Team 15, Lab : 5

Faiz Ahmed	Mithun Das	Mutasim Fuad Ansari	Mohammad Abir Reza
1152231	1151651	1152109	1151705
stu225473@mail.uni-kiel.de	stu225039@mail.uni-kiel.de	stu225365@mail.uni-kiel.de	stu225093@mail.uni-kiel.de

Exercise 1 (Loss functions and weight update formulae, theoretical considerations):

a) Show that for a single-layer perceptron with a linear activation function, the weights can be determined in closed-form from the training data, assuming a mean-squared error loss.

Answer

Please find the answer below.

b) The cross-entropy loss for a single-layer perceptron on a training set $D = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(p)}, y^{(p)})\}$ is defined as

$$L_{CE}(w, y, \hat{y}) = -\frac{1}{p} \sum_{\mu=1}^{p} [y^{(\mu)} \ln \hat{y}(w, x^{(\mu)}) + (1 - y^{(\mu)}) \ln(1 - \hat{y}(w, x^{(\mu)}))]$$

where $\hat{y} = \hat{y}(w, x^{(\mu)})$ is the output of the perceptron. Show that this expression is non-negative for $y, \hat{y} \in [0, 1]$ and assumes a minimum (as a function of the perceptron output $\hat{y}$) if $\hat{y}(w, x^{(\mu)}) = y^{(\mu)}$ for all μ, i.e. if the neuron output $\hat{y}(w, x^{(\mu)})$ corresponds to the target value $y^{(\mu)}$ for each training sample μ.

Answer

Please find the answer below.

Exercise 2 (Convolutional neural networks):

a) Explain the following terms related to learning in neural networks:

- Convolutional neural network
- Filter, Kernel
- Feature map
- Receptive field
- Pooling (subsampling) layer
- Fully convolutional network

Answer

Convolutional neural network:

A convolutional neural network is a process of deep neural network.It is often use to analyze visual image. It can be said that it is use as CNN or ConvNet. multilayer perceptrons has different versions. CNNs are called to be the regularized versions of multilayer perceptrons. This perceptrons usually occurs in one layer and all the other layer get conneted with it via network. And thus with the help of “fully-connectedness” process of these networks make them overfitting data. However, a different way towards regularization usually taken by CNNs: because of the hierarchical patten in data and assemble more complex patters they use this method to get benifited by comparing this patterns. This is one of the big reason that CNNs are more ahead than other in lower extreme to be exact.

Filter, Kernel:

A Filter or kernel known as component of CNN but not defined. But this is a part of layered architecture from CNN.When it comes to a practical example in Convolutional neural network a kernel is a smaller-sized matrix in comparison to the input dimensions of the image. ‘activation maps’ is called the input volume to obtain. The activated regions shows where features get to reconige the input. When the training set execute the learning process change the kernel matrix and show the result that indicate the correct region or the nearest region.

Feature Map:

A feature map is a complete output of a particular kernel to an proper input. With the given parameter such as input size of the image or the type of filter used in the image ot the padding giving around the input shows the whole area of the feature map. In the output image, it shows variety of places with feature map in the input image. Every filter in the input images redirect to its completely uniqe feature maps in the output image.Getting several features in the input image is possible after several filter in the same input image.

Receptive field:

A receptive field is basically a connectivity of the local region of the input volume that a connected with each neuron that get as an input of huge or big size images.At the time of using a huge size images as input that has high dimensions in it,it is not possible to connect each neuron with all neurons in the network.

Pooling:

A Pooling is a layer that adds the activations of large number of sections in a single section of the current layer. Pooling layer occurs in every feature map unconventionaly. Maximum activation of every units in the receptive field in Convolutional neural network that commonly used are max pooling that uses maximum activation of all units.

Fully convolutional network:

A fully convolutional network is a kind of network that connects without any fully connected layers hence all teh layers in the network are convolution layer. Difference between other network with fully convolutional network is that it fully connected layers is just as end to end learable as a fully convolutional network.It tries to make a proper decision on the local input it got in the networks.

b) (CNN on MNIST) The Jupyter notebook provides code to apply a CNN to the MNIST classification problem. Complete the script – choosing a suitable network architecture and providing values for the hyperparamters – and report the accuracy of the model. Try to improve the accuracy of the CNN (and / or to reduce the number of model parameters) by modifying the model structure and / or the values of the hyperparameters and settings. Visualize and discuss the model structure. Compare your results to the results of MNIST classification using a multilayer perceptron (lab sheet 4, exercise 3).

Note: The model can be visualized using the *plot_model* method of *tensorflow.keras.utils*:
plot_model(model, to_file='model.png', show_shapes=False, show_layer_names=True)
This should write a visualization of the model to the file *model.png*. However, this may not work in Colab. In this case, you may execute the method locally (without training).

In [ ]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Dropout, Flatt
from tensorflow.keras import Sequential
from tensorflow.keras.optimizers import SGD, Adam, Adadelta, Adagrad, Nadam, RM
from tensorflow.keras.utils import to_categorical
import tensorflow.keras.datasets as tfds
import tensorflow.keras.initializers as tfi
import tensorflow.keras.regularizers as tfr
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import backend as K
import os
import datetime
from tensorflow.keras.utils import plot_model
import sys

###--------
# load data
###--------

(training_input, training_target), (test_input, test_target)  = tfds.mnist.load
training_input = training_input.astype('float32')
test_input = test_input.astype('float32')

# plot some sample images
num_examples = 3
for s in range(num_examples):
  print("Example image, true label: %d" % training_target[s])
  plt.imshow(training_input[s], vmin=0, vmax=255, cmap=plt.cm.gray)
  plt.show()

# convert class vectors to binary class matrices
num_classes = 10 # MNIST: 10 digits    # FIX!!!
training_target = to_categorical(training_target, num_classes)
test_target = to_categorical(test_target, num_classes)

# input image dimensions
img_rows, img_cols = 28, 28

if K.image_data_format() == 'channels_first':
    training_input = training_input.reshape(training_input.shape[0], 1, img_row
    test_input = test_input.reshape(test_input.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    training_input = training_input.reshape(training_input.shape[0], img_rows,
    test_input = test_input.reshape(test_input.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)

# Reserve 10,000 samples for validation
validation_input = training_input[-10000:]
validation_target = training_target[-10000:]
training_input = training_input[:-10000]
training_target = training_target[:-10000]

print("training input shape: %s, training target shape: %s"  % (training_input.
print("validation input shape: %s, validation target shape: %s"  % (validation_
print("test input shape: %s, test target shape: %s"  % (test_input.shape, test_
# range of input values: 0 ... 255
print("\n")


###-----------
# process data
###-----------

# Note: shuffling is performed in fit method
```

Answer

Result for SGD

Model configuration:

num_feature_maps_per_layer = [4] # FIX!!!

num_hidden_neurons_fully_connected = [100] # FIX!!!

kernel_size = 3 # FIX!!!

pool_size = 2 # FIX!!!

padding = 'same'

number of layers: 12

final training loss: 1.490065

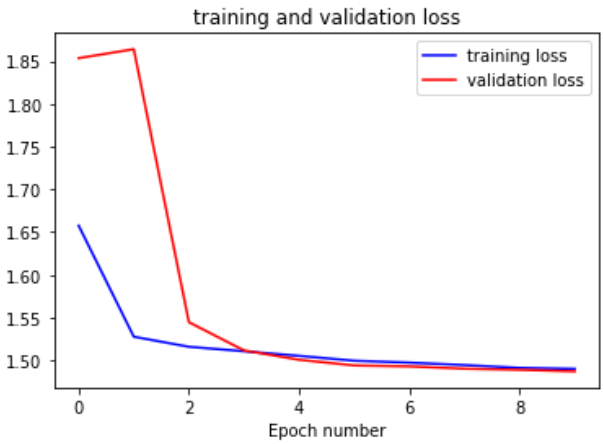
final training accuracy: 0.980180

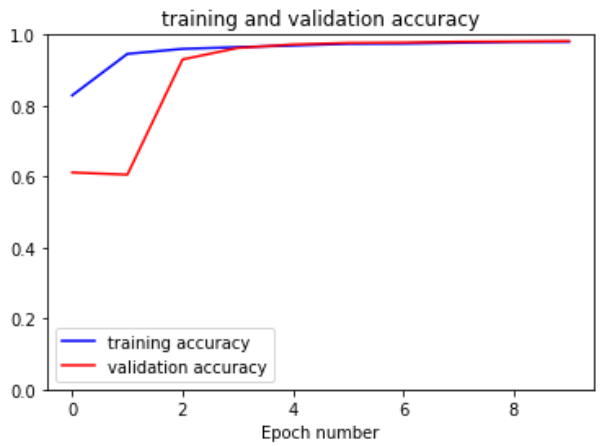
final validation loss: 1.486805

final validation accuracy: 0.981600

final test loss: 1.487094

final test accuracy: 0.982000





Findings

Changing batch size and number of features doesnt changes that much but after changing the kernel size

Exercise 3 (Number of parameters in a fully connected and a convolutional network):

a) For a fully connected multi-layer perceptron containing two hidden layers with 100 hidden units each which is designed for the MNIST classification problem, calculate the number of learnable parameters (i.e. parameters which are learned using the backpropagation algorithm).

Answer

For a fully connected multilayer perceptron containing 2 hidden layers, each hidden layer contains 100 hidden units, and the output layer with 10 possible output,the number of learnable parametres in this case the learnable waits will be as follows

let X be the size of input. Thus

Number of learbable parameters in input layer to 1st hidden layer = number of weights + number of bias

```
=(X*100) + 100
= 100X+100
```

Number of learbable parameters in 1st hidden layer to 2nd hidden layer = number of weights + number of bias

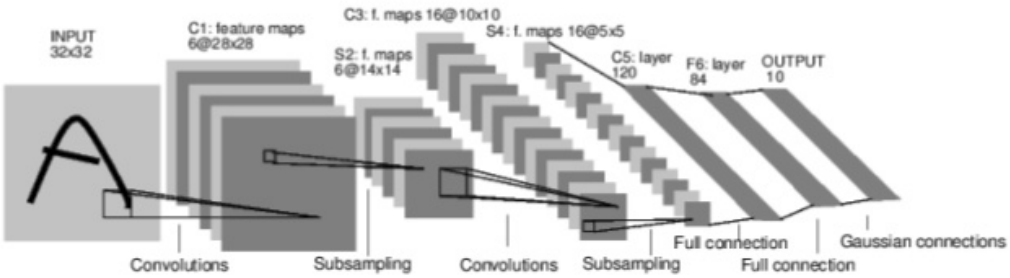
```
=(100*100) + 100
= 10100
```

Number of learbable parameters in 1st hidden layer to 2nd hidden layer = number of weights + number of bias

```
=(100*10) + 10
= 1010
```

The number of total learnable parameters = 100X+100+10100+1100 = 100X+11210

b) The figure shows the architecture of the famous LeNet-5 convolutional network. Calculate the number of trainable parameters and the number of connections (synaptic weights plus biases, i.e. in augmented space). Cx: Convolution layer x, Sx: Subsampling layer x, Fx: Fully connected layer x



Answer

Number of trainable parameters and connections

1st convolutional layer (C1)

In this layer the input image is turned into 6 feature maps each having the dimension (5*5).

Number of trainable parametres = number of weights + number of bias

$$= (5*5*6) + 6 = 156$$

Number of connections = 28 * 28 * 156 = 122304

2nd sub-sampling layer (S2)

In this layer sub-sampling is applied and the filter is modified into 6 layers with dimension (2*2)

*Number of trainable parameters = (number of coeffecient + number of bias) * number of filters)*

$$= (1+1)*6 = 12$$

Number of connections = 14 * 14 * 30 = 5880

3rd convolution layer (C3)

In this layer there are 16 feature maps with dimension (5*5) among which only 10 is connected with the previous layer.

Number of trainable parameter = number of weights + number of bias

$$= (5*5*6*10) + 16 = 1516$$

Number of connections = 10 * 10 * 1516 = 151600

4th sub-sampling layer (S4)

In this layer sub-sampling is applied and the filter is modified into 16 layers with dimension (2*2)

*Number of trainable parameters = (number of coeffecient + number of bias) * number of filters)*

$$= (1+1)*16 = 32$$

5th convolution layer (C5)

The is a fully connected convolutional layer with 120 feature maps with dimension 1×1. Each of the 120 units in this layer is connected to all the 400 nodes (5x5x16) in the fourth layer S4.

Number of trainable parameter = number of weights + number of bias

$$= (5*5*16*120) + 120 = 48120$$

Number of connections = 5 * 5 * 80 = 2000

6th layer (F6)

This is a fully connected layer with 84 units.

Number of trainable parameter = number of weights + number of bias

$$= (84*120) + 84 = 10164$$

Output layer

This layer produces 10 possible output.

Number of trainable parameter = number of weights + number of bias

c) (optional) Calculate the output dimensions and number of parameters of the AlexNet without grouping (the division into two separate paths), i.e. when the AlexNet is realized in a single path.

Answer

Write your answer here.

d) (optional) How do these numbers change in the ZF net?

Answer

Write your answer here.

Exercise 4 (Overfitting - CNN on CIFAR-10):

The jupyter notebook provides code to train a network to classify images of the CIFAR-10 dataset (https://www.cs.toronto.edu/~kriz/cifar.html) (https://www.cs.toronto.edu/~kriz/cifar.html)). Training is performed on a subset of 10000 images from the dataset. Run the code and discuss the results. Then, explore possibilities to counteract the observed behavior.

Remarks:

1) If you perform the experiments in Colab, you may explicitly select GPU execution: Select the “Edit” tab in Colab, then choose “Notebook settings”, and then select “GPU” as hardware accelerator. See also the accompanying documentation in Colab.

2) For Tensorboard to work, you have to explicitly activate cookies from all third parties in your browser. Furthermore, it has been found that Tensorboard could not be run successfully in all configuration or browsers. If it doesn’t work in your configuration (e.g. error “Google 403. That’s an error. That’s all we know.”) please outcomment the lines referring to Tensorboard.

3) Experiments with data augmentation may substantially increase runtime! Further information on data augmentation in Keras can be found at https://keras.io/preprocessing/image/ (https://keras.io/preprocessing/image/).

In [ ]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropo
from tensorflow.keras import Sequential
from tensorflow.keras.optimizers import SGD, Adam, Adadelta, Adagrad, Nadam, RM
from tensorflow.keras.datasets import cifar10
import tensorflow.keras.initializers as tfi
import tensorflow.keras.regularizers as tfr
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
from sklearn.utils import shuffle
import os
import datetime
import sys

###--------
# load data
###--------

(training_images, training_target), (test_images, test_target) = cifar10.load_d
training_target = np.reshape(training_target, training_target.shape[0])
test_target = np.reshape(test_target, test_target.shape[0])

# training_images: shape (50000, 32, 32, 3)
# training_target: shape (50000, 1)
# test_images:  shape (10000, 32, 32, 3)
# test_target:  shape (10000, 1)

# shuffle data
training_images, training_target = shuffle(training_images, training_target)
test_images, test_target = shuffle(test_images, test_target)

# plot some sample images
num_examples = 3
for s in range(num_examples):
  print("Example image, true label: %d" % training_target[s])
  plt.imshow(training_images[s])
  plt.show()

num_classes = 10 # CIFAR-10 has 10 classes

print("min. training data: %f" % np.min(training_images))
print("max. training data: %f" % np.max(training_images))
print("min. test data: %f" % np.min(test_images))
print("max. test data: %f" % np.max(test_images))

# scaling inputs from range 0 ... 255 to range [0,1] if desired
min_val = np.min(training_images) # 0
max_val = np.max(training_images) # 255
training_images = training_images.astype('float32')
test_images = test_images.astype('float32')
training_input = (training_images - min_val) / (max_val - min_val)
test_input = (test_images) / (max_val - min_val)

# use only part of training set
numTrainingExamplesUsed = 10000 # FIX!!!
print("USING ONLY %d TRAINING EXAMPLES" % numTrainingExamplesUsed)
training_input = training_input[:numTrainingExamplesUsed]
training_target = training_target[:numTrainingExamplesUsed]

# histograms of input values
nBins = 100
fig, axes = plt.subplots(1, 2, figsize=(15,10))
axes[0].hist(training_input.flatten(), nBins)
axes[0].set_xlabel("training")
axes[0].set_ylabel("counts")
axes[0].set_ylim((0.0.3*1e7))
```

As before, you can visualize your different runs via Tensorboard:

In [ ]:
```
%load_ext tensorboard
%tensorboard --logdir {logs_base_dir}
```

If you want to download the created logs you can create a zip file with the following command. Remember that any saved data in Colab will be lost after a reboot.

In [ ]:
```
# create zip from logs
!zip -r ./logs.zip ./logs
```

Answer

Some common properties

activation = 'elu'

num_epochs = 125 # FIX !!!

batch_size = 64 # FIX !!!

batch_normalization = True #

model : #1

dropout = 0

data_augmentation = False

Result

final training loss: 0.000000

final training accuracy: 1.000000

final test loss: 3.196528

final test accuracy: 0.716400

model : #2

dropout = 0.5

data_augmentation = False

Result

final training loss: 0.000000

final training accuracy: 1.000000

final test loss: 3.224531

final test accuracy: 0.714800

model : #3

dropout = 0.5

stddev = 0.2

model.add(Dropout(0.2 + i/10.0))

data_augmentation = False # FIX !!!

Result

final training loss: 0.036017

final training accuracy: 0.986900

final test loss: 1.638021

final test accuracy: 0.776500

Exercise 1 (Loss functions and weight update formulae, theoretical considerations):

a) Show that for a single-layer perceptron with a linear activation function, the weights can be determined in closed-form from the training data, assuming a mean-squared error loss.

Answer :

By definition of Mean- Squared Error (MSE) , we know ,

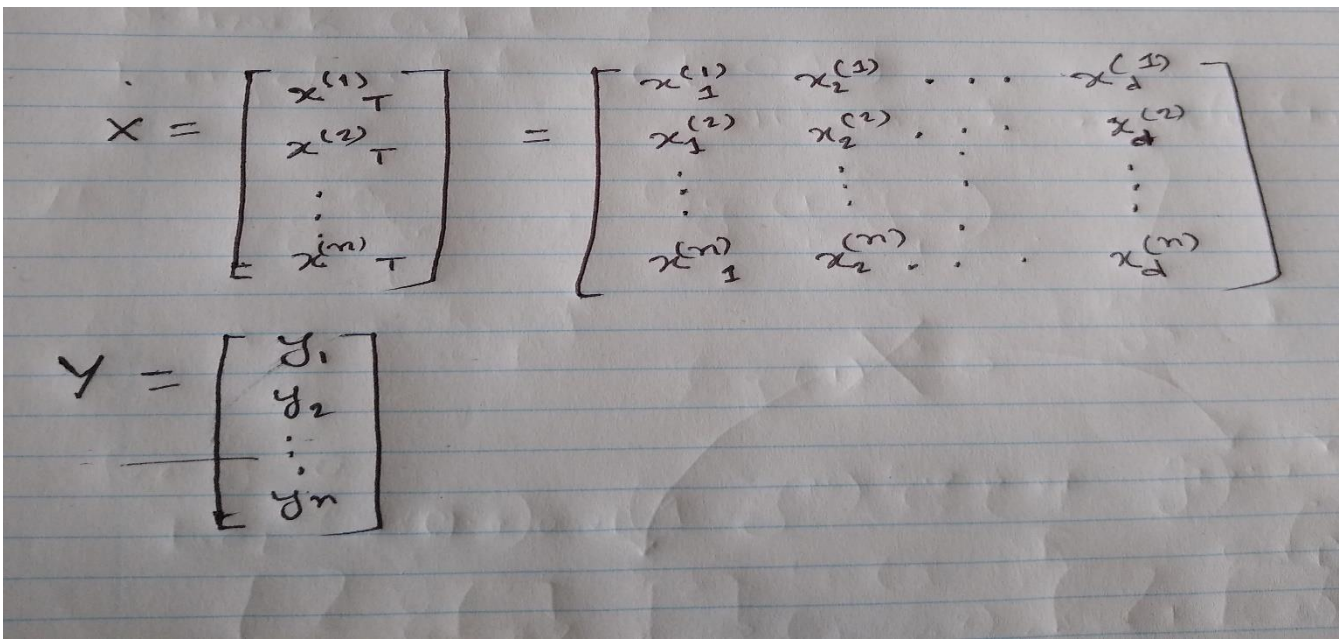
$L_{MSE}\ (w, y, \hat{y}) = \frac{1}{2p}\ \sum_{\mu=1}^{p}(\hat{y}\ (w, x^{(\mu)}) -\ y^{(\mu)})^2$

Lets assume we habe training data ,

D = $\{(x^1, y^1)\ , (x^2, y^2)\ , \ldots\ldots , (x^n, y^n)\ \}$

Here every input is a vector and for each input vector we get a scalar output y.

Now if we convert this notation in matrix form,



Lets assume x is a input vector and w is a weight matrix.

Now, perceptron with linear activation function in augmented notation : $\hat{y}$(x) = w^T.x = x^T.w

Thus we get, $L_{MSE}\ (w, y, \hat{y}) = \frac{1}{2p}\ (y - x.w)^T$. (y – x.w)

Let’s minimize the mean squared error loss :

w^* = arg min_w L(w,y, $\hat{y}$) = arg min_w $\frac{1}{2p}$ $(y - x.w)^T$. (y – x.w)

Therefore ,

$\frac{d\, L_{MSE}\,(w,y,\hat{y})}{d\, w^*}$ = -2 $(y - x.w^*)^T$ x = 0

=> $(y - x.w^*)^T$ x = 0

=> $y^T.x - (x.w^*)^T$ x = 0

=> $(x.w^*)^T$ x = $y^T.x$

=> x^T $(x.w^*)$ = x^T y

=> w^* = x^T y $(x^T.x)^{-1}$

Assume that $(x^T.x)^{-1}$ exists.

This is the closed form solution for weights,which is measured by training data. As, we are using linear activation function and mean – squared loss function , there must exists a closed form of solution. So, we can say that , for a single layer perceptron with a linear activation function, the weights can be determined in closed form from the training data, assuming mean-squared error loss function.■

1.(b) Given Training set is, D = { $(x^{(1)}, y^{(1)})$, $(x^{(2)}, y^{(2)})$, … … …, $(x^{(p)}, y^{(p)})$ } .

By definition of cross entropy loss function ,

$$L_{CE}(\boldsymbol{w}, y, \hat{y}) = -\frac{1}{p}\sum_{\mu=1}^{p} [\ \ y^{\mu}\ \ln \hat{y}\,(\boldsymbol{w}, \boldsymbol{x}^{\mu})\ +\ (1 - y^{\mu})\ \ln(1 - \hat{y}\,(\boldsymbol{w}, \boldsymbol{x}^{\mu})\,)]$$

Where, $\hat{y} = \hat{y}(\boldsymbol{w}, \boldsymbol{x}^{\mu})$ is the output of the perceptron. We have to show that, cross entropy is non negative for all $y, \hat{y} \in [0,1]$.

So, for each sample the cross entropy is ,

$L_{CE}(\boldsymbol{w}, y, \hat{y}) = -[y^{\mu}\ \ln \hat{y}\,(\boldsymbol{w}, \boldsymbol{x}^{\mu}) +\ (1 - y^{\mu})\ \ln(1 - \hat{y}\,(\boldsymbol{w}, \boldsymbol{x}^{\mu}))\,]$

For $x\ \in [0,1]$, ln(x) gives the value 0 or less than 0.

Thus we have, $y^{\mu}\ \ln \hat{y}\,(\boldsymbol{w}, \boldsymbol{x}^{\mu}) \leq 0$ for all $y, \hat{y} \in [0,1]$

Similarly, $(1 - y^{\mu})\ \ln(1 - \hat{y}\,(\boldsymbol{w}, \boldsymbol{x}^{\mu}))\ \leq 0$ for all $y, \hat{y} \in [0,1]$

$Therefore,\ L_{CE}(\boldsymbol{w}, y, \hat{y}) = -[y^{\mu}\ \ln \hat{y}\,(\boldsymbol{w}, \boldsymbol{x}^{\mu}) +\ (1 - y^{\mu})\ \ln(1 - \hat{y}\,(\boldsymbol{w}, \boldsymbol{x}^{\mu}))\,] \geq 0$

So, cross entropy loss is non negative for each sample and cross entropy loss average for all training set is also non-negative.■