Deep Learning Exam (OPT8)

Caio Corro, Michèle Sebag

November 20, 2019

When you answer questions, you must give an explanation that shows that you have a deep understanding of the answer. You can use examples (and even draw them). You can answer questions in English or French. All three parts are independent of each other.

- Students in the 1st year of master: answer questions in parts 1 to 4 (i.e. you can skip the deep generative models section)
- Students in the 2nd year of master: answer all questions!

Write explicitly on your document if you are in first or second year.

1 (5pts) Neural networks basics

1. **(2pts)** Describe a multilayer perceptron with one hidden layer for classification (e.g. MNIST digit classification) and the associated negative log-likelihood loss used for training. You should introduce the notation you use and describe the different components.
2. **(1pts)** What is the update expression for parameters during training? (we assume we rely on stochastic gradient descent with a single example per batch)
3. **(2pts)** Derive the gradient of the output layer parameters (output projection and output bias) with respect to the negative log-likelihood loss.

2 (10pts) Neural network training

1. **(1pts)** During training, we observe that the training loss is decreasing but the dev loss is increasing. What is happening? ("there is a bug in the code" is not a valid answer!) How can you prevent this issue?
2. **(1pts)** During training, we observe that the training loss is increasing. What is happening? ("there is a bug in the code" is not a valid answer!) How can we prevent this issue?
3. **(1pts)** What is the difference between the perceptron loss and the hinge loss?
4. **(1pts)** What is one problem of the sigmoid activation function for deep neural networks?
5. **(2pts)** Let's consider a (deep) multilayer perceptron using tanh activation functions for each hidden layer. What properties do you want your network to have at initialization? Why?
6. **(1pts)** What are the pros and cons of the relu activation function with respect to its gradient?
7. **(1pts)** What are the benefits of using batches of several instances during training?
8. **(2pts)** A simple way to optimize a neural network is to rely on the stochastic gradient descent algorithm. Let's put aside the fact that this method rely on a Monte-Carlo estimation of the gradient, what are two other limits of this optimization method? Are they easy to circumvent?

3 (6pts) Neural network implementation

1. **(1pts)** What can be problematic in the implementation of the softmax function? How do you fix this problem?
2. **(1pts)** What is a computation graph?
3. **(2pts)** Give a brief description of the back-propagation algorithm.
4. **(2pts)** What kind of information do you need to store to compute the gradient? Why are in-place operations problematic? Give an example.

4 (4pts) Convolutional Neural Networks (CNNs)

1. **(2pts)** What is the main property of CNNs?
2. **(2pts)** Describe the different components of a CNN.

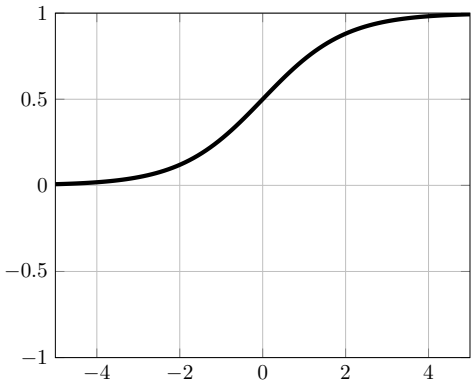
5 (10pts) Deep generative models: Variational Auto-Encoders (VAEs) and Generative Adversarial Networks (GANs)

Student in the 2nd year of master only.

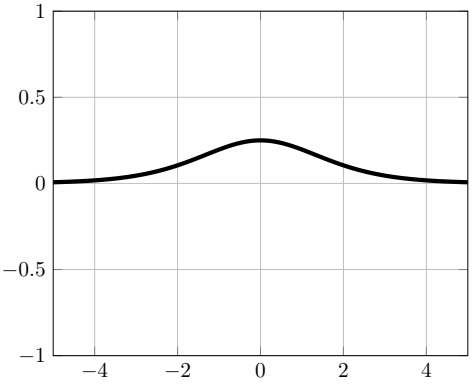
1. **(2pts)** How would you describe the differences between a standard Auto-Encoder and a Variational Auto-Encoder? For which purpose would you use one or the other?
2. **(2pts)** What technical constraints do VAEs impose on the latent variable distribution? Similar question for GANs? That is, is there distribution there cases where you can use one and not the other? (think about the loss functions and how you implement these networks) In which case?
3. **(1pts)** We assume we train a GAN with the binary cross entropy loss for the discriminator. What problem can we encounter when updating the generator parameters? Let $y \in \{0, 1\}$ be the gold label (0 if the discriminator input was generated, 1 if it comes from the dataset) and $w \in [0, 1]$ be the output of the discriminator that was compute with a sigmoid function, the binary cross entropy loss is:
$$-(y \log p + (1 - y) \log(1 - p))$$
4. **(2pts)** VAEs are usually trained by maximizing the Evidence Lower Bound. How can you interpret terms 1 and 2?
$$\max_{\theta,\phi} \quad \underbrace{\mathbb{E}_{q_\phi(z|x)}\left[p_\theta(x|z)\right]}_{1} - \underbrace{\mathrm{KL}\left[q_\phi(z|x), p(z)\right]}_{2}$$
5. **(1pts)** Let $q_\phi(z|x)$ and $p(z)$ be normal distributions. Can you compute these terms?
6. **(1pts)** What is the reparameterization trick and why is it necessary?
7. **(1pts)** Let $q_\phi(z|x)$ and $p(z)$ be categorical distributions (i.e. z is a random variable defined on a finite set). Is the ELBO computation tractable? Explain.

Cheat sheet

Sigmoid

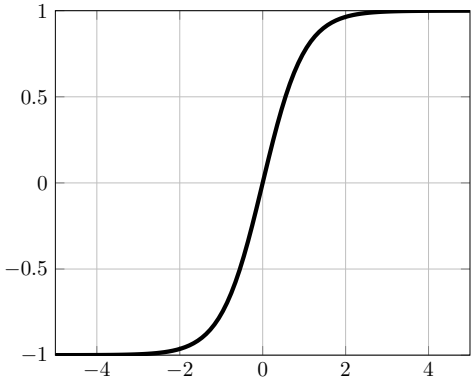


$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

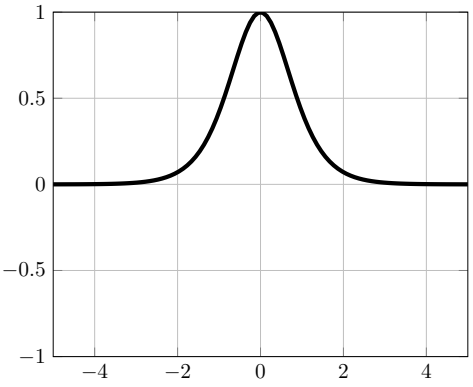


$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

Hyperbolic tangent

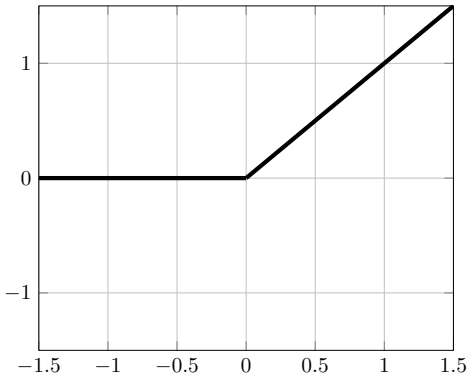


$$\tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}$$

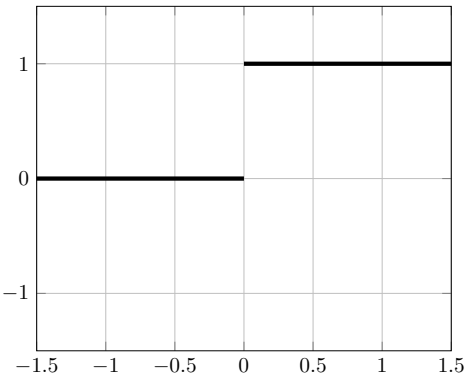


$$\tanh'(x) = 1 - \tanh(x)^2$$

Rectified Linear Unit



$$\text{relu}(x) = \begin{cases} 0 \text{ if } \leq 0 \\ x \text{ otherwise} \end{cases}$$



$$\text{relu}'(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x > 0 \\ \text{undefined otherwise} \end{cases}$$

1. Neural Networks basics

1. MLP with one single layer

Input data x $\longrightarrow$ $z^{(1)} = \sigma\left(w^{(1)} x + b^{(1)}\right)$ $\longrightarrow$ output $z^{(2)} = w^{(2)} z^{(1)} + b^{(2)}$

σ is the non linear activation function

$w^{(1)}$: weights of hidden layer

$b^{(1)}$: bias of hidden layer

$z^{(2)} \xrightarrow{\text{softmax}} \mu = \text{softmax}(z^{(2)})$

$\hat{y} = \text{argmax}\ y^T \mu$, $y \in \{K\}$

we use the negative log likelihood as loss function

$$\mathcal{L} = -\log p(y \mid \mu) = -\log y^T \mu = -\log \frac{\exp(y^T z^{(2)})}{Z} =$$

$$= -y^T z^{(2)} + \log \sum_j \exp(z_j^{(2)})$$

1. We use gradient descent to update the parameters

$$w^{(1)} = w^{(1)} - \varepsilon \cdot \nabla_{w^{(1)}} \mathcal{L}$$

$$b^{(1)} = b^{(1)} - \varepsilon \nabla_{b^{(1)}} \mathcal{L}$$

3. $\frac{\partial \mathcal{L}}{\partial b^{(2)}}$ and $\frac{\partial \mathcal{L}}{\partial w^{(2)}}$

a) $$\frac{\partial \mathcal{L}}{\partial b_i^{(2)}} = \sum_j \frac{\partial \mathcal{L}}{\partial z_j^{(2)}} \frac{\partial z_j^{(2)}}{\partial b_i^{(2)}} = \frac{\partial \mathcal{L}}{\partial z_i^{(2)}} = \nabla_{z^{(2)}} \mathcal{L} = -y + \text{softmax}(z^{(2)})$$

- $\frac{\partial z_j^{(2)}}{\partial b_i^{(2)}} = \mathbb{1}[i = j]$
- $\frac{\partial \mathcal{L}}{\partial z_j^{(2)}} = -y_j + \frac{\exp(z_j^{(2)})}{\sum_i \exp(z_j^{(2)})} = -y_j + \text{softmax}(z^{(2)})_j$

b) $$\frac{\partial \mathcal{L}}{\partial w_{k,l}^{(2)}} = \sum_j \frac{\partial \mathcal{L}}{\partial z_j^{(2)}} \frac{\partial z_j^{(2)}}{\partial w_{k,l}^{(2)}} = \frac{\partial \mathcal{L}}{\partial z_k^{(2)}} \cdot z_l^{(1)}$$

- $\frac{\partial z_j^{(2)}}{\partial w_{k,l}^{(2)}} = z_l^{(1)} \mathbb{1}[j = k]$

we can rewrite $z^{(2)}$ as follows:

$$z^{(2)}_k = \sum_i w^{(2)}_{k,i} z^{(1)}_i + b^{(2)}_k$$

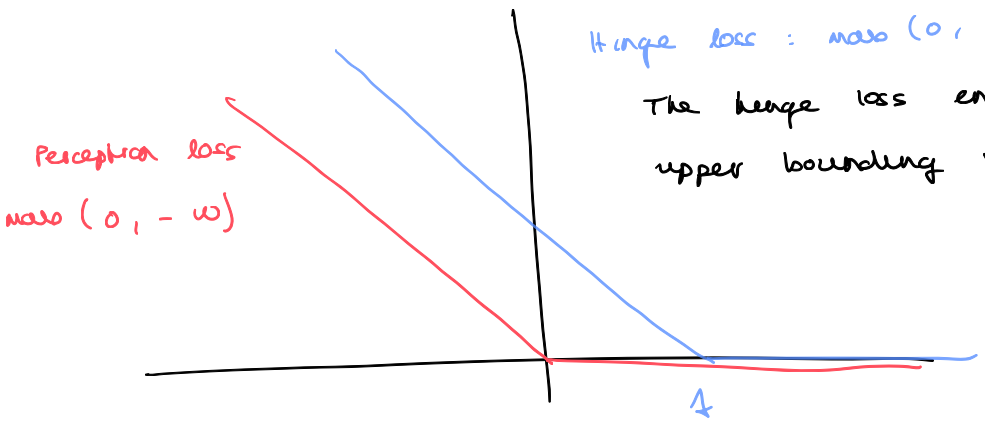
Hence, we have: $\nabla_{W^{(2)}} \mathcal{L} = \underbrace{\left(\nabla_{z^{(2)}} \mathcal{L}\right)^T z^{(1)}}_{\text{outer product}} = \left(-y + \text{softmax}(z^{(1)})\right)^T z^{(1)}$

2. Neural network training

1. This means that we are overfitting, hence, our NN is not able to generalize well and is learning by hard the training samples. To avoid this, we need to introduce some type of regularization. For example:
   1. Dropout: this turns off neurons during the training process but not during evaluation
   2. Regularization in the loss function

2. If the training loss is increasing, this could be because the learning rate is too big. Hence, we can either reduce it or add a weight decay.

3.



The hinge loss encourages a margin by upper bounding the number of mistakes

4. The sigmoid activation function has the following problems:

a) It is not zero centered, which means that it encourages one direction over the other

b) It is computationally expensive, since it involves exponentials

c) It suffers from vanishing gradient. This is because far from zero, the derivative of the sigmoid function is close to zero, and when performing gradient descent the weights are updated proportionally to the partial derivative of the loss wrt the weight. This partial derivative includes the derivative of the sigmoid function. Hence, if it is zero, there won't be updates

d) The derivative of the sigmoid takes values between 0 and 0.25. So even in the best case scenario there will be vanishing gradient

5. The derivative of the tanh takes values between 0 and 1. Hence we want the weights and bias to be initialized closed to zero. However, we cannot set them to 0 because this will prevent the algorithm from learning (there wouldn't be any update)

Hence, we can use Xavier initialisation, which is a random initialisation but close to zero.

The reason why we want them to be close to zero is so that the derivative is close to 1 and hence, the first layers can still learn as we prevent the vanishing gradient.

6.

PROS	CONS
• Easy to compute	• Dying relu: whenever the input is negative, the relu outputs 0, so the learning algorithm does not work anymore since there are no updates

- It is not zero centered

7. When using batches we don't need to calculate the gradient wrt to every single sample, so we can reduce the computational time of the backpropagation

8. It might take more time to converge and the steps might be noisier due to frequent updates. In addition, we cannot vectorize it.

3. Neural Network Implementation

1. Since we are computing exponentials, it can suffer from overflow. In order to avoid this, we can use the following trick

$$\text{softmax}(w)_i = \frac{\exp(w_i)}{\sum_j \exp(w_j)} = \frac{\exp(w_i)\exp(-b)}{\sum_j \exp(w_j)\exp(-b)} =$$

$$= \frac{\exp(w_i - b)}{\sum_j \exp(w_j - b)} \longrightarrow \text{usually we choose } b = \max_i w_i$$

2. A computation graph is a graphical representation of mathematical relations

The nodes of the computation graph correspond to mathematical operations

3. The backpropagation algorithm aims to calculate the derivatives of the loss function with respect to the weights and biases. In order to be able to use the backpropagation we need the loss function to be able to be written as the sum of the loss for every example, and also the loss has to be written as a function of the output

once the forward pass has been completed, we calculate the derivatives in reverse order (it is basically an application of the chain rule)

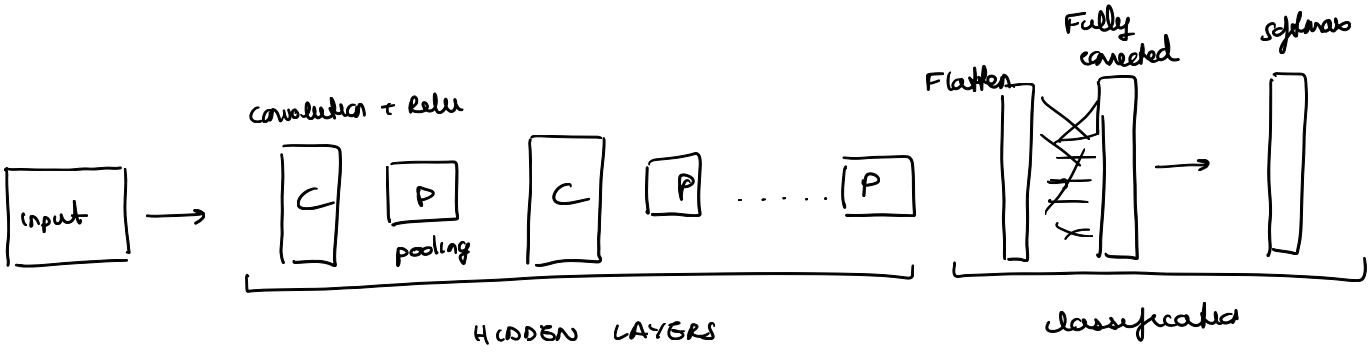
4. • To compute the gradient we need to store the backpointers

- The problem with in place operations is that instead of creating new tensors it updates the value of existing ones. Hence, this can erase the forward values and prevent the backpropagation algorithm from working
- Example:

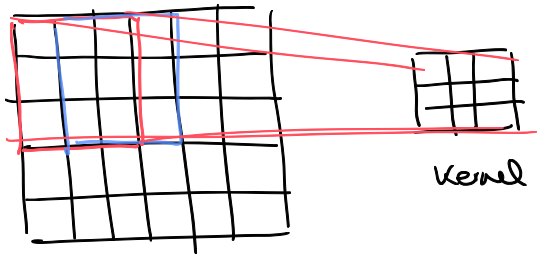
4. CNNs

1. one of the main properties of CNNs is that they are able to preserve spatial relations, contrary to MLP. In addition, the weights are shared so the number of parameters is significantly less than in MLP. (A addition CNNs are equivariant to translation

2. Description of CNN



CONVOLUTION LAYER: performs a dot product between two matrices. One is the kernel and the other one is the matrix that carries the information



we can choose the padding and stride.

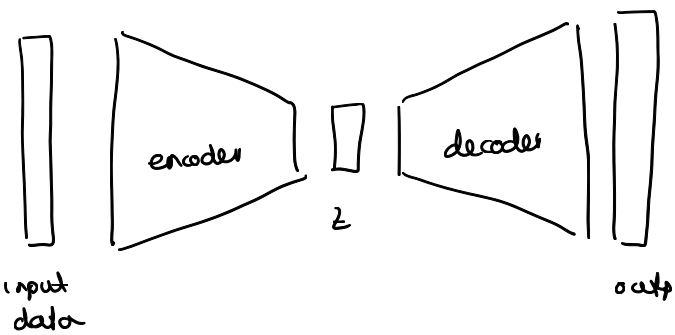
The stride is the size of the step and the padding is used to include "empty" pixels around the matrix so that the outer elements have the same weight as the ones in the center and also it ensures that after convolution we have the same dimension

POOLING : it aggregates the output and prevents the dimension from exploding it denotes a summary statistic of the outputs

FULLY CONNECTED LAYER: neurons are fully connected in this layer

5. DEEP GENERATIVE MODELS

1. Auto encoder → an autoencoder consists of two neural networks, an encoder and a decoder, and it is used for data compression. The objective is to learn a representation of data in a smaller dimension without loosing a lot of information. The idea is that we feed the encoder with some data, that is encoded and decoded, and we compare the ~~loss~~ output with the input. Then we backpropagate the loss to update the parameters of both neural networks

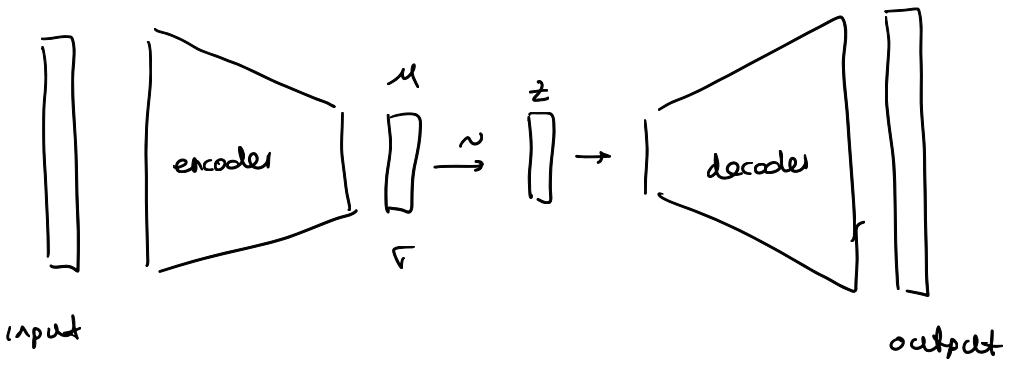


The problem with autoencoders is the lack of regularity of the latent space. Since they are trained to lose as little information as possible, they generally suffer from overfitting so we cannot guarantee that if

we take an element of the latent space and we decode it we will obtain a meaningful element of the input space. Hence, autoencoders cannot be used for data generation

This is where variational autoencoderes come in place

The main difference between VAE and AE is that the former, instead of generating a point of the latent space, it generates the parameters of a distribution, which is a Gaussian distribution. Hence, we have the following



However, we need to impose some regularization. This is because otherwise, the VAE could behave as a standard AE by outputing a zero σ, hence we generate single points in the latent space. To prevent this, the loss is composed of two terms:

- A latent term, which is the KL divergence between the prior distribution $p(z)$ and the encoder $q(z|x)$, and that we want to minimize so that we ensure that $q(z|x)$ is Gaussian.
- A reconstruction term, $E_q(\log p(x|z))$, which is the log likelihood and that we want to maximize.

2. VAEs → it has to be a Gaussian distribution

GANs → there are no constraints on the distribution

?

3.

4. Term 1: reconstruction error. We want to maximise this term, since it tells how close is our model from explaining the data

Term 2: KL divergence. We want to minimise it. It is a restriction we impose to ensure that we have a Gaussian distribution in the latent space

5. The KL has a closed formula for Gaussian distributions

$$KL\left(q_\phi(z|x) \,|\, p(z)\right) = \int q_\phi(z|x) \log \frac{q_\phi(z|x)}{p(z)} d$$

The first term has to be computed empirically, using for example MC aprox.

6. The reparametrization trick is necessary because $N(\mu, \sigma^2)$ is not differentiable

Hence, instead of sampling $z \sim N(\mu, \sigma^2)$, we do the following

$$\varepsilon \sim N(0, 1)$$

$$z = \mu + \sigma\varepsilon$$

The reason why we can train VAE using backpropagation is because we are using the reparametrization trick.

7. There is no close formula for the ELBO and it is not computationally tractable