Machine Learning & Deep Learning
Tutorial

MSCV/ESIREM

Antoine Lavault
antoine.lavault@u-bourgogne.fr

MLPs and CNNs

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

Basic concepts. In this exercise, we will use the term convolution to describe the cross-correlation operation to remain coherent with the vocabulary of CNNs. Note that the cross-correlation is a linear operation just like the convolution.

1. To get warmed up, let's review some calculations using convolution and different kinds of padding.
   (a) Suppose you have a $32 \times 32 \times 3$ image (a 32×32 image with 3 input channels). What are the resulting dimensions when you convolve with a $5 \times 5 \times 3$ filter with stride 1 and 0 padding? $28 \times 28 \times 1$. This is called "valid" padding in deep learning frameworks.
   (b) What if we zero-pad the input by 2 (all around)? $32 \times 32 \times 1$. This is called "same" padding in deep learning frameworks.
   (c) Suppose we stack 10 of these $5 \times 5 \times 3$ filters and continue to zero pad the input by 2. What is the new shape of the output, and how many parameters are in our filters (not including any bias parameters)? $32 \times 32 \times 10$. We have, per filter, $5 \times 5 \times 3$ parameters. Ten times. So, 750.
   (d) What would be the spatial dimensions after applying a 1×1 convolution? Think about what this does. A 1×1 convolution does not change the spatial dimensions. For every spatial location, it performs a linear map of the input channels pointwise over space.
   In practice, this is useful for changing the number of channels, for instance, in the skip connection of the ResNet or reducing the output feature map of a generative model to the right amount of channels (3 for RGB, 2 for stereo sound, etc.).
2. We should note that convolutions are a linear operation. Recalling linear algebra, any linear map (between finite-dimensional spaces) can be expressed as a matrix. In this question, we will see how to write a convolution as a matrix multiplication.
   Let's consider a 1D convolution. Consider an input $x \in \mathbb{R}^4$ and a filter $x \in \mathbb{R}^3$. We also write $\bar{x}$ to denote the result of zero-padding the input by 1 on each end. We want to find the matrix W such that:
$$W\bar{x} = \bar{x} * w$$

(a) Write down the expression of $\bar{x}$ and calculate the convolution $\bar{x} * w$.

$\bar{x} = [0, x_1, x_2, x_3, x_4, 0]^T$

$$\bar{x} * w = \begin{bmatrix} 0 + x_1w_2 + x_2w_3 \\ x_1w_1 + x_2w_2 + x_3w_3 \\ x_2w_1 + x_3w_2 + x_4w_3 \\ x_3w_1 + x_4w_2 + 0 \end{bmatrix}$$

(b) Derive from this the expression of W.

$$\begin{bmatrix} w_1 & w_2 & w_3 & 0 & 0 & 0 \\ 0 & w_1 & w_2 & w_3 & 0 & 0 \\ 0 & 0 & w_1 & w_2 & w_3 & 0 \\ 0 & 0 & 0 & w_1 & w_2 & w_3 \end{bmatrix}$$

(c) What remarks can you make about this result?

We can now see that if the filter size is significantly smaller than the input size, the resulting matrix will be highly sparse, indicating that these convolutions efficiently use spatial locality. Additionally, there is significant parameter reuse, as the convolutional filter weights are repeated multiple times within the explicit matrix.

This has several consequences. Firstly, it suggests that convolutional layers possess lower expressiveness than fully connected layers since arbitrary matrices can represent them. Another noteworthy implication is that we have highly efficient tools for performing matrix multiplications. While a straightforward implementation of convolution would involve iterating over all spatial dimensions, it often proves significantly faster to reframe the convolution as matrix multiplication, thanks to these optimizations.

3. Let's briefly review the transpose convolution operator.

Note that early convolutional and pooling layers tend to downsample or reduce the size of tensors, but sometimes we need to upsample, for example, in semantic segmentation or image generation. This operator increases the resolution of the intermediate tensors, which we often want if we want the output of our network to be an image (e.g., of the same size as the input images). Please note that it is sometimes referred to as a deconvolution operator, but it is not the preferred wording because it is an overloaded term with other commonly used definitions.

The transpose convolution can be thought of as flipping the forward and backward passes of the convolution step. In addition, the naming comes from how it can be implemented similarly to convolution but with the weight matrix transposed (along with different padding). Convolutional layers typically downsample images spatially, but sometimes we want to upsample.

(a) What kind of network architecture can be used for a segmentation task?

(b) Let our input be $\begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix}$ and the kernel be $\begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix}$ Assume the input and output channels are 1, padding 0, and stride 1. What is the output of the transpose convolution layer?

A U-Net is quite common for segmentation.

We have $\begin{bmatrix} 0 & 0 & 1 \\ 0 & 4 & 6 \\ 4 & 12 & 9 \end{bmatrix}$

4. Explain Dropout, why it could improve performance, and when we should use it.

   Dropout is a regularization technique that can improve the generalization ability of deep neural networks by reducing overfitting.

   It does so by randomly deactivating neurons during training, promoting a form of robustness by preventing excessive reliance on specific features and avoiding redundant representations.

   Dropout is especially useful for deep neural networks with many parameters and is typically applied to fully connected and convolutional layers.

   It may be less suitable for recurrent neural networks or specialized architectures like transformers, which often have their regularization methods.

5. What features of ResNet, besides better initialization techniques and Batch Normalization, alleviate the vanishing gradient problem?

   In the context of backpropagation, gradients are propagated through skip connections, enabling identity transfers within these connections to mitigate the vanishing gradient problem, even when gradient vanishing occurs within the skip connections.

   The original paper by the authors [He et al., 2016] highlights that the problem is largely addressed through improved initialization techniques and the incorporation of Batch Normalization.

6. Write the definition of the ReLU, leaky ReLU, and ELU activation functions. For $\alpha > 0$, $ReLU = \max(x, 0)$, $lReLU_\alpha = \max(\alpha x, x)$, $ELU_\alpha(x) = \alpha(\exp(x) - 1)$ if $x < 0, x$otherwise).

Problem 2

Deriving Xavier Initialization Let our activation be the $\tanh$ activation.

1. Compute the derivative of $\tanh = 2\sigma(2\cdot) - 1$. What can we say about this function about the outputs for small inputs?

2. Write the output of the i-th layer neuron z_i, for the $a_k, 1 \leq k \leq D_a$ and $W_{i,j}$ the neuron parameter, with the $\tanh$ activation

3. We furthermore assume that weights and inputs are i.i.d. and centered at zero, and biases are initialized as zero. We want the magnitude of the variance to remain constant with each layer.

   (a) Write the condition expressed above mathematically.

   (b) Given all the above assumptions, derive the Xavier Initialization for $\tanh$ activation, which initializes each weight as:

   $$W_{i,j} = \mathcal{N}(0, 1/D_a)$$

   where D_a is the dimensionality of a.

   Hint: one of the previous questions gives us a certain approximation of $\tanh$ *near a certain point. Also, for independent random variables X and Y, the variance of their sum or difference is the sum of their variances.*

$\tanh$ is linear near 0. If we have small enough weights, we have $z_i \approx \sum_j W_{i,j} a_j$. The condition we are after is $Var(z_i) = Var(a_i)$

Then $Var(z_i) = Var(\sum_j W_{i,j} a_j = \sum_j Var(W_{i,j} a_j)$

Be wary, $W_{i,j}$ is random!

$$\sum_{j=1}^{D_a} Var(W_{i,j} a_j) = \sum_{j=1}^{D_a} E(W_{i,j}^2 a_j^2) - E(W_{i,j} a_j)^2$$

Since a_j and $W_{i,j}$ are independent,

$$Var(z_i) = \sum_{j=1}^{D_a} [E(W_{i,j}^2) E(a_j^2) - E(W_{i,j})^2 E(a_j)^2]$$

Hence, since the weights and inputs are centered,

$$Var(z_i) = \sum_{j=1}^{D_a} [Var(W_{i,j}) Var(a_j)]$$

And a and W are both i.i.d.,

$$Var(z) = Var(W_{i,j}) Var(a_j) D_a$$

Our original condition can finally be expressed as $Var(W_{i,j}) = 1/D_a$

Problem 3

Let's get ReLU activated! *It's hard finding puns sometimes....*

1. Compute the output of the forward pass of a ReLU layer with input x given below:

$$\begin{bmatrix} 1.5 & 2.2 & 1.3 & 6.7 \\ 4.3 & -0.3 & -0.2 & 4.9 \\ -4.5 & 1.4 & 5.5 & 1.8 \\ 0.1 & -0.5 & -0.1 & 2.2 \end{bmatrix}$$

2. Recall the derivative of $ReLU$.

3. With the gradients with respect to the outputs dL/dy given below, compute the gradient of the loss with respect to the input x using the backward pass for a ReLU layer:

$$dL/dy = \begin{bmatrix} 4.5 & 1.2 & 2.3 & 1.3 \\ -1.3 & -6.3 & 4.1 & -2.9 \\ -0.5 & 1.2 & 3.5 & 1.2 \\ -6.1 & 0.5 & -4.1 & -3.2 \end{bmatrix}$$

Note: ReLU treats every entry independently, and so does it for the backward pass.

4. What advantages does using ReLU activations have over sigmoid activations?

5. ReLU layers have non-negative outputs. What is a negative consequence of this problem? What layer types were developed to address this issue?

The backward pass zeros of the same gradient entries zeroed out in the forward pass.

ReLU doesn't have a vanishing gradient problem like sigmoid. However, this is less of an advantage, given that one can use Batch Normalization to centralize inputs and remain in the "good" part of the sigmoid.

The biggest issue ReLU suffers is the dying ReLU problem, where this unit always outputs 0, no matter what the input is. Once the ReLU ends up in this state, it is unlikely to recover since its gradient is also 0, and any gradient descent methods will not alter its weights. This is why other activation functions were developed, including Leaky-ReLU or ELU.

Problem 4

Definition 1 (Latent Variable Models)**.** A latent variable model p is a probability distribution over observed variables x and latent variables z (variables that are not directly observed but inferred), $p_\theta(x, z)$.

Because we know z is unobserved, the learning methods we have seen in class (like supervised learning methods) are unsuitable.

Towards Variational Auto-Encoders (*) Our learning problem of maximizing the log-likelihood of the data turns is $\arg\max_\theta \sum_{i=1}^{N} \log p_\theta(x_i)$ and $p(x) = \int p(x|z)p(z)dz$. Unfortunately, the integral is intractable, but we will discuss ways to find a tractable lower bound.

1. Recall quickly the architecture of a (bottleneck) auto-encoder. Encoder, code, Decoder. More or less a PCA.

2. In the case of the VAE [Kingma and Welling, 2014], the encoder maps a high-dimensional input x (like the pixels of an image) and then (most often) outputs the parameters of a Gaussian distribution that specify the hidden variable z. In other words, it outputs $\mu_{z|x}$ and $\sigma_{z|x}$.

   What could be a clear drawback of using a random Gaussian to specify the hidden variable? VAEs learn an explicit distribution of the data by fitting it into a multivariate Gaussian, and the output is blurry because of the conditional independence assumption of the samples given latent variables. This is not true for most realistic distributions, like natural images. Indeed, since the real posterior is often far from a multivariate Gaussian. We observe a lot of variance/noise added to the model, and this causes blurriness when we decode since Maximum Likelihood training will distribute the probability mass diffusely over the data space

3. To train VAEs, we find parameters that maximize the likelihood of the data, but the integral inside is intractable. However, we can show that optimizing a tractable lower bound on the data likelihood is possible, called the Evidence Lower Bound (ELBO).

   Write out the log-likelihood objective of a **discrete** latent variable model.

   $\arg\max_\theta \sum_i \log p_\theta(x_i) = \sum \log(p_\theta(x_i|z)p_Z(z))$

4. Using the Jensen's inequality, show that:

$$\sum_{i=1}^{N} \log p_\theta(x_i) \geq \sum_{i=1}^{N} E_{q(z|x_i)}[\log p_Z(z) - \log q(z|x_i) + \log p_\theta(x_i|z)]$$

   With p_Z being the real latent distribution

   In that case, we use the concave case of the Jensen inequality, i.e., $\log E(x) \geq E(\log(x))$.

   The idea is to write $\frac{q(z|x_i)}{q(z|x_i)} p_Z(z) p_\theta(x_i|z)$, find out it looks a lot like an expectancy, and finally, apply the Jensen Inequality.

   QED.

5. Show that our derived approximation can be reformulated as the Evidence Lower Bound (ELBO),

$$L_i = E_{z \sim q_\varphi(z|x_i)}[\log p_\theta(x_i|z)] - D_{KL}(q_\varphi(z|x_i)||p(z))$$

   Hint: $D_{KL}(q \parallel p) = -E[\log p(x)] - H(q)$

   Trivial if we recall that $D_{KL}(q \parallel p) = -E[\log p(x)] + E[\log q]$, and using the question above.

6. To optimize the Variational Lower Bound derived in the previous problem, which distribution do we sample z from?

   q

7. The re-parametrization trick allows us to break $q_\varphi(z|x)$ into a deterministic and stochastic portion, and re-parametrize from $q_\varphi(z|x)$ to $g_\varphi(x, \varepsilon)$. In fact, we can let, $z = g_\varphi(x, \varepsilon) = g_0(x) + \varepsilon g_(x)$ where $\varepsilon \sim p(\varepsilon)$. This reparametrization trick is simple to implement and shows good behavior in practice.

   Let us get an intuition for how we might use re-parametrization in practice. Assume we have a normal distribution q, parametrized by θ, such that, $q_\theta(x) \sim N(\theta, 1)$, and we would like to solve,

$$\min_\theta E_q[x^2]$$

   Use the re-parametrization trick on x to derive the gradient.

   First, we can write $x = \theta + \varepsilon, \epsilon \sim N(0, 1)$, separating deterministic and stochastic parts.

   We then have $E_p[2(\theta + \varepsilon)]$ as the final result, which is easier to compute and separate.

References

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2016-Decem, pages 770–778.

[Kingma and Welling, 2014] Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. In Bengio, Y. and LeCun, Y., editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.