Practices in Visual Computing II
Spring 2023

Lab #3: Best Practices

Subjects

1. Advanced PyTorch
2. Python Best Practices
3. Neural Style Transfer

Datasets

- A collection of images and labels
- A collection of pairs of images
- A collection of sentences
- ...

Dataloaders

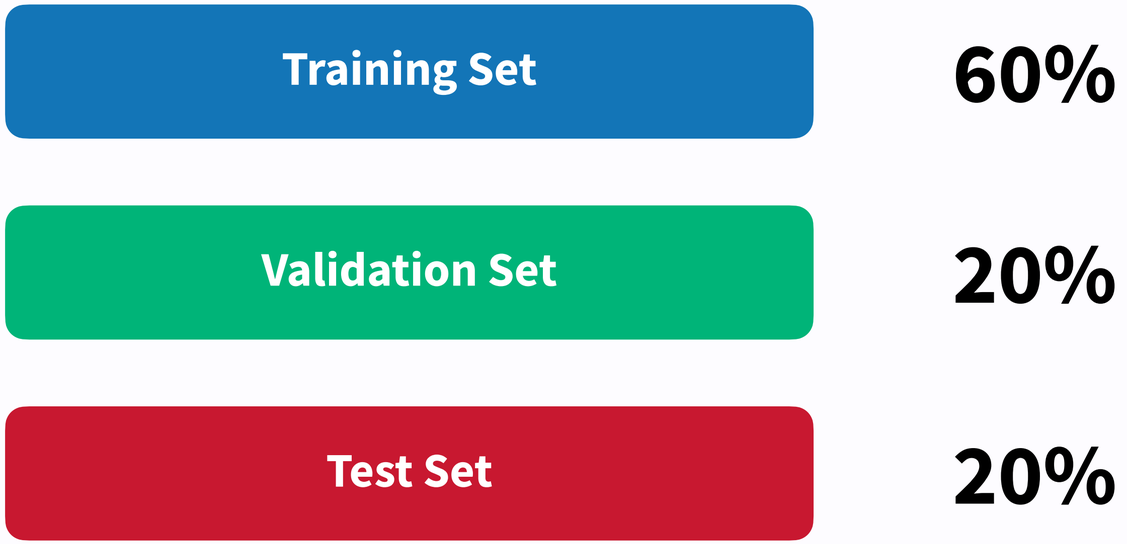
- Help fetching and preprocessing batches of data during our training
- Should implement `__len__` and `__getitem__`

PyTorch Dataset

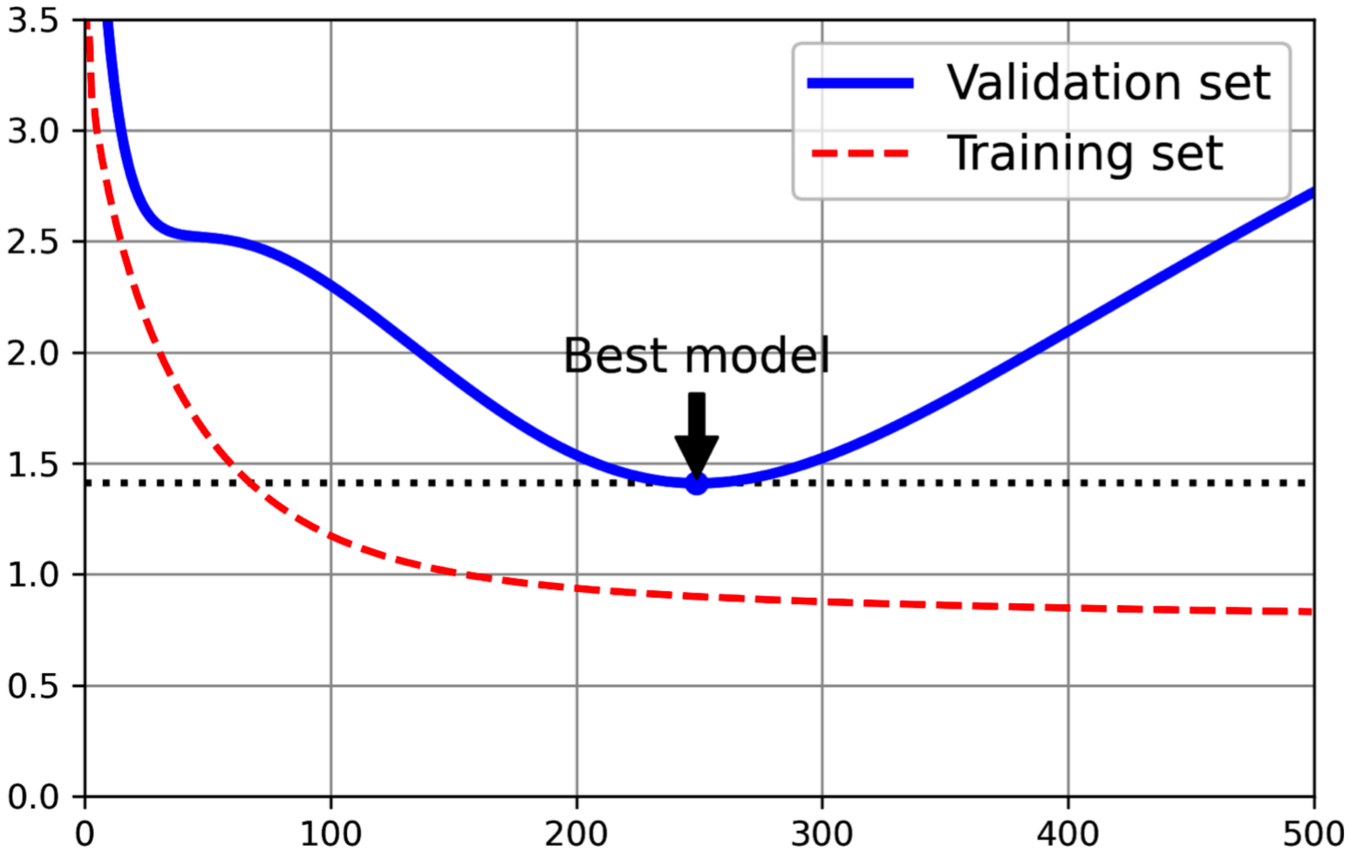
```
class MyDataset(torch.utils.data.Dataset):
    def __init__(self, img_paths, transform):
        self.path = img_paths
        self.transform = transform

    def __len__(self):
        return len(self.img_paths)

    def __getitem__(self, idx):
        img = load_image(self.img_paths[idx])
        img = self.transform(img)
        return img
```

PyTorch DataLoader

```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.4, 0.4, 0.4], std=[0.2, 0.2, 0.2])
])

dataset = MyDataset(img_paths=["a.jpg", "b.jpg"], transform)
dataloader = Dataloader(dataset, batch_size=32)

for batch in dataloader:
    # Do training
```


Training Set
60%
Validation Set
20%
Test Set
20%

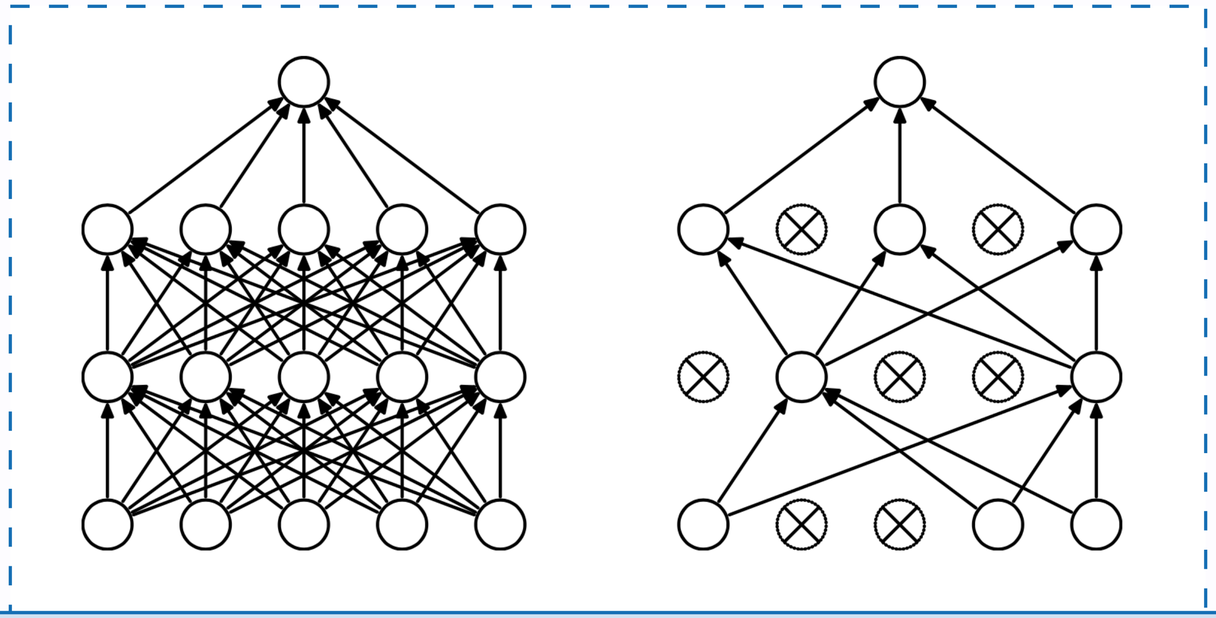
Early stopping regularization



L2 Regularization

```
optimizer = Adam(model.parameters(), lr=1e-4, weight_decay=1e-5)
```

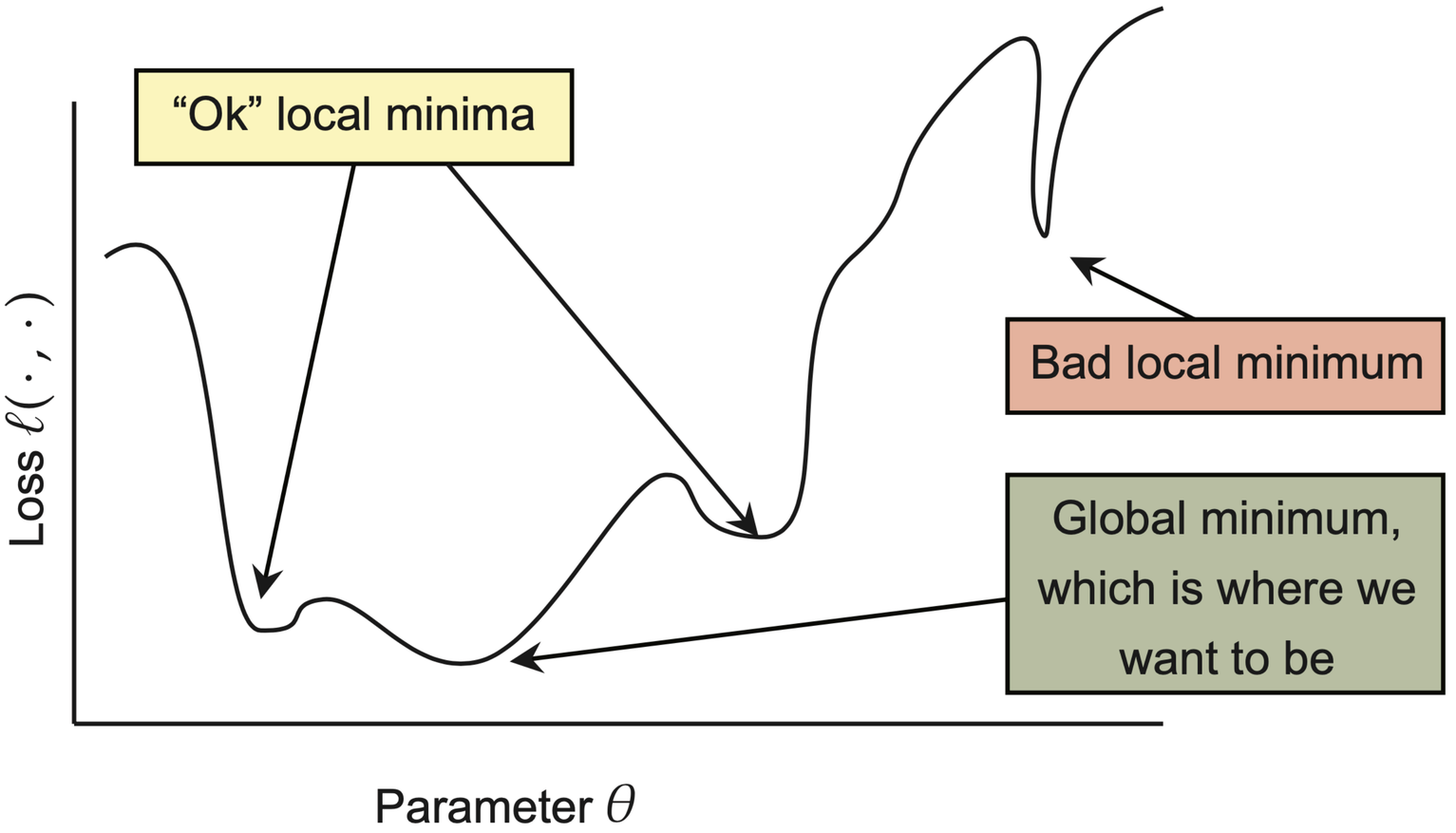
Dropout



Dropout

```
self.fc1 = nn.Linear(128, 64)
self.do1 = nn.Dropout(0.2)
```

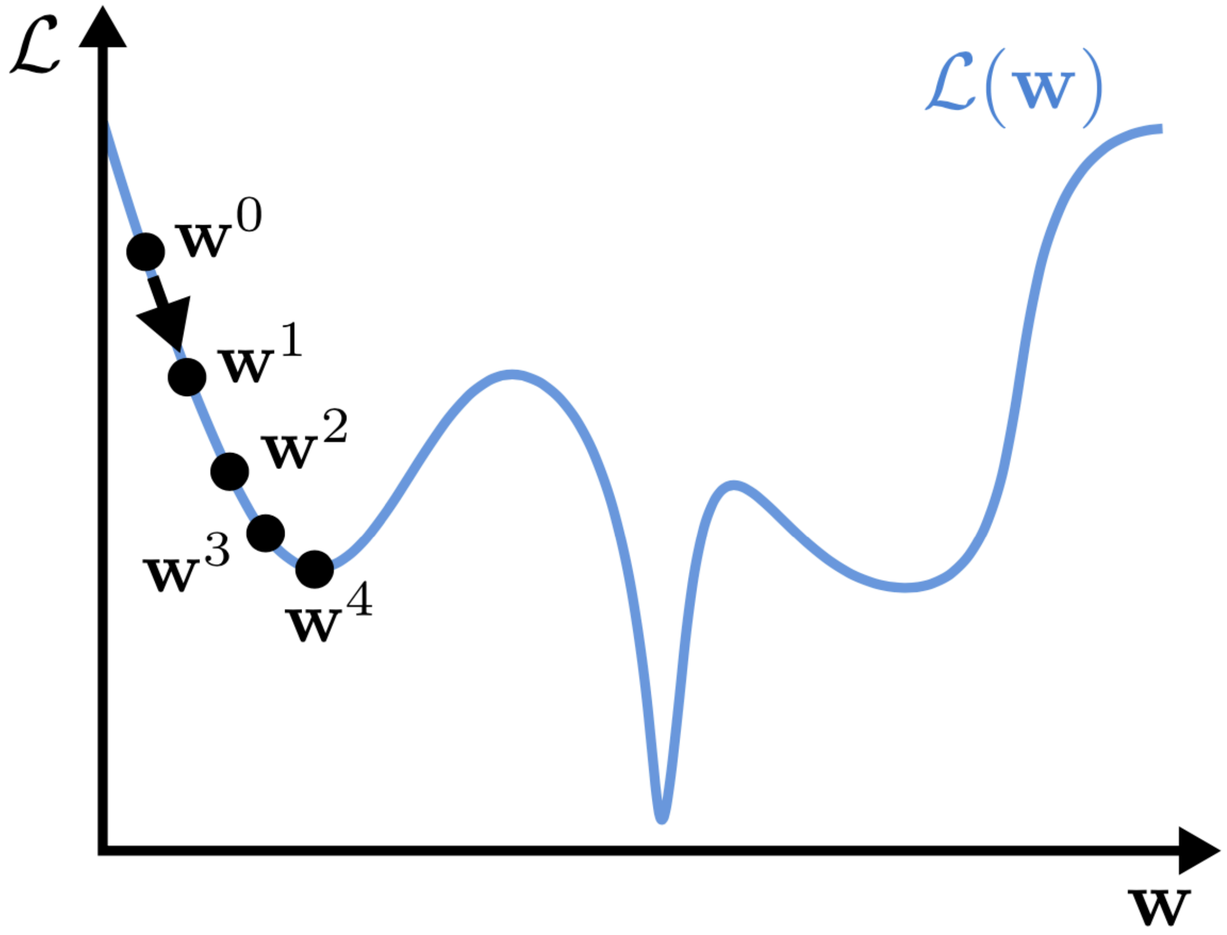
Deep Learning from optimization perspective



Gradient Descent Algorithm

$$w_{t+1} = w_t - \eta \cdot \nabla_{w_t} \ell \left(f_{w_t}(\mathbf{x}), y \right)$$

Gradient Descent Algorithm



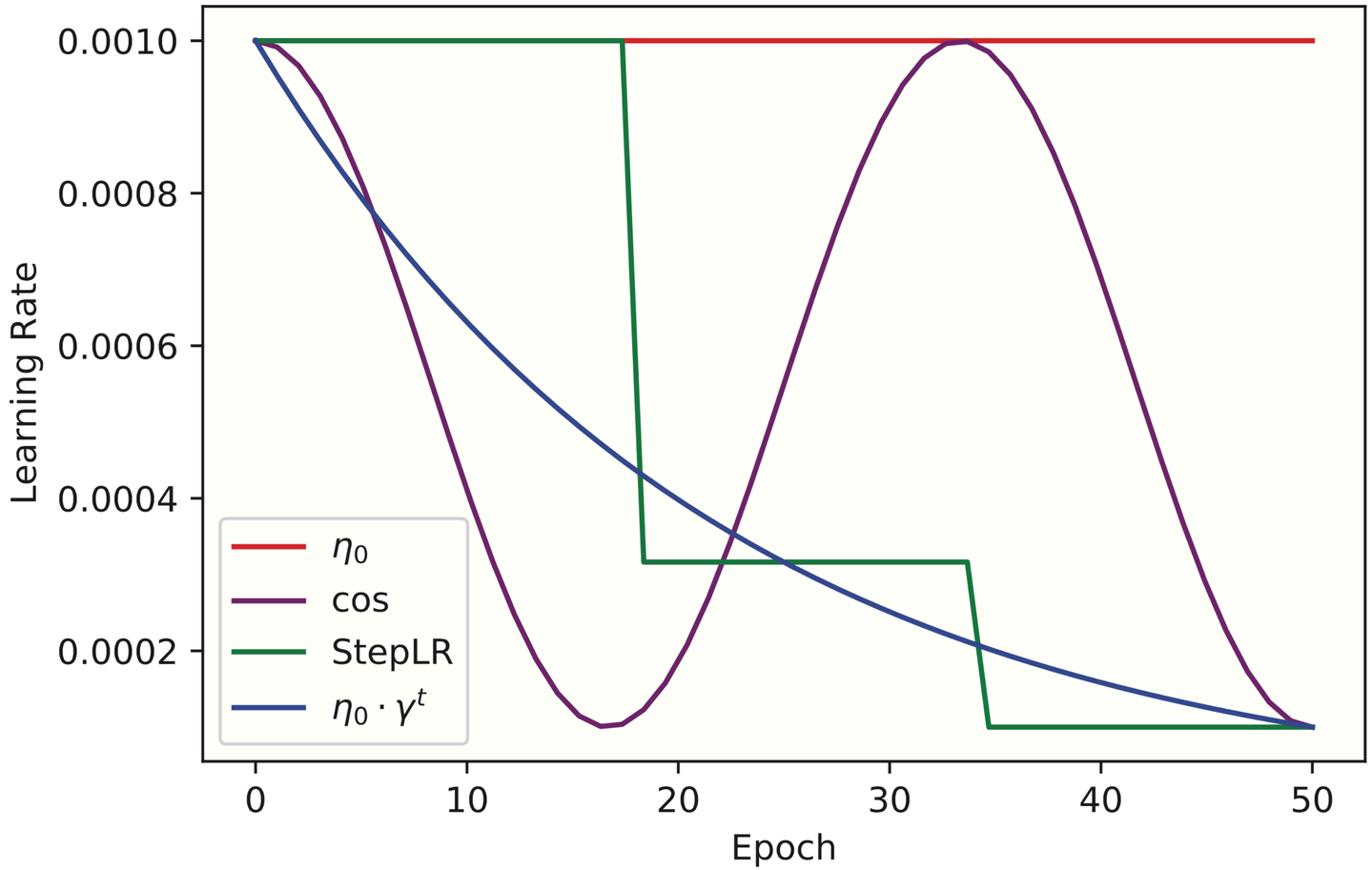
Scheduling function

$$w_{t+1} = w_t - S(\eta_0, t) \cdot \nabla_{w_t} \ell \left(f_{w_t}(\mathbf{x}), y \right)$$

Exponential LR

$$\eta_t = S_\gamma\left(\eta_0, t\right) = \eta_0 \gamma^t$$

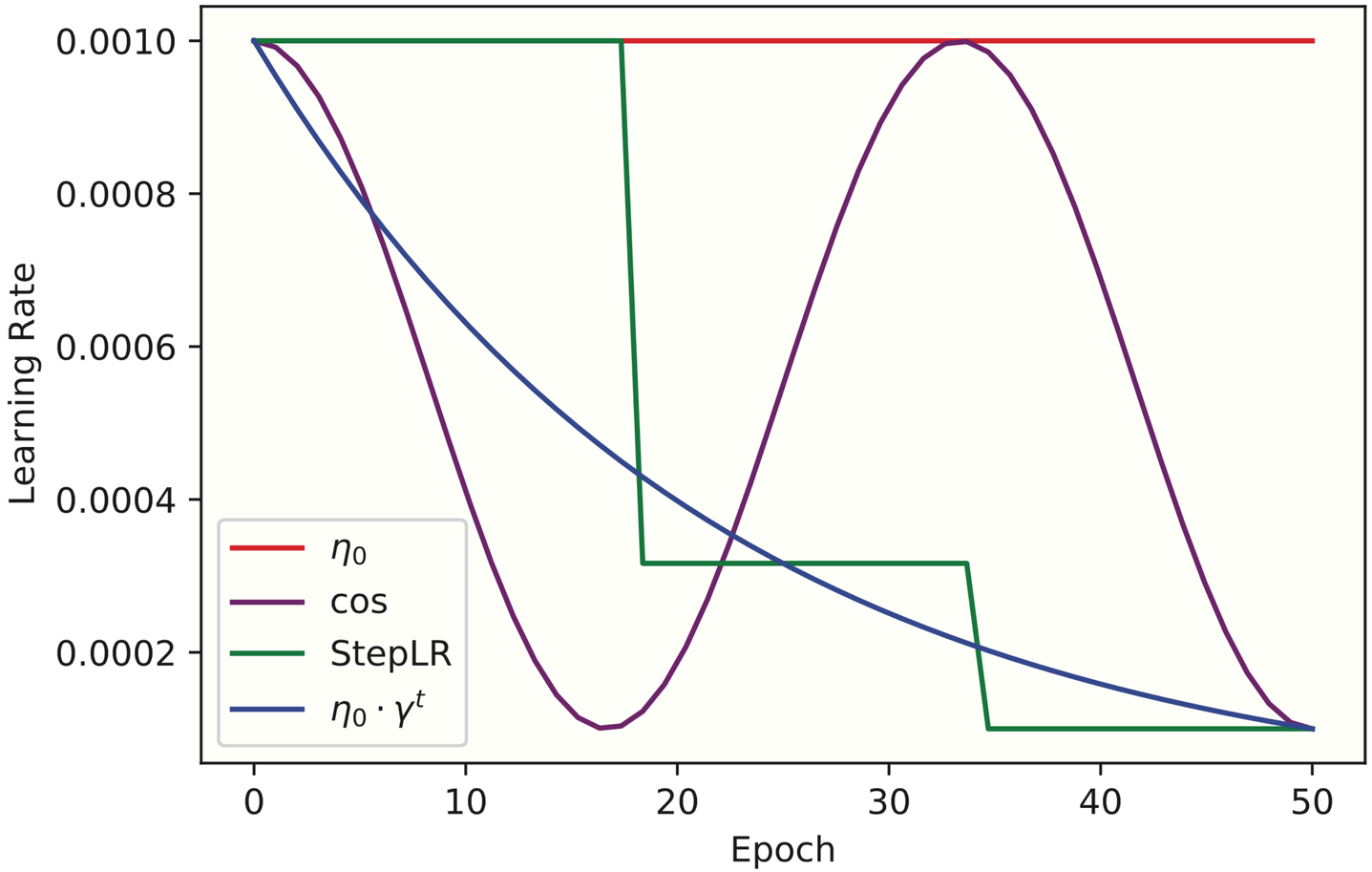
Schedulers


0.0010
0.0008
0.0006
0.0004
0.0002
Learning Rate
0
10
20
30
40
50
Epoch
η_0
cos
StepLR
$\eta_0 \cdot \gamma^t$

Step LR

$$\eta_t = S_\gamma\left(\eta_0, S, \gamma, t\right) = \eta_0 \gamma^{\lfloor t/S \rfloor}$$

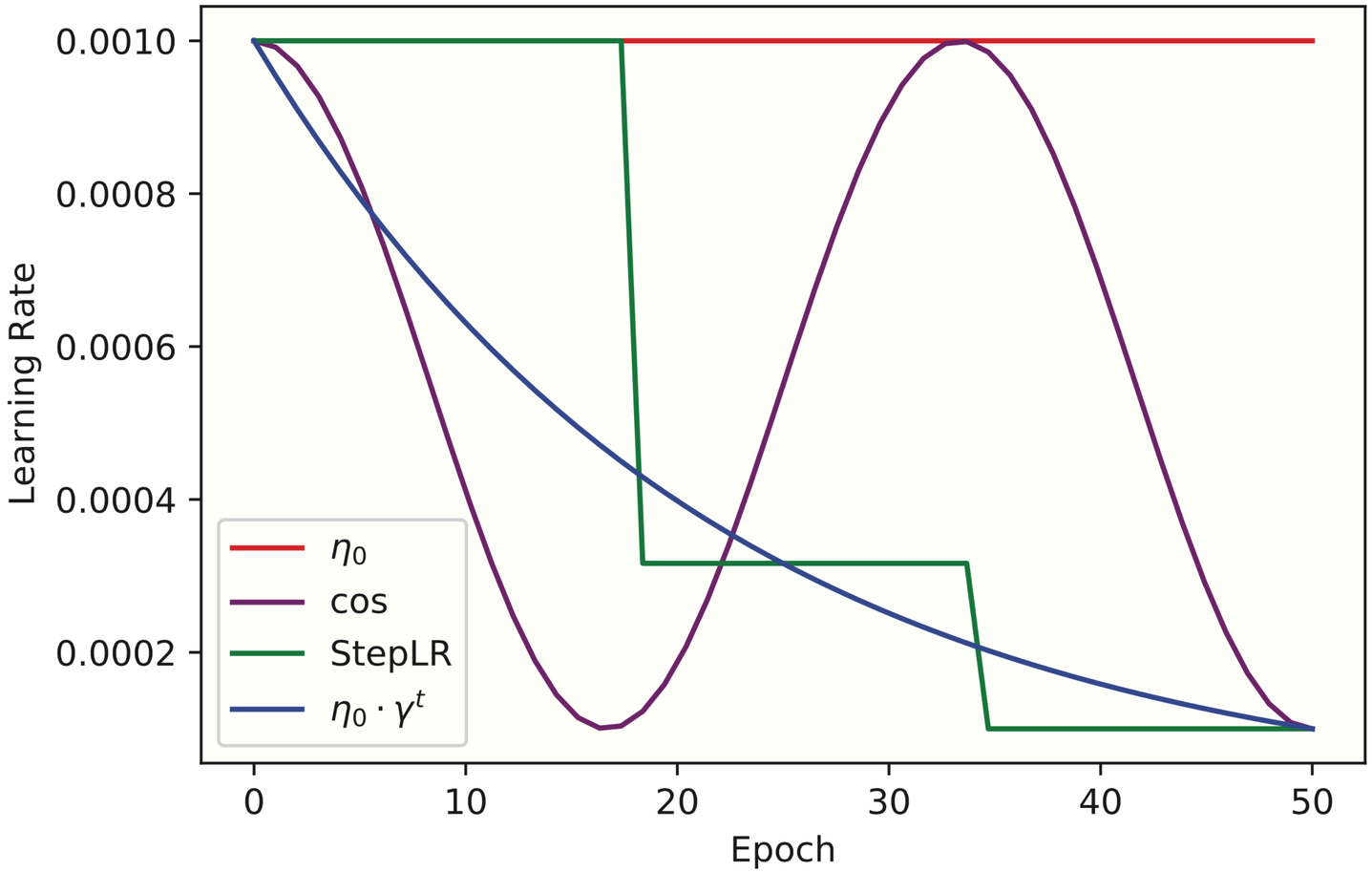
Schedulers


0.0010
0.0008
0.0006
0.0004
0.0002
Learning Rate
0
10
20
30
40
50
Epoch
η_0
cos
StepLR
$\eta_0 \cdot \gamma^t$

Cosine Annealing

$$\eta_t = \eta_{\min} + \left(\eta_0 - \eta_{\min}\right) \cdot \frac{1}{2}\left(1 + \cos\left(\frac{t}{T_{\max}} \cdot \pi\right)\right)$$

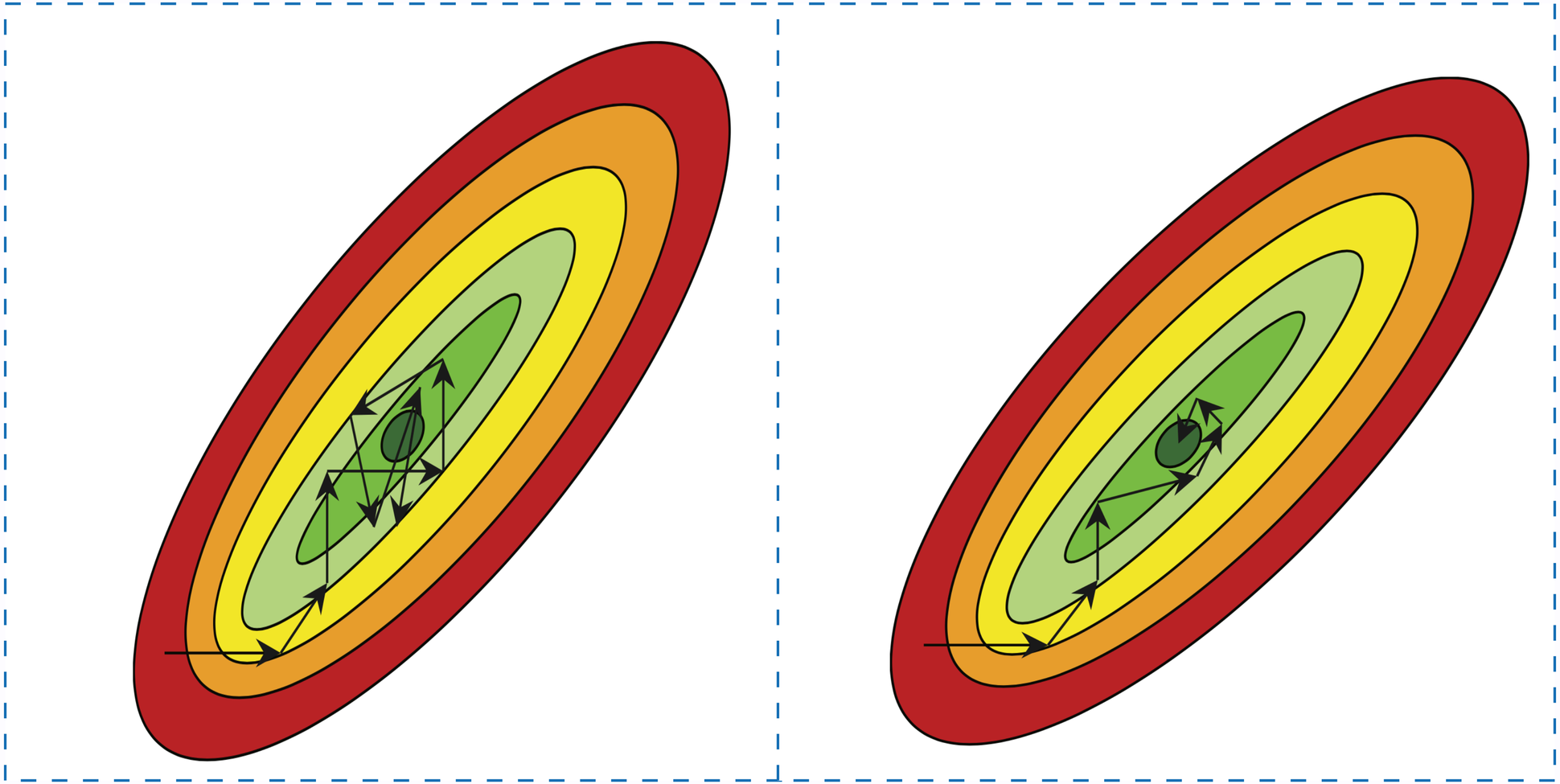
Schedulers


0.0010
0.0008
0.0006
0.0004
0.0002
Learning Rate
0
10
20
30
40
50
Epoch
η_0
cos
StepLR
$\eta_0 \cdot \gamma^t$

Schedulers

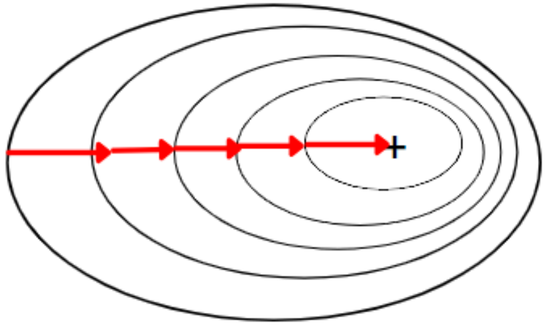
```
optimizer = torch.optim.SGD(model.parameters(), lr=eta_0)
# Exponential
scheduler = torch.optim.lr_scheduler.ExponentialLR(optimizer, 0.9)

# Step LR
scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
   epochs//4, gamma=0.3)

# Cosine Annealing
scheduler  =  torch.optim.lr_scheduler.CosineAnnealingLR(optimizer,
epochs//3, eta_min=0.0001)
```

Schedulers/Optimizers

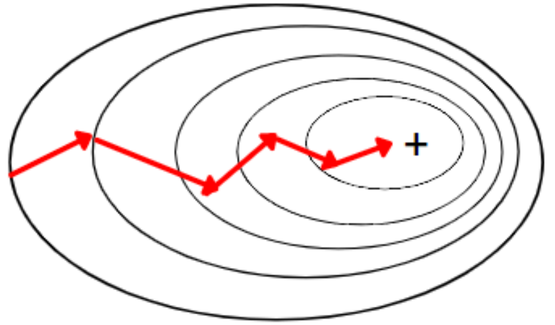


Optimizers

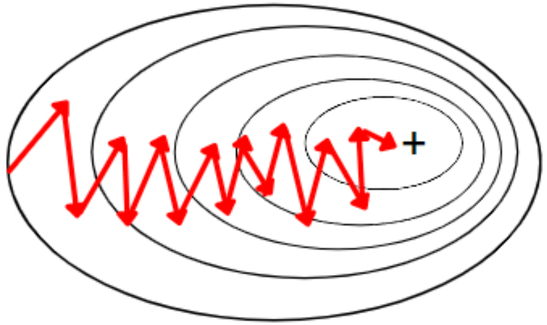
Batch Gradient Descent



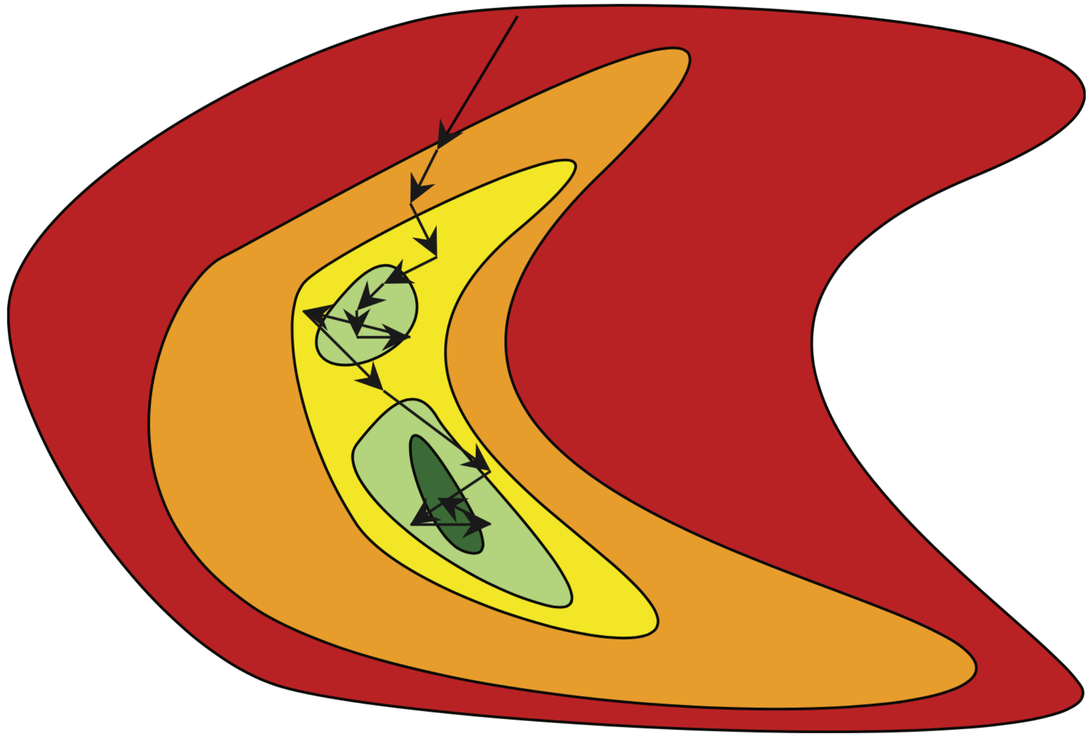
Mini-Batch Gradient Descent



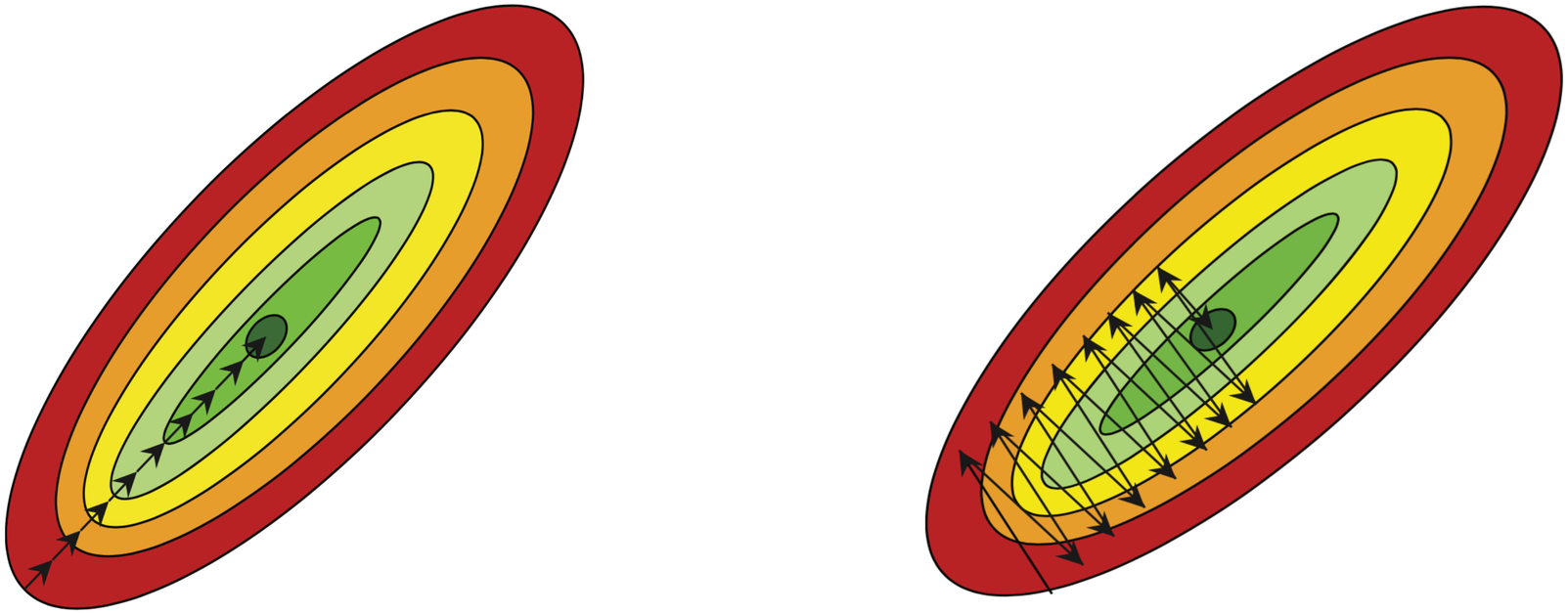
Stochastic Gradient Descent



Optimizers



Optimizers

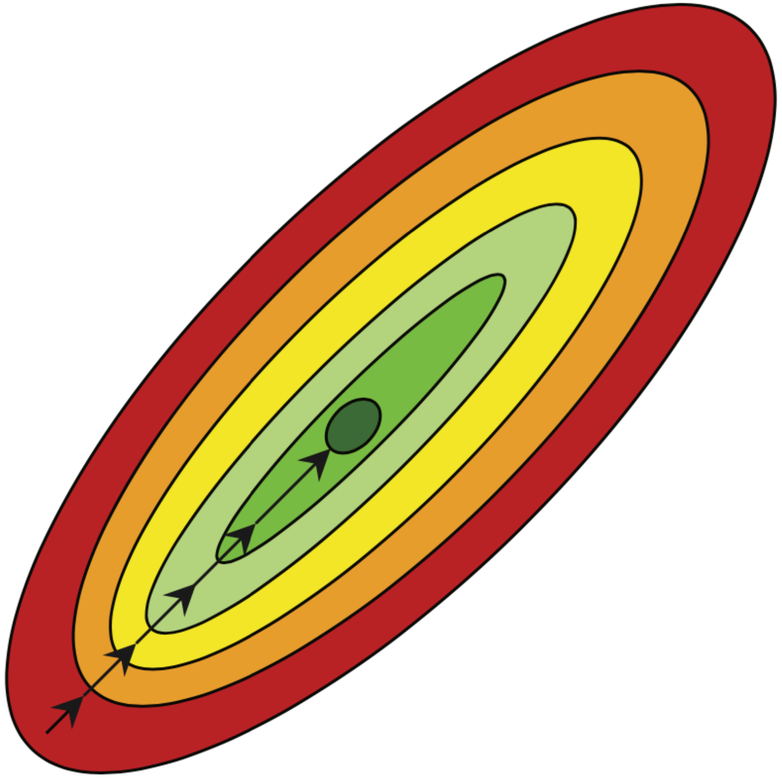


Gradient Descent with Momentum

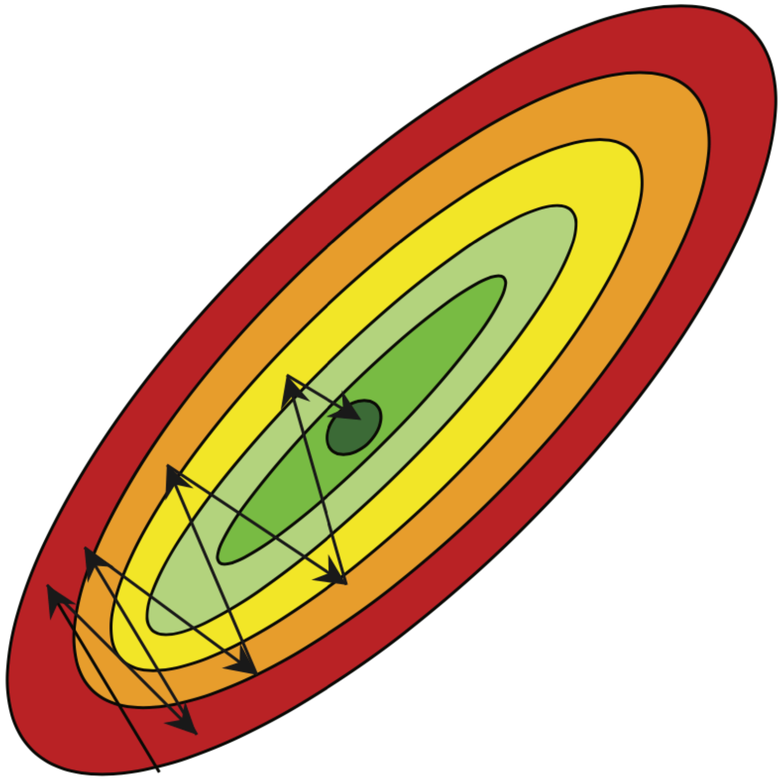
$$\boldsymbol{v}_{t+1} = \mu \boldsymbol{v}_t + \eta \cdot \nabla_{w_t} \ell \left(f_{w_t}(\boldsymbol{x}), y \right)$$

$$w_{t+1} = w_t - \mu \boldsymbol{v}_{t+1}$$

Gradient Descent with Momentum

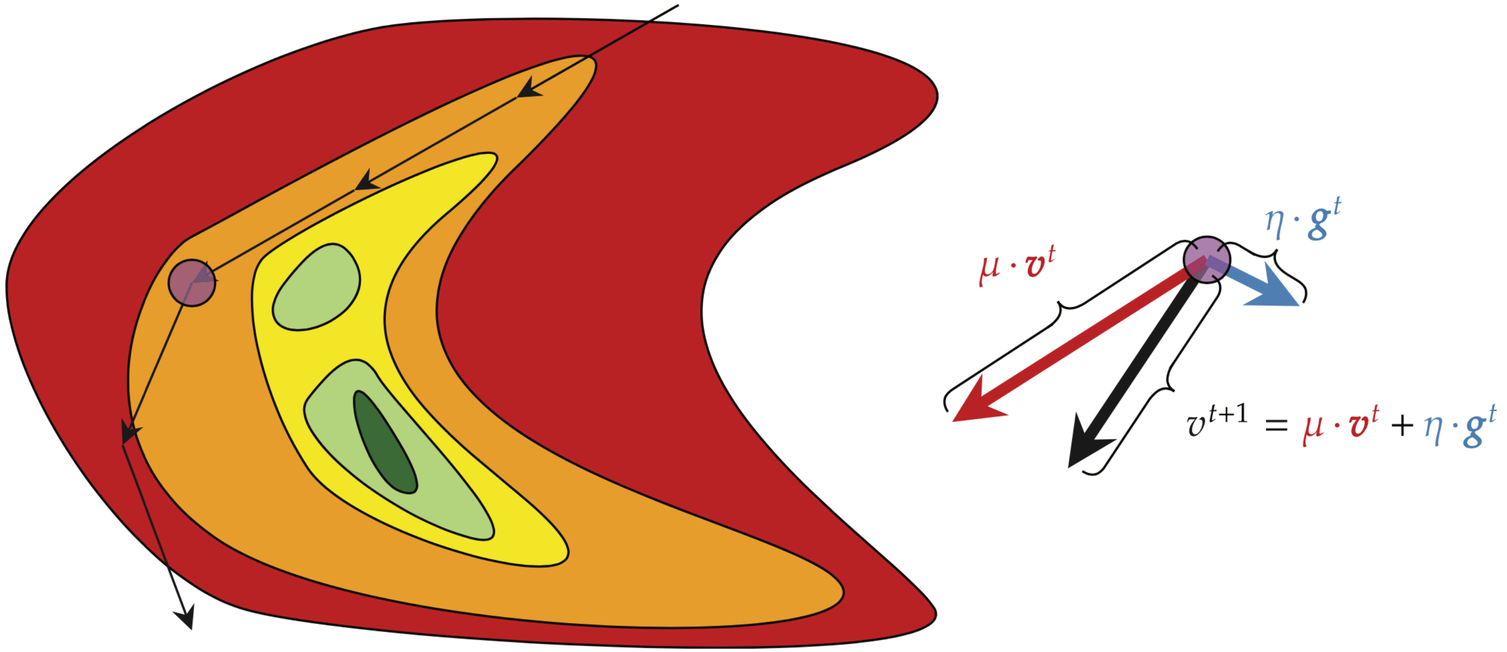


Solving simple small learning rate



Solving complex oscillation

Gradient Descent Algorithm



Optimizers

```
optimizer = torch.optim.SGD(model.parameters(), lr=eta_0,
momentum=0.9)

optimizer = torch.optim.AdamW(model.parameters())



# Gradients clamping
for p in model():
    p.register_hook(lambda grad: torch.clamp(grad, -5, 5))
```

Subjects

1. Advanced PyTorch
2. Python Best Practices
3. Neural Style Transfer

PEP8 Spacing Conventions

- Four spaces per level of indentation
- Put a Single Space Between Operators and Identifiers
- Put No Spaces Before Separators and a Single Space After Separators
- Don’t Put Spaces Before or After Periods
- Don’t Put Spaces After a Function, Method, or Container Name
- Don’t Put Spaces After Opening Brackets or Before Closing Brackets
- Put Two Spaces Before End-of-Line Comments
- separate functions with two blank lines, classes with two blank lines, and methods within a class with one blank line.
- Separate groups of code with a single line

PEP8 Importing Conventions

- PEP 8 also recommends grouping import statements into the following three groups in this order:
- Modules in the Python standard library, like math, os, and sys
- Third-party modules, like Selenium, Requests, or Django
- Local modules that are a part of the program

  These guidelines are optional, and Black won’t change the formatting
- of your code’s import statements.

Black and Linting

- Use Black for PEP8 checkup
- Use Linting for syntax errors and PEP8 checkups
- snake_case vs camelCase vs PascalCase

Black

```
python -m black my_code.py



# fmt: off
# Do sth here
# fmt: on
```

Naming conventions

- All letters should be ASCII letters—that is, uppercase and lowercase English letters that don’t have accent marks.
- Modules should have short, all lowercase names.
- Class names should be written in PascalCase.
- Constant variables should be written in uppercase SNAKE_CASE.
- Function, method, and variable names should be written in lowercase snake_case.
- The first argument for methods should always be named self in lowercase.
- Private attributes in classes should always begin with an underscore (_).
- Public attributes in classes should never begin with an underscore (_).

Naming conventions

- Appropriate Name Length
- Make Names Searchable
- Avoid Jokes, Puns, and Cultural References
- Don’t Overwrite Built-in Names
- The Worst Possible Variable Names Ever: **data**, **var**, **tmp**

Conda for separating environments

```
brew install --cask miniconda
conda init ${SHELL} # reset or reload shell
conda create -n main python=3.9.16
conda activate main
conda install PACCKAGE
conda env export --from-history --file environment.yml
```

Debugging

- Breakpoints
- Logpoints
- Watchlist

Misc

- Pathlib
- PIL
- fStrings

PIL Image

```
from PIL import Image
import numpy as np

# read image
img = Image.open('image.png')

# resize
img = img.resize((128, 128))

# convert to numpy array
img = np.array(img)

img = Image.fromarray(img)  # convert back to PIL

img.save('image.png')  # save
```

Pathlib

```
from pathlib import Path
Path.home()
Path("path1") / "path2" / "path3"
Path.cwd()

# Iterate
for f in Path(directory_path).glob('*.csv'):  # always lazy exec
    pass
```

fStrings

```
alpha = 55
beta = 65

# with formatting
print("value of alpha is {0} and beta is {1}".format(alpha, beta))

# with f-strings
print(f"value of alpha is {alpha} and beta is {beta}")
```

Subjects

1. Advanced PyTorch
2. Python Best Practices
3. Neural Style Transfer

Neural Style Transfer

Neural Style Transfer

- Task 1: Import a pre-trained VGG19
- Task 2: The transform function
- Task 3: Get activation maps
- Task 4: The Gram matrix
- Task 5: The loss functions