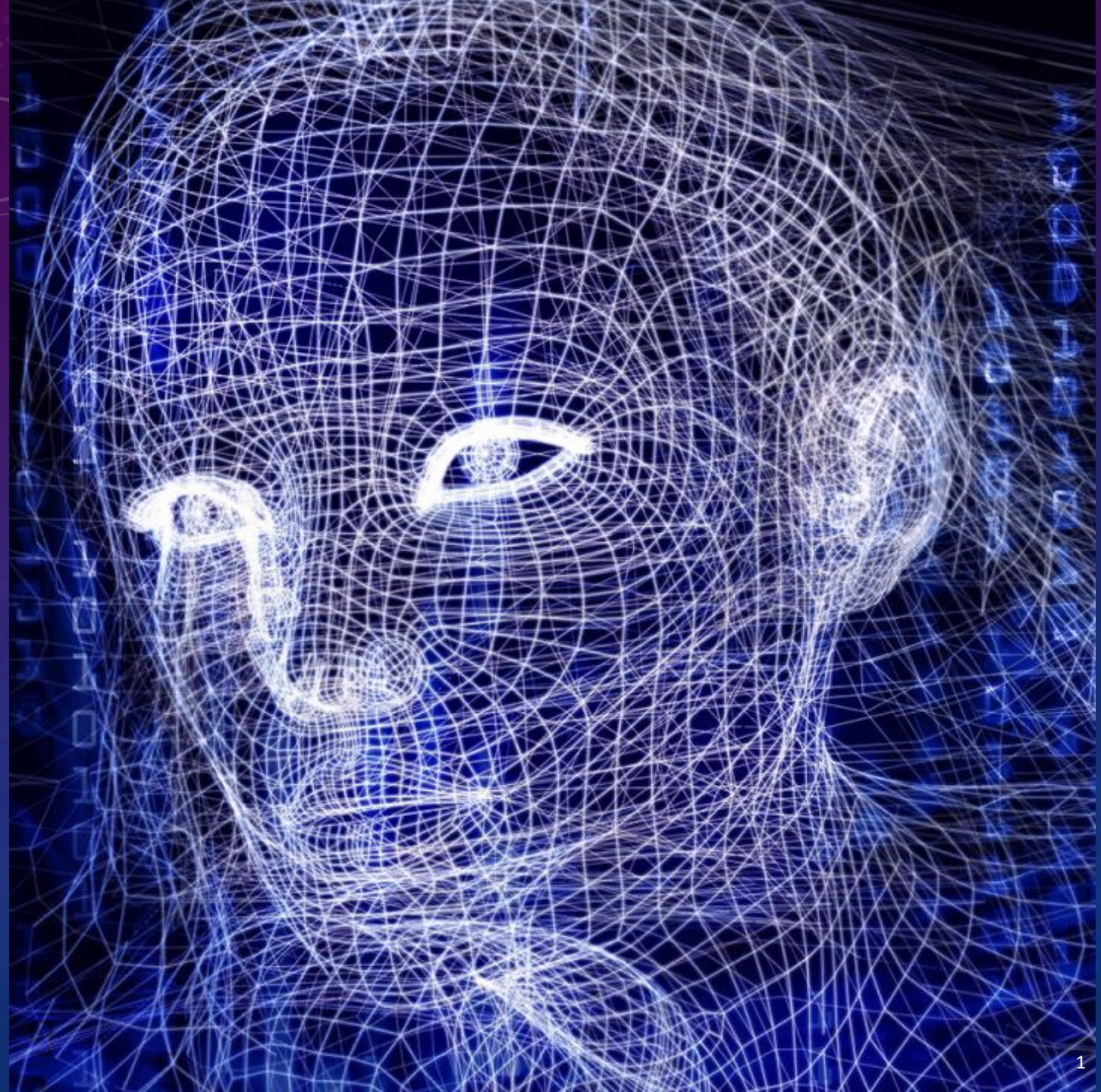
LECTURE SERIES FOR DIGITAL SURVEILLANCE SYSTEMS AND APPLICATION

Parameter Study in Deep Neural Networks



徐繼聖

Gee-Sern Jison Hsu

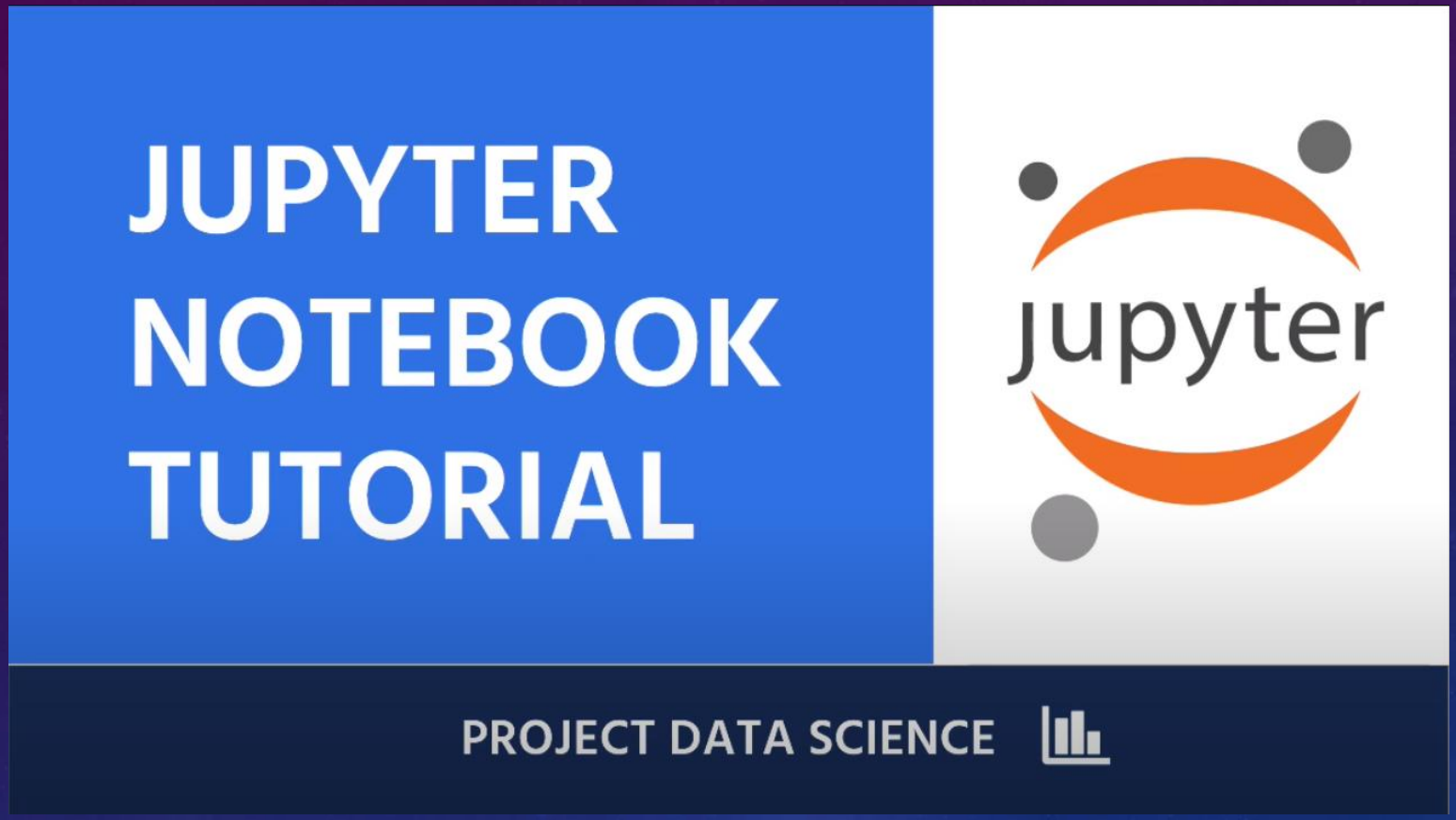
National Taiwan University of Science and Technology

Coming up…

- Introduction to Jupyter Notebook and Google Collab
- Example Problems
  - Weight Initialization
  - Different Activation Functions
  - Comparison of Different Channel on Feature Map
  - Comparison of Deep Networks

Jupyter Notebook

Jupyter Notebook Tutorial



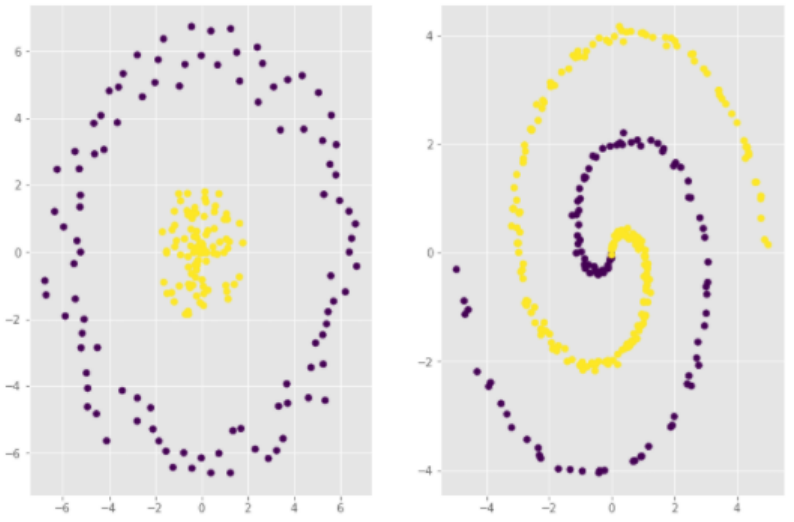
- A Comprehensive Introduction:
  - https://www.youtube.com/watch?v=DKiI6NfSIe8&ab_channel=ProjectDataScience[53:10]
  - https://www.youtube.com/watch?v=ZKp4vih2jhU [14:38]

Jupyter Notebook

- *Jupyter Notebook* is a web application for interactive data science and scientific computing.
- Using *Jupyter Notebook*, you can author engaging documents that combine live-code with narrative text, equations, images, video, and visualizations.
- By encoding a complete and reproducible record of a computation, the documents can be shared with others on GitHub, Dropbox, and the Jupyter Notebook Viewer.

In [4]:
```
1 X_circle,y_circle = generate_dummy_circle_data(100)
2 plt.subplot(121)
3 plt.scatter(X_circle[:,0],X_circle[:,1],c=y_circle)
4
5 X_spiral,y_spiral = generate_dummy_spiral_data(100,2)
6 plt.subplot(122)
7 plt.scatter(X_spiral[:,0],X_spiral[:,1],c=y_spiral)
```

Out[4]: <matplotlib.collections.PathCollection at 0x2a6cba88a20>



$X \in \mathbb{R}^{N,D}$ - like the multinomial logistic regression, our data is also represented as a matrix with N rows and D columns, where each row is a D-dimensional feature vector representing an instance / example in our dataset ($x_i \in \mathbb{R}^D$). In this particular example, $D = 2$.

$y \in \{0, .., C\}^N$ - Given C distinct classes, the prediction target is represented as a vector of length N and each example y_i is a scalar that can take on a value from 0 to C.

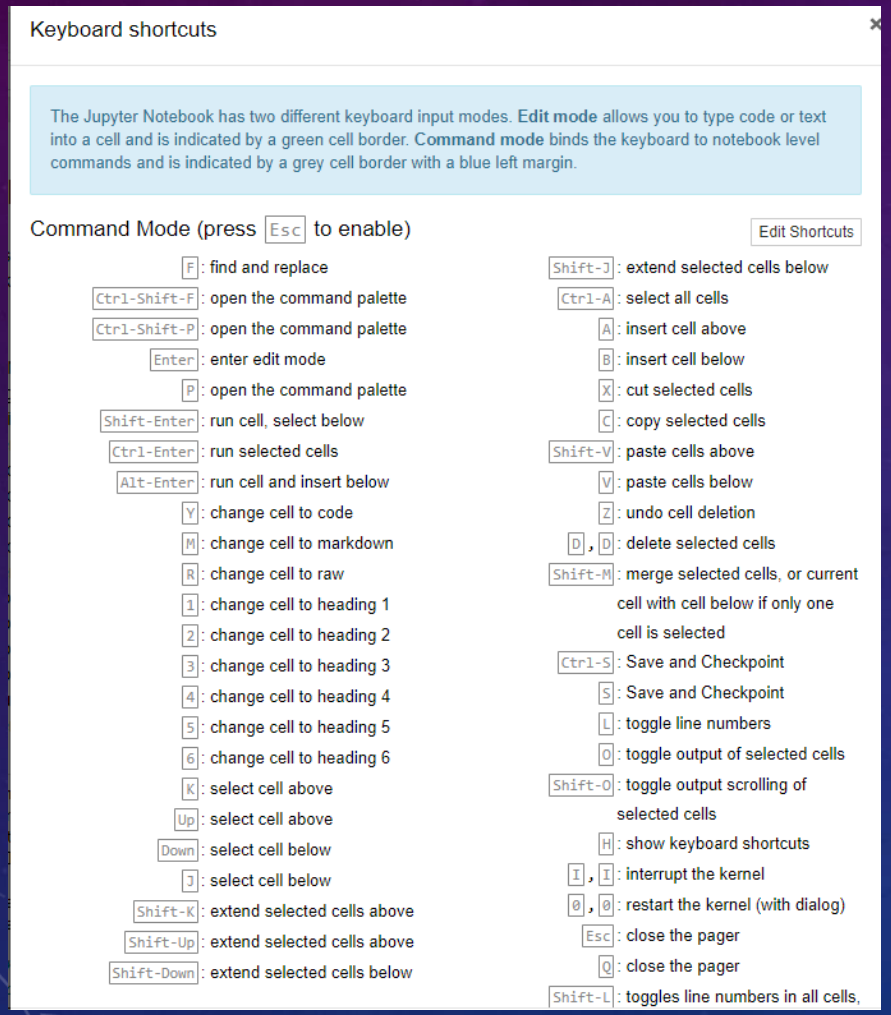
Note that the math expresses our target variable y_i as a one-hot encoding vector, where it has a value of 1 corresponding to the correct class and 0 everywhere else. In practice, we represent y_i as a scalar value denoting the index of the correct class instead. This is because it is not computationally and memory efficient to treat each y_i as a vector, specially for large number of classes, when almost all of its values are 0.

How to use Jupyter notebook

- The general command from CLI is: *jupyter notebook [Filename]*
- Make sure that you installed *jupyter notebook* before with the pip command in your virtual environment. *Pip install notebook*

```
(venv) D:\02_Documents\04_Projects\TA\examples>jupyter notebook Example3-1_Weight_Init_v2.ipynb
[I 11:21:02.370 NotebookApp] Serving notebooks from local directory: D:/
[I 11:21:02.371 NotebookApp] Jupyter Notebook 6.1.4 is running at:
[I 11:21:02.372 NotebookApp] http://localhost:8888/?token=bc19d915ef8d07feb6c0637badcd71ee13ae5d06e2a8c1ec
[I 11:21:02.372 NotebookApp]  or http://127.0.0.1:8888/?token=bc19d915ef8d07feb6c0637badcd71ee13ae5d06e2a8c1ec
[I 11:21:02.373 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 11:21:02.382 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///C:/Users/morit/AppData/Roaming/jupyter/runtime/nbserver-22536-open.html
    Or copy and paste one of these URLs:
        http://localhost:8888/?token=bc19d915ef8d07feb6c0637badcd71ee13ae5d06e2a8c1ec
     or http://127.0.0.1:8888/?token=bc19d915ef8d07feb6c0637badcd71ee13ae5d06e2a8c1ec
```

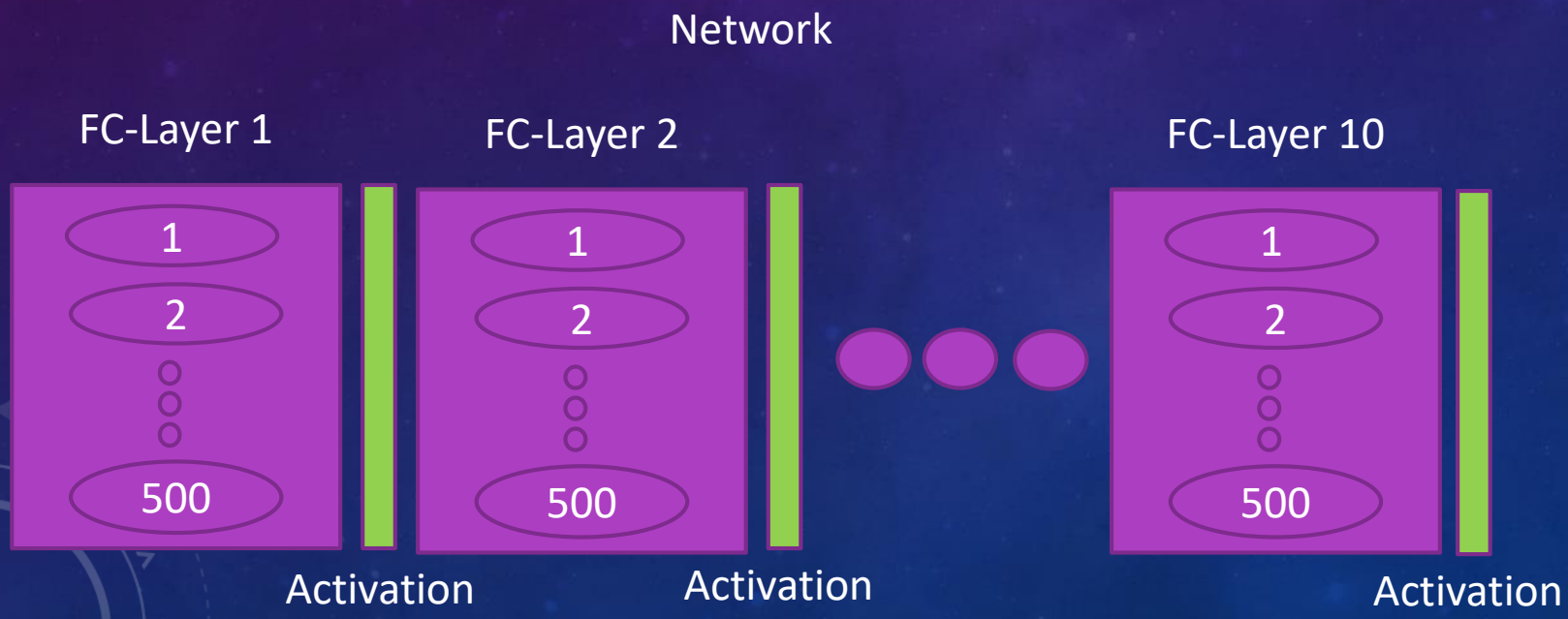
Jupyter notebook keyboard shortcuts



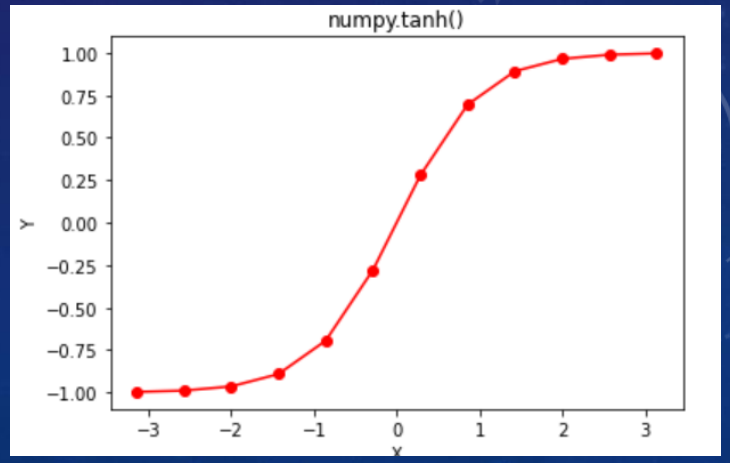
To improve your handling of jupyter notebook scripts, just press the letter ‘h’ in jupyter notebook and you will see a useful list of all shortcuts. It will take a while to get familiar with them, but it is worth to pick up as many shortcuts as you can. They can save you a lot of time and polish up your programming skills!

Example 1 – Weight Initialization

Example 1 examines a FC-Network with 500 Neurons on each layer and 10 layers in total. We are using tanh non-linearities as our activation functions and can change the initialization of the weights. We will try out a small, medium and large initialization. After initialization, we perform one forward pass. Then, we will look at the output values of the layers with a normally distributed input and estimate the Gradient.



Activation function tanh



Example 1 – Code Setup – Part 1

- For this example, we will use only the standard libraries numpy and matplotlib
- D is our input vector, which is created from a gaussian normal distribution
- In line 5, we define our hidden_layer_sizes.
- Line 6 sets up the activation functions. In our example, we only consider relu and tanh

```
1  #assume some unit gaussian 10-D input data
2  import numpy as np
3  import matplotlib.pyplot as plt
4  D = np.random.randn(1000, 500)
5  hidden_layer_sizes =[500]*10
6  nonlinearities =['relu']*len(hidden_layer_sizes)
```

Example 1 – Code Setup – Part 2

- For the activation function, we make use of the lambda function and dictionaries, which are both very useful in python.
- The for loop, starting in line 3, is activating our Network Architecture
  - Line 7 initializes the weight
  - Line 8 performs the forward pass
  - Line 9 is feeding the result of line 8 into our activation function
  - Finally we save our result in line 10 to Hs

```
1  act={'relu': lambda x: np.maximum(0,x), 'tanh': lambda x: np.tanh(x)}
2  Hs={}
3  for i in range(len(hidden_layer_sizes)):
4      X=D if i==0 else Hs[i-1] # input at this layer
5      fan_in = X.shape[1]
6      fan_out = hidden_layer_sizes[i]
7      W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in/2) #layer intialization
8      H=np.dot(X,W) #matrix multiplication
9      H=act[nonlinearities[i]](H)
10     Hs[i]=H
11
```

Lambda function in python

- A lambda function is a small anonymous function.
- A lambda function can take any number of arguments, but can only have one expression.
- The expression is executed and the result is returned:
- A very simple example is the following
  - In this example, we define the lambda argument *a*
  - We then define the expression *a+10*
  - Can you guess the result?

```
x = lambda a: a + 10
print(x(5))
```

Help with commands

numpy.random.randn

numpy.random.**randn**(*d0*, *d1*, ..., *dn*)

Return a sample (or samples) from the "standard normal" distribution.

If positive, int_like or int-convertible arguments are provided, **randn** generates an array of shape `(d0, d1, ..., dn)`, filled with random floats sampled from a univariate "normal" (Gaussian) distribution of mean 0 and variance 1 (if any of the d_i are floats, they are first converted to integers by truncation). A single float randomly sampled from the distribution is returned if no argument is provided.

This is a convenience function. If you want an interface that takes a tuple as the first argument, use numpy.random.standard_normal instead.

Parameters: d0, d1, ..., dn : *int, optional*

The dimensions of the returned array, should be all positive. If no argument is given a single Python float is returned.

Returns: Z : *ndarray or float*

A `(d0, d1, ..., dn)`-shaped array of floating-point samples from the standard normal distribution, or a single such float if no parameters were supplied.

See also:

standard_normal Similar, but takes a tuple as its argument.

Example 1 – Code Setup – Part 3

- To examine our results, we make use of the numpy module
- We will look at each layer's mean and standard deviation, with *np.mean* and *np.std*

```
# look at distribution at each layer
print('input layer had mean %f and std %f'.format(np.mean(D), np.std(D)))
layer_means = [np.mean(H) for i, H in Hs.items()]
layer_stds = [np.std(H) for i, H in Hs.items()]
for i,H in Hs.items():
    print('hidden layer {} had mean {} and std {}'.format(i+1, layer_means[1], layer_stds[i]))
```

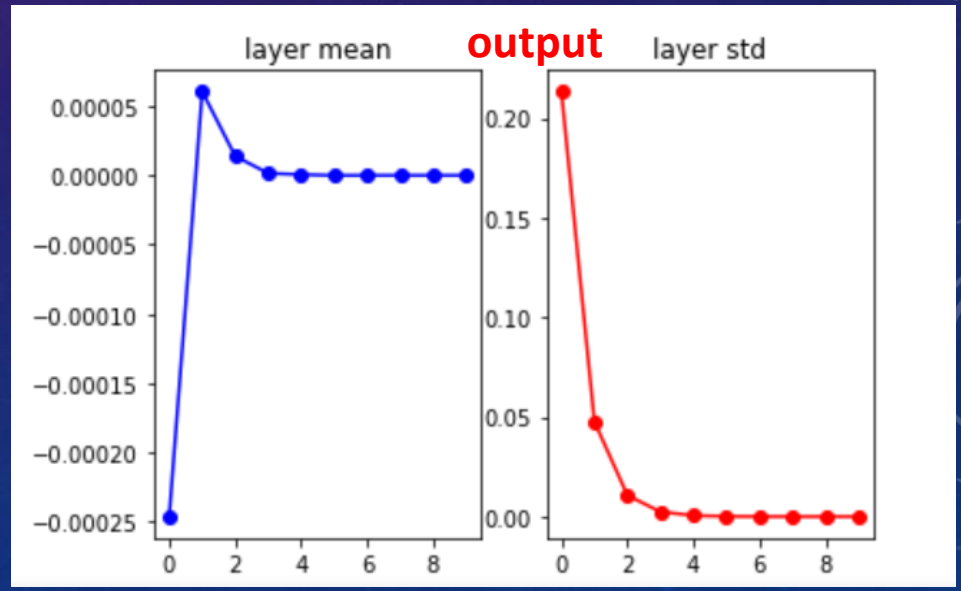
Example 1 – Code Setup – Part 4

- To plot our results, we make use of the matplotlib.pylot module
- We will plot the mean and standard deviation *std* from before.

code

```
#plot the means and standard deviations
plt.figure()
plt.subplot(121)
plt.plot(Hs.keys(), layer_means, 'ob-')
plt.title('layer mean')
plt.subplot(122)
plt.plot(Hs.keys(),layer_stds, 'or-')
plt.title('layer std')
```

output



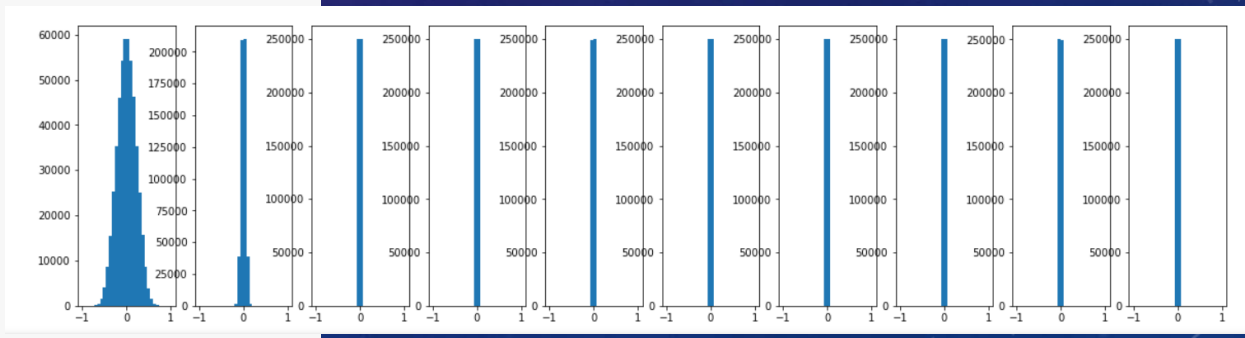
Example 1 – Code Setup – Part 5

- In the last part, we will draw a histogram of each layer's output after the activation function with the *plt.hist()* function in line 23. The *ravel()* command in line 23 is giving us a flattened array from the input array.

code

```
17  #plot the raw distributions
18
19  plt.figure(figsize=(20,5))
20  for i, H in Hs.items():
21      plt.subplot(1, len(Hs), i+1)
22      #print(H)
23      plt.hist(H.ravel(), 30, range=(-1,1))
```

output



Let’s have a look at the following examples

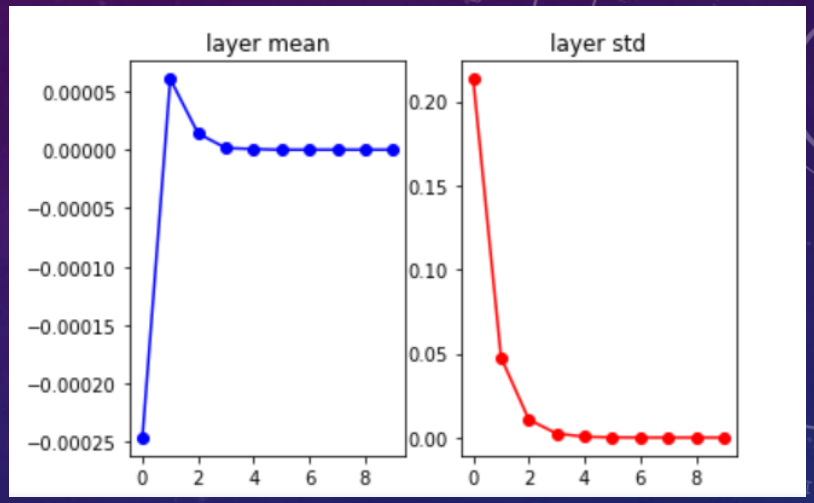
- Different dimensions of the input
  - Small
  - Medium
  - Large
- Normalized input
- How do the two activation functions tanh and ReLU behave?

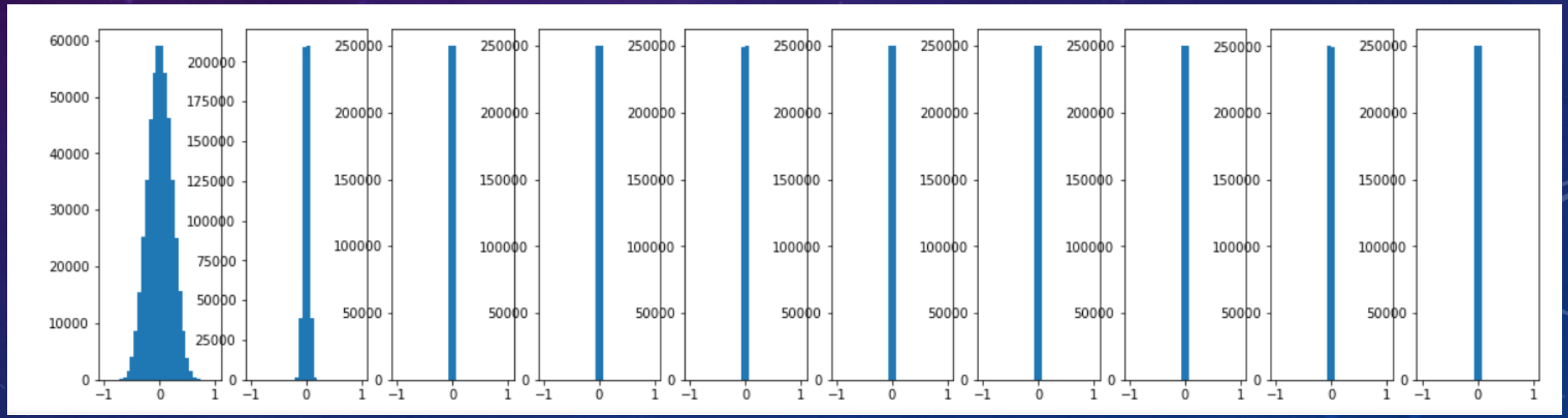
Example 1 Different Initialization of Neural Networks from Layer 1 to Layer 10

Small Weight Initialization:

Code Change:

```
W = np.random.randn(fan_in, fan_out)*0.01
```





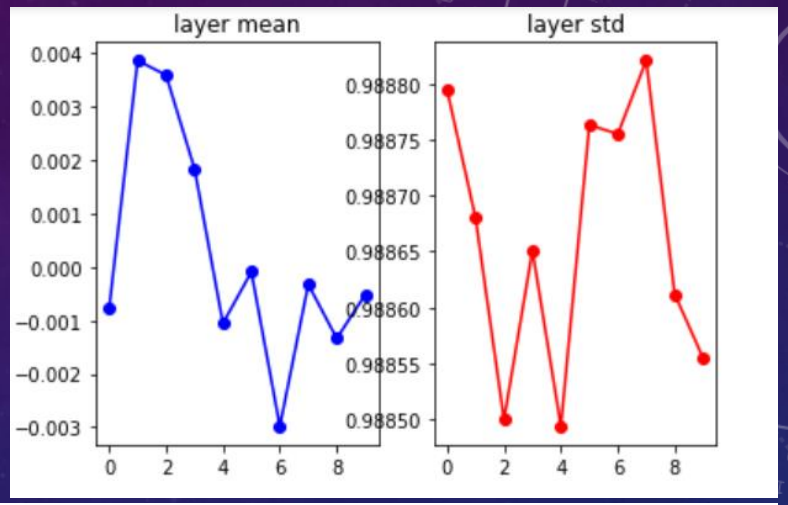
Issue: Activation functions approach 0. Therefore the gradients will be 0. How does the backward pass would look for this scenario?

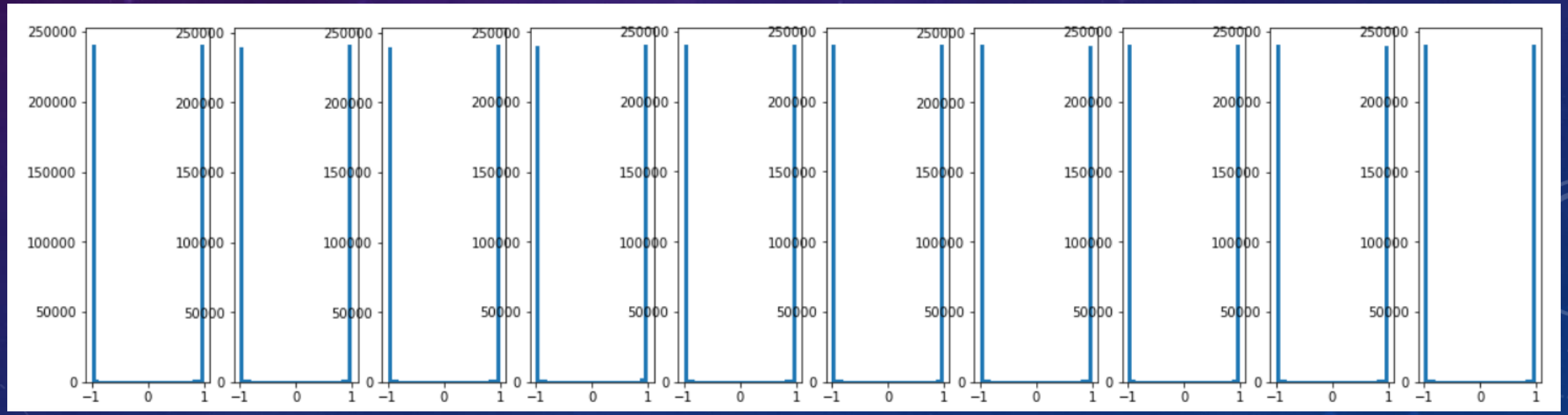
Example 1 Different Initialization of Neural Networks from Layer 1 to Layer 10

Large Weight Initialization:

Code Change:

```
W = np.random.randn(fan_in, fan_out)*1.6
```





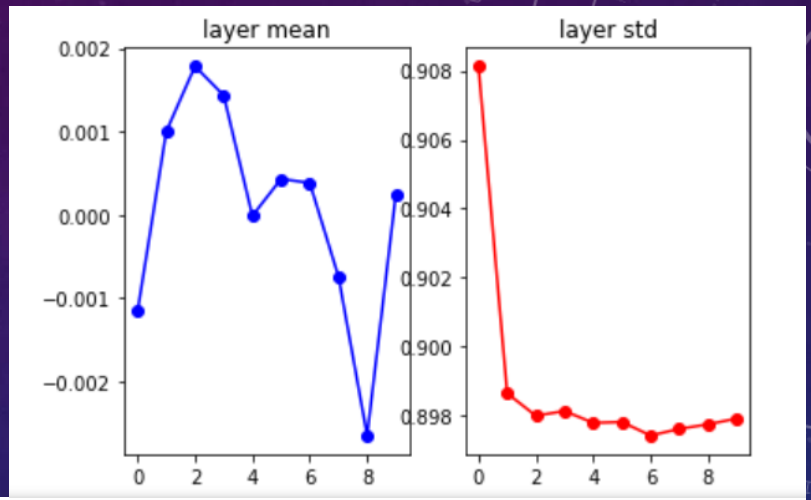
Issue: all neurons are completely saturated, either -1 or 1. Gradients will be zero. Backprop struggles and training is not possible

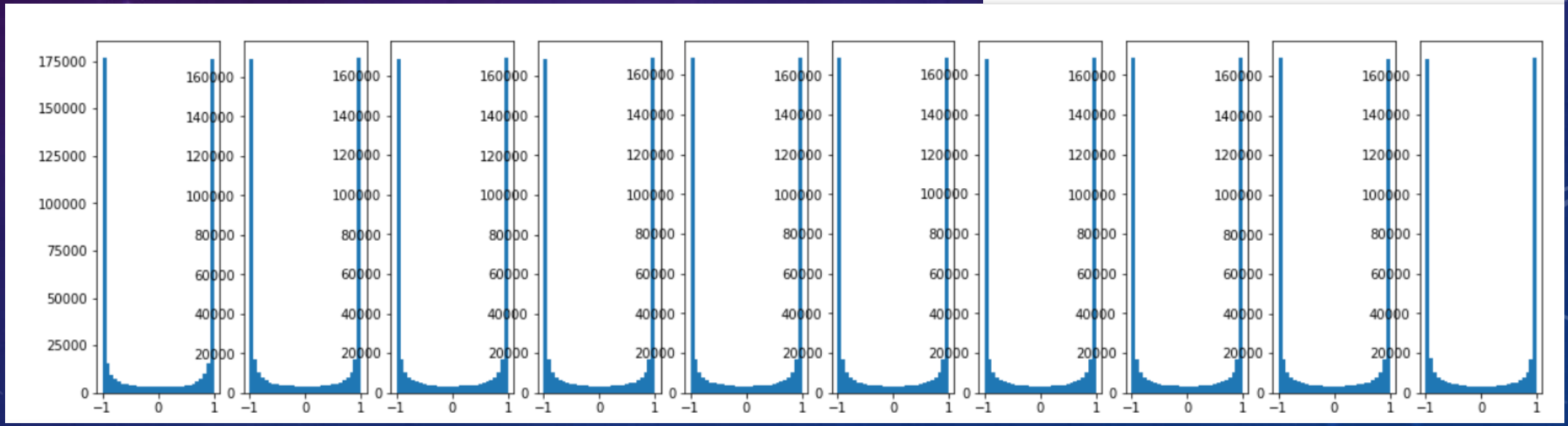
Example 1 Different Initialization of Neural Networks from Layer 1 to Layer 10

Code Change:

Medium Weight Initialization:

```
W = np.random.randn(fan_in, fan_out)*0.2
```





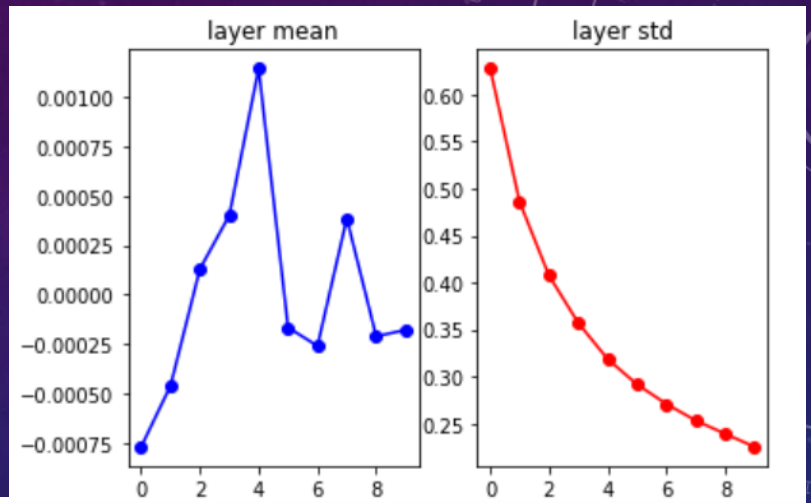
Observation: This looks already more like the result we are requesting in order to perform backprop, since the results are not collapsing in a single value.

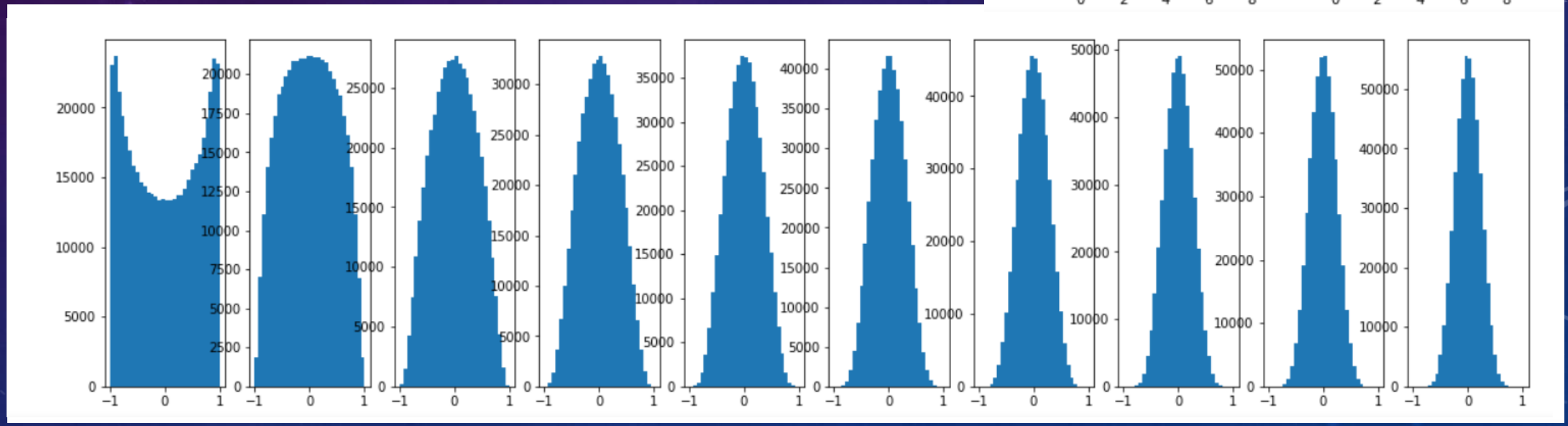
Example 1 Different Initialization of Neural Networks from Layer 1 to Layer 10

Now, we will use a kind of normalization.

Code Change:

```
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in)
```





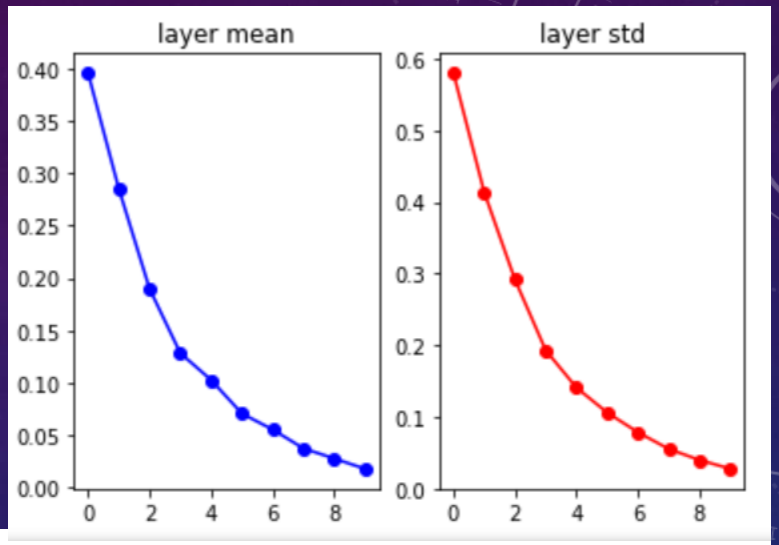
With this result, we receive a reasonable initialization for our Network setup, from where we have the base for a successful training.

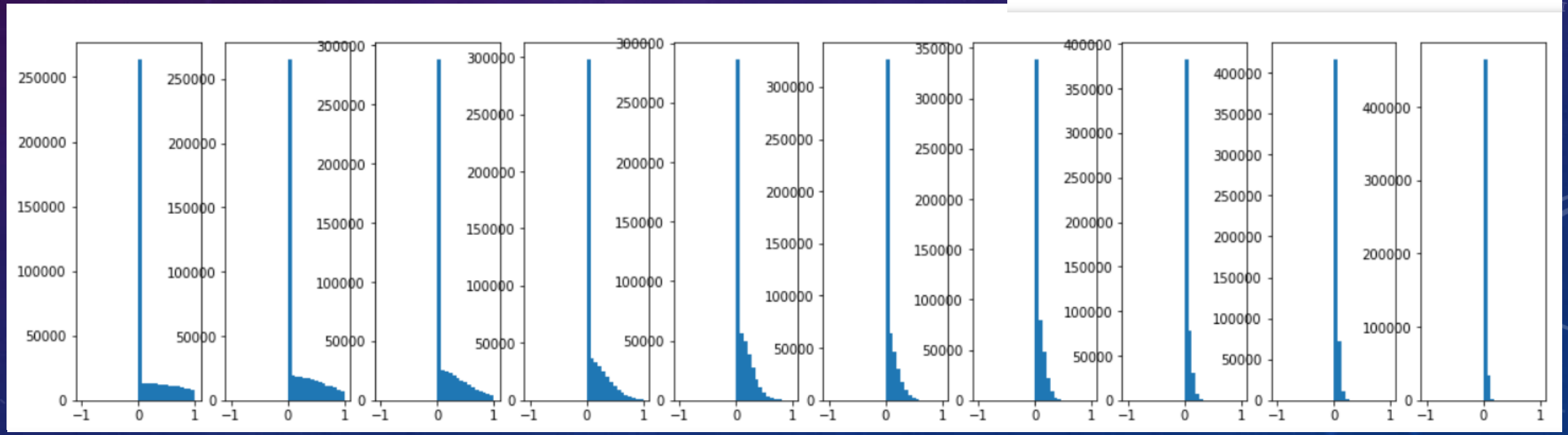
Example 1 Change Activation Function to Relu

Now, we will use normalization.

Code Change:

```
nonlinearities =['relu']*len(hidden_layer_sizes)
```





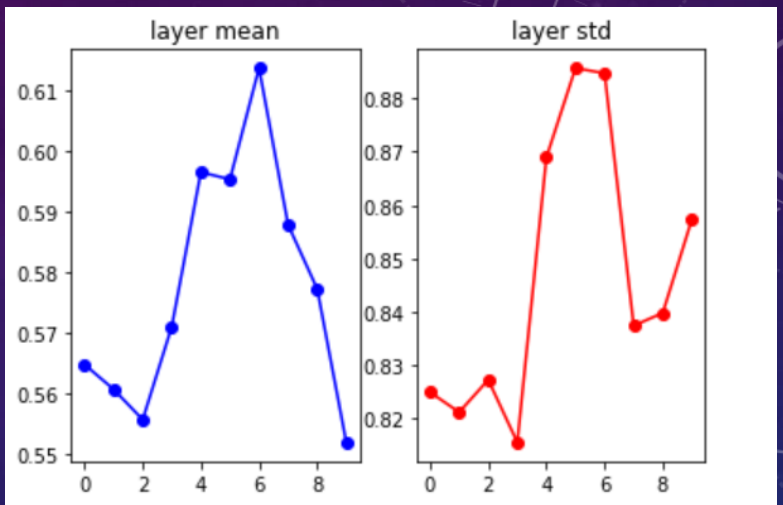
With a different activation function, our Network breaks again and backprop algorithm will fail.

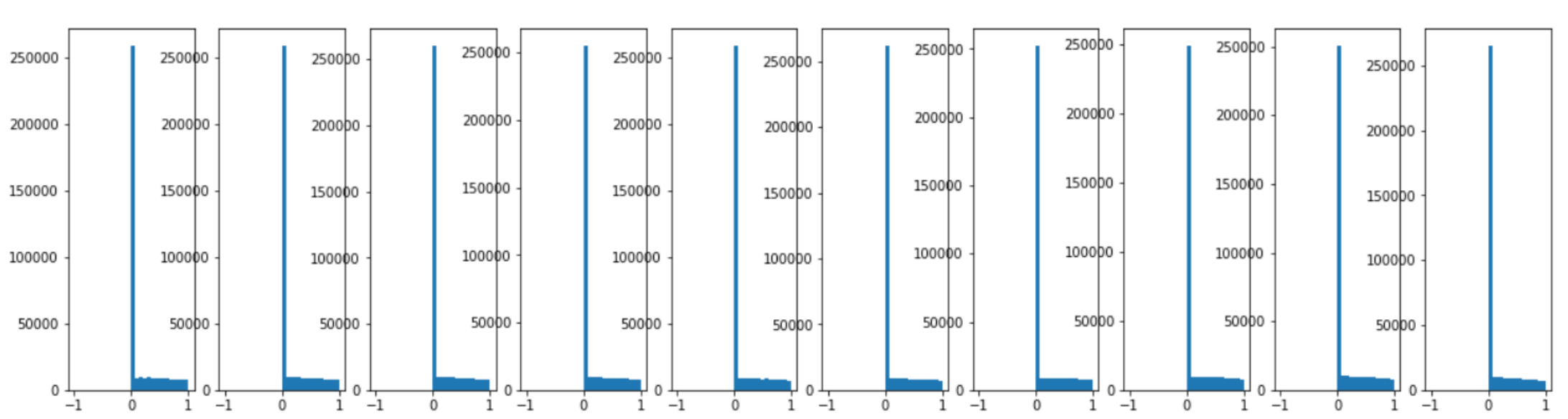
Example 1 Add a division by two during Initialization of W

Now, we will use a kind of normalization.

Code Change:

```
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in/2)
```

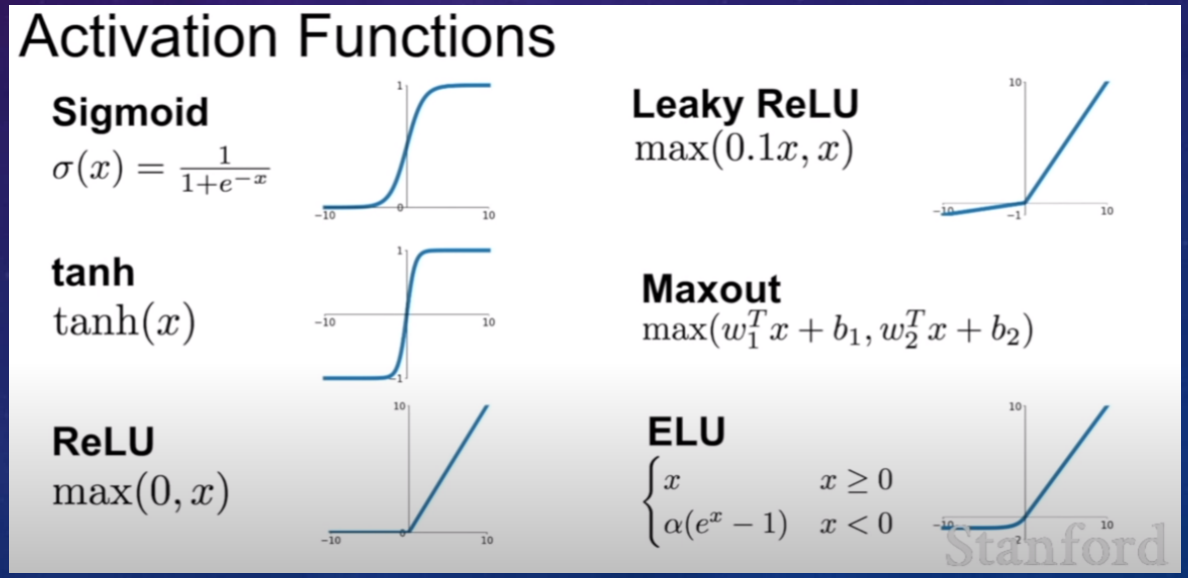




Small changes can have a big impact during initialization. The additional division by 2 enables back prop.

Exercise 1-1 – Weight Initialization and Forward Pass

- We have looked at two activation functions so far: tanh and ReLU. There are far more and it is an interesting area of research. Therefore, investigate the initialization behavior of two more activation functions of your choice. You can take some examples from the Stanford slides (below) or browse the internet for others. Use *Example1-1_Weight_Init_v2.ipynb* from moodle as a template.
- Upload your observations, comments and your code to Moodle.



Example 2 – Different Activation Functions

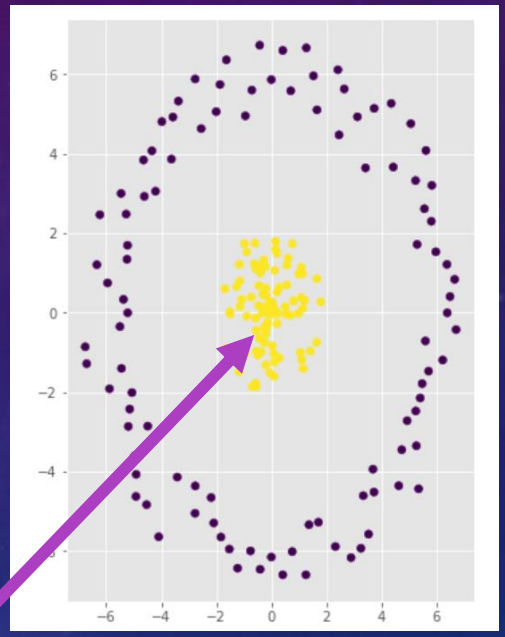
- In this example, we will combine previous ideas to create neural networks with an arbitrary number of layers to perform multi-class classification, also called multi-layered perceptrons.
- The example will show the following:
  - Compute Numerical Gradients to be used as gradient checkers
  - Build the general architecture of a Neural Network Model consisting of fully connected layers.
    - Initializing Parameters/Weights of each layer
    - Implement the forward pass
      - forward pass of fully connected layers
      - forward pass of sigmoid activation function
      - forward pass of tanh activation function
      - forward pass of ReLU activation function

Example 2 – Different Activation Functions

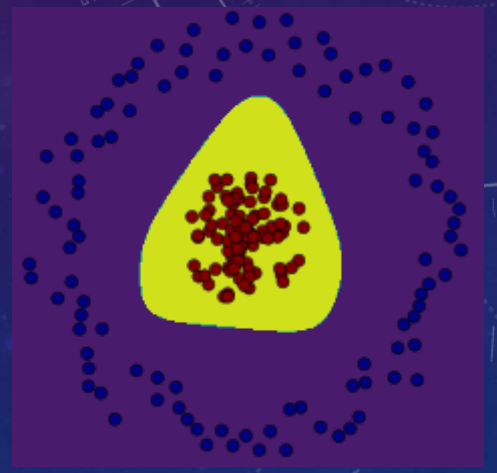
- We will also
  - Implement the backward pass to compute for gradients
    - backward pass of fully connected layers
    - backward pass of sigmoid activation function
    - backward pass of tanh activation function
    - backward pass of ReLU activation function
  - Calculating the Cost/Loss/Objective Function
  - Implement gradient descent to update the paramters

Example 2 – The Problem at one glance

- We create two datasets, where we will first focus on the circular data. Later on, we will try it out for the spiral data. The colors indicate the different classes and our goal is to find a boarder, that separates the two classes from each other.
- We consider 200 points in each dataset, 100 points in each class.
- Example for the input **sample 0** on the right



Decision boundary



X[0] = [-0.42987855 -0.64571507] y[0] = 1

Sample 0

In a mathematical notation…

$X \in \mathbb{R}^{N,D}$ - like the multinomial logistic regression, our data is also represented as a matrix with N rows and D columns, where each row is a D-dimensional feature vector representing an instance / example in our dataset ($x_i \in \mathbb{R}^D$). In this particular example, $D = 2$.

$y \in \{0, .., C\}^N$ - Given C distinct classes, the prediction target is represented as a vector of length N and each example y_i is a scalar that can take on a value from 0 to C.

Note that the math expresses our target variable y_i as a one-hot encoding vector, where it has a value of 1 corresponding to the correct class and 0 everywhere else. In practice, we represent y_i as a scalar value denoting the index of the correct class instead. This is because it is not computationally and memory efficient to treat each y_i as a vector, specially for large number of classes, when almost all of its values are 0.

Example 2 – Code Details - 1

- As previously, we will use the matplotlib and numpy modules
- The first box of codes specifies our plot configurations.
- *Np.random.seed(1)* is used to ensure that random numbers created with this script match, no matter which machine we are using.

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  %matplotlib inline
4  plt.style.use('ggplot')
5
6  plt.rcParams['figure.figsize'] = (12.0, 8.0) # set default size of plots
7  plt.rcParams['image.interpolation'] = 'nearest'
8
9  # Fix the seed of the random number
10 # generator so that your results will match ours
11 np.random.seed(1)
12
13 %load_ext autoreload
14 %autoreload 2
```

Why is Gradient Checking important?

- As you have seen in Example 1, the backprop algorithm requires a non-zero Gradient. Therefore, this exercise will perform a gradient checker, to ensure that our gradients will not approach zero.
- Carrying out the derivative **checking** procedure significantly increase your confidence in the correctness of your code. Saying it in one sentence: **Gradient Checking** is kind of debugging your back prop algorithm. **Gradient Checking** basically carries out the derivative **checking** procedure

Implement a Gradient Checker

- We can calculate the numerical gradient of a function with the following formula, which we implement in python in the *gradient_checker.py* file.
- We can also find the analytical gradient with our loss function, which is on the right.
- Deriving the loss function for our weights *W*, we receive the analytical gradient.

$$\frac{f(x+h) - f(x-h)}{2h}$$

```
loss = 0.5 * np.mean((X.dot(W) - y)**2)
```

```
grad['W'] = np.dot(X.T, X.dot(W) - y) / N
```

Comparing Numerical and Analytical Gradient

- To check our implementation, we create a test case with the input *X_dummy*, our weights *W_dummy* and the *groundtruth y_dummy*
- With these results, both analytical and numerical gradient must be identical

```python
from gradient_checker import compute_numerical_gradient, relative_error
np.random.seed(1)

# creates a dummy loss function
def dummy_loss_function(X, y, W):
    N, D = X.shape
    loss = 0.5 * np.mean((X.dot(W) - y)**2)

    grad = {}
    grad['W'] = np.dot(X.T, X.dot(W) - y) / N

    return loss, grad

X_dummy = np.random.randn(3,5)
y_dummy = np.random.randn(3,1)
W_dummy = np.random.randn(5,1)
```

Comparing Numerical and Analytical Gradient

- Printing the results as on the right results in the conclusion, that the relative error is negligible and we can use numerical gradients for our convenience.

```
# solves for the analytical gradient.
loss, grad = dummy_loss_function(X_dummy,y_dummy,W_dummy)

# lambda functions are anonymous functions defined without a name. We use this to
# pass in a function that has only W as its parameter and everything else is fixed.
# Since the compute_numerical_gradient expects a function that outputs a scalar value,
# we return only the first element which corresponds to the loss value
numerical_gradients = compute_numerical_gradient(lambda W: dummy_loss_function(X_dummy, y_dummy, W)[0], W_dummy)

print("Analytical Gradients")
print(grad['W'])

print("Numerical Gradients")
print(numerical_gradients)

print("Relative Error")
print(relative_error(grad['W'], numerical_gradients))
```

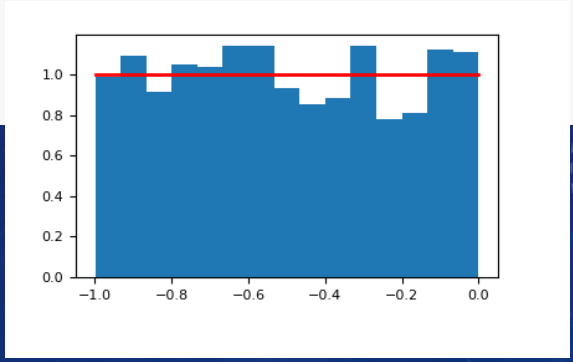
Result

```
Analytical Gradients
[[-0.74692649]
 [ 0.54694129]
 [-0.76442224]
 [-0.2044723 ]
 [ 0.35616946]]
Numerical Gradients
[[-0.74692649]
 [ 0.54694129]
 [-0.76442224]
 [-0.2044723 ]
 [ 0.35616946]]
Relative Error
5.0943132806378086e-11
```

Example 2 – Code Details – 2 – Data creation

- To create a circular toy dataset, we create a function that creates num_points randomly on two circles with an inner and an outer diameter. Points on the inner circle will be class 1, points on the outer circle will be class 0.
- The sample points for the dataset are created with the command *np.random.uniform(low,high,# of points),* which draws points from a uniform distribution
- At the end, we shuffle the data and return x and y coordinates.

```
def generate_dummy_circle_data(num_points):
    r = np.random.uniform(0,2,num_points)
    theta = np.random.uniform(0,2*np.pi,num_points)
    inner_circle = np.array([r*np.sin(theta), r*np.cos(theta)]).T

    r = np.random.uniform(5,7,num_points)
    theta = 2*np.pi*np.arange(num_points)/num_points
    outer_circle = np.array([r*np.sin(theta), r*np.cos(theta)]).T

    X = np.concatenate((inner_circle,outer_circle),axis=0)
    y = np.concatenate((np.ones(num_points),np.zeros(num_points)),axis=0)

    randIdx = np.arange(X.shape[0])
    np.random.shuffle(randIdx)

    X = X[randIdx]
    y = y[randIdx].astype(int)

    return X, y
```

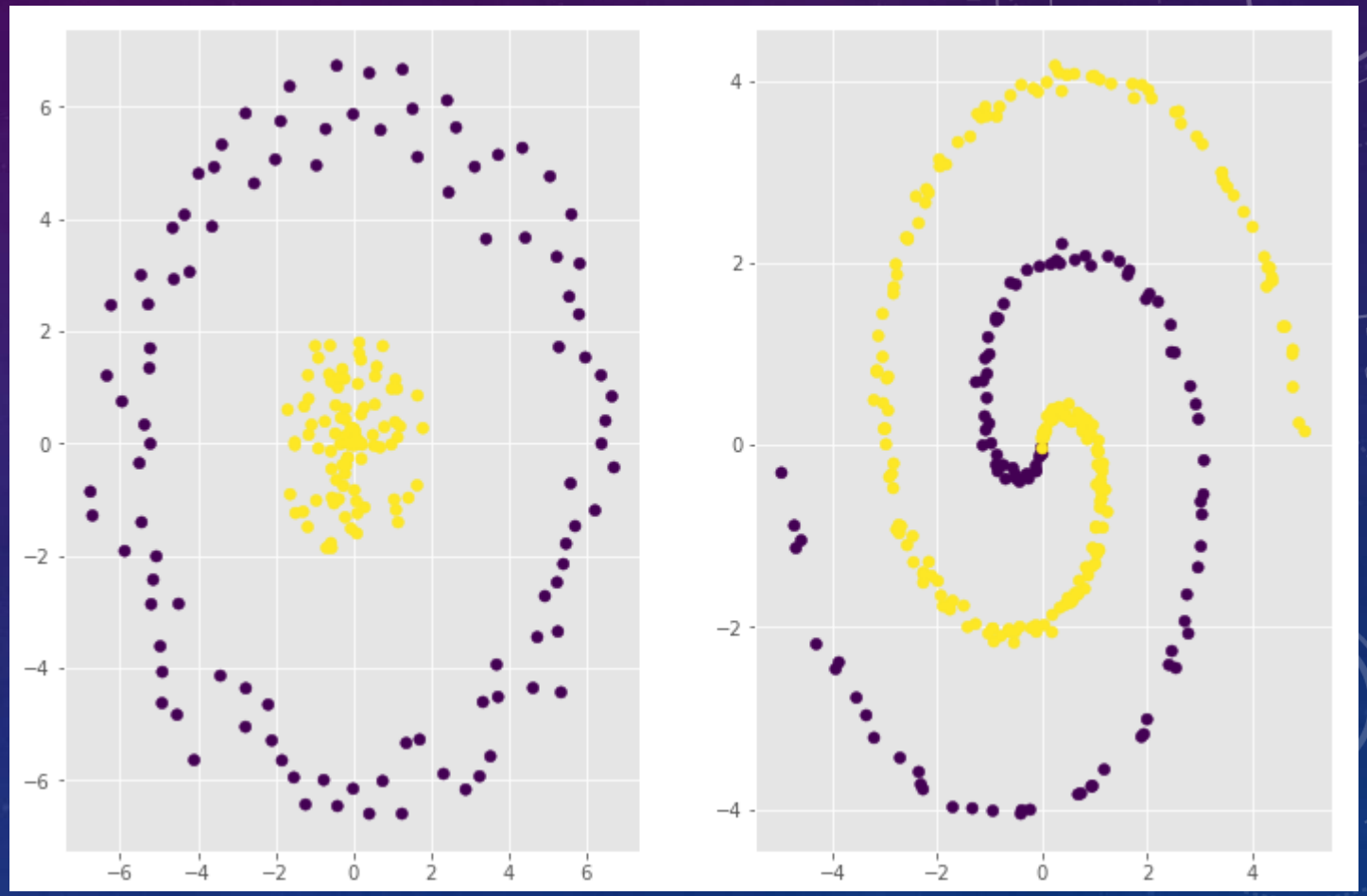


Uniform distribution

Example 2 – Code Details – 3 – Plotting the input data

To see our input data, we will use our created function and use the *plt.scatter()* function to plot the points.

```
X_circle,y_circle = generate_dummy_circle_data(100)
plt.subplot(121)
plt.scatter(X_circle[:,0],X_circle[:,1],c=y_circle)

X_spiral,y_spiral = generate_dummy_spiral_data(100,2)
plt.subplot(122)
plt.scatter(X_spiral[:,0],X_spiral[:,1],c=y_spiral)
```



Uniform distribution

Example 2 – Code Details – 4 – Neural Network

- We define several functions for forward and backward functions in our *Neural_Network.py* file. You can dig into this code later on. It is similar to the functions from pytorch that you are familiar with.
- For now remember: We are creating a Neural Network with 2 layers, relu activation, hidden layer size of 6 and three classes, because we have to classify between the yellow, purple and none of them.
- To check, we print out the parameters and the size of the layer. Here we consider i

```
1  np.random.seed(1)
2  net = NeuralNetwork(hidden_size=6,num_classes=3)
3  net.initialize_weights(input_dim=5)
4
5  for param in net.params:
6      print("Shape of",param, net.params[param].shape)
7
8  for param in net.params:
9      print(param, net.params[param])
```

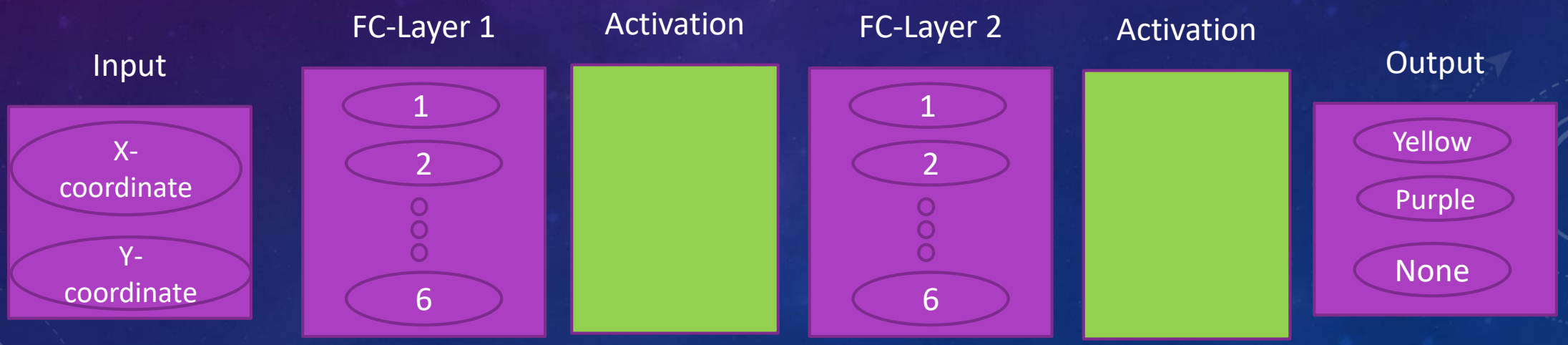
```
Shape of W1 (2, 6)
Shape of b1 (6,)
Shape of W2 (6, 3)
Shape of b2 (3,)
W1 [[ 0.01624345 -0.00611756 -0.00528172 -0.01072969  0.00865408 -0.02301539]
 [ 0.01744812 -0.00761207  0.00319039 -0.0024937   0.01462108 -0.02060141]]
b1 [0. 0. 0. 0. 0. 0.]
W2 [[-0.00322417 -0.00384054  0.01133769]
 [-0.01099891 -0.00172428 -0.00877858]
 [ 0.00042214  0.00582815 -0.01100619]
 [ 0.01144724  0.00901591  0.00502494]
 [ 0.00900856 -0.00683728 -0.0012289 ]
 [-0.00935769 -0.00267888  0.00530355]]
b2 [0. 0. 0.]
```

Layer 1

Layer 2

Network Architecture for the Toy Dataset

- Our Initialization is with small random values and biases are set to zero.
- We will analyze three different functions:
  - Sigmoid, tanh and Relu and also look how the Layer size affects the output.



Backward Pass of our FC-Layer

- In order to update the weights, we have to perform the backward pass. Therefore we numerically calculate the gradient with the *fully_connected_backward* () function in the *neural_networks.py* script
- Let's make sure, that the numerical approximation is not too far from the analytical one.
- As you can see in the results, the error is tiny.

```
np.random.seed(1)
net = NeuralNetwork()
W1 = np.random.randn(2,5)
b1 = np.random.randn(5)
dUpper = np.random.randn(5, 5)

out, cache = net.fully_connected_forward(X_circle[:5],W1, b1)

dX, dW, db = net.fully_connected_backward(dUpper,cache)

dX_num = compute_numerical_gradient(lambda X: np.sum(dUpper*net.fully_connected_forward(X, W1, b1)[0])
                            , X_circle[:5])

dW_num = compute_numerical_gradient(lambda W: np.sum(dUpper*net.fully_connected_forward(X_circle[:5], W, b1)[0])
                            , W1)

db_num = compute_numerical_gradient(lambda b: np.sum(dUpper*net.fully_connected_forward(X_circle[:5],W1, b)[0])
                            ,  b1)

print("Gradient dX Relative Error",relative_error(dX, dX_num))
print("Gradient dW Relative Error",relative_error(dW, dW_num))
print("Gradient db Relative Error",relative_error(db, db_num))
```

Result

```
Gradient dX Relative Error 1.635165242525734e-10
Gradient dW Relative Error 5.41618295512523e-09
Gradient db Relative Error 5.7091457389092345e-11
```

Forward and Backward Pass with Sigmoid

- Similarly, we also test our sigmoid activation function *net.sigmoid_forward()*

And the backward pass with *net.sigmoid_backward()*

```
np.random.seed(1)
net = NeuralNetwork()
out, cache = net.sigmoid_forward(np.random.randn(5,5))
print("Sigmoid Layer Output:")
print(out)
```

```
np.random.seed(1)
net = NeuralNetwork()
dUpper = np.random.randn(5, 5)
dummy_input = np.random.randn(5,5)
out, cache = net.sigmoid_forward(dummy_input)

dSigmoid = net.sigmoid_backward(dUpper,cache)

dSigmoid_num = compute_numerical_gradient(lambda X: np.sum(dUpper*net.sigmoid_forward(X)[0])
                                          , dummy_input)

print("Gradient dSigmoid Relative Error",relative_error(dSigmoid, dSigmoid_num))
```

Softmax cross entropy layer loss

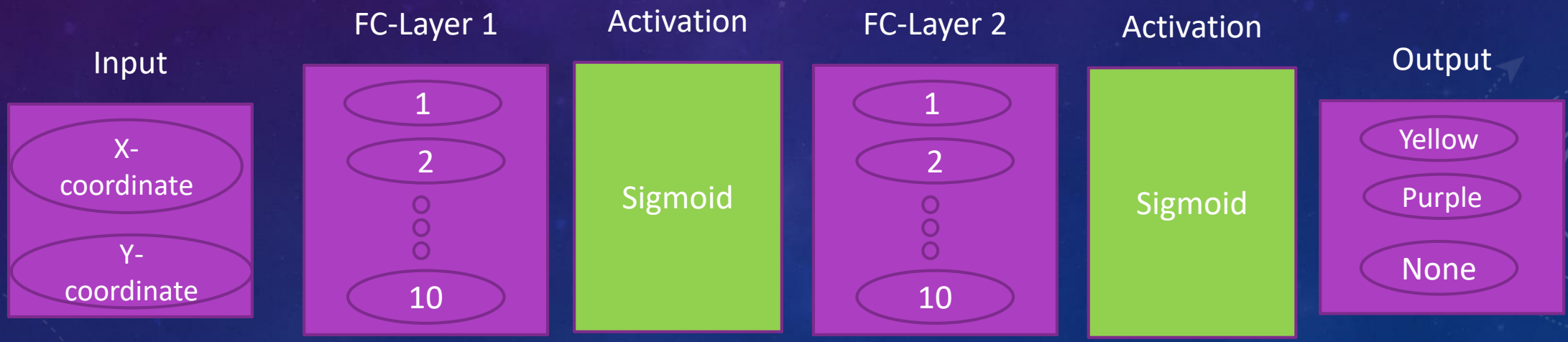
- Similarly, we also test our cross entropy loss function *net.softmax_corss_entropy_loss()*
- After we made sure, that all our functions are implemented correctly, and the results are complying with the Sanity results, we will now start with building a simple network in the next slides.

```
np.random.seed(1)
net = NeuralNetwork()
loss, dloss = net.softmax_cross_entropy_loss(np.random.randn(4,6),np.random.randint(0,6,size=4))
print("Softmax Cross-entropy Loss")
print(loss)
print()
print("Gradient of the loss with respect to the scores")
print(dloss)
```

Our Network

We set our network as described on the right. It has 2 hidden layers, 3 classes, 10 hidden neurons and uses the sigmoid activation function

```
np.random.seed(1)
net = NeuralNetwork(num_layers=2, num_classes=3, hidden_size=10, hidden_activation_fn="sigmoid")
net.initialize_weights(5,std_dev=0.1)

scores, cache_list  = net.network_forward(np.random.randn(5,5)*10)

print("Forward Pass:")
print(scores)
```



After setting up the Network, we train our Network

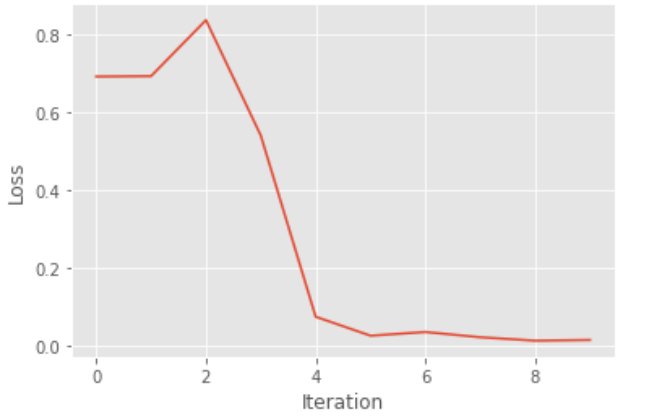
- Setup

```
np.random.seed(1)
X = X_circle
y = y_circle

net = NeuralNetwork(num_layers=2, num_classes=2, hidden_size=6, hidden_activation_fn="sigmoid")
loss_history = net.train(X, y, learning_rate=0.9, lambda_reg=0.0, num_iters=1000, batch_size=10, verbose=False)
```

- Plot the loss over the iterations
- And the performance in numbers

```
%matplotlib inline
plt.plot(loss_history)
plt.xlabel("Iteration")
plt.ylabel("Loss")
```

```
Text(0, 0.5, 'Loss')
```

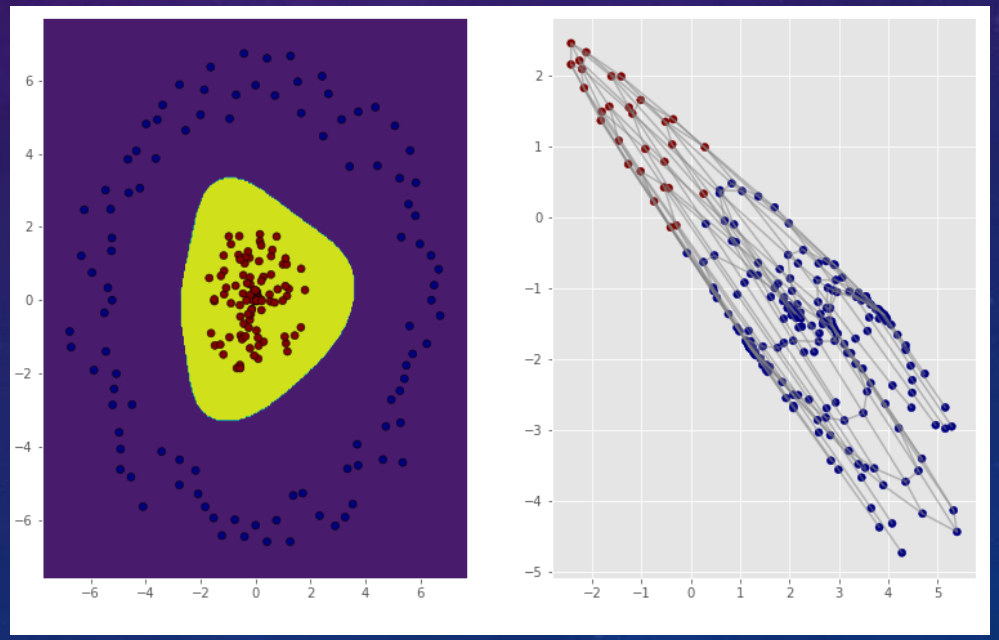



```
Y_train_pred = net.predict(X)
print("Train accuracy: {} %".format(np.mean(Y_train_pred == y) * 100))
```

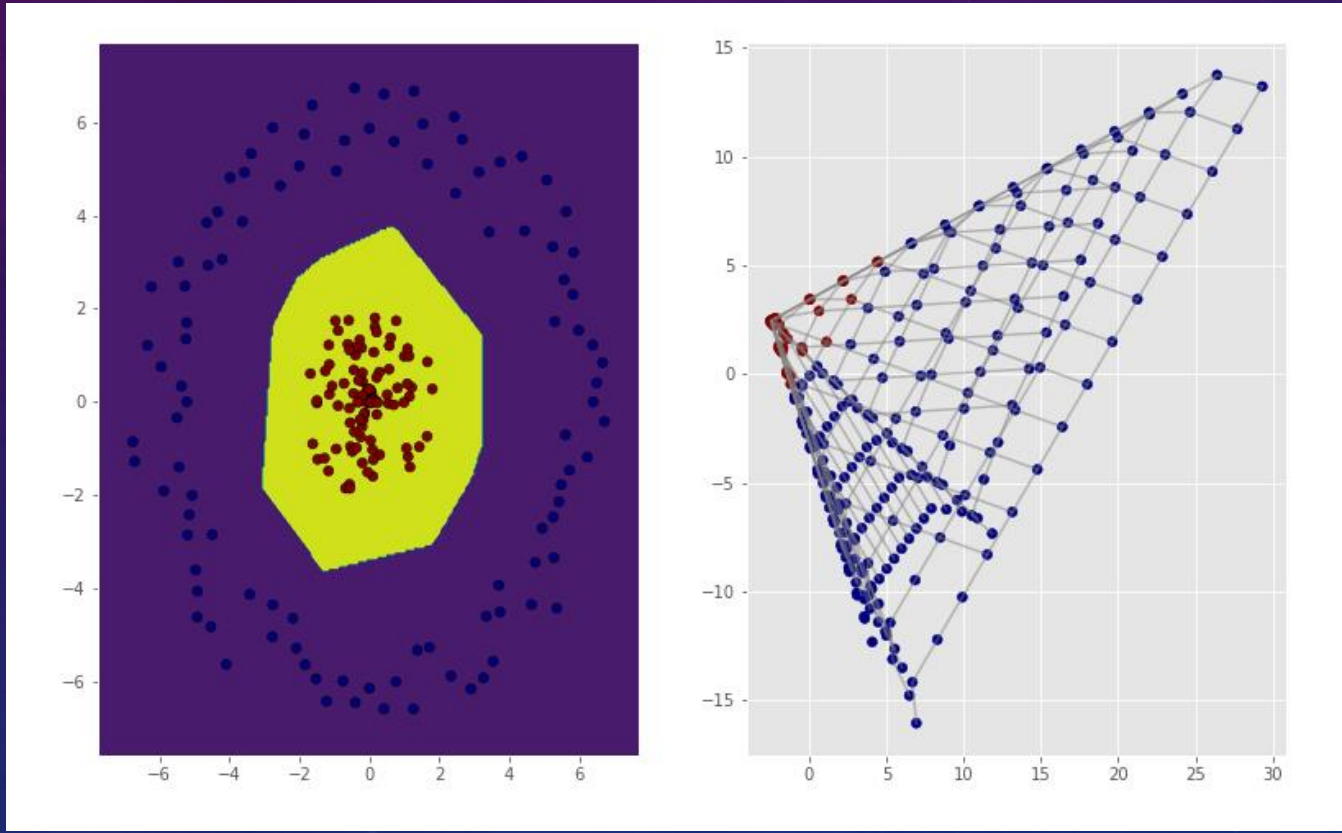
```
Train accuracy: 100.0 %
```

Now, after testing our functions, comes the interesting part:

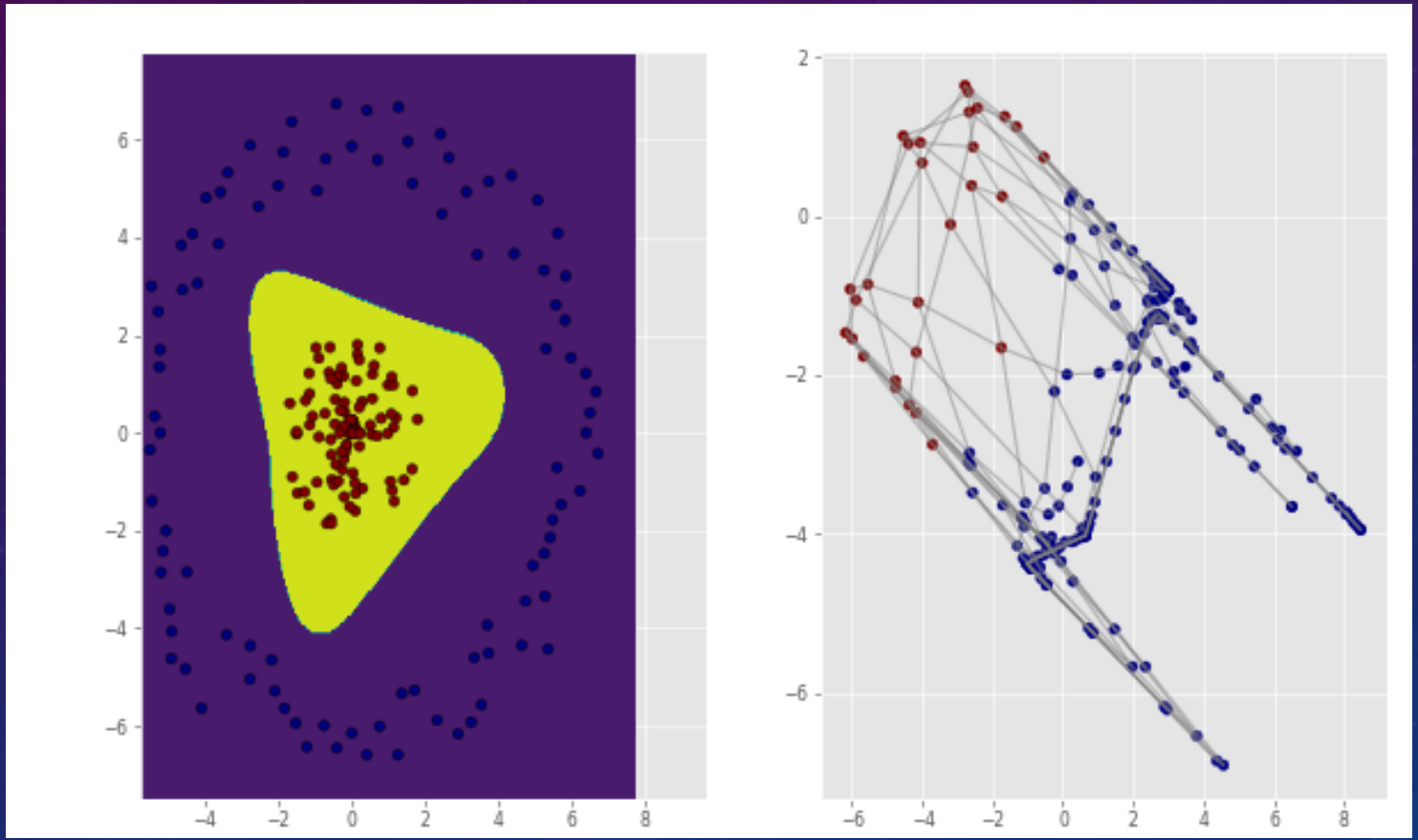
- In the next part we will visualize the training process over time.
- We will visualize the decision boundary(left) and the transformations(right) that the network learned during training
- You can run the script live and see how the network learns over our training period



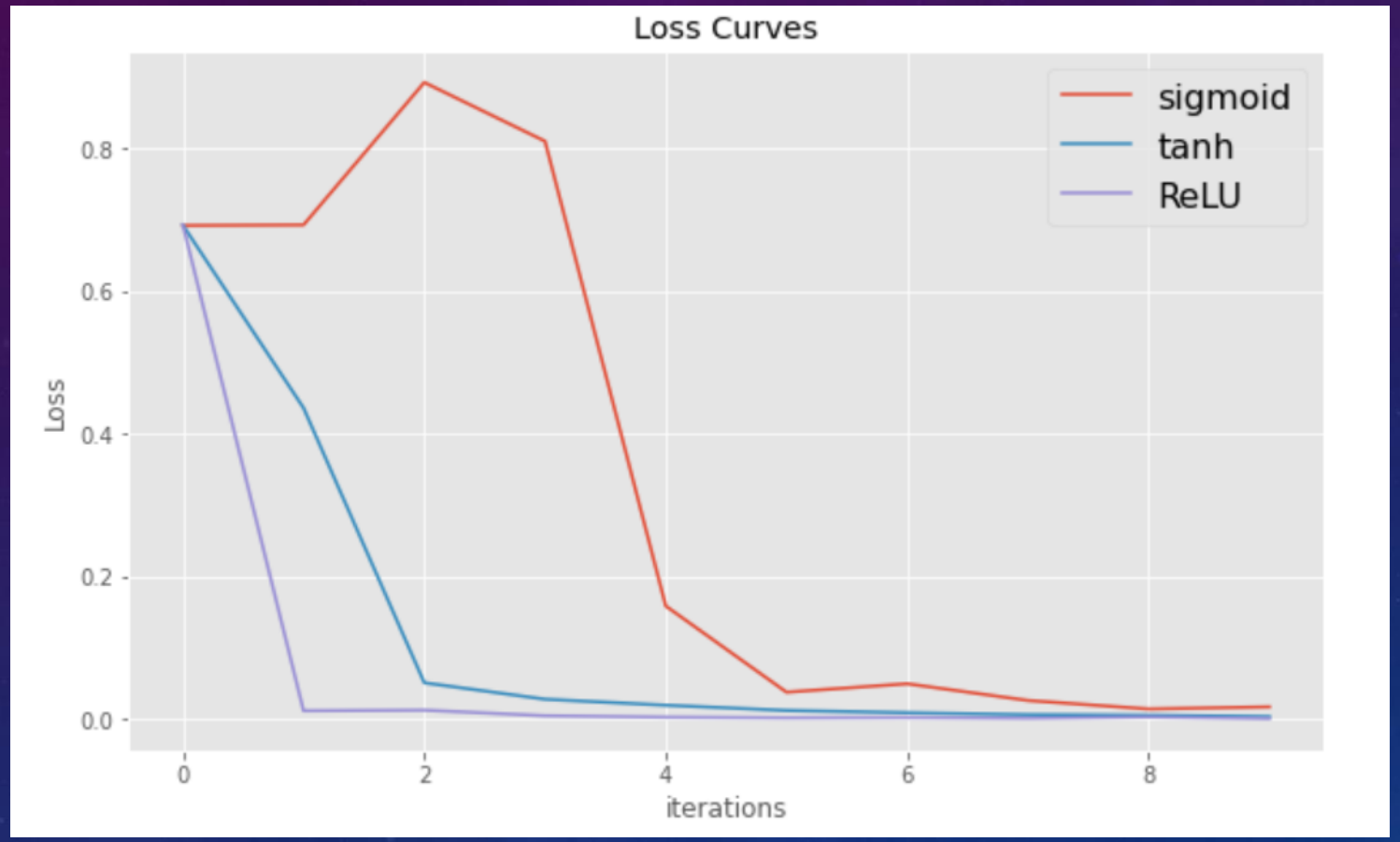
How does the training process look like with Relu Activations in comparison



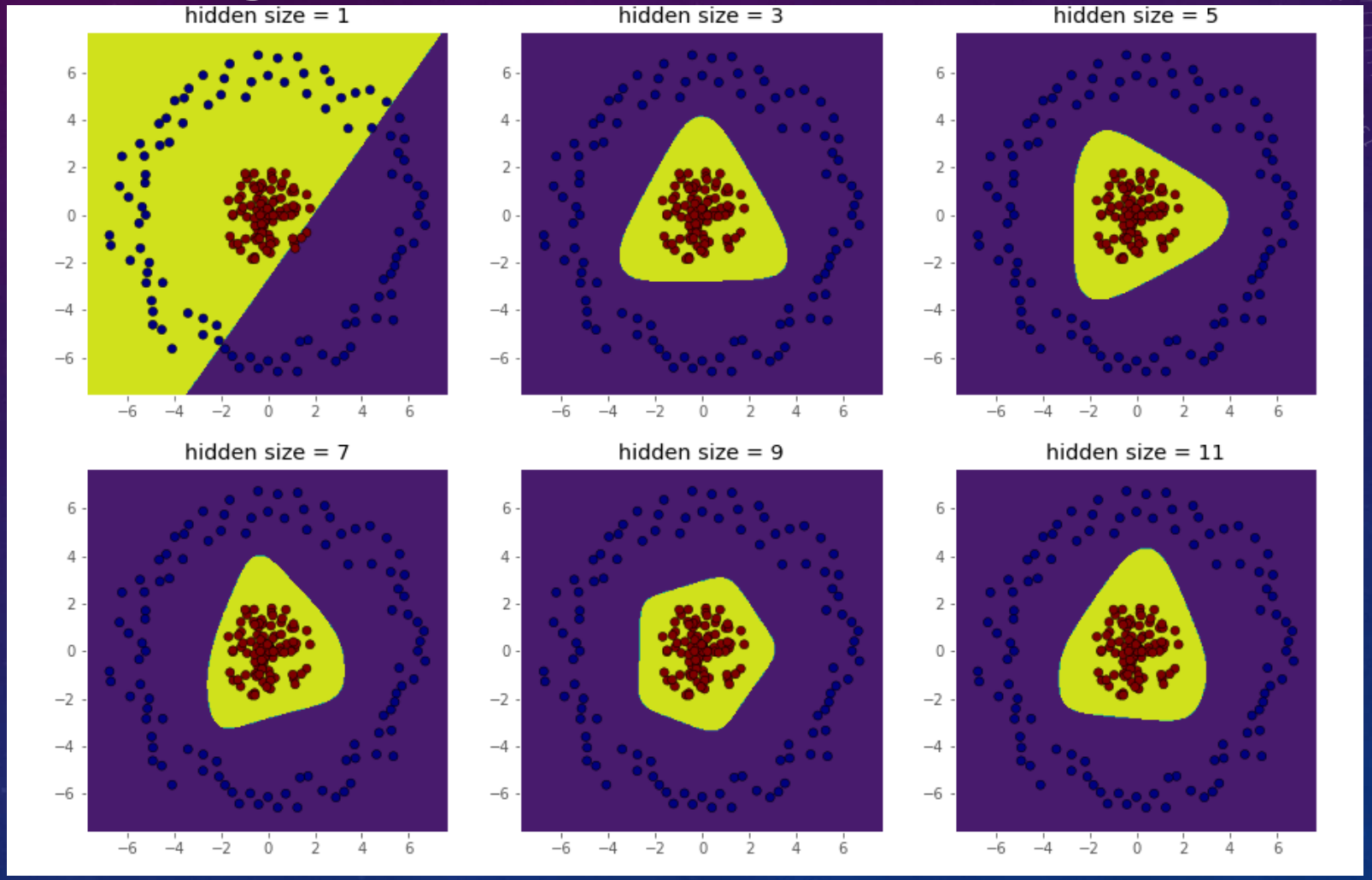
How does the training process look like with tanh Activations in comparison



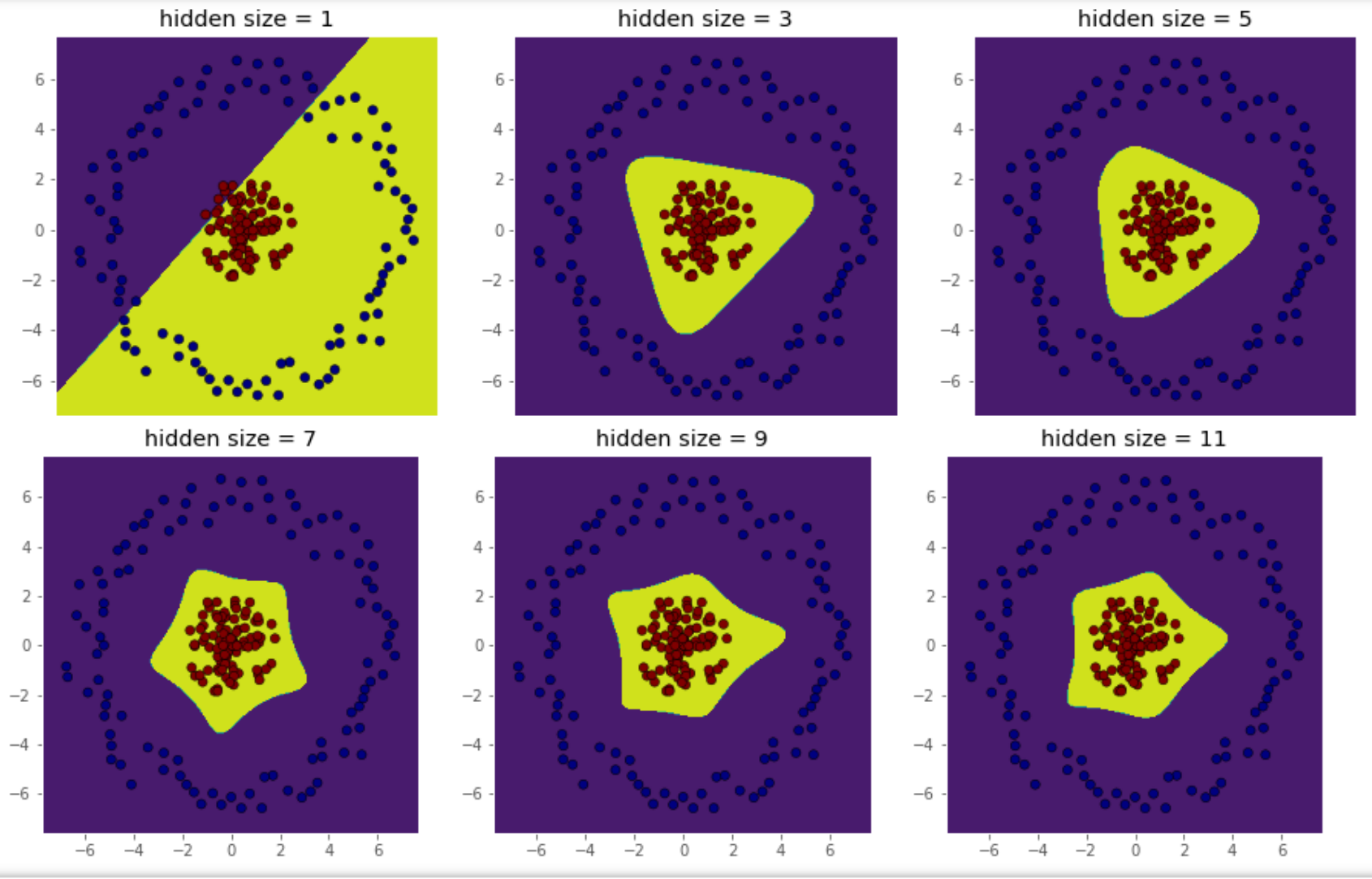
Comparing the Loss across the different activation functions for our Network



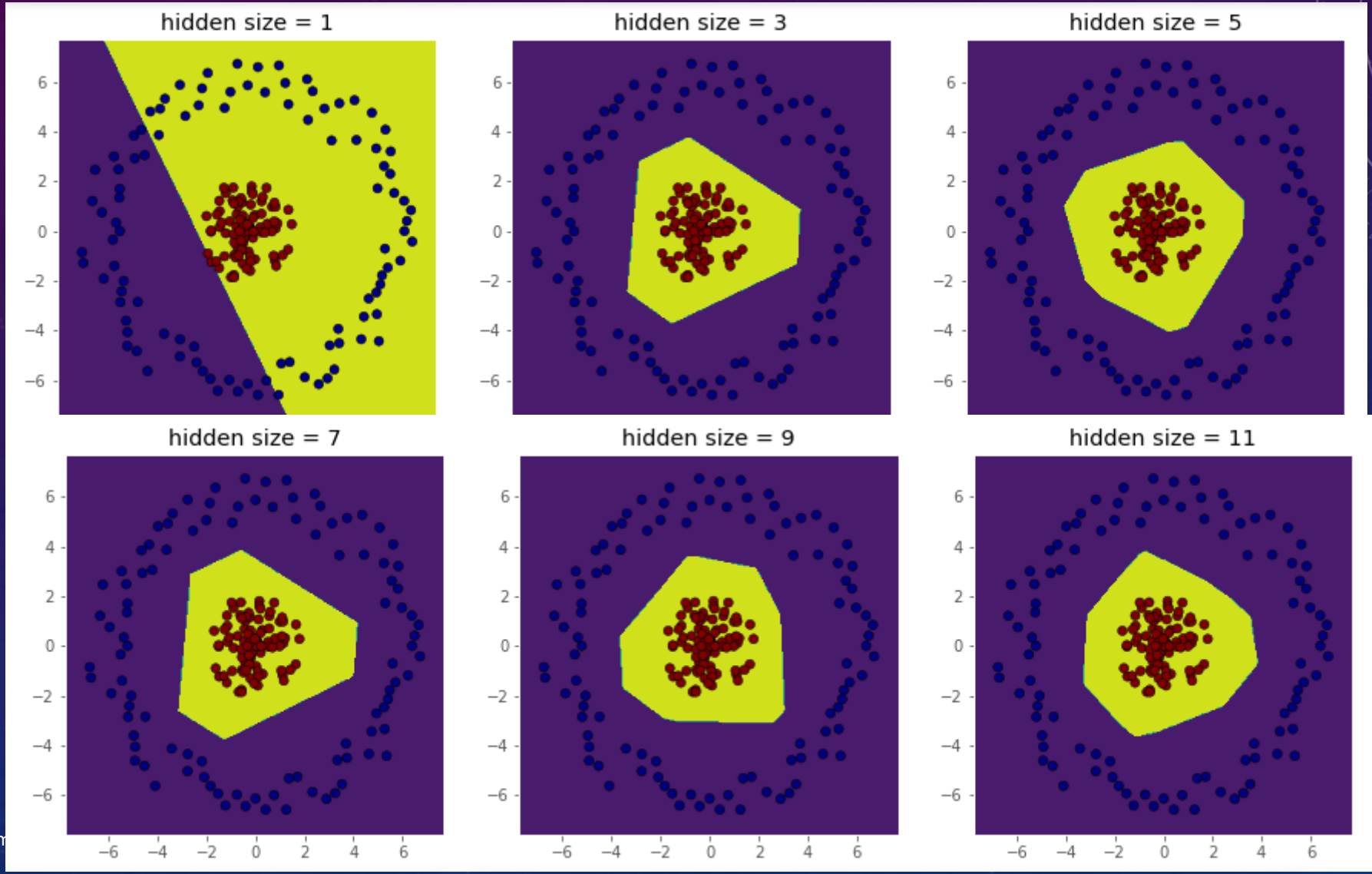
How does the size of the hidden layer effect our decision boundaries? - Sigmoid



How does the size of the hidden layer effect our decision boundaries? - Tanh

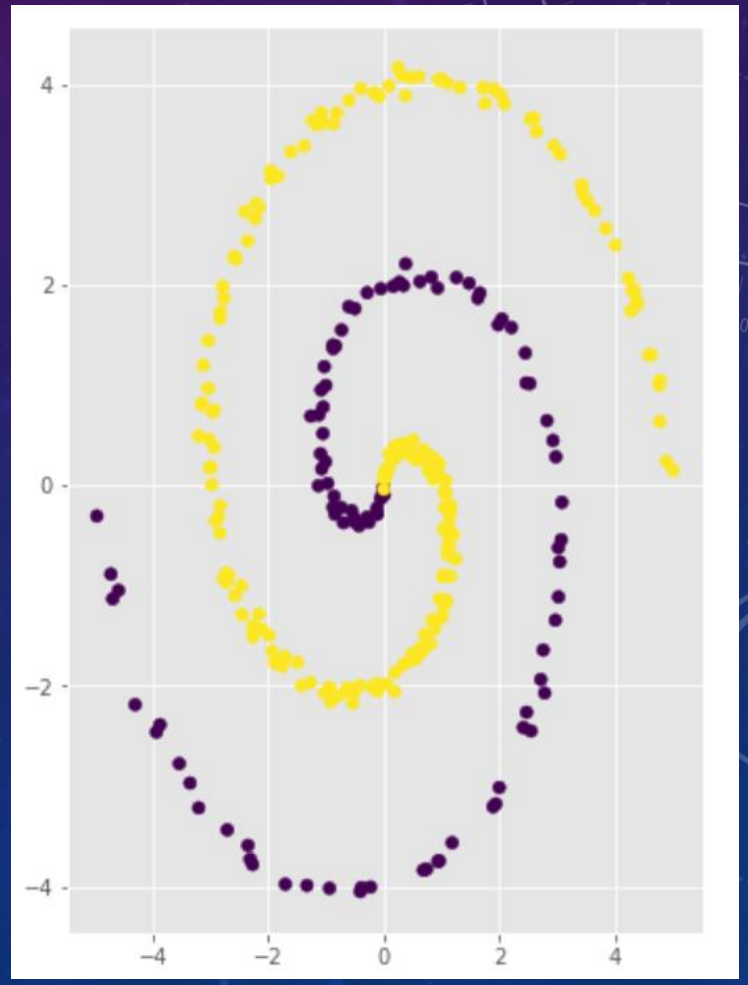


How does the size of the hidden layer effect our decision boundaries? - ReLU



Exercise 2-1 FCN Observe Training

- Now, we will use the spiral data. Please use the notebook, *Example2-1-FCN_Observe_Training.ipynb*, and run the same experiments for the spiral data. Make observations about the training and note them down
- You must modify the code in the following categories
  - Hidden layer size
  - Input data size
  - Number of layers
- Upload your observations, comments and your code to Moodle.



Colab

What Is Google Colab?

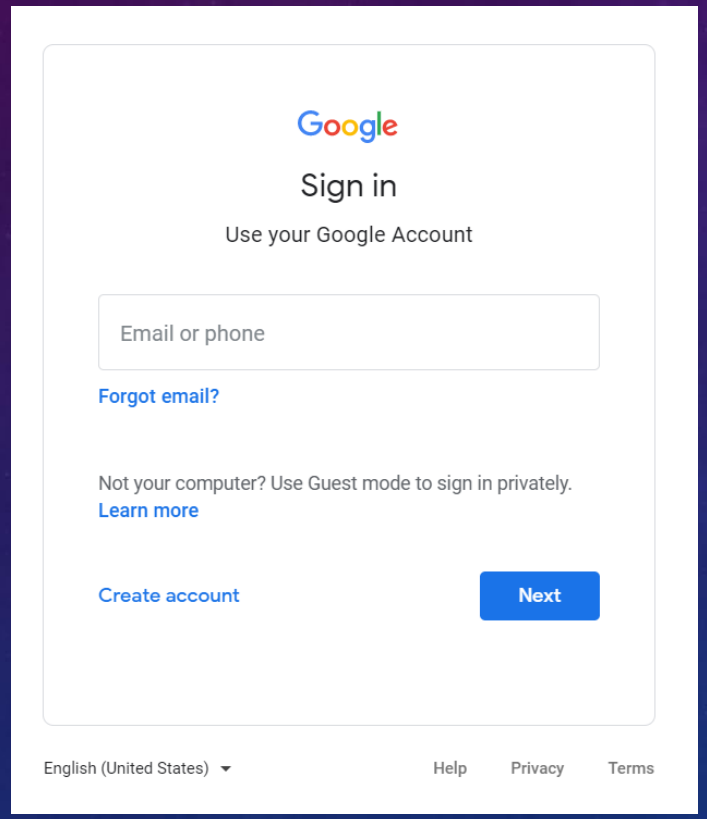
Google Colab is a free cloud service and now it supports free GPU!

You can:

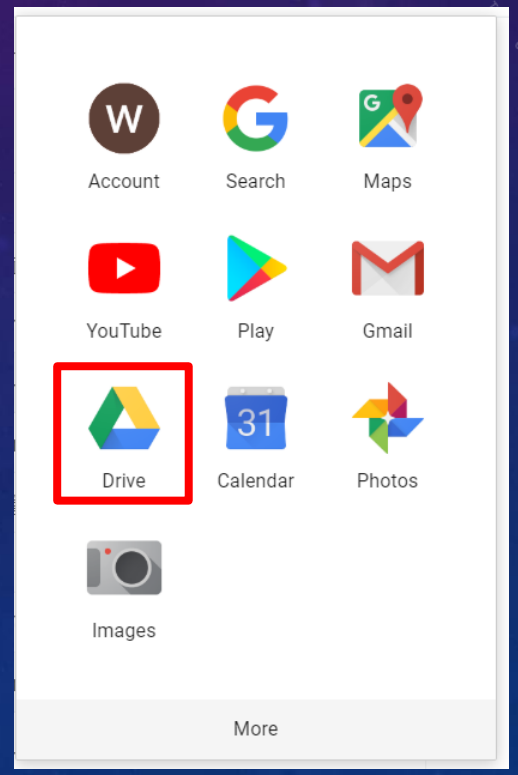
1. Improve your **Python** programming language coding skills.
2. Develop deep learning applications using popular libraries such as **Keras**, **TensorFlow**, **PyTorch,** and **OpenCV**.

Getting Google Colab Ready to Use

1. Sign in your Google account

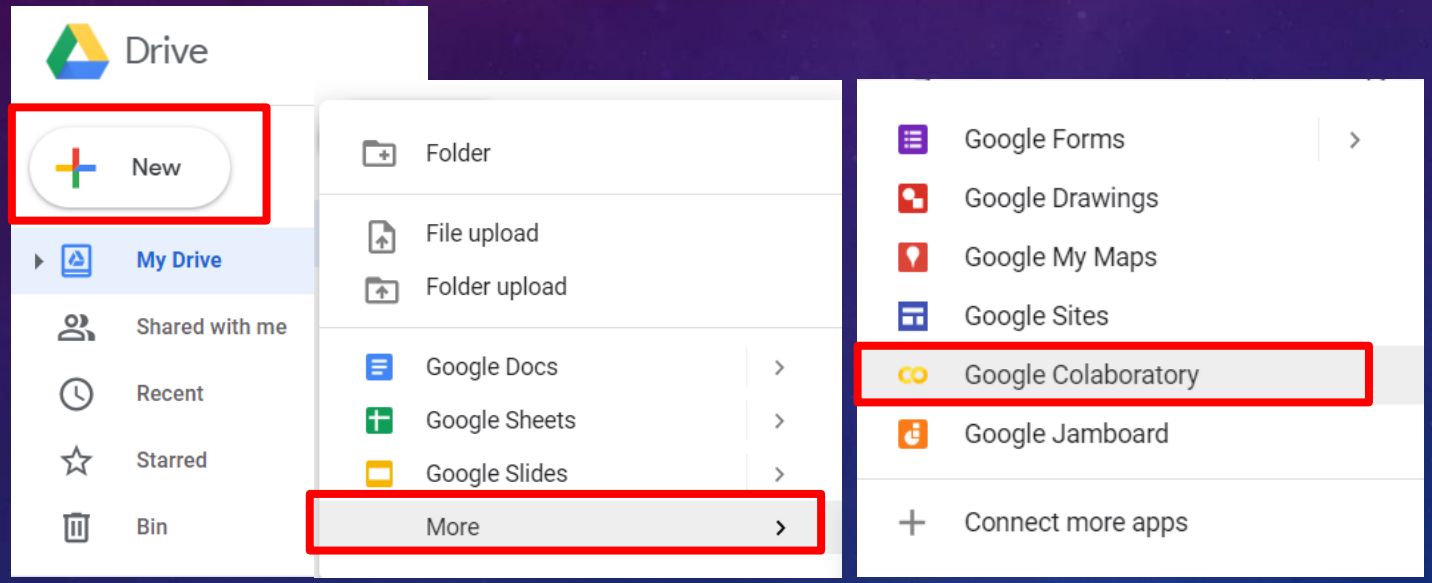


2. Open your Google drive

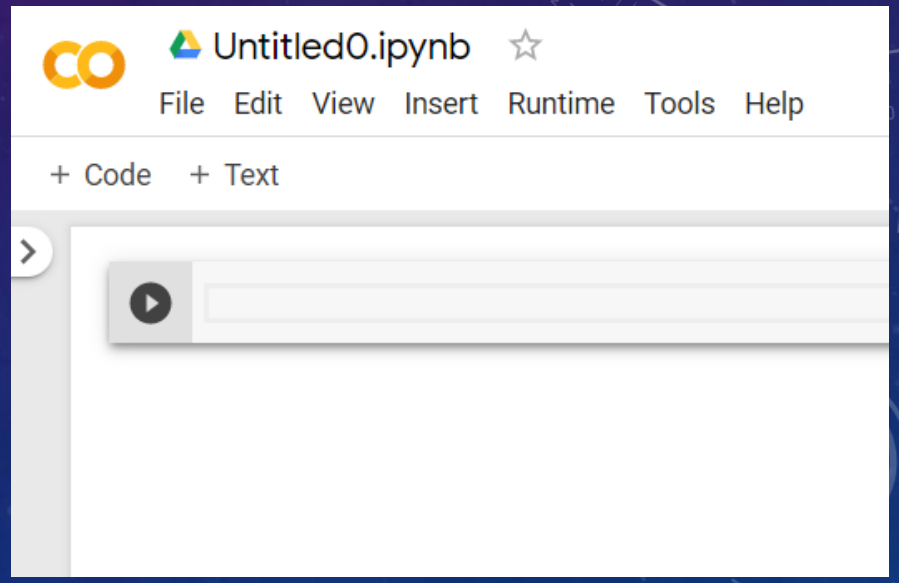


Create a new notebook

1. New > More > Colaboratory

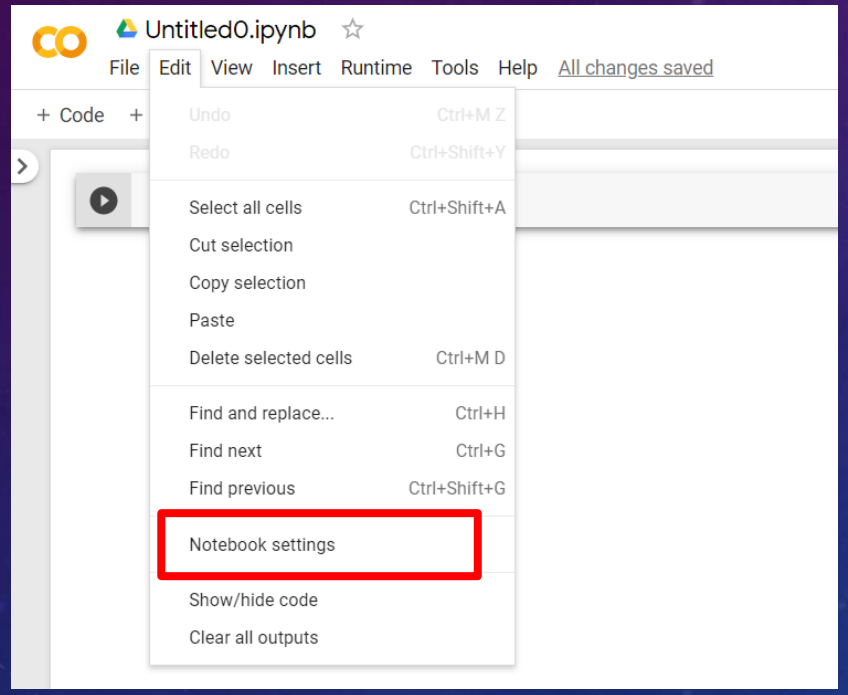


2. Create a new notebook.

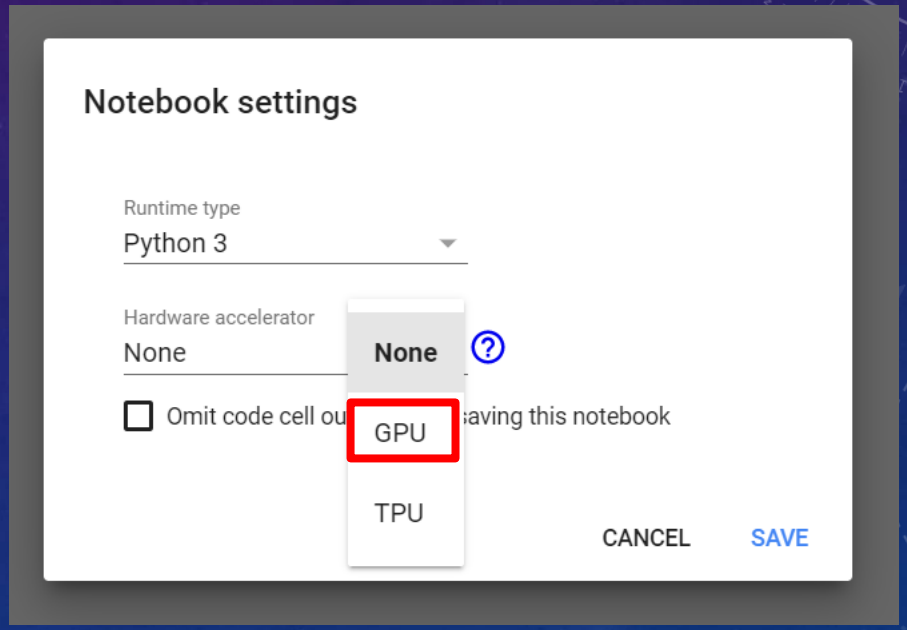


Setting Free GPU

1. Edit > Notebook settings



2. Hardware > GPU > SAVE



Comparison of Different Channel on Feature Map

Example 3: Comparison of Different Channel on Feature Map

- Training the networks of the different output channels and compare the difference in feature maps.

Example 3: Comparison of Different Channel on Feature Map

Loading and normalizing CIFAR10

Setting the datasets path

```
transform = transforms.Compose(
        [transforms.ToTensor(),
         transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)

trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                          shuffle=True, num_workers=8)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)

testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                         shuffle=False, num_workers=8)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

There are 10 classes in CIFAR10

Normalize a tensor image with mean and standard deviation. Given mean: (mean[1],...,mean[n]) and std: (std[1],..,std[n]) for n channels, this transform will normalize each channel of the input torch.*Tensor i.e., output[channel] = (input[channel] - mean[channel]) / std[channel]

Example 3: Comparison of Different Channel on Feature Map

Define network(vgg16)

```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()

        # calculate same padding:
        # (w - k + 2*p)/s + 1 = o
        # => p = (s(o-1) - w + k)/2

        self.feature = nn.Sequential(
            nn.Conv2d(in_channels=3,
                      out_channels=64,
                      kernel_size=(3, 3),
                      stride=(1, 1),
                      # (1(32-1)- 32 + 3)/2 = 1
                      padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.Conv2d(in_channels=64,
                      out_channels=64,
                      kernel_size=(3, 3),
                      stride=(1, 1),
                      padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(2, 2),
                         stride=(2, 2)),
```

You can change the output channel by yourself

```
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(2, 2),
                         stride=(2, 2))
        )

        self.classifier = nn.Sequential(
            nn.Linear(2048, 4096),
            nn.ReLU(True),
            nn.Dropout(p=0.65),
            nn.Linear(4096, 4096),
            nn.ReLU(True),
            nn.Dropout(p=0.65),
            nn.Linear(4096, 10),
        )

        for m in self.modules():
            if isinstance(m, torch.nn.Conv2d) or isinstance(m, torch.nn.Linear):
                nn.init.kaiming_uniform_(m.weight, mode='fan_in', nonlinearity='leaky_relu')

                if m.bias is not None:
                    m.bias.detach().zero_()
```

```
    def forward(self, x):

        x = self.feature(x)

        x = x.view(x.size(0), -1)
        x = self.classifier(x)

        return x
```

Define forward path

Example 3: Comparison of Different Channel on Feature Map

Define a Loss function and optimizer

Select Cross-Entropy loss for classification

```
criterion   =   nn.CrossEntropyLoss()
optimizer   =   optim.Adam(net.parameters(),   lr=0.001)
```

Using Adam optimizer

Example 3: Comparison of Different Channel on Feature Map

Train the network and save the checkpoint

```
for epoch in range(1):    # loop over the dataset multiple times

        running_loss = 0.0
        for i, data in enumerate(trainloader, 0):
            # get the inputs; data is a list of [inputs, labels]
            inputs, labels = data
            inputs, labels = inputs.to(device), labels.to(device)

            # zero the parameter gradients
            optimizer.zero_grad()

            # forward + backward + optimize
            outputs = net(inputs)

            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            # print statistics
            running_loss += loss.item()
            if i % 2000 == 1999:        # print every 2000 mini-batches

                    print("Epoch : {} steps : {} Training Loss : {}".format(epoch + 1, i + 1, running_loss / 2000) )
                    running_loss = 0.0
    save_checkpoint({'net':net.state_dict()}, 'test epoch{}'.format(epoch+1))

print('Finished Training')
```

Define the epoch(1)

Backpropagation the loss to update the weights.

Save the checkpoint

Example 3: Comparison of Different Channel on Feature Map

Compare the feature map and vector

```
myClass=FeatureVisualization('./jellyfish.jpg',2)
Compare=FeatureVisualization('./car.jpg',2)
print(myClass.pretrained_model2)

myClass.save_feature_to_img1()
Compare.save_feature_to_img2()
#  print("The  first  picture  classification  predict:")
myClass_vector  =  myClass.predict()
#  print("The  second  picture  classification  predict:")
Compare_vector  =  Compare.predict()
#Define  cosine  similarity
cos=  nn.CosineSimilarity(dim=1)
#Define  Euclidean  distance
euclidean_dist  =  torch.dist(myClass_vector,Compare_vector,p=2)
cosine_dist  =  cos(myClass_vector,Compare_vector)
print("Verification:")
if  cosine_dist  >  0.4:
        print("They  are  the  same!")
        print("Their  cosine_similarity:{}".format(cosine_dist))
else:
        print("They  are  not  the  same!")
        print("Their  cosine_similarity:{}".format(cosine_dist))

print("Their  euclidean_dist:{}".format(euclidean_dist))
```

Save the feature maps

Calculate the Euclidean distance between different pictures

Calculate the Cosine distance between different pictures

Define the threshold

Exercise 3-1: Comparison of Different Channel on Feature Map

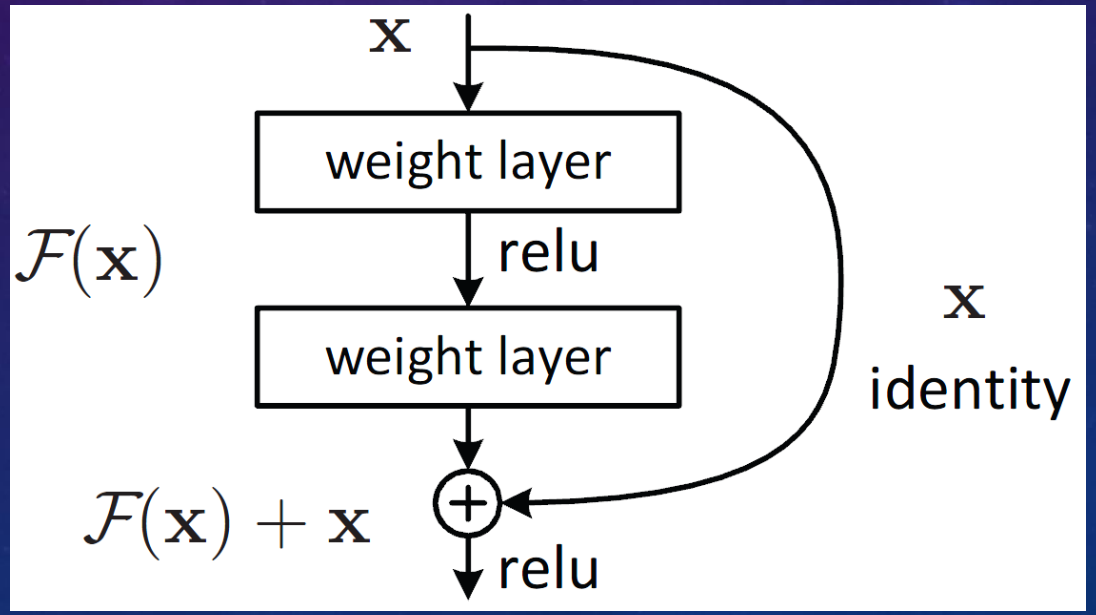
- Please download 3-1 Comparison of Different Channel on Feature Map.ipynb from moodle
- Change the output channel of the same layer, observe the feature map is different, and increase the training epoch(choosing images from Internet)

Upload your observations, comments and your code to Moodle in a docx.

Comparison of Deep Networks

ResNet

- Since AlexNet, the state-of-the-art CNN architecture is going deeper and deeper.
- However, increasing network depth does not work by simply stacking layers together. Deep networks are hard to train because of the notorious vanishing gradient problem
- The core idea of ResNet is introducing a so-called “identity shortcut connection” that skips one or more layers, as shown in the following figure:



Why ResNets Work

https://www.youtube.com/watch?v=RYth6EbBUqM&ab_channel=Deeplearning.ai[9:12]

Example 4: Comparison of Deep Networks

- Considering the following networks
  - N_1 has 10 conv layers and 1 Fc layer with Setup-1, please train N_1 on the CIFAR-10 for 3 epochs;
  - N_2 has 10 conv layers with specified shortcut connections and 1 Fc layer with Setup-2, please train N_2 on the CIFAR-10 for 3 epochs;

Example 4: Comparison of Deep Networks

Loading and normalizing CIFAR10

Setting the datasets path

```
transform = transforms.Compose(
        [transforms.ToTensor(),
         transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)

trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                          shuffle=True, num_workers=8)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)

testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                         shuffle=False, num_workers=8)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

There are 10 classes in CIFAR10

Normalize a tensor image with mean and standard deviation. Given mean: (mean[1],...,mean[n]) and std: (std[1],..,std[n]) for n channels, this transform will normalize each channel of the input torch.*Tensor i.e., output[channel] = (input[channel] - mean[channel]) / std[channel]

Example 4: Comparison of Deep Networks

Setup-1 Define network(without shortcut)(10conv 1fc)

```
net = Net().to(device)
```

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(),)
        self.conv2 = nn.Sequential(
            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(),)
```

```
        self.conv6 = nn.Sequential(
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(128),
            nn.ReLU(),)
        self.conv7 = nn.Sequential(
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(128),
            nn.ReLU(),)
```

```
    self.conv10 = nn.Sequential(
        nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1, bias=False),
        nn.BatchNorm2d(128),
        nn.ReLU(),)
self.fc1 = nn.Linear(131072, 10)
```

```
def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.conv4(x)
        x = self.conv5(x)
        x = self.conv6(x)
        x = self.conv7(x)
        x = self.conv8(x)
        x = self.conv9(x)
        x = self.conv10(x)

        x = x.view(-1, 131072)
        x = F.relu(self.fc1(x))

        return x
```

Define forward path

Example 4: Comparison of Deep Networks

Setup-1 Print network(without shortcut)(10conv 1fc)

```
Net(
  (conv1): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (conv2): Sequential(
    (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (conv3): Sequential(
    (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (conv4): Sequential(
    (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (conv5): Sequential(
    (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (conv6): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (conv7): Sequential(
    (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (conv8): Sequential(
    (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (conv9): Sequential(
    (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (conv10): Sequential(
    (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (fc1): Linear(in_features=131072, out_features=10, bias=True)
```

Example 4: Deep Network Comparison

Setup-2 Define network(with shortcut)(10conv 1fc)

```
net = ResNet18().to(device)
```

```
class ResidualBlock(nn.Module):
    def __init__(self, inchannel, outchannel, stride=1):
        super(ResidualBlock, self).__init__()
        self.left = nn.Sequential(
            nn.Conv2d(inchannel, outchannel, kernel_size=3, stride=stride, padding=1, bias=False),
            nn.BatchNorm2d(outchannel),
            nn.ReLU(inplace=True),
            nn.Conv2d(outchannel, outchannel, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(outchannel)
        )
        self.shortcut = nn.Sequential()
        if stride != 1 or inchannel != outchannel:
            self.shortcut = nn.Sequential(
                nn.Conv2d(inchannel, outchannel, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(outchannel)
            )

    def forward(self, x):
        out = self.left(x)
        out += self.shortcut(x)
        out = F.relu(out)
        return out
```

Define shortcut

```
def ResNet18():
    return ResNet(ResidualBlock)
```

```
class ResNet(nn.Module):
    def __init__(self, ResidualBlock, num_classes=10):
        super(ResNet, self).__init__()
        self.inchannel = 64
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(),
        )
        self.layer1 = self.make_layer(ResidualBlock, 64, 2, stride=1)
        self.layer2 = self.make_layer(ResidualBlock, 128, 2, stride=2)
        #self.layer3 = self.make_layer(ResidualBlock, 256, 2, stride=2)
        #self.layer4 = self.make_layer(ResidualBlock, 512, 2, stride=2)
        self.fc = nn.Linear(2048, num_classes)

    def make_layer(self, block, channels, num_blocks, stride):
        strides = [stride] + [1] * (num_blocks - 1)      #strides=[1,1]
        layers = []
        for stride in strides:
            layers.append(block(self.inchannel, channels, stride))
            self.inchannel = channels
        return nn.Sequential(*layers)
```

```
    def forward(self, x):
        out = self.conv1(x)
        out = self.layer1(out)
        out = self.layer2(out)
        #out = self.layer3(out)
        #out = self.layer4(out)
        out = F.avg_pool2d(out, 4)
        out = out.view(out.size(0), -1)
        out = self.fc(out)
        return out
```

Define forward path

Example 4: Deep Network Comparison

Setup-2 Print network(with shortcut)(10conv 1fc)

```
ResNet(
  (conv1): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (layer1): Sequential(
    (0): ResidualBlock(
      (left): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (shortcut): Sequential()
    )
    (1): ResidualBlock(
      (left): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (shortcut): Sequential()
    )
  )
  (layer2): Sequential(
    (0): ResidualBlock(
      (left): Sequential(
        (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (shortcut): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): ResidualBlock(
      (left): Sequential(
        (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (shortcut): Sequential()
    )
  )
  (fc): Linear(in_features=2048, out_features=10, bias=True)
```

shortcut

Example 4: Comparison of Deep Networks

Define a Loss function and optimizer

Select Cross-Entropy loss for classification

```
criterion   =   nn.CrossEntropyLoss()
optimizer   =   optim.Adam(net.parameters(),   lr=0.001)
```

Using Adam optimizer

Example 4: Comparison of Deep Networks

Train the network and save the checkpoint

```
for epoch in range(3):  # loop over the dataset multiple times

        running_loss = 0.0
        for i, data in enumerate(trainloader, 0):
            # get the inputs; data is a list of [inputs, labels]
            inputs, labels = data
            inputs, labels = inputs.to(device), labels.to(device)

            # zero the parameter gradients
            optimizer.zero_grad()

            # forward + backward + optimize
            outputs = net(inputs)

            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            # print statistics
            running_loss += loss.item()
            if i % 2000 == 1999:       # print every 2000 mini-batches

                print("Epoch : {} steps : {} Training Loss : {}".format(epoch + 1, i + 1, running_loss / 2000) )
                running_loss = 0.0
    save_checkpoint({'net':net.state_dict()}, 'test_epoch{}'.format(epoch+1))

print('Finished Training')
```

Define the epoch

Backpropagation the loss to update the weights.

Save the checkpoint

Example 4: Comparison of Deep Networks

Test the network on the test data

```
correct = 0
total = 0
with torch.no_grad():
        for data in testloader:
                images, labels = data
                images, labels =images.to(device), labels.to(device)
                outputs = net(images)

                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

print('Accuracy : %d %%' % (100 * correct / total))
```

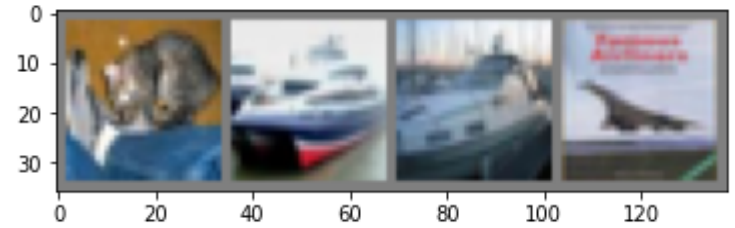
In the testing stage, the weights are fixed.

Check if predicted is the same as the labeled (G.T.)

Example 4: Comparison of Deep Networks

Result(with shortcut)

```
Epoch : 1 steps : 2000 Training Loss : 2.314868042707443
Epoch : 1 steps : 4000 Training Loss : 2.3025851249694824
Epoch : 1 steps : 6000 Training Loss : 2.3025851249694824
Epoch : 1 steps : 8000 Training Loss : 2.3025851249694824
Epoch : 1 steps : 10000 Training Loss : 2.3025851249694824
Epoch : 1 steps : 12000 Training Loss : 2.3025851249694824
Epoch : 2 steps : 2000 Training Loss : 2.3025851249694824
Epoch : 2 steps : 4000 Training Loss : 2.3025851249694824
Epoch : 2 steps : 6000 Training Loss : 2.3025851249694824
Epoch : 2 steps : 8000 Training Loss : 2.3025851249694824
Epoch : 2 steps : 10000 Training Loss : 2.3025851249694824
Epoch : 2 steps : 12000 Training Loss : 2.3025851249694824
Epoch : 3 steps : 2000 Training Loss : 2.3025851249694824
Epoch : 3 steps : 4000 Training Loss : 2.3025851249694824
Epoch : 3 steps : 6000 Training Loss : 2.3025851249694824
Epoch : 3 steps : 8000 Training Loss : 2.3025851249694824
Epoch : 3 steps : 10000 Training Loss : 2.3025851249694824
Epoch : 3 steps : 12000 Training Loss : 2.3025851249694824
Finished Training
```




```
GroundTruth:    cat  ship  ship plane
Predicted:  horse horse horse horse
Accuracy : 10 %
```

Result(without shortcut)

```
Epoch : 1 steps : 2000 Training Loss : 1.8878891510665416
Epoch : 1 steps : 4000 Training Loss : 1.4781636514812708
Epoch : 1 steps : 6000 Training Loss : 1.2983864627033472
Epoch : 1 steps : 8000 Training Loss : 1.1348232387583703
Epoch : 1 steps : 10000 Training Loss : 1.0744273342452944
Epoch : 1 steps : 12000 Training Loss : 0.9819176591066644
Epoch : 2 steps : 2000 Training Loss : 0.8823310074708425
Epoch : 2 steps : 4000 Training Loss : 0.8428827857617289
Epoch : 2 steps : 6000 Training Loss : 0.8335133502772077
Epoch : 2 steps : 8000 Training Loss : 0.8076976113315905
Epoch : 2 steps : 10000 Training Loss : 0.7748700085091405
Epoch : 2 steps : 12000 Training Loss : 0.7552551930428016
Epoch : 3 steps : 2000 Training Loss : 0.6636658706957823
Epoch : 3 steps : 4000 Training Loss : 0.6618142743564677
Epoch : 3 steps : 6000 Training Loss : 0.6509611685594427
Epoch : 3 steps : 8000 Training Loss : 0.6368713211108697
Epoch : 3 steps : 10000 Training Loss : 0.650435102526215
Epoch : 3 steps : 12000 Training Loss : 0.6216317716715858
Finished Training
```


```
GroundTruth:    cat  ship  ship plane
Predicted:    dog  ship   car plane
Accuracy : 76 %
```

Exercise 4-1: Comparison of Deep Networks

- Please download 4-1 Deep network comparsion.ipynb from moodle
- Modify the **Net's** architecture and make its gradient not disappear (can be deleted layer, or change the components)

Upload your observations, comments and your code to Moodle in a docx.