Machine Learning

Lab 3: Neural Network

Dr. Shuang LIANG

Basic lab instructions

- You may want to *bring your slides* to labs to look up syntax and examples.
- Have a question? *Ask a TA* for help, or look at the lecture slides.
- We encourage you to *talk to your classmates*; it's okay to share code and ideas during lab.
- Make good use of these tools: *search engines, API documentation* and *IDEs*.
- For the *basic sections*, you need to complete them and submit the results as required. Please submit *real experimental data*.
- For the *extension sections*, you don't have to finish all of the exercises. Just do as much as you can in the allotted time. You don't need to finish the rest after you leave the lab; there is no homework from lab.

Today's lab exercises

- Use the Python neural network library Keras to accomplish the following tasks:
  - Build a simple neural network and draw the network architecture diagram (20mins)
  - Use this network to complete the image classification on the *mnist* dataset and obtain the classification results(5mins)
  - Adjust network and training parameters to try to improve classification results (20mins)
  - Visual Training Curve (20mins)
  - Build VGG 16 network for image classification on Cifar10 dataset (*optional* 25mins)

Experimental data

- In the experiment, the *mnist* data set provided by Keras library was used. The training set consisted of 60,000 images and labels, and the test set consisted of 10,000 images and labels. The picture is a handwritten digital picture of *0 ~ 9* in gray matter with *28 × 28* pixels.
- The data set has been divided into training data (X_train, y_train) and validation data (X_test, y_test), and has been set as global variables. The data has undergone a series of preprocessing, such as the steps to turn labels into *one-hot* vectors.

Experiment environment

- Python 3.x
- Libraries: tensorflow, keras, numpy, matplotlib
- Note: `tensorflow<2.11` if you want to use GPU on windows
- Docs
  - Keras： https://keras.io/
  - Numpy： https://numpy.org/doc/stable/
  - Matplotlib： https://matplotlib.org/stable/tutorials/index.html
- Experiments 1 to 5 are run on the CPU, and experiment 6 can be run on the GPU if conditions permit

Experiment 1

- **Task1： Build Network**
- Build your network in the following sections of the code

```
'''第二步：构建网络层'''
# 在此处构建你的网络
#################################################################################################
#################################################################################################
```

- Read the documentation or consult the resources to see how the network layer, activation functions, etc. in keras should be used.

Experiment 1

- A simple networking example that includes only the full connection layer

```
'''第二步：构建网络层'''
# 在此处构建你的网络
#################################################################################################
model.add(Input((28,28,1)))

model.add(Flatten())

# 1000个神经元的全连接层
model.add(Dense(500))
model.add(Activation('tanh'))

model.add(Dense(500))
model.add(Activation('tanh'))

model.add(Dense(10)) # 输出结果是10个类别，所以维度是10
model.add(Activation('softmax')) # 使用softmax转换为概率
#################################################################################################
```

Experiment 1

- **Note:**
- To complete the classification, the *last two lines* of network code are fixed (i.e. a fully connected layer of 10 neurons with a softmax function).
- X_train and X_test can be entered directly into the convolution layer, but only after *flatten* can the fully connected layer be entered.
- Do not leave out the *Input* layer (it may not affect training, but it will affect the next step of the network structure diagram output)

Experiment 1

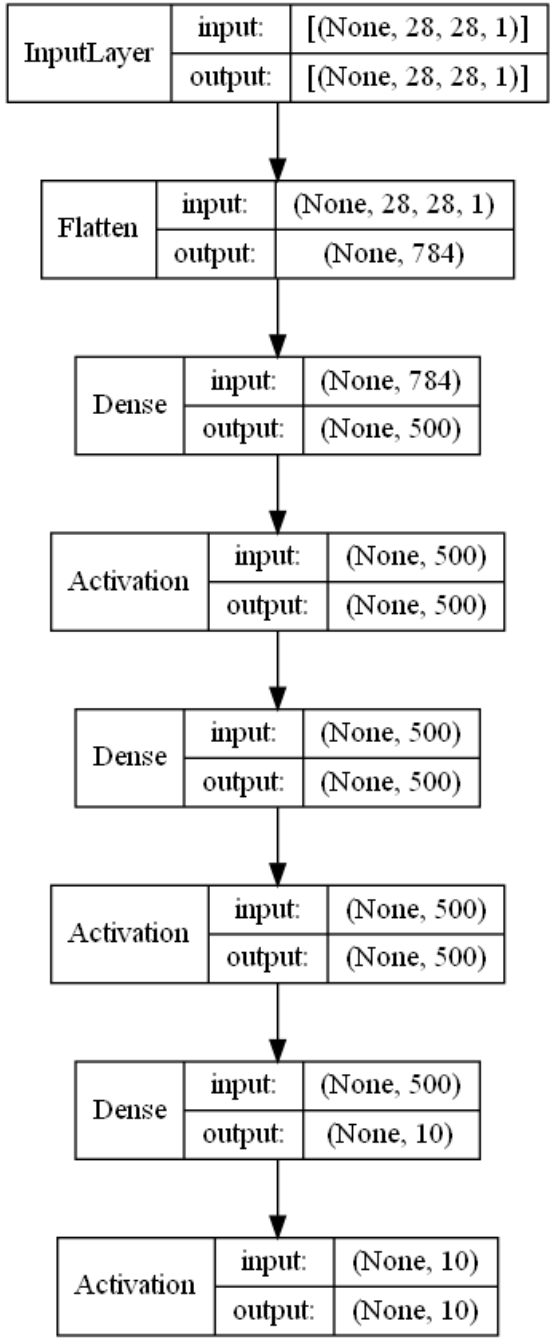
- **Task2: Draw the network architecture diagram**
- The following code outputs a diagram of the network architecture in png format

```
# 在此处输出网络的架构。此处参数可以不用调整。
# model表示自定义的模型名 to_file表示存储的文件名 show_shapes是否显示形状  rankdir表示方向T(top)B(Bottow)
from keras.utils.vis_utils import plot_model
plot_model(model,to_file='model.png',show_shapes=True,show_layer_names=False,rankdir='TB')
```

- To implement this function, you need to install *pydot-ng*(or *pydotplus*)and *graphviz*. The former is installed via *pip* and the latter needs to be downloaded at Download | Graphviz. When installing graphviz, be careful to select the option to add paths to environment variables.

Experiment 1

- Example: Network architecture diagram of the aforementioned network



Experiment 2

- **Task: Train the network and get classification accuracy on the test set**
- After the network model is built, if there is no error, you can directly run the code to start training. Without changing the output settings, you should have the following output format:

```
Epoch 1/50
329/329 - 3s - loss: 0.4510 - accuracy: 0.8700 - val_loss: 0.3164 - val_accuracy: 0.9096
Epoch 2/50
329/329 - 2s - loss: 0.2832 - accuracy: 0.9181 - val_loss: 0.2692 - val_accuracy: 0.9234
Epoch 3/50
329/329 - 2s - loss: 0.2443 - accuracy: 0.9296 - val_loss: 0.2482 - val_accuracy: 0.9277
Epoch 4/50
```

- After the training is completed, the test will be carried out and the accuracy rate will be output in the format of:

```
test set
79/79 [==============================] - 0s 4ms/step - loss: 0.0767 - accuracy: 0.9772
The test loss is 0.076728
The accuracy of the model is 0.977200
```

Experiment 3

- **Task: Adjust network and training settings for higher classification accuracy**
- It can be adjusted according to the following ideas
- Network:
  - Add network depth: Add some layers
  - Use convolution layers, BatchNorm, etc
  - Regularization methods such as Dropout are used to mitigate overfitting
  - Replace activation functions: ReLu, etc
  - Parameter adjustment of convolutional and fully connected layers: different number/size of convolutional nuclei, different number of neurons , etc
- Training setup:
  - Optimizer: SGD, Adam, etc
  - Loss function: cross entropy, MSE, etc
  - Learning rate: initial value, decay value
  - Number of Epochs
  - Batchsize

Experiment 3

- **Tip:**
- Do not modify the *random number seed*, otherwise the result may fluctuate greatly
- *Epochs* needs to be set according to the training situation, at least let the loss converge, but not so large that serious overfitting occurs
- Can you use multiple ways to get your test set more than 99% accurate?

Experiment 4

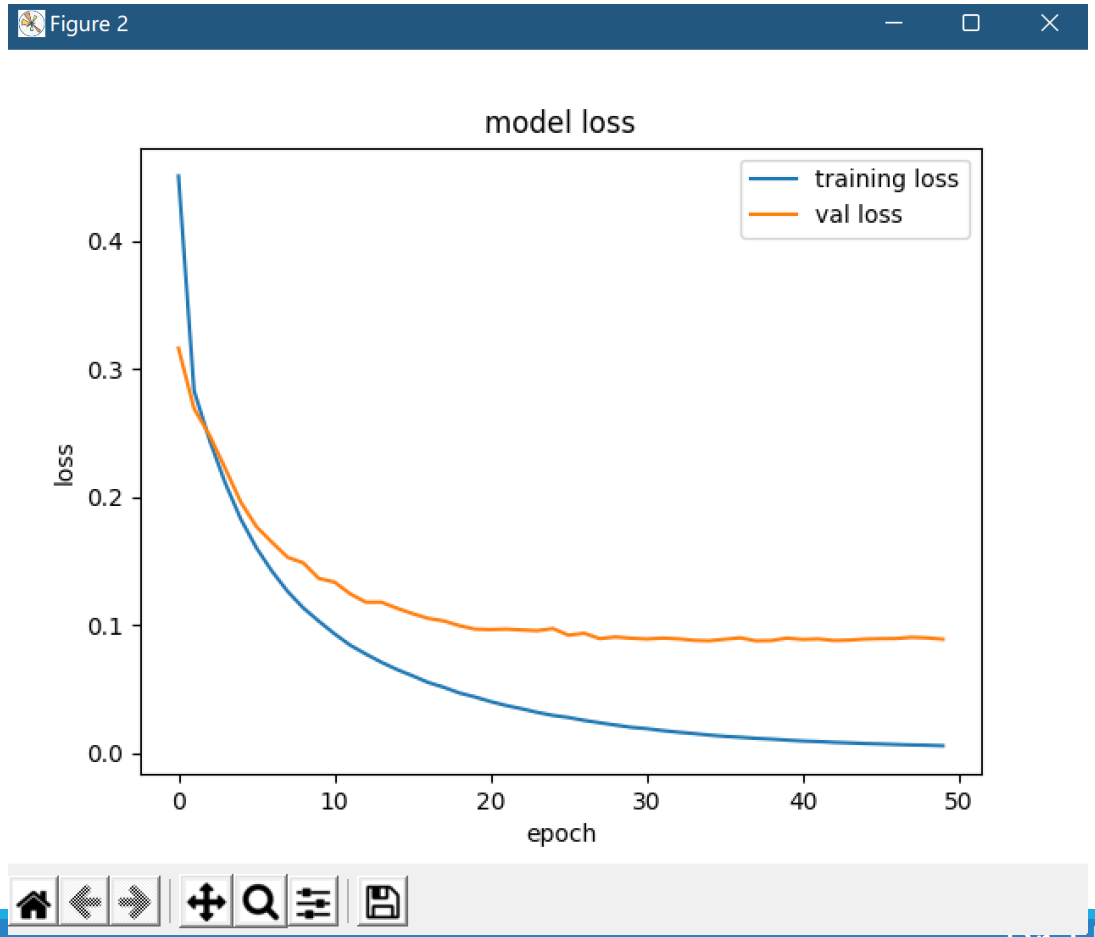
- **Task： Visual training curve (please submit)**
- Some intermediate results of visual training can help to make a judgment, such as whether it is overfitting
- This experiment requires visualization of loss curve and accuracy curve
- Curve meaning
  - X: epoch
  - Y: loss/accuracy value on the train/val set
- Supplement the code at the specified location

```
# 在此处实现你的可视化功能
##########################################################################################




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

Experiment 4

Submit

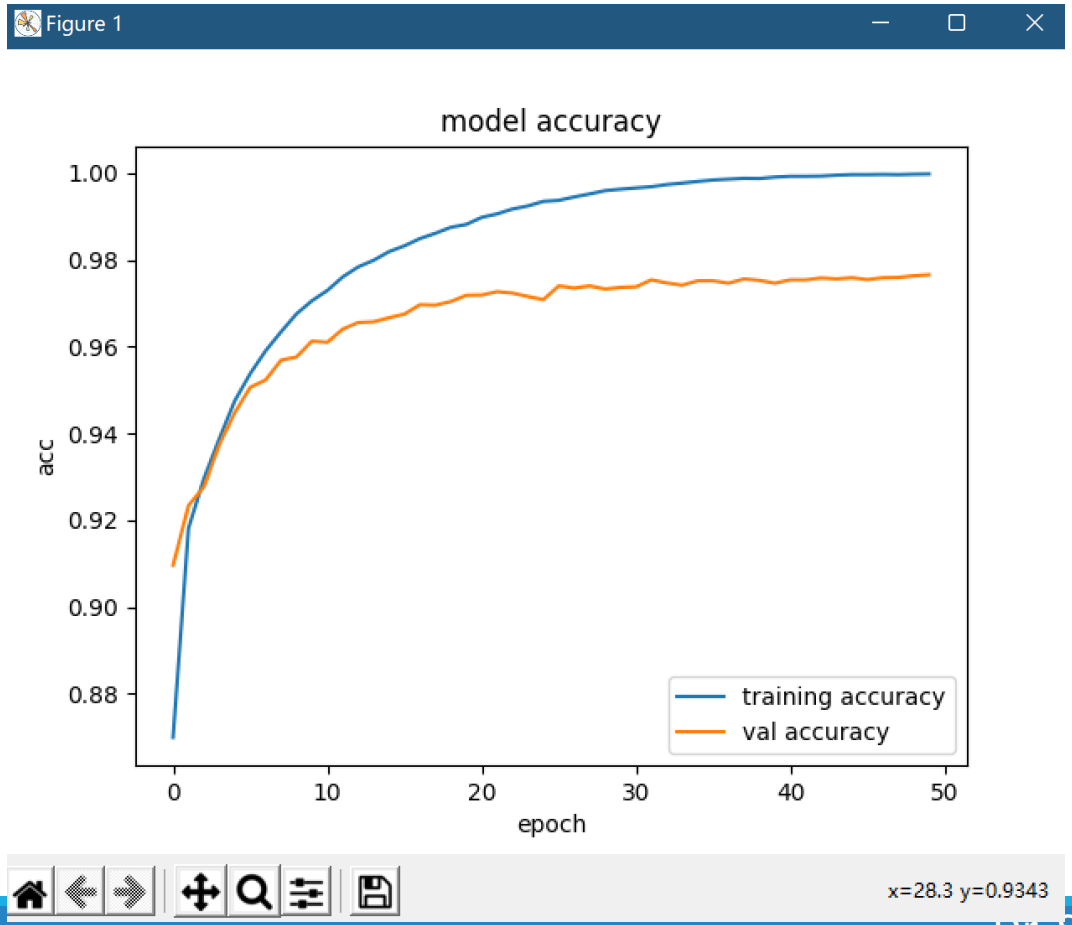
- loss curve example



Experiment 4

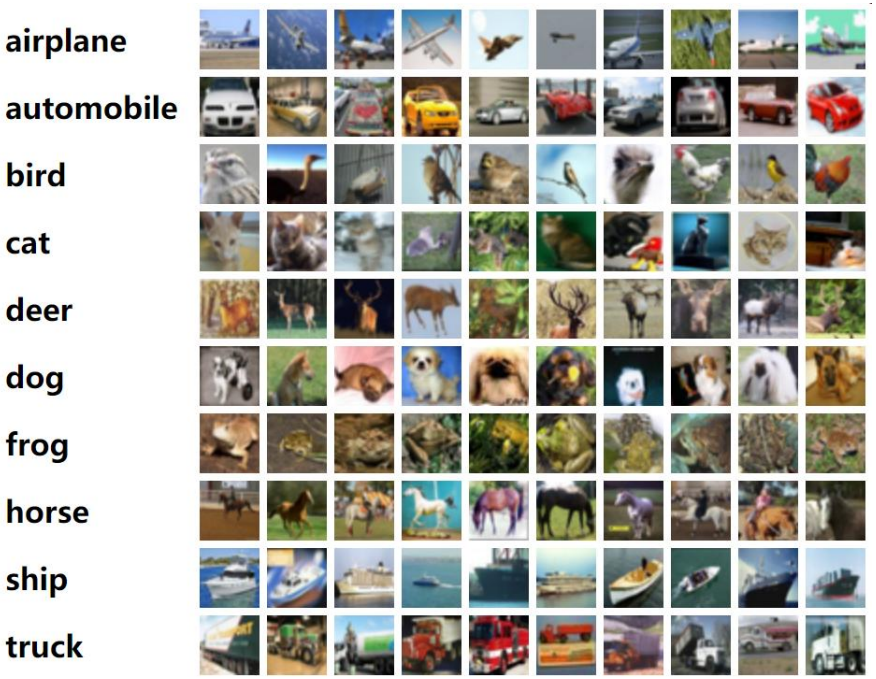
Submit

- accuracy curve example



Experiment 5 (expansion)

- **Cifar10 Dataset**
- CIFAR-10 is a small data set for the identification of universal objects. A total of *10 categories* of RGB color images (*3 channels*) are included. The picture size is *32 ×32*, and there are 50,000 training pictures and 10,000 test pictures in the data set. It is more complex than the mnist dataset.



Experiment 5 (expansion)

- **Task: Network training on the Cifar10 dataset**
- Data processing (using Keras's Cifar10 library)
- First, adjust the networks and use the related settings that you previously selected to perform best on the mnist data on Cifar10.
- Will this setup achieve the same good results as mnist? Adjust your network to achieve as much accuracy on the Cifar10 test set as possible by following the mnist reference.

Experiment 6 (expansion)

- **VGG**

- VGGNet, a deep convolutional network structure proposed by the *Visual Geometry Group* (VGG) at the University of Oxford, won runner-up in the 2014 ILSVRC classification task with an error rate of 7.32%, won the first place in the positioning task with 25.32% error rate.

- Paper address: https://arxiv.org/pdf/1409.1556.pdf

Experiment 6 (expansion)

- **VGG architecture**

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Experiment 6 (expansion)

- **Task: Build a network according to the idea of VGG and complete the classification on Cifar10**
- The network should have at least 8 convolutional layers (e.g., the 3rd and 4th stages using VGG 19)
- Due to the limitations of Cifar10 image size, you should use no more than two max pooling
- Does your previously tuned network classify better on Cifar10 than your newly built network?

Experiment 6 (expansion)

- **Tip**
- This experiment is recommended on a computer with an Nvidia GPU.
- If video memory allows, you can try running with a larger Batchsize.
- Memory is not allowed, then continuously reduce the Batchsize (generally need to be a multiple of 2).
- If you are using a device that can only use CPU training, try cutting out a small dataset for training to avoid consuming too much time.
- When running with a GPU, comment out the following two lines of code:

```
import os
os.environ["CUDA_VISIBLE_DEVICES"] = ""
```

If you've done all of the above...

- Try a new dataset: Cifar100 (comes with Keras)
- Try to deepen the depth of the VGG-like network you build and see how accurate it is.
- ……