Contrôle de Connaissance
Master Recherche Informatique, parcours AIC - Université Paris-Saclay
TC Deep Learning

Alexandre Allauzen, Michèle Sebag

9:00-12:00 Nov. 13th, 2018

Documents autorisés: supports et notes de cours

Lisez tout l'énoncé. Pour toutes les questions la clarté de la rédaction joue un rôle important ; justifiez vos réponses brièvement.

Partie I. Questions de cours (5 points)

1. During training, we observe that the loss function is increasing on training data. What is going on? (single answer)

   A. There is not enough regularization and the network is overfitting;

   B. There is too much regularization and the network is underfitting;

   C. The learning rate is too big;

   D. The learning rate is too small.

2. Let inputs of a network be vectors $x \in \mathbb{R}^d$. Describe the architecture of a convolutional network using these inputs. Which properties does it satisfy?

3. We now consider matrix inputs $x \in \mathbb{R}^{d \times d'}$, e.g. images. Describe a convolutional network and its properties.

4. How should a neural network be initialized when it is build with sigmoid activation functions? Which issue should be avoided? Does this problem exist with deep and/or shallow networks?

5. Will two differently initialized networks with the rule you proposed in question 4 attain the same solution after training?

6. Will a neural network be correctly trained if all weights are initialized to 0? and if all bias vectors are initialized to 0?

7. Let
$$\mathcal{E} = \{(x_i, y_i), i = 1 \ldots n, x_i \in \mathbb{R}^d, y_i \in \{-1, 1\}\}$$
be a set of examples. We assume an acyclic computation graph G with n neurons, where arcs (i, j) are sampled randomly and where weights $w_{i,j}$ are sampled from distribution $\mathcal{N}(0, 1)$. All neurons are connected to inputs. We only train the n output neurons. What function is optimized? Is it a convex or a non-convex optimization problem?

Partie II. Algorithme d'apprentissage (9 points)

We assume a feed-forward network with a single hidden layer. Let $\boldsymbol{x}^{(1)}$ be the input vector. The hidden layer is $\boldsymbol{y}^{(1)} = f^{(1)}(\boldsymbol{W}^{(1)}\boldsymbol{x}^{(1)})$. The output layer is $\boldsymbol{y}^{(2)} = f^{(2)}(\boldsymbol{W}^{(2)}\boldsymbol{x}^{(2)})$, with $\boldsymbol{x}^{(2)} = \boldsymbol{y}^{(1)}$. The activation function ($f^{(1)}$) of the hidden layer is hyperbolic tangent. We consider a binary classification task, so the output layer as a single neuron that can be interpreted as the probability that $\boldsymbol{x}^{(1)}$ belongs to class $c = 1$. We maximize the likelihood of training data, i.e.:

$$l(\boldsymbol{\theta}, \boldsymbol{x}, c) = -\big(c \log(\boldsymbol{y}^{(2)}) + (1 - c) log(1 - \boldsymbol{y}^{(2)})\big),$$

where $\boldsymbol{x}$ is a training sample, c the gold output ($c = 0$ or 1) and $\boldsymbol{\theta}$ the parameters of the network. Note that the output is a scalar so $\boldsymbol{W}^{(2)}$ is a row matrix.

1. Which function should we apply to the output?

2. Write the update rule for the output layer $w_j^{(2)}$ which correspond to element j of $\boldsymbol{W}^{(2)}$. To this end, proceed as follows:
   - Express the value $y^{(2)}$ in term of $\boldsymbol{x}^{(2)}$ and $\boldsymbol{W}^{(2)}$.
   - Compute the derivative of $w_j^{(2)}$ with respect to the loss.
   - Write and describe the update rule.
3. Similarly for hidden layer $w_{kj}^{(1)}$.[1]
4. Describe the learning algorithm for this network.

Partie III. Auto-encodeurs (5 points)

We consider an unlabeled dataset:

$$\mathcal{E} = \{x_i, i = 1 \dots n, x_i \in \mathbb{R}^d\}$$

- What is an auto-encoder? What is the associated loss function? What is the goal of auto-encoders?
- What is the loss function of a denoising auto-encoder? What is its purpose against standard auto-encoders?
- How do you choose the number of hidden neurons of an auto-encoder?
- Is the euclidean distance in the hidden state space a good bound on the distance in the initial space?

Partie IV. Exemples adversariaux (5 points)

We now consider adversarial examples. We assume a training set defined as follows:

$$\mathcal{E} = \{(x_i, y_i), i = 1 \dots n, x_i \in \mathbb{R}^d, y_i \in \{-1, 1\}\}$$

. The network is a feed-forward network with 3 hidden layers and d neurons per layer, trained on $\mathcal{E}$.

- What is an adversarial example? How to construct an adversarial example?
- Can a human be fooled by an adversarial example?
- Propose a loss function inspired by previous questions.

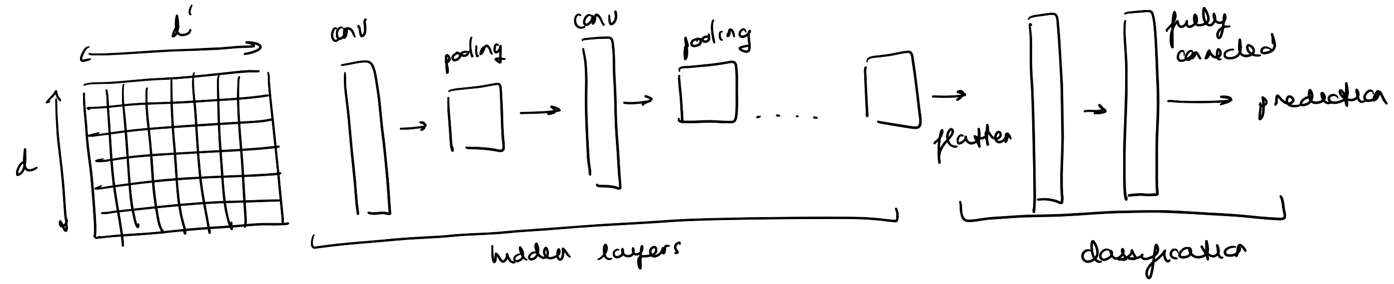
[1] The derivative of the hyperbolic tangent function $tanh$ is $tanh'(a) = 1 - tanh^2(a)$.

1. QUESTIONS DU COURS

1. D

2.

3 Suppose we have images as inputs. Hence we have a matrix, $X \in \mathbb{R}^{d \times d'}$



we select the filters or kernels that we want to apply

Elements

1. convolutional layer: we select the filter (hence, the type of features or relations that we are going to look for) It is used to extract features

2. Pooling: we extract statistical information. It ensures spatial /translation equivariance and takes into account spatial relations. It also prevents the dimensions from exploding

3. Fully connected layer: it is placed before the output and it is where the classification begins

4. Dropout: to avoid overfitting. We turn off some layers for training

5. Activation functions: it adds non linearity and allows more complex relations.

4. The problem of the sigmoid activation function is known as vanishing gradient. The gradient of the sigmoid takes values between 0 and 0.25.

Hence, when performing backpropagation, we are multiplying the gradient by at most 0.25. This prevents the first layers from learning and can make convergence much slower

In order to avoid 0th gradient, the weights and bias should be close to 0. However, 0 or constant values are not allowed since this would mean that all the weights learn the same features at each epoch, so the CNN will not learn anything.

One of the most commonly used methods is Xavier initialisation, which sets the bias to 0 while generating the weights from a uniform distribution.

5. Yes

6. No. We have to avoid zero initialisation, as this would make gradients equal and hence all the weights will remain the same, preventing from learning

2. ALGORITHM D'APPRENTISSAGE

$$x^{(1)} \in \mathbb{R}^{d \times 1} \longrightarrow \underbrace{y^{(1)} = f(w^{(1)} x^{(1)})}_{\text{hidden layer}} \longrightarrow \underbrace{y^{(2)} = w^{(2)} y^{(1)}}_{\text{output layer}} \longrightarrow \text{binary classification}$$

$y^{(2)}$ is a scalar
$w^{(2)}$ is a row matrix

$f = \tanh$

$$\ell(\theta, v, c) = -\left(c \cdot \log y^{(2)}\right) + (1-c)\log\left(1-y^{(2)}\right)$$

1. Since we have a binary classification, we should use the sigmoid function, which converts the values between 0 and 1

$$P(y=1 \mid u) = u = \sigma(y^{(2)})$$
$$P(y=0 \mid u) = 1-u = 1-\sigma(y^{(2)})$$

2. $$\frac{\partial \mathcal{L}}{\partial w_i^{(2)}} = \frac{\partial \mathcal{L}}{\partial y^{(2)}} \cdot \frac{\partial y^{(2)}}{\partial w_i^{(2)}} = \frac{\partial \mathcal{L}}{\partial y^{(2)}} \cdot y_i^{(1)} \sigma'(y^{(2)})$$

- $$y^{(2)} = \sigma\left(\sum_j w_j^{(2)} \cdot y_j^{(1)}\right)$$
- $$\frac{\partial \mathcal{L}}{\partial y^{(2)}} = -\frac{c}{y^{(2)}} - \frac{1-c}{1-y^{(2)}}$$

$$\frac{\partial \mathcal{L}}{\partial w^{(2)}} = \left[-\frac{c}{y^{(2)}} - \frac{1-c}{1-y^{(2)}}\right] \cdot y^{(1)} \sigma'(y^{(2)})$$

we use gradient descent:

$$w^{(2)} = w^{(2)} - \varepsilon \cdot \nabla_{w^{(2)}} \mathcal{L}$$

3. For hidden layer 1.

$$\frac{\partial \mathcal{L}}{\partial w^{(1)}_{k,l}} = \frac{\partial \mathcal{L}}{\partial y^{(2)}} \cdot \frac{\partial y^{(2)}}{\partial y^{(1)}} \cdot \frac{\partial y^{(1)}}{\partial w^{(1)}_{k,l}} \longrightarrow$$

- $$\frac{\partial \mathcal{L}}{\partial y^{(2)}} = -\frac{c}{y^{(2)}} - \frac{1-c}{1-y^{(2)}}$$
- $$\nabla_{y^{(1)}} y^{(2)} = w^{(2)} \sigma'(y^{(2)})$$
- $$\frac{\partial y^{(1)}_i}{\partial w^{(1)}_{k,l}} = f' x^{(1)}_l \mathbb{1}[i=k] \longrightarrow \nabla_{w^{(1)}_{k,l}} y^{(1)} = f'(x^{(1)})^T = f'(y^{(1)}) \odot (x^{(1)})^T$$

element wise multiplication

$$y^{(1)}_i = f\left(\sum_j w^{(1)}_{i,j} x^{(1)}_j\right)$$

element wise

$$\nabla_{w^{(1)}} \mathcal{L} = \underbrace{\left[\sigma'(y^{(2)}) \cdot \left(-\frac{c}{y^{(2)}} - \frac{1-c}{1-y^{(2)}}\right)\right]}_{\text{scalar}} \cdot \left(f'(y^{(1)}) \odot x^{(1)}\right) \cdot w^{(2)}$$

4. we feed the inputs through the hidden layers. we calculate the output and we compute the loss. we update the weights by backpropagating the loss (derivative of the loss wrt weights) we use gradient descent to update the weights.

3. AUTO ENCODERS

1. An auto encoder is a neural network that is formed by an encoder, a decoder and a loss function. Both encoder and decoder are neural networks. The objective of an autoencoder is to compress and decompress data loosing as little information as possible. Hence, the output is meant to be, ideally, a copy of the input

The loss function for autoencoders can be

a) MSE $\rightarrow \sum_i \| (w' \circ w)(x_i) - x_i \|^2$

b) BCE $\rightarrow \sum_k x_{ik} \log(\hat{x_{ik}}) + (1 - x_{ik}) \log(1 - \hat{x_{ik}})$

2. The loss function for a denoising autoencoder

$$\sum_i \| (w' \circ w)(x_i + \varepsilon) - x_i \|^2$$

We use denoising autoencoders to prevent the autoencoder from learning the identity function when we have more nodes than inputs

The idea is that we set some of the inputs to zero, so that we have corrupted inputs

3. Less than the input / output size

4. It is as long as the hidden space is continuous. Hence, if this is the case, if two samples of the hidden space are close, their decoded versions will be close

4. ADVERSARIAL EXAMPLES

1. An adversarial example is a generated copy from the training set with the intention of fooling a neural network, hence, they include slight modifications

2. Humans are not able to distinguish them

3.