高等机器学习

机器学习流程

柯国霖
微软亚洲研究院

Microsoft

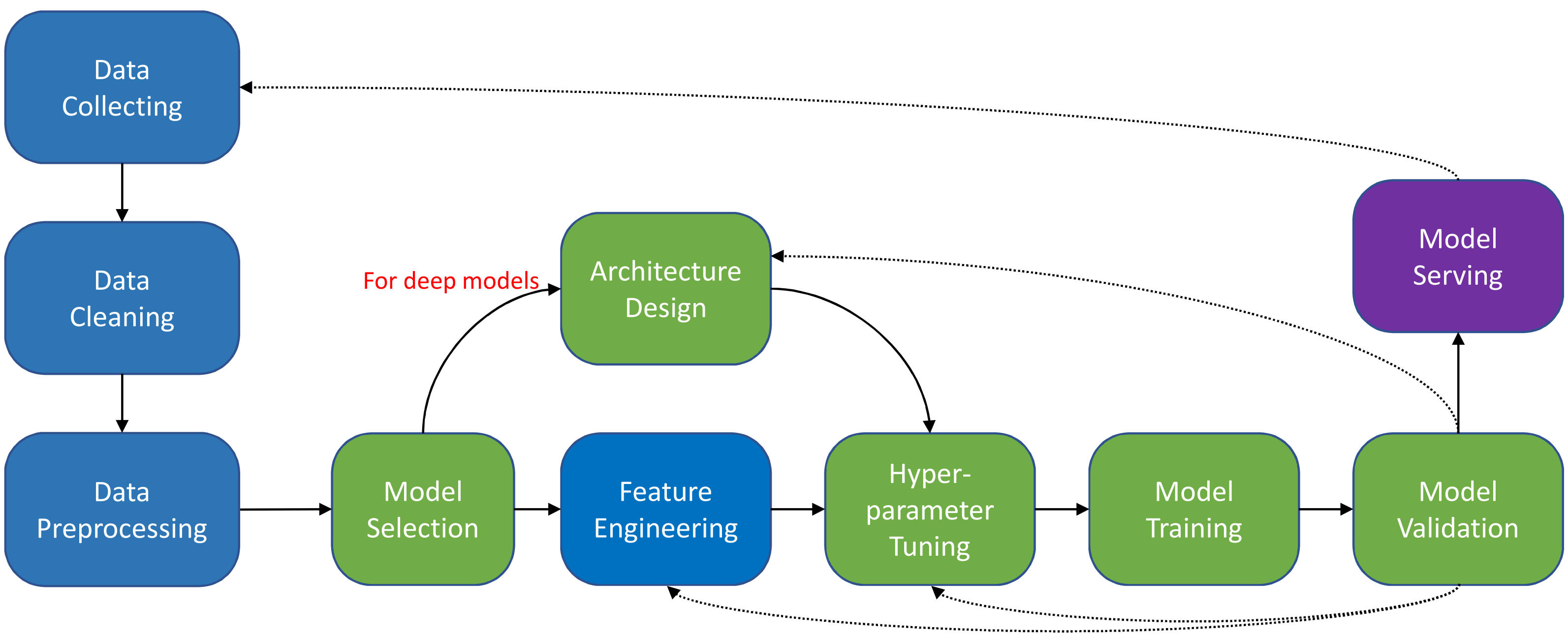
清華大學
Tsinghua University

Outline

- Machine Learning Pipeline
- Machine Learning Programing
- Hands-on Examples

Machine Learning Pipeline

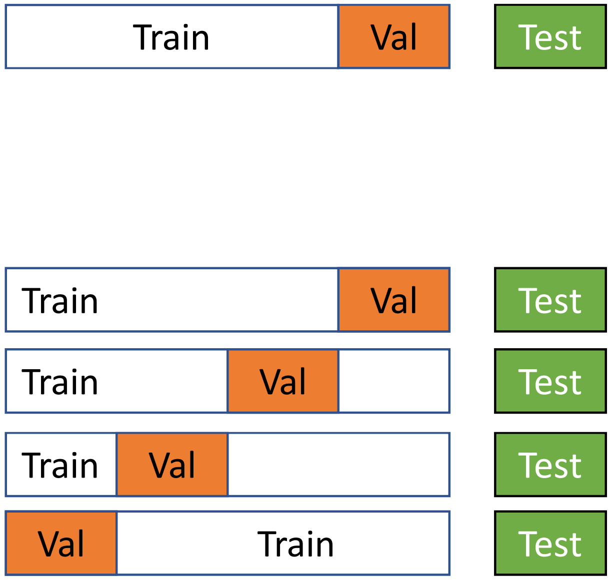
Overview



Data Collecting
Data Cleaning
Data Preprocessing
For deep models
Architecture Design
Model Selection
Feature Engineering
Hyper-parameter Tuning
Model Training
Model Validation
Model Serving

Data

- Training Set
  - Model will be fitted to this data
  - Most collected data are used
- Validation Set (a.k.a. dev set)
  - To verify Feature Engineering, Architecture Design and Hyper-parameter tuning
  - k-fold Cross Validation: 1 fold for validation, the rest for training, repeat k times with different validation folds
- Test set (holdout)
  - To verify the final result
  - Don’t tune the model towards test set

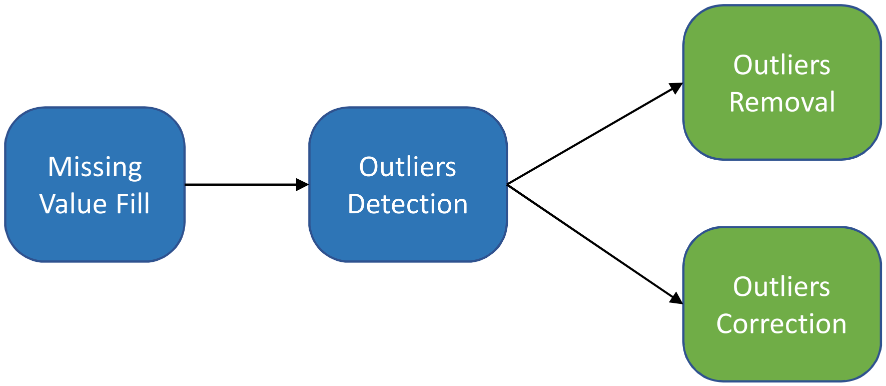


Data

- Partition ratios?
  - 0.8, 0.1, 0.1 for train, val, test
  - 0.9 for CV, 0.1 for test
- How to partition?
  - Randomly
  - Chronologically
  - …
- Retrain by all data after validation, if data is not enough

Data Cleaning

- Data is not always correct
  - Hardware issues cause data corruption or loss
  - Human labeling error
  - …

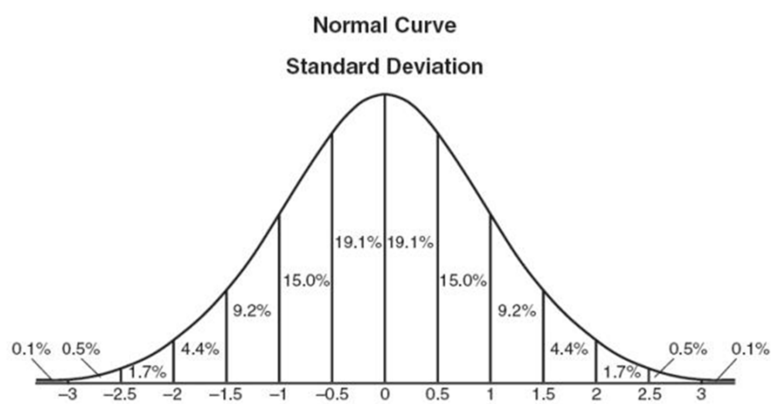


Data Cleaning – Missing Value Fill

- Constant fill
  - 0, -1, …
- Random fill based a normal or uniform distribution.
- Mean or median fill
- Missing categorial value
  - Most frequently value
  - A new category to represent missing values
- Directly remove rows/columns if too many missing values

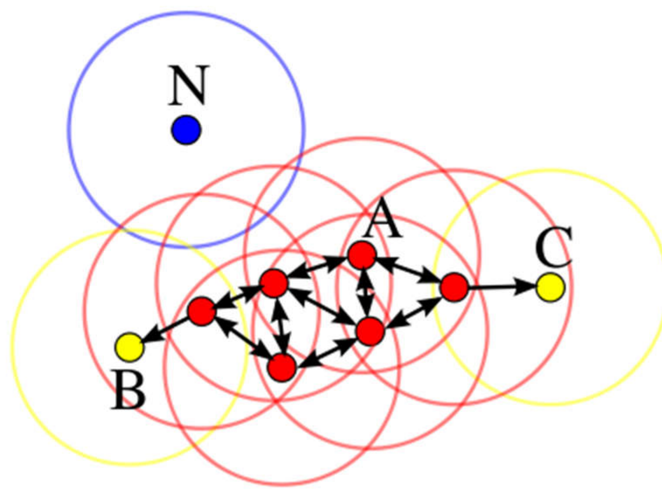
Data Cleaning – Outlier Detection

- Z-score
  - $z = \left|\frac{x-\mu}{\sigma}\right|$



Data Cleaning – Outlier Detection

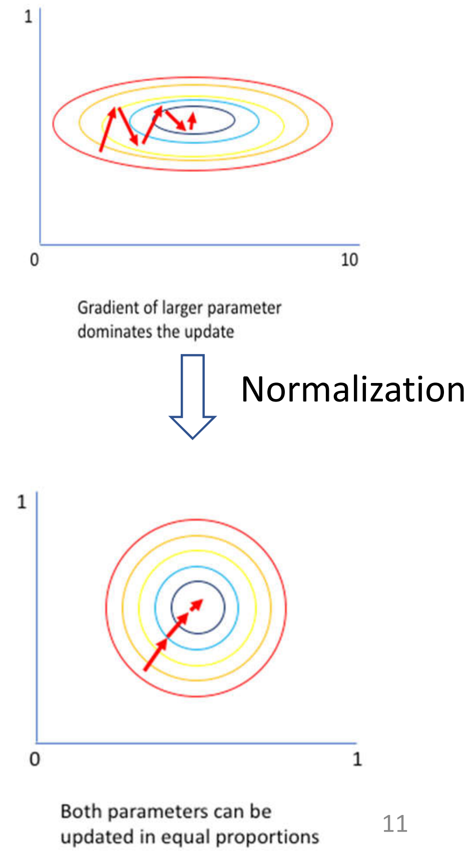
- Clustering based solution
- Refer to `sklearn.neighbors.LocalOutlierFactor`



Data Preprocessing

- Normalization / standardization
  - Often required for stochastic models, such as neural networks
  - GBDT doesn't need this
- Categorical values conversion
  - Most machine learning cannot handle categorical values directly
  - Need to convert to numerical values
    - one-hot encoding, ordinal encoding, target encoding, …

Categorical values	one hot encoding	Ordinal encoding
A	1, 0, 0	0
B	0, 1, 0	1
C	0, 0, 1	2
B	0, 1, 0	1



Data Type

	Pattern	Complete Information	Easy to Human	Permutation invariant
Image	Spatial locality	Yes	Yes	No
Sequence	Sequential dependency	Mostly	Yes (In Text & Speech)	No
Tabular	Unknown	No	No	Yes

Model Selection

- Choose an appropriate according to task/data and scenarios
- CNN
  - Image related tasks
- RNN, Transformer
  - Sequence
- GBDT
  - Adaptive and robustness
  - For all kinds of tasks with tabular data
- Linear model
  - Rapidly inference and online update

Feature Engineering

- Let model easily understand the data, leverage prior knowledge
  - Some patterns are unknown just from data
    - Data is too little to conclude these patterns
  - Experiences from domain experts
- Principle: Richer information, more is often better
- External information, such as holiday, geographic, etc
- Data analysis and visualization
- Note: Don’t leak any label information into features

Architecture Design

- In deep learning, feature engineering is not needed, but the Architecture matters
- Introduce prior into model architecture
- Appropriate model complexity
- Leverage existing architectures

Hyper-parameter Tuning

- There are many hyper-parameters needed to be tuned
  - Learning rate, number of epochs, …
- Use validation data / cross validation for tuning

Training Perf.	Validation Perf.	
Bad	Bad	under-fit, try complex settings, such as more iterations
Good	Bad	Over-fit, try simple settings
Good	Good	Good-fit

Automated ML

- Remove/Reduce human efforts in machine learning
  - Human efforts is needed in model selection, feature engineering, architecture design and hyper-parameter search
- Often need to search, most Auto-ML works aim to search an as good as possible solution within the limit time/resource
- More resource costs

Model Validation

- Measure metrics
  - Regression: MSE, MAE, …
  - Binary classification: error, logloss, auc, …
  - Classification: error, top-k accuracy, …
  - Ranking: MAP, NDCG, …
- Offline Test: Metrics over test set
  - Note: don’t tune the model according to the test set
- Online A/B test

Model Serving

- Deploy model into online production
  - Optimize for response time
  - Improve model inference speed
- Refresh/update model periodically
  - New data is generated every second in online production, and the distribution of it may change
  - Need to update the model, to ensure the real-time performance
- Online learning
  - Inference and learning simultaneously

Machine Learning Programing

Overview

- Python
  - The most widely-used program language for machine learning
- NumPy
  - Data processing, matrix manipulation
- SkLearn (scikit-learn)
  - Basic models
- XGBoost & LightGBM
  - For all kinds of tabular data
- Tensorflow & Pytorch
  - For image, text and speech

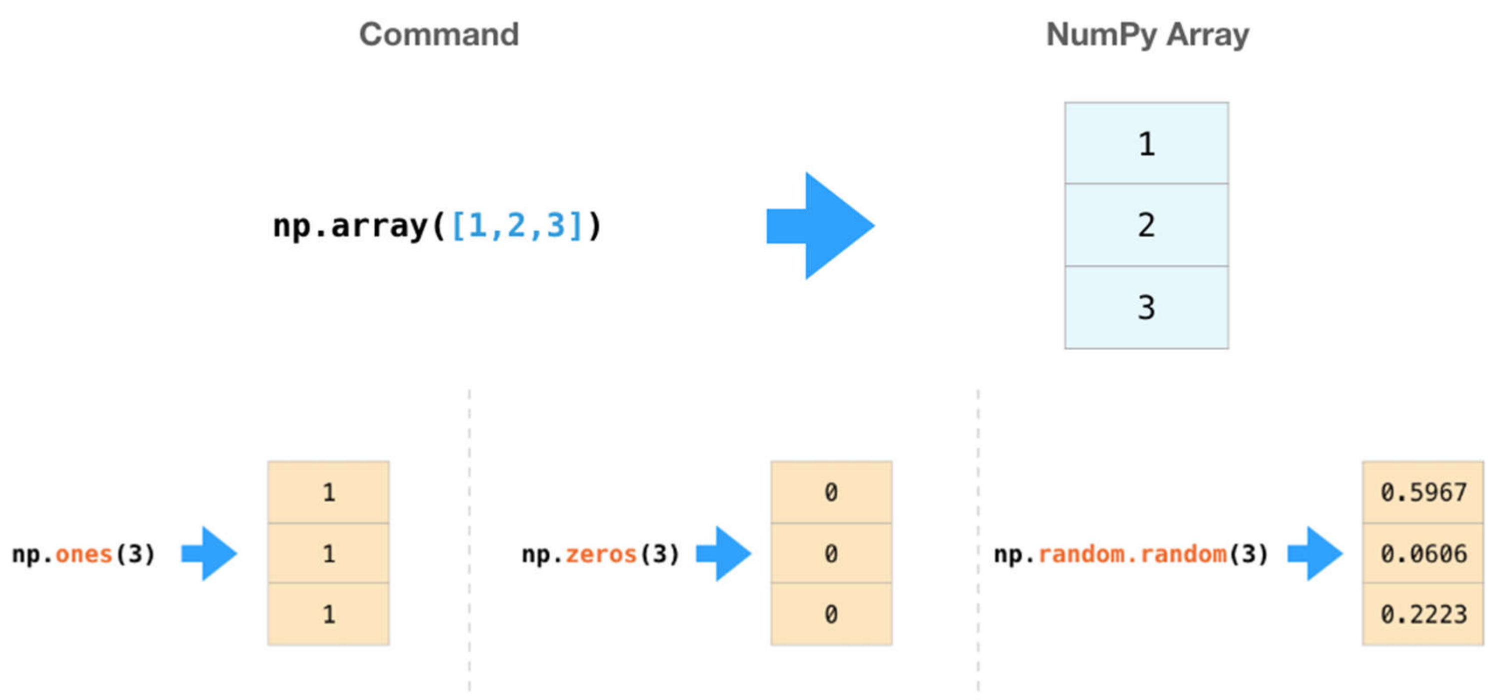
Python

- Brief introduction
  - http://www.voidspace.org.uk/python/articles/python_datatypes.shtml
  - https://scipy-lectures.org/
- Package Management: pip, conda
- Use python 3
- Virtual environments
  - Different python/package versions
  - Multi-user servers

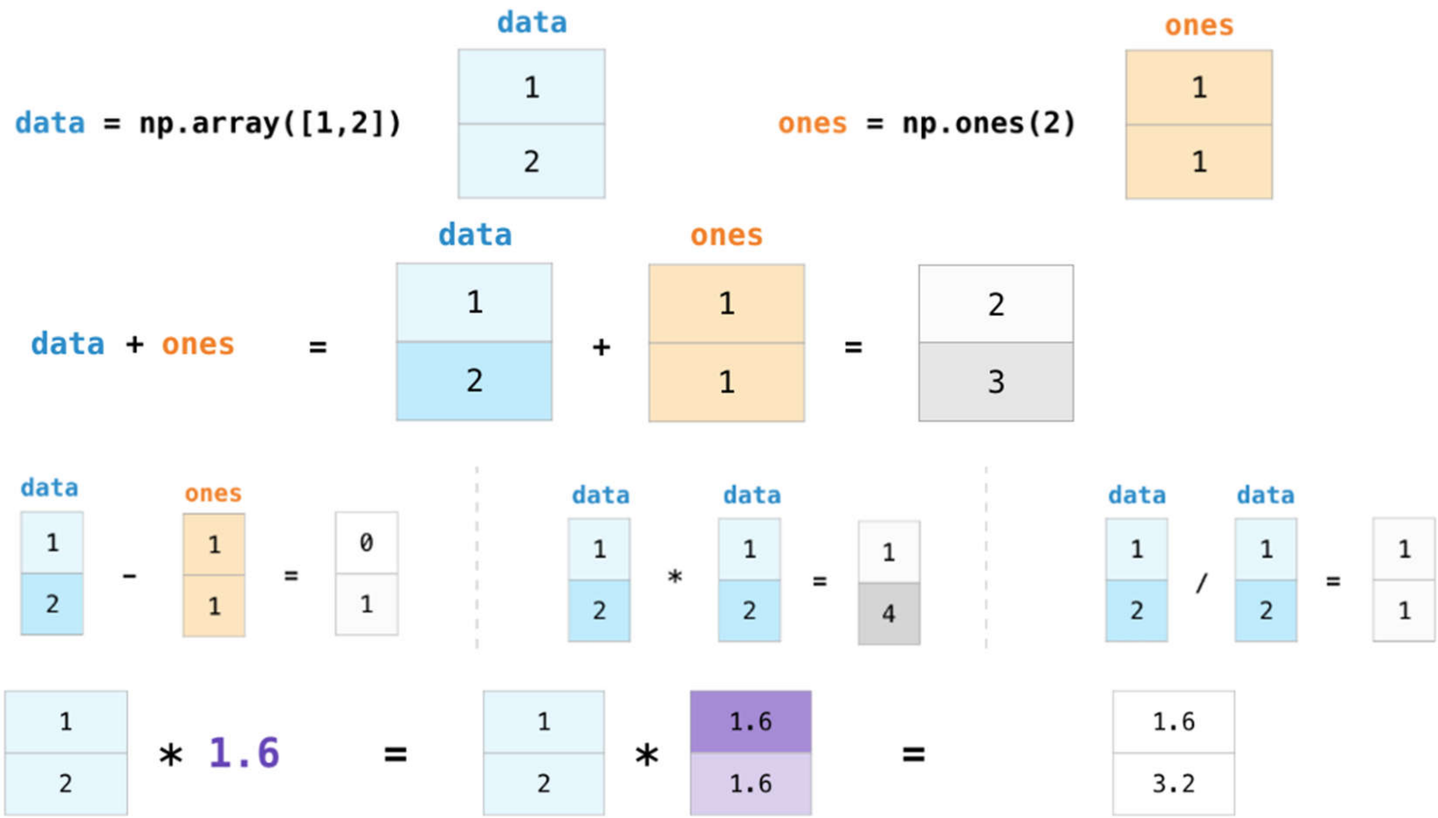
NumPy

- NumPy is the fundamental package for scientific computing with Python.
  - a powerful N-dimensional array object
  - sophisticated (broadcasting) functions
- Tutorial
  - https://www.numpy.org/devdocs/user/quickstart.html
- Many operators in Tensorflow and PyTorch are from NumPy

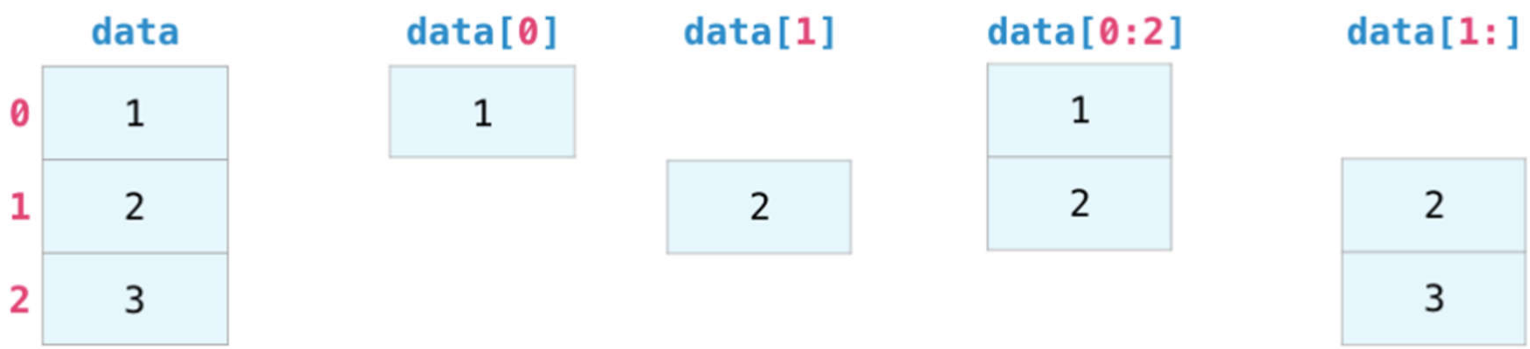
NumPy: Creating Arrays



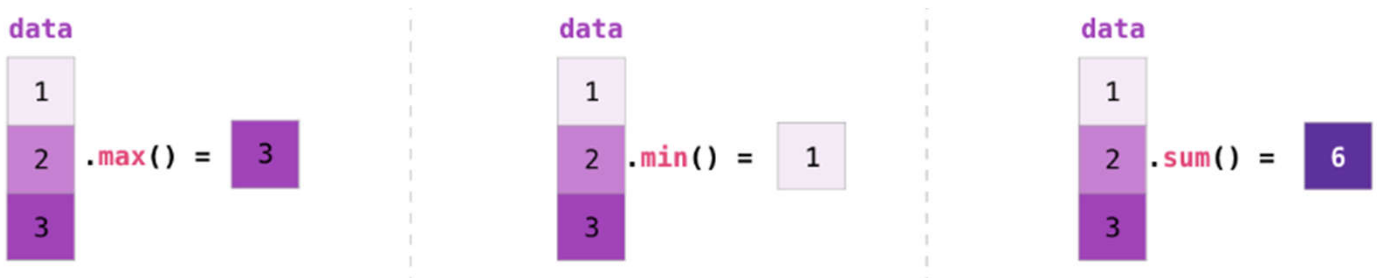
NumPy: Arithmetic



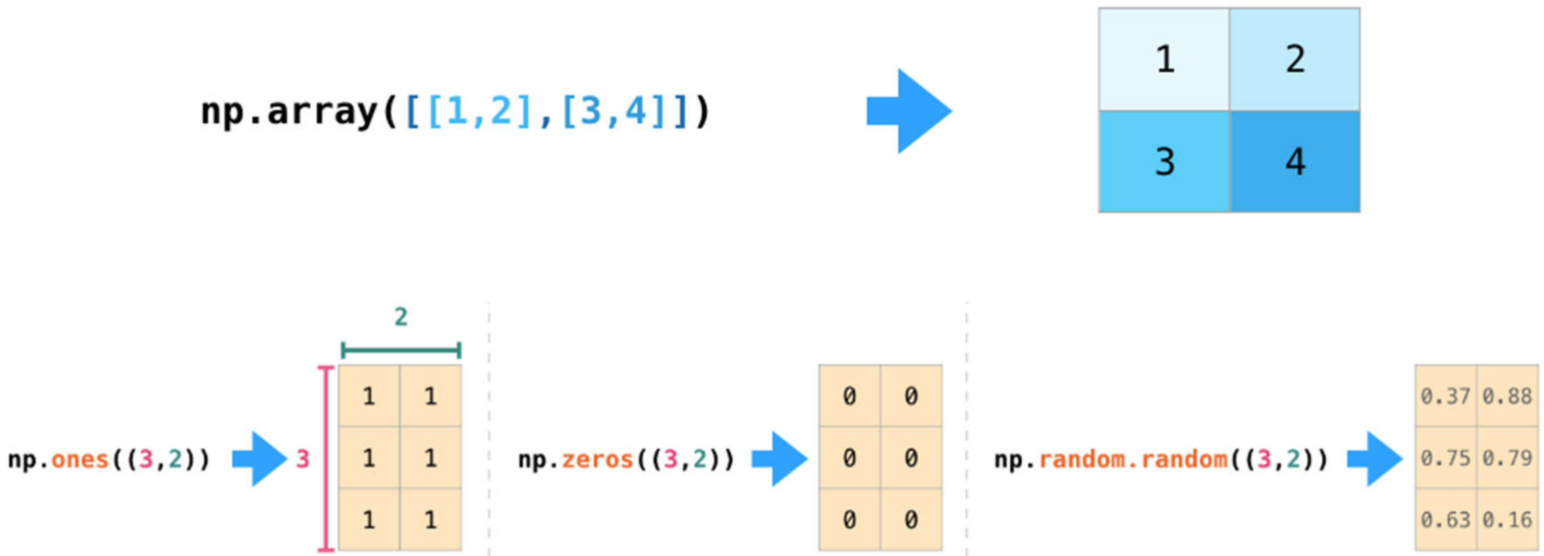
NumPy: Indexing



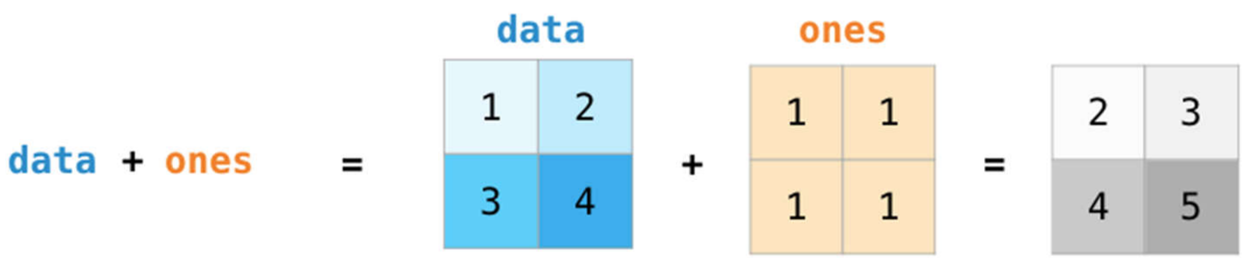
NumPy: Aggregation

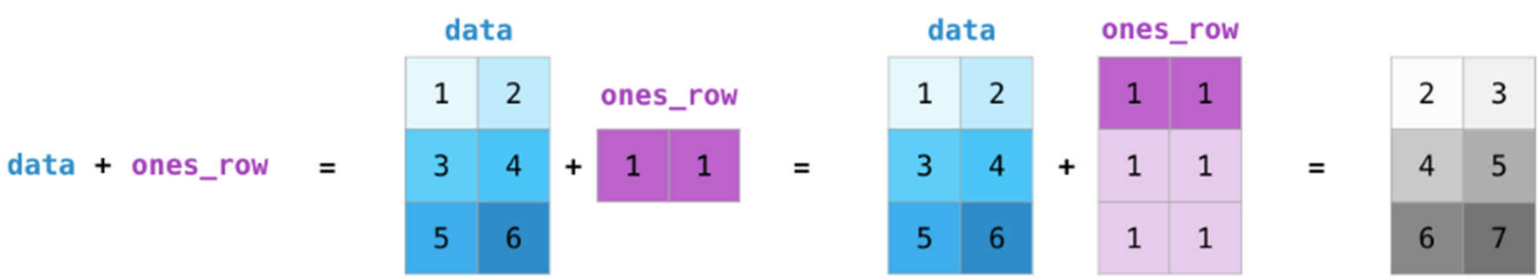


NumPy: Creating Matrices



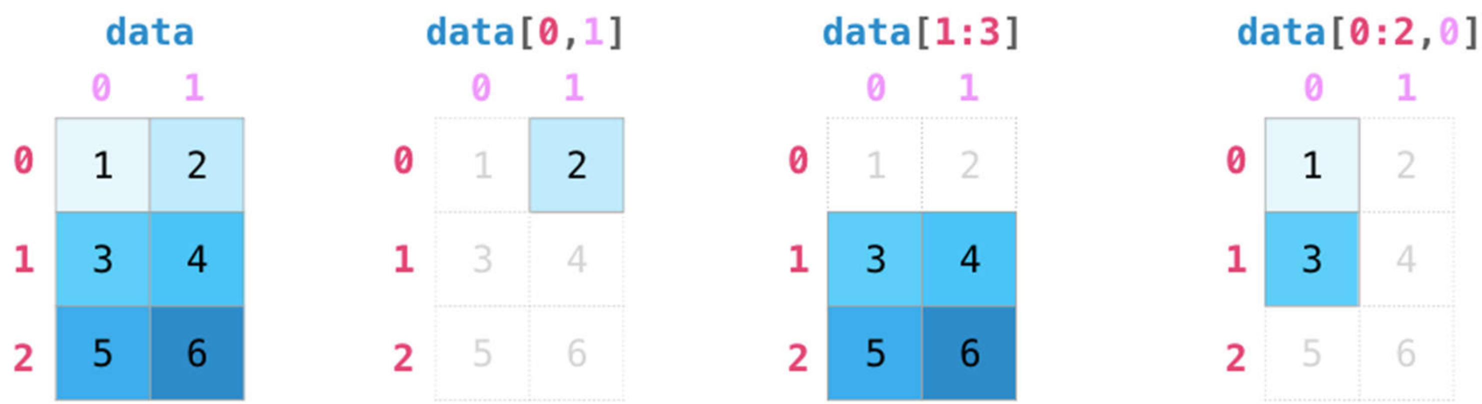
NumPy: Matrix Arithmetic



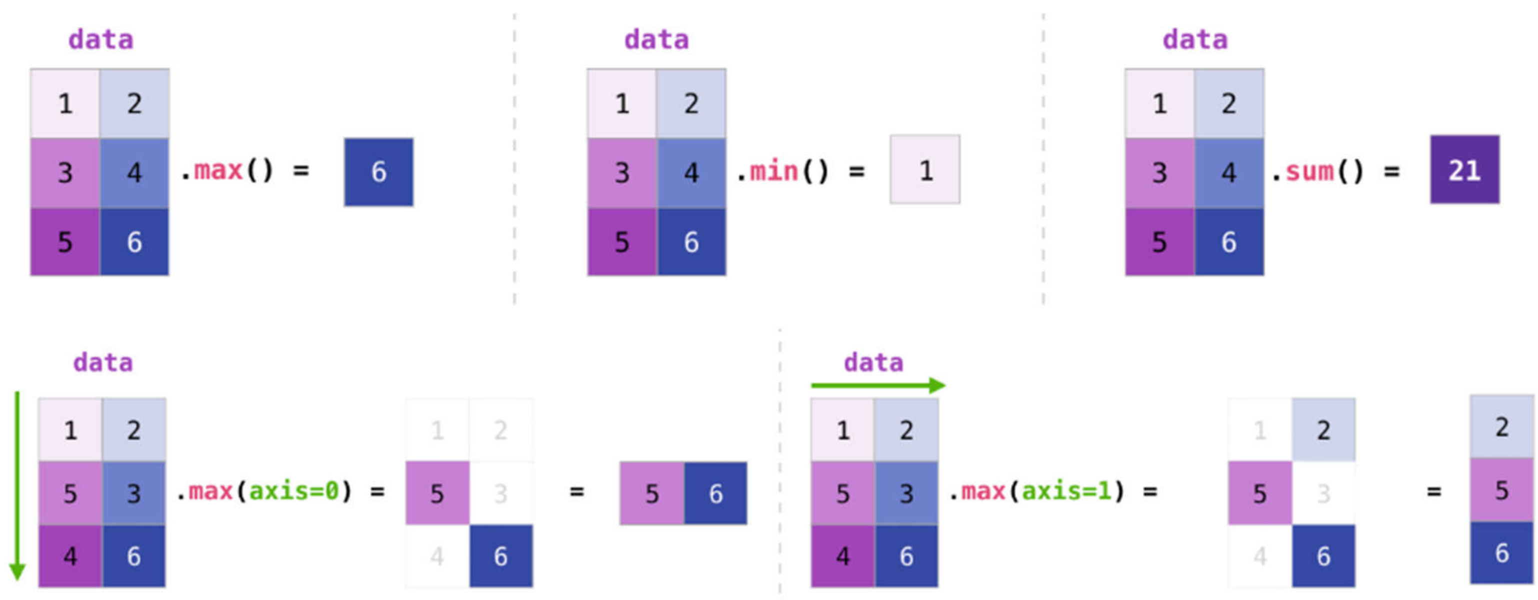


NumPy: Dot Product

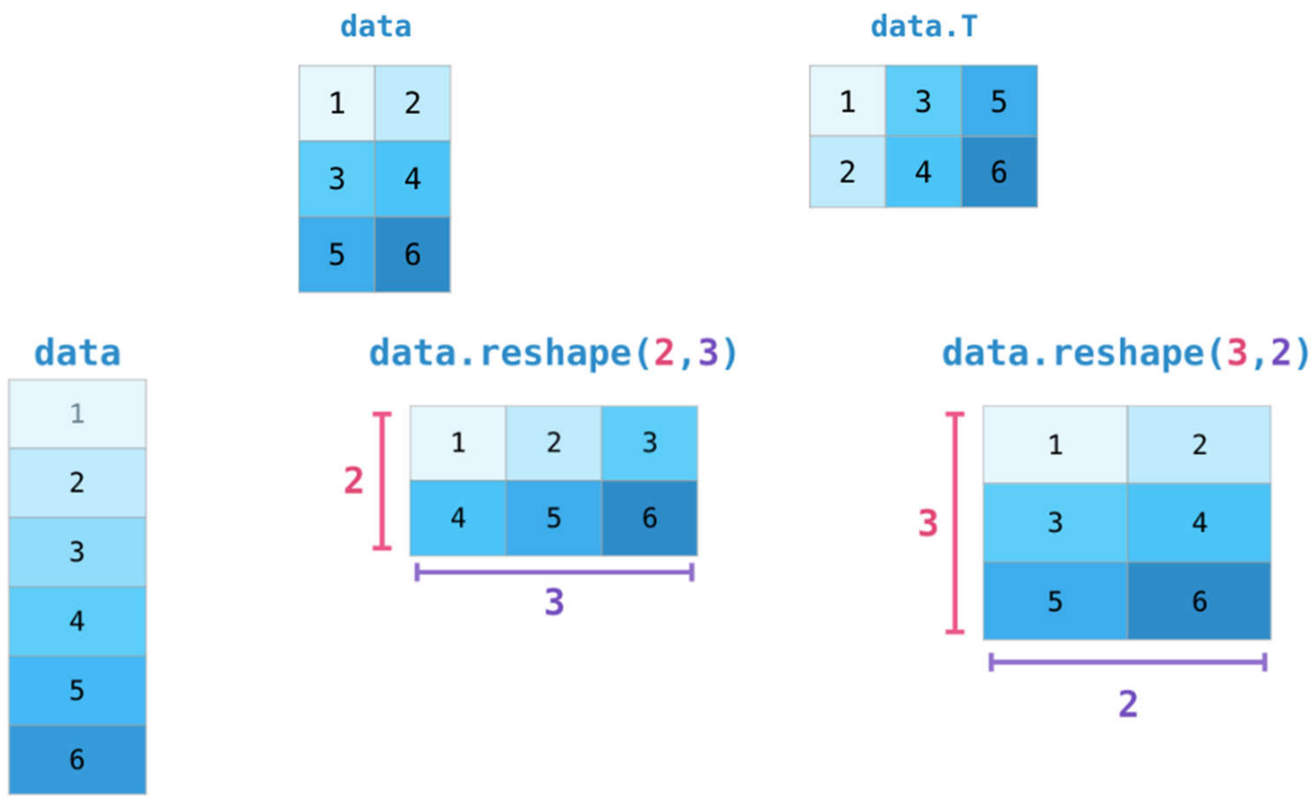
NumPy: Matrix Indexing



NumPy: Matrix Aggregation



NumPy: Matrix Shape Manipulation



SkLearn

- Basic Machine Learning library in Python
- Process:
  - 1. get the data
  - 2. define the model
  - 3. fit: train the model by data
  - 4. predict: use the fitted model to predict over the new data

In [1]:
```python
import numpy as np
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
iris = load_iris() # Get the dataset
X, y = iris.data, iris.target
X.shape, y.shape
```

Out[1]: ((150, 4), (150,))

In [2]:
```python
X[0], y
```

Out[2]:
```
(array([5.1, 3.5, 1.4, 0.2]),
 array([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, 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, 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, 2, 2, 2]))
```

In [3]:
```python
model = KNeighborsClassifier(n_neighbors=15)
model
```

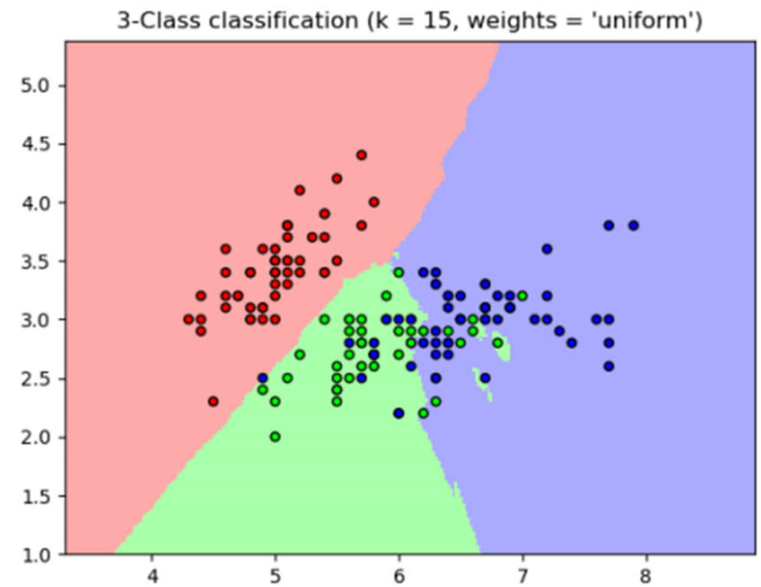
Out[3]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=15, p=2,
           weights='uniform')
```

In [4]:
```python
model.fit(X, y) # fit the data
```

Out[4]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=15, p=2,
           weights='uniform')
```

In [5]:
```python
# prediction
pred = model.predict(iris.data)
print("error rate:", np.sum(pred != y) / len(y))
```

```
error rate: 0.013333333333333334
```



XGBOOST & LightGBM

- Similar to sklearn
- Data preprocessing and feature engineering matters

In [1]:

```python
import lightgbm as lgb
from sklearn.datasets import load_boston
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
```

In [2]:

```python
boston = load_boston()
X_train, X_test, y_train, y_test = train_test_split(boston.data, boston.target, test_size=0.1, random_state=42)
```

In [3]:

```python
# define the model
gbm = lgb.LGBMRegressor(num_leaves=31,
                        learning_rate=0.1,
                        n_estimators=5)
```

In [4]:

```python
# start training
gbm.fit(X_train, y_train,
        eval_set=[(X_test, y_test)],
        eval_metric='l1',
        early_stopping_rounds=5)
```

```
[1]	valid_0's l2: 54.8782	valid_0's l1: 5.28594
Training until validation scores don't improve for 5 rounds.
[2]	valid_0's l2: 46.7178	valid_0's l1: 4.85663
[3]	valid_0's l2: 40.0558	valid_0's l1: 4.52401
[4]	valid_0's l2: 34.8406	valid_0's l1: 4.24429
[5]	valid_0's l2: 30.6244	valid_0's l1: 3.99647
Did not meet early stopping. Best iteration is:
[5]	valid_0's l2: 30.6244	valid_0's l1: 3.99647
```

Out[4]:

```
LGBMRegressor(boosting_type='gbdt', class_weight=None, colsample_bytree=1.0,
       importance_type='split', learning_rate=0.1, max_depth=-1,
       min_child_samples=20, min_child_weight=0.001, min_split_gain=0.0,
       n_estimators=5, n_jobs=-1, num_leaves=31, objective=None,
       random_state=None, reg_alpha=0.0, reg_lambda=0.0, silent=True,
       subsample=1.0, subsample_for_bin=200000, subsample_freq=0)
```

In [5]:

```python
#start prediction
y_pred = gbm.predict(X_test)
print('The rmse of prediction is:', mean_squared_error(y_test, y_pred) ** 0.5)
```

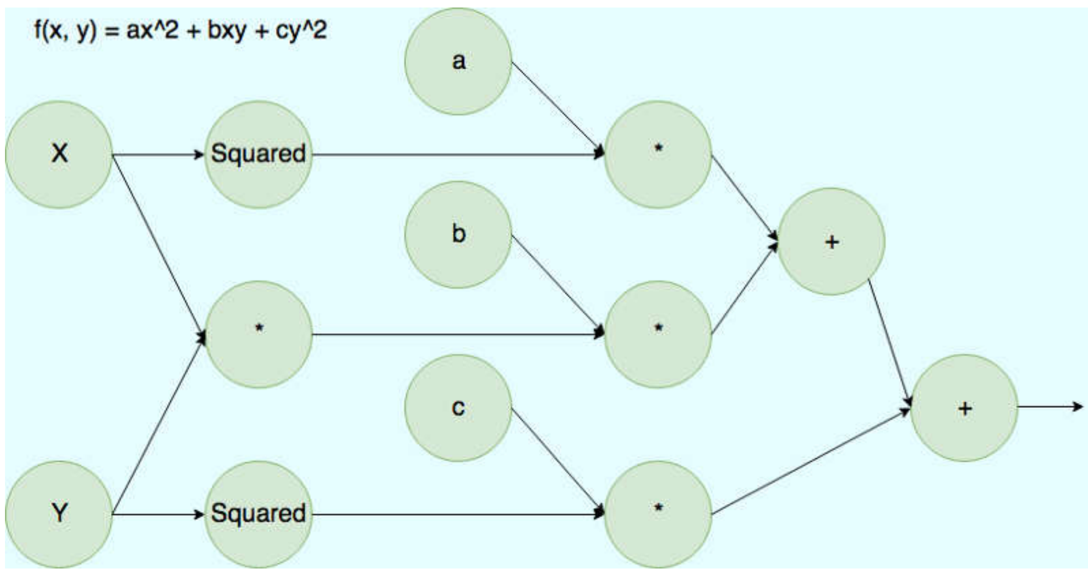
```
The rmse of prediction is: 5.533928264286112
```

Deep Learning Toolkits

- Unlike traditional ML models, NN is more like the building blocks, you need to build the model by yourself
  - Different NN models essentially are different models
- DNN toolkits contains rich basic blocks, and you can use them to build your own models
- Therefore, compared with sklearn, DNN toolkits are not so straight-forward to use

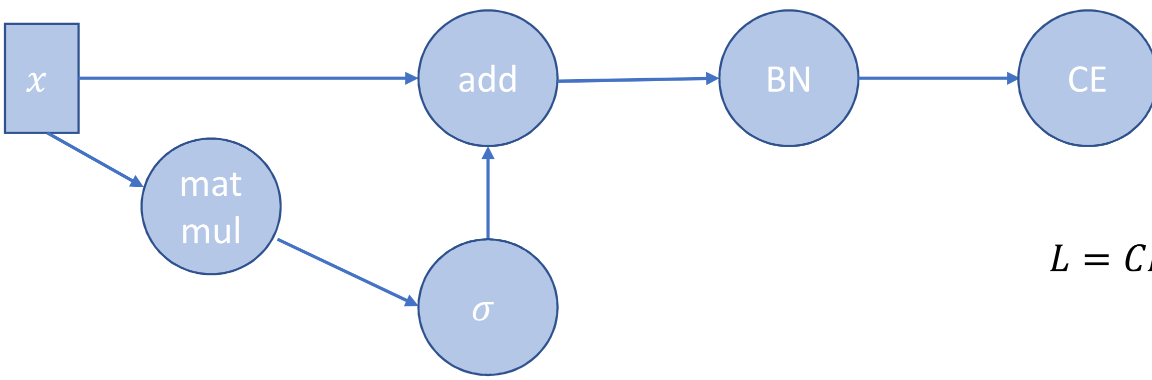
Computational Graph (CG)

- CG: represent a math function using the language of graph theory



Code DNN using Computational Graph

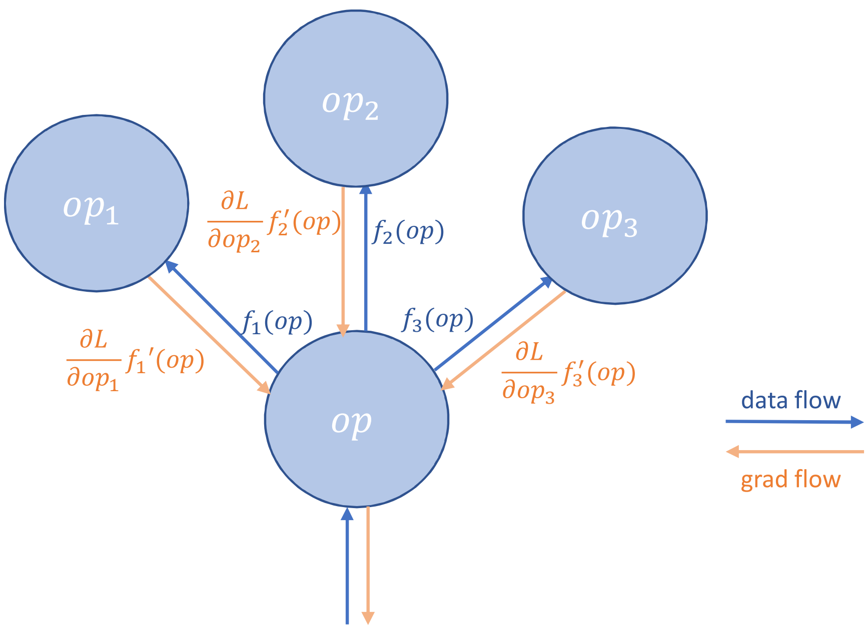
- DNN is a very complicated function
  - Represent it as a Directed Acyclic Graph
  - Node: operator
  - Edge: data flow

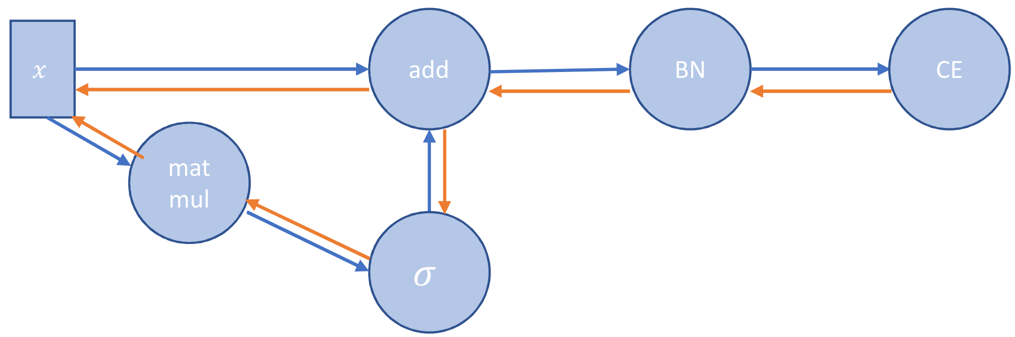


$$L = CE(BN(x + \sigma(W * x)))$$

Why CG: Facilitate Automatic Differentiation

- Chaining the gradient backward in the topological order of the graph nodes





Deep Learning Toolkits

- TensorFlow
- PyTorch

TensorFlow: Static CG

- Static CG: build the CG at first, the re-use it for several times
  - Define **and** Run
- Example:

```
#Feed
#创建占位符
input1 = tf.placeholder(tf.float32)
input2 = tf.placeholder(tf.float32)
#使用placeholder定义op
output = tf.multiply(input1, input2)

with tf.Session() as sess:
    #feed数据以字典的方式传入
    print(sess.run(output, feed_dict={input1: [7.], input2: [2.]}))
```

输出结果:

```
[ 14.]
```

PyTorch: Dynamic CG

- Dynamic CG: build the graph during runtime
  - Define **by** run
- Example:

```
for t in range(500):
    # Forward pass: compute predicted y
    h = x.dot(w1)
    h_relu = np.maximum(h, 0)
    y_pred = h_relu.dot(w2)

    # Compute and print loss
    loss = np.square(y_pred - y).sum()
    print(t, loss)

    # Backprop to compute gradients of w1 and w2 with respect to loss
    grad_y_pred = 2.0 * (y_pred - y)
    grad_w2 = h_relu.T.dot(grad_y_pred)
    grad_h_relu = grad_y_pred.dot(w2.T)
    grad_h = grad_h_relu.copy()
    grad_h[h < 0] = 0
    grad_w1 = x.T.dot(grad_h)

    # Update weights
    w1 -= learning_rate * grad_w1
    w2 -= learning_rate * grad_w2
```

Comparison of Static GC and Dynamic GC

	Static GC	Dynamic GC
Modify graph at runtime	Hard	Easy
Varying length inputs handle	Hard	Easy
Difficulty to code	Hard	Easy
Performance optimization	High	Low

Hands-on Examples

Contents

- Sklearn
  - House price
    - https://github.com/ageron/handson-ml/blob/master/02_end_to_end_machine_learning_project.ipynb
- LightGBM
  - Click Prediction
    - https://nbviewer.jupyter.org/github/microsoft/recommenders/blob/444e6c4546f13203e1390e06ba9f9fc95081e29e/notebooks/00_quick_start/lightgbm_tinycriteo.ipynb
- PyTorch
  - Image classification
    - https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html