지능형생물정보학

한양대학교 컴퓨터 소프트웨어학부
DATA MINING & BIOINFORMATICS LAB.

문희상

Index

1. Environment setup
2. Preparing input data
3. Construct & Training a model
4. Visualize the result with TensorBoard
5. Evaluate the result
6. Let’s start the practice
7. References

Overview: environment setup

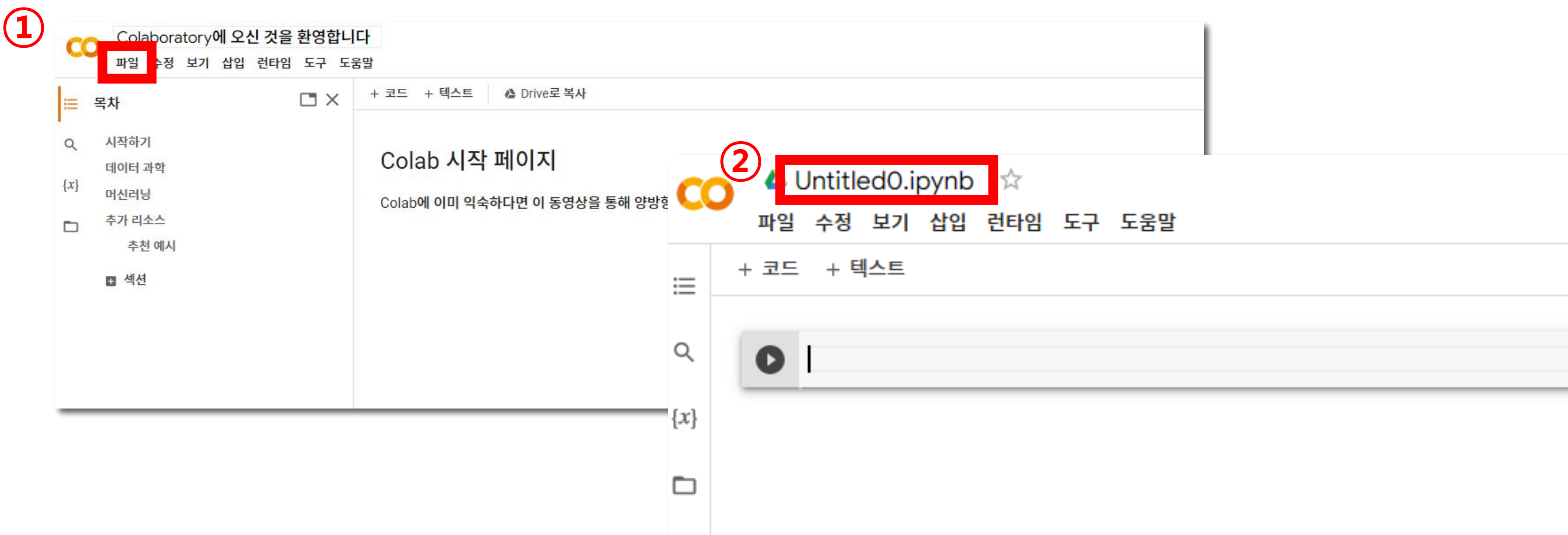
- Colab
  - Python 3.7.13
  - CUDA 11.2
  - Numpy 1.21.6
  - Scikit-learn 1.0.2
- Pytorch 1.12.0
- Pytorch-lightning 1.6.5
- Tensorboard 2.9.1
- Pytorch-geometric 2.0.4 (graph modeling)

Environment setup – Colab

https://colab.research.google.com/?hl=ko

① [파일]-[새 노트]

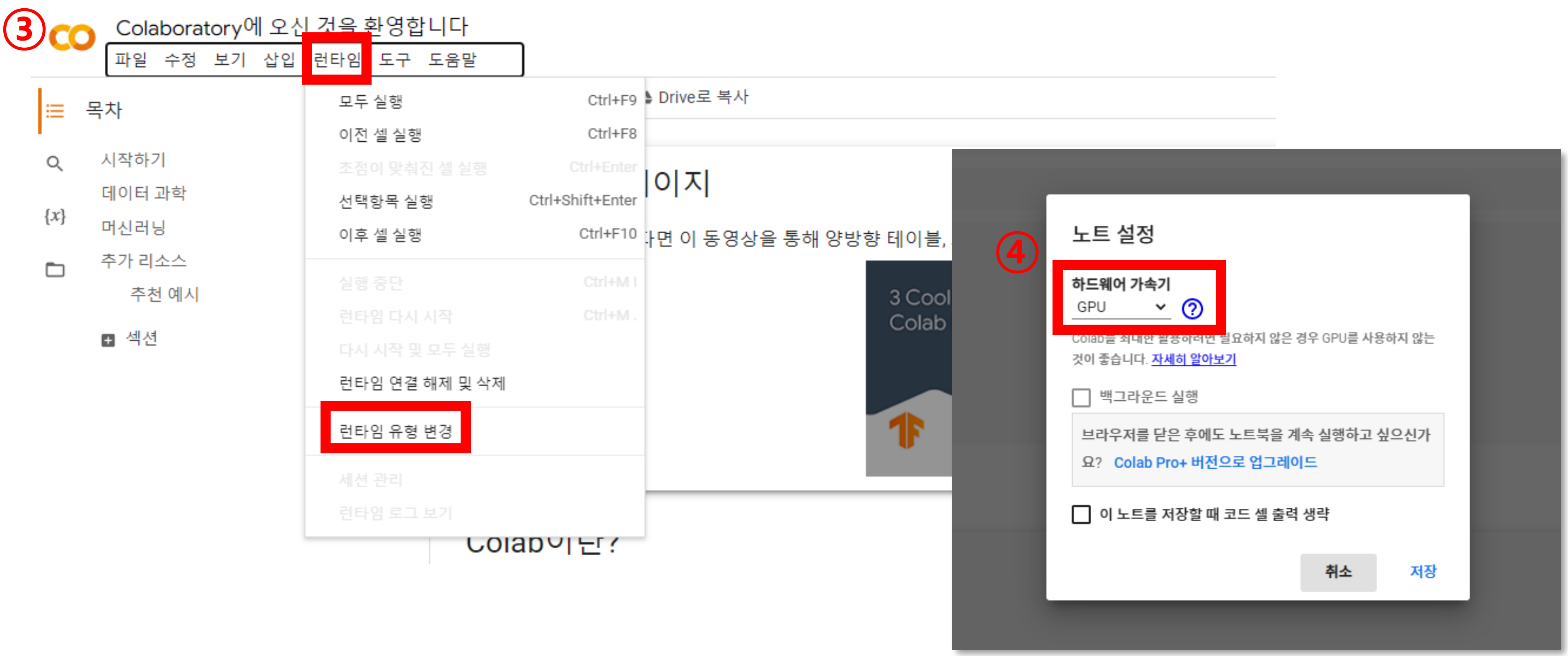
② 노트 명 변경



Environment setup – Colab

③ [런타임]-[런타임 유형 변경]

④ 하드웨어 가속기 → GPU



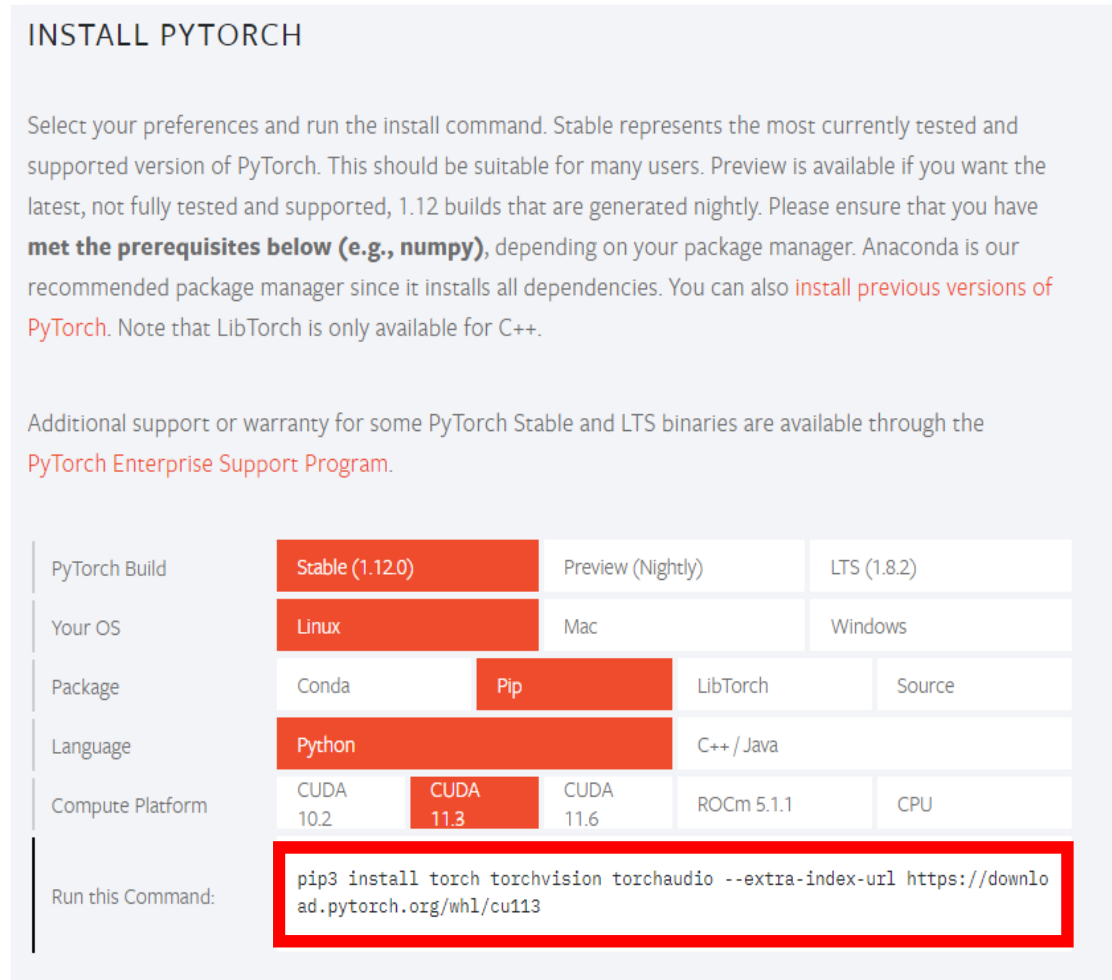
Environment setup – Colab

⑤ 코드 셀에 설치 명령어 작성

※ Shell 명령어 → !로 시작 (ex. !pip)

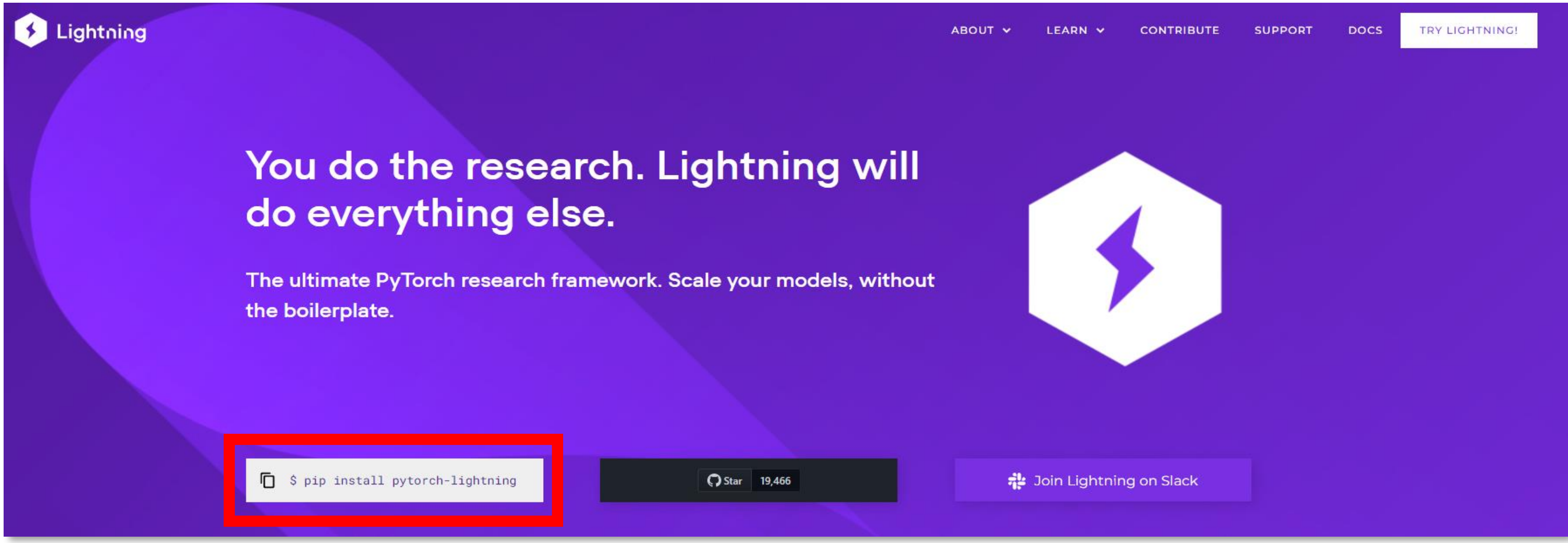
Environment setup – PyTorch

- https://pytorch.org/
- 1.12.0
- Linux
- Pip
- Python
- CUDA 11.3



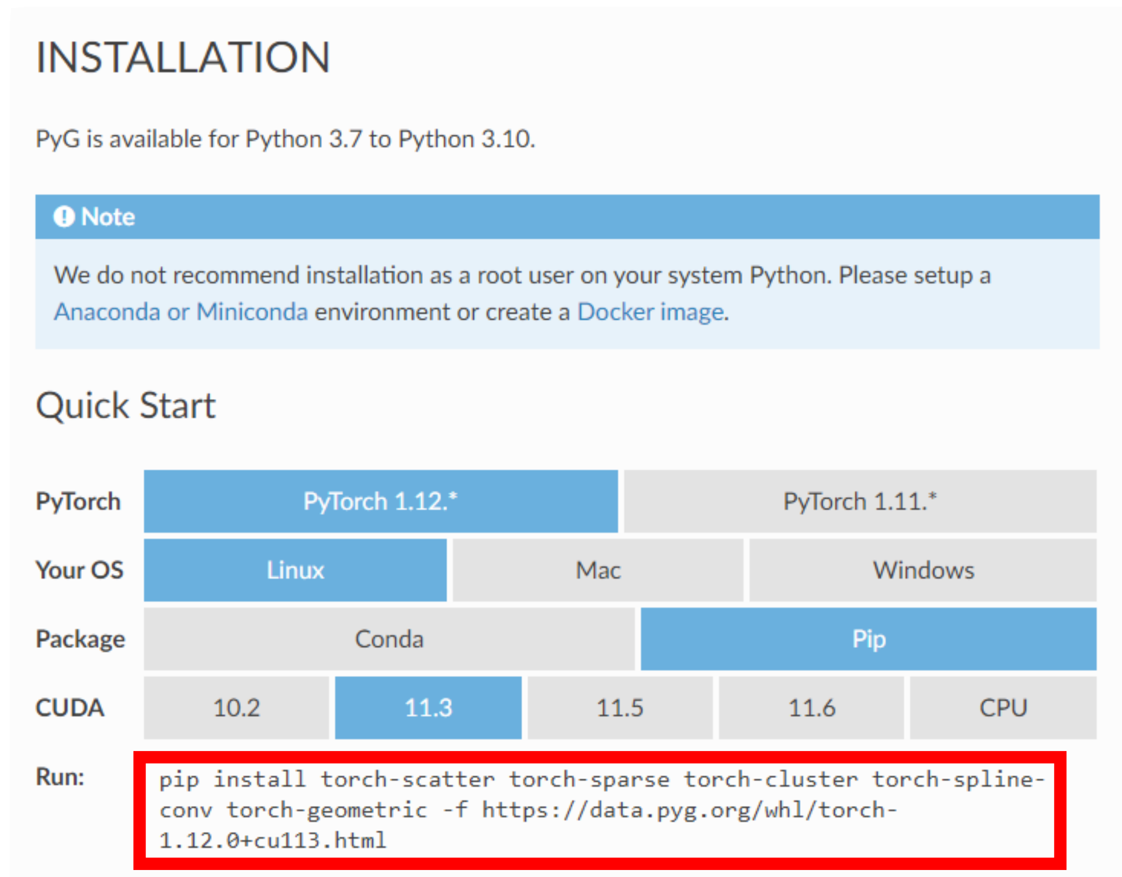
Environment setup – PyTorch-Lightning

- https://www.pytorchlightning.ai/
- Latest version



Environment setup – PyTorch Geometric

- https://pytorch-geometric.readthedocs.io/en/latest/notes/installation.html
- Pytorch 1.12.0
- Linux
- Pip
- CUDA 11.3



INSTALLATION

PyG is available for Python 3.7 to Python 3.10.

Note

We do not recommend installation as a root user on your system Python. Please setup a Anaconda or Miniconda environment or create a Docker image.

Quick Start

PyTorch	PyTorch 1.12.*		PyTorch 1.11.*		
Your OS	Linux	Mac	Windows		
Package	Conda		Pip		
CUDA	10.2	11.3	11.5	11.6	CPU

Run:

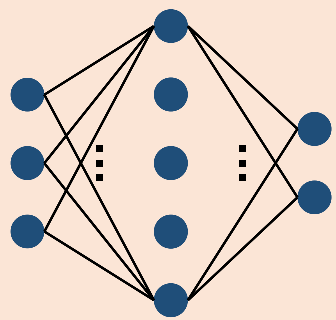
```
pip install torch-scatter torch-sparse torch-cluster torch-spline-
conv torch-geometric -f https://data.pyg.org/whl/torch-
1.12.0+cu113.html
```

Deep learning process

Step1

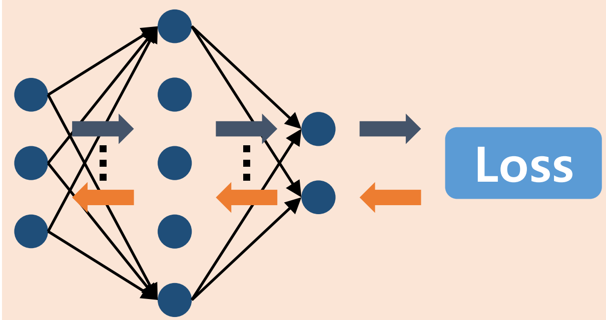
- Collecting data
- Preprocessing
- Dataset
- Data-loader

Step2



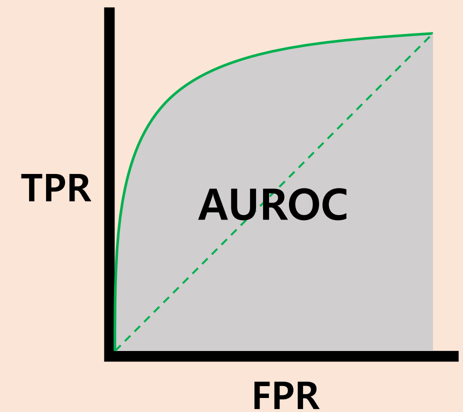
- Selecting method
- Building model

Step3



- Hyper-parameters
- Loss function
- Optimizer
- Training

Step4

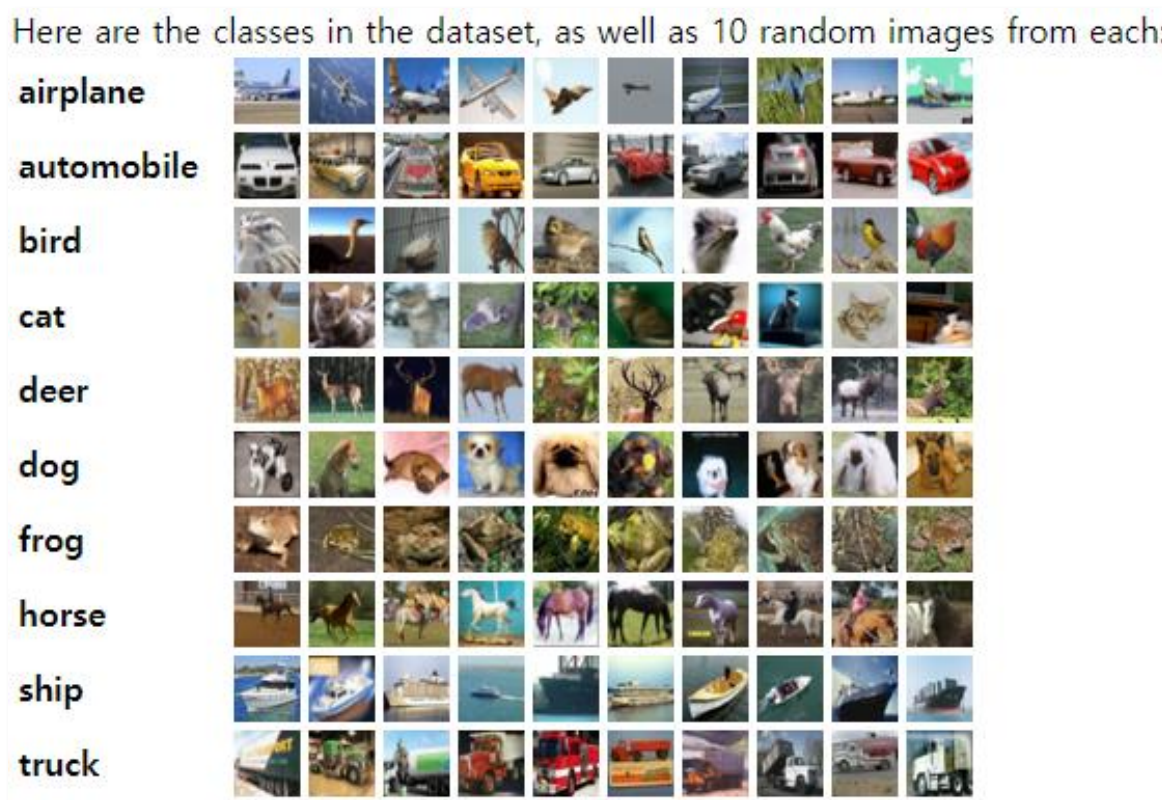


- Predicting
- Evaluating

Preparing input data

CIFAR 10 Dataset

- https://www.cs.toronto.edu/~kriz/cifar.html
- The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class.



Preparing input data

1. Load and normalize CIFAR10 datasets using torchvision.

```
# Get Datset
# https://tutorials.pytorch.kr/beginner/blitz/cifar10_tutorial.html
import torch
import torchvision
import torchvision.transforms as transforms
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

train_set = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
test_set = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

print('*==========Train Set==========*')
print(train_set)
print('*========== Test Set ==========*')
print(test_set)
```

Preparing input data

1. Load and normalize CIFAR10 datasets using torchvision.

```
# Get Datset
# https://tutorials.pytorch.kr/beginner/blitz/cifar10_tutorial.html
import torch
import torchvision
import torchvision.transforms as transforms
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

train_set = torchvision.datase
test_set = torchvision.datase
classes = ('plane', 'car', 'b

print('*==========Train Set==
print(train_set)
print('*========== Test Set ==
print(test_set)
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100% 170498071/170498071 [00:05<00:00, 44168057.72it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
*==========Train Set==========*
Dataset CIFAR10
    Number of datapoints: 50000
    Root location: ./data
    Split: Train
    StandardTransform
Transform: Compose(
               ToTensor()
               Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5))
           )
*========== Test Set ==========*
Dataset CIFAR10
    Number of datapoints: 10000
    Root location: ./data
    Split: Test
    StandardTransform
Transform: Compose(
               ToTensor()
               Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5))
           )
```

Preparing input data

- Watch the image in datasets

```
import matplotlib.pyplot as plt
import numpy as np

img = train_set[1][0]
label = train_set[1][1]
img = img.numpy()

① print(img.shape)
② print(f'{classes[label]}-{label}')

origin_img = img / 2 + 0.5 # unnormalized
plt.imshow(np.transpose(origin_img, (1, 2, 0)))
③ plt.show()


plt.imshow(np.transpose(img, (1, 2, 0)))
④ plt.show()
```

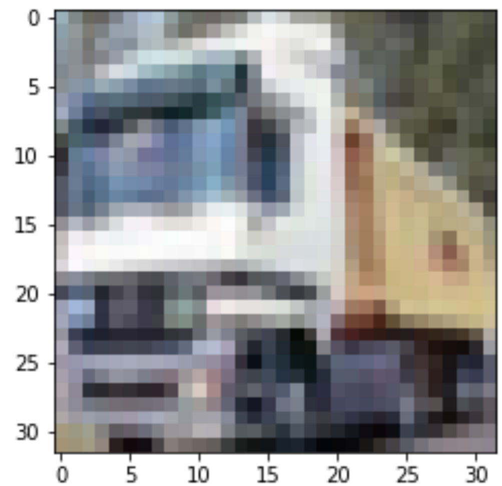
Preparing input data

- Watch the image in datasets

```
import matplotlib.pyplot a
import numpy as np

img = train_set[1][0]
label = train_set[1][1]
img = img.numpy()

print(img.shape)
print(f'{classes[label]}-

origin_img = img / 2 + 0.
plt.imshow(np.transpose(o
plt.show()


plt.imshow(np.transpose(i
plt.show()
```

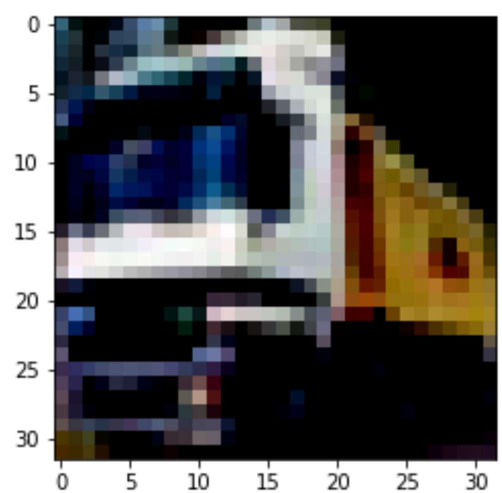
①, ②, ③, ④

```
(3, 32, 32) ①
truck-9 ②
```



③

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



④

← Normalize 적용

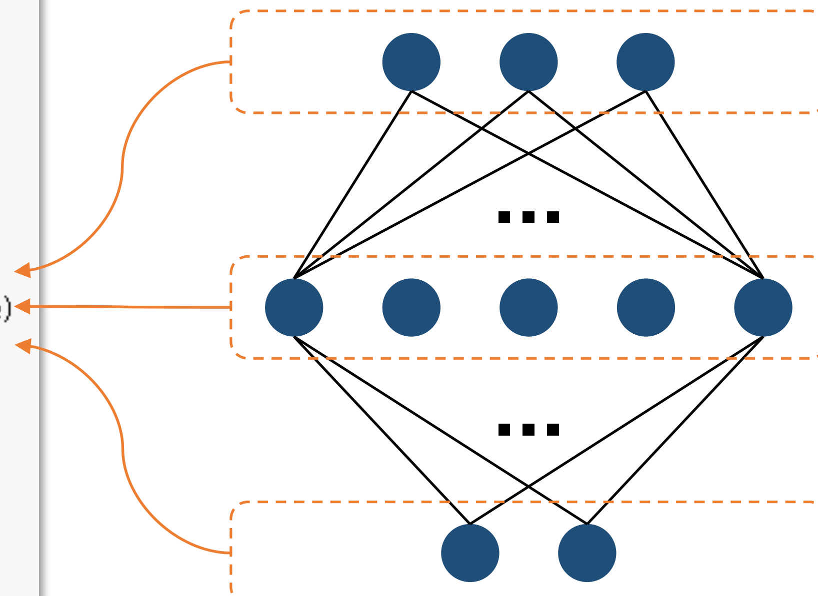
Preparing input data

2. Define a data module

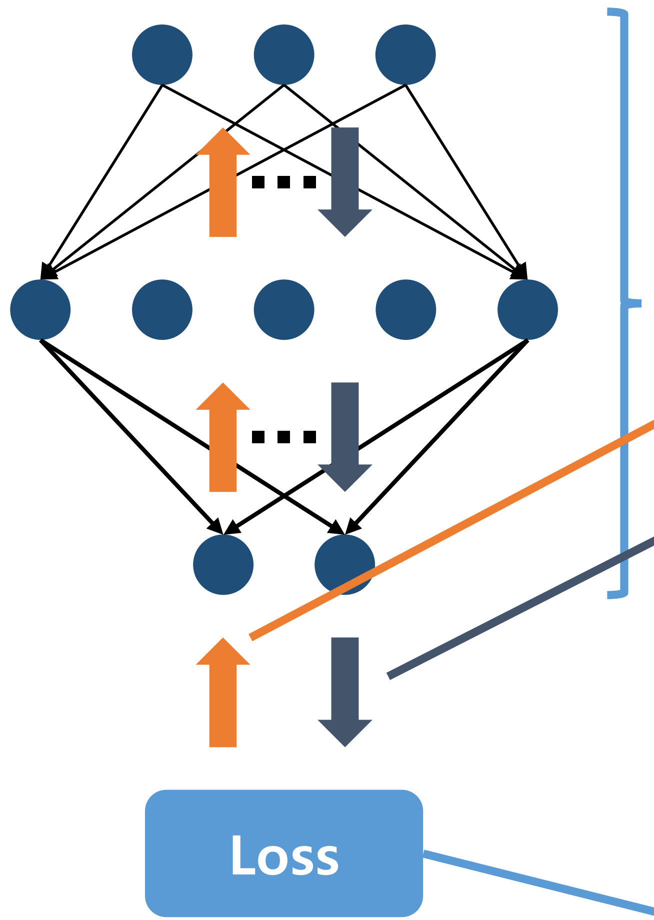
```
from torch.utils.data import DataLoader
from pytorch_lightning import LightningDataModule

class CustomDataModule(LightningDataModule):
  def __init__(self, training_set, test_set, batch_size=128, num_workers=2):
    super().__init__()
    self.training_set = training_set
    self.test_set = test_set

    # set parameters
    self.batch_size = batch_size
    self.num_workers = num_workers

  #how to make a batch
  def collate_function(self, batch):
    new_x = [x.flatten() for x, _ in batch]
    new_y = [torch.tensor(y, dtype=torch.long) for _, y in batch]
    return [torch.stack(new_x), torch.stack(new_y)]

  def train_dataloader(self):
    return DataLoader(self.training_set, batch_size=self.batch_size,
                      num_workers=self.num_workers, shuffle=True,
                      collate_fn=self.collate_function)

  def test_dataloader(self):
    return DataLoader(self.test_set, batch_size=self.batch_size,
                      num_workers=self.num_workers,
                      collate_fn=self.collate_function)

# data_module = CustomDataModule(train_set, test_set)
# trainer.fit(model, datamodule = data_module)
```

Construct a model

```python
import torch.nn as nn
import torch.nn.functional as F

class DeepNeuralNet(nn.Module):
  def __init__(self, input_size, label_size, layer_size=64):
    super().__init__()
    self.input_size = input_size
    self.layer_size = layer_size
    self.label_size = label_size
    self.set_layer()

  def set_layer(self):
    # Build layer
    self.first_layer = nn.Linear(self.input_size, self.layer_size)
    self.second_layer = nn.Linear(self.layer_size, self.layer_size)
    self.last_layer = nn.Linear(self.layer_size, self.label_size)

  def forward(self, x):
    # Define operation
    x = self.first_layer(x)
    x = F.relu(x)
    x = self.second_layer(x)
    x = F.relu(x)
    x = self.last_layer(x)

    return x
```



Construct a model




```python
from pytorch_lightning import LightningModule

from torch.nn import CrossEntropyLoss, Softmax
from torch.optim import Adam

class LitCIFAR10Model(LightningModule):
  def __init__(self, model, learning_rate=1e-3):
    super().__init__()
    self.model =  model
    self.lr = learning_rate

  def forward(self, x):
    out = self.model(x)
    return Softmax(dim=1)(out)

  def configure_optimizers(self):
    optimizer=Adam(self.parameters(), lr=self.lr)
    return optimizer

  def training_step(self, batch, batch_idx):
    x, y = batch
    logits = self(x)
    loss = self.loss_function(logits, y)
    predict = torch.argmax(logits, dim=1)
    return {'loss': loss, 'predict': predict, 'answer': y}

  def training_epoch_end(self, outputs):
    loss = [output['loss'] for output in outputs]
    avg_loss = sum(loss) / len(outputs)
    self.logger.experiment.add_scalar("Loss/Epoch",
                                      avg_loss,
                                      self.current_epoch)

  def predict_step(self, batch, batch_idx):
    x, y = batch
    logits = self.model(x)
    predict = torch.argmax(logits, dim=1)
    return predict

  def loss_function(self, output, target):
    return CrossEntropyLoss()(output, target)
```

Training a model

```python
from pytorch_lightning import Trainer
from pytorch_lightning.loggers import TensorBoardLogger

logger = TensorBoardLogger('./log', name="LitCIFAR10-DeepNN-Adam-128,1e-3")
data_module = CustomDataModule(train_set, test_set)

architecture = DeepNeuralNet(32*32*3, 10) #32*32(pixels)*3 channels, labels
model = LitCIFAR10Model(architecture)

trainer = Trainer(max_epochs=10, accelerator='gpu', devices=[0], logger=logger)
#trainer = Trainer(max_epochs=10, accelerator='cpu', logger=logger)

trainer.fit(model, datamodule=data_module)
```

```
  | Name  | Type          | Params
----------------------------------------
0 | model | DeepNeuralNet | 201 K
----------------------------------------
201 K     Trainable params
0         Non-trainable params
201 K     Total params
0.806     Total estimated model params size (MB)
/usr/local/lib/python3.7/dist-packages/pytorch_lightning/loggers/tensorboard.py:251: UserWarning: Could not log computational graph since the `model.exampl
  "Could not log computational graph since the"
Epoch 9: 100% 391/391 [00:11<00:00, 35.33it/s, loss=1.14, v_num=0, train_loss=1.150]
INFO:pytorch_lightning.utilities.rank_zero:`Trainer.fit` stopped: `max_epochs=10` reached.
```

Visualize the result with TensorBoard

```
logger = TensorBoardLogger('./log', name="LitCIFAR10-DeepNN-Adam-128,1e-3")
```

```
trainer = Trainer(max_epochs=10, accelerator='gpu', devices=[0], logger=logger)
```

```
def training_epoch_end(self, outputs):
  loss = [output['loss'] for output in outputs]
  avg_loss = sum(loss) / len(outputs)
  self.logger.experiment.add_scalar("Loss/Epoch",
                                    avg_loss,
                                    self.current_epoch)
```

```
%load_ext tensorboard
%tensorboard --logdir "./log"
```

Visualize the result with TensorBoard

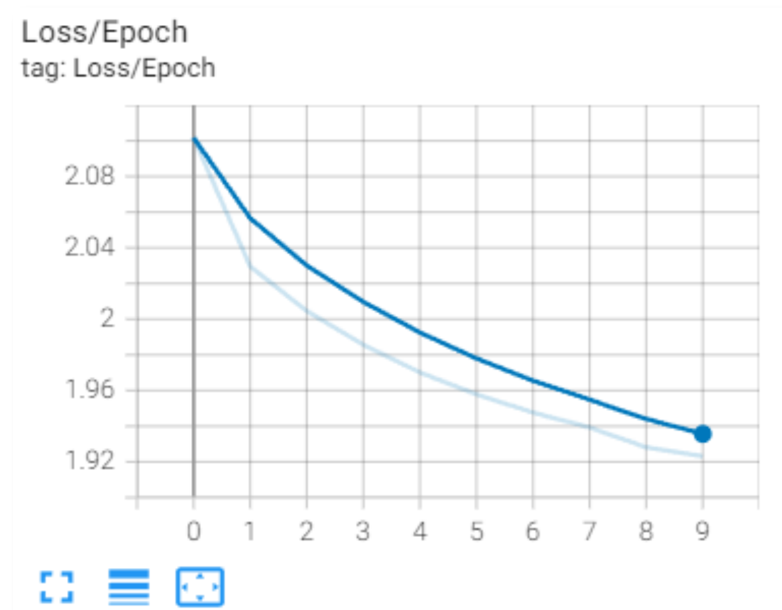
```
logger = TensorBoardLogger('./log', name="LitCIFAR10-DeepNN-Adam-128,1e-3")
```

```
trainer = Trainer(max_epochs=10, accelerator='gpu', devices=[0], logger=logger)
```

```
def training_epoch_end(self, outputs):
  loss = [output['loss'] for output in outputs]
  avg_loss = sum(loss) / len(outputs)
  self.logger.experiment.add_scalar("Loss/Epoch",
                                    avg_loss,
                                    self.current_epoch)
```

```
%load_ext tensorboard
%tensorboard --logdir "./log"
```

403 error -> 쿠키 허용 -> 재시작



Evaluate the result

- $Accuracy = \frac{correct\ predictions}{all\ predictions}$

```
predict = trainer.predict(model, dataloaders=data_module.test_dataloader())
predict = torch.cat(predict)

answer = [batch[1] for batch in data_module.test_dataloader()]
answer = torch.cat(answer)

correct = (predict == answer).sum()
total = answer.shape[0]
accuracy = correct / total

print(f'correct: {correct}, total: {total}, accuracy: {accuracy}')
```

- torchmetrics API can be used for this instead

```
# You can use torchmetric
import torchmetrics

acc = torchmetrics.functional.accuracy(predict, answer)
print(f"accuracy-torchmetrics: {acc}")
```

Evaluate the result

- $Accuracy = \dfrac{correct\ predictions}{all\ predictions}$

```python
predict = trainer.predict(model, dataloaders=data_module.test_dataloader())
predict = torch.cat(predict)

answer = [batch[1] for batch in data_module.test_dataloader()]
answer = torch.cat(answer)

correct = (predict == answer).sum()
total = answer.shape[0]
accuracy = correct / total

print(f'correct: {correct}, total: {total}, accuracy: {accuracy}')
```

- torchmetrics API can be used for this instead

```python
# You can use torchmetric
import torchmetrics

.functional.accuracy(predict, answer)
rchmetrics: {acc}")
```

```
correct: 4879, total: 10000, accuracy: 0.4878999888896942
accuracy-torchmetrics: 0.4878999888896942
```

Practice – GNN

1. TUDataset

- https://chrsmrrs.github.io/datasets/
- 120+ datasets for graph classification and regression
- Tox21 AhR (example)

```
Tox21_AhR_training(8169)     Data(edge_index=[2, 52], x=[25, 50], edge_attr=[52, 4], y=[1])
Tox21_AhR_testing(272)       Data(edge_index=[2, 44], x=[20, 51], edge_attr=[44, 4], y=[1])
Tox21_AhR_evaluation(607)    Data(edge_index=[2, 118], x=[53, 53], edge_attr=[118, 4], y=[1])
```

  - x: Node features
  - y: Answer
  - Edge_index: Directed edge (Beginning node, End node)
  - Edge_attr: Edge features

```python
# Dataset
from torch_geometric.datasets import TUDataset

training_dataset = TUDataset('./dataset', 'Tox21_AhR_training')
validation_dataset = TUDataset('./dataset', 'Tox21_AhR_testing')
test_dataset = TUDataset('./dataset', 'Tox21_AhR_evaluation')
```

Practice – GNN

2. Construct model

```
# Pytorch-lightning datamodule
from torch.utils.data import DataLoader
from pytorch_lightning import LightningDataModule
from torch_geometric.data import Data, Batch
from torch.nn.functional import pad

class CustomData(LightningDataModule):
  def __init__(self, training_set, validation_set, test_set, batch_size=128, num_workers=1):
    super().__init__()
    self.training_set = training_set
    self.validation_set = validation_set
    self.test_set = test_set
    self.batch_size = batch_size
    self.num_workers = num_workers

  def collate_function(self, batch):
    return Batch.from_data_list([Data(edge_index=data.edge_index,
                                      x=pad(data.x, (0,3), 'constant', 0.)[:,:53],
                                      edge_attr=data.edge_attr,
                                      y=data.y.unsqueeze(0).float()) for data in batch])

  def train_dataloader(self):
    return DataLoader(self.training_set, batch_size=self.batch_size, num_workers=self.num_workers, shuffle=True, collate_fn=self.collate_function)

  def val_dataloader(self):
    return DataLoader(self.validation_set, batch_size=self.batch_size, num_workers=self.num_workers, collate_fn=self.collate_function)

  def test_dataloader(self):
    return DataLoader(self.test_set, batch_size=self.batch_size, num_workers=self.num_workers, collate_fn=self.collate_function)
```

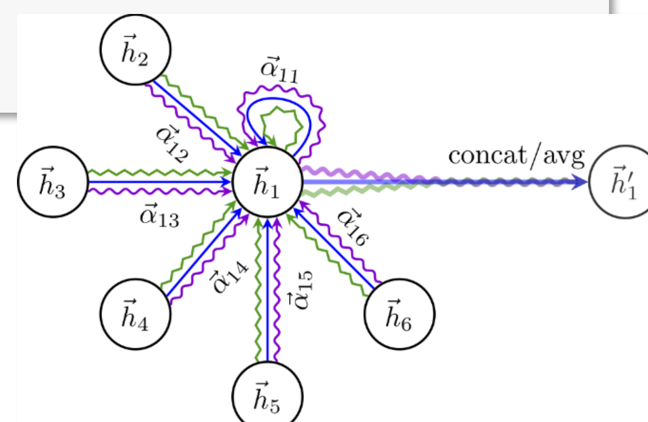
Practice – GNN

2. Construct model

- GATConv
  - In_channels
  - out_channels
  - Dropout
  - Heads
    - Multi-head attention
  - Concat
  - …

```
# Pytorch and pytorch-geomtric module
from torch.nn import Module, Linear
from torch_geometric.nn import GATConv, global_mean_pool

class CustomGAT(Module):
  def __init__(self, input_size, label_size, layer_size=64, dropout=0.1, heads=2):
    super().__init__()
    self.input_size = input_size
    self.layer_size = layer_size
    self.dropout = dropout
    self.heads = heads
    self.label_size = label_size
    self.setup()

  def setup(self):
    self.first_layer = GATConv(self.input_size, self.layer_size, dropout=self.dropout, heads=self.heads, concat=False)
    self.last_layer = GATConv(self.layer_size, self.layer_size, dropout=self.dropout, heads=self.heads, concat=False)
    self.ffnn = Linear(self.layer_size, self.label_size)

  def convert_graph_into_single_vector(self, graph_hidden, batch_index):
    return global_mean_pool(graph_hidden, batch_index)

  def forward(self, batch):
    z = self.first_layer(batch.x, batch.edge_index)
    z = self.last_layer(z, batch.edge_index)
    z = self.convert_graph_into_single_vector(z, batch.batch)
    z = self.ffnn(z)
    return z
```

- Golobal_mean_pool
  - Returns batch-wise graph-level-outputs by averaging node features across the node dimension



GAT image from https://github.com/PetarV-/GAT

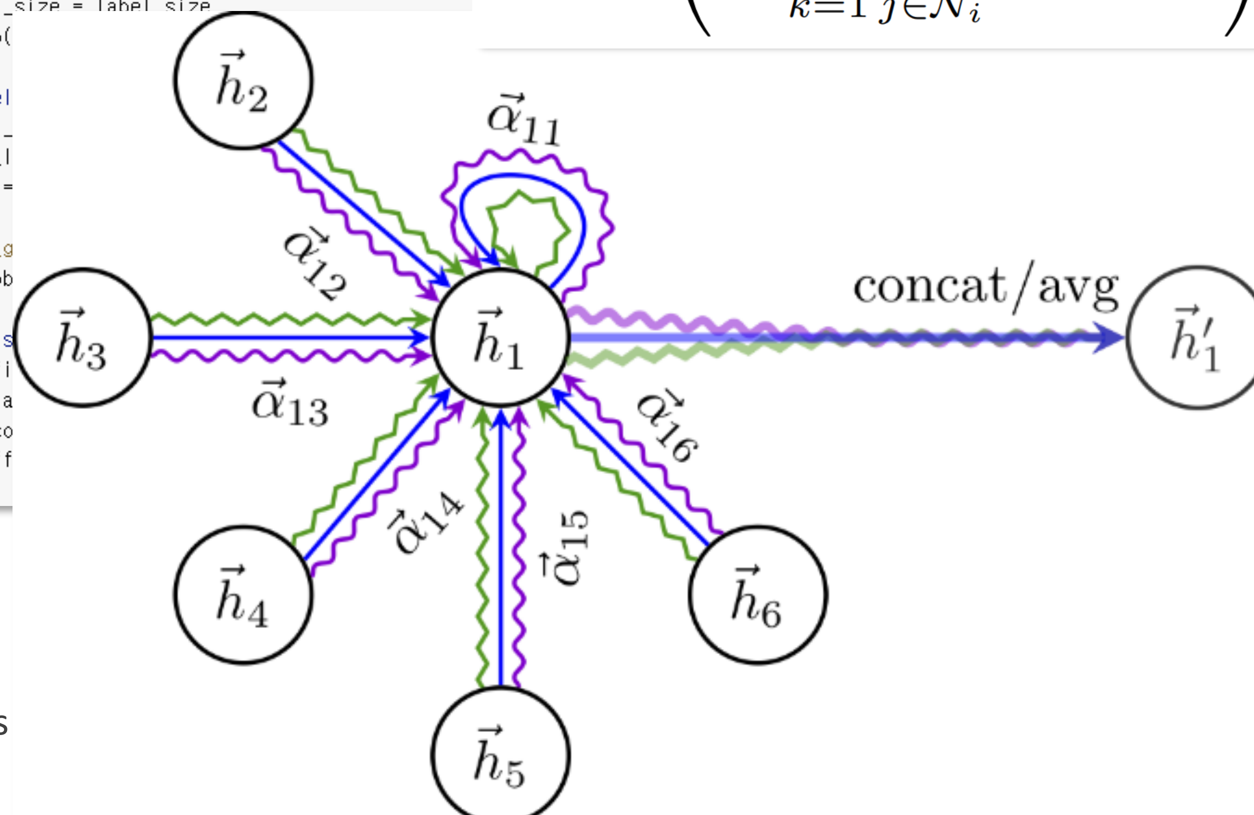
Practice – GNN

2. Construct model

- GATConv
  - In_channels
  - out_channels
  - Dropout
  - Heads
    - Multi-head attention
  - Concat
  - …

```
# Pytorch and pytorch-geomtric module
from torch.nn import Module, Linear
from torch_geometric.nn import GATConv, global_mean_pool

class CustomGAT(Module):
  def __init__(self, input_size, label_size, layer_size
    super().__init__()
    self.input_size = input_size
    self.layer_size = layer_size
    self.dropout = dropout
    self.heads = heads
    self.label_size = label_size
    self.setup(

  def setup(sel
    self.first_
    self.last_l
    self.ffnn =

  def convert_g
    return glob

  def forward(s
    z = self.fi
    z = self.la
    z = self.co
    z = self.ff
    return z
```

$$\vec{h}'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i}\alpha_{ij}^{k}\mathbf{W}^k\vec{h}_j\right)$$



- Golobal_mean_pool
  - Returns batch-wise graph-level-outputs across the node dimension

GAT image from https://github.com/PetarV-/GAT

Practice – GNN

2. Construct model

```python
# Pytorch-lightning module
from pytorch_lightning import LightningModule
from torch.nn import  BCEWithLogitsLoss
from torch.optim import Adam

class CustomModel(LightningModule):
  def __init__(self, model, learning_rate=1e-3):
    super().__init__()
    self.model = model
    self.lr = learning_rate

  def forward(self, batch, mode):
    z = self.model(batch)
    loss = self.loss_function(z, batch.y)
    self.log(f"{mode}_loss", loss, batch_size=batch.y.size(0), prog_bar=True, on_step=False, on_epoch=True)
    return loss, z, batch.y

  def training_step(self, batch, batch_idx):
    loss, predict, answer = self(batch, 'train')
    return {'loss':loss, 'predict':predict, 'answer':answer}

  def validation_step(self, batch, batch_idx):
    loss, predict, answer = self(batch, 'val')
    return {'loss':loss, 'predict':predict, 'answer':answer}

  def test_step(self, batch, batch_idx):
    loss, predict, answer = self(batch, 'test')
    return {'loss':loss, 'predict':predict, 'answer':answer}

  def predict_step(self, batch, batch_idx):
    predict = self.model(batch)
    return predict

  def loss_function(self, output, target):
    return BCEWithLogitsLoss(reduction='mean')(output, target)

  def configure_optimizers(self):
    optimizer = Adam(self.parameters(), lr=self.lr)
    return optimizer
```

Practice – GNN

3. Training the model

```
# Training
from pytorch_lightning import Trainer

import warnings
warnings.filterwarnings(action='ignore')
# warnings.filterwarnings(action='default')

data_module = CustomData(training_dataset, validation_dataset, test_dataset)

gat = CustomGAT(53, 1)
model = CustomModel(gat)

trainer = Trainer(max_epochs=1, accelerator='gpu', devices=[0])
trainer.fit(model, datamodule=data_module)
```

```
  | Name  | Type      | Params
------------------------------------
0 | model | CustomGAT | 15.7 K
------------------------------------
15.7 K    Trainable params
0         Non-trainable params
15.7 K    Total params
0.063     Total estimated model params size (MB)
Epoch 0: 100% 67/67 [00:12<00:00, 5.16it/s, loss=0.384, v_num=0, val_loss=0.361, train_loss=0.463]
`Trainer.fit` stopped: `max_epochs=1` reached.
```

Practice – GNN

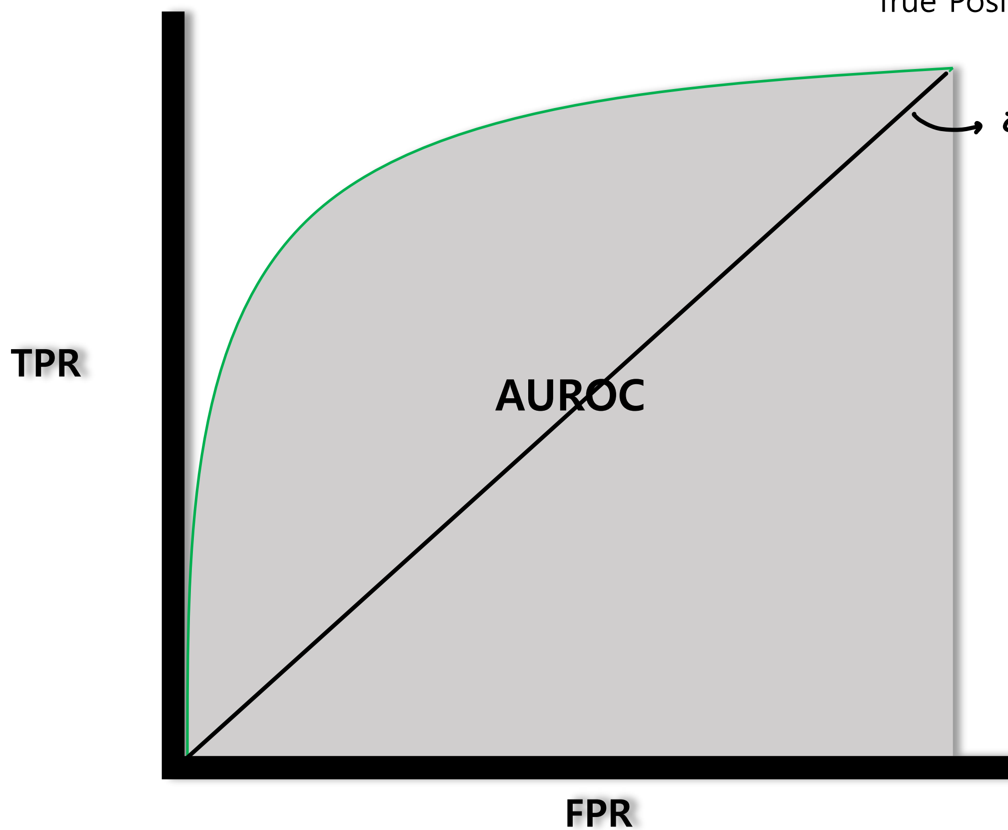
4. Evaluate the result

- Metric – torchmetrics.functional.auroc
  - 0.5 == random

Confusion Matrix		Real	
		Positive	Negative
Predict	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

True Positive Rate

False Positive Rate



랜덤하게 찍음
Positive / Negative 가 반반

Practice – GNN

4. Evaluate the result

- Metric – torchmetrics.functional.auroc

```
torchmetrics.functional. auroc ( preds , target , num_classes = None , pos_label = None ,
```

```
# Evaluation
import torch
from torchmetrics.functional import auroc

outputs = trainer.predict(model, dataloaders=data_module.test_dataloader())
y = torch.concat(outputs)
x = torch.concat([batch.y for batch in data_module.test_dataloader()]).int()

evaluation = auroc(y, x)
print(f"auc-roc: {evaluation}")
```

```
Predicting DataLoader 0: 100% 5/5 [00:00<00:00, 20.68it/s]
auc-roc: 0.4389864206314087
```

References

- https://www.cs.toronto.edu/~kriz/cifar.html
- https://tutorials.pytorch.kr/beginner/blitz/cifar10_tutorial.html
- https://pytorch-lightning.readthedocs.io/en/stable/notebooks/lightning_examples/cifar10-baseline.html
- Golovko, Vladimir, et al. "A shallow convolutional neural network for accurate handwritten digits classification." *International Conference on Pattern Recognition and Information Processing*. Springer, Cham, 2016.
- Veličković, Petar, et al. "Graph attention networks." *arXiv preprint arXiv:1710.10903* (2017).

Thank you