Lab 7

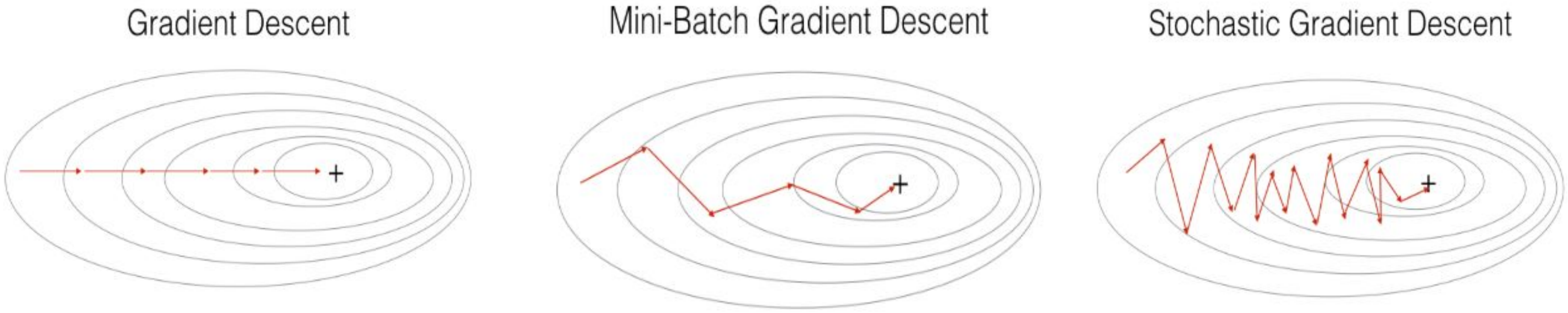
Optimization

BGD, MBGD, SGD

(Batch) Gradient Descent - whole dataset

Mini-Batch Gradient descent - part of the dataset

Stochastic Gradient Descent - one sample



SGD

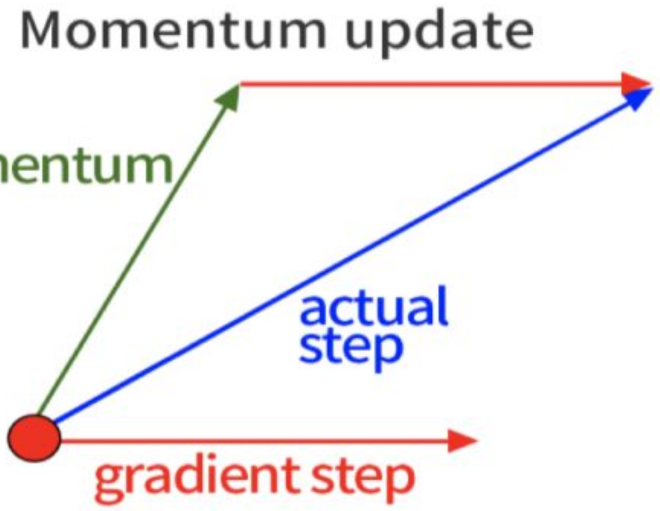
https://pytorch.org/docs/stable/generated/torch.optim.SGD.html

optimizer = torch.optim.SGD(model.parameters(), lr=0.1)

torch.optim.SGD can be Gradient Descent, Mini-Batch Gradient descent or Stochastic Gradient Descent depend on the batch size.

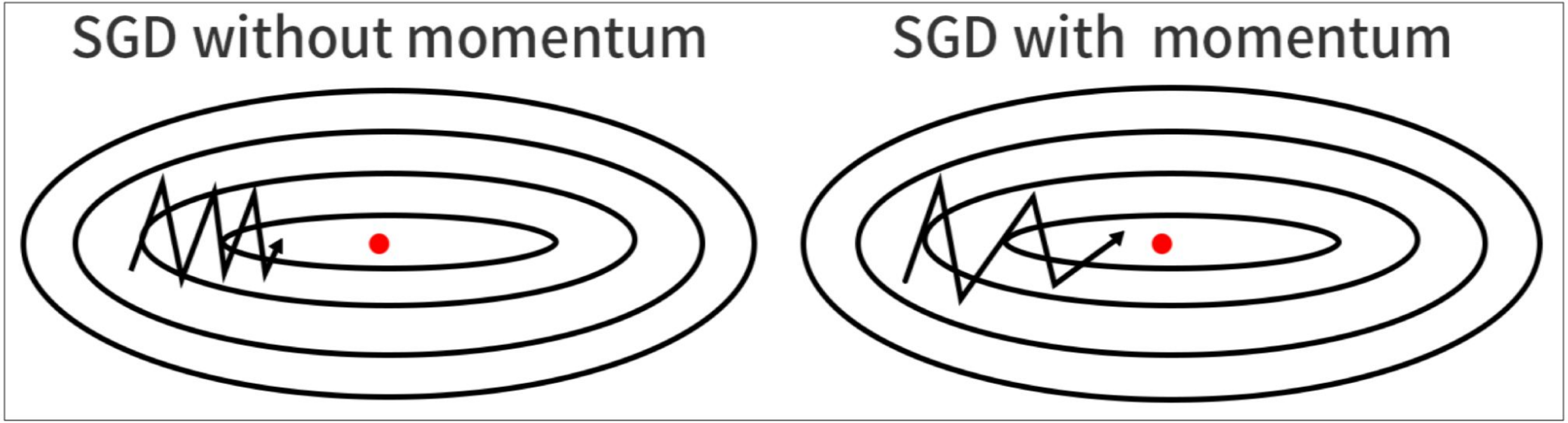
SGD with momentum

optimizer = torch.optim.SGD(model.parameters(), lr=0.1, momentum=0.9)
actual step = momentum x previous step + gradient step



https://iopscience.iop.org/article/10.1088/1742-6596/1229/1/012046/pdf

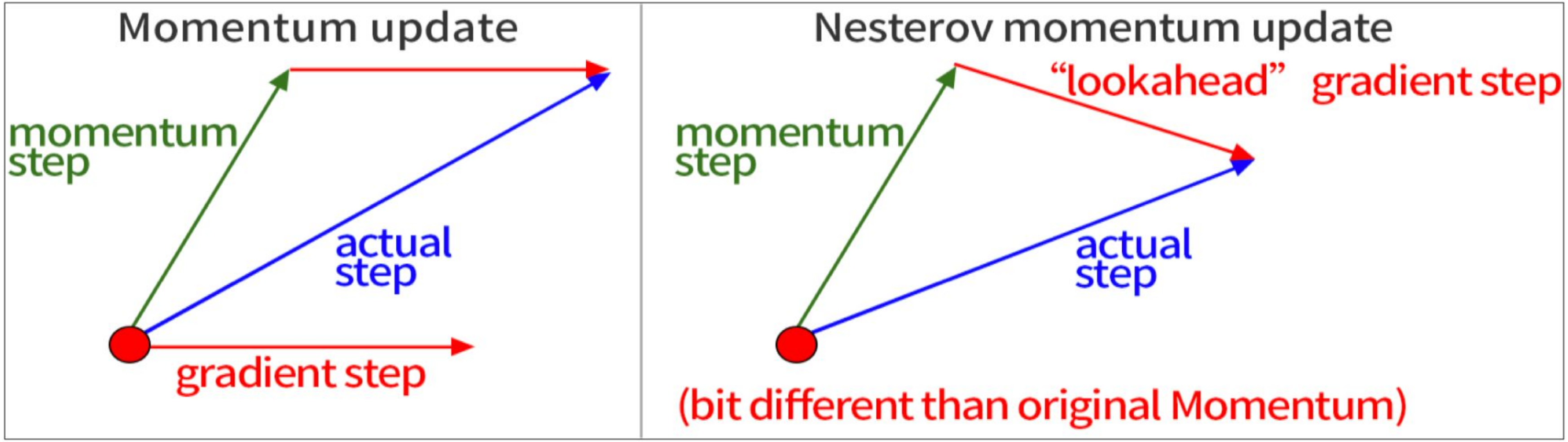
SGD with momentum



https://iopscience.iop.org/article/10.1088/1742-6596/1229/1/012046/pdf

SGD with Nesterov momentum

optimizer = torch.optim.SGD(model.parameters(), lr=0.1, momentum=0.9, nesterov=True)
actual step: momentum x previous step, then gradient step at that new position



https://iopscience.iop.org/article/10.1088/1742-6596/1229/1/012046/pdf

AdaGrad

$$V_t = \sum g_t^2$$

This is an Adaptive Subgradient Method. Simplified formula:

$$\eta_t = \frac{\eta}{\varepsilon + \sqrt{V_t}}$$

Gradient estimate Vt is the sum of squares of all previous gradient

ε is a very small number to avoid 0 denominator

η is learning rate, ηt is current learning rate

Some parameters are frequently updated (large Vt), they may fit the data very well, so we decrease the learning rate for these parameters (small ηt).

Same, rare update -> small Vt -> large learning rate ηt .

Problem: gradient estimate Vt keeps increasing, learning rate may get close to 0 before it arrives the minimum.

RMSprop

RMSprop mainly uses the gradient in several previous steps.

Gradient estimate Vt won't keep increasing, then we won't get 0 learning rate.

RMSprop

$$V_t = \beta_2 V_{t-1} + (1 - \beta_2) g_t^2$$

$$\eta_t = \frac{\eta}{\varepsilon + \sqrt{V_t}}$$

AdaGrad

$$V_t = \sum g_t^2$$

$$\eta_t = \frac{\eta}{\varepsilon + \sqrt{V_t}}$$

Adam

Adam = SGD momentum + RMSprop

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$

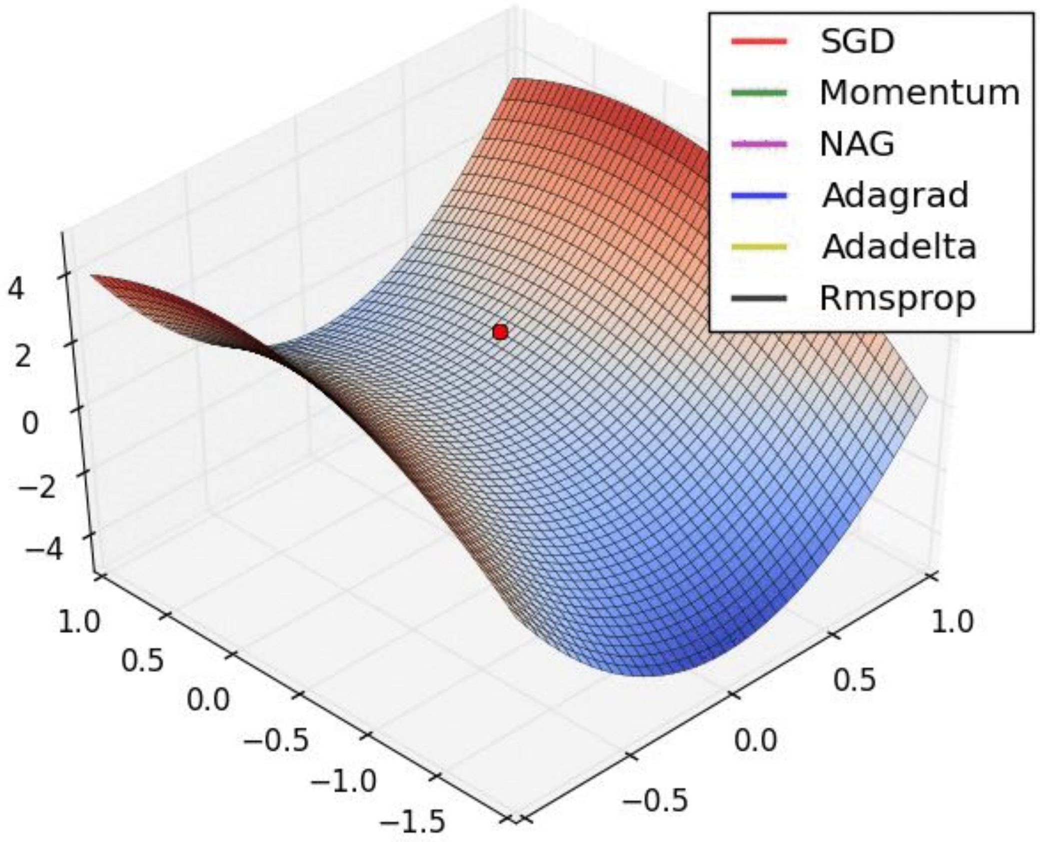
$$\bar{m_t} = \frac{m_t}{1 - \beta_1^t}$$

$$\theta_{t+1} = \theta_t - \frac{\alpha \bar{m}_t}{\sqrt{v_t \bar{+} \varepsilon}}$$

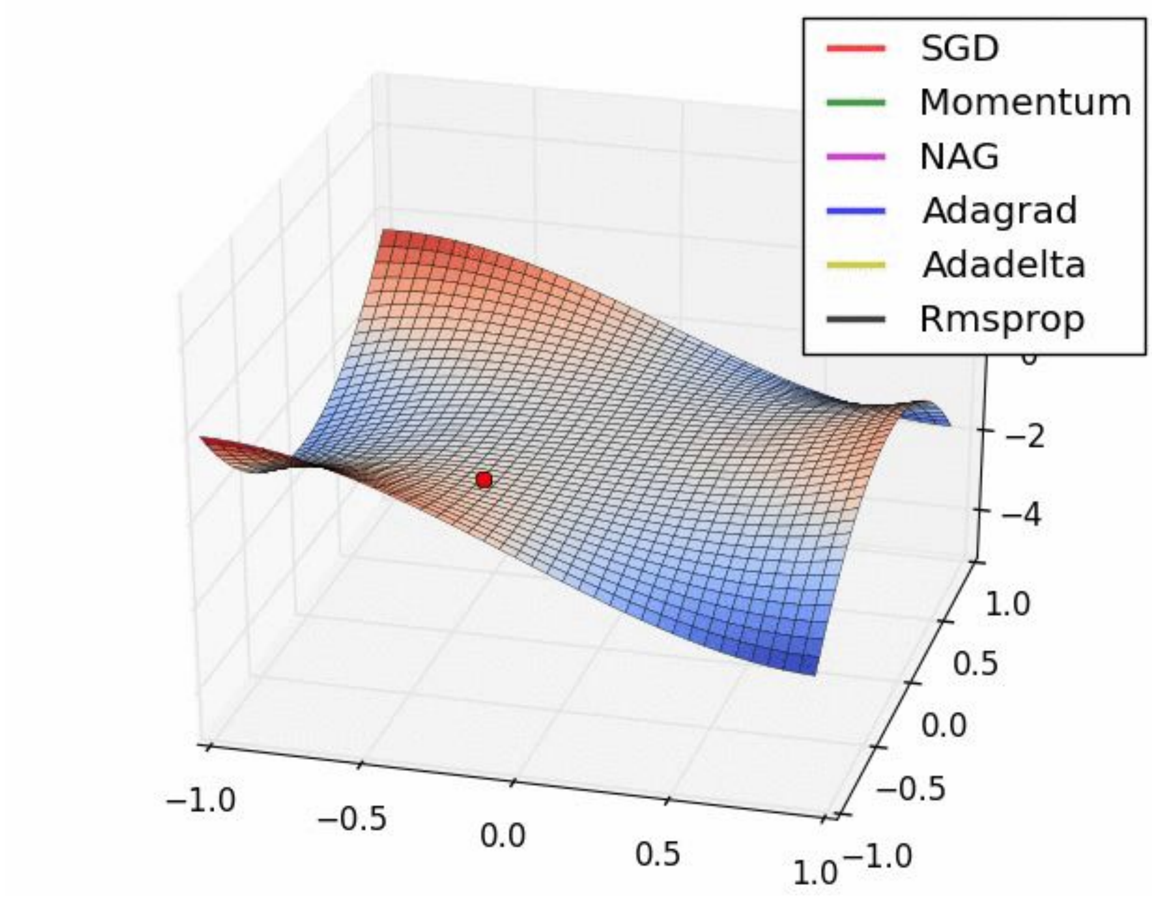
$$V_t = \beta_2 V_{t-1} + (1 - \beta_2) g_t^2$$

$$\bar{v_t} = \frac{v_t}{1 - \beta_2^t}$$

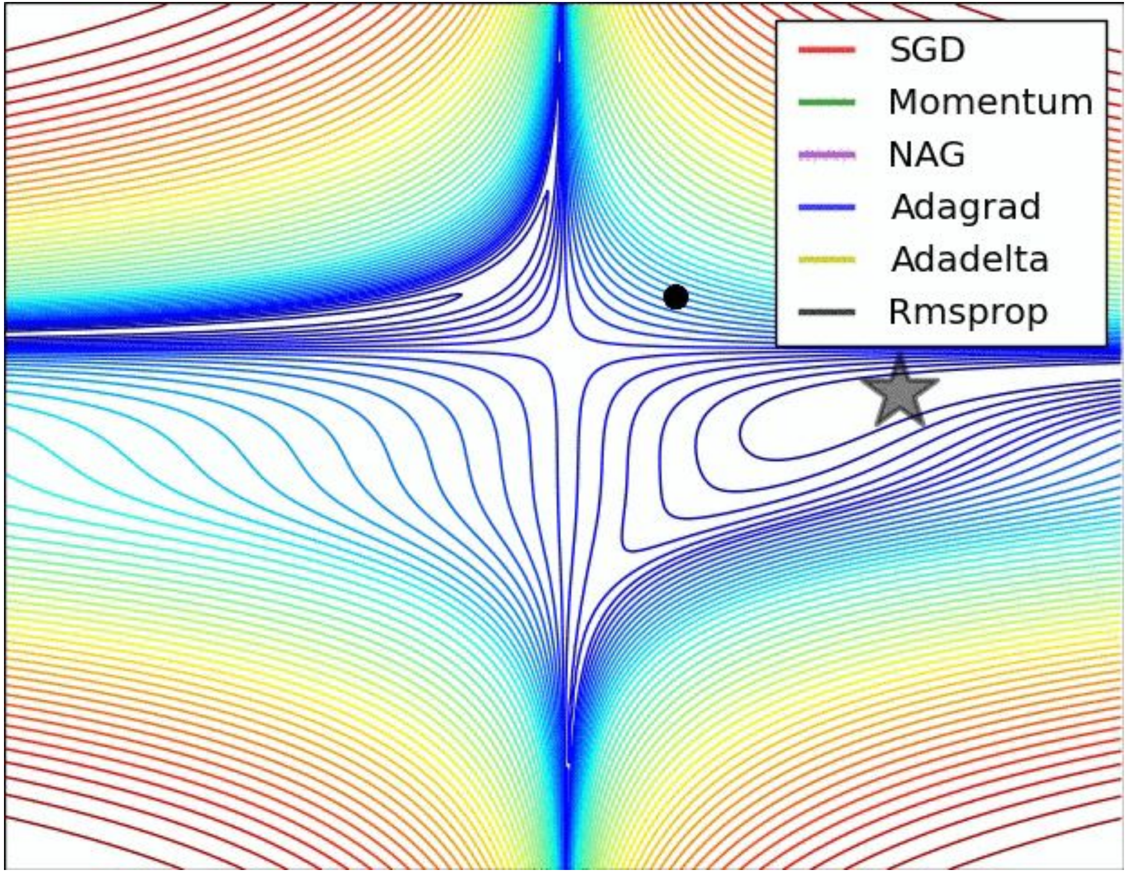
SGD and momentum stuck at the saddle point.



SGD momentum has increasing accelerated speed.



SGD and momentum converge slower.



Practical Application

- Adam converges fast.
- SGD usually ends up with better result, but takes longer to train.
- Adam+SGD: use Adam first, then use SGD for fine-tuning.

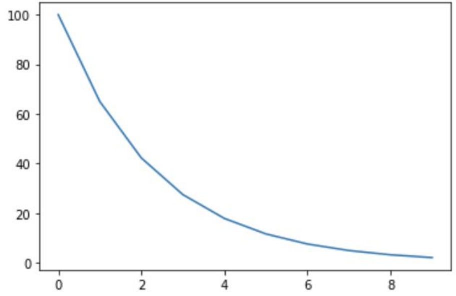
Scheduler: Learning Rate Scheduling

1. LAMBDA LR

Sets the learning rate of each parameter group to the initial lr times a given function. When last_epoch=-1, sets initial lr as lr.

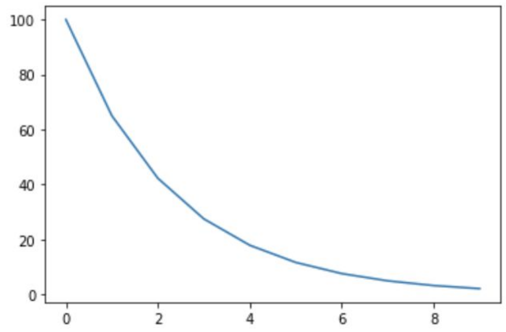
$$lr_{\text{epoch}} = lr_{\text{initial}} * Lambda(epoch)$$

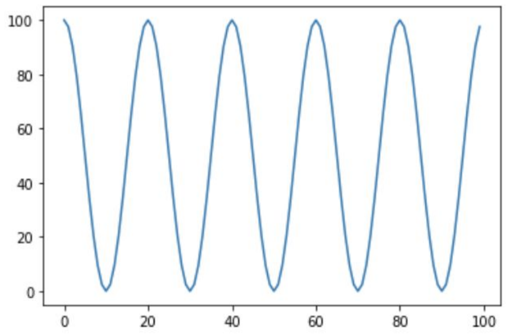
```
model = torch.nn.Linear(2, 1)
optimizer = torch.optim.SGD(model.parameters(), lr=100)
lambda1 = lambda epoch: 0.65 ** epoch
scheduler = torch.optim.lr_scheduler.LambdaLR(optimizer, lr_lambda=lambda1)


lrs = []

for i in range(10):
    optimizer.step()
    lrs.append(optimizer.param_groups[0]["lr"])
#     print("Factor = ", round(0.65 ** i,3)," , Learning Rate = ",round(optimizer.param_groups[0]
["lr"],3))
    scheduler.step()

plt.plot(range(10),lrs)
```

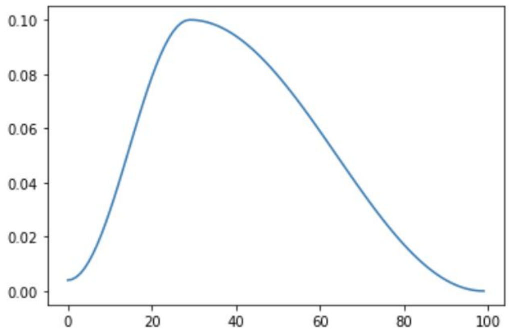


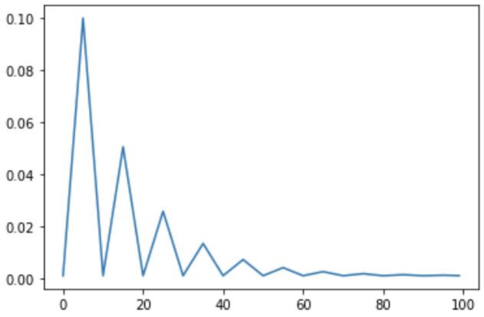
Scheduler

https://www.kaggle.com/code/isbhargav/guide-to-pytorch-learning-rate-scheduling/notebook

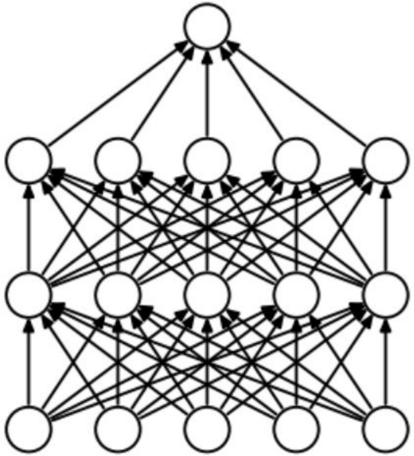




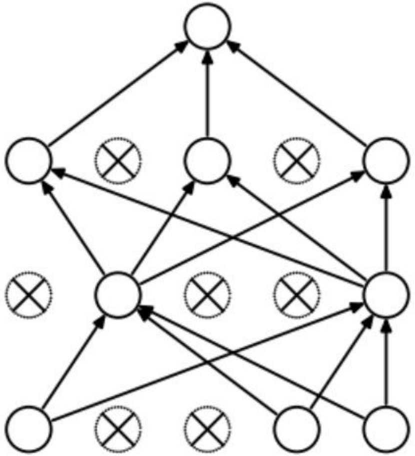




Dropout:
https://towardsdatascience.com/machine-learning-part-20-dropout-keras-layers-explained-8c9f6dc4c9ab



(a) Standard Neural Net



(b) After applying dropout.

Batch Normalization: Normalize among N (batch), H, W, not including C (channel)