Ứng dụng Trí tuệ nhân tạo trong Nuôi trồng thủy sản

NGUYỄN HẢI TRIỀU[1]

[1] Bộ môn Kỹ thuật phần mềm,

Khoa Công nghệ thông tin, Trường ĐH Nha Trang

NhaTrang, September 2024

Save Hyperparameters

Trong quá trình huấn luyện, chúng ta nên lưu lại trạng thái của mô hình ở một số bước huấn luyện cụ thể, quá trình này gọi là checkpoint. Ưu điểm lớn nhất của checkpoint là bạn có thể load lại model ở bước huấn luyện gần nhất mà không phải huấn luyện lại từ đầu. Ngoài ra, checkpoint còn

- Lưu trữ các **trọng số** và **tham số** của mô hình.
- Tuy nhiên chúng ta thường *không lưu tham số mô hình* mà *chỉ lưu trọng số* để giảm kích thước file lưu trữ và khi inference mode sẽ gọi lại kiến trúc model + trọng số đã lưu trước đó.

Saving & Loading a Model Checkpoint I

Lưu checkpoint

Trong bước huấn luyện của code nhận diện các loại bệnh cá *"FishDesease.ipynb"*, ta lưu checkpoint sau 10 epoch. Lưu ý, chúng ta không không lưu tham số của model: *save_weights_only=True*.

```
1
2  # Set up the checkpoint callback to save the model every 10 epochs
3  checkpoint_callback = ModelCheckpoint(
4    dirpath=f"Fish_Desease_checkpoints/{model_name}/",
5    filename="{epoch}-{val_loss:.2f}",
6    save_top_k=-1, # Save all checkpoints based on the interval
7    save_weights_only=True, # Save only the model weights
8    every_n_epochs=10, # Save after every 10 epochs
9  )
```

Saving & Loading a Model Checkpoint II

Load lại model và tái sử dụng

Trong trường hợp chúng ta huấn luyện đến một epoch cụ thể, bây giờ chúng ta muốn tái sử dụng lại model, ta tiến hành load lại model như sau:

```
1  pytorch_model = torch.hub.load('pytorch/vision', resnet_type, weights=None)
2  num_ftrs = pytorch_model.fc.in_features
3  pytorch_model.fc = torch.nn.Linear(num_ftrs, 7)
4  lightning_model =
       LightningModel.load_from_checkpoint(checkpoint_path="/content/drive/MyDrive/C
       Notebooks/Fish_Desease_checkpoints/fish-disease-resnet18-baseline/epoch=27-va
5    model=pytorch_model, learning_rate=0.005)
```

Saving & Loading a Model Checkpoint III

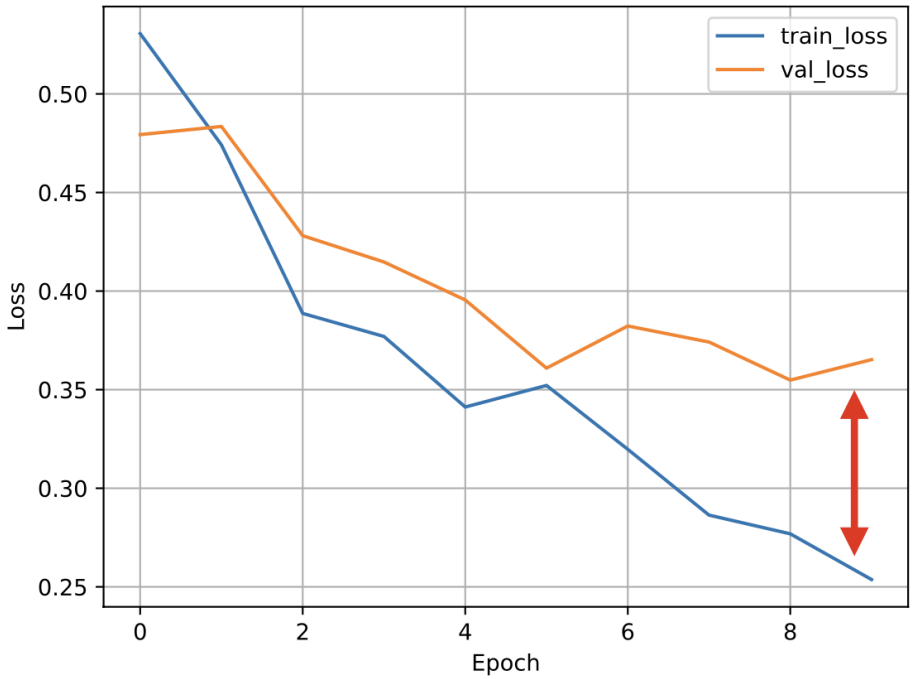
Dự đoán trên dữ liệu mới

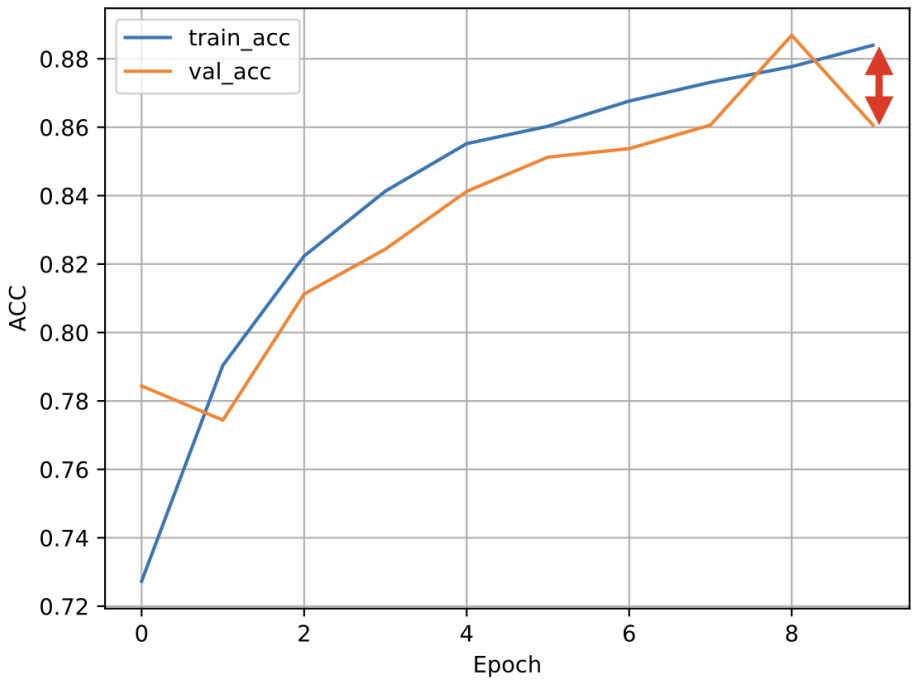
Sau khi load lại model, chúng ta có thể tái sử dụng để dự đoán trên dữ liệu mới trong *inference_mode*

```
1  lightning_model.eval()
2  with torch.inference_mode():
3    logit = lightning_model(dm.single[0].unsqueeze(0).to(device))
4    pred = torch.argmax(logit, dim=1)
```

Early Stopping

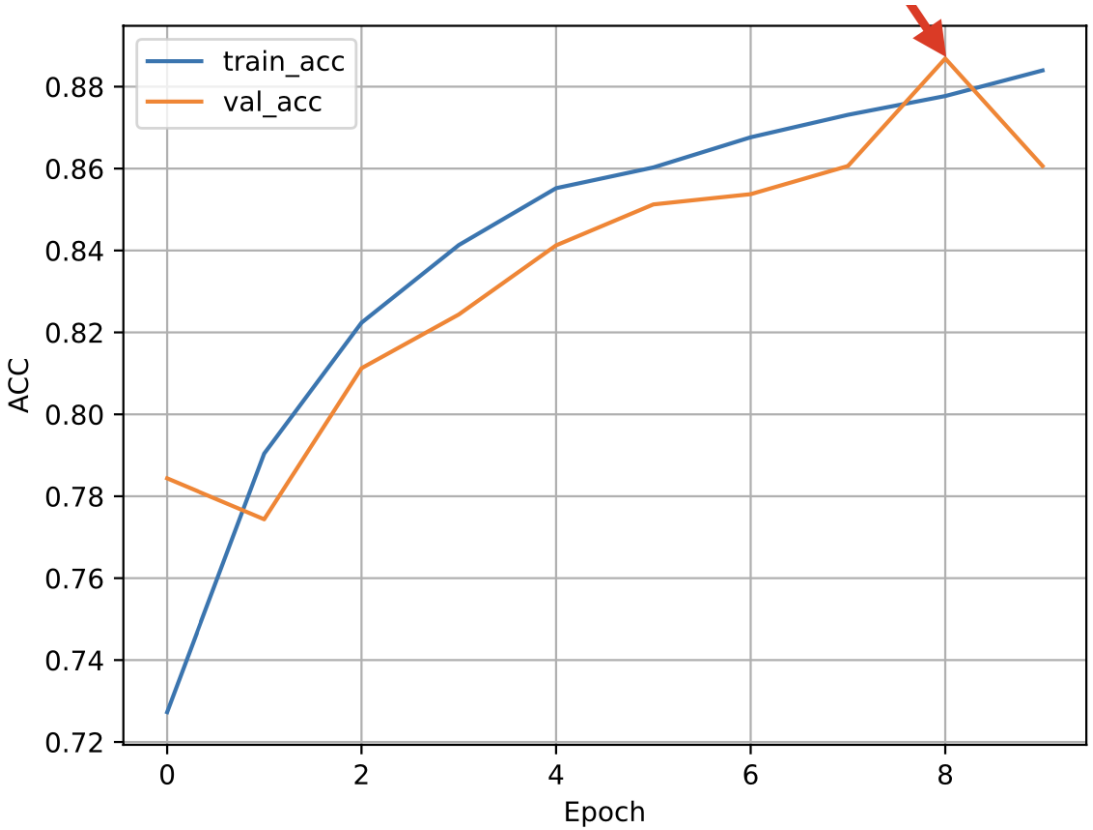
Trong quá trình huấn luyện, **không phải lúc nào epoch cuối cùng cũng cho ta kết quả tốt nhất**. Ví dụ, ta có kết quả huấn luyện như hình bên dưới, rõ ràng ở epoch thứ 9 bắt đầu xuất hiện overfitting.





Early Stopping I

Có cần thiết phải chạy lại huấn luyện mô hình đến bước trước không? Chỉ cần lưu mô hình tốt nhất trong quá trình huấn luyện.



Early Stopping II

Trong bước huấn luyện của code nhận diện các loại bệnh cá *"FishDesease.ipynb"*, ta lưu lại model tốt nhất với tuỳ chọn: save_top_k=1

```
checkpoint_callback = ModelCheckpoint(
  dirpath=f"Fish_Desease_checkpoints/{model_name}/",
  filename="{epoch}-{val_loss:.2f}",
  save_top_k=1, # Save the best model
  save_weights_only=True, # Save only the model weights
  every_n_epochs=10, # Save after every 10 epochs
)
```

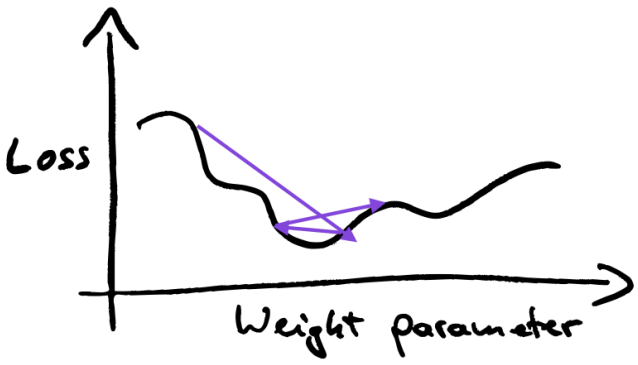
Learning rate

Learning rate là một trong những tham số siêu (hyperparameters) quan trọng nhất cần điều chỉnh trong quá trình huấn luyện. Learning rate ảnh hưởng trực tiếp đến quá trình cập nhật trọng số được biểu diễn ở hình bên dưới.

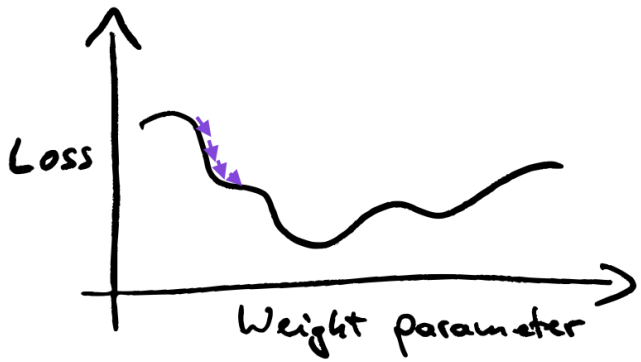
Learning rate

Với learning rate cao, bước nhảy của trọng số mỗi lần cập nhật là rất lớn, dẫn đến việc hàm loss bỏ qua điểm tối ưu và chỉ dao động xung quanh điểm tối ưu làm hàm loss không hội tụ được.

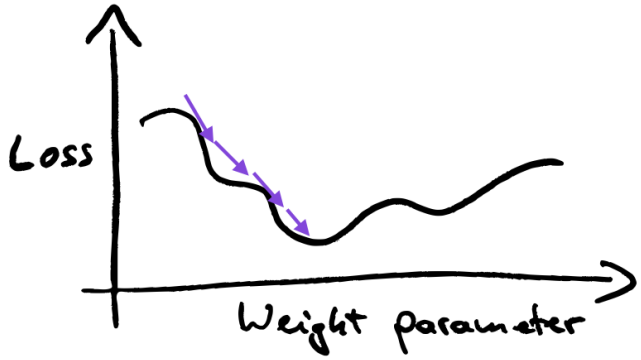
High learning rate



Low learning rate



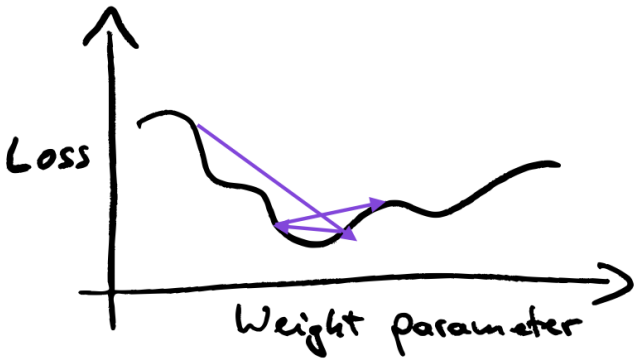
Good learning rate



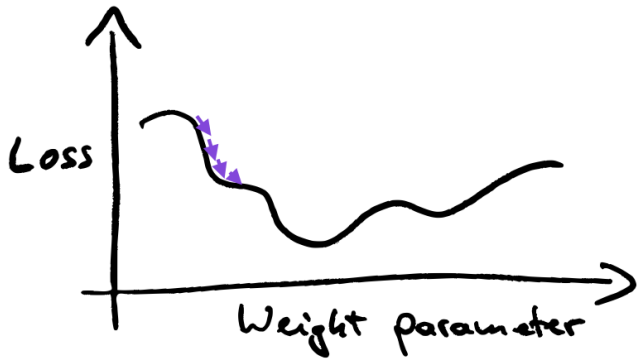
Learning rate

Ngược lại, với learning rate thấp, ta cần nhiều epochs hơn để hàm loss hội tụ, làm tăng quá trình huấn luyện. Ngoài ra, có thể xảy ra hiện tượng hàm loss hội tự về các cực tiểu cục bộ (local minima) thay vì cực tiểu toàn cục (global minimum).

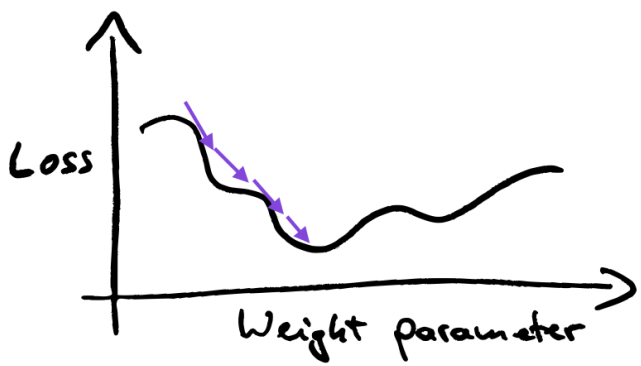
High learning rate



Low learning rate



Good learning rate

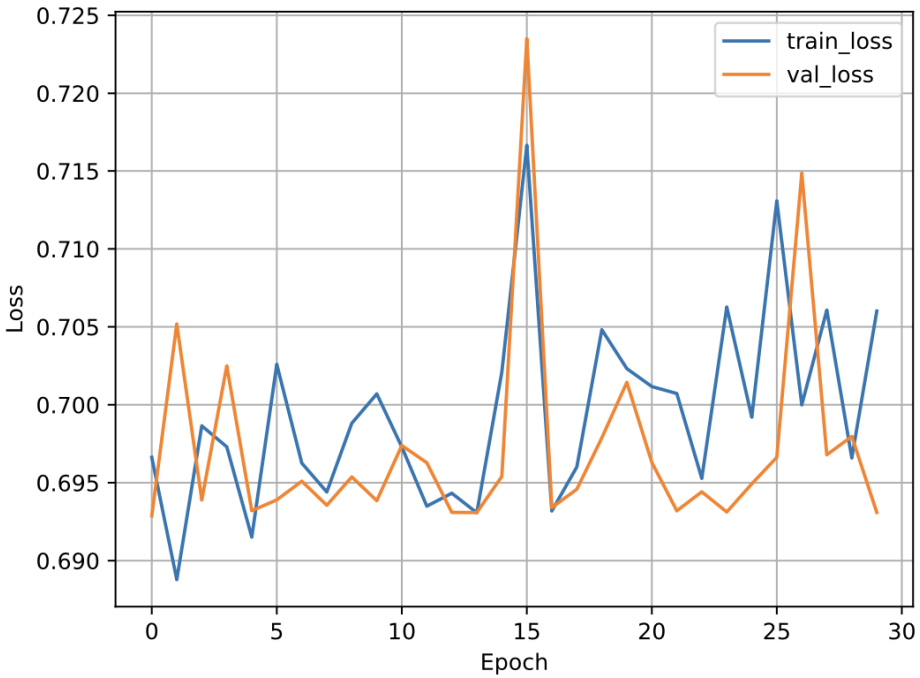


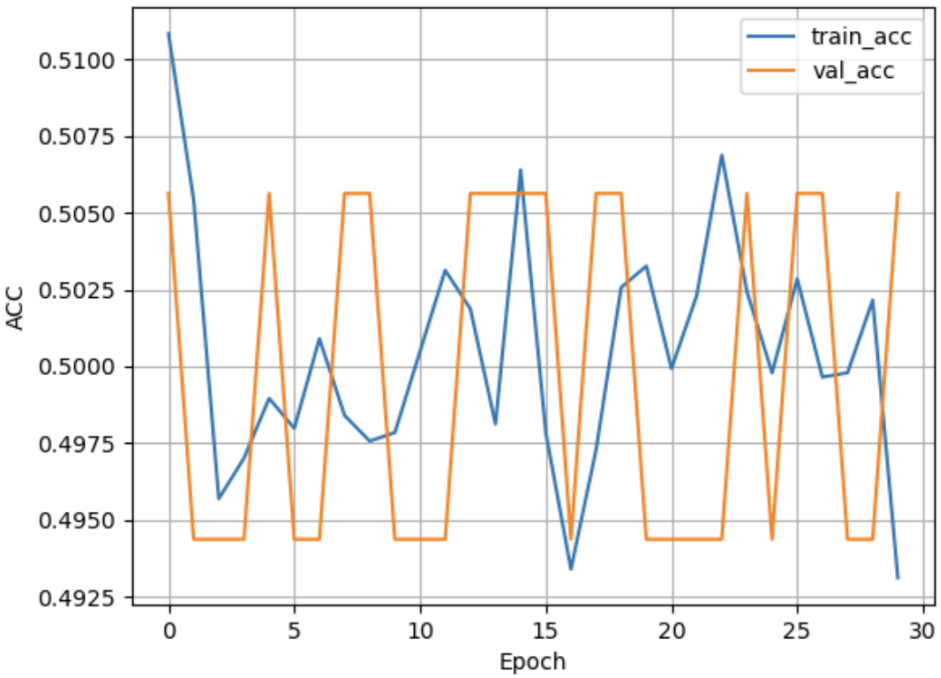
Learning rate

Learning rate α is too high

$$\mathbf{w} = \mathbf{w} + \alpha \nabla L_{\mathbf{w}},\ b = b + \alpha \nabla L_{\mathbf{b}}.$$

The weight and bias unit updates will be too large. And the loss will jump erratically.



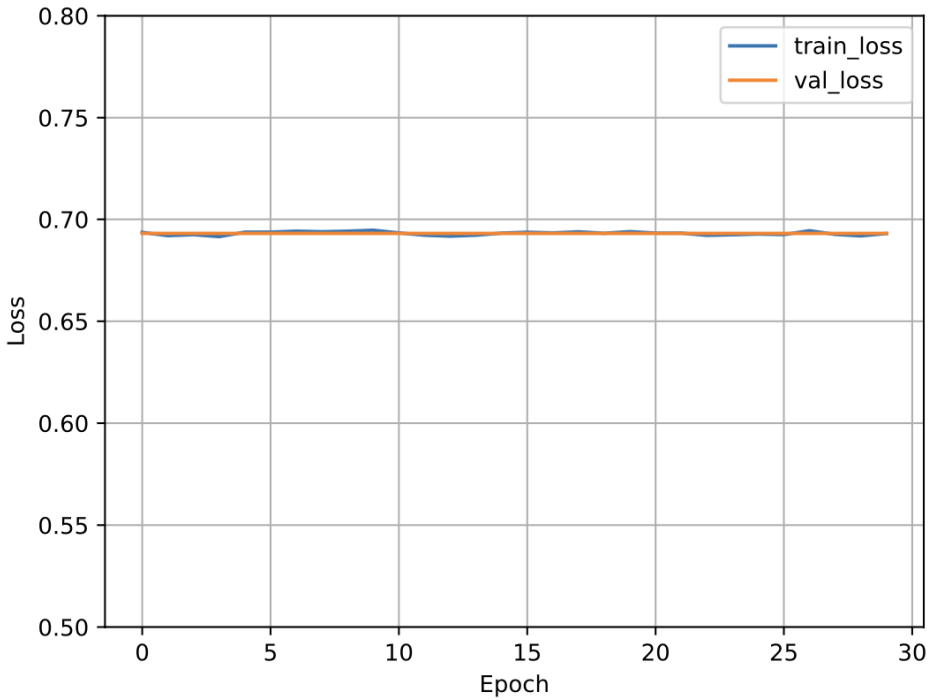


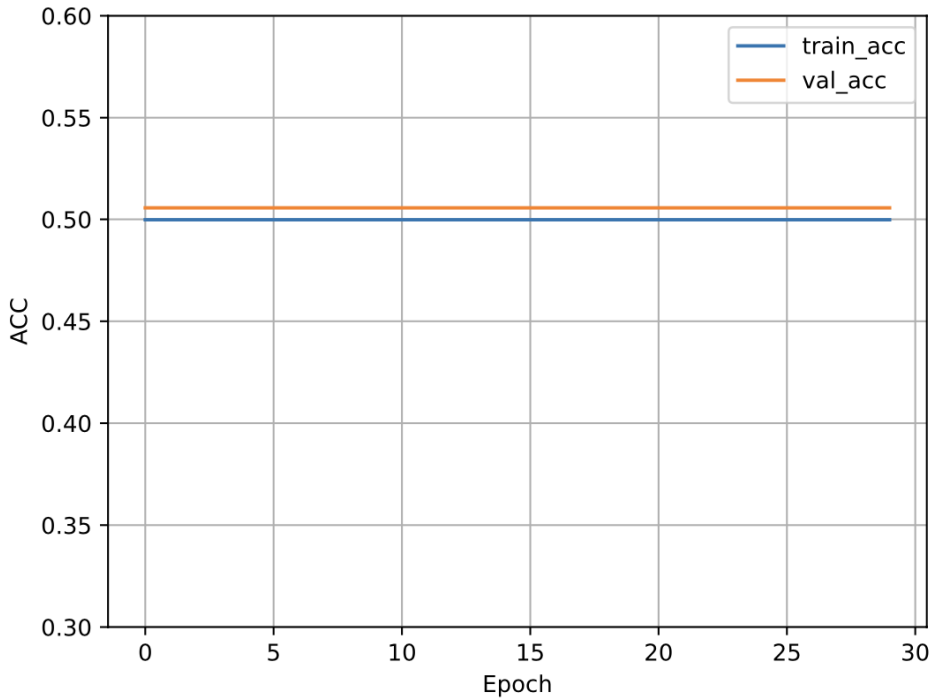
Learning rate

Learning rate α is too low

$$\mathbf{w} = \mathbf{w} + \alpha \nabla L_{\mathbf{w}},\ b = b + \alpha \nabla L_{\mathbf{b}}.$$

The weight and bias unit updates will be too small.





Learning rate

Strategy for finding *lr*

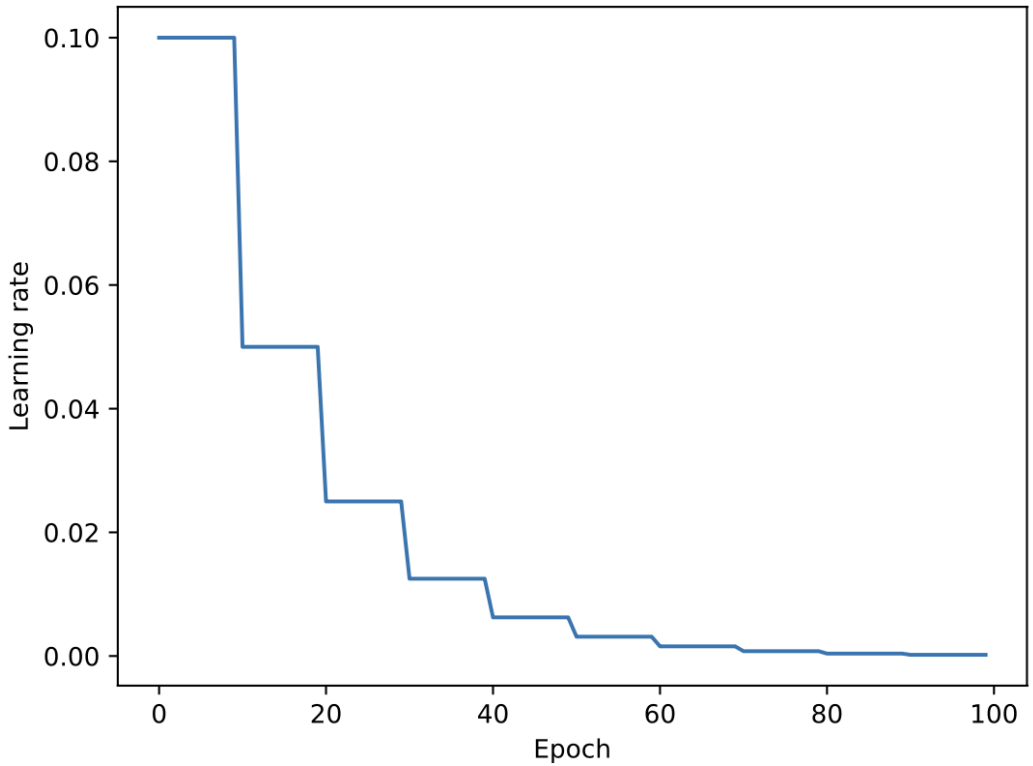
Làm thế nào để tìm được giá trị *lr* phù hợp để huấn luyện mô hình?

1. Bằng cách làm thủ công: thử nghiệm bắt đầu với learning rate đủ lớn, sau đó giảm nó xuống cho đến khi đủ nhỏ để làm mô hình huấn luyện tốt hơn.
2. Tự động tìm learning rate phù hợp sử dụng các thuật toán tích hợp trong Lighning.

```
trainer = L.Trainer(
    max_epochs=30,
    auto_lr_find=True,
    accelerator="cpu",
    devices="auto",
    logger=CSVLogger(save_dir="logs/", name="my-model"),
    deterministic=True,
)

results = trainer.tune(model=lightning_model, datamodule=dm)
```

Learning Rate Schedulers

Learning rate schedulers change (often decay) the learning rate over time.



Learning Rate Schedulers

Tại sao chúng ta cần phải sử dụng Learning Rate Scheduler?

1. Better convergence
2. Better accuracy

Lưu ý khi sử dụng Learning Rate Scheduler

1. Decay learning rate too slowly = no advantage
2. Decay learning rate too fast = training will get stuck (loss không hội tụ)

Lưu ý: ban đầu, chúng ta không nên sử dụng *LR Scheduler* để huấn luyện model. Sau khi huấn luyện với *lr* cố định, chúng ta sẽ áp dụng *LR Scheduler* trong các thử nghiệm để tăng tốc độ hội tụ và độ chính xác.

Learning Rate Schedulers

Learning Rate Schedulers gồm có nhiều phương pháp như:

- StepLR
- decay on plateau
- cosine annealing

Trong đó phương pháp StepLR là đơn giản nhất.

StepLR

Learning rate is decayed by *gamma* (multiplicative factor) every *step_size epochs.*
Nghĩa là nhân gamma cho LR sau một số lượng epoch nhất định.

Learning Rate Schedulers

StepLR

Learning rate is decayed by *gamma* (multiplicative factor) every *step_size epochs*.
Nghĩa là nhân gamma cho LR sau một số lượng epoch nhất định.

```
opt = torch.optim.SGD(pytorch_model.parameters(), lr=0.1)
sch = torch.optim.lr_scheduler.StepLR(opt, step_size=10, gamma=0.5)


lrs = []
max_epochs = 100

for epoch in range(max_epochs):
    opt.step()
    lrs.append(opt.param_groups[0]["lr"])
    sch.step()

plt.plot(range(max_epochs), lrs)
plt.xlabel('Epoch')
plt.ylabel('Learning rate')

plt.show()
```

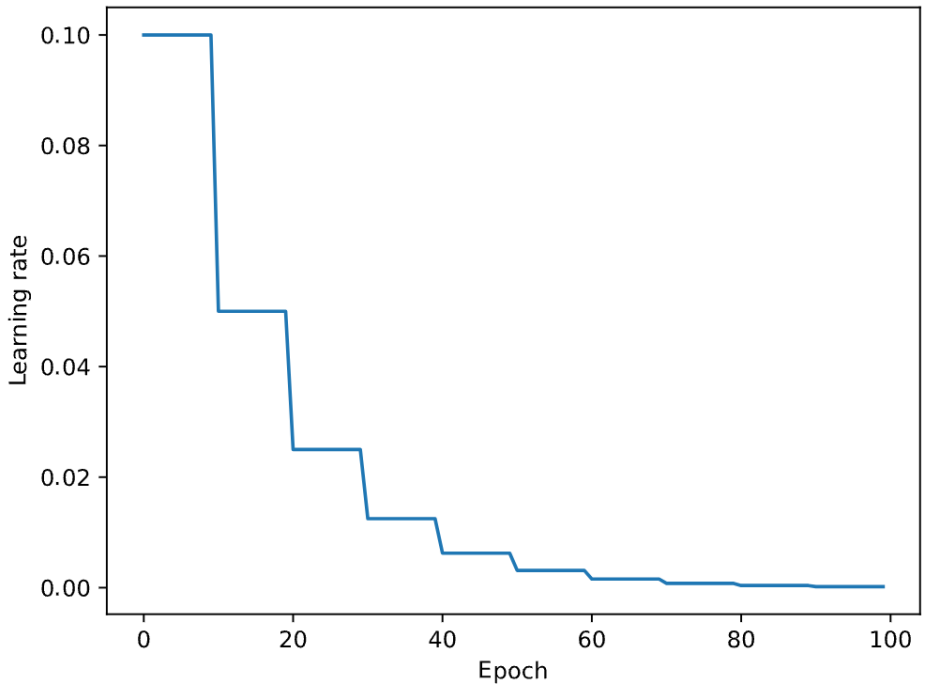
Half the learning rate every 10 epochs

Learning Rate Schedulers

StepLR

Learning rate giảm một nữa cứ sau mỗi 10 epochs.

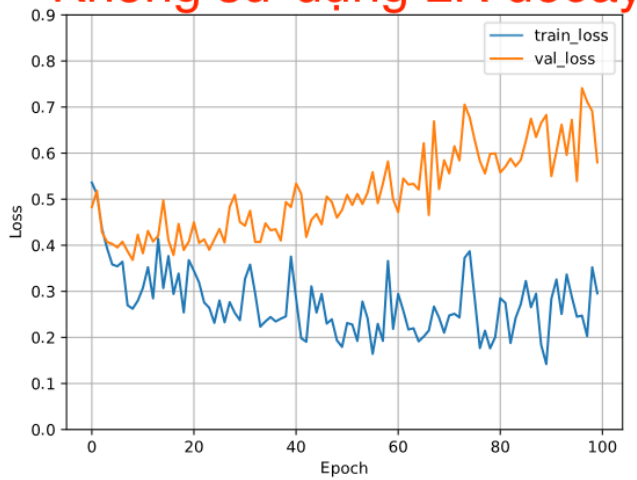
```
opt = torch.optim.SGD(pytorch_model.parameters(), lr=0.1)
sch = torch.optim.lr_scheduler.StepLR(opt, step_size=10, gamma=0.5)


lrs = []
max_epochs = 100

for epoch in range(max_epochs):
    opt.step()
    lrs.append(opt.param_groups[0]["lr"])
    sch.step()

plt.plot(range(max_epochs), lrs)
plt.xlabel('Epoch')
plt.ylabel('Learning rate')

plt.show()
```

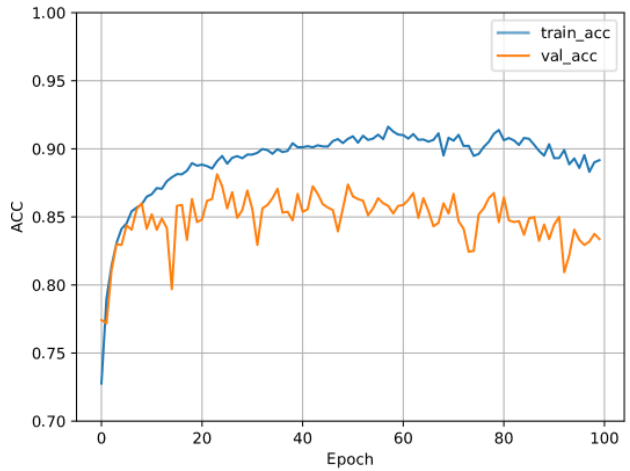


Learning Rate Schedulers

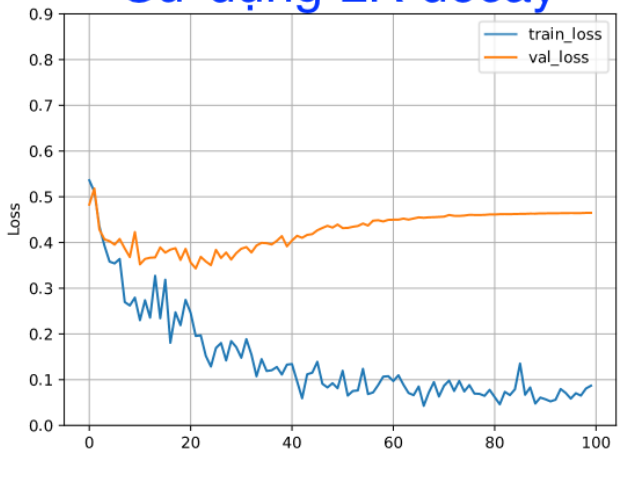
So sánh huấn luyện sử dụng LR Scheduler và không sử dụng

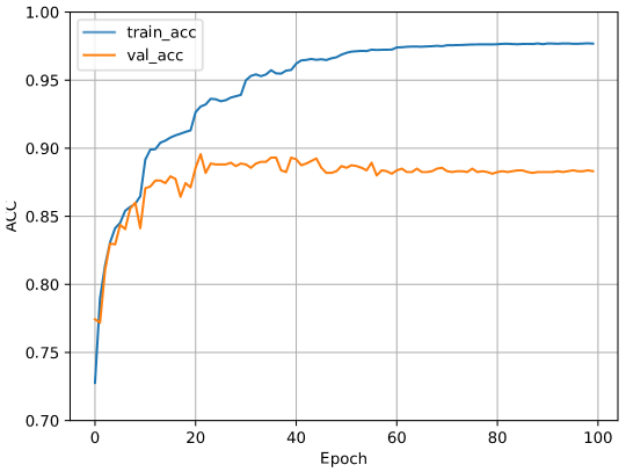
Không sử dụng LR decay





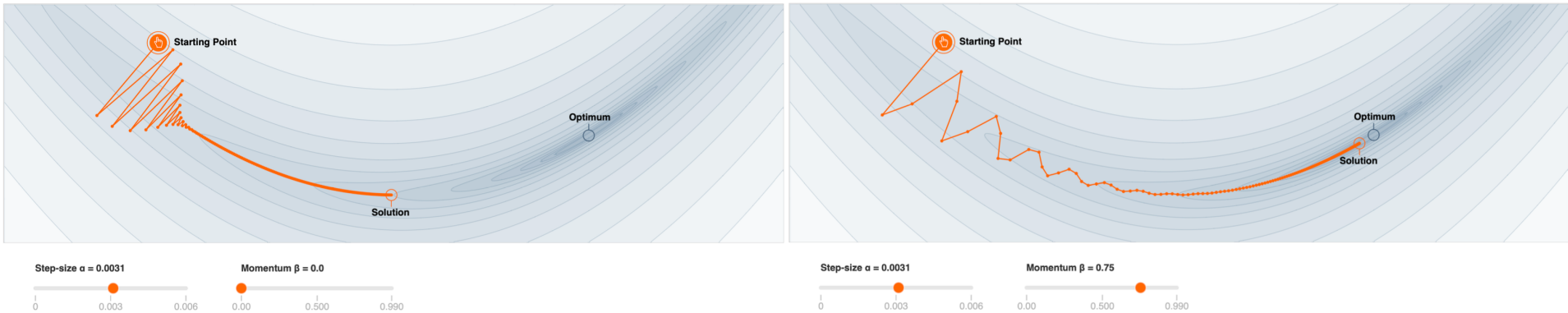
Sử dụng LR decay





SGD with momentum

Thuật toán *SGD with momentum* dựa trên ý tưởng thêm vào động lượng *momentum*để giúp việc huấn luyện mô hình tăng tốc hội tụ và giảm dao động.



(a) Without Momentum (b) With Momentum

Hình 1: Comparison of SGD with and without momentum

SGD with momentum

Key mechanism behind momentum

Cách thức hoạt động của thuật toán *SGD with Momentum*:

1. Move in the (opposite) direction of the gradient: Giống như SGD, ta cũng cập nhật tham số theo hướng ngược lại của gradient $(w_t = w_{t-1} - \alpha g_t)$.
2. Move to the "**averaged**" direction of the last updates: Không giống như SGD, thay vì chỉ dựa trên gradient ở bước t hiện tại, nó kết hợp thêm các Gradient ở các bước phía trước$(\mathbf{b}_t = \mu \mathbf{b}_{t-1} + g_t)$. Vậy nên ta có thể xem $\mathbf{b}_t$ là đại diện cho hướng trung bình mà thuật toán nên di chuyển.

Trong đó, μ là momentum coefficient giúp ghi nhớ từ các cập nhật trước; $\mathbf{b}$ là vector động lượng; g_t gradient tại bước thứ t; α, w lần lượt là learning rate và trọng số mô hình.

Chi tiết thuật toán SGD with momentum I

input : α (lr), w_0 (params), $L(w)$ (loss), μ (momentum)

for $t = 1$ **to** ... **do**

$g_t \leftarrow \nabla_w L_t(w_{t-1})$

if $\mu \neq 0$

if $t > 1$

$\mathbf{b}_t \leftarrow \mu \mathbf{b}_{t-1} + g_t$

else

$\mathbf{b}_t \leftarrow g_t$

$g_t \leftarrow \mathbf{b}_t$

$w_t \leftarrow w_{t-1} - \alpha g_t$

Momentum steps

return $\mathbf{w_t}$

Bước $t = 1$ giống như SGD.

Chi tiết thuật toán SGD with momentum II

input : α (lr), w_0 (params), $L(w)$ (loss), μ (momentum)

for $t = 1$ **to** ... **do**

$g_t \leftarrow \nabla_w L_t(w_{t-1})$

if $\mu \neq 0$

if $t > 1$

$\mathbf{b}_t \leftarrow \mu \mathbf{b}_{t-1} + g_t$

else

$\mathbf{b}_t \leftarrow g_t$

$g_t \leftarrow \mathbf{b}_t$

$w_t \leftarrow w_{t-1} - \alpha g_t$

return $\mathbf{w_t}$

First round where t = 1

(g_t is the gradient at step t)

Chi tiết thuật toán SGD with momentum III

---

input : α (lr), w_0 (params), $L(w)$ (loss), μ (momentum)

---

for $t = 1$ **to** $\ldots$ **do**

$g_t \leftarrow \nabla_w L_t(w_{t-1})$

if $\mu \neq 0$

if $t > 1$

$\mathbf{b}_t \leftarrow \mu \mathbf{b}_{t-1} + g_t$

else

$\mathbf{b}_t \leftarrow g_t$

$g_t \leftarrow \mathbf{b}_t$

$w_t \leftarrow w_{t-1} - \alpha g_t$

All rounds after the initial round

---

return $\mathbf{w_t}$

Chi tiết thuật toán SGD with momentum IV

$\mu = 0.9$

---

input : α (lr), w_0 (params), $L(w)$ (loss), μ (momentum)

---

for $t = 1$ **to** ... **do** assume $t = 1$

  $g_t \leftarrow \nabla_w L_t(w_{t-1})$ compute gradient

  if $\mu \neq 0$

    if $t > 1$

      $\mathbf{b}_t \leftarrow \mu \mathbf{b}_{t-1} + g_t$

    else

      $\mathbf{b}_t \leftarrow g_t$

    $g_t \leftarrow \mathbf{b}_t$ update the gradient

  $w_t \leftarrow w_{t-1} - \alpha g_t$ update weight by negative gradient times learning rate

---

return $\mathbf{w_t}$

Chi tiết thuật toán SGD with momentum V

$\mu = 0.9$

---

input : α (lr), w_0 (params), $L(w)$ (loss), μ (momentum)

---

for $t = 1$ **to** ... **do** assume $t = 2$

  $g_t \leftarrow \nabla_w L_t(w_{t-1})$ compute gradient

  if $\mu \neq 0$

    if $t > 1$

      $\mathbf{b}_t \leftarrow \mu \mathbf{b}_{t-1} + g_t$ 0.9 times previous gradient

    else

      $\mathbf{b}_t \leftarrow g_t$

    $g_t \leftarrow \mathbf{b}_t$

  $w_t \leftarrow w_{t-1} - \alpha g_t$ update weight by negative gradient times learning rate

---

return $\mathbf{w_t}$

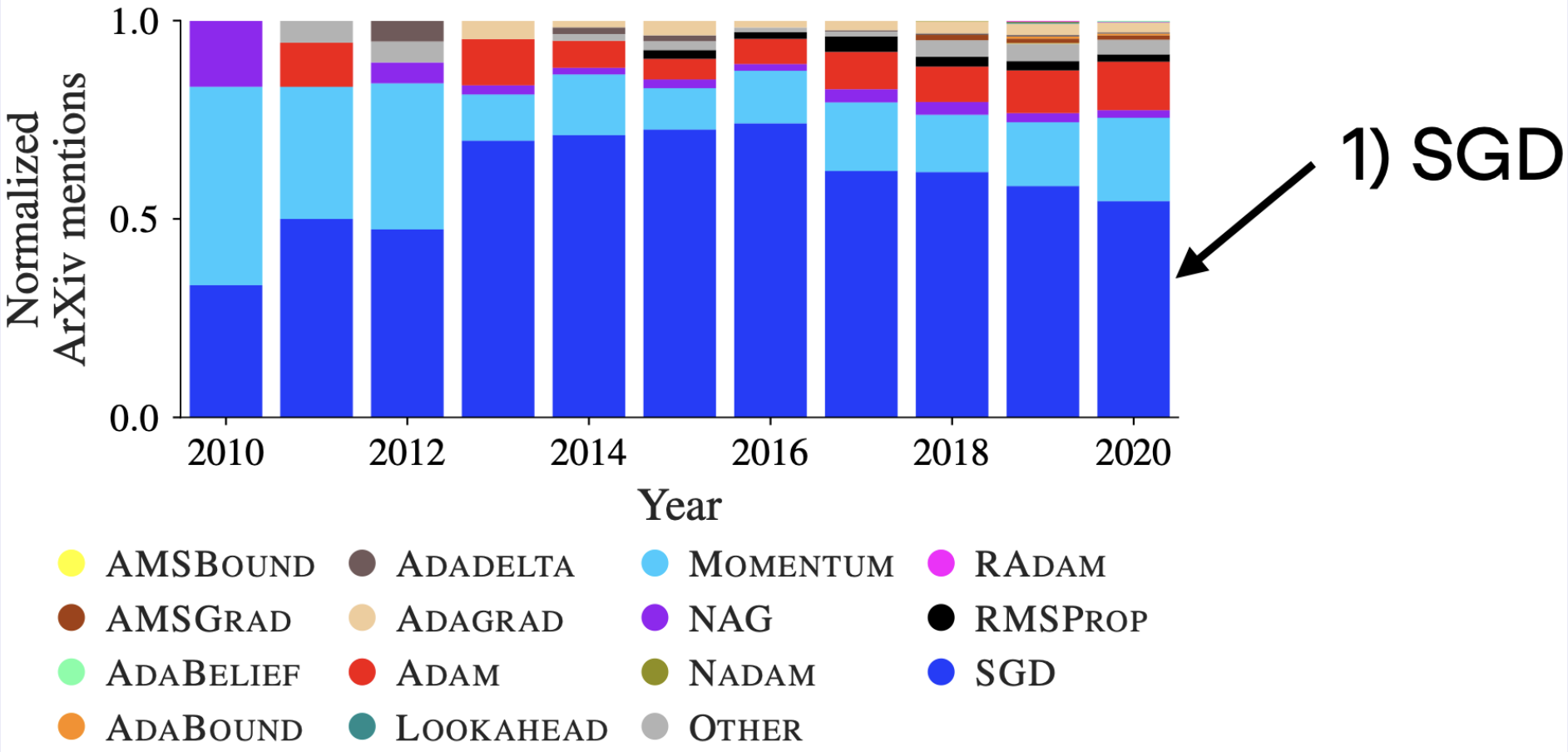
Chi tiết thuật toán SGD with momentum VI

Using momentum in PyTorch

```
class LightningModel(L.LightningModule):
    def __init__(self, model, learning_rate):
        super().__init__()

        self.learning_rate = learning_rate
        self.model = model

        self.save_hyperparameters(ignore=["model"])


    ...

    def configure_optimizers(self):
        optimizer = torch.optim.SGD(self.parameters(), lr=self.learning_rate, momentum=0.9)
        return optimizer
```

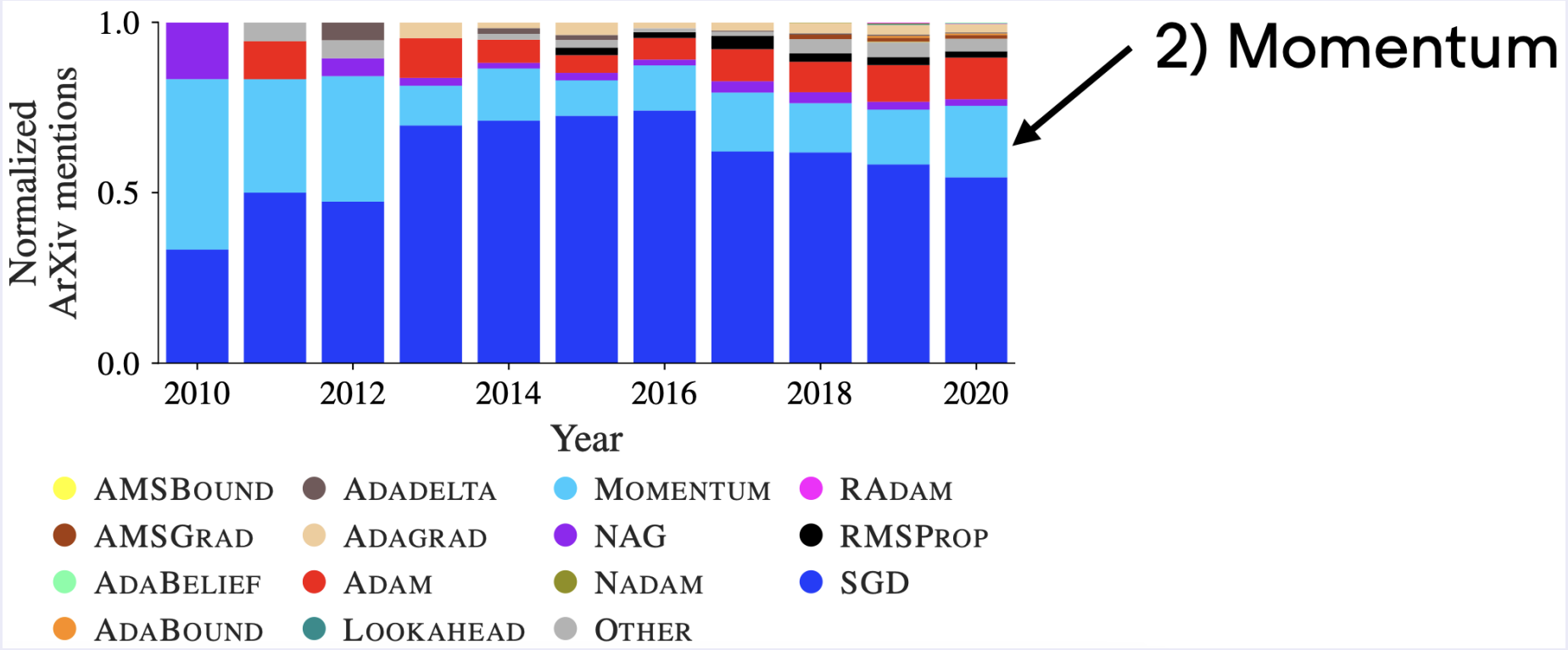
Adaptive Learning Rates

What optimization algorithms do most people prefer?



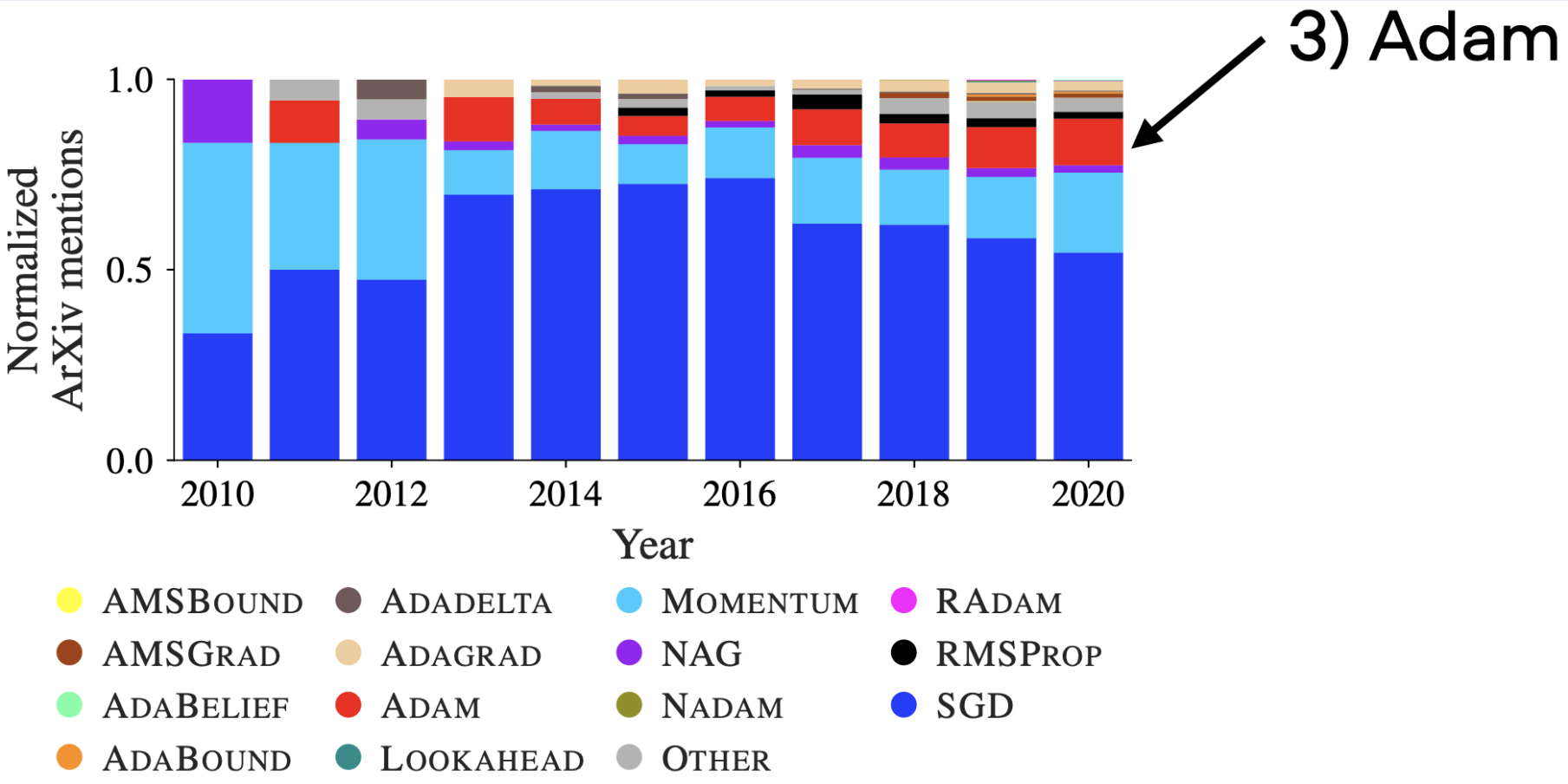
Adaptive Learning Rates

What optimization algorithms do most people prefer?



Adaptive Learning Rates

What optimization algorithms do most people prefer?



Adam: **Ada**ptive learning rates with **m**omentum.

Adaptive Learning Rates

Adam

Adam dựa trên ý tưởng:

1. Tự động điều chỉnh lr cho từng tham số riêng biệt dựa trên gradient thay vì sử dụng một lr cố định chung cho tất cả tham số. Gradient lớn sẽ có lr nhỏ hơn, trong khi gradient nhỏ sẽ có lr lớn hơn.
2. Sử dụng momentum dựa vào các gradient ở các bước trước để giảm dao động và hội tụ.

Adaptive Learning Rates: RMSProp

Giới thiệu RMSProp

Define moving average of the squared gradient of each weight:

$$r = MeanSquare(w_t) = \beta \times MeanSquare(w_{t-1}) + (1-\beta)\left(\frac{\partial L}{\partial w_t}\right)^2,$$

where β is usually between *0.9* and *0.999*. Weight update (similar to SGD, but now scaled):

$$w_t = w_t - \alpha \frac{\frac{\partial L}{\partial w_t}}{\sqrt{r} + \epsilon},$$

where ϵ is small value to prevent divizion by zero. Term $\sqrt{r} + \epsilon$ is root mean square.

Adaptive Learning Rates

Adaptive learning rate via RMSProp

Về bản chất, ý tưởng của thuật toán Adam dựa trên "Adaptive learning rate via RMSProp"

1. RMSProp:

$$w_t = w_t - \alpha \frac{\frac{\partial L}{\partial w_t}}{\sqrt{r} + \epsilon},$$

2. Adam:

$$w_t = w_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{r}} + \epsilon}.$$

Trong đó, $\hat{m}_t = \frac{m_t}{1-\beta_1^t} = \frac{\beta_1 \times m_{t-1} + (1-\beta_1)(\partial L/\partial w_t)}{1-\beta_1^t}$; $\hat{r}_t = \frac{r}{1-\beta_2^t}$

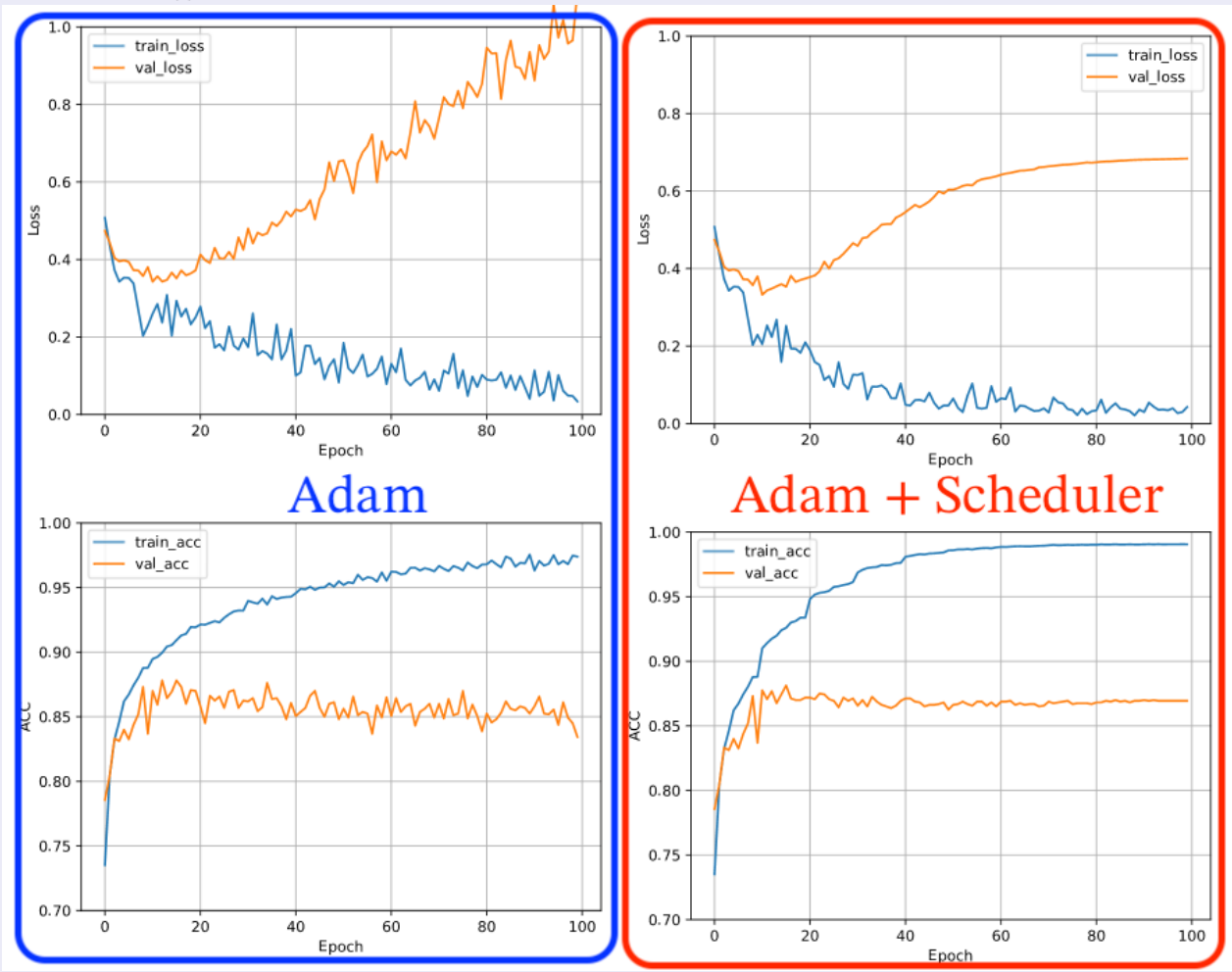
Adaptive Learning Rates

Sử dụng Adam trong PyTorch

```
class LightningModel(L.LightningModule):
    def __init__(self, model, learning_rate):
        super().__init__()

        self.learning_rate = learning_rate
        self.model = model

        self.save_hyperparameters(ignore=["model"])


    ...

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

Adaptive Learning Rates

Trong thực tế, chúng ta có thể kết hợp giữa Adam và Scheduler để thu được kết quả tốt hơn.



Tài liệu tham khảo

Sebastian Raschka, Yuxi (Hayden) Liu, Vahid Mirjalili
Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python (2022). Published by Packt Publishing Ltd, ISBN 978-1-80181-931-2.

Sebastian Raschka
MACHINE LEARNING Q AND AI: 30 Essential Questions and Answers on Machine Learning and AI (2024). ISBN-13: 978-1-7185-0377-9 (ebook).

LightningAI
LightningAI: PyTorch Lightning (2024) .