Neural Networks

Dr. Ashish Tendulkar

IIT Madras

Machine Learning Practice

- In this week, we will study how to implement Multilayer Perceptron neural network models for classification and regression tasks with sklearn.

Multilayer Perceptron (MLP)

- It is a supervised learning algorithm.

- MLP learns a non-linear function approximator for either classification or regression depending on the given dataset.

- In sklearn, we implement MLP using:

1. MLPClassifier for classification
2. MLPRegressor for regression

- MLPClassifier supports multi-class classification by applying Softmax as the output function.
- It also supports supports multi-label classification in which a sample can belong to more than one class.

- MLPRegressor also supports multi-output regression, in which a sample can have more than one target.

Training data

Array X : holds the training samples

shape $\rightarrow$ (n_samples, n_features)

Array y : holds the target

shape $\rightarrow$ (n_samples,)

MLPClassifier

- How to implement MLPClassifier?

Step 1: Instantiate a MLP classifier estimator.

```
1 from sklearn.neural_network import MLPClassifier
2 MLP_clf = MLPClassifier()
```

Step 2: Call fit method on MLP classifier object with training feature matrix and label vector as arguments.

```
1 # Model training with feature matrix X_train and
2 # label vector or matrix y_train
3 MLP_clf.fit(X_train, y_train)
```

MLPClassifier

Step 3: After fitting (training), the model can make predictions for new samples (X_test) using two methods:

```
1 MLP_clf.predict(X_test)
2 MLP_clf.predict_proba(X_test)
```

predict →

- gives labels for new samples
- for example:

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

predict_proba →

- gives vector of probability estimates per sample
- for example:

```
array([1.967...e-04, 9.998...-01])
```

- MLPClassifier supports only the Cross-Entropy loss function

How to set the number of hidden layers?

`hidden_layer_sizes`

- This parameter sets the number of layers and the number of neurons in each layer.

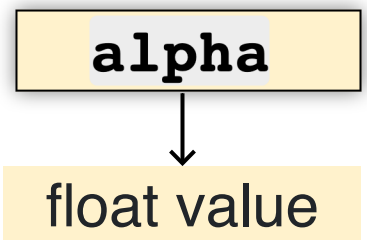
- It is a **tuple** where each element in the tuple represents the number of neurons at the i th position where i is the index of the tuple.
- The length of tuple denotes the total number of hidden layers in the network.

To create a 3 hidden layer neural network with 15 neurons in first layer, 10 neurons in second layer and 5 neurons in third layer:

```
1 MLPClassifier(hidden_layer_sizes=(15,10,5))
```

How to perform regularization in MLPClassifier?

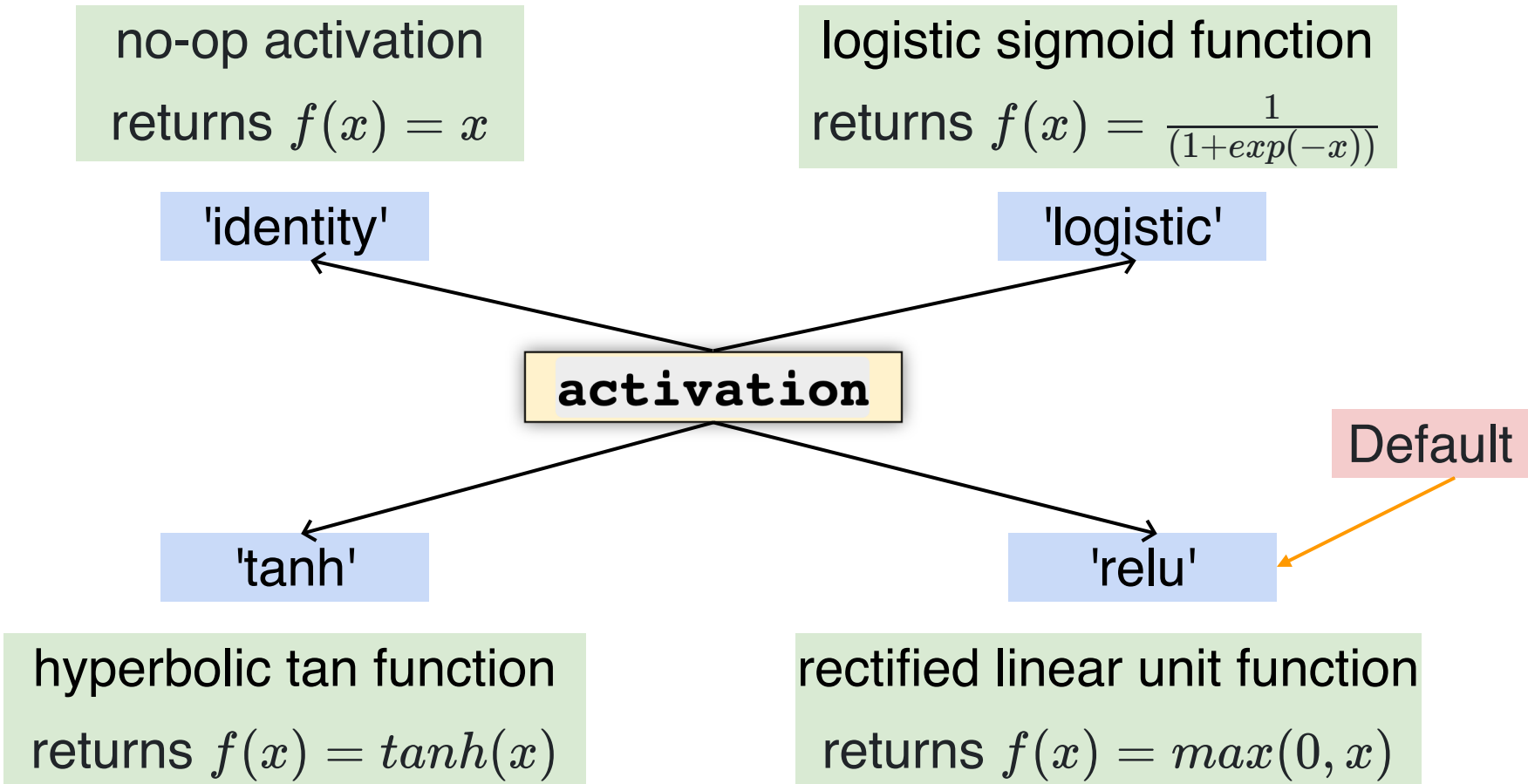
- The alpha parameter sets L2 penalty Regularization parameter



Default:

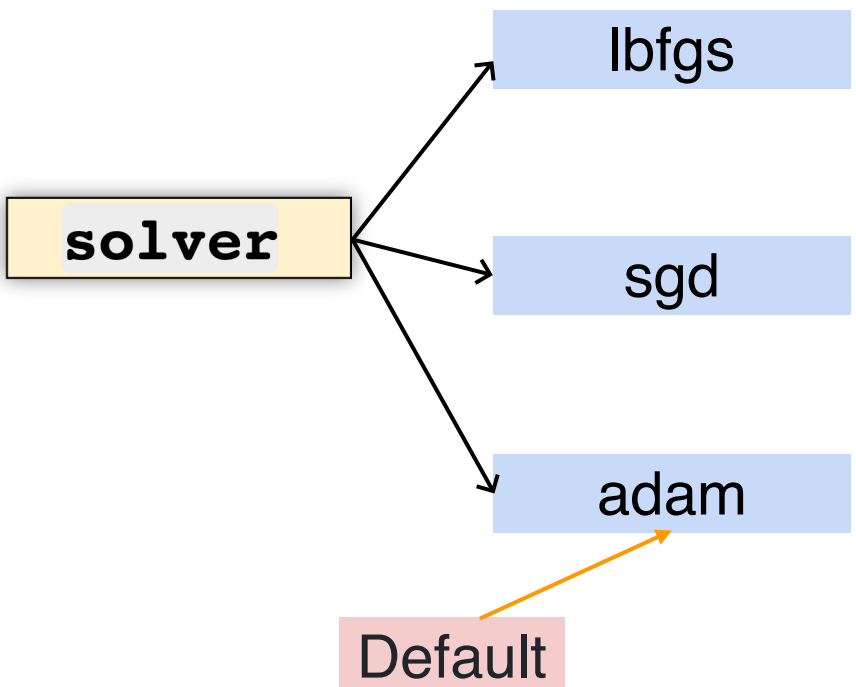
```
1 alpha = 0.0001
```

How to set the activation function for the hidden layers?



How to perform weight optimization in MLPClassifier?

- MLPClassifier optimizes the log-loss function using LBFGS or stochastic gradient descent



- If the **`solver`** is 'lbfgs', the classifier will not use minibatch.
- Size of minibatches can be set to other stochastic optimizers: **`batch_size`** (int)

- default batch_size is 'auto'.

```
1 batch_size=min(200, n_samples)
```

How to view weight matrix coefficients of trained MLPClassifier?

coefs_

- It is a list of shape (n_layers - 1,)
- The i th element in the list represents the weight matrix corresponding to layer i.

Example:

- "weights between input and first hidden layer:"

```
1 print(MLP_clf.coefs_[0])
```

- "weights between first hidden and second hidden layer:"

```
1 print(MLP_clf.coefs_[1])
```

```
weights between input and first hidden layer:
[[-0.14203691 -1.18304359 -0.85567518 -4.53250719 -0.60466275]
 [-0.69781111 -3.5850093  -0.26436018 -4.39161248  0.06644423]]

weights between first hidden and second hidden layer:
[[ 0.29179638 -0.14155284]
 [ 4.02666592 -0.61556475]
 [-0.51677234  0.51479708]
 [ 7.37215202 -0.31936965]
 [ 0.32920668  0.64428109]]
```

How to view bias vector of trained MLPClassifier?

`intercepts_`

- It is a list of shape (n_layers - 1,)
- The i th element in the list bias vector corresponding to layer $i + 1$.

Example:

- "Bias values for first hidden layer:"

```
1 print(MLP_clf.intercepts_[0])
```

- "Bias values for second hidden layer:"

```
1 print(MLP_clf.intercepts_[1])
```

```
Bias values for first hidden layer:
[-0.14962269 -0.59232707 -0.5472481   7.02667699 -0.87510813]

Bias values for second hidden layer:
[-3.61417672 -0.76834882]
```

Some parameters in MLPClassifier

`learning_rate`

- 'constant'
- 'invscaling'
- 'adaptive'

default: 'constant'

`learning_rate_init`

float value

default: 0.001

`power_t`

float value

default: 0.5

`max_iter`

int value

default: 500

- `learning_rate` and `power_t` are used only for `solver = 'sgd'`
- `learning_rate_init` is used when `solver='sgd'` or ‘adam’.
- `shuffle` is used to shuffle samples in each iteration when `solver='sgd' or 'adam'`
- `momentum` is used for gradient descent update when `solver='sgd'`

MLPRegressor

- MLPRegressor trains using backpropagation with no activation function in the output layer.
- Therefore, it uses the square error as the loss function, and the **output is a set of continuous values**.

The parameters of MLPRegressor are the same as that of MLPClassifier.

How to implement MLPRegressor?

Step 1: Instantiate a MLP regressor estimator.

```
1 from sklearn.neural_network import MLPRegressor
2 MLP_reg = MLPRegressor()
```

Step 2: Call fit method on MLP regressor object with training feature matrix and label vector as arguments.

```
1 # Model training with feature matrix X_train and
2 # label vector or matrix y_train
3 MLP_reg.fit(X_train, y_train)
```

Step 3: After fitting (training), the model can make predictions for new samples (X_test):

```
1 MLP_reg.predict(X_test)
```

- returns predicted values for new samples
- for example:

```
array([-0.9..., -7.1...])
```

```
1 MLP_reg.score(X_test,y_test)
```

- returns R^2 score
- for example:

```
0.45678889
```