Thư viện lập trình học máy

AI Academy Vietnam

scikit-learn

Dữ liệu trong scikit-learn

- Lưu trữ trong mảng hai chiều
- [n_samples, n_features]
- n_samples: Các đối tượng dữ liệu
- n_features: Các đặc trưng dữ liệu

Tập dữ liệu Iris

```
from sklearn.datasets import load_iris
iris = load_iris()
```

Tập dữ liệu Iris

```
from sklearn.datasets import load_iris
iris = load_iris()
```

```
iris.keys()
```

```
['target_names', 'data', 'target',
'DESCR', 'feature_names']
```

Tập dữ liệu Iris

```
n_samples, n_features = iris.data.shape
print n_samples
print n_features
print iris.data[0]
```

```
150
4
[ 5.1  3.5  1.4  0.2]
```

Tập dữ liệu Iris

```
print iris.data.shape
print iris.target.shape
```

```
(150, 4)
(150,)
```

Tập dữ liệu Iris

```
print iris.target
```

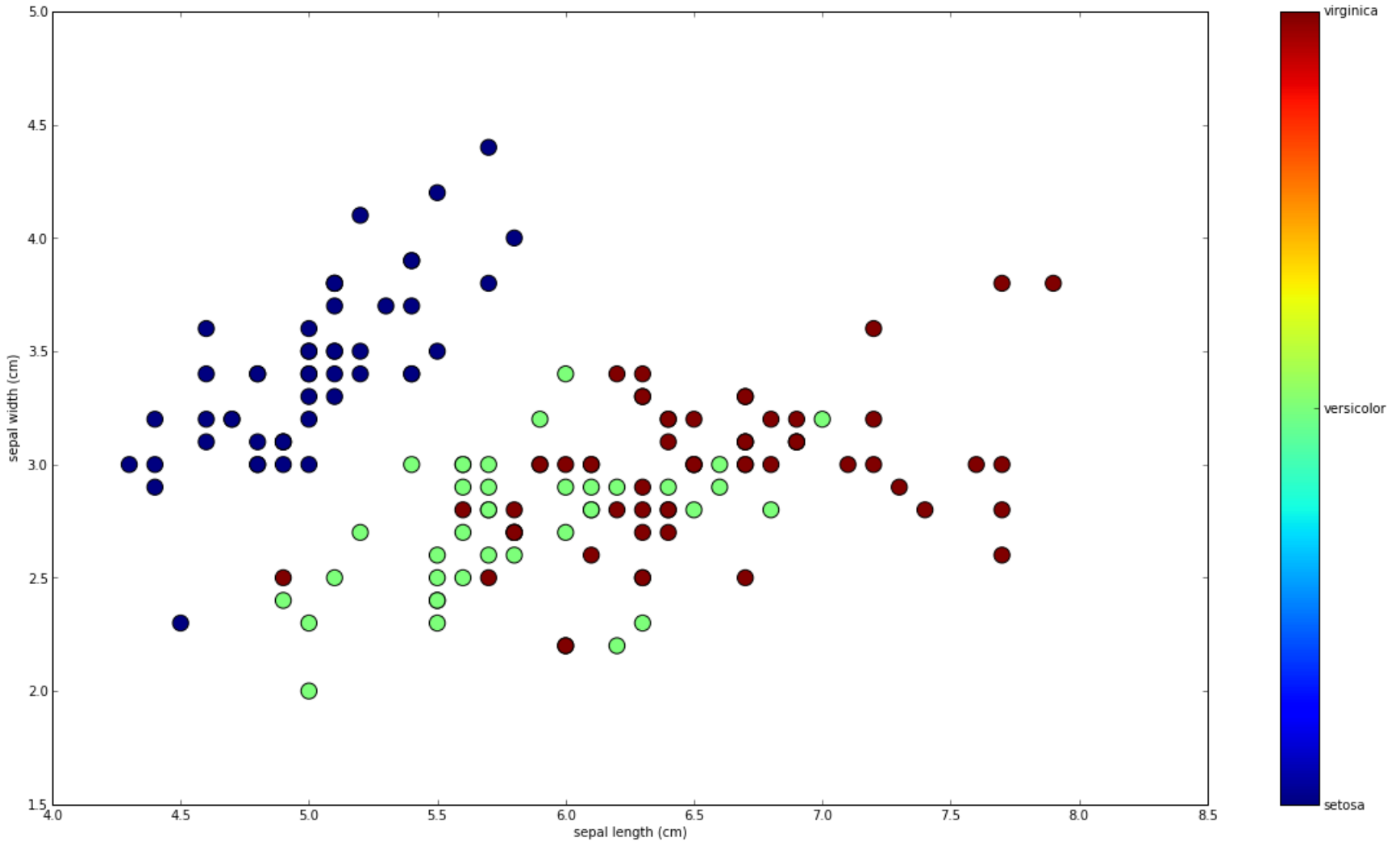
```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Tập dữ liệu Iris

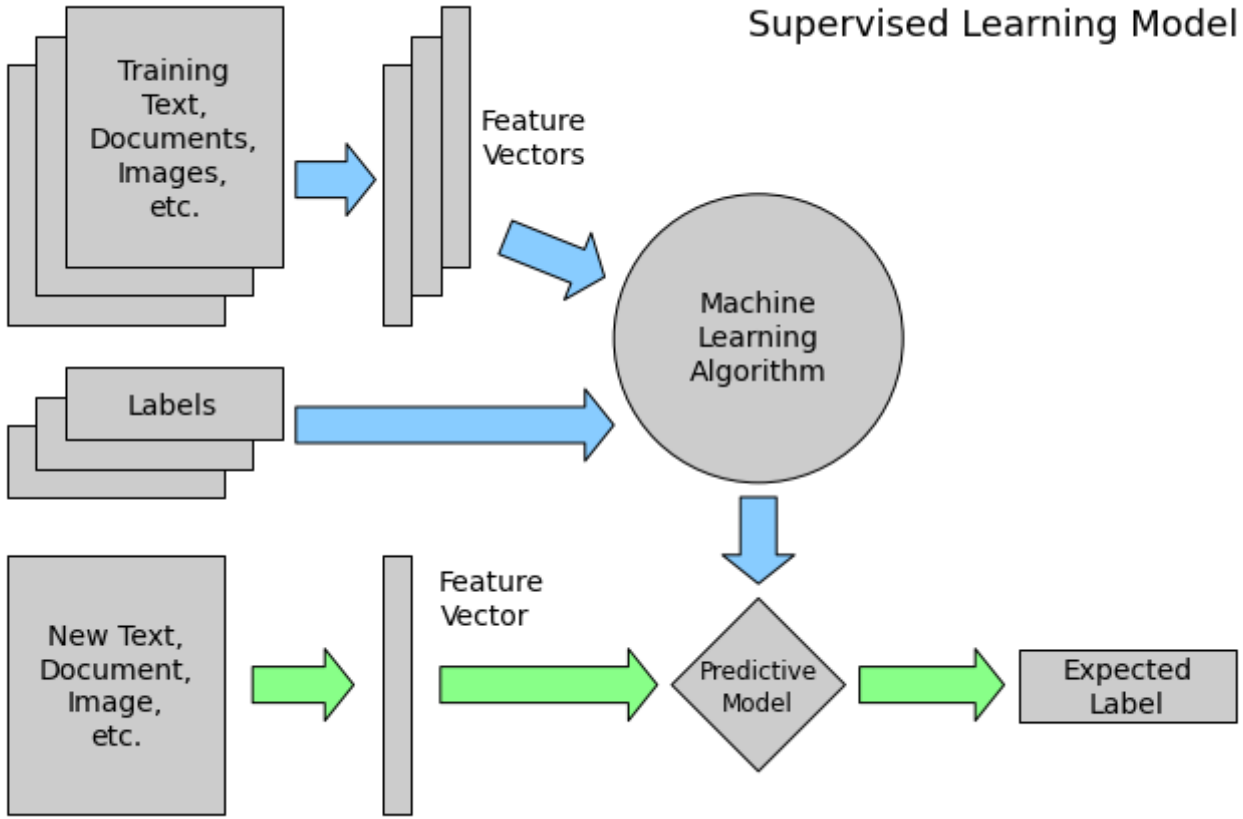
```
print iris.target_names
```

```
['setosa' 'versicolor' 'virginica']
```

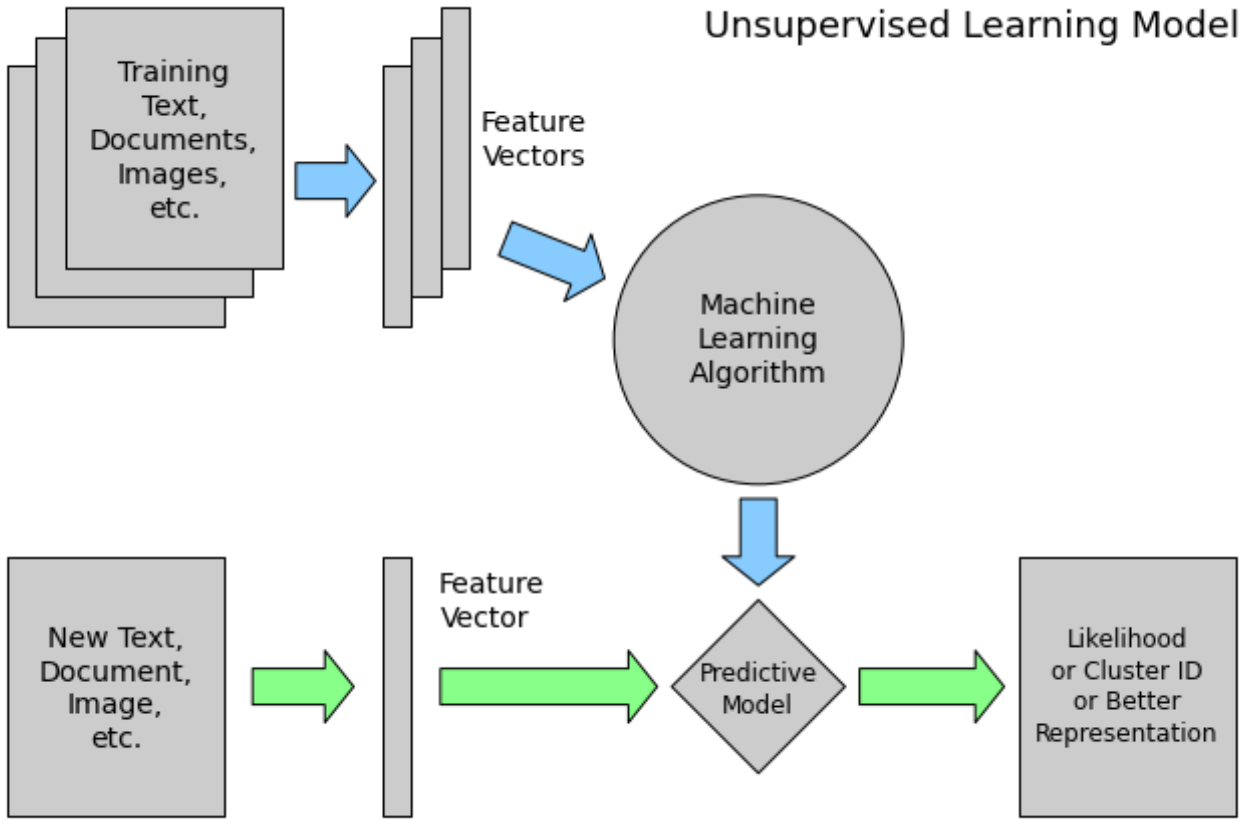
Trực quan hoá tập dữ liệu Iris



Machine Learning: Học giám sát



Machine Learning: Học không giám sát



Ví dụ trên Scikit-learn

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

```
print model
```

```
LinearRegression(copy_X=True, fit_intercept=True,
normalize=False)
```

Trích xuất đặc trưng

- Thông thường, dữ liệu phi cấu trúc và không phải dạng số
- Dữ liệu văn bản

Trích xuất đặc trưng

- Thông thường, dữ liệu phi cấu trúc và không phải dạng số
- Dữ liệu văn bản
- Dữ liệu hình ảnh

Trích xuất đặc trưng

- Thông thường, dữ liệu phi cấu trúc và không phải dạng số
- Dữ liệu văn bản
- Dữ liệu hình ảnh
- Dữ liệu âm thanh

Học có giám sát: Phân lớp

```
from sklearn.datasets import load_iris
iris = load_iris()
```

Học có giám sát: Phân lớp

```
from sklearn.datasets import load_iris
iris = load_iris()
```

```
X = iris.data
y = iris.target

print X.shape
print y.shape
```

```
(150, 4)
(150,)
```

Học có giám sát: Phân lớp

```
from sklearn.svm import LinearSVC
```

Học có giám sát: Phân lớp

```
from sklearn.svm import LinearSVC
```

```
clf = LinearSVC(loss='l2')
```

Học có giám sát: Phân lớp

```
from sklearn.svm import LinearSVC
```

```
clf = LinearSVC(loss='l2')
```

```
print clf
```

```
LinearSVC(C=1.0, class_weight=None,
dual=True, fit_intercept=True,
     intercept_scaling=1, loss=l2,
multi_class=ovr, penalty=l2,
     random_state=None, tol=0.0001,
verbose=0)
```

Học có giám sát: Phân lớp

```
clf = clf.fit(X, y)
```

Học có giám sát: Phân lớp

```
clf = clf.fit(X, y)
```

```
clf.coef_
```

```
array([[ 0.18423673,  0.45122616,
-0.80793496, -0.45071305],
```

```
clf.intercept_
```

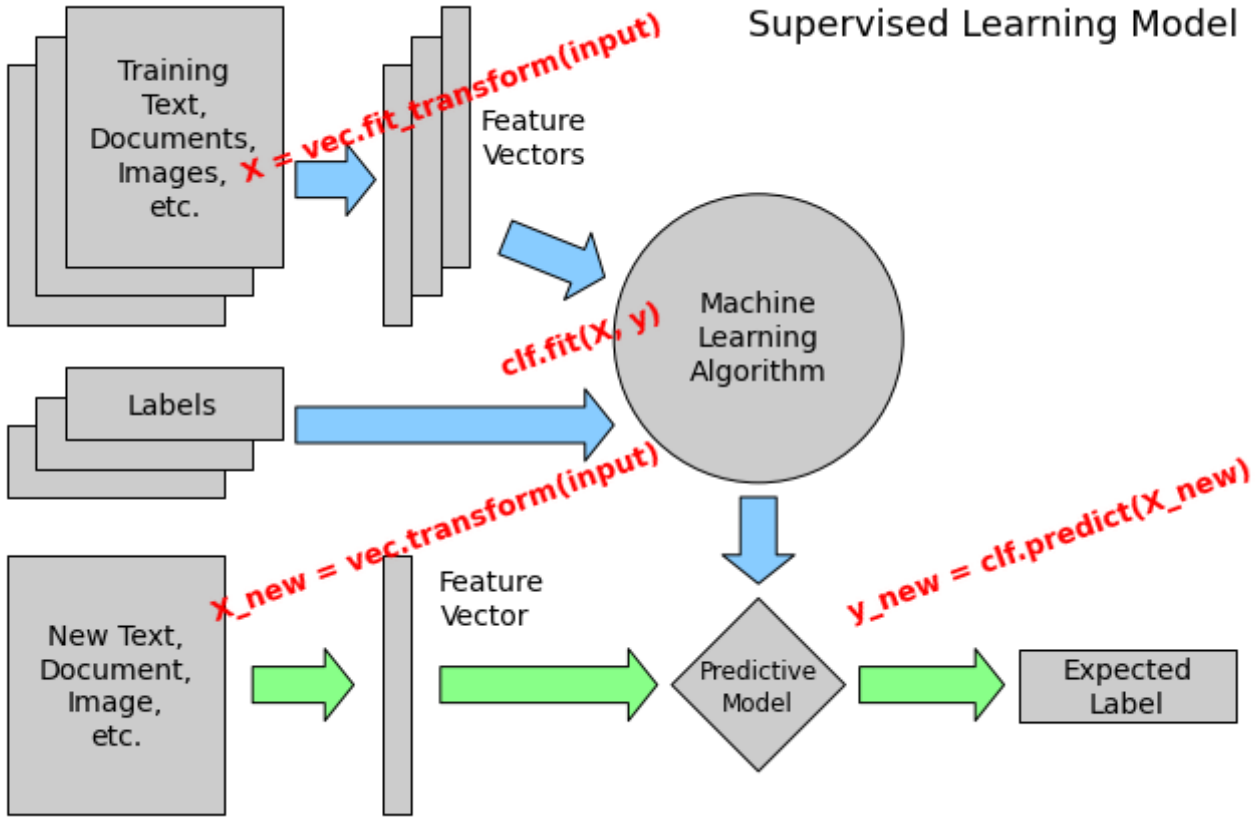
```
array([ 0.10956131,  1.66940723,
-1.70954847])
```

Học có giám sát: Phân lớp

```
X_new = [[ 5.0,  3.6,  1.3,  0.25]]

clf.predict(X_new)
```

```
array([0])
```

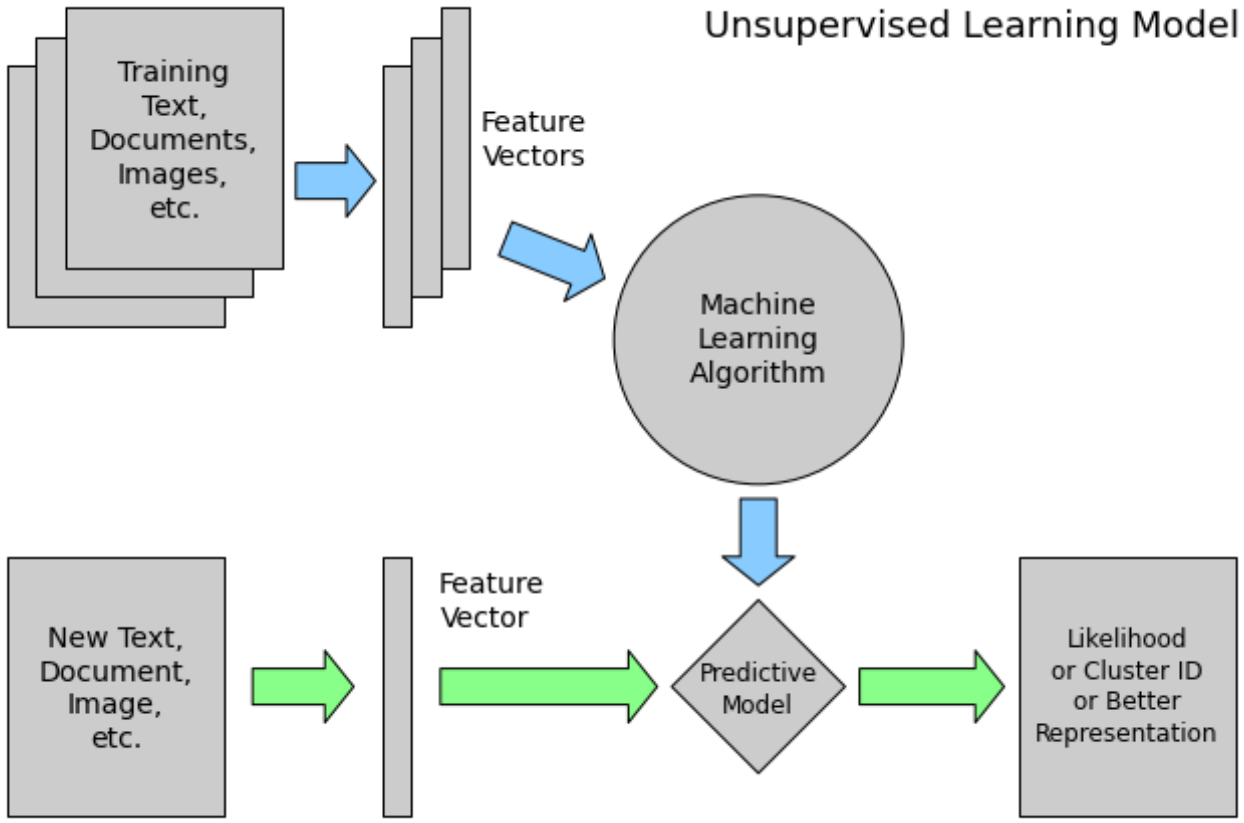
Học có giám sát: Phân lớp



Học có giám sát: Phân lớp

- Phân loại Email
- Phân loại ngôn ngữ
- Phân loại bản tin
- Phân tích ngữ nghĩa
- Nhận dạng khuôn mặt
- ...

Học không giám sát



Giảm chiều dữ liệu

```
X = iris.data
y = iris.target
```

Principal Component Analysis

```
X = iris.data
y = iris.target
```

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2, whiten=True)
pca.fit(X)
```

```
PCA(copy=True, n_components=2,
whiten=True)
```

Học không giám sát: PCA

```
X = iris.data
y = iris.target
```

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2, whiten=True)
pca.fit(X)
```

```
PCA(copy=True, n_components=2,
whiten=True)
```

```
X_pca = pca.transform(X)
```

Học không giám sát : PCA

```
import numpy as np
np.round(X_pca.mean(axis=0), decimals=5)
```

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

Học không giám sát: PCA

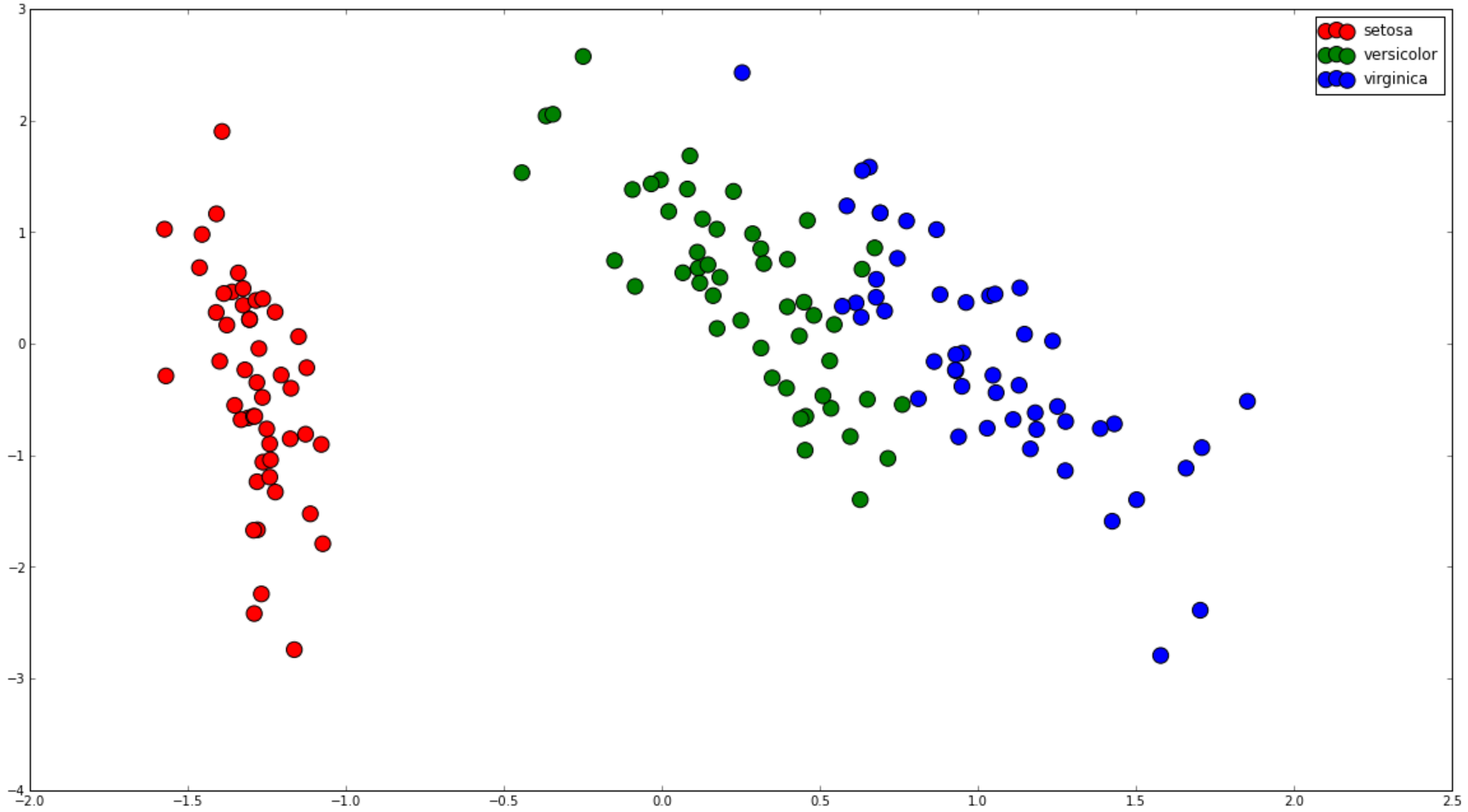
```
import numpy as np
np.round(X_pca.mean(axis=0), decimals=5)
```

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

```
np.round(X_pca.std(axis=0), decimals=5)
```

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

Học không giám sát: PCA



Validation & Testing

```
# Get the data
X = iris.data
y = iris.target
```

Validation & Testing

AI Academy Vietnam

```
# Get the data
X = iris.data
y = iris.target
```

```
# Instantiate and train the classifier
clf = LinearSVC(loss='l2')
clf.fit(X, y)
```

Validation & Testing

AI Academy Vietnam

```
# Get the data
X = iris.data
y = iris.target
```

```
# Instantiate and train the classifier
clf = LinearSVC(loss='l2')
clf.fit(X, y)
```

```
# Check input vs. output labels
print clf.score(X, y)
```

```
0.966666666667
```

Overfitting

Cross-Validation

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = \
  train_test_split(X, y, test_size=0.25, random_state=0)
print X.shape, X_train.shape, X_test.shape
```

```
(150, 4) (112, 4) (38, 4)
```

Cross-Validation

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = \
  train_test_split(X, y, test_size=0.25, random_state=0)
print X.shape, X_train.shape, X_test.shape
```

```
(150, 4) (112, 4) (38, 4)
```

```
clf = LinearSVC(loss='l2').fit(X_train, y_train)
y_pred = clf.predict(X_test)
print (y_pred == y_test)
```

```
[ True  True  True  True  True  True  True  True  True
True  True  True
```

Cross-Validation

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = \
  train_test_split(X, y, test_size=0.25, random_state=0)
print X.shape, X_train.shape, X_test.shape
```

```
(150, 4) (112, 4) (38, 4)
```

```
clf = LinearSVC(loss='l2').fit(X_train, y_train)
y_pred = clf.predict(X_test)
print (y_pred == y_test)
```

```
[ True  True  True  True  True  True  True  True  True
True  True  True
```

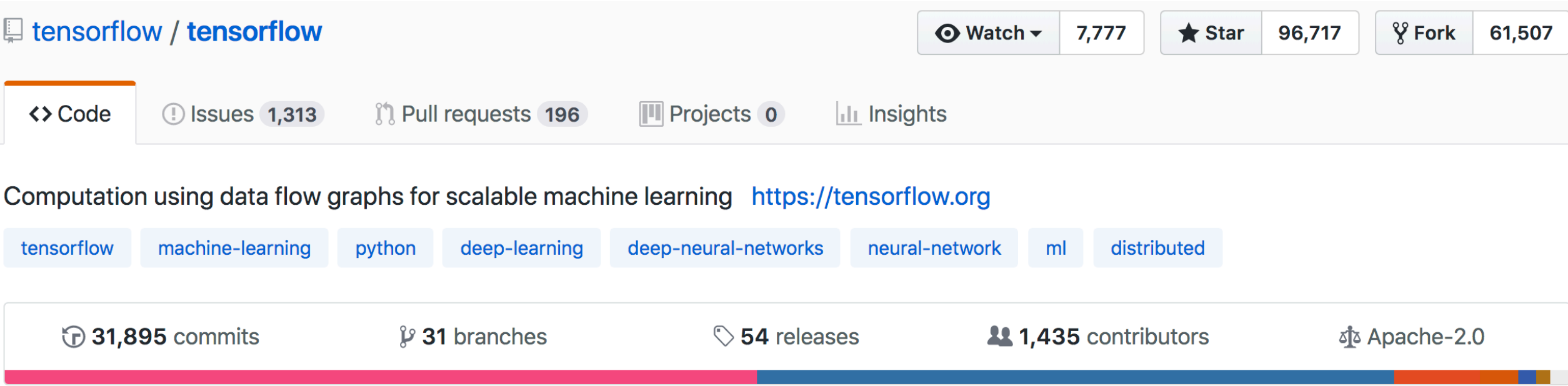
```
clf.score(X_test, y_test)
```

```
0.92105263157894735
```

TensorFlow

TensorFlow

AI Academy Vietnam



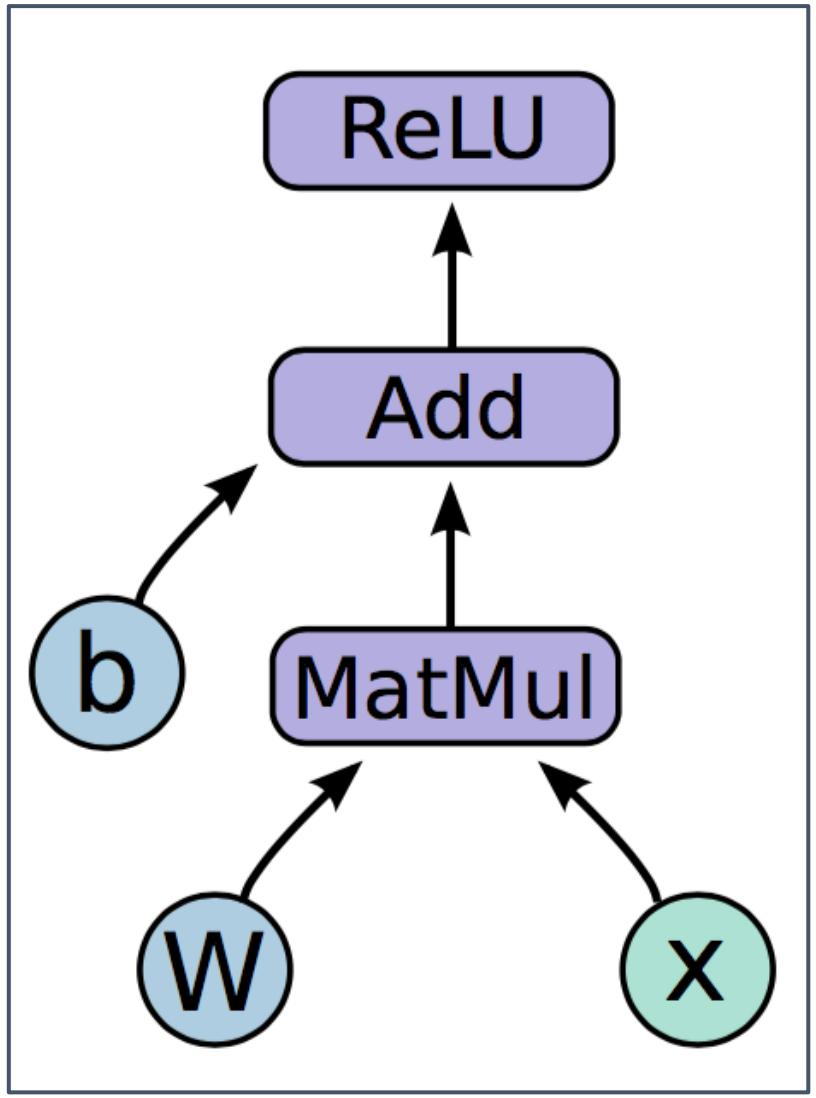
- Thư viện mã nguồn mở cho tính toán số học sử dụng đồ thị luông dữ liệu (data flow graphs)
- Được phát triển bởi Google Brain cho các nghiên cứu về học máy

TensorFlow là gì?

- **Ý tưởng chính**: mô tả các tính toán toán học trên đồ thị (**graph**)
- Các node của đồ thì là các phép tính toán (**operations**) với dữ liệu vào ra là các con số.
- Các cạnh được gọi là **tensors**, là luồng giữa các node.

Mô hình tính toán

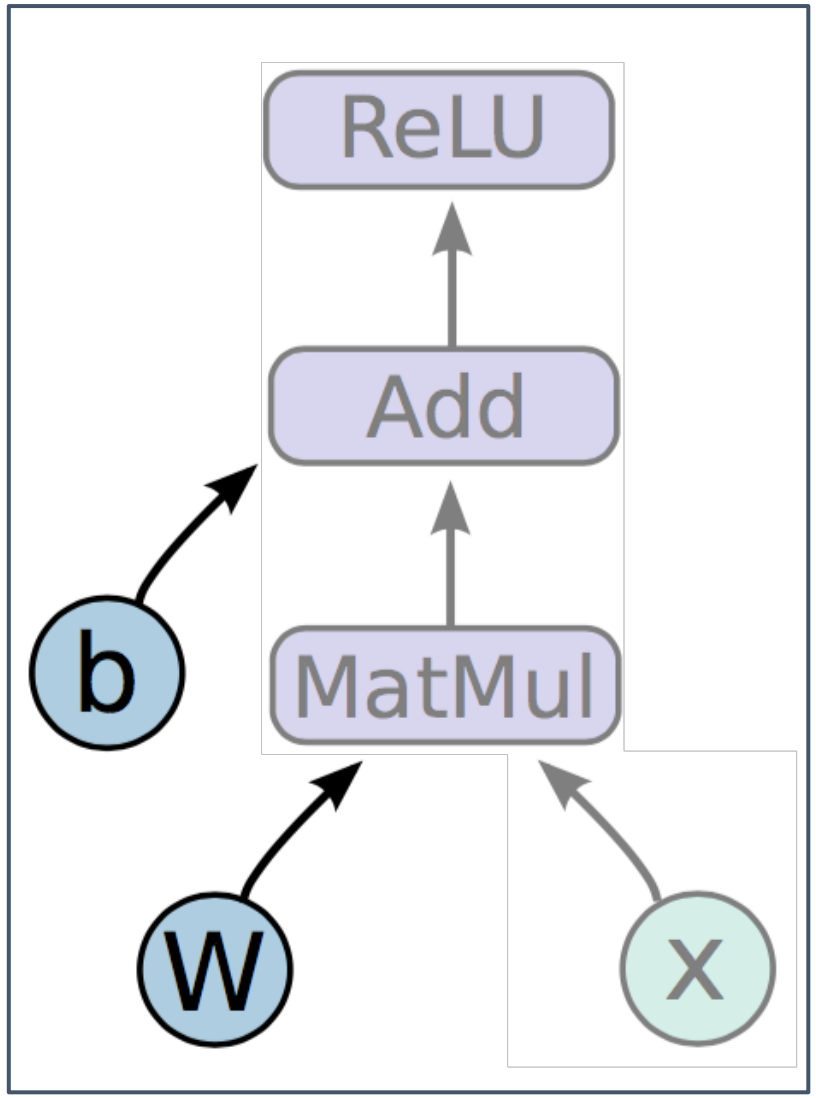
$$h = ReLU(Wx + b)$$



Mô hình tính toán

$$h = ReLU(Wx + b)$$

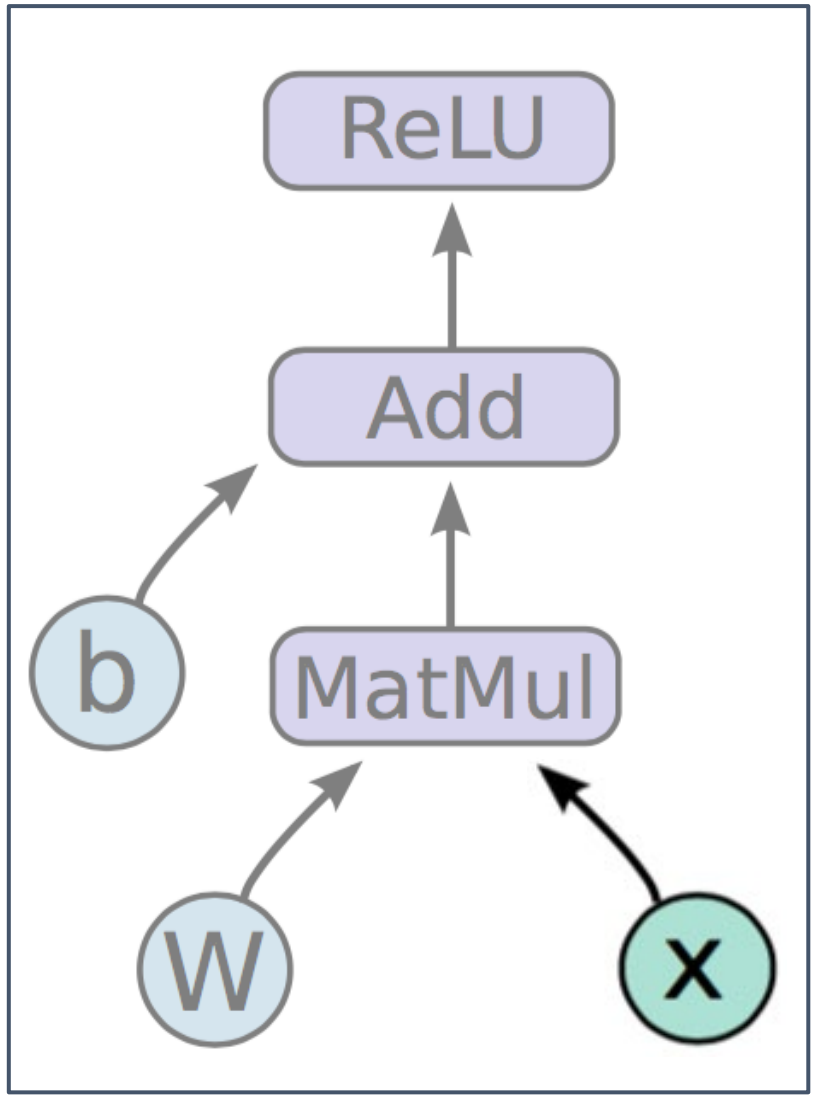
Variables (Biến số): là các node với output là giá trị hiện thời. Trạng thái sẽ được cố định qua các lần thực thi của đồ thị (có thể gọi là tham số)



Mô hình tính toán

$$h = ReLU(Wx + b)$$

Placeholders là các node trong đó giá trị được đưa vào trong mỗi lần thực thi tính toán đồ thị (input, nhãn ...)



Mô hình tính toán

$$h = ReLU(Wx + b)$$

Mathematical operations:

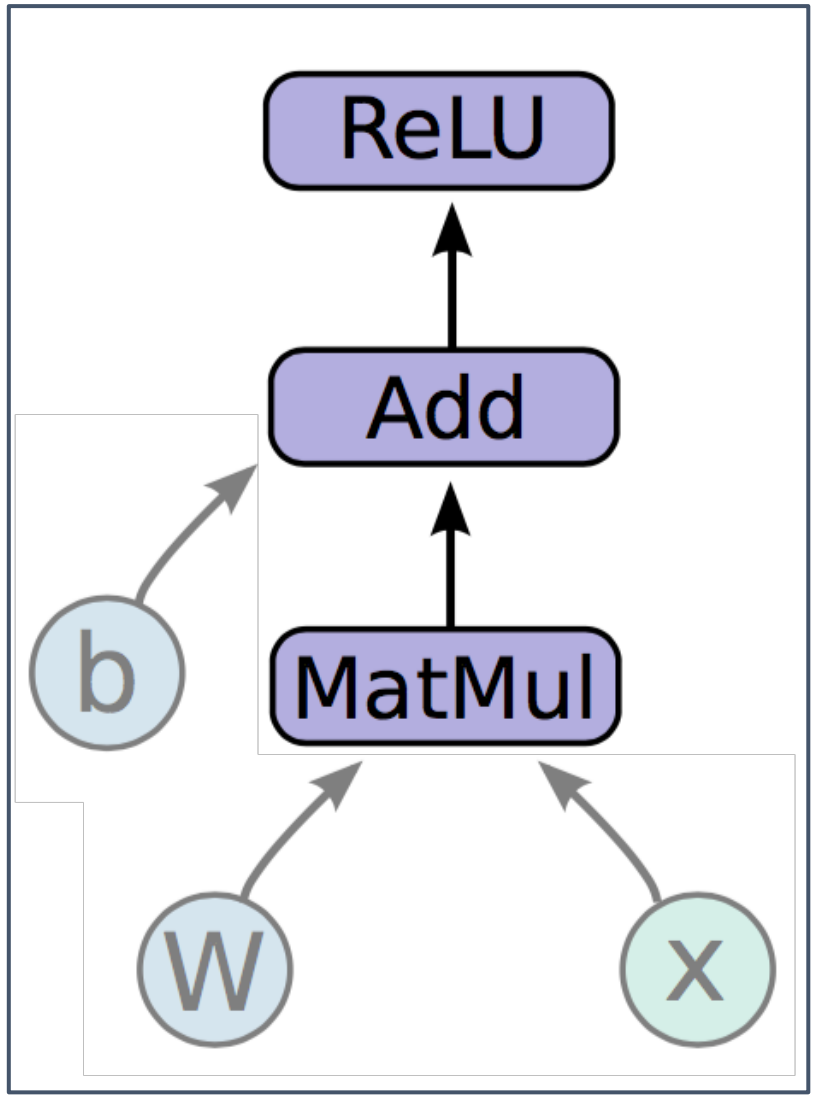
(Phép tính toán học)

MatMul: Nhân hai ma trận

Add: Cộng nguyên tố

ReLU: Hàm kích hoạt rectified linear function

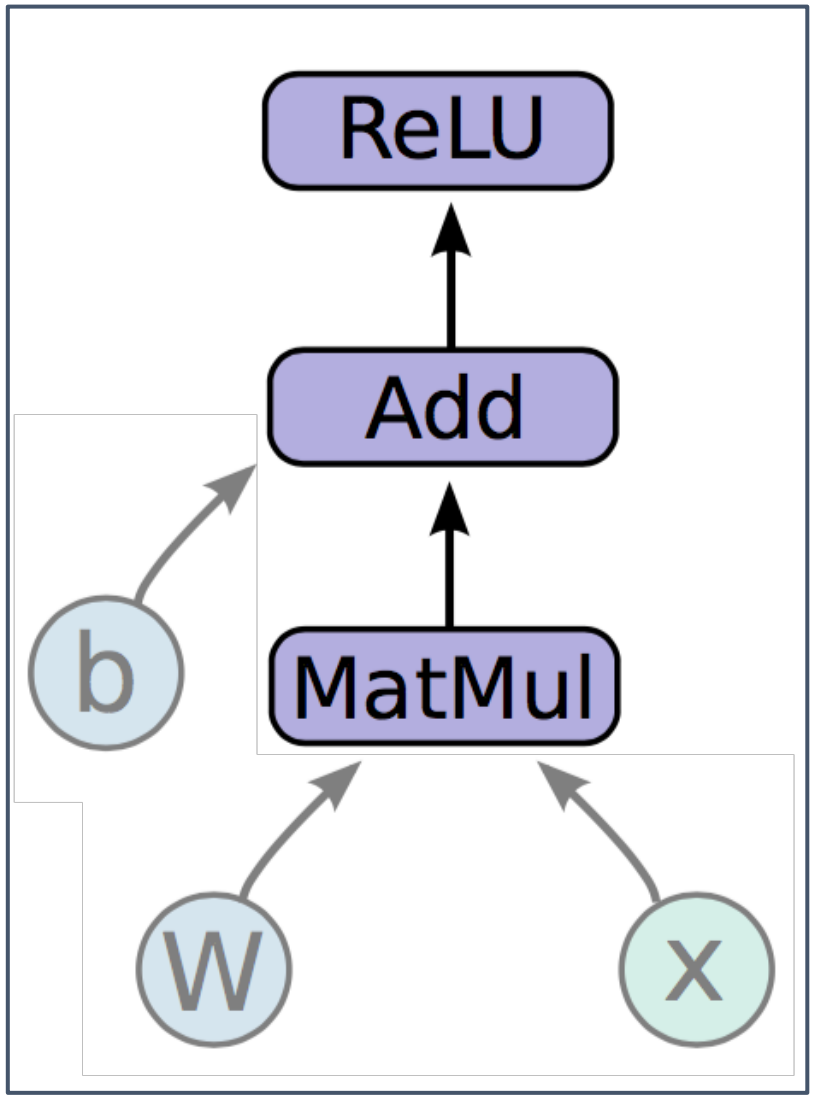
$$ReLu(x) = \begin{cases} 0, & x <= 0 \\ x, & x > 0 \end{cases}$$



Code ví dụ

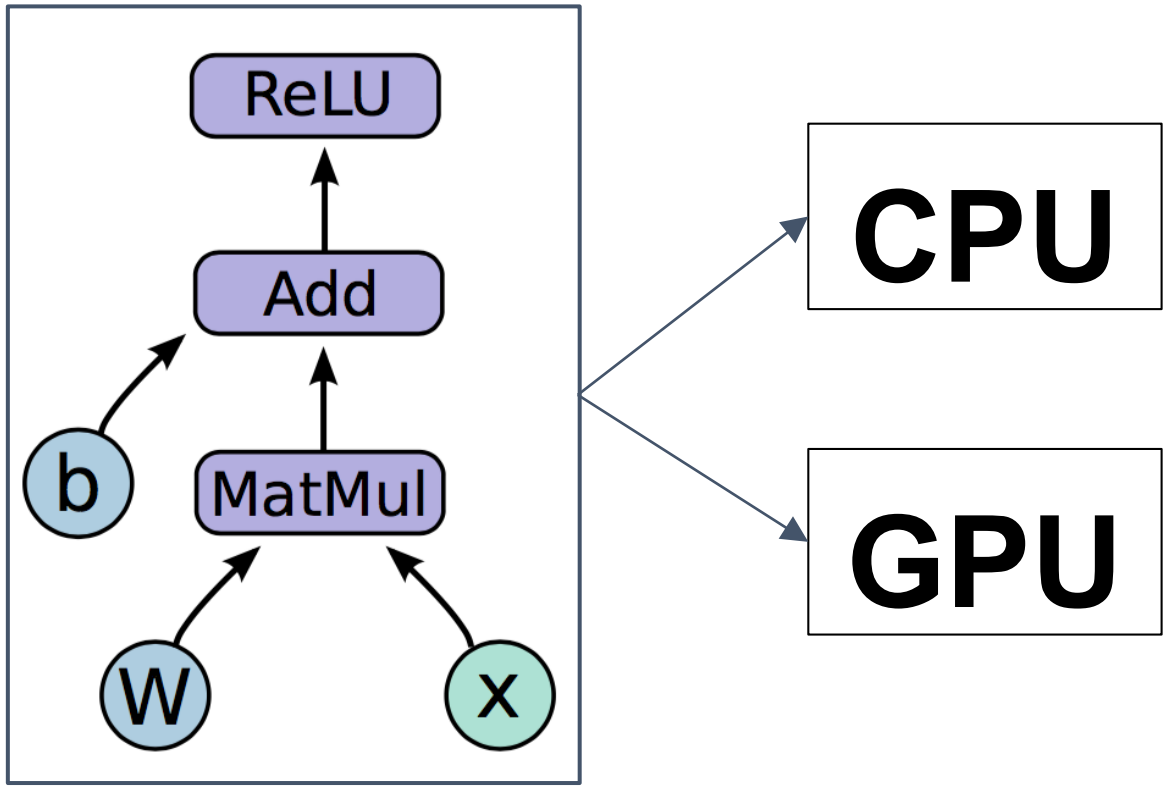
```
import tensorflow as tf

b = tf.Variable(tf.zeros((100,)))
W = tf.Variable(tf.random_uniform((784, 100), -1, 1))

x = tf.placeholder(tf.float32, (1, 784))

h = tf.nn.relu(tf.matmul(x, W) + b)
```

$$h = ReLU(Wx + b)$$



Thực thi trên đồ thị tính toán

Đồ thị tính toán được thực thi trên một phiên (**session**), là mỗi lần triển khai và thực thi trên CPU hoặc GPU



Huấn luyện mô hình

- Đã thảo luận:
  - Xây dựng đồ thị tính toán sử dụng **variables** và **placeholders**
  - Triển khai đồ thị lên các phiên tínht oán

- Tiếp tục: huấn luyện mô hình
  - Định nghĩa hàm mất mát (hàm giá - loss function)
  - Tính gradients

Hàm mất mát

- Sử dụng **placeholder** như là nhãn (**labels)**
- Xây dựng loss node sử dụng nhãn và dự đoán (**prediction)**

```
prediction = tf.nn.softmax(...)  #Output of neural network
label = tf.placeholder(tf.float32, [100, 10])

cross_entropy = -tf.reduce_sum(label * tf.log(prediction), axis=1)
```

Lan truyền ngược (Backpropagation)

```
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

`tf.train.GradientDescentOptimizer` là phương thức tối tưu (**Optimizer)**

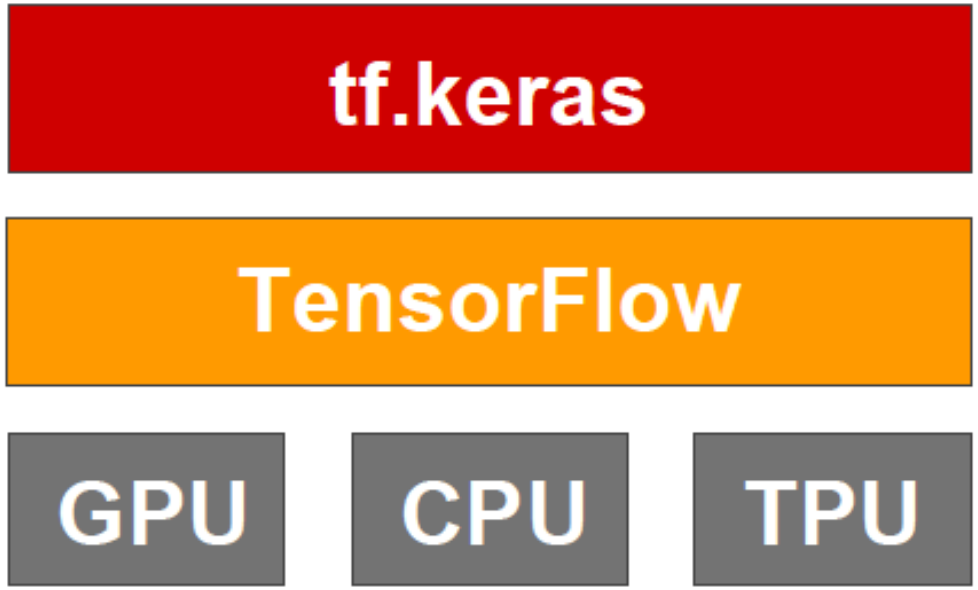
```
tf.train.GradientDescentOptimizer(lr).minimize(cross_entropy)
Đưa phương thưc tối ưu (optimization operation) vào đồ thị
  tínht toán
```

Tensoflow sẽ tự động tính toán gradient bằng phương ppháp lan truyền ngược (backpropagation)

Keras

High-level API của TensorFlow

- Trở thành một phần của TensorFlow từ v1.4
- Đầy đủ các API của Keras
- Các bộ tối ưu tốt hơn TF
- Đầy đủ các tính năng đặc trưng của TF
  - execution



Điểm đặc biệt của Keras

- Tập trung vào trải nghiệm của người dùng
- Công đồng khoa học và ứng dụng lớn
- Đa dạng backend và platform
- Dễ dàng tái sử dụng các models

300,000 Keras developer

Ba kiểu Keras API

- The Sequential Model
  - Sử dụng dễ dàng
  - Dành cho single-input, single-output, các lớp chồng lên nhau của network
  - Ứng dụng trong hơn 70% các trường hợp
- The functional API
  - Giống như chơi Lego với các khối
  - Dành cho multi-input, multi-output, các cấu trúc đồ thị khác nhau
  - Ứng dụng trong hơn 95% trường hợp
- Model subclassing
  - Tuỳ biến mềm dẻo tối đa
  - Dễ phát sinh các lỗi trong quá trình phát triển

Sequential API

```
import keras
from keras import layers

model = keras.Sequential()
model.add(layers.Dense(20, activation='relu', input_shape=(10,)))
model.add(layers.Dense(20, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

model.fit(x, y, epochs=10, batch_size=32)
```

Functional API

```
import keras
from keras import layers

inputs = keras.Input(shape=(10,))
x = layers.Dense(20, activation='relu')(x)
x = layers.Dense(20, activation='relu')(x)
outputs = layers.Dense(10, activation='softmax')(x)

model = keras.Model(inputs, outputs)
model.fit(x, y, epochs=10, batch_size=32)
```

Model subclassing

```
import keras
from keras import layers

class MyModel(keras.Model):

    def __init__(self):
        super(MyModel, self).__init__()
        self.dense1 = layers.Dense(20, activation='relu')
        self.dense2 = layers.Dense(20, activation='relu')
        self.dense3 = layers.Dense(10, activation='softmax')

    def call(self, inputs):
        x = self.dense1(x)
        x = self.dense2(x)
        return self.dense3(x)

model = MyModel()
model.fit(x, y, epochs=10, batch_size=32)
```

Q&A

Thank you!