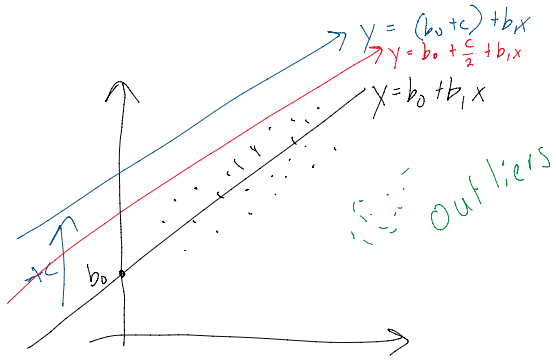
20211103 Neural Networks

Wednesday, November 3, 2021 3:47 PM

- HW2
  - For the first question, there are some points you are supposed to plot
  - Use logistic and linear regression to separate points



HW2 Question 1:

- If you don't generate the points this way, then you won't get a logical answer that will give us a line
  - So just making one or two blobs is not sufficient
    - They won't take away "that many points"
- Fit the line on all the points, ignoring the label

- He does have office hours tonight, email with a few suggested times if needed

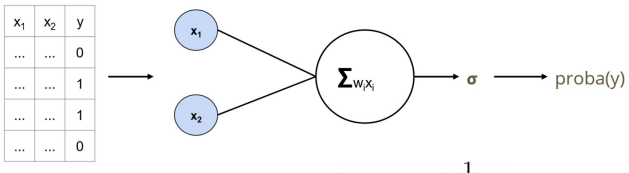
- Next Monday: no lecture during regular time. Can either attend live or watch the recording Monday
  - A1 class: 12:20pm???? on Monday

Neural Networks

Boston University CS 506 - Lance Galletti

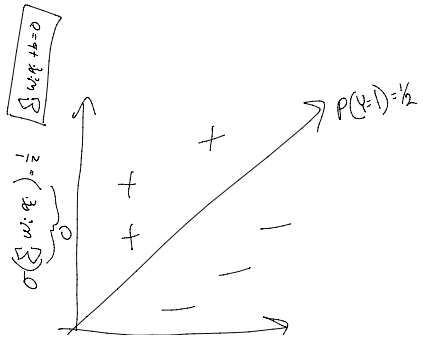
-

Logistic Regression Re-Revisited



where $\sigma(x) = \frac{1}{1+e^{-x}}$

- With logisitic regression, we can split this up into a number of steps
- Sigmoid/logistic function

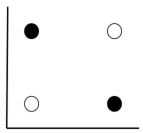


Logistic Regression Re-Revisited

Logistic Regression Re-Revisited

XOR function

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

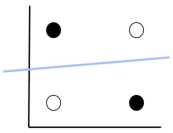


- Why is decision boundary in logistic regression a line?
- Can't split up XOR function with single line

Logistic Regression Re-Revisited

XOR function

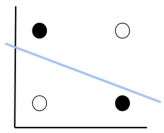
x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0



Logistic Regression Re-Revisited

XOR function

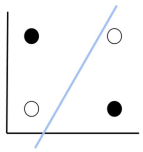
x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0



Logistic Regression Re-Revisited

XOR function

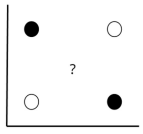
x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0



Logistic Regression Re-Revisited

XOR function

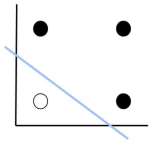
x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0



Logistic Regression Re-Revisited

Recall, the **OR** function is linearly separable:

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	1



Logistic Regression Re-Revisited

XOR(x_1, x_2) = **OR**(**AND**($x_1 = 0$, $x_2 = 1$), **AND**($x_1 = 1$, $x_2 = 0$))

= ($x_1 = 0 \wedge x_2 = 1$) $\vee$ ($x_1 = 1 \wedge x_2 = 0$)

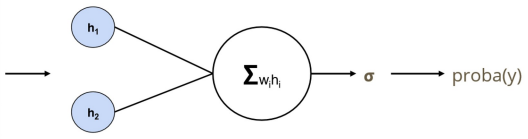
= $h_1 \vee h_2$

x_1	x_2	h_1	h_2	y
0	0	0	0	0
1	0	0	1	1
0	1	1	0	1
1	1	0	0	0

- Want to automatically learn our features to separate our classes

Logistic Regression Re-Revisited

h_1	h_2	y
0	0	0
0	1	1
1	0	1
0	0	0

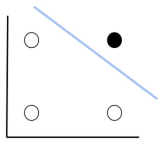


where $\sigma(x) = \frac{1}{1 + e^{-x}}$

Logistic Regression Re-Revisited

But, the **AND** function is also linearly separable:

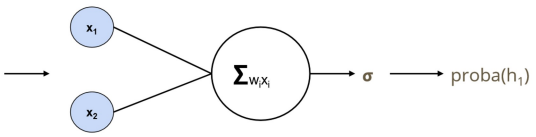
x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1



Logistic Regression Re-Revisited

Since we can learn h_1 and h_2 automatically through logistic regression

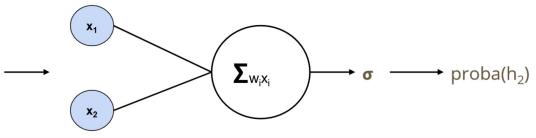
x_1	x_2	h_1
0	0	0
0	1	0
1	0	1
1	1	0



Logistic Regression Re-Revisited

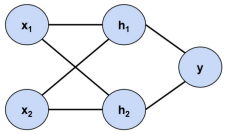
Since we can learn h_1 and h_2 automatically through logistic regression

x_1	x_2	h_2
0	0	0
0	1	1
1	0	0
1	1	0



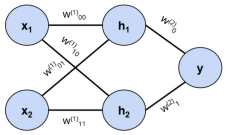
Neural Networks

Putting it all together:



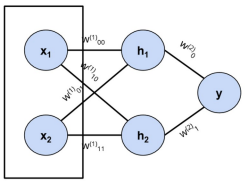
Neural Networks

Putting it all together:



Neural Networks

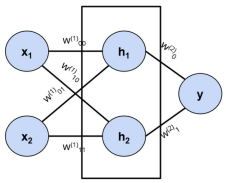
Putting it all together:



Input layer

Neural Networks

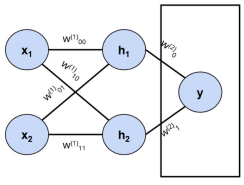
Putting it all together:



Hidden layer

Neural Networks

Putting it all together:



Output layer

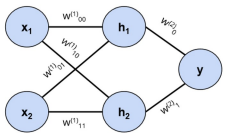
Neural Networks

We need to define:

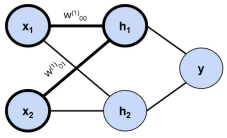
1. How input flows through the network to get the output (forward propagation)
2. How the weights and biases gets updated (Backpropagation)

- How do we feed information to the model?
- How does information flow through the model?
- How do you update the weights?

Neural Networks - Forward Propagation



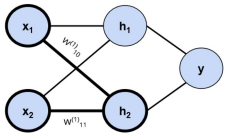
Neural Networks - Forward Propagation



$h_1 = \sigma(w^{(1)}_{00} x_1 + w^{(1)}_{01} x_2 + b^{(1)}_1)$

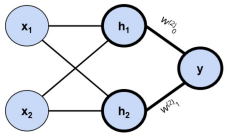
- Weighted sum
- Activate using the sigmoid function
- 6 weights, 3 biases, a lot to keep track of

Neural Networks - Forward Propagation



$h_2 = \sigma(w^{(1)}_{10} x_1 + w^{(1)}_{11} x_2 + b^{(1)}_2)$

Neural Networks - Forward Propagation



$y = \sigma(w^{(2)}_0 h_1 + w^{(2)}_1 h_2 + b^{(2)}_1)$

Neural Networks - Forward Propagation

Using matrix notation:

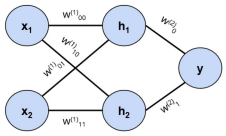
$$\begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \sigma\left(\begin{bmatrix} w^{(1)}_{00} & w^{(1)}_{01} \\ w^{(1)}_{10} & w^{(1)}_{11} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b^{(1)}_1 \\ b^{(1)}_2 \end{bmatrix} \right)$$

$$y = \sigma(\begin{bmatrix} w^{(2)}_{00} \\ w^{(2)}_{01} \end{bmatrix}^T \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} + b^{(2)})$$

- It gets very cumbersome to track all the weights, so just having a weight matrix for that layer is easier

Neural Networks - Forward Propagation

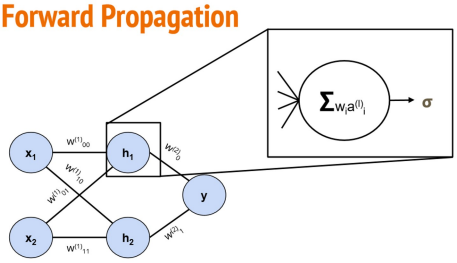
Q: if all the weights and biases are initialized to 0, what will be the output of the network?



- What happens all weights are 0?
  - Then we get 1/2 as the prediction for the model

Neural Networks - Forward Propagation

Q: why have **σ** at all?



- Why do we have sigma at all?
  - Then we are basically just doing logistic regression on X1 and X2 again
    - With different weights
  - It is usually important to introduce some sort of function after this weighted sum, in order to make sure you aren't just doing logistic regression again
  - Sigma: activation function

Neural Networks - Forward Propagation

If we don't, we just end up with normal logistic regression on x_1 and x_2.

$h_1 = w^{(1)}_{00} x_1 + w^{(1)}_{01} x_2 + b^{(1)}_1$

$h_2 = w^{(1)}_{10} x_1 + w^{(1)}_{11} x_2 + b^{(1)}_2$

Then

$y = \sigma(w^{(2)}_0 h_1 + w^{(2)}_1 h_2 + b^{(2)}_1)$

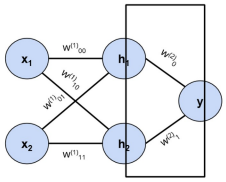
$= \sigma(w^{(2)}_0(w^{(1)}_{00} x_1 + w^{(1)}_{01} x_2 + b^{(1)}_1) + w^{(2)}_1(w^{(1)}_{10} x_1 + w^{(1)}_{11} x_2 + b^{(1)}_2) + b^{(2)}_1)$

$= \sigma(w_1 x_1 + w_2 x_2 + b_2)$

-

Neural Networks - BackPropagation

How do weights and biases get updated?



$W^{(2)}$ & $b^{(2)}$

-

Neural Networks - BackPropagation

Using the chain rule:

$$\frac{\partial C}{\partial W^{(2)}} = \frac{\partial C}{\partial u^{(2)}} \frac{\partial u^{(2)}}{\partial W^{(2)}} \quad \text{where} \quad u^{(2)} = W^{(2)} h + b^{(2)}$$

$$= \frac{\partial C}{\partial u^{(2)}} \cdot h = \frac{1}{n} \sum_{i=1}^{n} h(y_i - \sigma(u^{(2)}))$$

$h = \sigma(W^{(1)} X + b^{(1)})$

- Cost function C
  - Look at gradient descent slides
- The math isn't as important as why this algorithm works
- Weighted sum of the h's
- Partial derivatives of cost with respect to u, and the partial derivate of u with respect to W

Neural Networks - BackPropagation

Similarly:

$$\frac{\partial C}{\partial b^{(2)}} = \frac{\partial C}{\partial u^{(2)}} \frac{\partial u^{(2)}}{\partial b^{(2)}} = \frac{1}{n} \sum_{i=1}^{n} y_i - \sigma(u^{(2)})$$

Neural Networks - BackPropagation

So we can update $W^{(2)}$ and $b^{(2)}$ as follows:

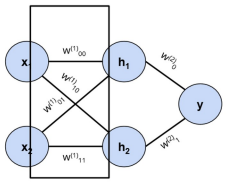
$$\begin{bmatrix} W^{(2)}_{new} \\ \\ b^{(2)}_{new} \end{bmatrix} = -\alpha \begin{bmatrix} \frac{\partial C}{\partial W^{(2)}} \\ \\ \frac{\partial C}{\partial b^{(2)}} \end{bmatrix} + \begin{bmatrix} W^{(2)} \\ \\ b^{(2)} \end{bmatrix}$$

But how do we update $W^{(1)}$ and $b^{(1)}$

- Similarly we can do the same for b
- So far this is just gradient descent, logisitc regression model…
  - with different variables

Neural Networks - BackPropagation

How do weights and biases get updated?



$W^{(1)}$ & $b^{(1)}$

Neural Networks - BackPropagation

Using the chain rule:

$$\frac{\partial C}{\partial W^{(1)}} = \frac{\partial C}{\partial h} \cdot \frac{\partial h}{\partial W^{(1)}} = \frac{\partial C}{\partial h} \cdot \frac{\partial h}{\partial u^{(1)}} \cdot \frac{\partial u^{(1)}}{\partial W^{(1)}} \quad \text{where} \quad u^{(1)} = w^{(1)}x + b^{(1)}$$

$$= \frac{\partial C}{\partial u^{(2)}} \cdot \frac{\partial u^{(2)}}{\partial h} \cdot \frac{\partial h}{\partial u^{(1)}} \cdot \frac{\partial u^{(1)}}{\partial W^{(1)}} = \frac{\partial C}{\partial u^{(2)}} \cdot W^{(2)} \cdot \sigma'(u^{(1)}) \cdot x$$

Already computed

- Need to apply the chain rule a couple of times
- Can update these using gradient descent

Neural Networks - BackPropagation

Similarly:

$$\frac{\partial C}{\partial b^{(1)}} = \frac{\partial C}{\partial u^{(2)}} \cdot W^{(2)} \cdot \sigma'(u^{(1)})$$

Already computed

-

Neural Networks - BackPropagation

Backpropagation: update $W^{(1)}$ and $b^{(1)}$ without recomputing values that are computed when getting the gradients of the previously updated layer.

http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf

- We could technically do gradient descent the way we know how to do it
- But there is a better way to do this so we aren't recomputing the terms
- Backpropagation
  - Efficiently tracking values so you aren't recalculating them
  - Backpropagation avoids recomputing values (unlike gradient descent)
- Recommended paper on efficient back propagation

Neural Networks - BackPropagation

Important Note:

$$\frac{\partial C}{\partial W^{(1)}} = \frac{\partial C}{\partial h} \cdot \frac{\partial h}{\partial W^{(1)}} = \frac{\partial C}{\partial h} \cdot \frac{\partial h}{\partial u^{(1)}} \cdot \frac{\partial u^{(1)}}{\partial W^{(1)}} \quad \text{where} \quad u^{(1)} = w^{(1)}x + b^{(1)}$$

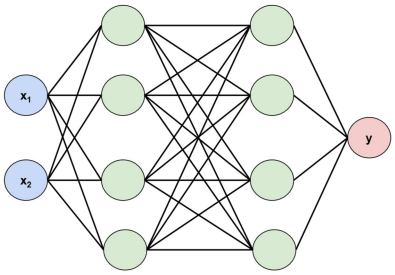
$$= \frac{\partial C}{\partial u^{(2)}} \cdot \frac{\partial u^{(2)}}{\partial h} \cdot \frac{\partial h}{\partial u^{(1)}} \cdot \frac{\partial u^{(1)}}{\partial W^{(1)}} = \frac{\partial C}{\partial u^{(2)}} \cdot W^{(2)} \cdot \sigma'(u^{(1)}) \cdot x$$

Depends on both data and weights
Initializing all weights to zero then is not a good idea

- Depends on both the weights AND the data
  - The weights you learn effects the ability to learn

Feedforward Neural Networks

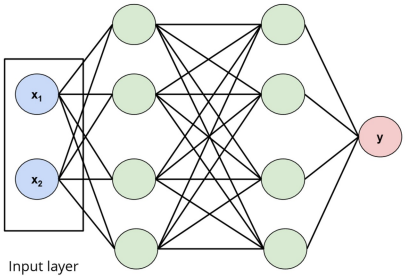
In general:



- Can have any number of neurons within a layer
- Can have any number of hidden layers
- Can do multiclass outcome

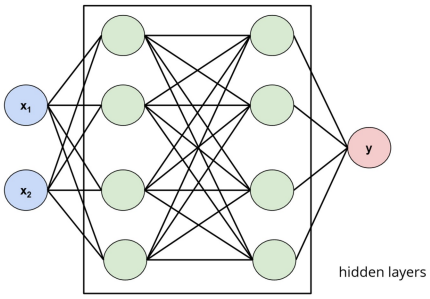
Feedforward Neural Networks

In general:



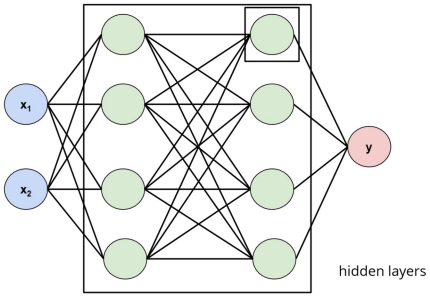
Feedforward Neural Networks

In general:



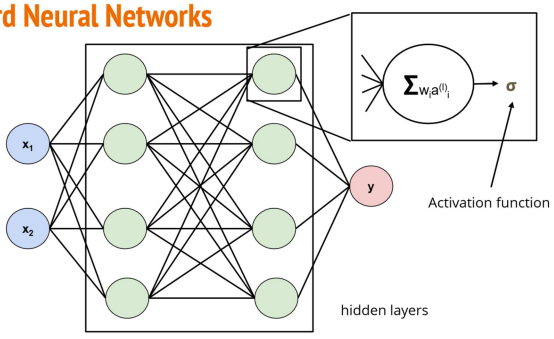
Feedforward Neural Networks

In general:



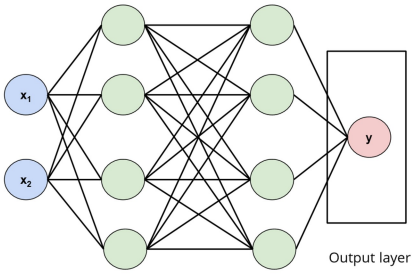
Feedforward Neural Networks

In general:



Feedforward Neural Networks

In general:



Feedforward Neural Networks

The hope:

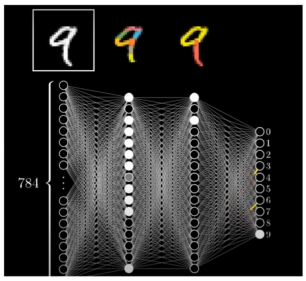


Image from 3b1b

- Hope you learn more abstract features as you go through the model

Feedforward Neural Networks

The reality:

(a) Husky classified as wolf

(b) Explanation

Figure 11: Raw data and explanation of a bad model's prediction in the "Husky vs Wolf" task.

	Before	After
Trusted the bad model	10 out of 27	3 out of 27
Snow as a potential feature	12 out of 27	25 out of 27

Table 2: "Husky vs Wolf" experiment results.

Image from "Why Should I Trust You?": Explaining the Predictions of Any Classifier (2016)Marco Tulio Ribeiro, Sameer Singh, Carlos Guestrin

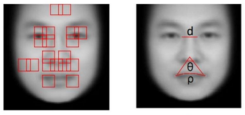
- Every wolf had a background of snow, and it learned that
- Has ability to learn, but may learn the wrong thing

Feedforward Neural Networks

The scary reality:

(a) Three samples in criminal ID photo set S_c.

(b) Three samples in non-criminal ID photo set S_n



(a) (b)

Figure 8. (a) FGM results; (b) Three discriminative features ρ, d and θ.

from "Automated Inference on Criminality using Face Images", Xiaolin Wu, Xi Zhang

According to this model, if you don't smile, you're a criminal

- Learned a smile detector
- Learning features automatically means a non-technical person can try to apply it
  - But you want someone to have spent some time thinking about a model that's effecting your life

Neural Networks

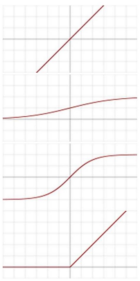
Can do both **Classification** and **Regression**

Neural Networks - Tuning Parameters

1. Step size α
2. Number of BackPropagation iterations
3. Batch Size
4. Number of hidden layers
5. Size of each hidden layer
6. Activation function used in each layer
7. Cost function
8. Regularization (to avoid overfitting)

- Step size like in gradient descent
- Batch size
  - Stochastic gradient descent
  - ….
- Activation function
  - You could use a variety of things
- Regularization to avoid overfitting

Activation Functions



Identity -> x

Sigmoid -> σ(x)

Tanh -> tanh(x)

ReLU -> max(0, x)

Note: can use any function you want in order to introduce non-linearity. These are just the popular ones that have been shown to work in practice.

Tuning the activation function is equivalent to feature engineering.

- Some people have done it so much they know what works but there isn't rules
  - Black boxes we don't really understand
    - Why a model is predicting something wrong, especially in medicine, matters
      - Start has been made in trying to crack these open and understand
- The more you tune these activation functions, the more you are actually doing feature engineering
- It was complicated with 2 inputs, 2 hidden, and an output
  - But you could have a large number of neurons and even more weights to keep track of

Demo

- Check out the juyptr notebook in the Neural Network folder in git
- Can have any number of hidden layers
- Epoch is a term used in a lot of deep learning libraries
- May need to pip3 install pypt5?
- Would like to learn a circle decision boundary
- Learned new features, then linearly separate those new features
  - If you project this back into the original space, it's no longer

a line anymore
- 10H vs sigmoid
- May learn more with more epochs
- Just because decision boundary is cosine, doesn't mean decision boundary activation should be cosine
  - How can you use cosine with only one neuron?
    - You have to choose an activation function for the entire layer
      - That's kind of a limitation
- What activations functions you should or shouldn't use there is a lot of rigidity in the field
  - But you can do whatever you want and it may improve things
- Singular value decomposition on an image
  - You can do the same thing with a neural network
  - Create a neural network on principal component extraction?
  - Will use a cost function to compare the output to the input, and adjust weights
  - Pretty similar to SVD
  - Boat images again
  - Can compare the Frobenius Distance between the rank 10 approximation in SVD to the neural network one
    - Was pretty close
  - You can generate data like this, neural networks are good for more than classification
- Code all posted on the repo, try this out on your own
- Try a different number of layers, different number of neurons, etc.
- Its bad both if we learn something wrong or it takes too long
  - So opitimize
- There is some guidelines, but it's hard to know how to do this wrong

==========

- Normalizing your data is important
- What proportion of the data is responsible for creating that feature?
  - Here you are learning features on parts of the dataset, unlike traditional analysis methods
  - As data moves away from 0, you really quickly move to all or none of the data activate a neuron
    - This is really bad
    - This just turns this into a linear combination of the data
    - This isn't a the case with just RLU for activation function
    - If you don't have normalized data, neurons can be frozen
    - If you don't normalize your data, your gradient descent algorithm often will not work

- https://medium.com/mlearning-ai/tuning-neural-networks-part-i-normalize-your-data-6821a28b2cd8
- If

Neural Networks

First: Normalize your data

https://medium.com/mlearning-ai/tuning-neural-networks-part-i-normalize-your-data-6821a28b2cd8

Neural Networks - Initialization Gotchas

https://medium.com/mlearning-ai/tuning-neural-networks-part-ii-considerations-for-initialization-4f82e525da69

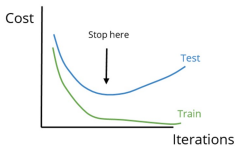
Neural Networks - Challenges

1. High risk of overfitting as you're optimizing on the training set.
2. As the dimensionality of the input increases:
    a. So does the number of weights
    b. The gradients typically get smaller: Vanishing gradient problem
3. Doesn't do well for computer vision where the object of detection can be anywhere in the image
4. Doesn't handle sequences of inputs (i.e. providing context for data)

Neural Networks - Regularization

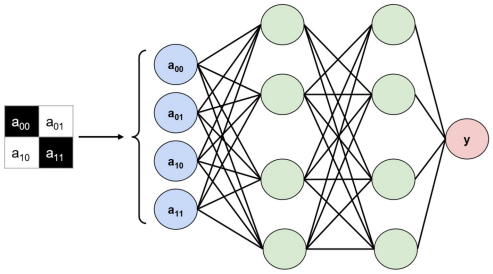
Two main ways:

1. Early termination of weight / bias updates

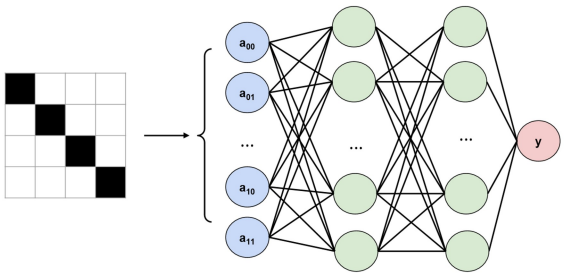


1. Dropout - kill neurons (by setting them to 0) randomly

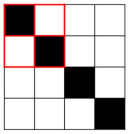
Neural Networks - Convolutional Neural Networks



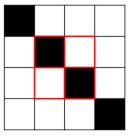
Neural Networks - Convolutional Neural Networks



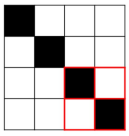
Neural Networks - Convolutional Neural Networks



Neural Networks - Convolutional Neural Networks



Neural Networks - Convolutional Neural Networks

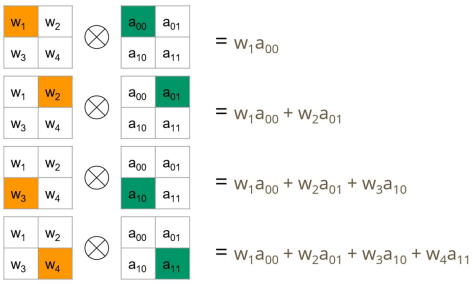


Neural Networks - Convolutional Neural Networks

Recall: Our network learns weights for each cell

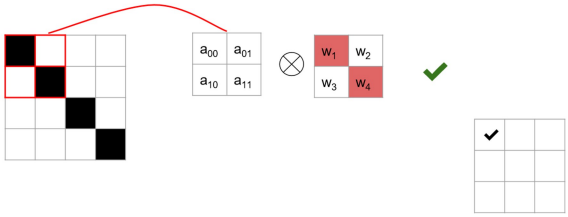
w_1	w_2
w_3	w_4

Neural Networks - Convolutional Neural Networks



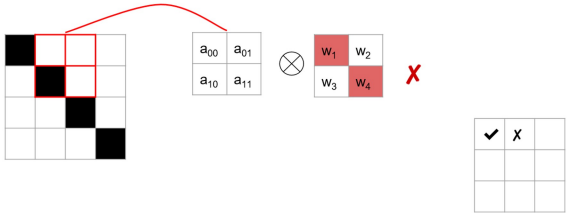
Neural Networks - Convolutional Neural Networks

Knowing this, what happens if we slide this filter across the larger diagonal?



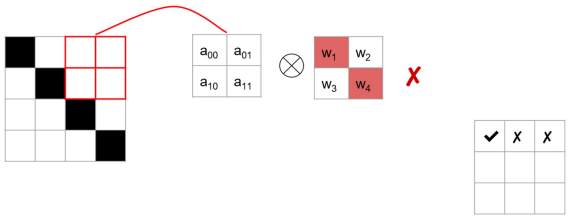
Neural Networks - Convolutional Neural Networks

Knowing this, what happens if we slide this filter across the larger diagonal?



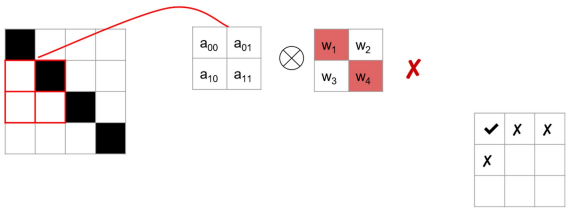
Neural Networks - Convolutional Neural Networks

Knowing this, what happens if we slide this filter across the larger diagonal?



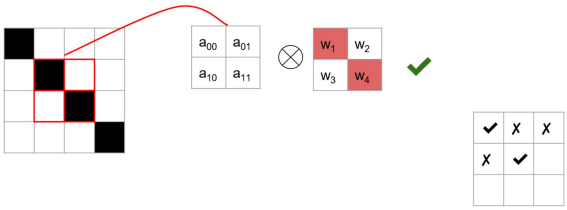
Neural Networks - Convolutional Neural Networks

Knowing this, what happens if we slide this filter across the larger diagonal?



Neural Networks - Convolutional Neural Networks

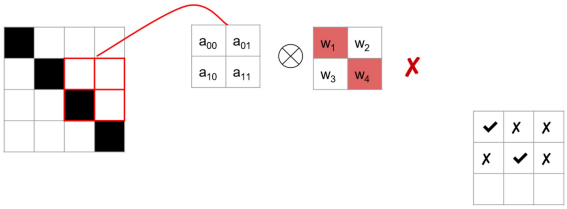
Knowing this, what happens if we slide this filter across the larger diagonal?



✓ ✗ ✗
✗ ✓

Neural Networks - Convolutional Neural Networks

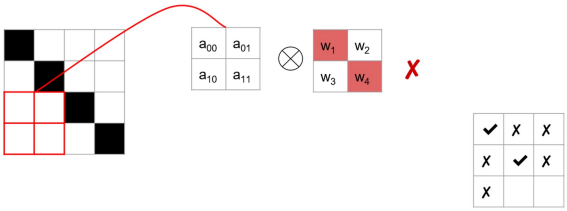
Knowing this, what happens if we slide this filter across the larger diagonal?



✓ ✗ ✗
✗ ✓ ✗

Neural Networks - Convolutional Neural Networks

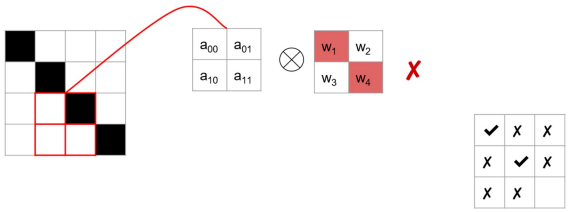
Knowing this, what happens if we slide this filter across the larger diagonal?



✓ ✗ ✗
✗ ✓ ✗
✗

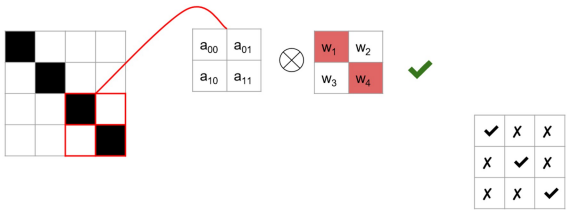
Neural Networks - Convolutional Neural Networks

Knowing this, what happens if we slide this filter across the larger diagonal?



Neural Networks - Convolutional Neural Networks

Knowing this, what happens if we slide this filter across the larger diagonal?



Neural Networks - Convolutional Neural Networks

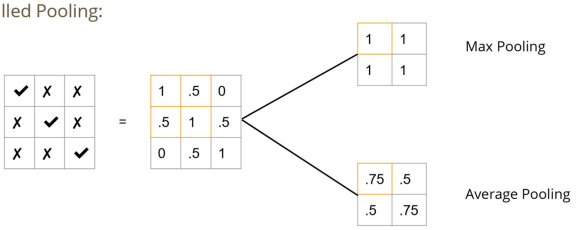
Creating such a filter allows us to:

1. Reduce the number of weights
2. Capture features all over the image

The process of applying a filter (or kernel) is called a convolution

Neural Networks - Convolutional Neural Networks

To reduce the weights even further, another phase is done after convolution called Pooling:



Neural Networks - Convolutional Neural Networks

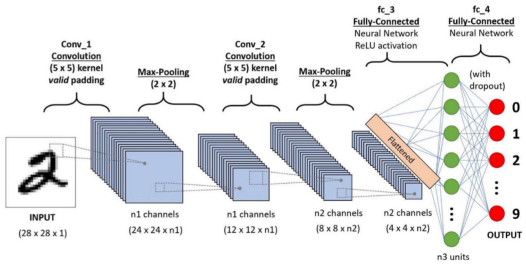


Image from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Neural Networks - Convolutional Neural Networks

Main application: Computer vision

File - D:\Users\Wolfs\Documents\GitHub\CS506-Fall2021\11-neural-networks\visualizing_neural_networks.ipynb

```
#%%
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import Image
import sklearn.datasets as datasets

from kviz.dense import DenseGraph
```

```
import matplotlib.pyplot as plt
from IPython.display import Image
import sklearn.datasets as datasets

from kviz.dense import DenseGraph
import tensorflow as tf
from tensorflow import keras, math, random, stack
from tensorflow.keras import layers, initializers
from tensorflow.keras.activations import relu

tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)

ACTIVATION = 'sigmoid'
model = keras.models.Sequential()
model.add(layers.Dense(2, input_dim=2, activation=ACTIVATION))
model.add(layers.Dense(1, activation=ACTIVATION))
model.compile(loss="binary_crossentropy")

dg = DenseGraph(model)
dg.render('graph.png')
Image(filename='graph.png')
#%%
centers = [[0, 0]]
t, _ = datasets.make_blobs(n_samples=200, centers=centers, cluster_std=1,
                                random_state=1)

# CIRCLE
def generate_circle_data(t, centers):
    # create some space between the classes
    X = np.array(list(filter(
        lambda x : (x[0] - centers[0][0])**2 + (x[1] - centers[0][1])**2 < 1
        or (x[0] - centers[0][0])**2 + (x[1] - centers[0][1])**2 > 1.5, t
    )))
    Y = np.array([1 if (x[0] - centers[0][0])**2 + (x[
```



```
1] - centers[0][1])**2 >= 1 else 0 for x in X])
    return X, Y

# LINE
def generate_line_data(t):
    # create some space between the classes
    X = np.array(list(filter(lambda x : x[0] - x[1] < -.5 or x[0] - x[1] > .5, t)))
    Y = np.array([1 if x[0] - x[1] >= 0 else 0 for x in X])
    return X, Y

# CURVE
def generate_curve_data(t):
    # create some space between the classes
    X = np.array(list(filter(lambda x : m.cos(4*x[0]) - x[1] < -.5 or m.cos(4*x[0]) - x[1] > .5, t)))
    Y = np.array([1 if m.cos(4*x[0]) - x[1] >= 0 else 0 for x in X])
    return X, Y

# XOR
def generate_xor_data():
    X = np.array([
        [0,0],
        [0,1],
        [1,0],
        [1,1]])
    Y = np.array([x[0]^x[1] for x in X])
    return X, Y

X, Y = generate_xor_data()
colors = np.array([x for x in 'bgrcmykbgrcmykbgrcmykbgrcmyk'])
colors = np.hstack([colors] * 20)

```

```
colors = np.array([x for x in 'bgrcmykbgrcmykbgrcmykbgrcmyk'])
colors = np.hstack([colors] * 20)

plt.scatter(X[:,0],X[:,1],color=colors[Y].tolist(), s=100, alpha=.9)
plt.show()
#%%
dg.animate_learning(X, Y, filename='animate', verbose=
```



```
0, epochs=1000)
Image(filename='animate.gif')
#%%
dg.animate_activations(X, filename='act')
Image(filename='act_stacked.gif')
#%%
X, Y = generate_circle_data(t, centers)

plt.scatter(X[:,0],X[:,1],color=colors[Y].tolist(), s=100, alpha=.9)
plt.show()
#%%
dg = DenseGraph(model)
dg.animate_learning(X, Y, filename='animate', duration=300, verbose=0, epochs=1000, batch_size=50)
Image(filename='animate.gif')
#%%
dg.animate_activations(X, filename='act', duration=50)
Image(filename='act_stacked.gif')
#%%
%matplotlib widget
from mpl_toolkits.mplot3d import Axes3D

model = keras.models.Sequential()
model.add(layers.Dense(3, input_dim=2, activation='sigmoid'))
model.add(layers.Dense(1, activation=ACTIVATION))
model.compile(loss="binary_crossentropy")

history = model.fit(X, Y, batch_size=50, epochs=1000, verbose=0)

# Show the transformation of the input at the first hidden layer
layer = model.layers[0]
print(layer.get_config(), layer.get_weights())
keras_function = keras.backend.function([model.input], [layer.output])
layerVals = np.array(keras_function(X))[0]
fig = plt.figure()
```



```
ax = Axes3D(fig)
ax.scatter(layerVals[:,0], layerVals[:, 1], layerVals[:, 2], color=colors[Y].tolist(), s=100, alpha=.9)
plt.show()
```

```
ax = Axes3D(fig)
ax.scatter(layerVals[:,0], layerVals[:, 1], layerVals
    [:, 2], color=colors[Y].tolist(), s=100, alpha=.9)
plt.show()
```

```python
import math as m
import numpy as np
import matplotlib.pyplot as plt
import sklearn.datasets as datasets
from tensorflow import keras, math, random, stack
from tensorflow.keras import layers, initializers
from tensorflow.keras.activations import relu


#
#       x[0] --- h1
#            \ /    \
#             X       output
#            / \    /
#       x[1] --- h2
#
# This is the base model - nothing fancy here

# modify this line
ACTIVATION = "sigmoid"

# DONT MODIFY
# this is just for declaring
LABEL = None
THRESH = None
# Set random seed for reproducibility
np.random.seed(1)
random.set_seed(1)

if ACTIVATION=="tanh":
    LABEL = -1
    THRESH = 0
if ACTIVATION=="sigmoid":
    LABEL = 0
    THRESH = 0.5


# Can use a custom activation function
def custom_activation(x):
    x_0 = math.cos(x[...,0])

```

```
42      return relu(x, threshold=0)
43
44  # 10 H seeming learned the decision boundary better
    than 10H
45
46  # Basically started with two neurons, which wasn't
    enough to approximate decision boundary
47  # Move into higher dimension and separate using plane
48
49  model = keras.models.Sequential()
50  # Can try different activation functions
51  # reduce number of epochs (iterations) to get this to
    run faster
52  model.add(layers.Dense(2, input_dim=2, activation=
    custom_activation))
53  model.add(layers.Dense(1, activation=ACTIVATION))
54  #opt = keras.optimizers.Adam(learning_rate=.01)
55  model.compile(loss="binary_crossentropy") #, optimizer
    =opt)
56
57  centers = [[0, 0]]
58  t, _ = datasets.make_blobs(n_samples=200, centers=
    centers, cluster_std=1,
59                                  random_state=1)
60
61  # CIRCLE
62  def generate_circle_data(t):
63      # create some space between the classes
64      X = np.array(list(filter(lambda x : (x[0] -
    centers[0][0])**2 + (x[1] - centers[0][1])**2 < 1 or (
    x[0] - centers[0][0])**2 + (x[1] - centers[0][1])**2
     > 1.5, t)))
65      Y = np.array([1 if (x[0] - centers[0][0])**2 + (x[
    1] - centers[0][1])**2 >= 1 else LABEL for x in X])
66      return X, Y
67
68  # LINE
69  def generate_line_data(t):
70      # create some space between the classes
71      X = np.array(list(filter(lambda x : x[0] - x[1
    ] < -.5 or x[0] - x[1] > .5, t)))
```



```
72      Y = np.array([1 if x[0] - x[1] >= 0 else LABEL
    for x in X])
73      return X, Y
74
75  # CURVE
76  def generate_curve_data(t):
77      # create some space between the classes
78      X = np.array(list(filter(lambda x : m.cos(4*x[0
    ]) - x[1] < -.5 or m.cos(4*x[0]) - x[1] > .5, t)))
79      Y = np.array([1 if m.cos(4*x[0]) - x[1] >= 0 else
     LABEL for x in X])
80      return X, Y
81
82  # XOR
83  def generate_xor_data():
84      X = np.array([
85          [0,0],
86          [0,1],
87          [1,0],
88          [1,1]])
89      Y = np.array([x[0]^x[1] for x in X])
90      return X, Y
91
92  X, Y = generate_circle_data(t)
93  # X, Y = generate_line_data(t)
94  # X, Y = generate_curve_data(t)
95  # X, Y = generate_xor_data()
96
97  colors = np.array([x for x in '
    bgrcmykbgrcmykbgrcmykbgrcmyk'])
98  colors = np.hstack([colors] * 20)
```

```
 96
 97 colors = np.array([x for x in '
    bgrcmykbgrcmykbgrcmykbgrcmyk'])
 98 colors = np.hstack([colors] * 20)
 99
100 plt.scatter(X[:,0],X[:,1],color=colors[Y].tolist(), s
    =100, alpha=.9)
101 plt.show()
102
103 history = model.fit(X, Y, batch_size=50, epochs=1000)
104
105 # Show the transformation of the input at the first
    hidden layer
106 layer = model.layers[0]
```


File - D:\Users\Wolfs\Documents\GitHub\CS506-Fall2021\11-neural-networks\base_model.py

```
107 print(layer.get_config(), layer.get_weights())
108 keras_function = keras.backend.function([model.input
    ], [layer.output])
109 layerVals = np.array(keras_function(X))[0]
110 plt.scatter(layerVals[:,0], layerVals[:, 1], color=
    colors[Y].tolist(), s=100, alpha=.9)
111 plt.show()
112
113 # create a mesh to plot in
114 h = .02  # step size in the mesh
115 x_min, x_max = layerVals[:, 0].min() - .5, layerVals
    [:, 0].max() + 1
116 y_min, y_max = layerVals[:, 1].min() - .5, layerVals
    [:, 1].max() + 1
117 xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
118                      np.arange(y_min, y_max, h))
119 meshData = np.c_[xx.ravel(), yy.ravel()]
120
121 # Plot the decision boundary. For that, we will
    assign a color to each
122 # point in the mesh
123 fig, ax = plt.subplots()
124 layer = model.layers[-1]
125
126 intermediateModel = keras.models.Sequential()
127 intermediateModel.add(layers.Dense(1, input_dim=2,
    activation=ACTIVATION))
128 intermediateModel.compile(loss="binary_crossentropy")
129 intermediateModel.layers[0].set_weights(layer.
    get_weights())
130
131 Z = intermediateModel.predict(meshData)
132 Z = np.array([LABEL if x < THRESH else 1 for x in Z])
133 Z = Z.reshape(xx.shape)
134 ax.contourf(xx, yy, Z, alpha=.3, cmap=plt.cm.Paired)
135
136 T = intermediateModel.predict(layerVals)
137 T = np.array([LABEL if x < THRESH else 1 for x in T])
138 T = T.reshape(layerVals[:, 0].shape)
139 ax.scatter(layerVals[:, 0], layerVals[:, 1], color=
    colors[T].tolist(), s=100, alpha=.9)
```

```
140 ax.set_xlabel("h0")
141 ax.set_ylabel("h1")
142 plt.show()
143
144 # create a mesh to plot in
145 h = .02  # step size in the mesh
146 x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + 1
147 y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + 1
148 xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
149                      np.arange(y_min, y_max, h))
150 meshData = np.c_[xx.ravel(), yy.ravel()]
151
152 fig, ax = plt.subplots()
153 Z = model.predict(meshData)
154 Z = np.array([LABEL if x < THRESH else 1 for x in Z])
155 Z = Z.reshape(xx.shape)
156 ax.contourf(xx, yy, Z, alpha=.3, cmap=plt.cm.Paired)
157 ax.axis('off')
158
159 # Plot also the training points
160 T = model.predict(X)
161 T = np.array([LABEL if x < THRESH else 1 for x in T])
162 T = T.reshape(X[:,0].shape)
163 ax.scatter(X[:, 0], X[:, 1], color=colors[T].tolist
    (), s=100, alpha=.9)
164 plt.show()
165
```

```python
import numpy as np
import matplotlib.pyplot as plt
import sklearn.datasets as datasets
from tensorflow import keras
from tensorflow.keras import layers

# The model we want to learn is
#
#   if x[0]**2 + x[1]**2 >= 1 then 1 else 0
#
#       x[0] --- h11 --- h11
#            \ /     \ /   \
#             X       X      output
#            / \     / \   /
#       x[1] --- h21 --- h21
#
# Try adding multiple layers or the same size - which number works best?

# modify this line
ACTIVATION = "sigmoid"

# DONT MODIFY
# this is just for declaring
LABEL = None
THRESH = None

if ACTIVATION=="tanh":
    LABEL = -1
    THRESH = 0
if ACTIVATION=="sigmoid":
    LABEL = 0
    THRESH = 0.5

model = keras.models.Sequential()
model.add(layers.Dense(2, input_dim=2))
model.add(layers.Dense(2))
model.add(layers.Dense(1, activation=ACTIVATION))
model.compile(loss="binary_crossentropy")

centers = [[0, 0]]
```

```
41 t, _ = datasets.make_blobs(n_samples=750, centers=
   centers, cluster_std=1,
42                                 random_state=0)
43
44 # create some space between the classes
45 X = np.array(list(filter(lambda x : x[0]**2 + x[1]**2
    < 1 or x[0]**2 + x[1]**2 > 1.5, t)))
46 Y = np.array([1 if x[0]**2 + x[1]**2 >= 1 else LABEL
   for x in X])
47
48 colors = np.array([x for x in '
   bgrcmykbgrcmykbgrcmykbgrcmyk'])
49 colors = np.hstack([colors] * 20)
50
51 plt.scatter(X[:,0],X[:,1],color=colors[Y].tolist(), s=
   10, alpha=0.8)
52 plt.show()
53
54 history = model.fit(X, Y, batch_size=50, epochs=100)
55
56 # Show the transformation of the input at the last
   hidden layer
57 layer = model.layers[9]
58 print(layer.get_config(), layer.get_weights())
59 keras_function = keras.backend.function([model.input
   ], [layer.output])
60 layerVals = np.array(keras_function(X))[0]
61 plt.scatter(layerVals[:,0], layerVals[:, 1], color=
   colors[Y].tolist(), s=10, alpha=.8)
62 plt.show()
63
64 # create a mesh to plot in
65 h = .02  # step size in the mesh
66 x_min, x_max = layerVals[:, 0].min() - .5, layerVals
   [:, 0].max() + 1
67 y_min, y_max = layerVals[:, 1].min() - .5, layerVals
   [:, 1].max() + 1
68 xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
69                      np.arange(y_min, y_max, h))
70 meshData = np.c_[xx.ravel(), yy.ravel()]
71
```



```
 72 # Plot the decision boundary. For that, we will
    assign a color to each
 73 # point in the mesh
 74 fig, ax = plt.subplots()
 75 layer = model.layers[10]
 76
 77 intermediateModel = keras.models.Sequential()
 78 intermediateModel.add(layers.Dense(1, input_dim=2,
    activation=ACTIVATION))
 79 intermediateModel.compile(loss="binary_crossentropy")
 80 intermediateModel.layers[0].set_weights(layer.
    get_weights())
 81
 82 Z = intermediateModel.predict(meshData)
 83 Z = np.array([LABEL if x < THRESH else 1 for x in Z])
 84 Z = Z.reshape(xx.shape)
 85 ax.contourf(xx, yy, Z, alpha=.3, cmap=plt.cm.Paired)
 86 ax.axis('off')
 87
 88 T = intermediateModel.predict(layerVals)
 89 T = np.array([LABEL if x < THRESH else 1 for x in T])
 90 T = T.reshape(layerVals[:, 0].shape)
 91 ax.scatter(layerVals[:, 0], layerVals[:, 1], color=
    colors[T].tolist(), s=1, alpha=0.9)
 92 plt.show()
 93
 94 # create a mesh to plot in
 95 h = .02  # step size in the mesh
 96 x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + 1
 97 y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + 1
 98 xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
```

```
 95 h = .02  # step size in the mesh
 96 x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + 1
 97 y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + 1
 98 xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
 99                       np.arange(y_min, y_max, h))
100 meshData = np.c_[xx.ravel(), yy.ravel()]
101
102 fig, ax = plt.subplots()
103 Z = model.predict(meshData)
104 Z = np.array([LABEL if x < THRESH else 1 for x in Z])
105 Z = Z.reshape(xx.shape)
106 ax.contourf(xx, yy, Z, alpha=.3, cmap=plt.cm.Paired)
107 ax.axis('off')
108
```


File - D:\Users\Wolfs\Documents\GitHub\CS506-Fall2021\11-neural-networks\multi_layer.py

```
109 # Plot also the training points
110 T = model.predict(X)
111 T = np.array([LABEL if x < THRESH else 1 for x in T])
112 T = T.reshape(X[:,0].shape)
113 ax.scatter(X[:, 0], X[:, 1], color=colors[T].tolist
    (), s=1, alpha=0.9)
114 plt.show()
115
```

File - D:\Users\Wolfs\Documents\GitHub\CS506-Fall2021\11-neural-networks\multi_neuron.py

```python
import numpy as np
import matplotlib.pyplot as plt
import sklearn.datasets as datasets
from tensorflow import keras
from tensorflow.keras import layers

# The model we want to learn is
#
#   if x[0]**2 + x[1]**2 >= 1 then 1 else 0
#
#       x[0] --- h1
#         |  \ /    \
#         \   X      \
#          \ / \      \
#           X  h2 --- output
#          / \ /      /
#         /   X      /
#         |  / \    /
#       x[1]  --- h3
#
# Try adding more neurons for the given hidden layer

# modify this line
ACTIVATION = "sigmoid"

# DONT MODIFY
# this is just for declaring
LABEL = None
THRESH = None

if ACTIVATION=="tanh":
    LABEL = -1
    THRESH = 0
if ACTIVATION=="sigmoid":
    LABEL = 0
    THRESH = 0.5

model = keras.models.Sequential()
model.add(layers.Dense(3, input_dim=2, activation='
relu'))
model.add(layers.Dense(1, activation=ACTIVATION))
```

```
41 model.compile(loss="binary_crossentropy")
42
43 centers = [[0, 0]]
44 t, _ = datasets.make_blobs(n_samples=200, centers=
   centers, cluster_std=1,
45                                 random_state=1)
46
47 # create some space between the classes
48 X = np.array(list(filter(lambda x : x[0]**2 + x[1]**2
    < 1 or x[0]**2 + x[1]**2 > 1.5, t)))
49 Y = np.array([1 if x[0]**2 + x[1]**2 >= 1 else LABEL
   for x in X])
50
51 colors = np.array([x for x in '
   bgrcmykbgrcmykbgrcmykbgrcmyk'])
52 colors = np.hstack([colors] * 20)
53
54 plt.scatter(X[:,0],X[:,1],color=colors[Y].tolist(), s=
   100, alpha=.9)
55 plt.show()
56
57 history = model.fit(X, Y, batch_size=50, epochs=250)
58
59 # Show the transformation of the input at the first
   hidden layer
60 layer = model.layers[0]
61 print(layer.get_config(), layer.get_weights())
62 keras_function = keras.backend.function([model.input
   ], [layer.output])
63 layerVals = np.array(keras_function(X))[0]
64 fig = plt.figure()
65 ax = fig.add_subplot(111, projection='3d')
66 ax.scatter(layerVals[:,0], layerVals[:, 1], layerVals
   [:, 2], color=colors[Y].tolist(), s=100, alpha=.9)
67 plt.show()
68
69 # create a mesh to plot in
70 h = .1  # step size in the mesh
71 x_min, x_max = layerVals[:, 0].min() - .5, layerVals
   [:, 0].max() + 1
72 y_min, y_max = layerVals[:, 1].min() - .5, layerVals
```



```
 72 [:, 1].max() + 1
 73 xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
 74                       np.arange(y_min, y_max, h))
 75 meshData = np.c_[xx.ravel(), yy.ravel(), np.zeros(len
    (xx.ravel()))]
 76
 77 # Plot the decision boundary. For that, we will
    assign a color to each
 78 # point in the mesh
 79 fig, ax = plt.subplots()
 80 layer = model.layers[-1]
 81
 82 intermediateModel = keras.models.Sequential()
 83 intermediateModel.add(layers.Dense(1, input_dim=3,
    activation=ACTIVATION))
 84 intermediateModel.compile(loss="binary_crossentropy")
 85 intermediateModel.layers[0].set_weights(layer.
    get_weights())
 86
 87 Z = intermediateModel.predict(meshData)
 88 Z = np.array([LABEL if x < THRESH else 1 for x in Z])
 89 Z = Z.reshape(xx.shape)
 90 ax.contourf(xx, yy, Z, alpha=.3, cmap=plt.cm.Paired)
    # plot in 2D
 91 ax.axis('off')
 92
 93 T = intermediateModel.predict(layerVals)
 94 T = np.array([LABEL if x < THRESH else 1 for x in T])
 95 T = T.reshape(layerVals[:, 0].shape)
 96 ax.scatter(layerVals[:, 0], layerVals[:, 1], color=
    colors[T].tolist(), s=100, alpha=.9) # plot in 2D
```

```
 94 T = np.array([LABEL if x < THRESH else 1 for x in T])
 95 T = T.reshape(layerVals[:, 0].shape)
 96 ax.scatter(layerVals[:, 0], layerVals[:, 1], color=
    colors[T].tolist(), s=100, alpha=.9) # plot in 2D
 97 plt.show()
 98
 99
100 # create a mesh to plot in
101 h = .1  # step size in the mesh
102 x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + 1
103 y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + 1
104 xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
105                      np.arange(y_min, y_max, h))
106 meshData = np.c_[xx.ravel(), yy.ravel()]
```


File - D:\Users\Wolfs\Documents\GitHub\CS506-Fall2021\11-neural-networks\multi_neuron.py

```
107
108 # Plot the decision boundary. For that, we will
    assign a color to each
109 # point in the mesh
110 fig, ax = plt.subplots()
111 Zp = model.predict(meshData)
112 Z = np.array([np.array([LABEL]) if x < THRESH else np
    .array([1]) for x in Zp])
113 Z = Z.reshape(xx.shape)
114 ax.contourf(xx, yy, Z, alpha=.3, cmap=plt.cm.Paired)
115 ax.axis('off')
116
117 # Plot also the training points
118 T = model.predict(X)
119 T = np.array([LABEL if x < THRESH else 1 for x in T])
120 T = T.reshape(X[:,0].shape)
121 ax.scatter(X[:, 0], X[:, 1], color=colors[T].tolist
    (), s=100, alpha=.9)
122 plt.show()
123
```

```python
import numpy as np
import matplotlib.cm as cm
import matplotlib.pyplot as plt
from tensorflow import keras, norm
from tensorflow.keras import layers

# Principal Component Extraction using Neural Net
#
#
#       x[0]            x[0]'
#          \            /
#           \          /
#            \        /
#       x[1]  \      /  x[1]'
#             \ \    / /
#         .    \\  //    .
#         .  --- z ---   .
#         .    //  \\    .
#             / /    \ \
#      x[n-1] /      \ x[n-1]'
#            /        \
#           /          \
#       x[n]            x[n]'
#
# where x' is the approximation of x based on the z components extracted

# MODIFY THIS LINE
RANK = 10

def custom_loss(y_true, y_pred):
    return norm(y_true - y_pred, ord='euclidean')

def frobenius(X, Y):
    return np.linalg.norm(X - Y, ord='fro')

boat = np.loadtxt('data/boat.dat')
plt.figure()
_ = plt.imshow(boat,cmap = cm.Greys_r)
plt.show()

```

```
41 u,s,vt=np.linalg.svd(boat,full_matrices=False)
42 _ = plt.plot(s)
43 plt.title('Singular values of boat image')
44 plt.show()
45
46 # construct a rank-RANK version of the boat
47 scopy = s.copy()
48 scopy[RANK:]=0
49 boatApprox = u.dot(np.diag(scopy)).dot(vt)
50
51 model = keras.models.Sequential()
52 model.add(layers.Dense(RANK, use_bias=False, input_dim
   =len(boat)))
53 model.add(layers.Dense(len(boat), use_bias=False))
54 model.compile(loss=custom_loss)
55
56 history = model.fit(boat, boat, batch_size=50, epochs=
   500)
57
58 boatNNApprox = model.predict(boat)
59
60 print("Frobenius Distance between boat and rank-"+str(
   RANK)+" approximation: ", frobenius(boat, boatApprox))
61 print("Frobenius Distance between boat and NN output
   with 1 hidden layer of "+str(RANK)+" neurons: ",
   frobenius(boat, boatNNApprox))
62
63 plt.figure(figsize=(12,9))
64 plt.subplot(1,3,1)
65 plt.imshow(boatApprox,cmap = cm.Greys_r)
66 plt.title('Rank ' + str(RANK) + ' SVD Boat')
67 plt.subplot(1,3,2)
68 plt.imshow(boatNNApprox,cmap = cm.Greys_r)
69 plt.title('Rank ' + str(RANK) + ' NN Boat')
70 plt.subplot(1,3,3)
71 plt.imshow(boat,cmap = cm.Greys_r)
72 plt.title('Original Boat')
73
74 _ = plt.subplots_adjust(wspace=0.5)
75 plt.show()
76
```