MLSM 2020 - Quiz 4
26/10/2020 - 04/11/2020 at 23:59

Questions based on Lecture 6 and 7

(1) (1.0 pt.)

Let $\mathbf{K}_1$ and $\mathbf{K}_2$ be two kernel matrices. They are positive semi-definite and symmetric, with the same size. Which of these three operations does not provide a correct kernel matrix in the general case?

Be aware, the one false statement needs to be selected! Think about the differences between the expressions of the elements of the product matrices.

(1) The linear combination of $\mathbf{K}_1$ and $\mathbf{K}_2$ with non-negative real weights.
(2) The itemwise (pointwise, Hadamard) product of $\mathbf{K}_1$ and $\mathbf{K}_2$.
(3) *** The matrix product of $\mathbf{K}_1$ and $\mathbf{K}_2$.

Answer: The first two cases can be seen on Slide "Several ways to get to a kernel" in Lecture 6.

In the third case, the product of the symmetric positive semi-definite matrices could be not a symmetric matrix. An example:

$$\mathbf{K}_1 = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}, \quad \mathbf{K}_2 = \begin{bmatrix} 1 & 1 \\ 1 & 3 \end{bmatrix}, \; \mathbf{K}_1\mathbf{K}_2 = \begin{bmatrix} 3 & 5 \\ 3 & 7 \end{bmatrix} \neq \mathbf{K}_2\mathbf{K}_1 = \begin{bmatrix} 3 & 3 \\ 5 & 7 \end{bmatrix}.$$

The matrix product depends on the order of the matrices, this operation is not commutative.

(2) (1.0 pt.)

A polynomial kernel has the form $\kappa_{pol}(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T\mathbf{z} + c)^q$. Let $\kappa(\mathbf{x}, \mathbf{z})$ be any valid kernel. Let the kernel $\kappa(\mathbf{x}, \mathbf{z})$ substitute the $\mathbf{x}^T\mathbf{z}$ term in the polynomial kernel, namely we have $\kappa_{pol\_mod}(\mathbf{x}, \mathbf{z}) = (\kappa(\mathbf{x}, \mathbf{z}) + c)^q$. Which of these statements is true?

Hint: How is the kernel function defined in the corresponding Hilbert space? How can it be represented?

(1) *** κ_{pol_mod} is a valid, positive semi-definite kernel for any κ.
(2) There are cases where κ_{pol_mod} is not positive semi-definite.
(3) κ_{pol_mod} is only a valid kernel if κ is a Gaussian one.

Answer: Firstly, a polynomial kernel is valid kernel if it is defined on two vectors of a Hilbert space. Secondly, if a kernel is valid then we have the representation in a Reproducing Kernel Hilbert space: $\kappa(\mathbf{x}, \mathbf{z}) = (\phi(\mathbf{x}), \phi(\mathbf{z}))$, see the slides "Hilbert space*" and "The kernel matrix" in Lecture 6, where $\tilde{\mathbf{x}} = \phi(\mathbf{x})$ and $\tilde{\mathbf{z}} = \phi(\mathbf{z})$ are vectors of the corresponding Hilbert space. Therefore substituting those vectors into a polynomial kernel provides a valid kernel. Thus we can write $(\tilde{\mathbf{x}}^T\tilde{\mathbf{z}} + c)^q$ which is a well defined polynomial kernel.

(3) (1.0 pt.) Let the feature vector of a polynomial kernel, $\kappa_{pol}(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T\mathbf{z} + c)^q$, be represented explicitly. What is the dimension of the explicit feature vector? Assume that the degree of the polynomial is 3, and the polynomial kernel is defined on the vector space of dimension 2.

Hint: See the representation of the polynomial kernel in Lecture 6.

(1) 6
(2) *** 10
(3) 15

Answer: Let us start at Slide "Example: Polynomial kernel on 2D inputs" in Lecture 6. On that slide, the degree 2 case is shown. From that case we can got to the more general situation. We have that

$(\mathbf{x}^T\mathbf{z}+c) = ([\sqrt{c},\mathbf{x}],[\sqrt{c},\mathbf{z}])$. For dimension 2, the component wise form is $([\sqrt{c},x_1,x_2],[\sqrt{c},z_1,z_2])$. Let $\sqrt{(}c)$ be indexed by 0, and we have $([x_0,x_1,x_2],[z_0,z_1,z_2])$, where $x_0 = z_0 = \sqrt{c}$. Now we can write by the help of the known formula of the multinomial coefficients

(1) $$([x_0,x_1,x_2],[z_0,z_1,z_2]) = (\sum_{i=0}^{2} x_i z_i)^3 = \sum_{k_1+k_2+k_3=3} \binom{3!}{k_1!k_2!k_3!} \prod_{i=0}^{2}(x_i z_i)^{k_i}.$$

The description of this formula can be found, for example "https://dlmf.nist.gov/26.4" on the web site of the National Institute of Standards and Technology of the US Government,

or "https://en.wikipedia.org/wiki/Multinomial_theorem", and several other combinatorial reference sites.

Starting on (1) we can decompose the product

$$\begin{aligned}
&\sum_{k_1+k_2+k_3=3} \binom{3!}{k_1!k_2!k_3!} \prod_{i=0}^{2}(x_i z_i)^{k_i} = \sum_{k_1+k_2+k_3=3} \binom{3!}{k_1!k_2!k_3!} \prod_{i=0}^{2}(x_i)^{k_i} \prod_{i=0}^{2}(z_i)^{k_i} \\
&= \left(\sum_{k_1+k_2+k_3=3} \underbrace{\left(\sqrt{\binom{3!}{k_1!k_2!k_3!}} \prod_{i=0}^{2}(x_i)^{k_i}\right)}_{\phi(\mathbf{x})_{k_1,k_2,k_3}} \underbrace{\left(\sqrt{\binom{3!}{k_1!k_2!k_3!}} \prod_{i=0}^{2}(z_i)^{k_i}\right)}_{\phi(\mathbf{z})_{k_1,k_2,k_3}} \right) \\
&= (\phi(\mathbf{x}),\phi(\mathbf{z})).
\end{aligned}$$

Hence, the feature vector $\phi(\mathbf{x})$ has as many components as the number of different sums of $k_1,k_2,k_3 = 0,1,2$ when they satisfy the constraint $k_1+k_2+k_3 = 3$, That number is $\binom{3+2}{2}$ which gives 10.

Following the same line, the general case when the dimension of the vector space is N, the number of components in the polynomial feature vector for degree d is equal to $\binom{N+d}{d}$. You can also find a similar description in the lecture book "Understanding Machine Learning: From Theory to Algorithms", Section 16.2, Page 220.

(4) (2.0 pt.)

Implement a regression problem via the backpropagation algorithm of the neural network based learner, Lecture 7, Slides about the "Backpropagation training algorithm". Both the input and the output are one dimensional real numbers. The network contains an input layer with one node, a hidden layer with $H = 5$ nodes, and on output layer with one node. The activation function in the hidden layer is the sigmoid function, see on the corresponding slides. No bias is considered when the weights are applied.

The data description, and the learning parameters for the neural network are given by the following python code.

```
import numpy as np

  ## Load the data
  m0=100                          ## m0+1 gives the index of the 0 input
  m=2*m0+1                        ## number of points
  xx=2*np.arange(m+1)/m-1         ## input data is in the range [-1,+1], with steps 0.01
  rr=xx**2                        ## output values: a parabola

  ## set the learner parameters
  niteration=100                  ## number of iterations
  H=5                             ## number of nodes in the hidden layer
  eta=0.3                         ## learning speed, step size

  ## Random initialization of the edge weights
  ## Set the seed to a fixed number
```

```
  np.random.seed(12345)
  ## weights between the input and the hidden layers
  W=2*(np.random.rand(H)-0.5)/100          ## random uniform in [-0.01,0.01]
  ## weights between the hidden and the output layers
  V=2*(np.random.rand(H)-0.5)/100          ## random uniform in [-0.01,0.01]
```

Be aware, the initialization of the weights needs to follow the scheme shown above to receive the proper result. The data points are processed sequentially from -1 to 1 without randomization.

Assume that the vector yy of size m contains the predicted values for all input examples. Which interval contains the error after $niteration = 100$, at the input value 0, when the error is computed in this way $rr[m0+1] - yy[m0+1]$?

(1) $[+0.2, +0.3]$

(2) *** $[-0.1, +0.1]$

(3) $[-0.3, -0.2]$

Answer: The algorithm can be built on the pseudo-code of the Slide "Backpropagation training algorithm for two-layer MLP for regression" of Lecture 7. In this question the $\mathbf{w}$ and $\mathbf{v}$ are vectors instead matrices, since both the input and the output are one dimensional.

The complete code could be similar to this one:

```
import numpy as np

## ################################
def main():

  ## Load the data
  m0=100                              ## m0+1 gives the index of the 0 input
  m=2*m0+1                            ## number of points
  xx=2*np.arange(m+1)/m-1             ## input data is in the range [-1,+1], with steps 0.01
  rr=xx**2                            ## output: parabola

  ## set the learner parameters
  niteration=100                      ## number of iteations
  H=5                                 ## number of nodes in the hidden layer
  eta=0.3                             ## learning speed, step size

  ## Random initialization of the edge weights
  ## Set the seed to a fixed number
  np.random.seed(12345)
  ## weights between the input and the hidden layers
  W=2*(np.random.rand(H)-0.5)/100     ## random uniform in [-0.01,0.01]
  ## weights between the hidden and the output layers
  V=2*(np.random.rand(H)-0.5)/100     ## random uniform in [-0.01,0.01]

  ## predicted output
  yy=np.zeros(m+1)

  irandom=0    ## =1 random sample, =0 iteration on the input
```

```
  for t in range(niteration):
    xi=np.arange(m) ## indexes of the points
    if irandom==1:
      ## process examples in a random order
      np.random.shuffle(xi)

    for ix in range(m):
      ## draw a training example
      i=xi[ix]   ## read the indexes
      x=xx[i]    ## input
      r=rr[i]    ## output
      ## forward propagation
      zlin=x*W
      ## apply sigmoid activation function
      z=1/(1+np.exp(-zlin))
      ## compute prediction       )
      y=np.dot(V,z)
      ## store the prediction
      yy[i]=y
      ## Backpropagation
      ## change of the weights between the hidden and the output layers
      dV=eta*(r-y)*z
      ## change of the weights between the input and the hidden layers
      dW=eta*(r-y)*V*z*(1-z)*x
      ## update the weights
      V=V+dV
      W=W+dW

  print('Error, r-y, at x=0:',rr[m0+1]-yy[m0+1])

## ######################################################
## ######################################################################
if __name__ == "__main__":
  main()

## ######################################################
```