高等机器学习

机器学习流程

张超（图信所）
清华大学 电子工程系

修改自MSRA施宇研究员课件

Microsoft

清華大學
Tsinghua University

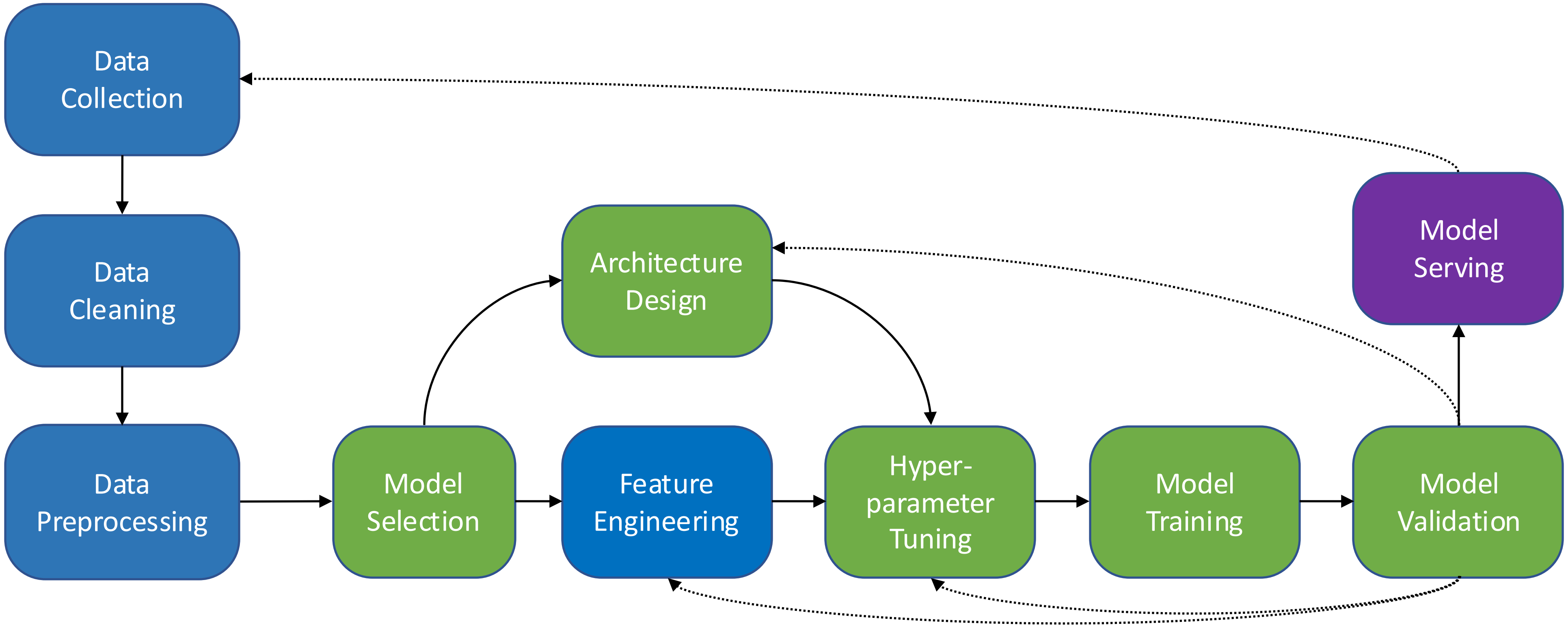
Outline

- Machine Learning Pipeline
    - Data processing procedure
    - Tricks for improved performance

- Machine Learning Programing
    - NumPy & operations
    - CUDA & Computational graph
    - TensorFlow vs PyTorch

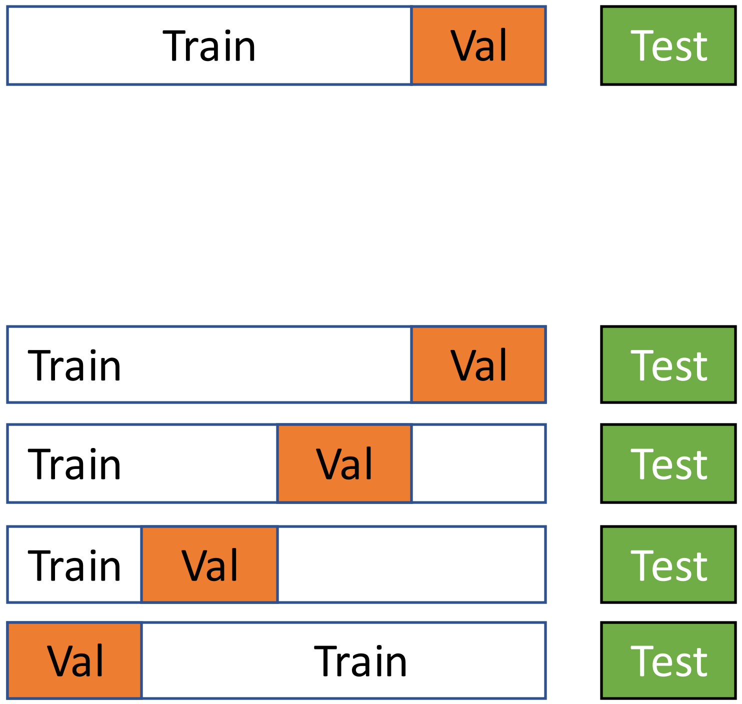
Machine Learning Pipeline

Overview


Data Collection
Data Cleaning
Data Preprocessing
Model Selection
Architecture Design
Feature Engineering
Hyper-parameter Tuning
Model Training
Model Validation
Model Serving

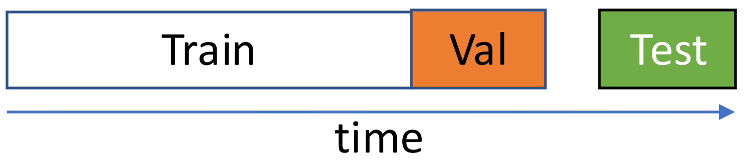
Data

- Training Set
  - Model will be fitted to this data
  - Most collected data are used
- (Held-out) Validation Set (dev set)
  - To verify Feature Engineering, Architecture Design and Hyper-parameter tuning
  - k-fold Cross Validation: one for validation, the others for training, repeat k times with different folds
- Test set (eval set)
  - To verify final model performance
  - Don’t tune the model based on test set



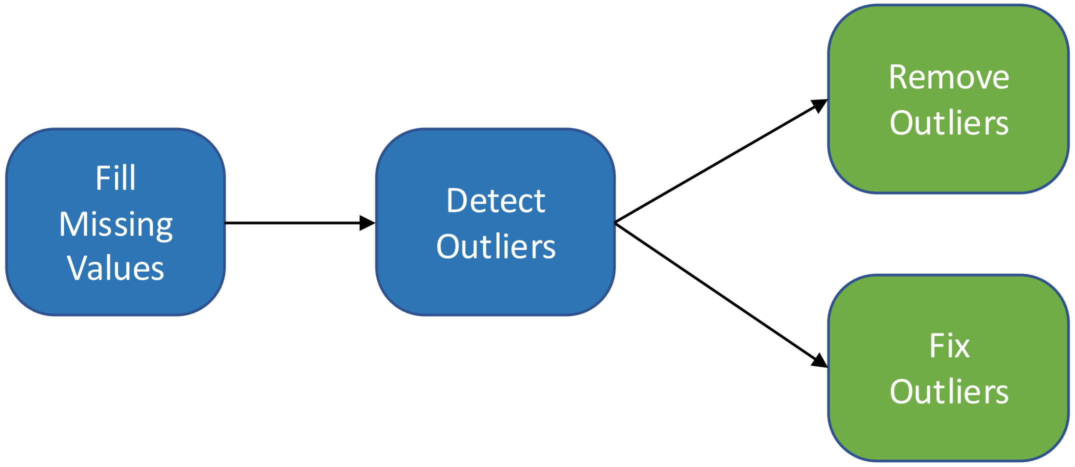
Data

- Partition ratios?
  - 0.8, 0.1, 0.1 for train, validation, test
  - 5-fold / 10-fold cross-validation
- How to partition?
  - Randomly
  - Chronologically (e.g., stock price, user behaviors etc.)
  - Other factors (e.g., speaker/session etc.)
  - …
- Retrain with all data (incl. validation set)?
  - Benefits
  - Risks



Data Cleaning

- Data is not always correct
  - Missing values
  - Data corruption or lost
  - Human labeling error
  - …

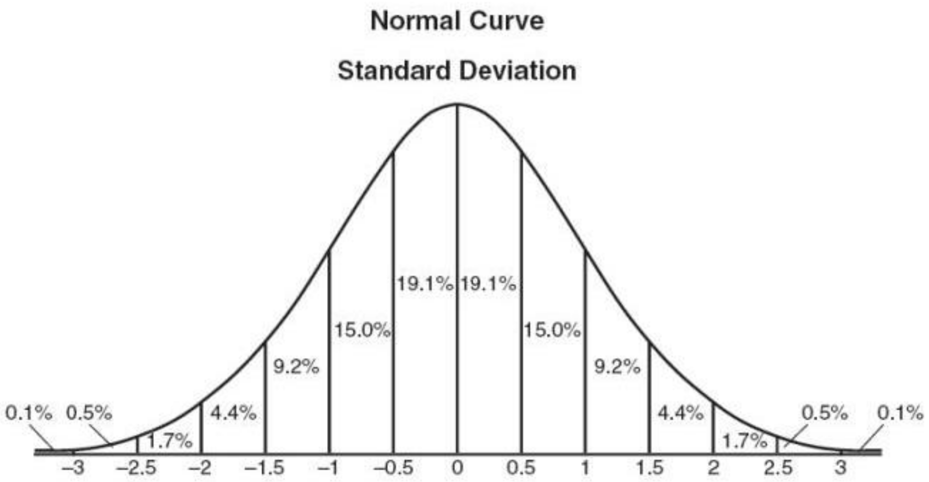


Data Cleaning – Fill Missing Values

- Feature types
  - Numerical features: e.g., age, height
  - Categorical features: e.g., gender, occupation
- Missing numerical values
  - Constant fill: e.g., 0, …
  - Random fill based on a normal or uniform distribution.
  - Mean or median fill
- Missing categorial value
  - Most frequent value
  - A new category to represent missing values
- Remove the sample if too many missing values

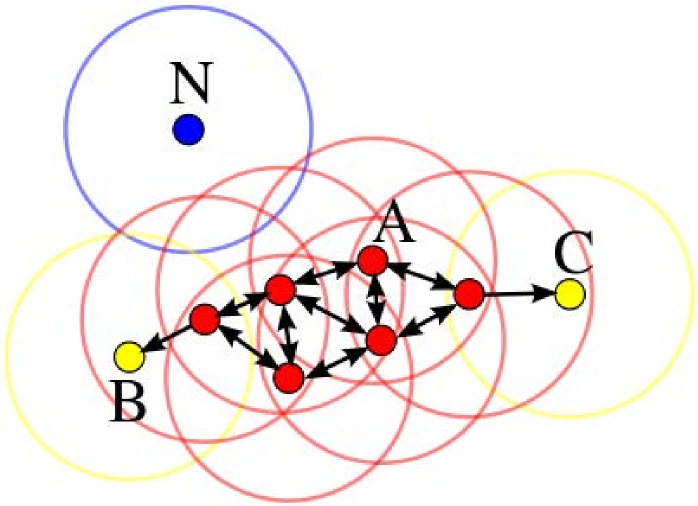
Data Cleaning – Detect Outlier

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



Data Cleaning – Detect Outlier

- Based on sample density
- Refer to `sklearn.neighbors.LocalOutlierFactor`

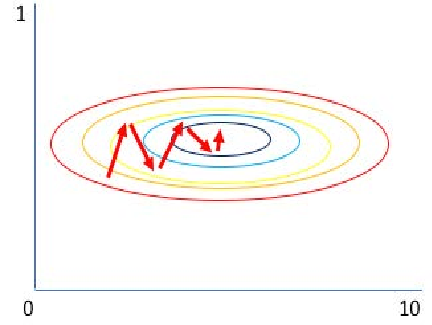


- Use a pre-trained model to remove samples with very high error rates.

Data Preprocessing

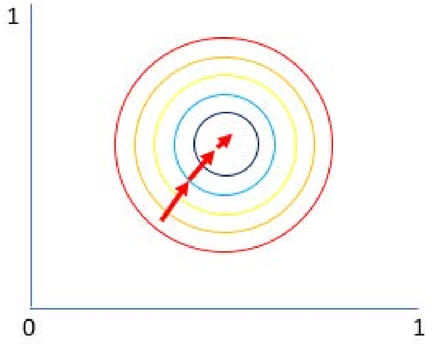
- Normalization / standardization
  - Scale different features to the same range: $\frac{x-\mu}{\sigma}$, $\frac{x-x_{\min}}{x_{\max}-x_{\min}}$
  - Often required for neural networks, trained by gradient descent
- Categorical values conversion
  - Some machine learning algorithms cannot handle categorical values
  - Convert to numerical values
    - one-hot encoding, pre-trained model (extracted) embeddings

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



Gradient of larger parameter dominates the update

Normalization



Both parameters can be updated in equal proportions

Data Type

- Image
- Low-frequency sequence
  - Text
  - Video
- High-frequency sequence
  - Speech
  - EEG/MEG etc.
  - Stock price
- Graphs
- Other vectors

Model Selection

- Choose an appropriate model according to task/data and scenarios
- CNN, Transformer [1]
  - Image related tasks
- RNN, Transformer
  - Sequence
- GNN, Transformer [2]
  - Graph inputs
- Linear model
- Others
  - Decision trees, k-nearest-neighbors etc.

[1] A. Dosovitskiy et al., “An image is worth 16x16 words: Transformers for image recognition at scale”, in Proc. ICLR, 2021.

[2] C. Ying et al., “Do Transformers really perform badly for graph representation?”, in Proc. NeurIPS, 2021.

Feature Engineering

- Use prior knowledge
  - Some patterns are in the training set
  - Experience provided by domain experts
- Principle: More information is often better
  - More relevant attributes: e.g. holiday & geographic etc.
  - More complete inputs: longer text sequence, complete speech waveform
  - Note: Don’t leak any label information into features
- Data analysis and visualization

Architecture Design

- Deep neural network: A classifier/linear regression with a jointly learned deep feature transform

- In deep learning, feature engineering is less of a problem, but the architecture matters
    - Consider prior knowledge in architecture design
    - To have proper computation & storage complexities
    - Use existing architectures

Hyper-parameter Tuning

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

Training Performance	Validation Performance	Solutions
Bad	Bad	Under-fitting, less regularization, bigger model, more epochs
Good	Bad	Over-fitting, more regularization, smaller model, early stopping
Good	Good	

Auto-ML

- Reduce human efforts in parameter/architecture search
  - In model selection, feature engineering, architecture design and hyper-parameter search
- Search is needed to find a good solution
  - Evaluate the search results
  - Meet the resource constraint
- More efficient than brutal force grid search

Model Validation

- Popular Metrics
  - Regression: (root) mean square error, mean absolute error, correlation scores etc.
  - Classification: error rate/accuracy, top-k
  - Event detection: precision, recall, F1, area under curve
  - Ranking: mean average precision, normalized discounted cumulative gain
  - Uncertainty: entropy, KL-divergence, likelihood
  - Speech & language:
    - Objective: Word error rate (ins., del. & sub.), diarization error rate
    - Subjective: MOS, A/B test, Fluency, Relevance, Coherence, Naturalness, Grammaticality…
    - Approximated: BLEU, PESQ, STOI, Inception score etc.
    - Dataset-based general scores: GLEU, Superb etc.
  - Metrics are often used as training loss functions
- Note again: don't tune the model according to the test set

Model Serving

- Deploy model to online production
  - Optimize for response time/latency
  - Improve test-time cost
- Refresh/update model periodically – A recommender system
  - Use new data from online applications (may have a distribution shift)
  - Does the new model improve?
- 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)
  - SVM, k-NN, k-Means, spectral clustering, PCA…
- Tensorflow & Pytorch (& JAX)
  - Image, text and speech synthesis
  - Speech recognition: NeMo, AxLearn, ESPnet, SpeechBrain, K2, TorchAudio/Fairseq, Lingvo
- LLMs & Copilot
  - For everything

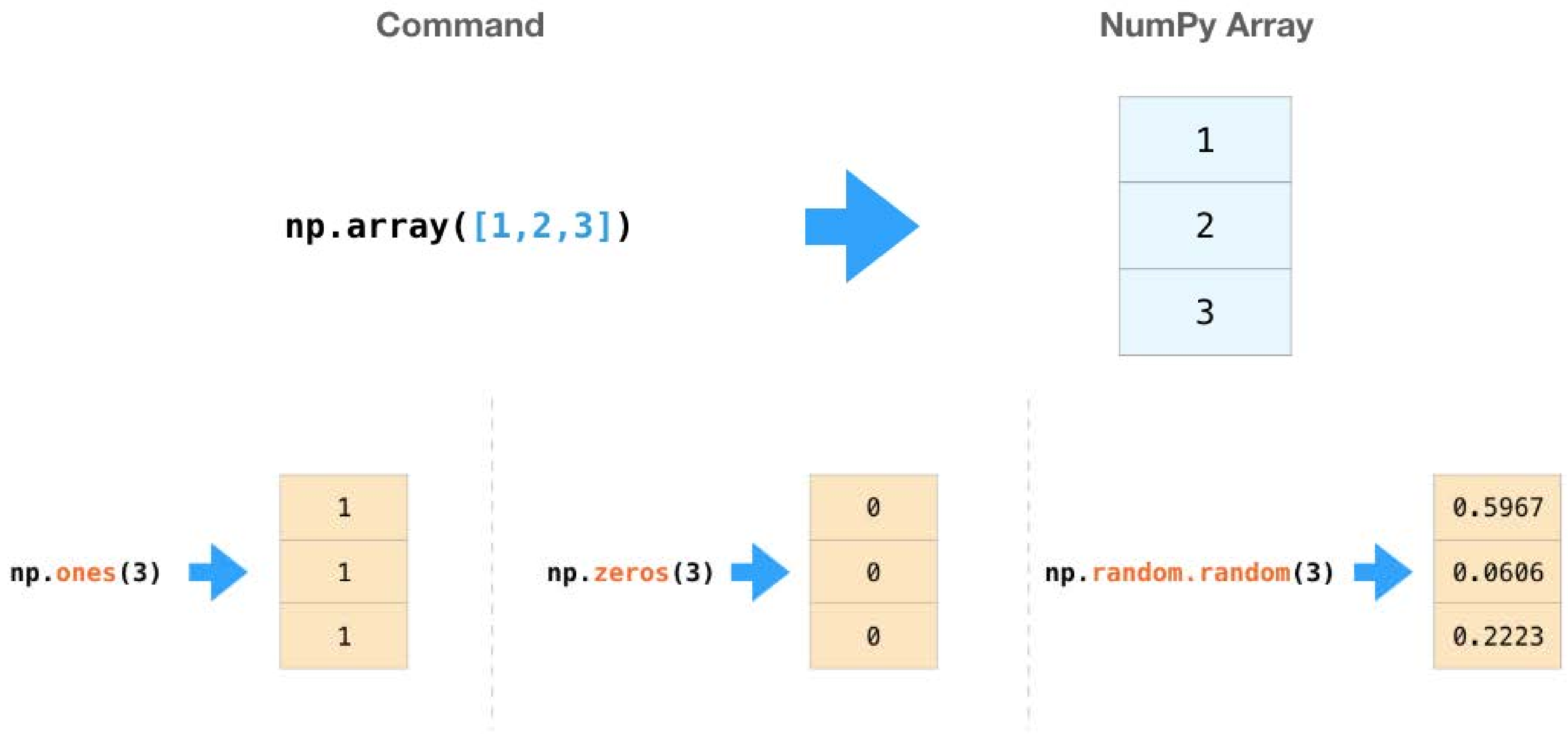
Python

- Brief introduction
  - https://www.w3schools.com/python/python_datatypes.asp
  - https://scipy-lectures.org/
- Package management: pip, conda
- Virtual environments: with conda (Anaconda / Miniconda)
  - Different python/package versions
  - Multi-user servers

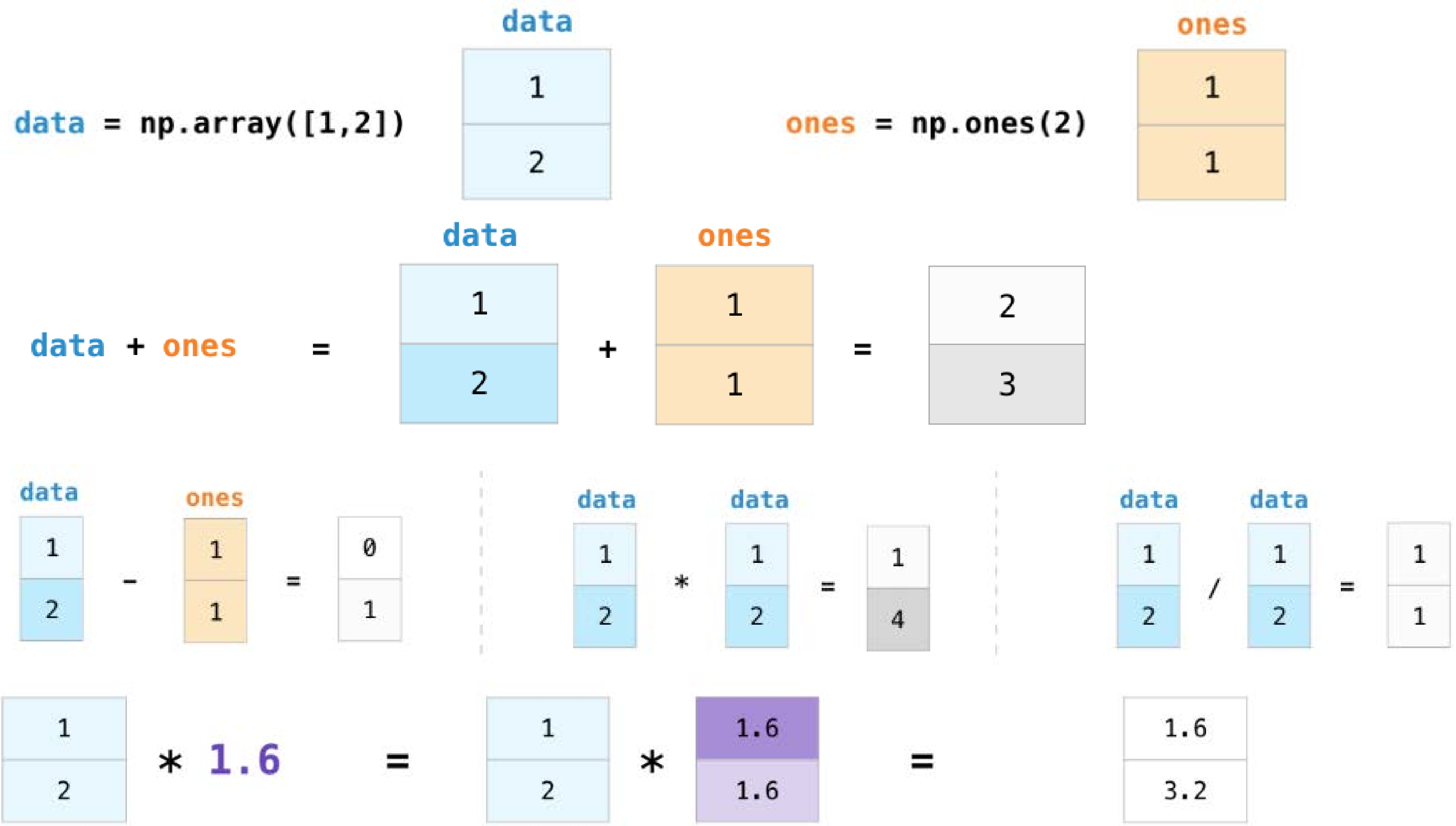
NumPy

- NumPy is the fundamental package for scientific computing with Python.
  - Powerful N-dimensional arrays (vector → matrix → tensor)
  - Include complex array-based operations
  - Optimized core C code
  - CPU only
- Tutorial
  - https://www.numpy.org/devdocs/user/quickstart.html
- Many operators in Tensorflow and PyTorch are from NumPy

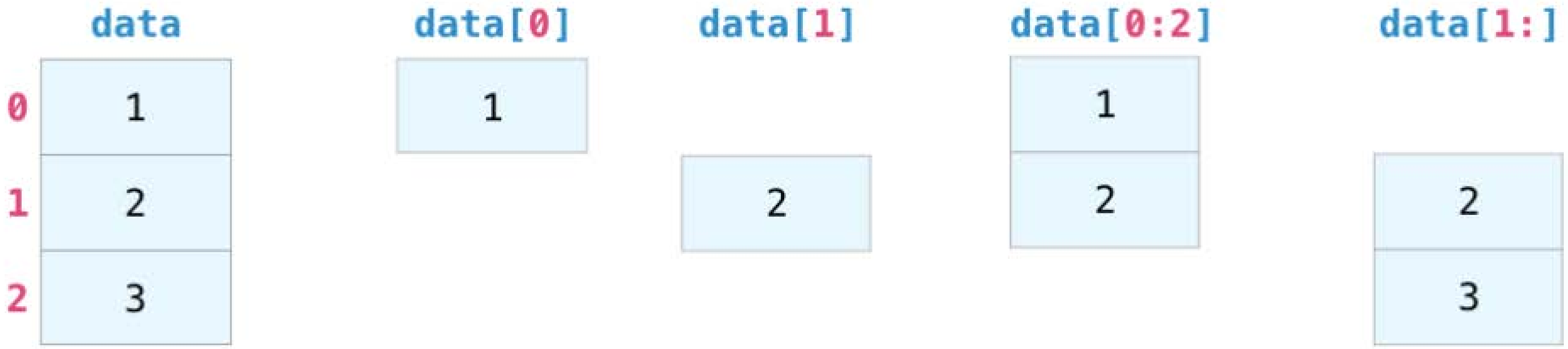
NumPy: Creating Arrays



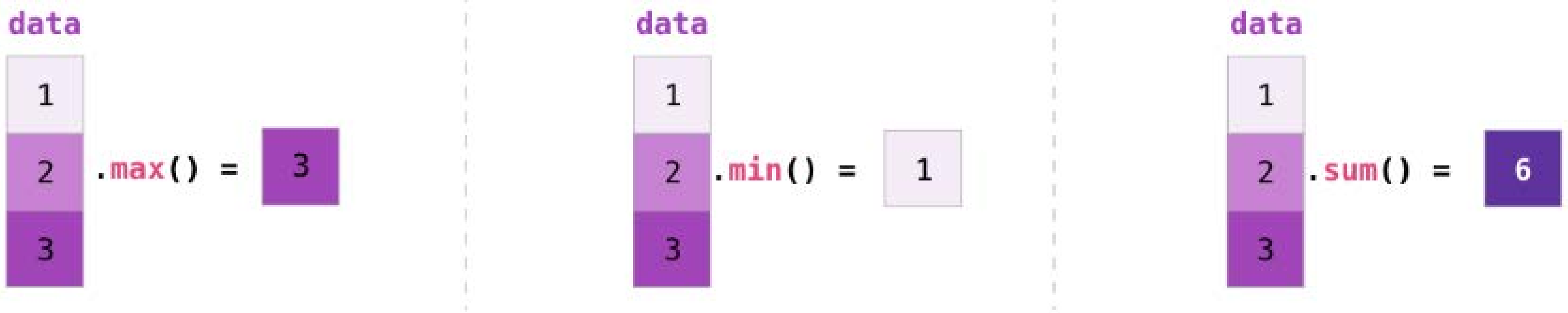
NumPy: Arithmetic



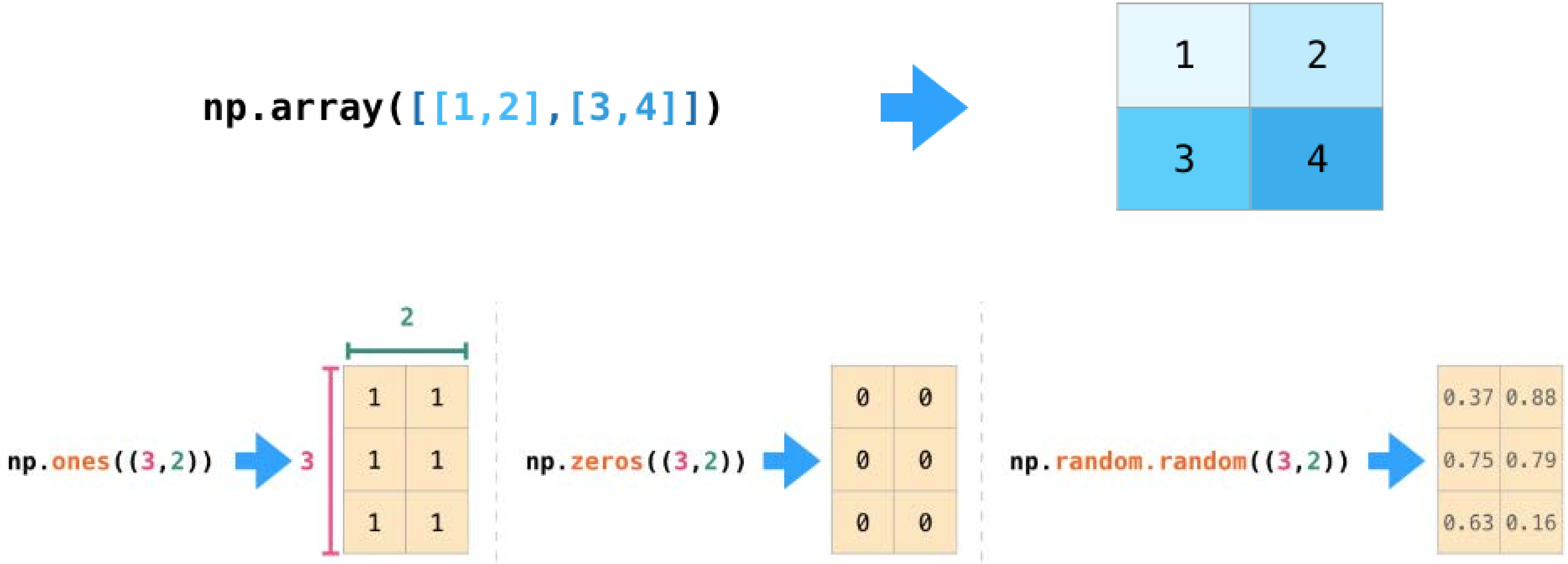
NumPy: Indexing



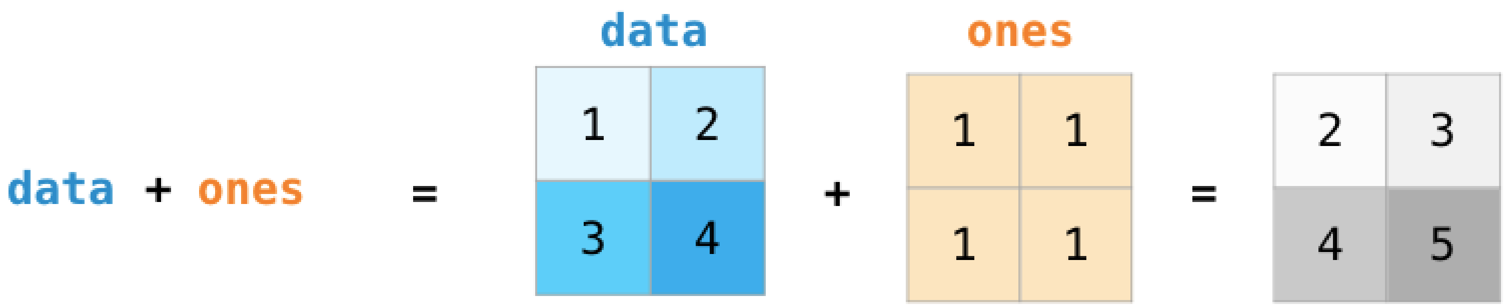
NumPy: Aggregation

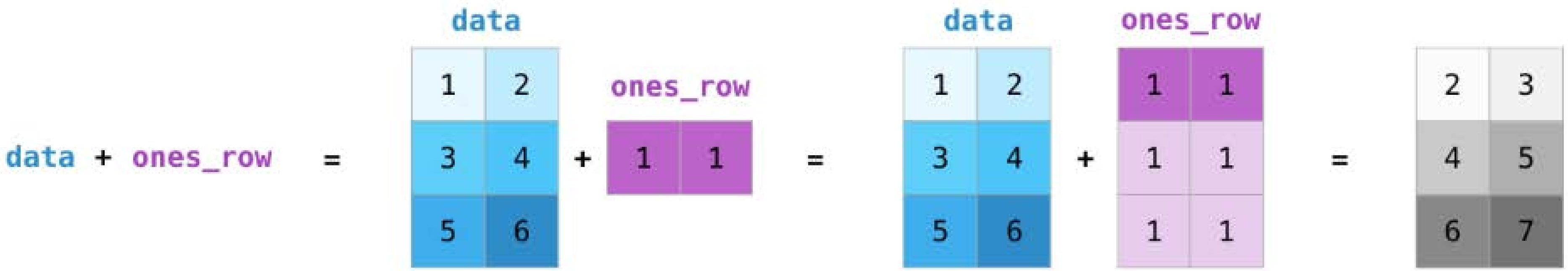


NumPy: Creating Matrices

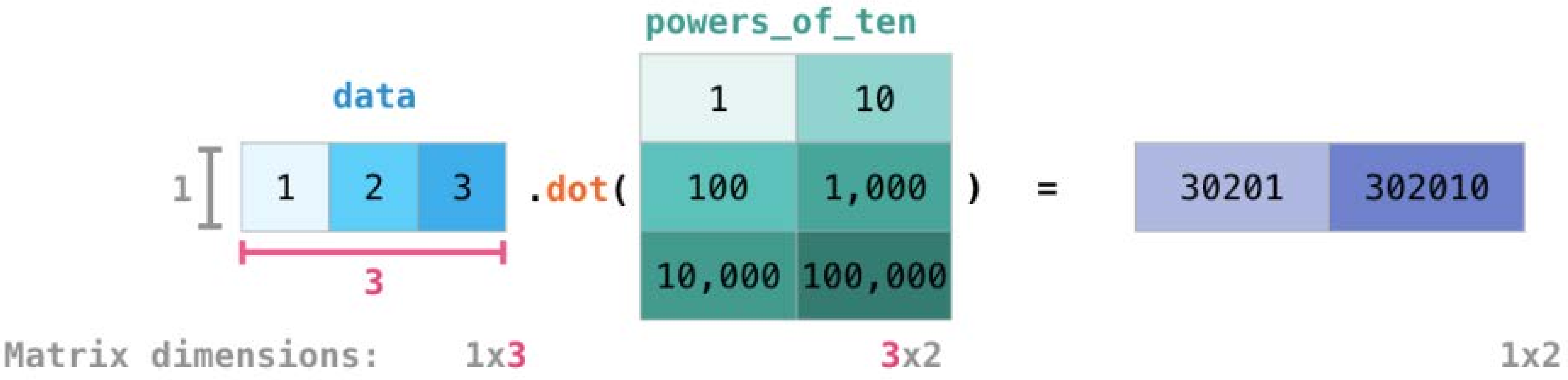


NumPy: Matrix Arithmetic

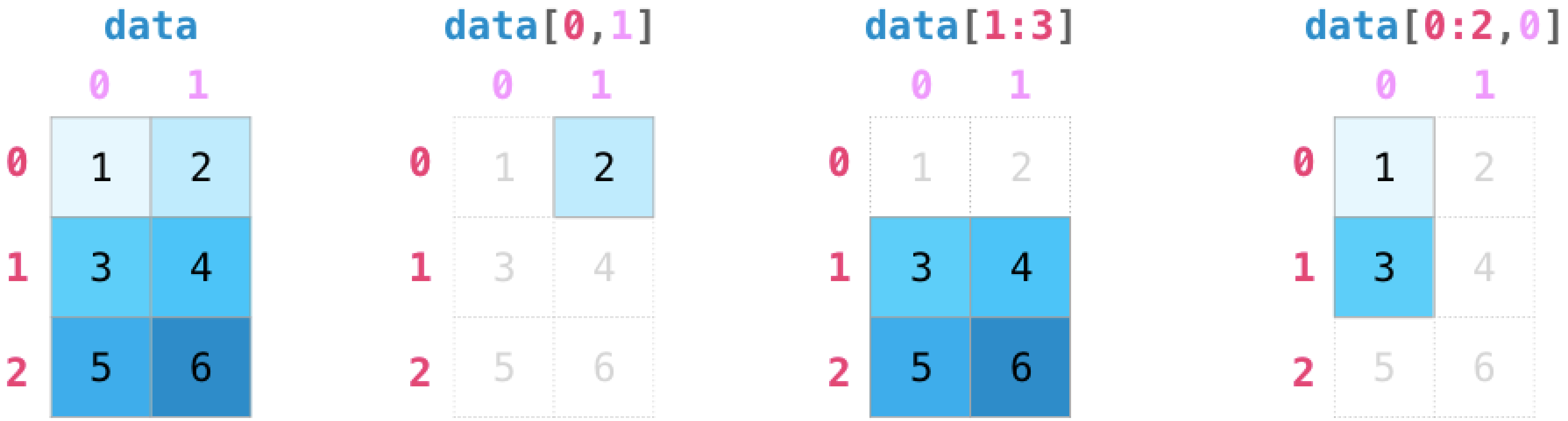




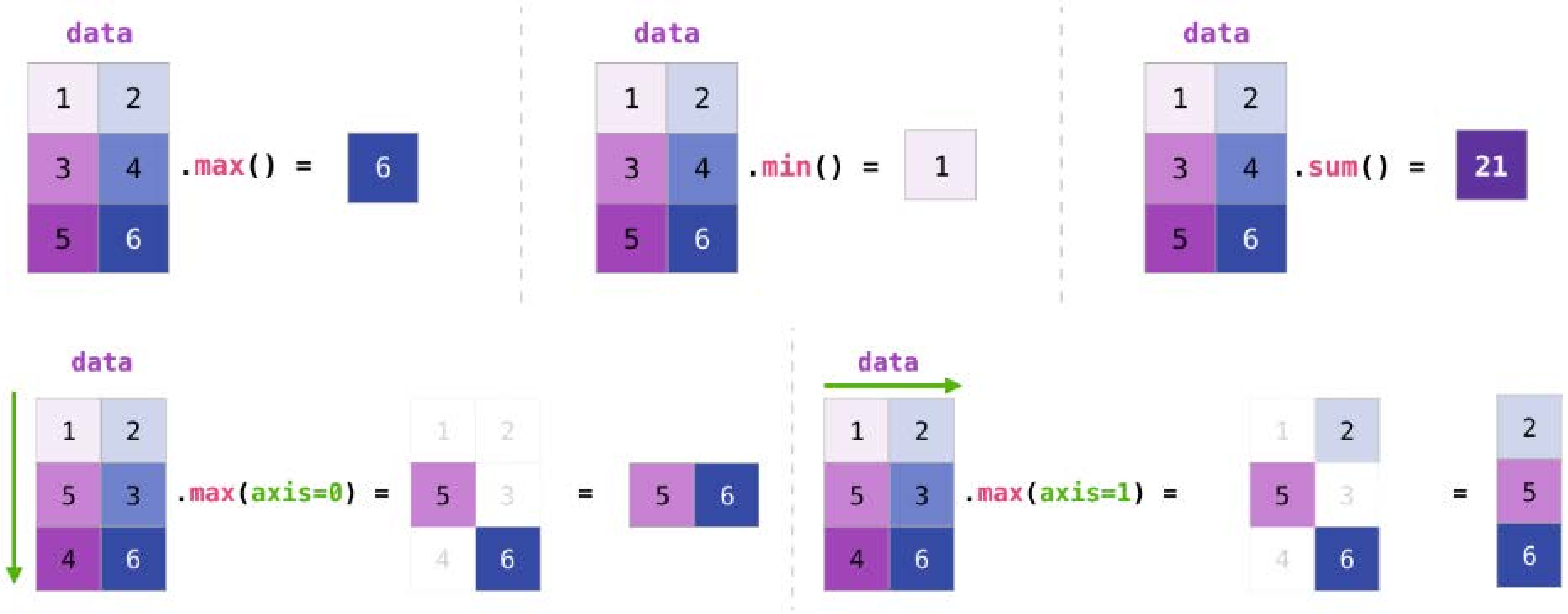
NumPy: Dot Product



NumPy: Matrix Indexing

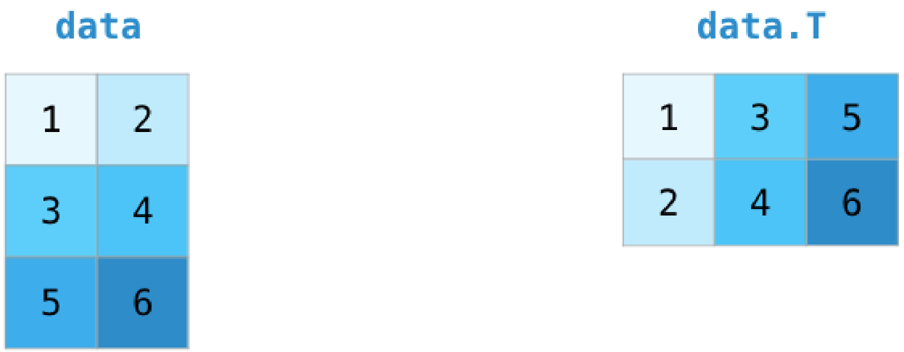


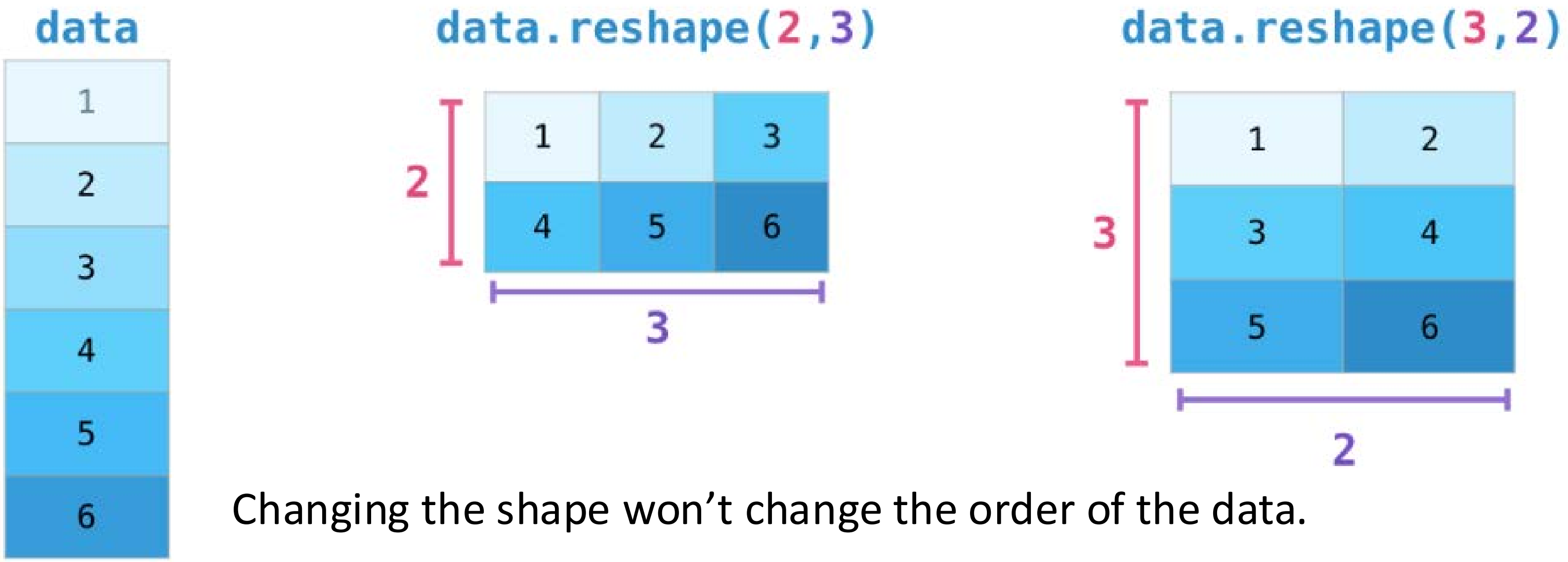
NumPy: Matrix Aggregation



axis specifies the dimension to be collapsed, rather than that to be returned

NumPy: Matrix Shape Manipulation

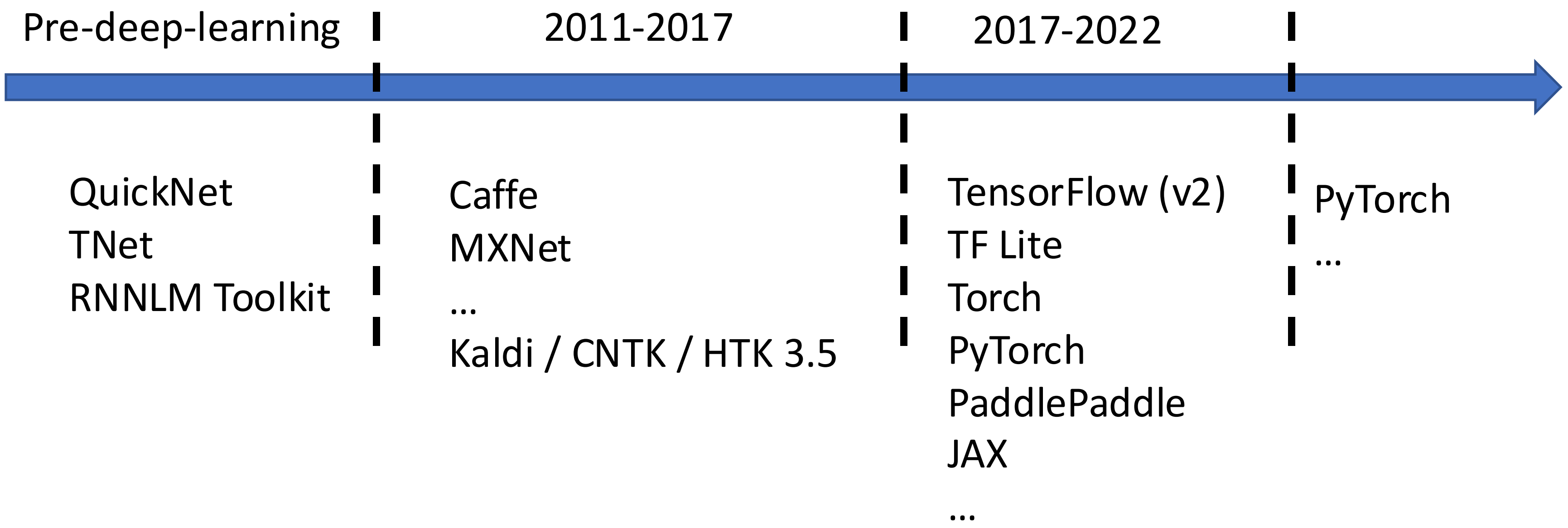




Changing the shape won't change the order of the data.

Deep Learning Toolkits

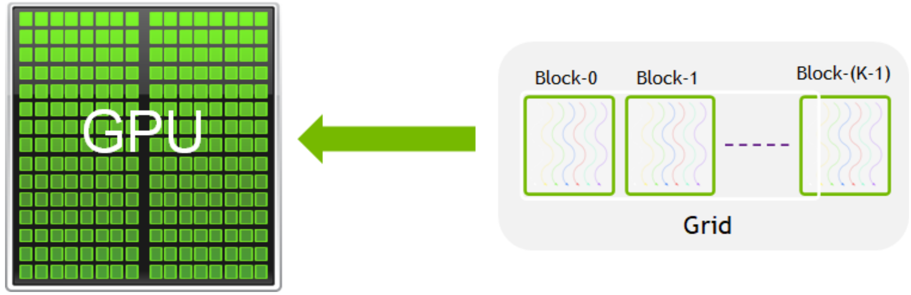
- Why deep learning toolkits are so popular?
  - Well-optimized GPU-based kernel functions (operators)
  - Auto-differentiation.
  - Unified platforms for reusing existing building blocks & sharing source codes

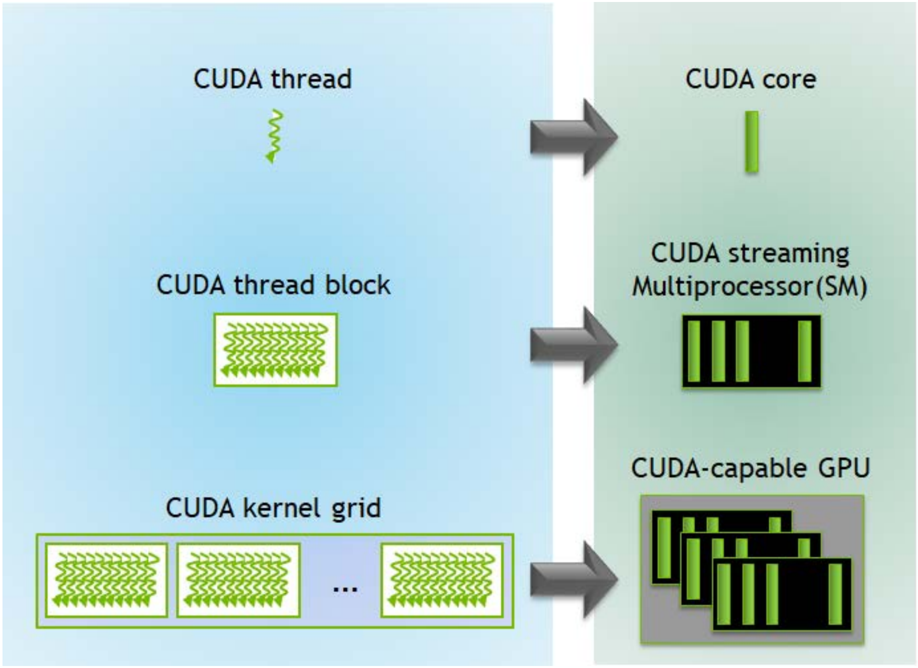


The CUDA Programming Model

- A (GP)GPU has
  - Global memory, L2 Cache
  - Many SMs: w/ registers, L1 cache, etc.
- CUDA: Compute unified device architecture
  - 1024 threads → A CUDA block
  - Many CUDA block → A grid
  - Each CUDA block is executed by an SM

    blockDim, blockIdx, threadIdx , __syncthreads
  - GEMM functions are vital to the speed
- Alternatives to GPU & CUDA
  - BLAS (basic linear algebra subprograms) /MKL
  - TPU etc.

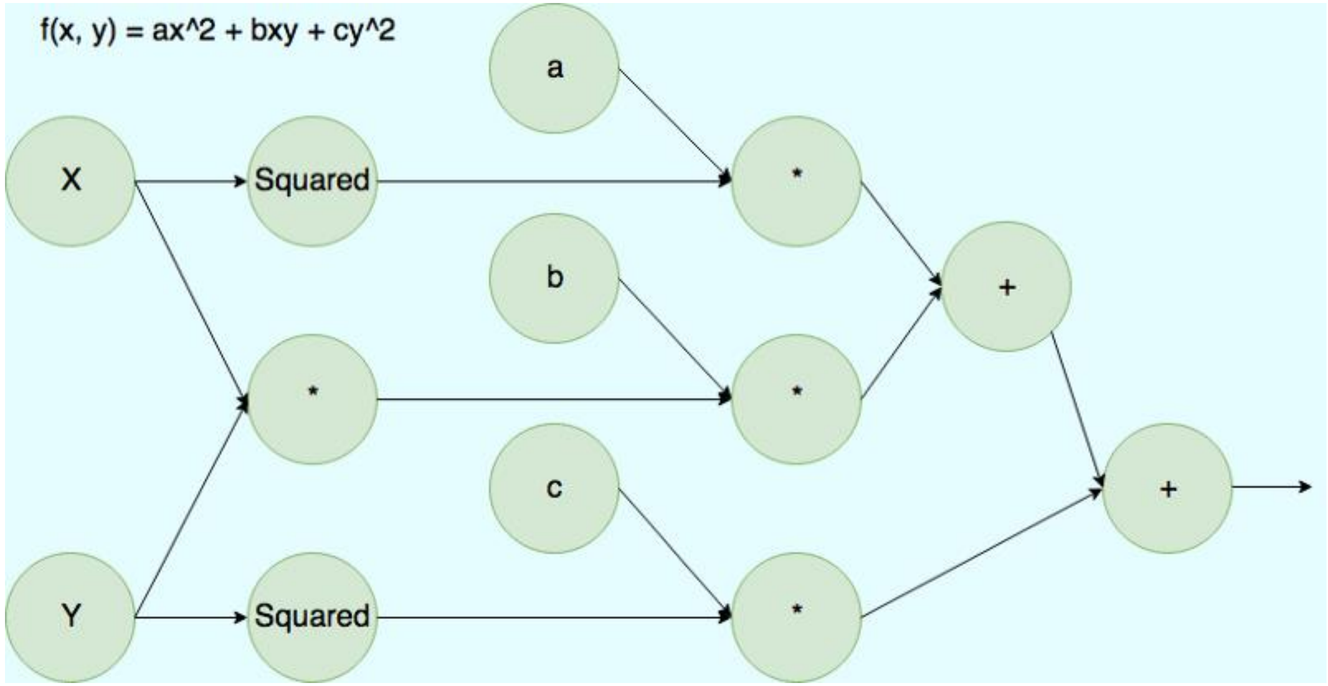




https://developer.nvidia.com/blog/cuda-refresher-cuda-programming-model/

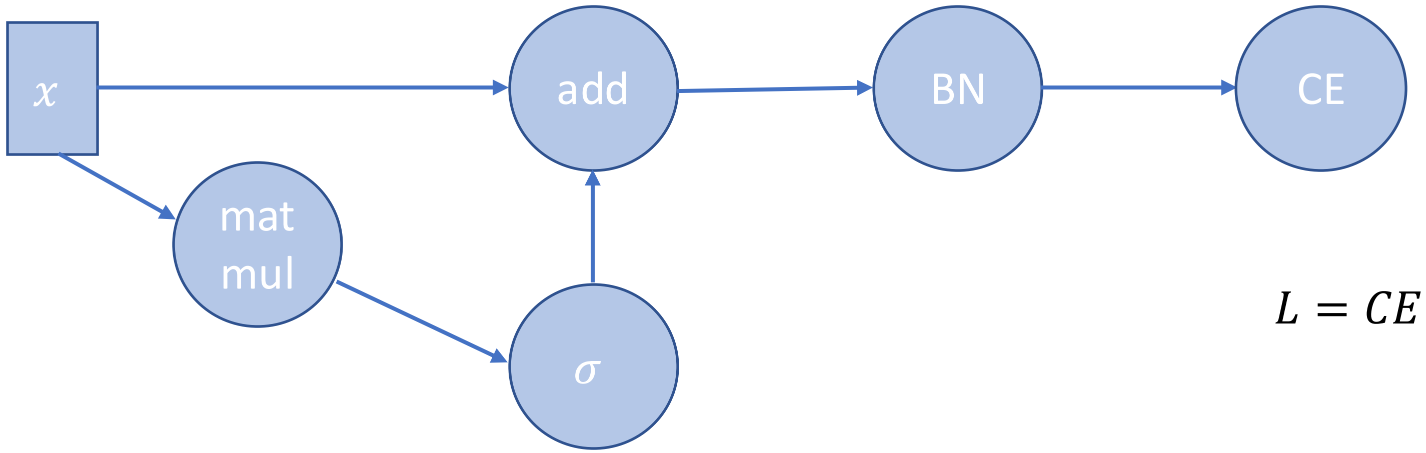
Computational Graph (CG)

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



Code A Fully-connected Layer using CG

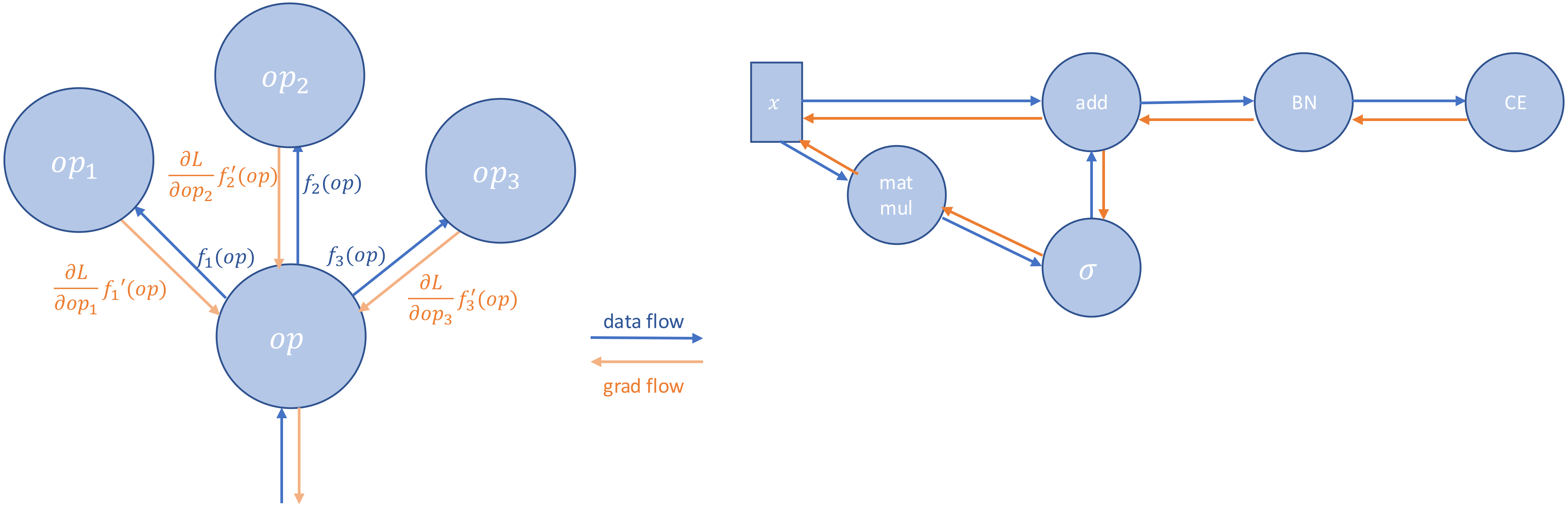
- A fully connected layer is presented as an example:
  - Represent it as a Directed Acyclic Graph
  - Node: operator/input
  - Edge: data flow



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

Why CG: Automatic Differentiation

- Backward procedure computes gradients following the reverse order of forward procedure → chain rule of partial differentiation



Deep Learning Toolkits

- TensorFlow
- PyTorch

TensorFlow: Static CG

- Static CG: build CG first, then re-use it
  - Different CGs can be used for training and test.
- 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
- Example:

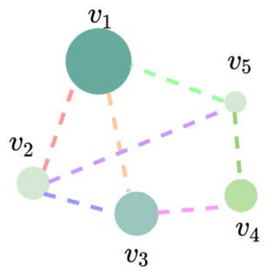
```
for t in range(500):
    # Forward pass: compute predicted y
    h = x.dot(w1)
    h_relu = np.maximum(h, 0)
    y_pred = h_relu.dot(w2)

    # 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

Customized Operators (with C++/CUDA)

- Implement customized computation



shortest path on graph

```
template <typename int_t>
__global__ void floyd_warshall_cuda_one_iter_kernel(
  int_t num_nodes,
  torch::PackedTensorAccessor64<int_t, 2> output_dist,
  torch::PackedTensorAccessor64<int_t, 2> output_pred,
  int_t k) {
```

Step 1: write C++/CUDA methods

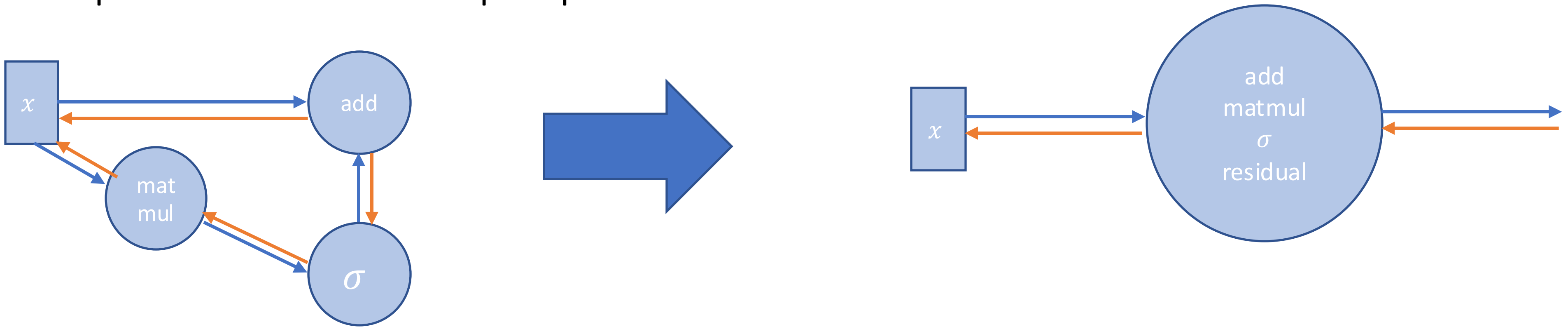
```
setup(
    name='graphormer_preprocess_cuda',
    ext_modules=[
        CUDAExtension('graphormer_preprocess_cuda', [
            'graphormer_preprocess_cuda.cpp',
            'graphormer_preprocess_cuda_kernel.cu',
        ]),
    ],
    cmdclass={
        'build_ext': BuildExtension
    })
```

Step 2: compile into python module

```
import graphormer_preprocess_cuda
graphormer_preprocess_cuda.floyd_warshall_batch
```

Step 3: call in pytorch code

- Fused operators: combine multiple operations into one



- A tutorial for Pytorch
  https://pytorch.org/tutorials/advanced/cpp_extension.html

Some Hands-on Examples

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

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, 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]))
```

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

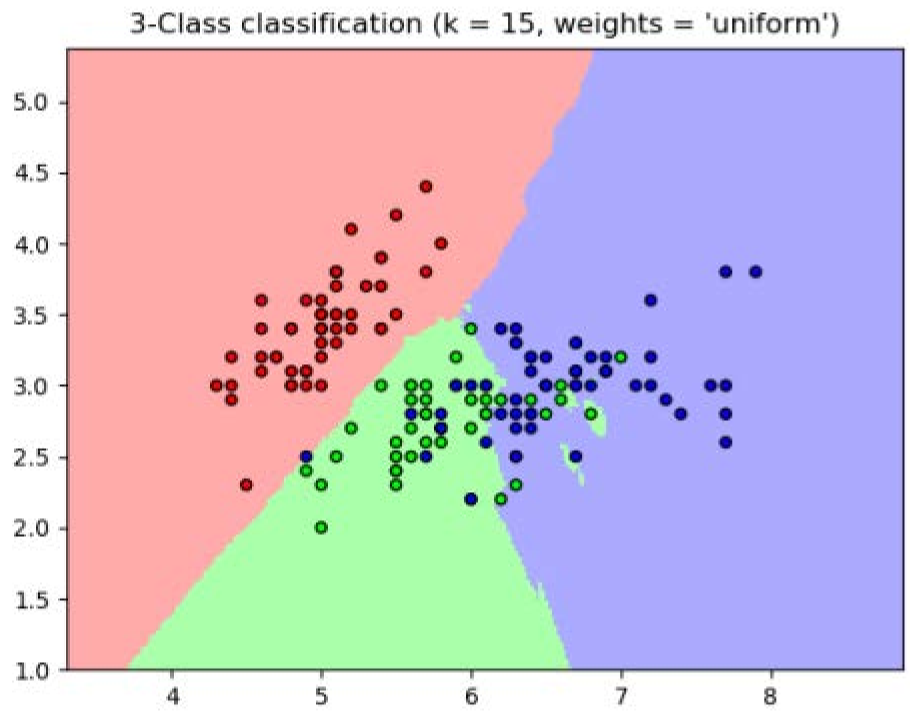
Out[3]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=15, p=2,
           weights='uniform')
```

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
```



LLMs and Copilot (using LightGBM & Boston dataset as an example)

ChatGPT o3-mini

Please help to write the code to use LightGBM to solve the Boston dataset

Reasoned for a few seconds >

Below is an example Python script that uses LightGBM to solve the Boston housing regression problem. Note that the Boston dataset has been deprecated in newer versions of scikit-learn due to ethical concerns. If you encounter any warnings or issues, you might consider using an alternative dataset. This example uses LightGBM's scikit-learn API via `LGBMRegressor`.

python Copy

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

# Load the Boston dataset
# Note: The Boston dataset is deprecated in recent versions of scikit-learn.
boston = load_boston()
X, y = boston.data, boston.target

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

# Create a LightGBM regressor model
model = lgb.LGBMRegressor(
    n_estimators=100,
    learning_rate=0.1,
    num_leaves=31,
    random_state=42
)
```

```python
# Train the model
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.2f}")
```

Explanation

- **Data Loading:** The Boston dataset is loaded using `load_boston()`.
- **Data Splitting:** The dataset is split into training and test sets using an 80-20 ratio.
- **Model Definition:** An instance of `LGBMRegressor` is created with some common hyperparameters.
- **Model Training:** The model is fitted on the training data.
- **Prediction & Evaluation:** Predictions are made on the test set and evaluated using mean squared error (MSE).

This script gives you a basic pipeline using LightGBM to solve a regression problem on the Boston dataset.

LLMs and Copilot

ChatGPT o3-mini

I want to also include a validation set

Reasoned about model with validation set for a few seconds >

Below is an updated code example that splits the data into training, validation, and test sets. The validation set is then used for early stopping during training with LightGBM:

python Copy

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

# Load the Boston dataset (note: this dataset is deprecated in recent scikit-learn ve
boston = load_boston()
X, y = boston.data, boston.target

# First, split the data into training and temporary sets (70% train, 30% temporary)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state

# Then split the temporary set into validation and test sets (50% each, resulting in 
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_

# Create a LightGBM regressor model
model = lgb.LGBMRegressor(
    n_estimators=1000,       # Increase the maximum number of iterations
    learning_rate=0.1,
    num_leaves=31,
    random_state=42
)
```

Copy

```python
# Train the model with a validation set for early stopping
model.fit(
    X_train,
    y_train,
    eval_set=[(X_val, y_val)],
    early_stopping_rounds=10,  # Stop if validation score doesn't improve for 10 roun
    verbose=True
)

# Predict on the test set
y_pred = model.predict(X_test)

# Evaluate the model using Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error on test set: {mse:.2f}")
```

Explanation

- **Data Splitting:**
  - First, the data is split into a training set (70%) and a temporary set (30%).
  - The temporary set is further split equally into validation and test sets (each 15% of the original dataset).
- **Model Training with Early Stopping:**
  - The model is set with a higher number of estimators (`n_estimators=1000`) so that early stopping can determine the optimal number of iterations.
  - The `eval_set` parameter is used to provide the validation set, and training stops early if the validation metric does not improve for 10 rounds.
- **Evaluation:**
  - After training, the model's performance is evaluated on the test set using Mean Squared

Thanks for your listening!

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
```