y_L}{\partial y_\ell} \in \mathbb{R}^{1 \times n_\ell}$ and $\frac{\partial y_\ell}{\partial (A_\ell)_{ij}} \in \mathbb{R}^{n_\ell \times 1}$. Once the derivatives have been computed for each i and j, then find a vectorized (or matricized) expression of the result.

Remark. Note that

$$\frac{\partial y_L}{\partial y_\ell} = \frac{\partial y_L}{\partial y_{L-1}} \frac{\partial y_{L-1}}{\partial y_{L-2}} \cdots \frac{\partial y_{\ell+1}}{\partial y_\ell}, \qquad \text{for } \ell = 1 \ldots, L.$$

(the RHS is a product of matrices) is the chain rule of vector calculus.

Solution.

(a) First, the case $\ell = L$ is trivial.
Next, we have

$$y_\ell = \sigma(A_\ell y_{\ell-1} + b_l).$$

Note that ith component of y_ℓ is

$$(y_\ell)_i = \sigma((A_\ell y_{\ell-1} + b_l)_i)$$

since σ is applied element-wise. Thus

$$\left(\frac{\partial y_\ell}{\partial b_\ell}\right)_{ij} = \begin{cases} \sigma'((A_\ell y_{\ell-1} + b_\ell)_i) \cdot 1 & \text{if } j = i \\ 0 & \text{otherwise} \end{cases}$$

which corresponds to

$$\frac{\partial y_\ell}{\partial b_\ell} = \operatorname{diag}\left(\sigma'(A_\ell y_{\ell-1} + b_\ell)\right).$$

Similarly,

$$\left(\frac{\partial y_\ell}{\partial y_{\ell-1}}\right)_{ij} = \sigma'((A_\ell y_{\ell-1} + b_\ell)_i) \cdot (A_\ell)_{ij}$$

which corresponds to

$$\frac{\partial y_\ell}{\partial y_{\ell-1}} = \operatorname{diag}\left(\sigma'(A_\ell y_{\ell-1} + b_\ell)\right) A_\ell.$$

(b) First, we are given

$$y_L = A_L y_{L-1} + b_L = \Sigma_j (A_L)_{1j} (y_{L-1})_j + b_L.$$

Following the given definition of $\frac{\partial y_L}{\partial A_L} \in \mathbb{R}^{1 \times n_{L-1}}$, we have

$$\frac{\partial y_L}{\partial (A_L)_{1j}} = (y_{L-1})_j.$$

So the desired result follows.

Next, we follow the chain rule for the computation of the Jacobian matrix,

$$\frac{\partial y_L}{\partial A_\ell} = \frac{\partial y_L}{\partial y_\ell} \frac{\partial y_\ell}{\partial A_\ell}.$$

Here we have

$$\begin{aligned} y_\ell &= \sigma(A_\ell y_{\ell-1} + b_\ell) \\ &= \begin{bmatrix} \vdots \\ \sigma\left((A_\ell y_{\ell-1} + b_\ell)_i\right) \\ \vdots \end{bmatrix} \\ &= \begin{bmatrix} \vdots \\ \sigma\left(\Sigma_j (A_\ell)_{ij} (y_{\ell-1})_j + (b_\ell)_i\right) \\ \vdots \end{bmatrix}. \end{aligned}$$

Like the problem (a), $\frac{\partial y_\ell}{\partial A_\ell}$ is

$$\frac{\partial y_\ell}{\partial (A_\ell)_{ij}} = \begin{bmatrix} 0 \\ \vdots \\ \sigma'((A_\ell y_{\ell-1} + b_\ell)_i) \cdot (y_{\ell-1})_j \\ \vdots \\ 0 \end{bmatrix}$$

where the shown term is entry of the ith row of the column vector, $1 \leq i \leq n_\ell$, $1 \leq j \leq n_{\ell-1}$.
Back to $\frac{\partial y_L}{\partial A_\ell}$, the chain rule now gives

$$\left(\frac{\partial y_L}{\partial A_\ell}\right)_{ij} = \frac{\partial y_L}{\partial (A_\ell)_{ij}} = \left(\frac{\partial y_L}{\partial y_\ell}\right)_i \sigma'((A_\ell y_{\ell-1} + b_\ell)_i) \cdot (y_{\ell-1})_j$$

This is exactly

$$\frac{\partial y_L}{\partial A_\ell} = \operatorname{diag}\left(\sigma'(A_\ell y_{\ell-1} + b_\ell)\right) \left(\frac{\partial y_L}{\partial y_\ell}\right)^\intercal y_{\ell-1}^\intercal, \qquad \text{for } \ell = 1 \ldots, L.$$

■

Problem 7: In a standard convolutional layer, a convolutional filter acts on all channels of the input. However, the C3 layer of the original LeNet5 architecture uses convolutions that act only on a subset of the input channels. See Table 1. In this problem, implement the original C3 layer as described in LeCun et al.'s paper [1]. Use the starter code `lenet_original.py`. Compared to the regular `conv2d` layer, what is the reduction in parameter count? Does the reduction in parameter count as observed from the `print` statement of the starter code agree with what you would expect from hand calculations?

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

TABLE I

EACH COLUMN INDICATES WHICH FEATURE MAP IN S2 ARE COMBINED BY THE UNITS IN A PARTICULAR FEATURE MAP OF C3.

Hint. In the `__init__` method, create 16 `Conv2d` modules, 6 taking in 3 channels, 9 taking in 4 channels, and 1 taking in 6 channels. You may find `nn.ModuleList` useful in organizing them. In the `forward` method, compute the 16 output channels separately and concatenate them with `torch.cat`. For each of the 16 convolutional modules, use advanced indexing and provide `x[:, list, :, :]` as input.

Remark. The purpose of this problem is to serve as an exercise of implementing a somewhat complex model in PyTorch, rather than to present any significant practical improvement. LeCun et al.'s original intention in designing this layer was to force "a break of symmetry in the network"; without this symmetry breaking, the symmetric channels would wastefully serve duplicate roles. However, the modern view is that the random initialization of the weights and biases is sufficient to break symmetry. When pushed further, having incomplete connections in convolutional layers can significantly reduce the number of trainable parameters. We will later see this in the *split-transform-merge* structure of GoogLeNet, Inception, and ResNext architectures.

Solution. See `lenet_original_sol.py`. The original LeNet5 architecture has 60806 trainable parameters, while the modern one contains 61706. ■

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, *Proceeding of IEEE*, 1998.

[2] K. Janocha and W. M. Czarnecki, On loss functions for deep neural networks in classification, *TFML*, 2017.

[3] L. Hui and M. Belkin, Evaluation of neural architectures trained with square loss vs cross-entropy in classification tasks, *ICLR*, 2021.