# Lab 3: Hyperparameter Tuning

11210IPT 553000

Deep Learning in Biomedical Optical Imaging

2023/10/02

# Outlines

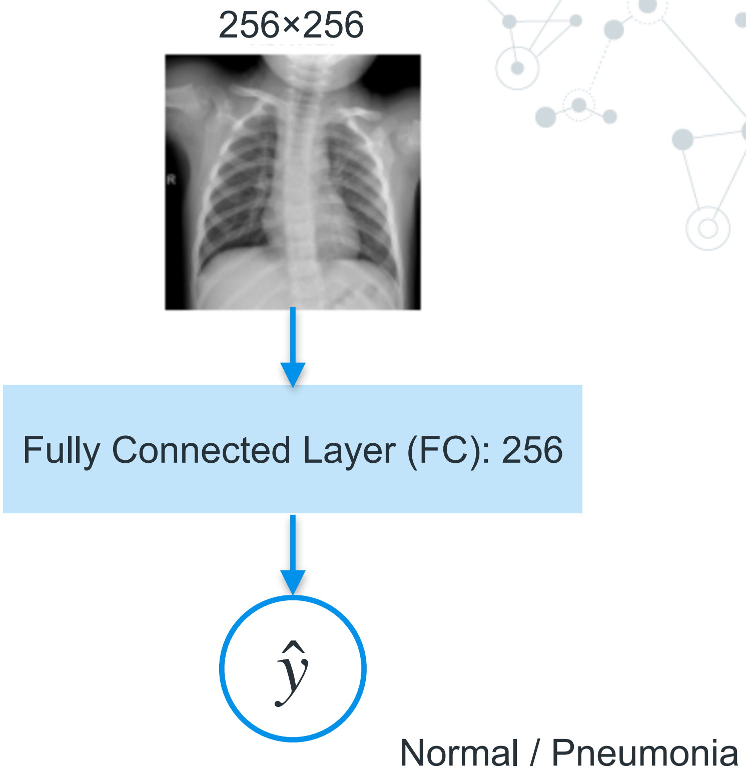
- Hyperparameters
- Learning Rate Scheduler
- Homework 2

# Hyperparameters - Model

## B. Defining Neural Networks in PyTorch

```python
import torch.nn as nn

# Model definition
model = nn.Sequential(
    nn.Flatten(),
    nn.Linear(256*256*1, 256),
    nn.ReLU(),
    nn.Linear(256, 1)
).cuda()

print(model)
```

```
Sequential(
  (0): Flatten(start_dim=1, end_dim=-1)
  (1): Linear(in_features=65536, out_features=256, bias=True)
  (2): ReLU()
  (3): Linear(in_features=256, out_features=1, bias=True)
)
```



# Hyperparameters

Number of Units/Neurons

Fully Connected Layer (FC): 32

$\hat{y}$

Number of Layers

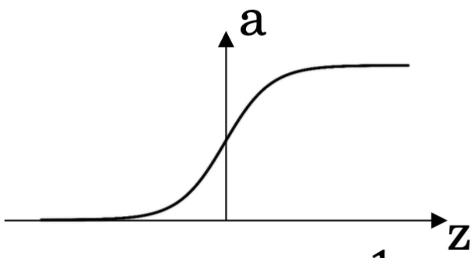
Fully Connected Layer (FC): 256

Fully Connected Layer (FC): 256

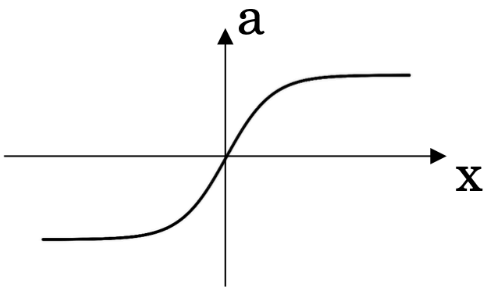
Fully Connected Layer (FC): 256

# Hyperparameters

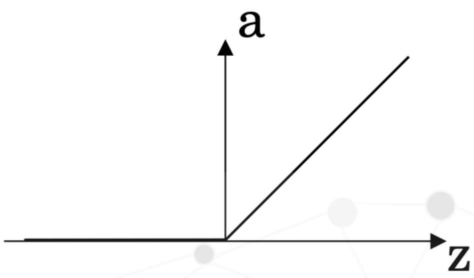
Activation Function



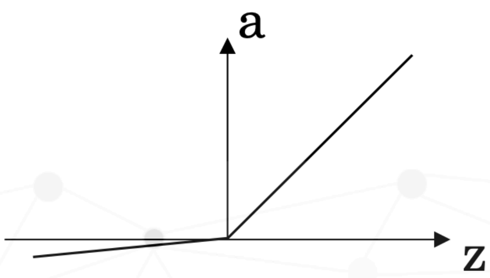

$$\text{sigmoid:}\quad a = \frac{1}{1+e^{-z}}$$




$$tanh: a = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$




$$Relu: a = \max(0, z)$$




$$Leaky\ Relu: a = \max(0.01z, z)$$

# Hyperparameters

- Dropout
- Batch Normalization
- Regularization (L1, L2)
- ……

Various layers you can add !

## TORCH.NN

These are the basic building blocks for graphs:

torch.nn

- Containers
- Convolution Layers
- Pooling layers
- Padding Layers
- Non-linear Activations (weighted sum, nonlinearity)
- Non-linear Activations (other)
- Normalization Layers
- Recurrent Layers
- Transformer Layers
- Linear Layers
- Dropout Layers
- Sparse Layers
- Distance Functions
- Loss Functions
- Vision Layers
- Shuffle Layers
- DataParallel Layers (multi-GPU, distributed)
- Utilities
- Quantized Functions
- Lazy Modules Initialization

torch.nn

# Hyperparameters

## A. Data Loading and Preprocessing

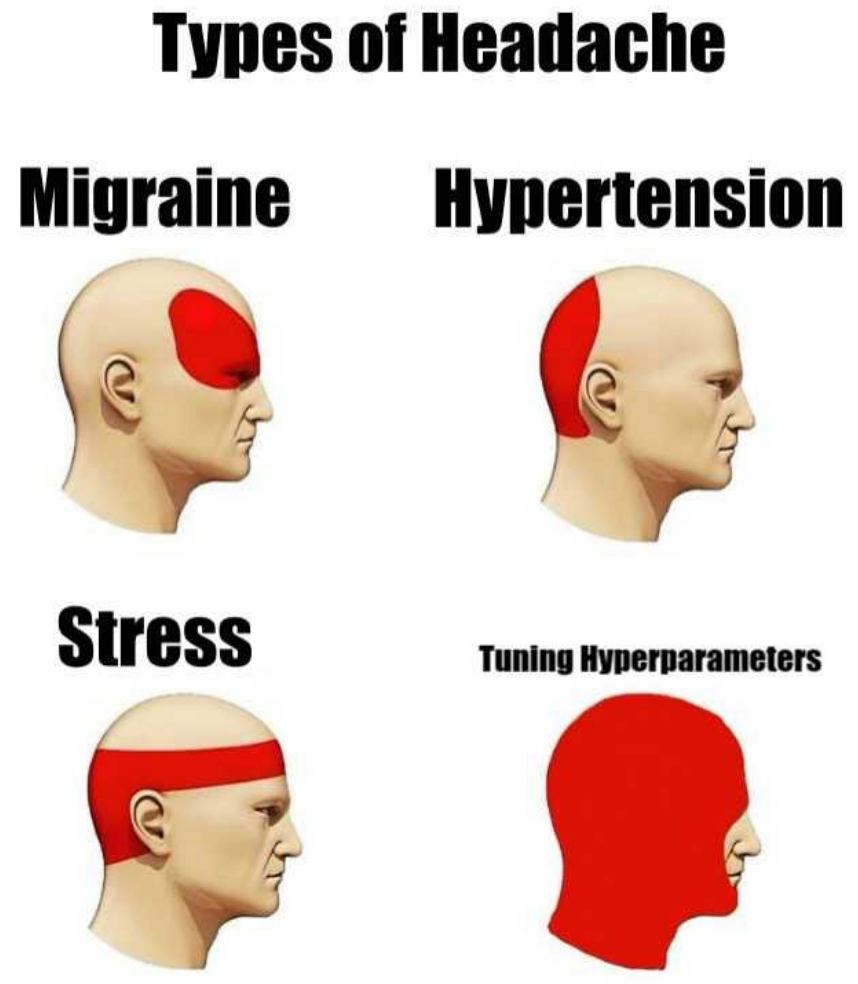
```
# Create dataloaders
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
```

- Batch size

## C. Training the Neural Network

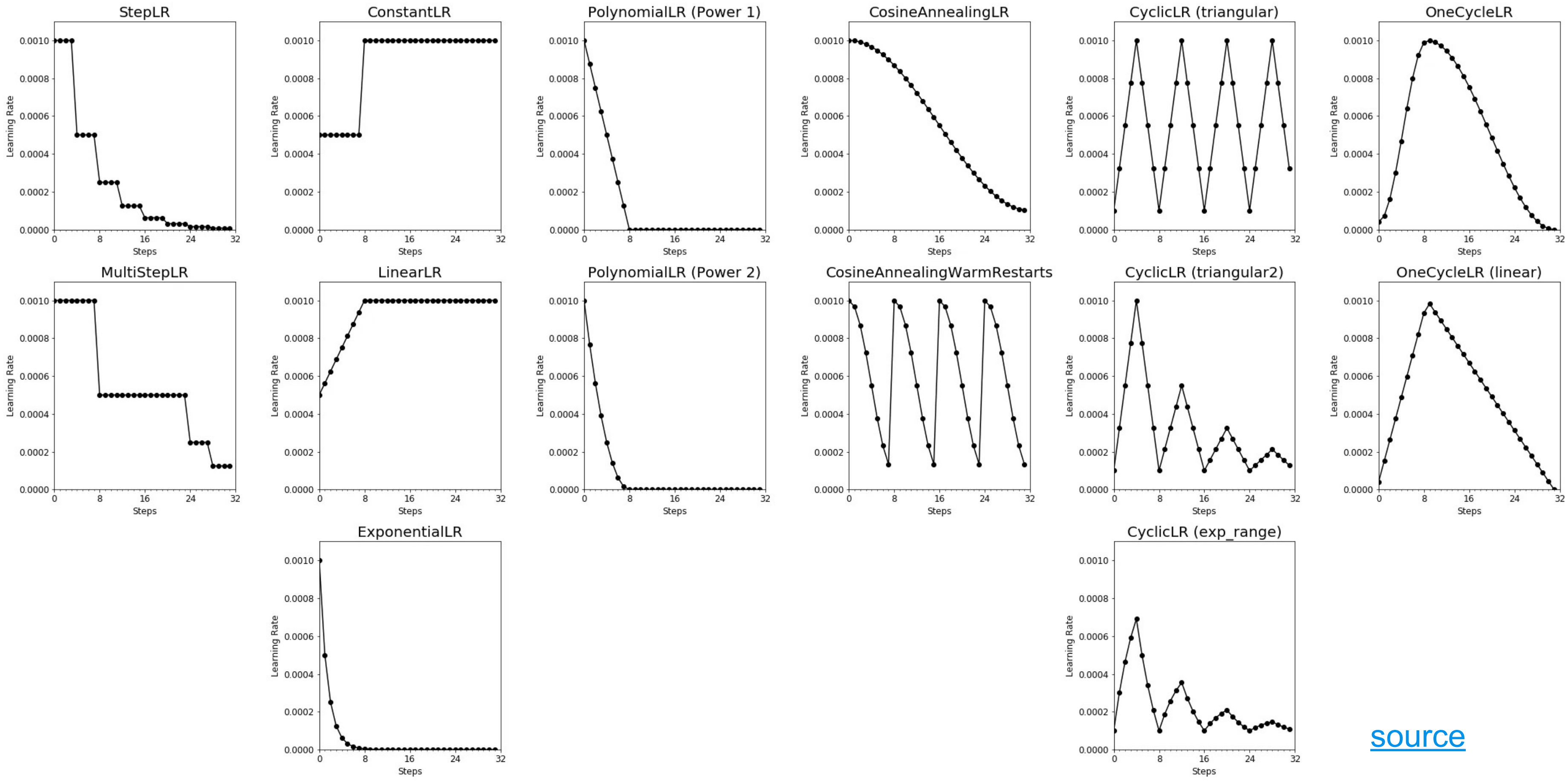
```
import torch.optim as optim
from torch.optim.lr_scheduler import CosineAnnealingLR

train_losses = []
val_losses = []
train_accuracies = []
val_accuracies = []

epochs = 30
best_val_loss = float('inf')

# Criterion and Optimizer
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)
lr_scheduler = CosineAnnealingLR(optimizer, T_max=len(train_loader)*epochs, eta_min=0)
```

- Number of epochs
- Loss function
- Optimizer
- Learning rate
- Learning rate scheduler


Types of Headache
Migraine
Hypertension
Stress
Tuning Hyperparameters

# Learning Rate Scheduler

- Dynamically adjusts the learning rate during the training process. It helps to optimize training and sometimes escape local minima.
- Critical role in training machine learning models effectively for several reasons:
  - Convergence speed
  - Training stability
  - Model Performance
  - …


StepLR
ConstantLR
PolynomialLR (Power 1)
CosineAnnealingLR
CyclicLR (triangular)
OneCycleLR
MultiStepLR
LinearLR
PolynomialLR (Power 2)
CosineAnnealingWarmRestarts
CyclicLR (triangular2)
OneCycleLR (linear)
ExponentialLR
CyclicLR (exp_range)
Learning Rate
Steps
0.0000
0.0002
0.0004
0.0006
0.0008
0.0010
0
8
16
24
32

source

```python
from torch.optim.lr_scheduler import CosineAnnealingLR
lr_scheduler = CosineAnnealingLR(optimizer, T_max=len(train_loader)*epochs, eta_min=0)
```

Docs > torch.optim

| | |
|---|---|
| `lr_scheduler.PolynomialLR` | Decays the learning rate of each parameter group using a polynomial function in the given total_iters. |
| `lr_scheduler.CosineAnnealingLR` | Set the learning rate of each parameter group using a cosine annealing schedule, where $\eta_{max}$ is set to the initial lr and $T_{cur}$ is the number of epochs since the last restart in SGDR: |
| `lr_scheduler.ChainedScheduler` | Chains list of learning rate schedulers. |
| `lr_scheduler.SequentialLR` | Receives the list of schedulers that is expected to be called sequentially during optimization process and milestone points that provides exact intervals to reflect which scheduler is supposed to be called at a given epoch. |
| `lr_scheduler.ReduceLROnPlateau` | Reduce learning rate when a metric has stopped improving. |
| `lr_scheduler.CyclicLR` | Sets the learning rate of each parameter group according to cyclical learning rate policy (CLR). |
| `lr_scheduler.OneCycleLR` | Sets the learning rate of each parameter group according to the 1cycle learning rate policy. |

https://pytorch.org/docs/stable/optim.html

# COSINEANNEALINGLR

CLASS torch.optim.lr_scheduler.CosineAnnealingLR(*optimizer*, *T_max*, *eta_min=0*, *last_epoch=- 1*, *verbose=False*) [SOURCE]

Set the learning rate of each parameter group using a cosine annealing schedule, where $\eta_{max}$ is set to the initial lr and $T_{cur}$ is the number of epochs since the last restart in SGDR:

$$
\begin{aligned}
\eta_t &= \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})\left(1 + \cos\left(\frac{T_{cur}}{T_{max}}\pi\right)\right), & T_{cur} \neq (2k+1)T_{max}; \\
\eta_{t+1} &= \eta_t + \frac{1}{2}(\eta_{max} - \eta_{min})\left(1 - \cos\left(\frac{1}{T_{max}}\pi\right)\right), & T_{cur} = (2k+1)T_{max}.
\end{aligned}
$$

When last_epoch=-1, sets initial lr as lr. Notice that because the schedule is defined recursively, the learning rate can be simultaneously modified outside this scheduler by other operators. If the learning rate is set solely by this scheduler, the learning rate at each step becomes:

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})\left(1 + \cos\left(\frac{T_{cur}}{T_{max}}\pi\right)\right)$$

It has been proposed in SGDR: Stochastic Gradient Descent with Warm Restarts. Note that this only implements the cosine annealing part of SGDR, and not the restarts.

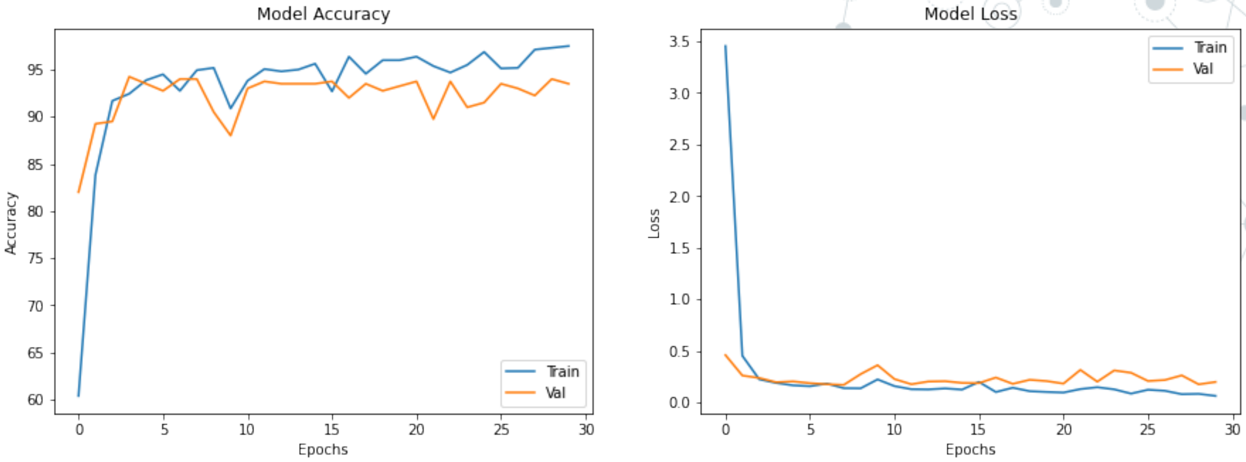
**Parameters:**

- **optimizer** (*Optimizer*) – Wrapped optimizer.
- **T_max** (*int*) – Maximum number of iterations.
- **eta_min** (*float*) – Minimum learning rate. Default: 0.
- **last_epoch** (*int*) – The index of last epoch. Default: -1.
- **verbose** (*bool*) – If `True`, prints a message to stdout for each update. Default: `False`.

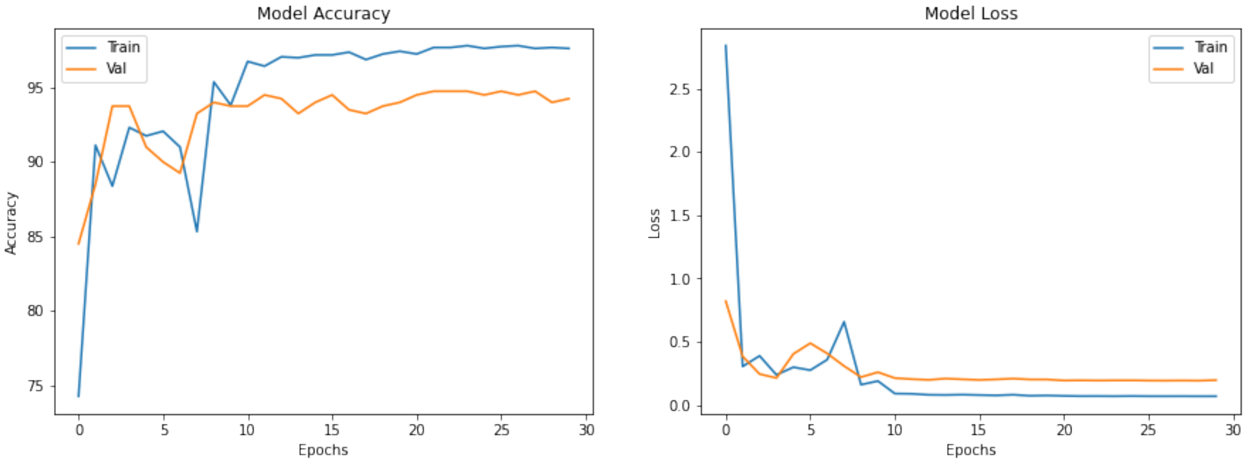
CosineAnnealingLR

```python
model = nn.Sequential(
    nn.Flatten(),
    nn.Linear(256*256*1, 256),
    nn.ReLU(),
    nn.Linear(256, 1)
).cuda()
```

## CosineAnnealingLR

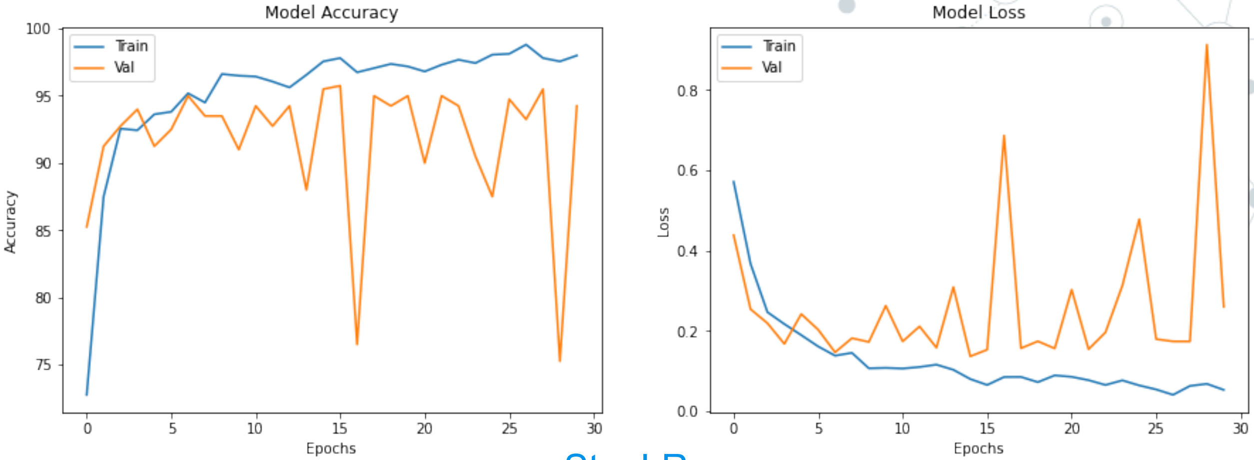



## StepLR

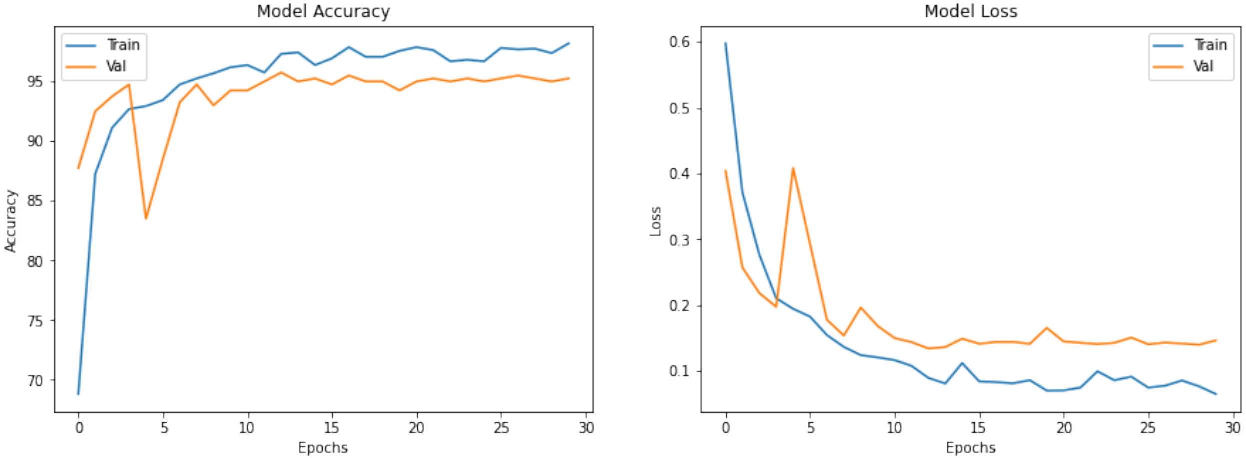


```python
model = nn.Sequential(
    nn.Flatten(),

    nn.Linear(256*256*1, 64),
    nn.BatchNorm1d(64),
    nn.ReLU(),
    nn.Dropout(0.5),

    nn.Linear(64, 64),
    nn.BatchNorm1d(64),
    nn.ReLU(),
    nn.Dropout(0.5),

    nn.Linear(64, 64),
    nn.BatchNorm1d(64),
    nn.ReLU(),
    nn.Dropout(0.5),

    nn.Linear(64, 1)
).cuda()

print(model)
```

## CosineAnnealingLR



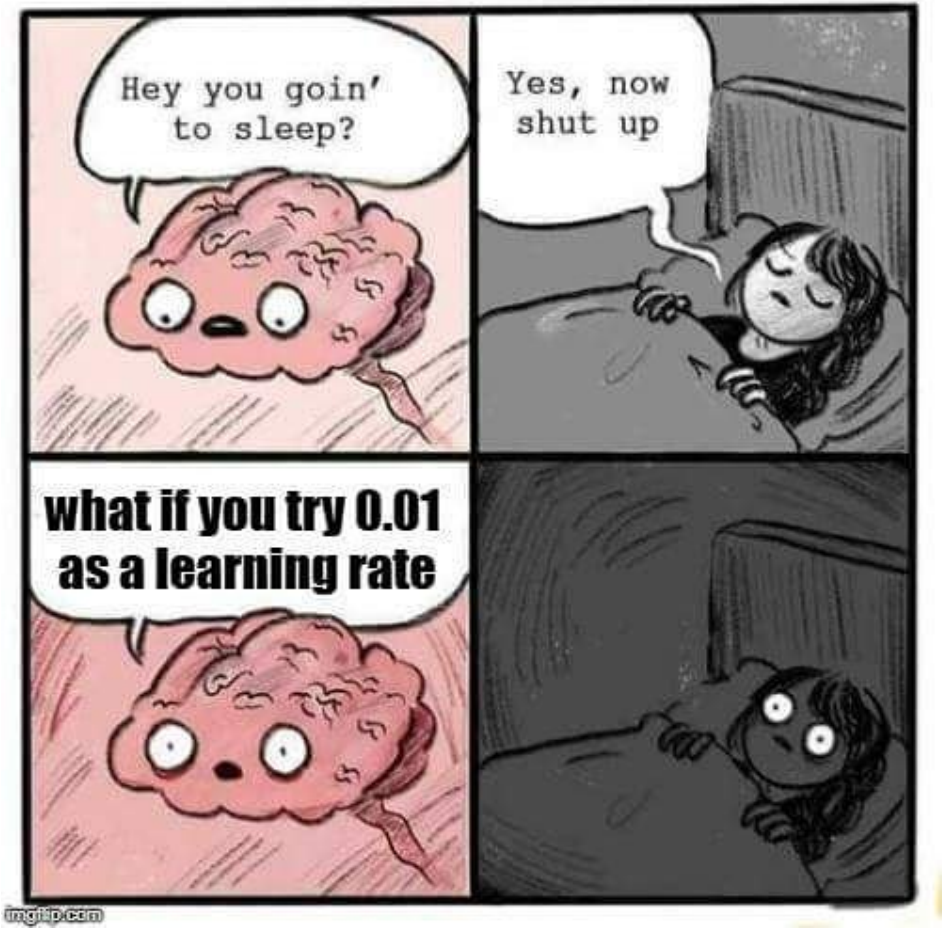

## StepLR



# Homework 2

- **Deadline**: 23:59, 16th Oct. (GMT+8)
- We have 2 parts, the code and the report. Details are in hw2_description.pdf
- 🚨 **Important**: Make sure your commit is timestamped before the deadline. Late submissions might not be graded or could incur a penalty. Only the GitHub link is required on NTHU EEclass.

# CODING TIME!!


Hey you goin' to sleep?
Yes, now shut up
what if you try 0.01 as a learning rate
imgflip.com