창의융합프로젝트 Project 3
- Baseline example using PyTorch

2021. 05. 14.

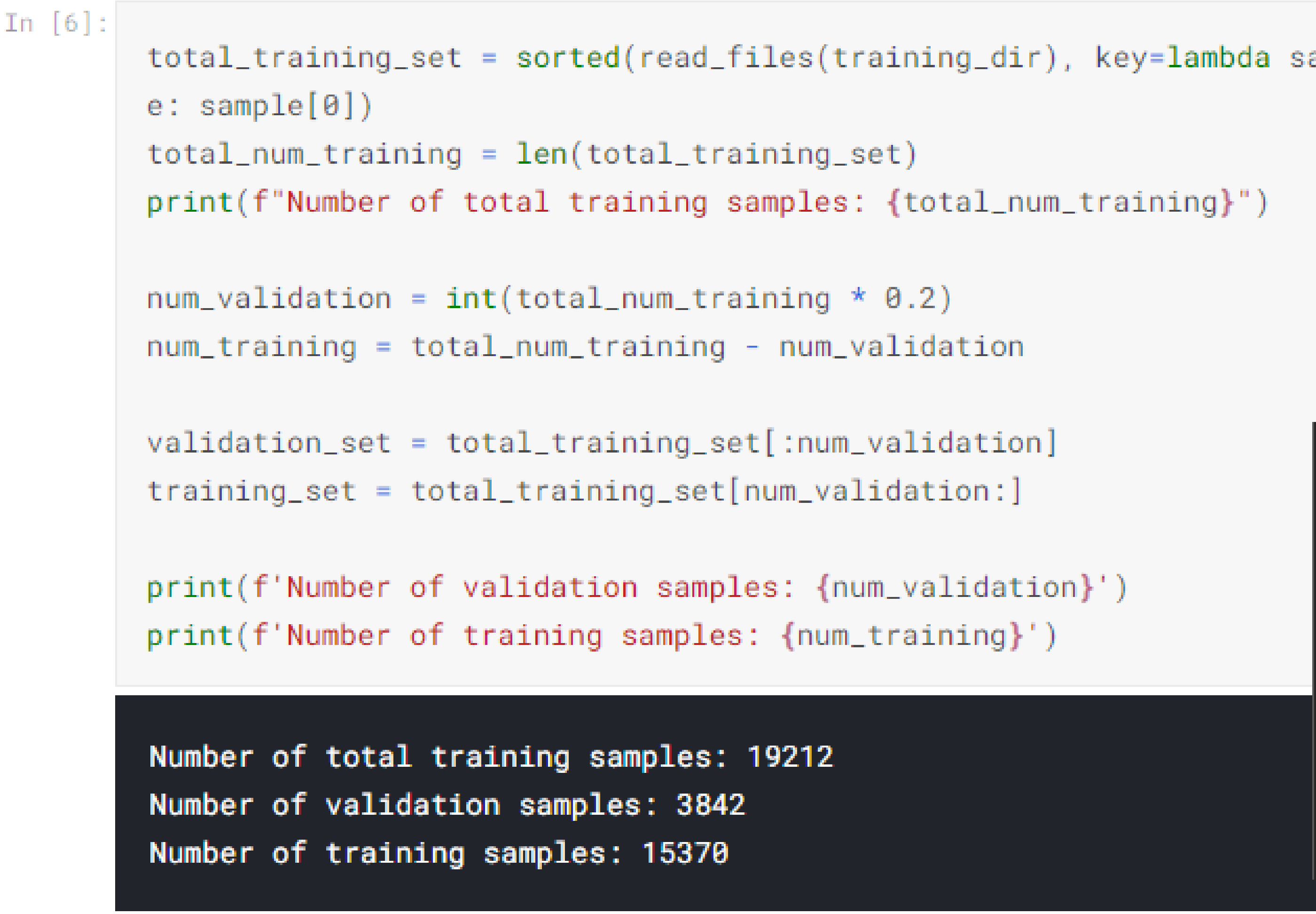
서장원

Seoul National University

Graduate School of Convergence Science and Technology

Applied Data Science Lab.

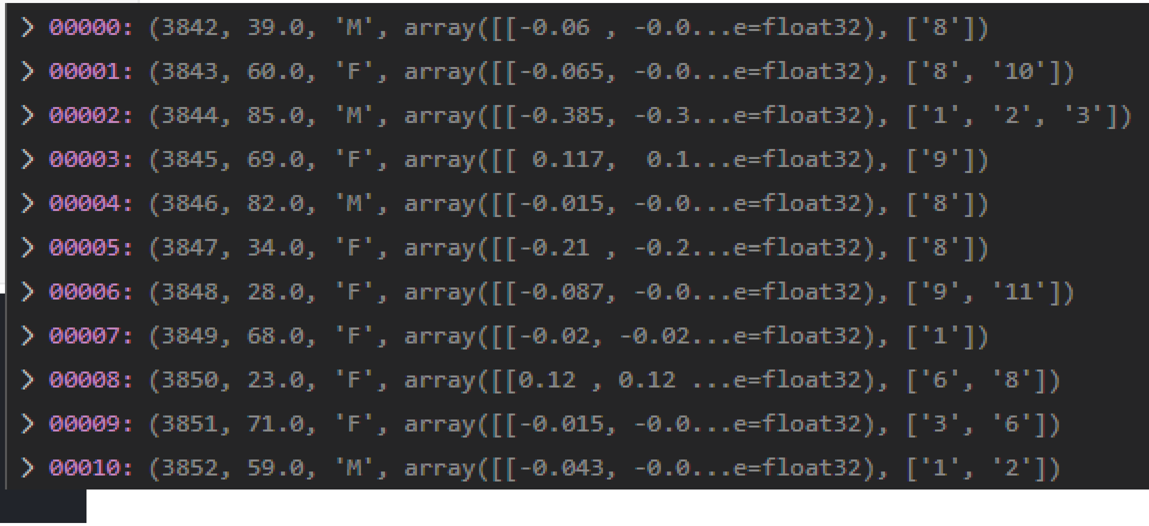
Setup



```
In [6]:
total_training_set = sorted(read_files(training_dir), key=lambda sampl
e: sample[0])
total_num_training = len(total_training_set)
print(f"Number of total training samples: {total_num_training}")

num_validation = int(total_num_training * 0.2)
num_training = total_num_training - num_validation

validation_set = total_training_set[:num_validation]
training_set = total_training_set[num_validation:]

print(f'Number of validation samples: {num_validation}')
print(f'Number of training samples: {num_training}')
```

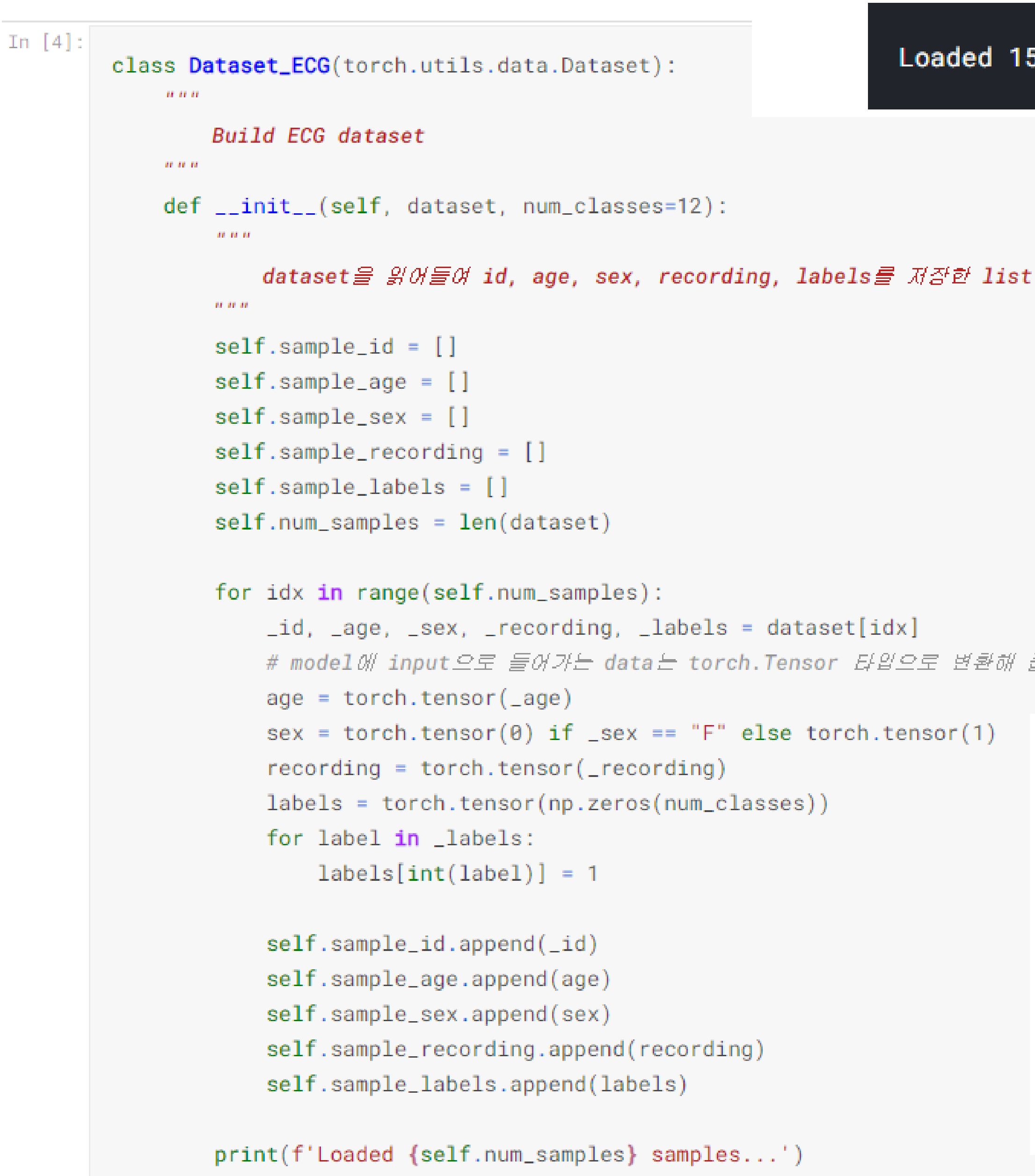
```
Number of total training samples: 19212
Number of validation samples: 3842
Number of training samples: 15370
```

training_set: list of tuples (id, age, sex, recording, labels)



```
> 00000: (3842, 39.0, 'M', array([[-0.06 , -0.0...e=float32), ['8'])
> 00001: (3843, 60.0, 'F', array([[-0.065, -0.0...e=float32), ['8', '10'])
> 00002: (3844, 85.0, 'M', array([[-0.385, -0.3...e=float32), ['1', '2', '3'])
> 00003: (3845, 69.0, 'F', array([[ 0.117,  0.1...e=float32), ['9'])
> 00004: (3846, 82.0, 'M', array([[-0.015, -0.0...e=float32), ['8'])
> 00005: (3847, 34.0, 'F', array([[-0.21 , -0.2...e=float32), ['8'])
> 00006: (3848, 28.0, 'F', array([[-0.087, -0.0...e=float32), ['9', '11'])
> 00007: (3849, 68.0, 'F', array([[-0.02, -0.02...e=float32), ['1'])
> 00008: (3850, 23.0, 'F', array([[0.12 , 0.12 ...e=float32), ['6', '8'])
> 00009: (3851, 71.0, 'F', array([[-0.015, -0.0...e=float32), ['3', '6'])
> 00010: (3852, 59.0, 'M', array([[-0.043, -0.0...e=float32), ['1', '2'])
```

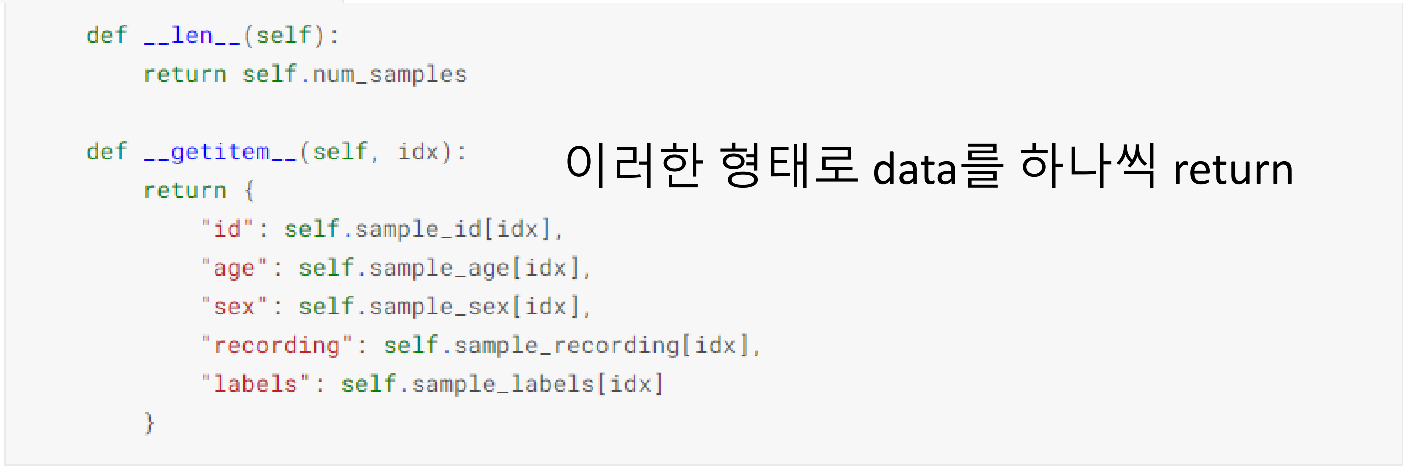
Dataset



In [4]:

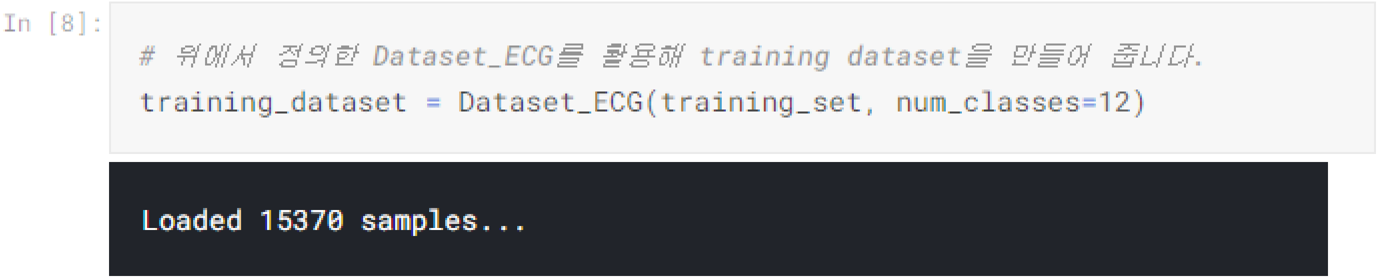
```python
class Dataset_ECG(torch.utils.data.Dataset):
    """
        Build ECG dataset
    """
    def __init__(self, dataset, num_classes=12):
        """
            dataset을 읽어들여 id, age, sex, recording, labels를 저장한 list를 만들어 줍니다.
        """
        self.sample_id = []
        self.sample_age = []
        self.sample_sex = []
        self.sample_recording = []
        self.sample_labels = []
        self.num_samples = len(dataset)

        for idx in range(self.num_samples):
            _id, _age, _sex, _recording, _labels = dataset[idx]
            # model에 input으로 들어가는 data는 torch.Tensor 타입으로 변환해 줍니다.
            age = torch.tensor(_age)
            sex = torch.tensor(0) if _sex == "F" else torch.tensor(1)
            recording = torch.tensor(_recording)
            labels = torch.tensor(np.zeros(num_classes))
            for label in _labels:
                labels[int(label)] = 1

            self.sample_id.append(_id)
            self.sample_age.append(age)
            self.sample_sex.append(sex)
            self.sample_recording.append(recording)
            self.sample_labels.append(labels)

        print(f'Loaded {self.num_samples} samples...')
```

PyTorch custom dataset



```python
    def __len__(self):
        return self.num_samples

    def __getitem__(self, idx):
        return {
            "id": self.sample_id[idx],
            "age": self.sample_age[idx],
            "sex": self.sample_sex[idx],
            "recording": self.sample_recording[idx],
            "labels": self.sample_labels[idx]
        }
```

이러한 형태로 data를 하나씩 return

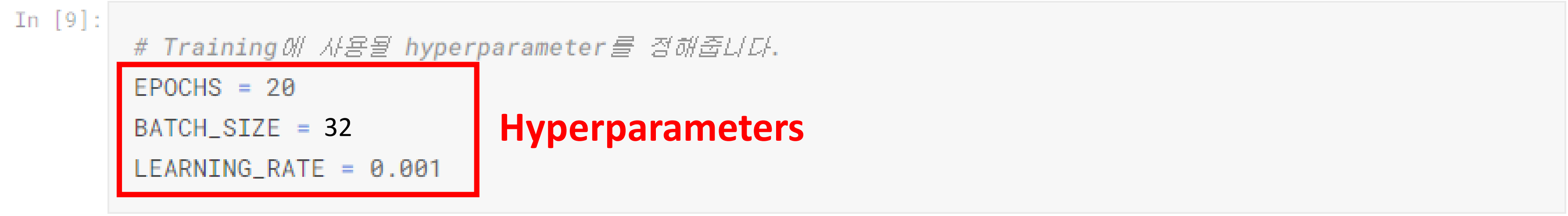


In [8]:

```python
# 위에서 정의한 Dataset_ECG를 활용해 training dataset을 만들어 줍니다.
training_dataset = Dataset_ECG(training_set, num_classes=12)
```

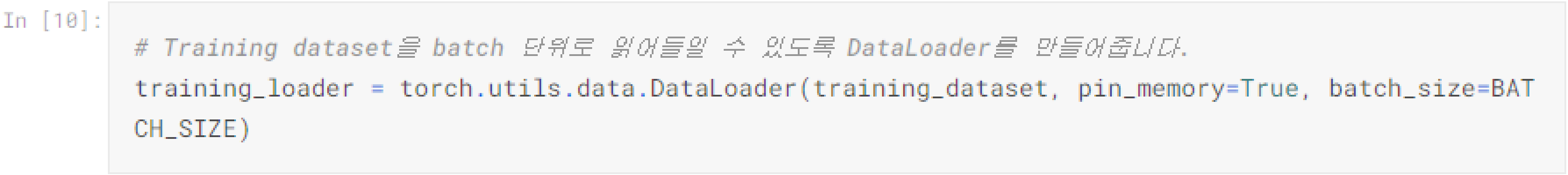
```
Loaded 15370 samples...
```

Data loader



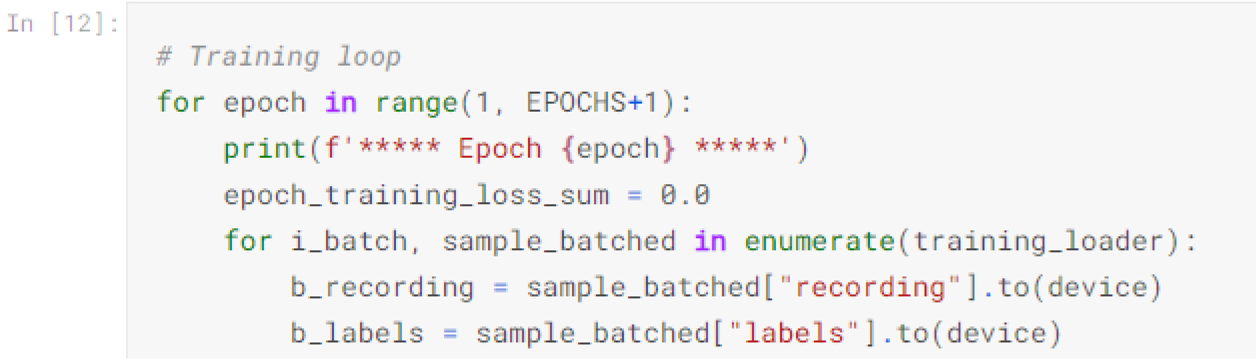
```
In [9]:
# Training에 사용될 hyperparameter를 정해줍니다.
EPOCHS = 20
BATCH_SIZE = 32
LEARNING_RATE = 0.001
```

Hyperparameters



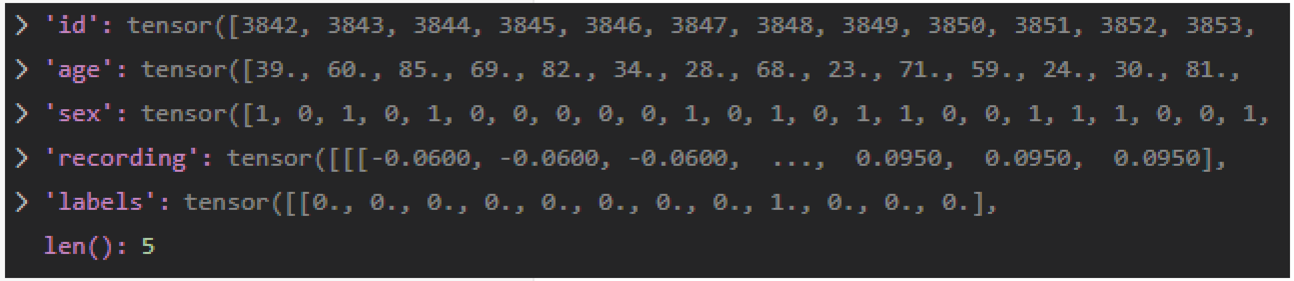
```
In [10]:
# Training dataset을 batch 단위로 읽어들일 수 있도록 DataLoader를 만들어줍니다.
training_loader = torch.utils.data.DataLoader(training_dataset, pin_memory=True, batch_size=BAT
CH_SIZE)
```

training_loader는 batch_size 만큼의 samples을 한꺼번에 return



```
In [12]:
# Training loop
for epoch in range(1, EPOCHS+1):
    print(f'***** Epoch {epoch} *****')
    epoch_training_loss_sum = 0.0
    for i_batch, sample_batched in enumerate(training_loader):
        b_recording = sample_batched["recording"].to(device)
        b_labels = sample_batched["labels"].to(device)
```

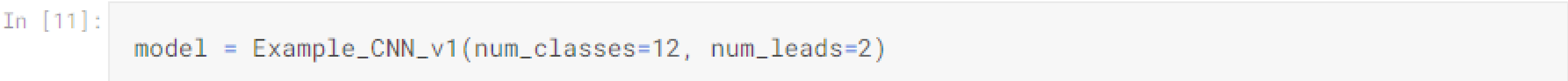
sample_batched: dictionary



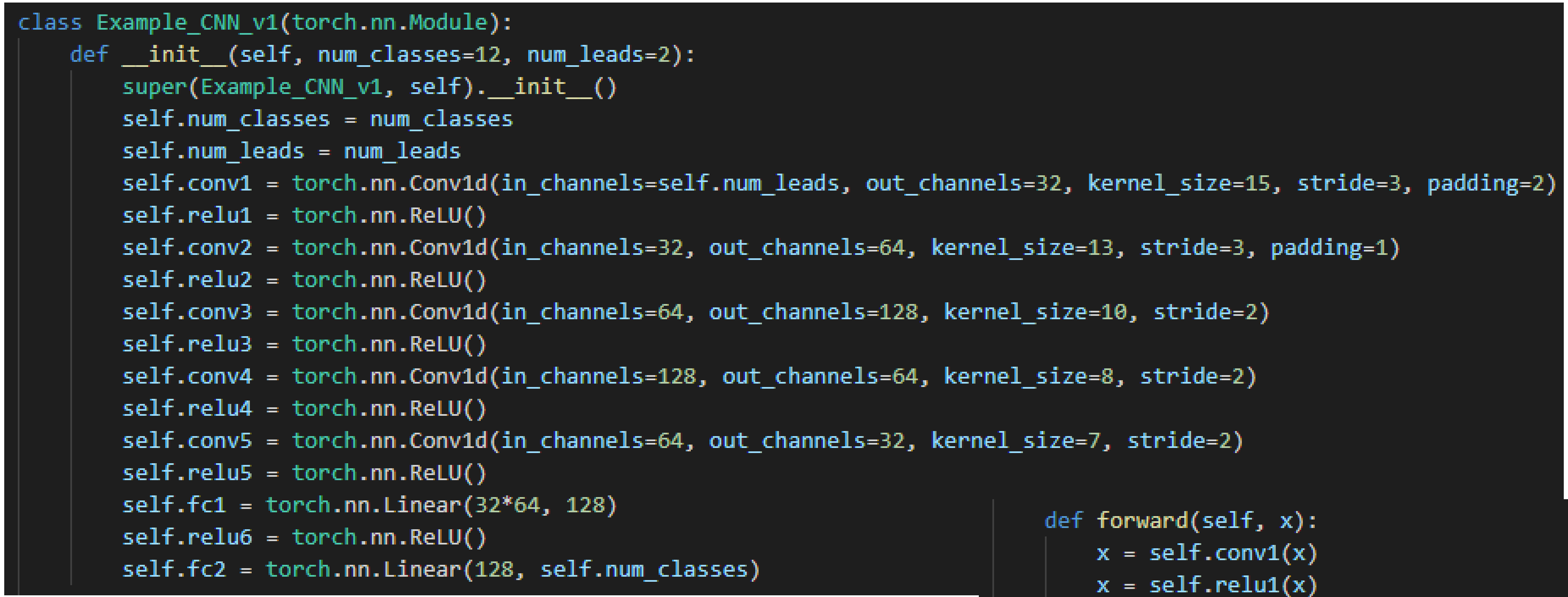
```
> 'id': tensor([3842, 3843, 3844, 3845, 3846, 3847, 3848, 3849, 3850, 3851, 3852, 3853,
> 'age': tensor([39., 60., 85., 69., 82., 34., 28., 68., 23., 71., 59., 24., 30., 81.,
> 'sex': tensor([1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1,
> 'recording': tensor([[[-0.0600, -0.0600, -0.0600,  ...,  0.0950,  0.0950,  0.0950],
> 'labels': tensor([[0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
  len(): 5
```

sample_batched[‘recording’].shape: (32,2,5000)
sample_batched[‘labels’].shape: (32, 12)

Model

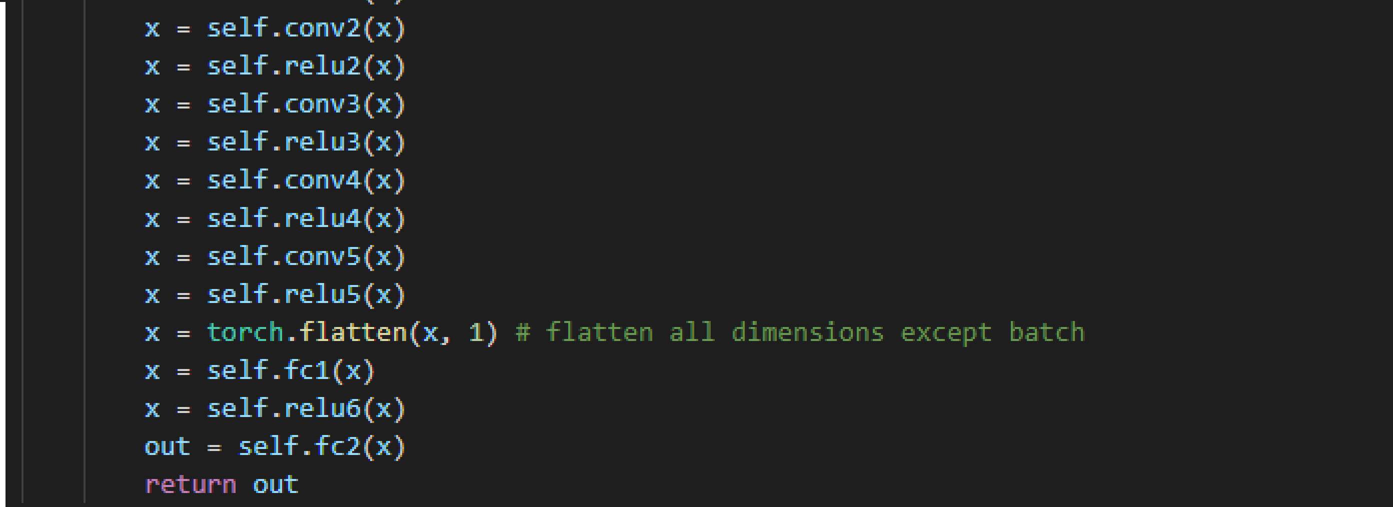


```
In [11]: model = Example_CNN_v1(num_classes=12, num_leads=2)
```



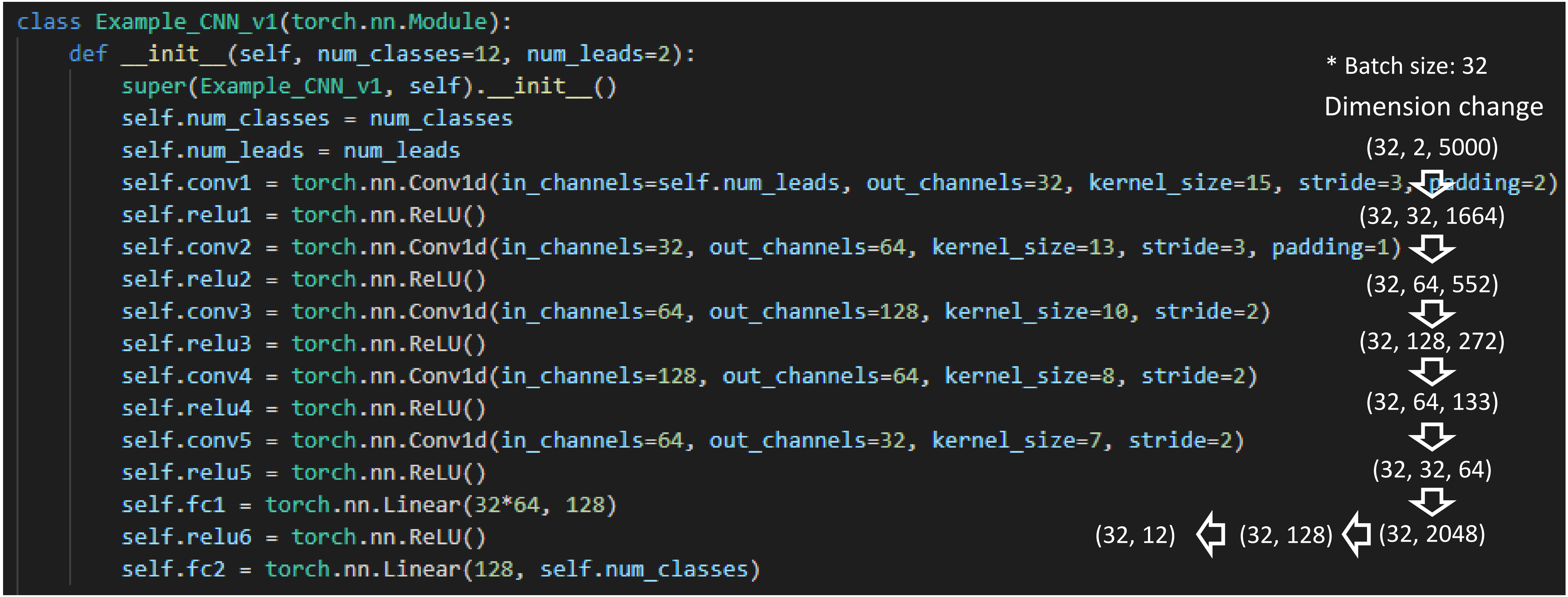
```
class Example_CNN_v1(torch.nn.Module):
    def __init__(self, num_classes=12, num_leads=2):
        super(Example_CNN_v1, self).__init__()
        self.num_classes = num_classes
        self.num_leads = num_leads
        self.conv1 = torch.nn.Conv1d(in_channels=self.num_leads, out_channels=32, kernel_size=15, stride=3, padding=2)
        self.relu1 = torch.nn.ReLU()
        self.conv2 = torch.nn.Conv1d(in_channels=32, out_channels=64, kernel_size=13, stride=3, padding=1)
        self.relu2 = torch.nn.ReLU()
        self.conv3 = torch.nn.Conv1d(in_channels=64, out_channels=128, kernel_size=10, stride=2)
        self.relu3 = torch.nn.ReLU()
        self.conv4 = torch.nn.Conv1d(in_channels=128, out_channels=64, kernel_size=8, stride=2)
        self.relu4 = torch.nn.ReLU()
        self.conv5 = torch.nn.Conv1d(in_channels=64, out_channels=32, kernel_size=7, stride=2)
        self.relu5 = torch.nn.ReLU()
        self.fc1 = torch.nn.Linear(32*64, 128)
        self.relu6 = torch.nn.ReLU()
        self.fc2 = torch.nn.Linear(128, self.num_classes)
```

Architecture design



```
    def forward(self, x):
        x = self.conv1(x)
        x = self.relu1(x)
        x = self.conv2(x)
        x = self.relu2(x)
        x = self.conv3(x)
        x = self.relu3(x)
        x = self.conv4(x)
        x = self.relu4(x)
        x = self.conv5(x)
        x = self.relu5(x)
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = self.fc1(x)
        x = self.relu6(x)
        out = self.fc2(x)
        return out
```

Model



```
class Example_CNN_v1(torch.nn.Module):
    def __init__(self, num_classes=12, num_leads=2):
        super(Example_CNN_v1, self).__init__()
        self.num_classes = num_classes
        self.num_leads = num_leads
        self.conv1 = torch.nn.Conv1d(in_channels=self.num_leads, out_channels=32, kernel_size=15, stride=3, padding=2)
        self.relu1 = torch.nn.ReLU()
        self.conv2 = torch.nn.Conv1d(in_channels=32, out_channels=64, kernel_size=13, stride=3, padding=1)
        self.relu2 = torch.nn.ReLU()
        self.conv3 = torch.nn.Conv1d(in_channels=64, out_channels=128, kernel_size=10, stride=2)
        self.relu3 = torch.nn.ReLU()
        self.conv4 = torch.nn.Conv1d(in_channels=128, out_channels=64, kernel_size=8, stride=2)
        self.relu4 = torch.nn.ReLU()
        self.conv5 = torch.nn.Conv1d(in_channels=64, out_channels=32, kernel_size=7, stride=2)
        self.relu5 = torch.nn.ReLU()
        self.fc1 = torch.nn.Linear(32*64, 128)
        self.relu6 = torch.nn.ReLU()
        self.fc2 = torch.nn.Linear(128, self.num_classes)
```

* Batch size: 32

Dimension change

(32, 2, 5000)

(32, 32, 1664)

(32, 64, 552)

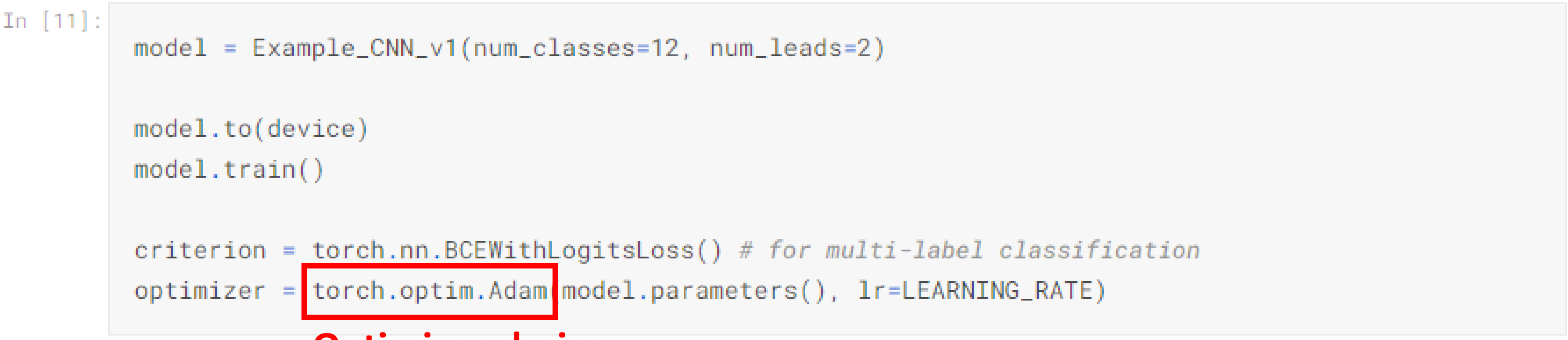
(32, 128, 272)

(32, 64, 133)

(32, 32, 64)

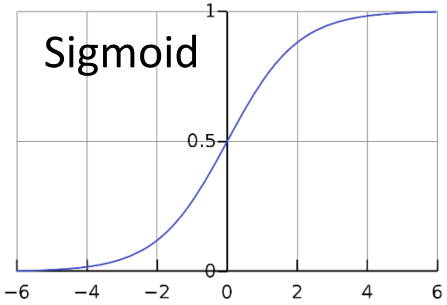
(32, 12) ⇦ (32, 128) ⇦ (32, 2048)

Training



```
In [11]:  model = Example_CNN_v1(num_classes=12, num_leads=2)

          model.to(device)
          model.train()

          criterion = torch.nn.BCEWithLogitsLoss() # for multi-label classification
          optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)
```

Optimizer choice



* BCEWithLogitsLoss

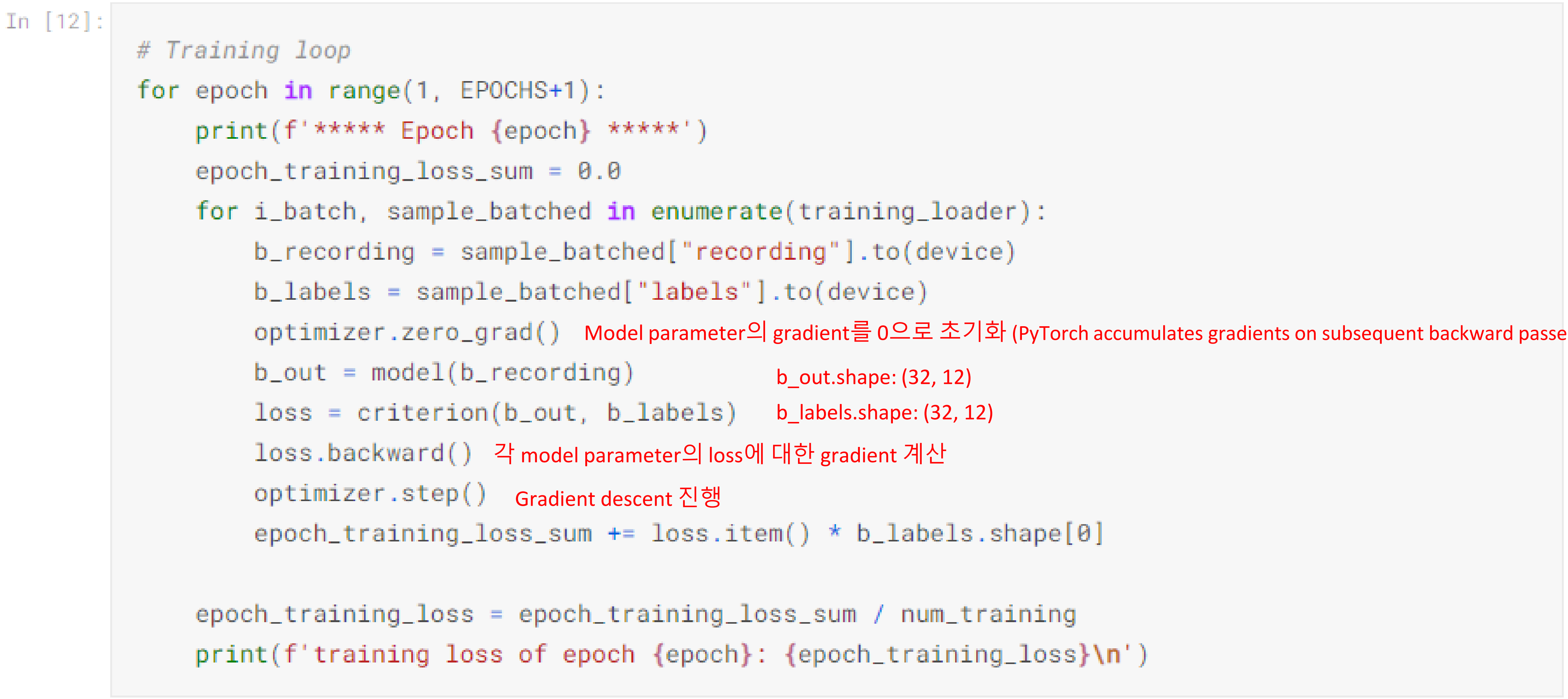
Predicted: 0.1 5.1 -1.7 -2.4 0.2 3.8 …. -1.4

True label: 0 1 0 0 0 1 …. 0

Loss

Sigmoid
+
Binary cross entropy
&
mean

Training



In [12]:

```
# Training loop
for epoch in range(1, EPOCHS+1):
    print(f'***** Epoch {epoch} *****')
    epoch_training_loss_sum = 0.0
    for i_batch, sample_batched in enumerate(training_loader):
        b_recording = sample_batched["recording"].to(device)
        b_labels = sample_batched["labels"].to(device)
        optimizer.zero_grad()
        b_out = model(b_recording)
        loss = criterion(b_out, b_labels)
        loss.backward()
        optimizer.step()
        epoch_training_loss_sum += loss.item() * b_labels.shape[0]

    epoch_training_loss = epoch_training_loss_sum / num_training
    print(f'training loss of epoch {epoch}: {epoch_training_loss}\n')
```

Model parameter의 gradient를 0으로 초기화 (PyTorch accumulates gradients on subsequent backward passes)

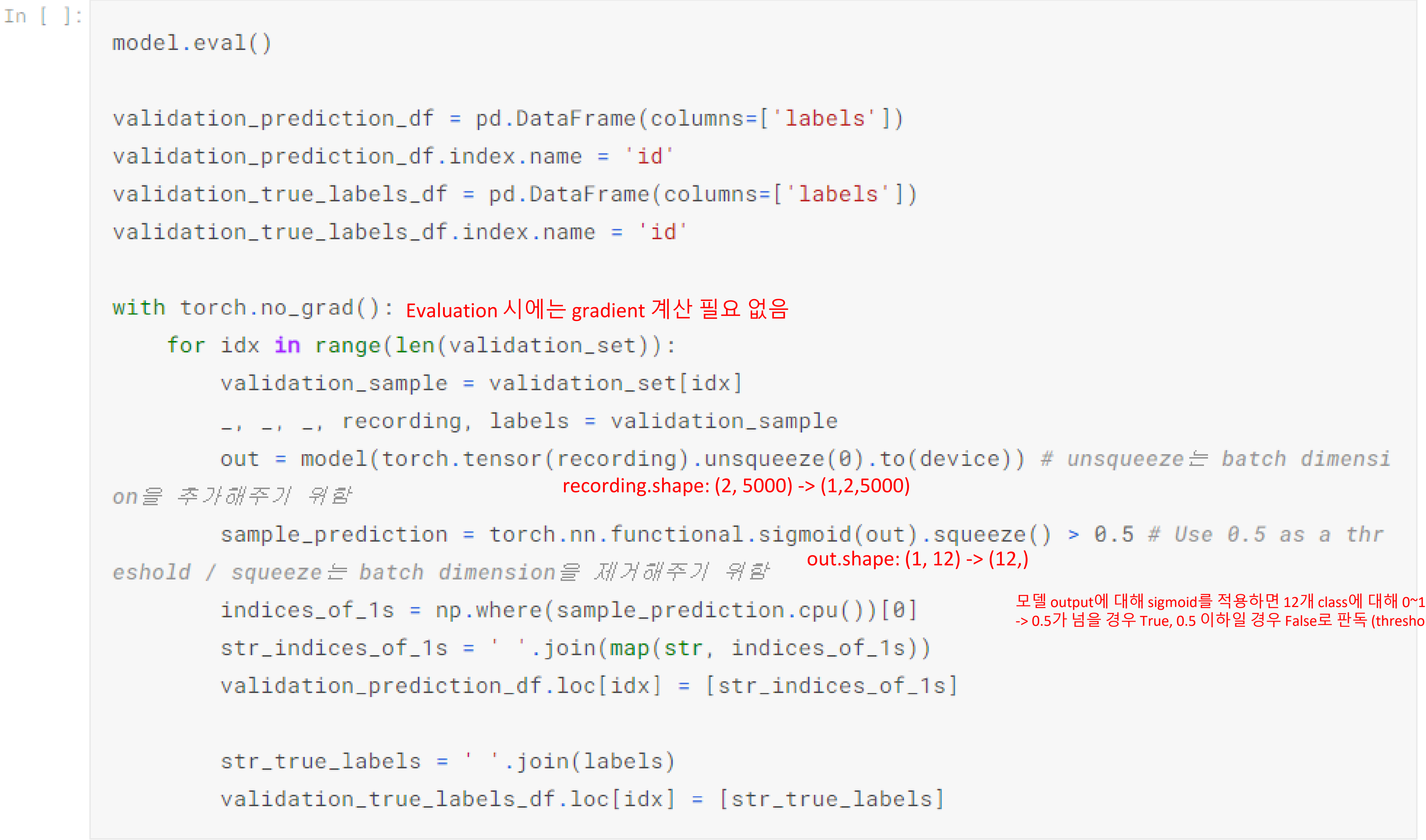
b_out.shape: (32, 12)

b_labels.shape: (32, 12)

각 model parameter의 loss에 대한 gradient 계산

Gradient descent 진행

Validation



```
In [ ]:
model.eval()

validation_prediction_df = pd.DataFrame(columns=['labels'])
validation_prediction_df.index.name = 'id'
validation_true_labels_df = pd.DataFrame(columns=['labels'])
validation_true_labels_df.index.name = 'id'

with torch.no_grad():
    for idx in range(len(validation_set)):
        validation_sample = validation_set[idx]
        _, _, _, recording, labels = validation_sample
        out = model(torch.tensor(recording).unsqueeze(0).to(device)) # unsqueeze는 batch dimensi
on을 추가해주기 위함
        sample_prediction = torch.nn.functional.sigmoid(out).squeeze() > 0.5 # Use 0.5 as a thr
eshold / squeeze는 batch dimension을 제거해주기 위함
        indices_of_1s = np.where(sample_prediction.cpu())[0]
        str_indices_of_1s = ' '.join(map(str, indices_of_1s))
        validation_prediction_df.loc[idx] = [str_indices_of_1s]

        str_true_labels = ' '.join(labels)
        validation_true_labels_df.loc[idx] = [str_true_labels]
```

Evaluation 시에는 gradient 계산 필요 없음

recording.shape: (2, 5000) -> (1,2,5000)

out.shape: (1, 12) -> (12,)

모델 output에 대해 sigmoid를 적용하면 12개 class에 대해 0~1의 확률값을 얻음
-> 0.5가 넘을 경우 True, 0.5 이하일 경우 False로 판독 (threshold를 바꿔가며 성능 평가 가능)

Thank you!