HW1 guideline

DEEP LEARNING, 2024

OUTLINE

- **Installation and setup**
- **HW1 guideline**

Installation and setup

Anaconda

- Anaconda is an open-source distribution of Python and R programming languages
- Designed for data science, machine learning, and scientific computing
- Contains over 1,500 data science packages
- Simplifies package and environment management
- Available for Windows, MacOS, and Linux

Installation and setup

Anaconda offers multiple IDE options:

1. Spyder: Python IDE designed for scientific computing
2. VS Code: General-purpose code editor, installable through Anaconda
3. Jupyter Notebook: Web-based interactive development environment

Visual Studio Code

Spyder

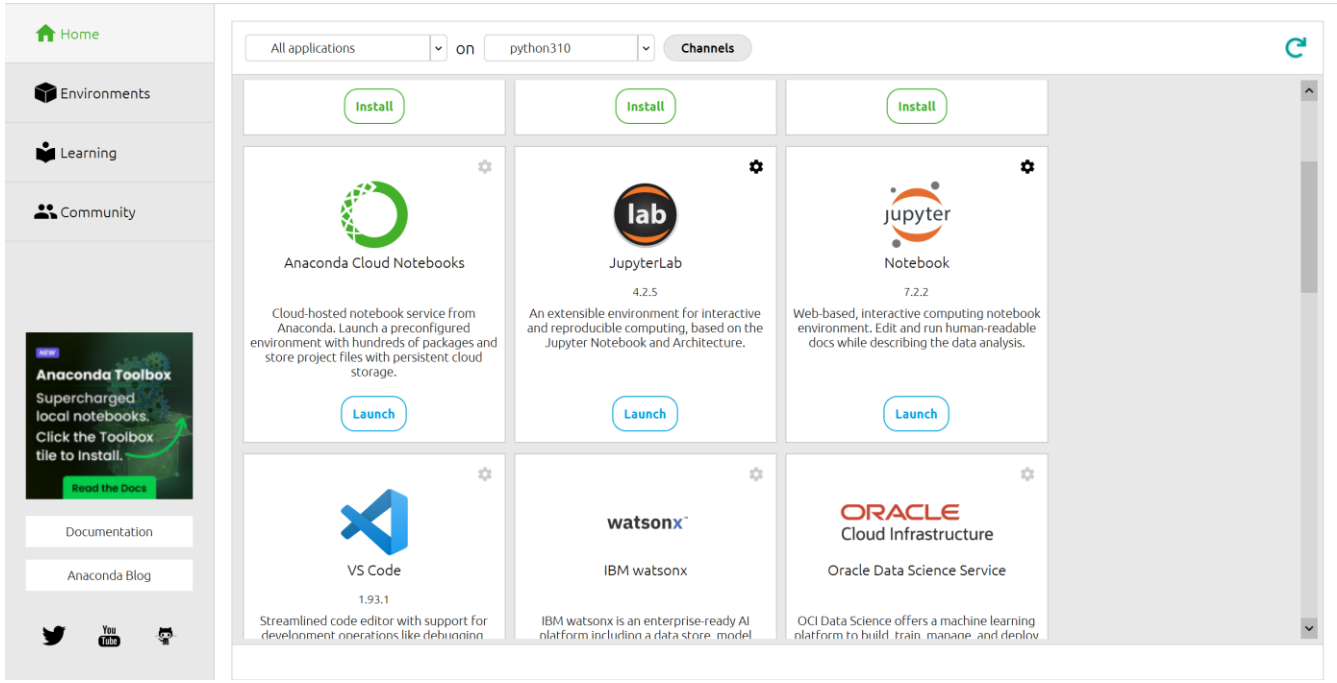
Jupyter Notebook

Installation and setup

Anaconda Navigator is the GUI for Anaconda

Key features:

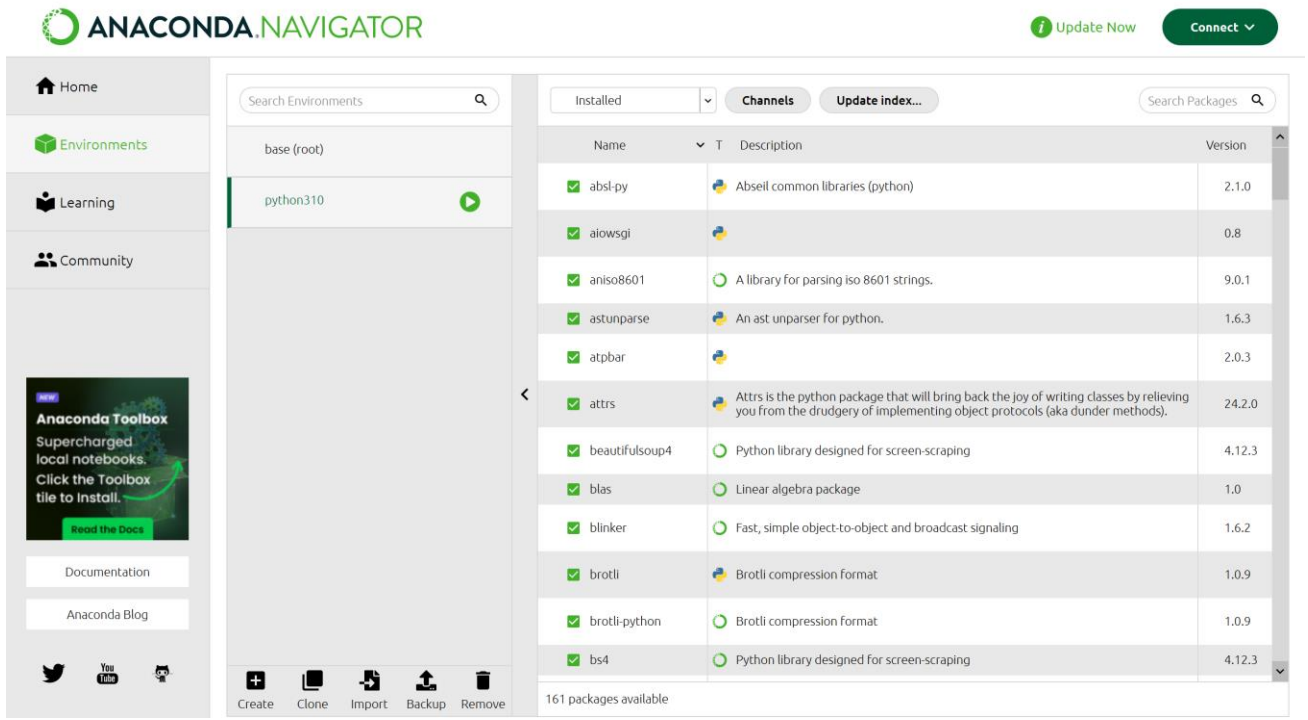
- Manage environments and packages
- Launch applications (e.g., Jupyter Notebook, Spyder)
- Install new packages and applications



Installation and setup

Package management features:

- Install, update, and remove packages
- Resolve package dependencies



Installation and setup

Installation Guide

1. Installing on Windows
2. Installing on macOS
3. Installing on Linux

Installation Video Tutorial

1. Installing on Windows
2. Installing on macOS
3. Installing on Linux

Installation and setup

Install **Numpy** 

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

```
# Best practice, use an environment rather than install in the base env
conda create -n my-env
conda activate my-env
# If you want to install from conda-forge
conda config --env --add channels conda-forge
# The actual install command
conda install numpy
```

Install Numpy with anaconda prompt

NumPy QuickStart — NumPy v1.21 Manual

Installation and setup

Install **matplotlib**

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged.[3] SciPy makes use of Matplotlib.

Conda packages

Matplotlib is available both via the *anaconda main channel*

```
conda install matplotlib
```

as well as via the *conda-forge community channel*

```
conda install -c conda-forge matplotlib
```

Tutorials — Matplotlib 3.4.3 documentation

Installation and setup

Install **pandas**

When working with tabular data, such as data stored in spreadsheets or databases, pandas is the right tool for you. pandas will help you to explore, clean, and process your data. In pandas, a data table is called a DataFrame.

The final step required is to install pandas. This can be done with the following command:

```
conda install pandas
```

To install a specific pandas version:

```
conda install pandas=0.20.3
```

[10 minutes to pandas — pandas 1.3.4 documentation (pydata.org)]

HW1 guideline

In this homework, you are asked to build a deep neural network model by using the Backpropagation and Stochastic Gradient Descent algorithm. You may design the network architecture by yourself, including the number of hidden layers, the number of hidden units, learning rate, the number of epochs and mini-batch size.

Do not use available Machine Learning or Deep Learning packages.

HW1 guideline - Try to follow these steps:

First, dealing with data:

1.import the data(**pandas**)

2.shuffle data samples for training and for testing(**pandas**)

3.one hot encoding(for loop and **Numpy**)

4.store training sample/label and test sample/label properly(python list and **Numpy**)

Second, build nn layers(better use functions to modularize your architecture!) :

1.define layer(python function and **Numpy**)

including weights and biases' dimension, initial value…

2.define activation function(python function and mathematical operation)

Sigmoid, tanh…

3.define the derivative of your activation function and layer

Derivative of sigmoid and tanh, layer for back-propagation

Make sure you know the difference between np.dot and np.multiply

Third, build nn architecture(better use functions to modularize your architecture!) :

1.define the forward propagation (list, for loop and **Numpy**)

Implement forward propagation yourself

(How many and what kind of layers you want? Order of these layers?)

Return output value of forward propagation

2.define backward propagation

Implement backward propagation to find the gradient yourself

(derivative of object function, use chain rule to find gradient)

Return gradient

3.define update

How to use gradient to update your weight and bias?(gradient, proper learning rate)

Return new weight and bias

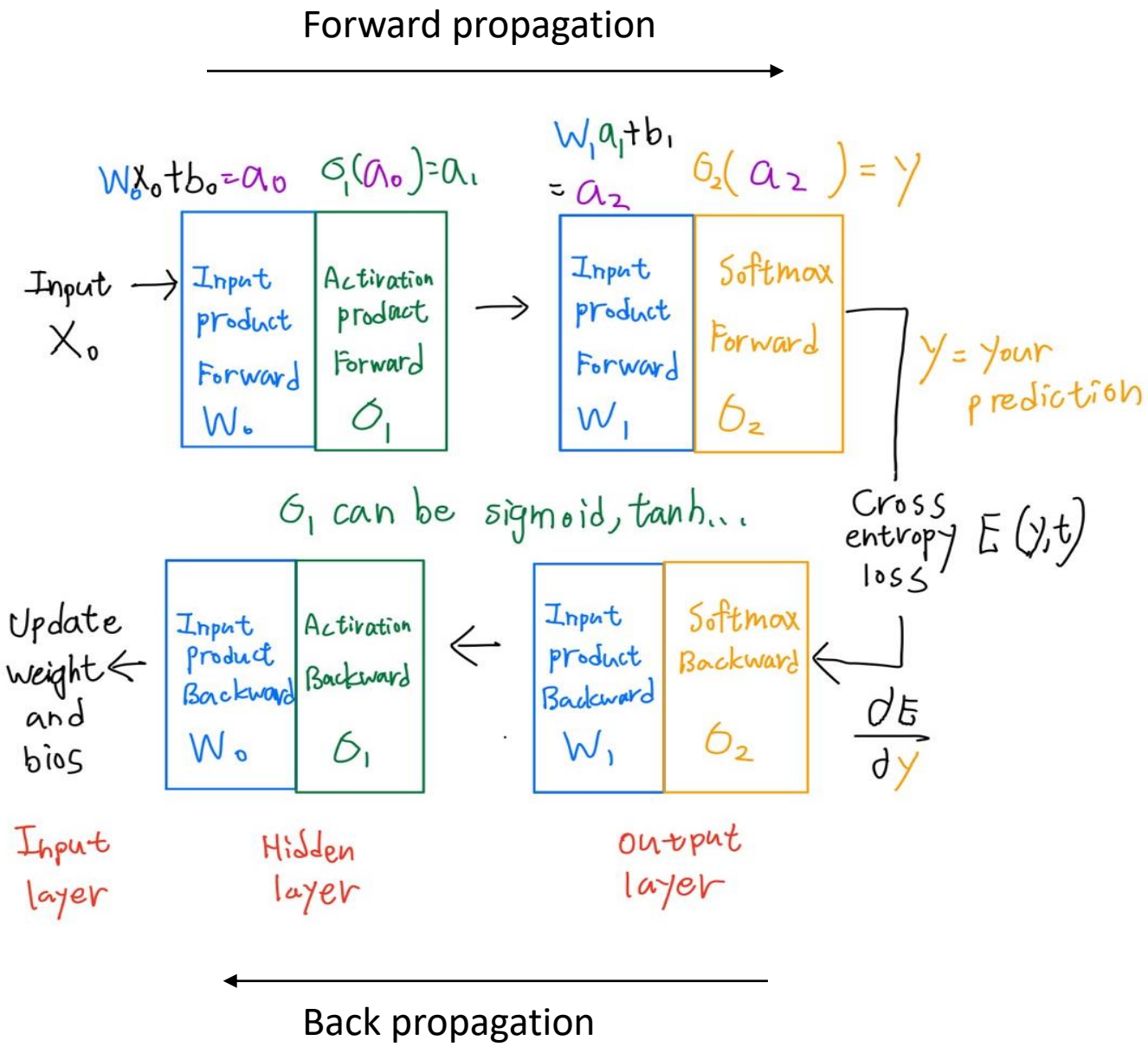
Make sure you know the difference between np.dot and np.multiply!

Take the architecture composed of input layer, hidden layer*1, and output layer as an example.

Correction

Note:

The activation function before output (σ_2 in this example) for classification is SoftMax, and regression does not require activation function before output.



Weight and bias updating

$$\boldsymbol{W}^{new} = \boldsymbol{W}^{old} - \eta \nabla_{\boldsymbol{W}} E$$

$$\boldsymbol{b}^{new} = \boldsymbol{b}^{old} - \eta \nabla_{\boldsymbol{b}} E$$

ex:

$$\nabla_{w_0} E = \frac{\partial a_0}{\partial w_0} \frac{\partial a_1}{\partial a_0} \frac{\partial a_2}{\partial a_1} \frac{\partial y}{\partial a_2} \frac{\partial E}{\partial y}$$

$$\nabla_{b_0} E = \frac{\partial a_0}{\partial b_0} \frac{\partial a_1}{\partial a_0} \frac{\partial a_2}{\partial a_1} \frac{\partial y}{\partial a_2} \frac{\partial E}{\partial y}$$

$$\nabla_{w_1} E = \frac{\partial a_2}{\partial w_1} \frac{\partial y}{\partial a_2} \frac{\partial E}{\partial y}$$

$$\nabla_{b_1} E = \frac{\partial a_2}{\partial b_1} \frac{\partial y}{\partial a_2} \frac{\partial E}{\partial y}$$

The formulas above is according to the architecture on the previous slide with 2 weights and 2 biases. (Based on this, the formula for different architectures can be calculated.)

Forth, design training flow(combine your module you define before!)

Define your training process(python function and **Numpy**),EX:

```
def train(X, Y, nn_architecture, epochs, learning_rate):
    weight, bias = init_layers(nn_architecture)
    print('Training parameters:\n epochs = %d learning rate = %f' % (epochs, learning_rate))

for i in range(epochs):
        Y_hat, cache = forward_propagation(X.T, weight, bias, nn_architecture)
        w_grad, b_grad = backward_propagation(Y_hat, Y, cache, weight, nn_architecture)
        weight, bias = update(weight, bias, w_grad, b_grad, nn_architecture, learning_rate)
```

…

Also remember to calculate your loss while training:

Define your loss function and decide how frequently you want to calculate your loss:

…

print('epoch %d loss : %f’ % (i, your loss function)

…

Finally, **Plot your result(matplotlib, or other data visualization toolkit you prefer)**

learning curve, training error rate, test error rate