Chem 277B Spring 2024 Tutorial 3

Outline

- Scikit-learn:
  - Data preprocessing
  - Data splitting
- Artificial Neural Network:
  - Activation function

---

1. Scikit-learn

A package that provides implementation of various machine learning algorithms (including supervised learning and unsupervised learning), as well as tools for data preprocessing and analysis.

Documentations:

- Scikit-learn
- StandardScaler
- OneHotEncoder
- KFold

In [ ]:
```
import numpy as np, pandas as pd

df = pd.read_csv('titanic.csv')

# data cleaning
df = df[["Pclass", "Sex", "SibSp", "Parch", "Embarked", "Age", "Fare", "Surv
df
```

```
/var/folders/k8/mg372j_55z30k1z4y_8mb0w00000gn/T/ipykernel_72213/1640840549.
py:1: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major releas
e of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and bet
ter interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54
466

  import numpy as np, pandas as pd
```

Out[ ]:

	Pclass	Sex	SibSp	Parch	Embarked	Age	Fare	Survived
0	3	male	1	0	S	22.0	7.2500	0
1	1	female	1	0	C	38.0	71.2833	1
2	3	female	0	0	S	26.0	7.9250	1
3	1	female	1	0	S	35.0	53.1000	1
4	3	male	0	0	S	35.0	8.0500	0
...	...	...	...	...	...	...	...	...
885	3	female	0	5	Q	39.0	29.1250	0
886	2	male	0	0	S	27.0	13.0000	0
887	1	female	0	0	S	19.0	30.0000	1
889	1	male	0	0	C	26.0	30.0000	1
890	3	male	0	0	Q	32.0	7.7500	0

712 rows × 8 columns

In [ ]:
```
# Split into categorical and continuous features
categorical_features = df[['Pclass', 'Sex', 'Embarked', 'SibSp','Parch']] #
continuous_features = df[['Age','Fare']] #...
```

1.1 Data preprocessing

Normalize continuous features

$$X_{\text{scale}} = \frac{X - \text{avg}(X)}{\text{std}(X)}$$

In [ ]:
```
X_cont = continuous_features.values # n_sample * n_feature
print(X_cont.shape)

# calculate avg. and std.
```

```
print("Avg:", np.mean(X_cont, axis=0))
print("Std:", np.std(X_cont, axis=0))
```

```
(712, 2)
Avg: [29.6420927 34.5672514]
Std: [14.4827517 52.9014591]
```

TODO: Normalize manually

In [ ]:
```
X_cont_norm = (X_cont - X_cont.mean(0)) / X_cont.std(0)

print("After normalization:")
print("Avg:", X_cont_norm.mean(0))
print("Std:", X_cont_norm.std(0))
```

```
After normalization:
Avg: [ 2.94396218e-16 -6.73618464e-17]
Std: [1. 1.]
```

But we can also do it through the sklearn package using StandardScaler funtion

In [ ]:
```
from sklearn.preprocessing import StandardScaler

# scale (or normalize)
scaler = StandardScaler()
X_norm = scaler.fit_transform(X_cont)

# avg. and std. of scaled data
print("After scaling:")
print("Avg:", np.mean(X_norm, axis=0))
print("Std:", np.std(X_norm, axis=0))
```

```
After scaling:
Avg: [ 2.94396218e-16 -6.73618464e-17]
Std: [1. 1.]
```

OnehotEncoder

Good approach to represent categorical features

Fruits	Label Encoding	One-hot Encoding
Apple	1	[0, 1]
Banana	2	[1, 0]

In [ ]:
```
df.head()
```

Out[ ]:

	Pclass	Sex	SibSp	Parch	Embarked	Age	Fare	Survived
0	3	male	1	0	S	22.0	7.2500	0
1	1	female	1	0	C	38.0	71.2833	1
2	3	female	0	0	S	26.0	7.9250	1
3	1	female	1	0	S	35.0	53.1000	1
4	3	male	0	0	S	35.0	8.0500	0

In [ ]:

```python
from sklearn.preprocessing import OneHotEncoder

X_cate = categorical_features.values

print('All categories:\n')
for j in range(X_cate.shape[1]):
    print(np.unique(X_cate[:,j]))

encoder = OneHotEncoder()
encoder.fit(X_cate)
X_onehot = encoder.transform(X_cate).toarray()
print('shape of one hot encoded data',X_onehot.shape)
X_onehot
```

```
All categories:

[1 2 3]
['female' 'male']
['C' 'Q' 'S']
[0 1 2 3 4 5]
[0 1 2 3 4 5 6]
shape of one hot encoded data (712, 21)
```

Out[ ]:

```
array([[0., 0., 1., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       ...,
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.]])
```

In [ ]:

```python
# Decode the onehot data
print(encoder.inverse_transform(X_onehot))
print()
print(X_cate)
```

```
[[3 'male' 'S' 1 0]
 [1 'female' 'C' 1 0]
 [3 'female' 'S' 0 0]
 ...
 [1 'female' 'S' 0 0]
 [1 'male' 'C' 0 0]
 [3 'male' 'Q' 0 0]]

[[3 'male' 'S' 1 0]
 [1 'female' 'C' 1 0]
 [3 'female' 'S' 0 0]
 ...
 [1 'female' 'S' 0 0]
 [1 'male' 'C' 0 0]
 [3 'male' 'Q' 0 0]]
```

1.2 Data splitting

Overfitting

No description has been provided for this image

Split data to Train/Validation/Test set can resolve this issue to some extent:

No description has been provided for this image

No description has been provided for this image

Prepare the Titanic data:

In [ ]:
```python
# combine categorical & continuous features
X = np.hstack((X_norm, X_onehot))

# it is recommended to reshape the outputs
# to (n_samples, 1) in order to avoid unexpected broadcasting
Y = df['Survived'].values.reshape(-1, 1)

# print the dimensions
print(X.shape, Y.shape)
```

```
(712, 23) (712, 1)
```

Train-test split

In [ ]:
```python
from sklearn.model_selection import train_test_split

X_train,X_test,Y_train,Y_test = train_test_split(X,Y,
                                                 test_size=0.2,
                                                 random_state=42)
print(X_train.shape, Y_train.shape)
print(X_test.shape, Y_test.shape)
```

```
(569, 23) (569, 1)
(143, 23) (143, 1)
```

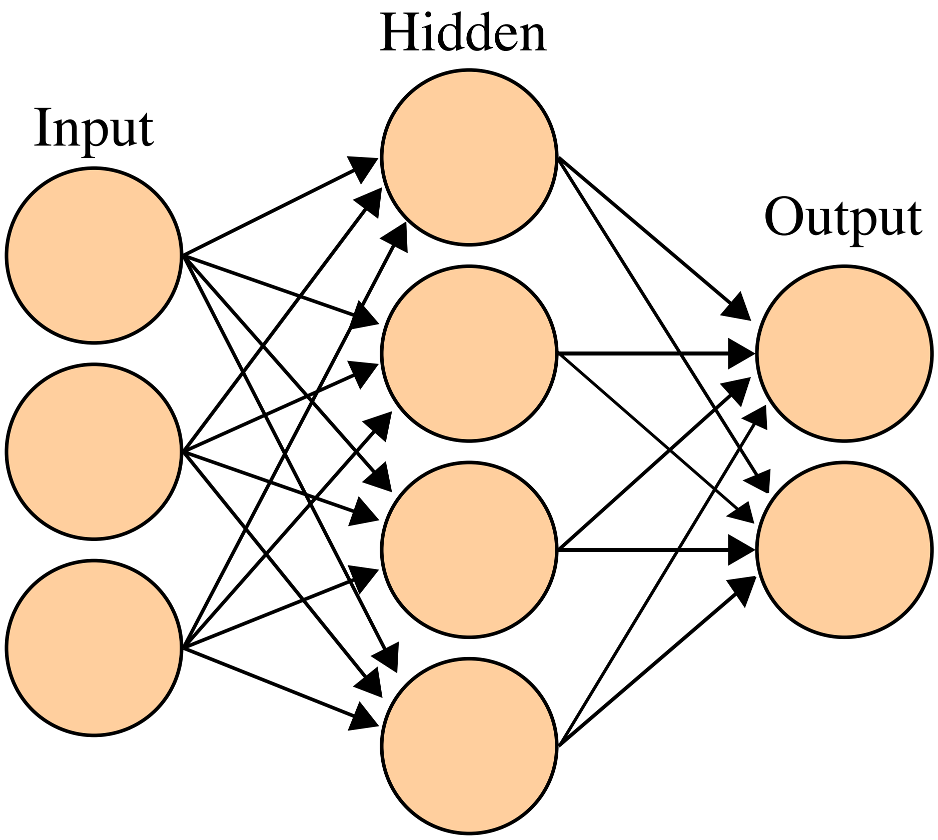
K-Fold

No description has been provided for this image

```
In [ ]: from sklearn.model_selection import KFold

        kf = KFold(n_splits=5,shuffle=True)
        for i, (train_index, val_index) in enumerate(kf.split(X[:20])):
            print(f"\nFold {i+1}:")
            print(f"  Train:       index={train_index}")
            print(f"  Validation:  index={val_index}")
```

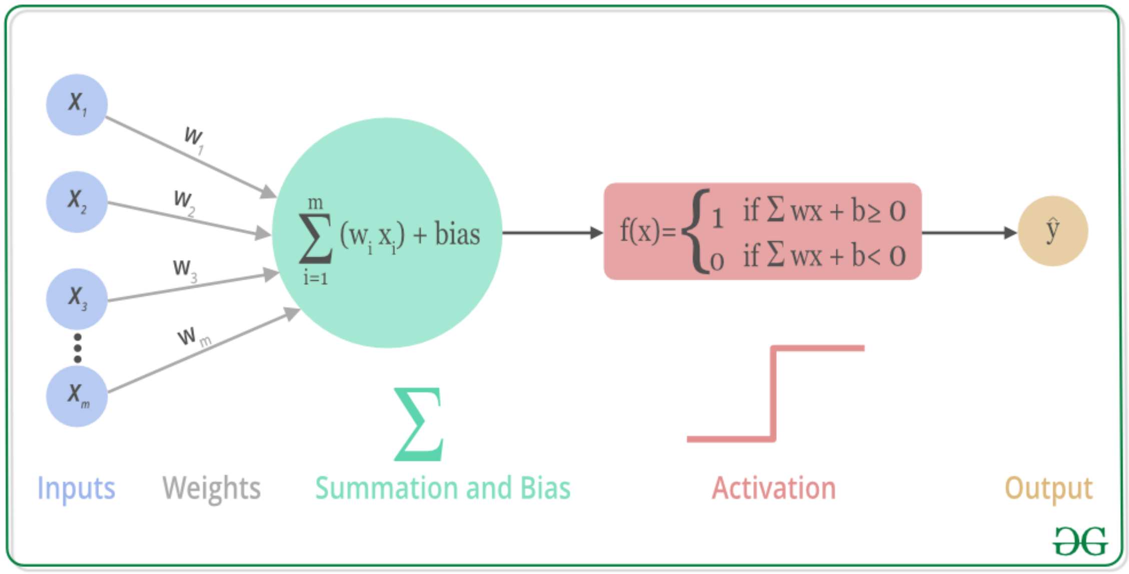
```
Fold 1:
  Train:       index=[ 0  1  2  3  4  5  8  9 10 11 12 13 15 16 18 19]
  Validation:  index=[ 6  7 14 17]

Fold 2:
  Train:       index=[ 2  4  5  6  7  8  9 10 11 12 13 14 15 17 18 19]
  Validation:  index=[ 0  1  3 16]

Fold 3:
  Train:       index=[ 0  1  3  5  6  7  9 10 12 13 14 15 16 17 18 19]
  Validation:  index=[ 2  4  8 11]

Fold 4:
  Train:       index=[ 0  1  2  3  4  5  6  7  8 10 11 14 16 17 18 19]
  Validation:  index=[ 9 12 13 15]

Fold 5:
  Train:       index=[ 0  1  2  3  4  6  7  8  9 11 12 13 14 15 16 17]
  Validation:  index=[ 5 10 18 19]
```

2. ANN



In [ ]:
```
import matplotlib.pyplot as plt


def plot(func, name):
    x = np.linspace(-5, 5, 200)
    y = func(x)
    fig, ax = plt.subplots(1, 1, figsize=(4, 3))
    ax.plot(x, y)
    ax.grid(True)
    ax.set_title(name)
```

2.1 Activation Function



x_1
x_2
x_3
x_m
w_1
w_2
w_3
w_m
$\sum_{i=1}^{m}(w_i x_i) + \text{bias}$
$f(x)=\begin{cases}1 & \text{if } \sum wx+b \geq 0\\ 0 & \text{if } \sum wx+b < 0\end{cases}$
$\hat{y}$
Inputs
Weights
Summation and Bias
Activation
Output

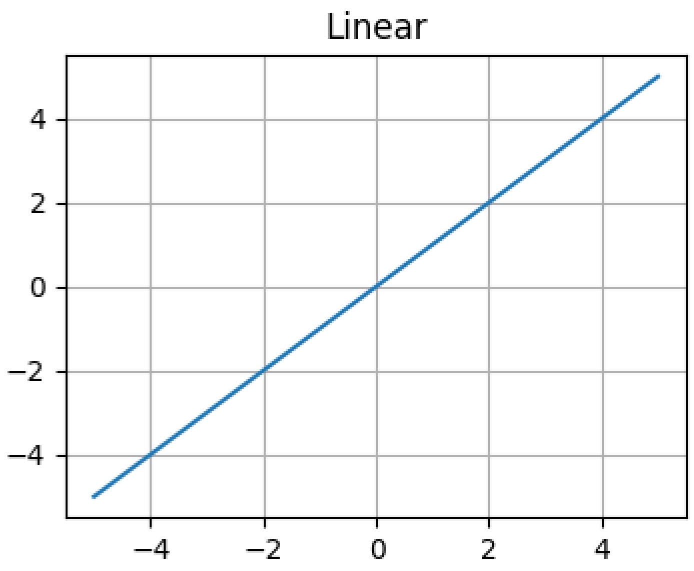
TODO: Fill in the following

Linear

$$z(x) = x$$

$$z'(x) = 1$$

```
In [ ]: plot(lambda x: x, "Linear")
```

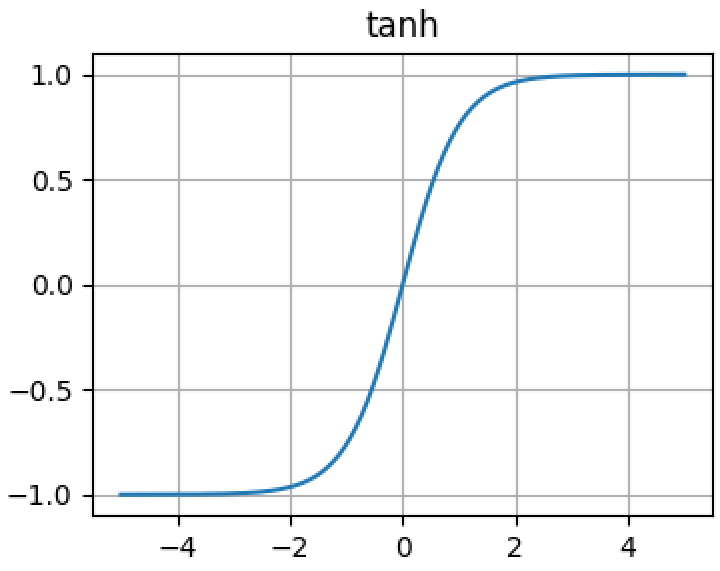


Tanh

$$z(x) = \tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$z'(x) = 1 - \tanh^2 x$$

In [ ]:
```
plot(lambda x: np.tanh(x), "tanh")
```



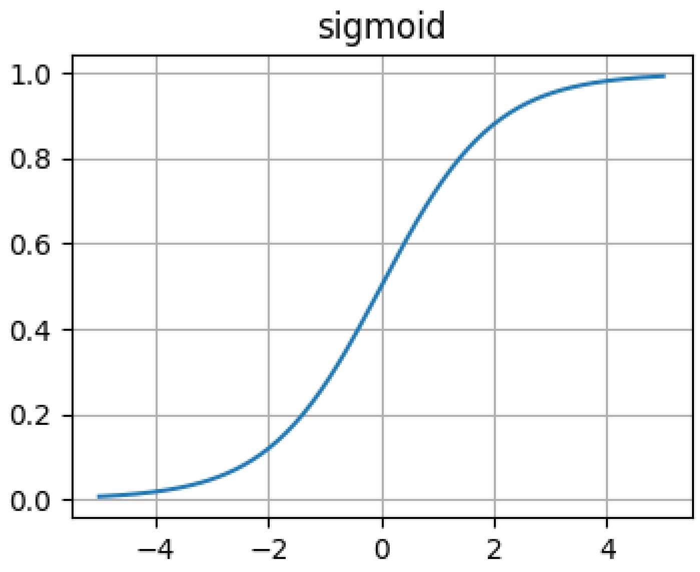
Sigmoid

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

$$z'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = z(x)[1-z(x)]$$

Hint for HW3: You are going to use this in Logistic Regression

In [ ]:
```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

plot(sigmoid, "sigmoid")
```

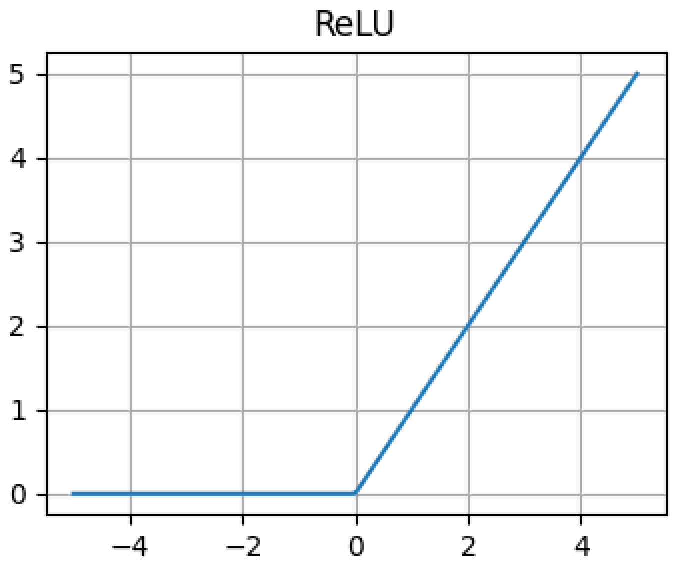


ReLU

$$z(x) = \max(0, x)$$

$$z'(x) = \begin{cases} 1 & x > 0 \\ 0 & x < 0 \end{cases}$$

In [ ]:
```
def relu(x):
    # return np.maximum(0, x)
    return x * (x > 0)

plot(relu, "ReLU")
```



More activation functions

Check here