Machine Learning & Deep Learning

Tutorial

MSCV/ESIREM

Antoine Lavault
antoine.lavault@u-bourgogne.fr

General Review

Any question or exercise marked with a "*" is typically more technical or goes further into developing the tools and notions seen during class.

***

Problem 1

Regularization and Instance Noise Say we have m labeled data points $(xi, yi)_{i=1}^{m}$, where each $x_i \in R^n$ and each $y_i \in R$. We perform data augmentation by adding noise to each vector every time we use it in SGD. This means for all points i, we have a true input x_i and add noise N_i to get the effective random input seen by SGD:

$$\tilde{X}_i = x_i + N_i$$

The i.i.d. random noise vectors Ni are distributed as $N_i \sim \mathcal{N}(0, \sigma^2 I_n)$.

We can conceptually arrange these noise-augmented data points into a random matrix $\tilde{X} \in R^{m \times n}$, where the row $\tilde{X}_i^T$ represents one augmented datapoint. Similarly, we arrange the labels yi into a vector y.

One way of thinking about what SGD might do is to consider learning weights that minimize the expected least squares objective for the noisy data matrix:

$$\arg\min_w E[||\tilde{X}w - y||^2]$$

1. We will first show that this problem is equivalent to a regularized least squares problem.
   (a) Write the squared norm of a vector as an inner product.
   (b) Remembering that expectation is linear, expand the inside square.
   (c) What is the value of $E[N_i N_i^T]$?
   (d) Finally, show that $E[||\tilde{X}w - y||^2] = ||Xw - y||^2 + m\sigma^2||w||^2$
   (e) Find λ such that we have: $\arg\min_w \frac{1}{m}||Xw - y||^2 + \lambda||w||^2$

First, let's write $X = [X_1^T, \cdots, X_n^T]^T$ as described

Then, $E[||\tilde{X}w - y||^2] = E[\sum(\tilde{X}_i^T w - y_i)^2] = E[\sum((X_i + N_i)^T w - y_i)^2]$

The inner term can be expanded as :

$\sum(X_i^T w + N_i^T w - y_i)^2] = (X_i^T w - y_i)^2 - 2(N_i^T w)(X_i^T w - y_i) + (N_i^T w)^2$

Since the expectancy is linear, we can take the sum of the expectancy and then separate the terms.

We also know that $E[N_i N_i^T]$ is the covariance for N_i i.e. $\sigma^2 I_n$ given the assumptions.

If we simplify the thing above, we have:

$E(\cdots) = \sum_i E[(X_i^T w - y_i)^2] - 2E[(N_i^T w)(X_i^T w - y_i)] + E[(N_i^T w)^2]$

The expectancy does not affect the first term (it is deterministic). The second can see $(X_i^T w - y_i)$ getting out, and the rest is 0 (assumption and the expectancy is linear). The last term can be "expanded," a $N_i^T N_i$ will appear, giving us a $\sigma^2 I_n$.

The final result is then:

$$\sum_i^m [(X_i^T w - y_i)^2 - 0 + w^T \sigma^2 I_n w] = ||Xw - y||^2 + m\sigma^2 ||w||^2$$

Dividing the whole thing by m to match the desired form of the last question gives us $\lambda = \sigma^2$

Let us now consider a simplified example where we only have a single scalar datapoint $x \in R$, and its corresponding label $y \in R$. We are going to analyze this in the context of gradient descent. For the t-th step of gradient descent, we use a noisy datapoint $\tilde{X}_t = x + N_t$ generated by adding different random noise values $N_t \sim \mathcal{N}(0, \sigma^2)$ to our underlying data point x. The noise values for each iteration of gradient descent are i.i.d. We want to learn a weight w such that the squared-loss function $L(w) = \frac{1}{2}(\tilde{X}w - y)^2$ is minimized. We initialize our weight to be $w_0 = 0$.

2. Let w_t be the weight learned after the t-th iteration of gradient descent with data augmentation. Write the gradient descent recurrence relation between $E[w_{t+1}]$ and $E[w_t]$ in terms of x, σ^2, y, and the learning rate η.

   Hint: First, try to get the expression of $\frac{\partial L}{\partial w}$. Then, write the update and remark that w_t and N_t are independent since the N_i are.

   Given the hint, $\frac{\partial L}{\partial w} = (w\tilde{X} - y)\tilde{X} = w(x^2 + 2xN_t + N_t^2) - y(x - N_t)$

   And now, we can write the gradient descent update: $w_{t+1} = w_t - \eta \nabla_w L(w_t)$

   Which can be expanded: $w_t(1 - \eta(x^2 + 2xN_t + N_t^2)) - \eta y(x + N_t)$

   With the expectation, we have: $E[w_{t+1}] = E[w_t](1 - \eta(x^2 + \sigma^2)) + \eta yx$

   The hint helps separate the product of w_t and N_t.

3. For what values of learning rate η do we expect the expectation of the learned weight to converge using gradient descent?

   Hint: For the gradient descent to converge, we need the coefficient on the recurrent term to be between -1 and 1.

   $$-1 < 1 - \eta(x^2 + \sigma^2); 1 - \eta(x^2 + \sigma^2) < 1$$

   This implies $0 < \eta < \frac{2}{x^2 + \sigma^2}$

4. Assuming that we are in the range of η for which the gradient-descent converges, what would we expect $E[w_t]$ to converge to as $t \to +\infty$? How does this differ from the optimal value of w if no noise was used to augment the data?

   One way of doing this is to take an expectation of the gradient and set it to 0. Using the fact that $E[N_t] = 0$ and $E[N_t^2] = \sigma^2$. Another way to see it is to take $E[w_{t+1}] = E[w_t]$.

   This gives us: $w^*(x^2 + \sigma^2) - yx = 0$

   The optimal value of w, if there were no data-augmenting noise, would be $w = y/x$. This means the optimal expected value with noise augmentation is scaled down by a factor of $\frac{1}{1+\frac{\sigma^2}{x^2}}$. This looks like the "shrinkage" term on the eigenvalues with Ridge regression. The learned weight barely changes when σ is small relative to x. When σ is large relative to x, the learned weight shrinks significantly towards zero.

We have shown here that instance noise is a regularization term similar to the Ridge Regression.

Problem 2

Understanding Convolution as Finite Impulse Response Filter For the discrete-time signal, the output of the linear time-invariant system is defined as:

$$y[n] = x[n] * h[n] = \sum_{i=-\infty}^{+\infty} x[n-i]h[i] = \sum_{i=-\infty}^{+\infty} x[i]h[n-i] \tag{1}$$

where x is the input signal, and h is the impulse response (also referred to as the filter). Please note that the convolution operation is to "flip and drag."

1. Is the convolution described in eq. (1) the same as the one used in neural networks? If not, does it change the type of operation?

   For neural networks, the convolutional layer implements eq. (1) without flipping, and this operation is called correlation.

   Those two operations are equivalent in CNN because filter weights are initialized and updated. Even though you implement 'true' convolution, you get the flipped kernel.

   For the rest of the exercise, we will follow the definition in eq. (1).

2. Now, let's consider a rectangular signal with the length of L (sometimes also called the "rect" for short, or the "boxcar" signal).

   The impulse response we want to use is defined as $h(n) = (1/2)^n u(n)$, where $u(n) = 1 for n \geq 0, and 0 everywhere else.$

   Compute and plot the convolution of $x(n)$ with $h(n)$. For illustrative purposes, your plot should start at -6 and end at +12.

   It is suggested to divide the computation into three cases: $n < 0$, $0 \leq n < L-1$, and $n \geq L-1$. Tedious enough...

3. Now let's shift $x(n)$ by N , i.e. $x_2(n) = x(n-N)$. Let's choose $N = 5$. Then, compute $y_2(n) = h(n) * x_2(n)$. Which property of the convolution can you find here?

The output shifts the result by N. Therefore, the output of this convolution is $y[n-N] = y[n-5]$. The convolution operation's translational invariance (shift invariance/equivariance) property can be observed.

Problem 3

Feature Dimensions of Convolutional Neural Network In this problem, we compute the output feature shape of convolutional layers and pooling layers, which are building blocks of CNN. Let's assume that input feature shape is $W \times H \times C$, where W is the width, H is the height, and C is the number of channels of the input feature.

1. A convolutional layer has 4 architectural hyperparameters: the filter size (K, assumed to describe a $K \times K$ filter), the padding size (P), the stride step size (S), and the number of filters (F). How many weights and biases are in this convolutional layer? And what is the shape of the output feature that this convolutional layer produces?

   Number of weights = K^2CF

   Number of biases = F

   W ' = $\lfloor (W - K + 2P)/S \rfloor + 1$

   H' = $\lfloor (H - K + 2P)/S \rfloor + 1$

   C' = F

2. A max pooling layer has 2 architectural hyperparameters: the stride step size (S) and the pooling window size (K, assumed to describe a $K \times K$ window). What is the output feature shape that this pooling layer produces?

   No learned parameters here.

   W ' = $\lfloor (W - K)/S + 1 \rfloor$

   H' = $\lfloor (H - K)/S + 1 \rfloor$

   C' = C

3. Let's take a real example. We are going to describe a convolutional neural network using the following layers:

   - CONV3-F denotes a convolutional layer with F different filters, each of size $3 \times 3 \times C$, where C is the depth (i.e., number of channels) of the activations from the previous layer. Padding is 1, and stride is 1.
   - POOL2 denotes a 2×2max-pooling layer with stride 2 (pad 0)
   - FLATTEN turns whatever shape input tensor into a one-dimensional array with the same values.
   - FC-K denotes a fully connected layer with K output neurons. Note: All CONV3-F and FC-K layers have biases as well as weights. Do not forget the biases when counting parameters.

   Now, we will use this network to infer a single input. Fill in the missing entries in table 1 of the size of the activations at each layer and the number of parameters at each layer. You can/should write your answer as a computation (e.g., $128 \times 128 \times 3$) in the style of the already filled-in entries of the table.

Layer	Number of Parameters	Dimension of Activations
Input	0	$28 \times 28 \times 1$
CONV3-10	$3 \times 3 \times 1 \times 10 + 10$	$28 \times 28 \times 10$
POOL2	0	$14 \times 14 \times 10$
CONV3-10	$3 \times 3 \times 10 \times 10 + 10$	$14 \times 14 \times 10$
POOL2	0	$7 \times 7 \times 10$
FLATTEN	0	490
FC-3	$490 \times 3 + 3$	3

Table 1: A Neural Network

4. In a regular convolutional operation, the kernel slides over the input data contiguously. However, in dilated convolution, the kernel is "dilated" by introducing gaps between its elements, resulting in a larger receptive field for each output pixel, as shown in fig. 1

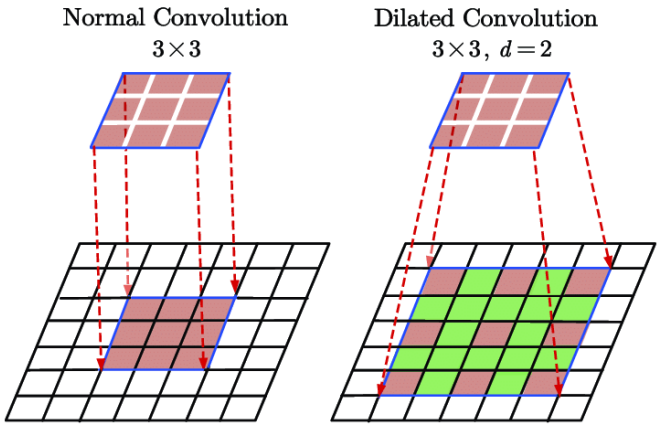


Figure 1: An illustration of Dilated Convolution

The dilation (d) determines the spacing between kernel elements i.e. a dilation of d introduces $d-1$ gaps between two kernel elements. The example above illustrates a 3×3 1-dilated kernel ($d = 1$ is equivalent to a regular convolutional kernel) and a 3×3 2-dilated ($d = 2$).

(a) You are given an input matrix M and 2×2 filter k below. Compute their dilated convolution with $d = 2$. Assume that gaps in the kernel are filled with zeros, stride is 1, and padding is 0.

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}, k = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

Dilate the kernel (it becomes 3×3) and then convolve. The answer is a scalar equal to 20.

(b) Consider a two-layer architecture: $DilatedConv1(3 \times 3, d = 1) \rightarrow DilatedConv2(3 \times 3, d = 2)$

Both layers use the same sized kernel (3x3) but different dilation rates. Compute the size of the receptive field at the output of the final layer. Recall that the receptive field is the region in the *original input image* whose pixels affect the output for a pixel in the specified layer.

7×7. The dilated filter is 5×5 with gaps. But the layer before is 3×3.

Now, time to draw...

Problem 4

Comparing Distributions with the KL divergence

1. Divergence metrics provide a principled measure of the difference between a pair of distributions (P, Q). One such example is the Kullback-Leibler Divergence. Recall the definition of $D_{KL}(P \parallel Q)$.

2. For a fixed target distribution P , we call $D_{KL}(P \parallel Q)$ the forward-KL, while calling $D_{KL}(Q \parallel P)$ the reverse-KL. Due to the asymmetric nature of KL divergence, distributions Q that minimize $D_{KL}(P \parallel Q)$ can be different from those minimizing $D_{KL}(Q \parallel P)$.

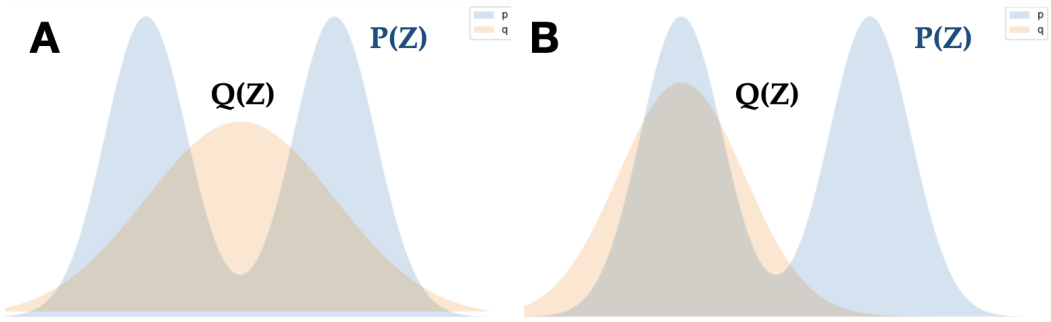


Figure 2: Distribution plots

From the plots in fig. 2, identify which (A, B) corresponds to minimizing forward vs. reverse KL. Give a brief reasoning. Here, only the mean and standard deviation of Q can vary during the minimization.

The easiest key to understanding this is to look at example B. Q is extremely tiny in some places where P is big – this would make the forward-KL $D_{KL}(P \parallel Q)$ enormous.

Meanwhile, everywhere that Q is big, we see that P is also reasonably large in B. So, the reverse-KL $D_{KL}(Q \parallel P)$ is not enormous. This tells us that B corresponds to minimizing reverse-KL. Meanwhile, looking at A, we see that this corresponds to minimizing forward-KL DKL(P ‖Q) because the Q is strongly trying to avoid being small where P is large and trying to strike an "average" type balance between the two modes of P. Doing so puts a higher probability in a location where P is smaller — this would be costly in reverse KL but not that expensive in forward KL.

Problem 5

GANs - Vanishing Gradient with Minimax Objective (*) Let us consider L_G the generator loss for the vanilla GAN, with G the generator and D the discriminator. Show that $\nabla L_G \to 0$ when the discriminator output $D(G(z)) \approx 0$. Why is this problematic for training the generator when the discriminator is well-trained in identifying fake samples?

To simplify the solution, we will introduce σ, the sigmoid function, and h, the discriminator output considered logits. We will also consider the generator and discriminator as parametric models parametrized by ϕ and θ.

$$L_G(\theta;\phi) = E_{z\sim N(0,I)}[\log(1-\sigma(h_\phi(G_\theta(z))]$$

Then, recallaing that $\sigma'(x) = \sigma(x)(1-\sigma(x))$.

$$\frac{\partial L_G}{\partial \theta} = -E_{z\sim N(0,I)}[D_\phi(G_\theta(z))\frac{\partial}{\partial_\theta}h_\phi(G_\theta(z))]$$

The idea is to note that $\log(1-\sigma(f)) \to -\sigma'(f)/(1-\sigma(f))f'$

From the above derivation, it follows that for $D(G(z)) \approx 0$, suggesting that $\partial L/\partial\theta \to$. As the generator update is proportional to the gradient, the vanishing gradient causes generator optimization to be slow and stagnant.

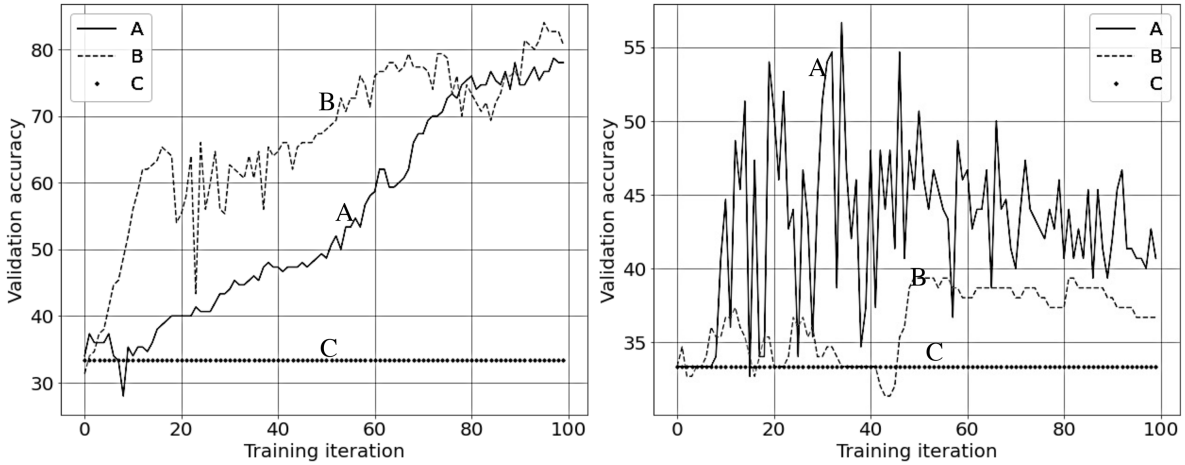
Problem 6

Finding bugs and mishaps in a Neural Network Recall that we worked on the Xavier initialization for $\tanh$ in a previous tutorial, and this method can be extended for sigmoid functions. However, it has been demonstrated that ReLU responds better using He initialization.

Suppose we train two different deep CNNs to classify images: (i) using ReLU as the activation function, and (ii) using sigmoids as the activation function.

For each of them, we try initializing weights with the different initialization methods (Xavier, He, and all zeros), while the biases are always initialized to all zeros.

We plot the validation accuracies with different training iterations in fig. 3:



(i) CNN with ReLU as the activation

(ii) CNN with sigmoid as the activation

Figure 3: Accuracy for two different configurations

1. In Figure 3, which configuration correspond to which initializer?
2. You are designing a neural network to perform image classification. You want to use data augmentation to regularize the training for your model and write this code.

```python
import random
from torch.utils.data import DataLoader, Subset

train_dataset = load_train_dataset()
augmented_dataset = apply_augmentations(train_dataset)
num_data = len(augmented_dataset)
indices = list(range(num_data))
random.shuffle(indices)
split = int(0.8 * num_data)
train_idxs, val_idxs = indices[:split], indices[split:]
train_data = Subset(augmented_dataset, train_idxs)
val_data = Subset(augmented_dataset, val_idxs)
test_data = load_test_dataset()
train_loader = DataLoader(train_data, batch_size=32)
val_loader = DataLoader(val_data, batch_size=32)
test_loader = DataLoader(test_dataset, batch_size=32)

# Train the model
for epoch in range(10):
    for images, labels in train_loader:
        # Training code here
    # Validation code here
# Testing code here
```

On running this code, you find that you get a high training accuracy and validation accuracy but a significantly lower testing accuracy as compared to the validation accuracy.

Someone, somewhere, suggests that there is a bug in your code that is causing this behavior. Identify the bug, briefly explain why it causes the observed behavior, and show how to fix it.