CSPB 3202 Artificial Intelligence

Optimization and Tips for Neural Network Training

Geena Kim

Outline and Keywords

Optimization methods

Stochastic Gradient Descent

- Learning rate
- Momentum
- Decay

Advanced Gradient Descent methods

- Learning rate scheduling
- Nestrov momentum

- AdaGrad
- AdaDelta
- RMSprop
- Adam

Tips for neural network training

- General tips for overfitting
- Regularization methods
  - Dropout
  - Batch normalization

Gradient Descent

Optimization Goal

Find a set of (optimized) weights which minimize the error (or loss function) at the output

Weight update rule

$$\frac{\partial \mathcal{L}}{\partial W^L_{nm}}$$

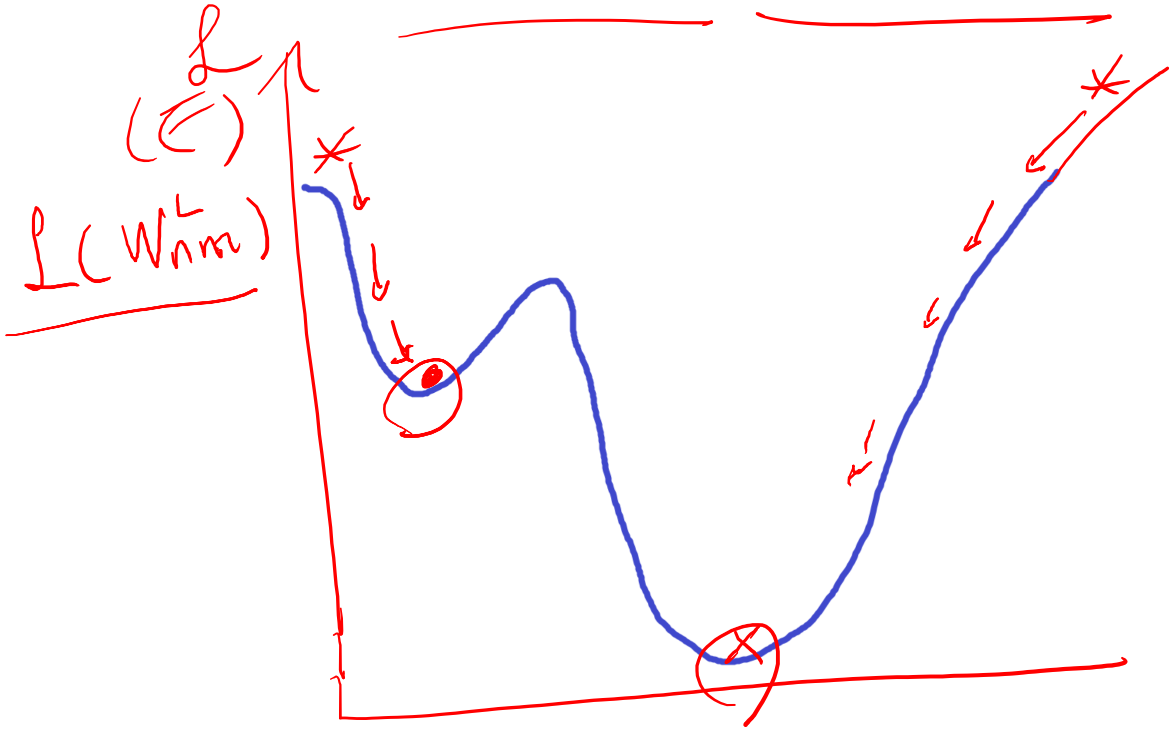
$$W'^{L}_{nm} \leftarrow W^{L}_{nm} - \alpha * \delta W^{L}_{nm}$$

n+1

nth

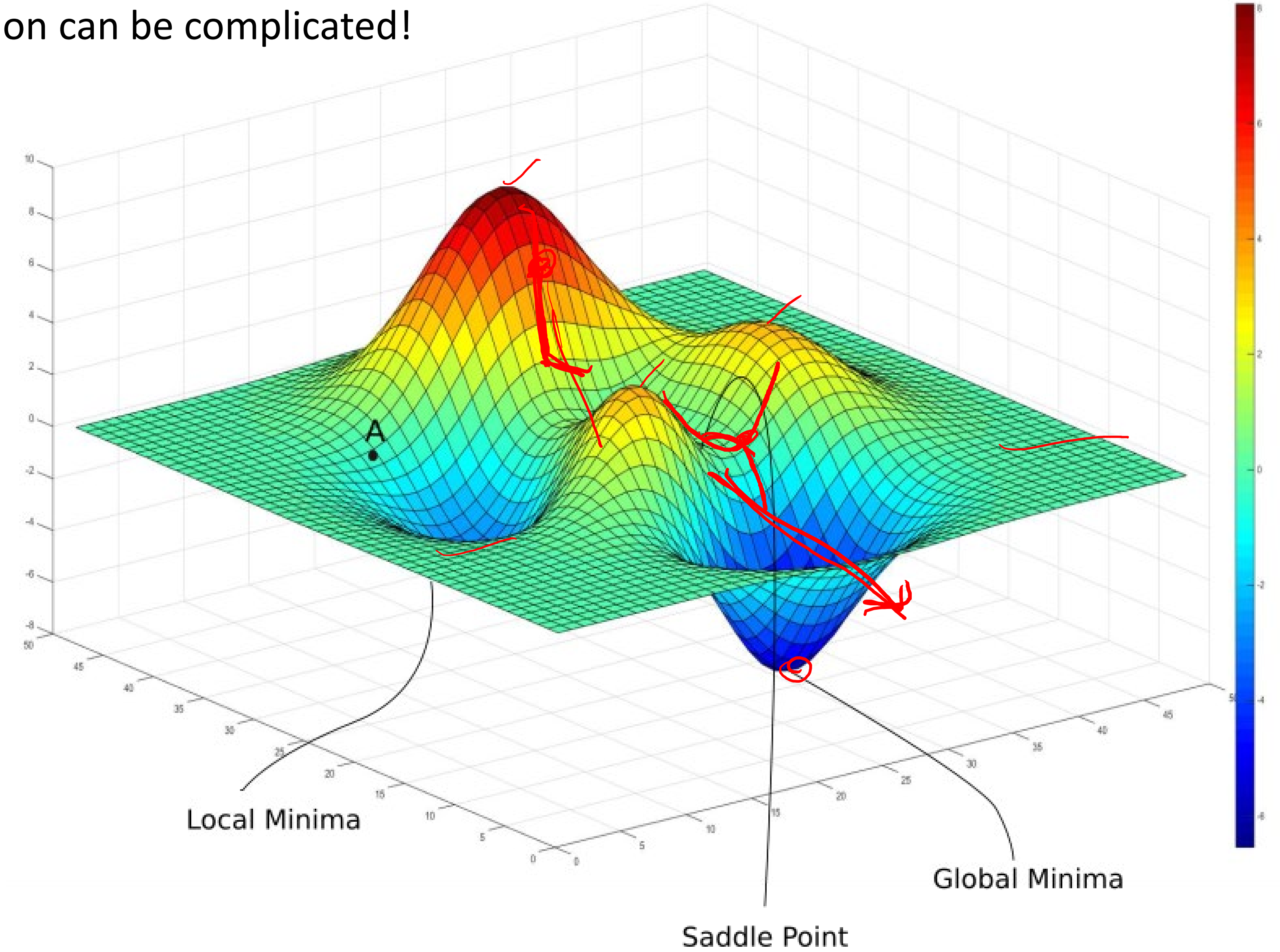
$$W_{ij} \leftarrow W_{ij} - \alpha \frac{\partial \mathcal{L}}{\partial W_{ij}}$$

Global minimum vs. local minimum



Gradient Descent

Error surface in the multi dimension can be complicated!



also… cliffs and plateaus.

Stochastic Gradient Descent

How many training samples at a time do we include to calculate the error?

$\mathcal{L}(w_{ij}, x_k)$

#1 . . . 10k m

10

$$W_{ij} \leftarrow W_{ij} - \alpha \frac{\partial \mathcal{L}}{\partial W_{ij}}$$

Practically we use mini batches

bs = 1

Training speed and accuracy vs. minibatch size

Stochastic Gradient Descent

With decreasing learning rate (Learning rate scheduling)

Algorithm 8.1 Stochastic gradient descent (SGD) update

Require: Learning rate schedule $\epsilon_1, \epsilon_2, \ldots$

Require: Initial parameter $\boldsymbol{\theta}$ $\leftrightarrow$ w

$k \leftarrow 1$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$ $\boxed{\varepsilon_k}$

$k \leftarrow k + 1$

end while

---

warning- different notations used (from deeplearningbook.org)

Stochastic Gradient Descent with momentum

SGD with learning rate alone is slow to converge

Adding a momentum (moving average) can make it faster

$$\boldsymbol{v} \leftarrow \boxed{\alpha}\boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right),$$

<1

$- \underset{(\alpha)}{\varepsilon} \frac{\partial \mathcal{L}}{\partial \theta}$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}.$$

warning- different notations used (from deeplearningbook.org)

```
tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.0, nesterov=False, name='SGD', **kwargs
)
```

l_0

SGD tuning parameters

```
tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.0, nesterov=False, name='SGD', **kwargs
)
```

Popular options to tweak

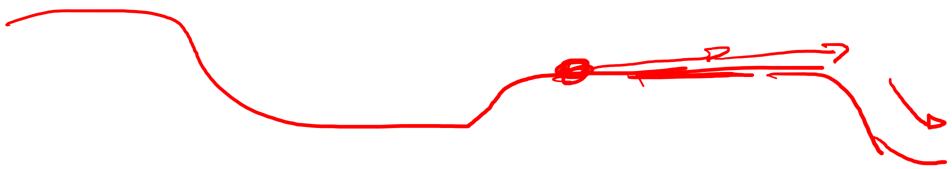
- learning_rate: the base learning rate
- momentum
- decay
- nestrov
- (advanced) callback

Stochastic Gradient Descent with momentum

SGD with learning rate alone is slow to converge

Adding a momentum (moving average of a weight) can make it faster

$$
\begin{aligned}
\boldsymbol{v} &\leftarrow \alpha \boldsymbol{v} - \cancelto{0}{\epsilon \nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right)}, \\
\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \boldsymbol{v}.
\end{aligned}
$$



** see what happens when the gradient is 0 (on plateau)

warning- different notations used (from deeplearningbook.org)

Stochastic Gradient Descent with decay

Learning rate scheduling using decay

For iteration k (epoch)

$$\epsilon_k = (1-\alpha)\epsilon_0 + \alpha\epsilon_\tau \qquad \alpha = \frac{k}{\tau}$$

$\frac{k}{\tau}$

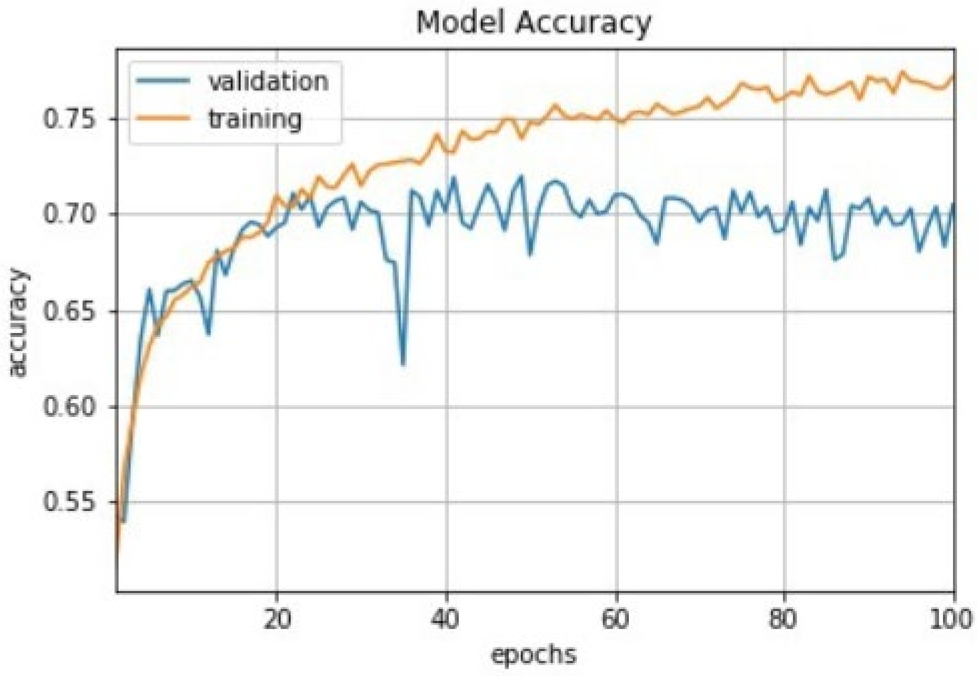
** In the algorithm pseudocode k is for step (each mini batch),
and decay learning rate by step,
but normally we decrease learning rate each epoch

warning- different notations used (from deeplearningbook.org)

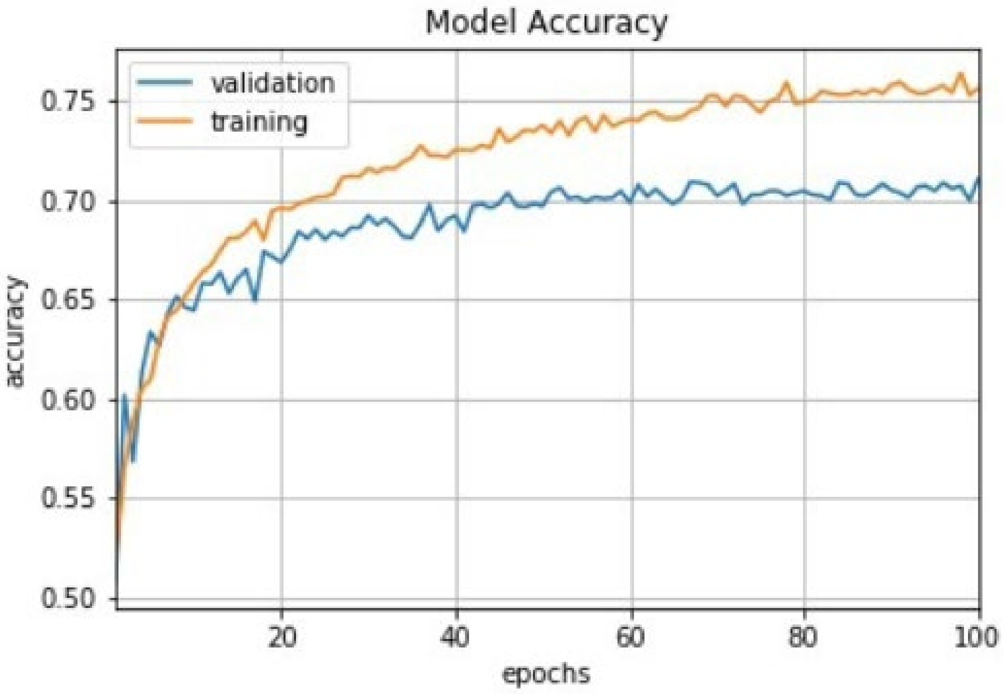
Learning rate scheduling

```
tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.0, nesterov=False, name='SGD', **kwargs
)
```

learning_rate=0.1,
momentum=0, decay=0, nestrov=False



learning_reate=0.1,
momentum=0.8, decay=learning_rate/epochs



Learning rate scheduling (custom)

```
tf.keras.callbacks.LearningRateScheduler(
    schedule, verbose=0
)
```

Ex2

drop lr by half every 10 epochs

```
def step_decay(epoch):
   initial_lrate = 0.1
   drop = 0.5
   epochs_drop = 10.0
   lrate = initial_lrate * math.pow(drop,
           math.floor((1+epoch)/epochs_drop))
   return lrate


lrate = LearningRateScheduler(step_decay)
```

Ex1

```
# This function keeps the learning rate at 0.001 for the first ten epochs
# and decreases it exponentially after that.
def scheduler(epoch):
  if epoch < 10:
    return 0.001
  else:
    return 0.001 * tf.math.exp(0.1 * (10 - epoch))

callback = tf.keras.callbacks.LearningRateScheduler(scheduler)
model.fit(data, labels, epochs=100, callbacks=[callback],
          validation_data=(val_data, val_labels))
```

Learning rate scheduling (custom)

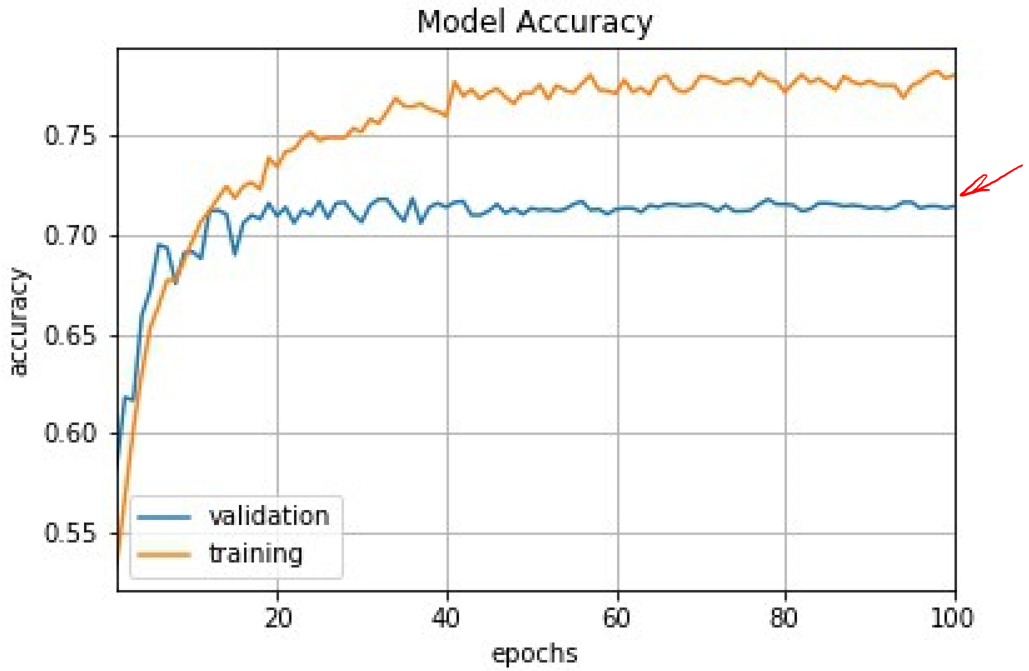
```
tf.keras.callbacks.LearningRateScheduler(
    schedule, verbose=0
)
```

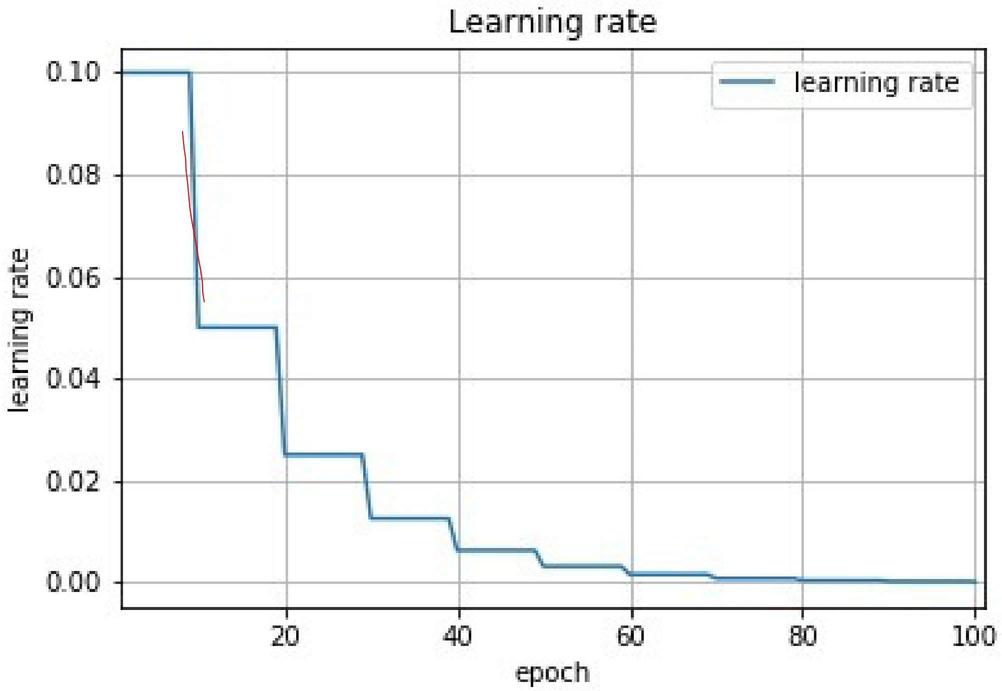
Ex2

```
def step_decay(epoch):
   initial_lrate = 0.1
   drop = 0.5
   epochs_drop = 10.0
   lrate = initial_lrate * math.pow(drop,
           math.floor((1+epoch)/epochs_drop))
   return lrate

lrate = LearningRateScheduler(step_decay)
```

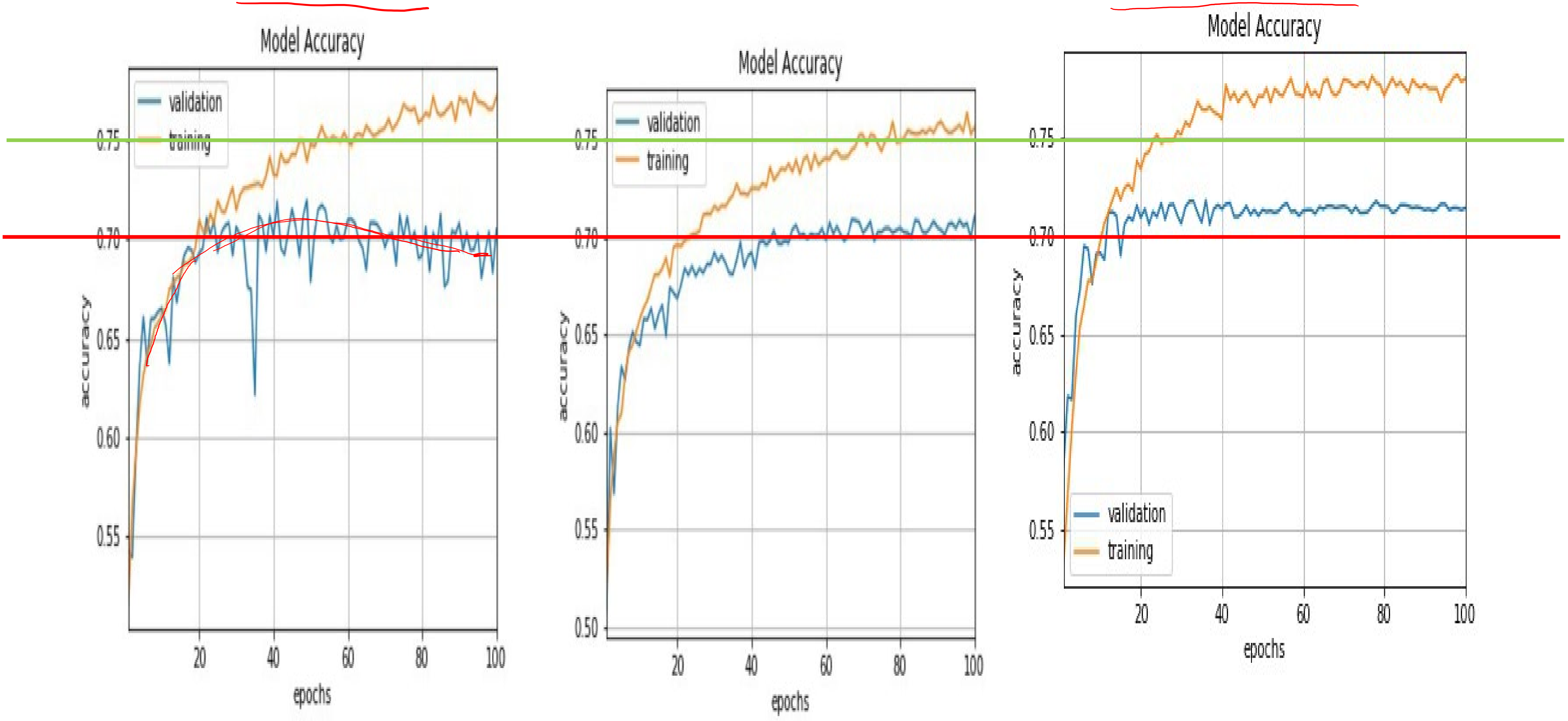
drop lr by half every 10 epochs





comparison

Base (fixed lr) | with momentum and decay | custom learning rate schedule (step drop)



Nestrov momentum

Nestrov momentum does early correction on gradient
It's supposed to make converge faster, but on SGD it doesn't do much

Regular momentum

$$\boldsymbol{v} \leftarrow \boxed{\alpha \boldsymbol{v}} \boxed{- \epsilon \nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right)},$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}.$$

w

Nestrov momentum

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left[\frac{1}{m} \sum_{i=1}^{m} L\left(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boxed{\boldsymbol{\theta} + \alpha \boldsymbol{v}}), \boldsymbol{y}^{(i)} \right) \right],$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v},$$

warning- different notations used (from deeplearningbook.org)

Advanced optimization

(Advanced)

Adagrad

learning rate is normalized by the sqrt of the total sum of the gradient

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

An overview of gradient descent optimization algorithms
https://arxiv.org/pdf/1609.04747.pdf

warning- different notations used

Advanced optimization

Adadelta

learning rate is normalized by the RMS of the gradient
Weight change is proportional to the RMS ratio

$$\Delta\theta_t = -\frac{\eta}{RMS[g]_t} g_t$$

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

An overview of gradient descent optimization algorithms
https://arxiv.org/pdf/1609.04747.pdf

warning- different notations used

Advanced optimization

RMSprop

Variant of Adadelta
RMSprop takes a moving average when it calculate the RMS of the gradient

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

An overview of gradient descent optimization algorithms
https://arxiv.org/pdf/1609.04747.pdf

warning- different notations used

Advanced optimization

$w \leftarrow w - \eta \cdot g$

Adaptive Moment Estimation (Adam)

$w - \frac{\eta}{\sqrt{E[g^2] + \epsilon}} g$

Mimics momentum for gradient and gradient-squared

m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients

momentum

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

lr

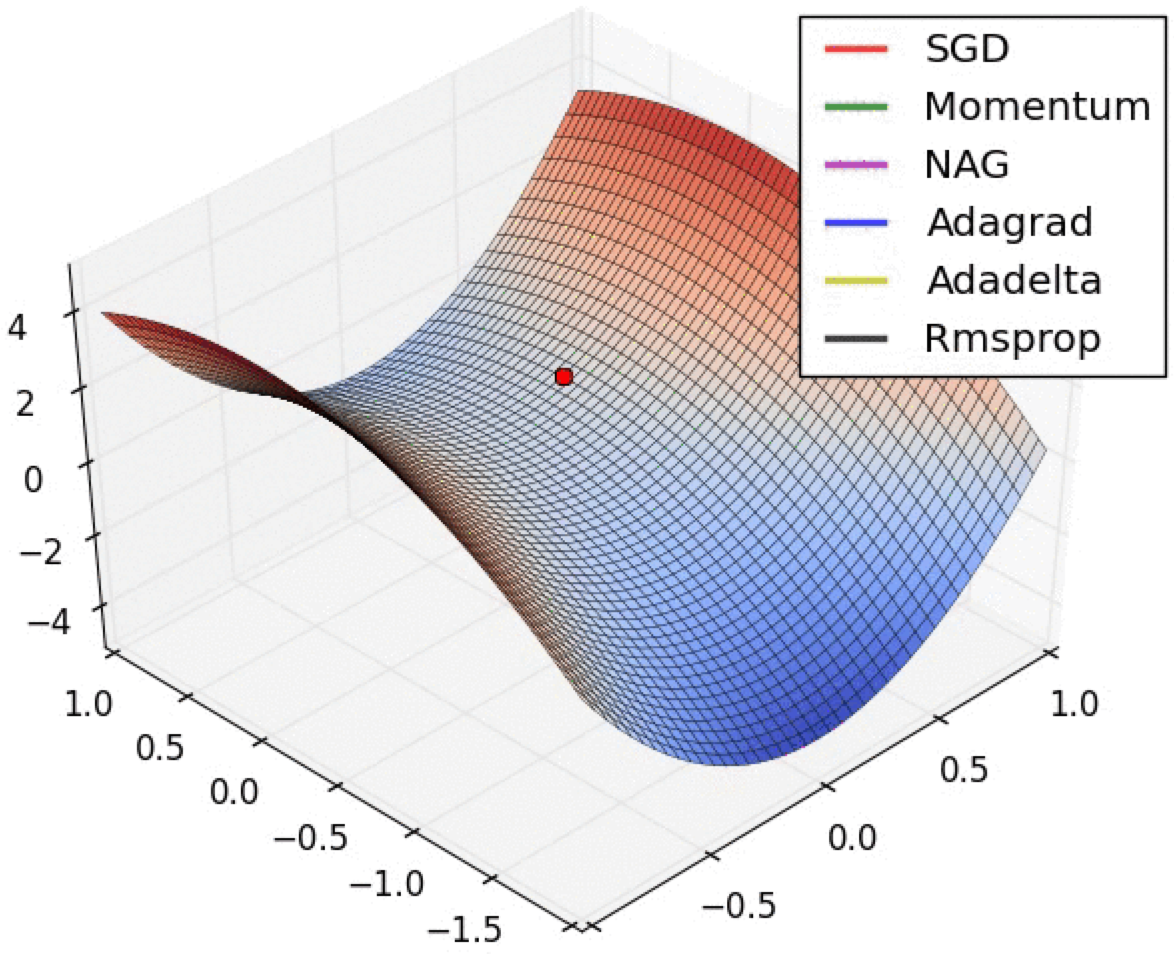
$$\boxed{\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t}$$

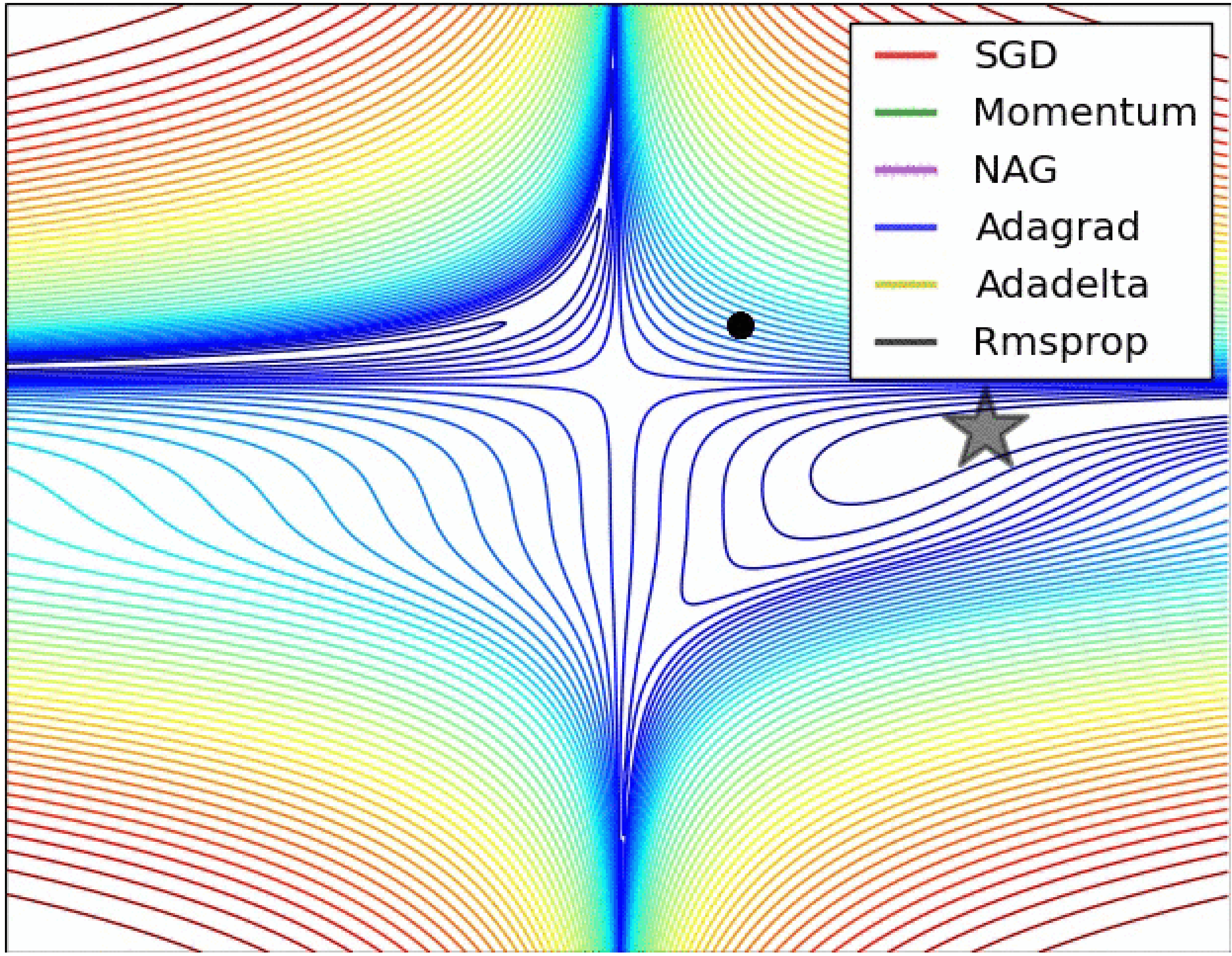
RMS g $E(g^2)$

An overview of gradient descent optimization algorithms
https://arxiv.org/pdf/1609.04747.pdf

warning- different notations used

Advanced optimization





animated image source: https://imgur.com/a/Hqolp

Tips for training NN

Monitor overfitting as epoch goes

Train hyperparameter tuning : learning rate and other hyperparams

Architecture hyperparameter tuning : NN architecture, # layers, # neurons, activation ft, etc

Try different optimization methods

Regularization: Dropout and Batch Normalization, or add L1/L2 reg on the loss

$\sum |\theta|$, $\sum \theta^2$

Monitoring Overfitting in Training

Dataset split
Train / Validation / Test

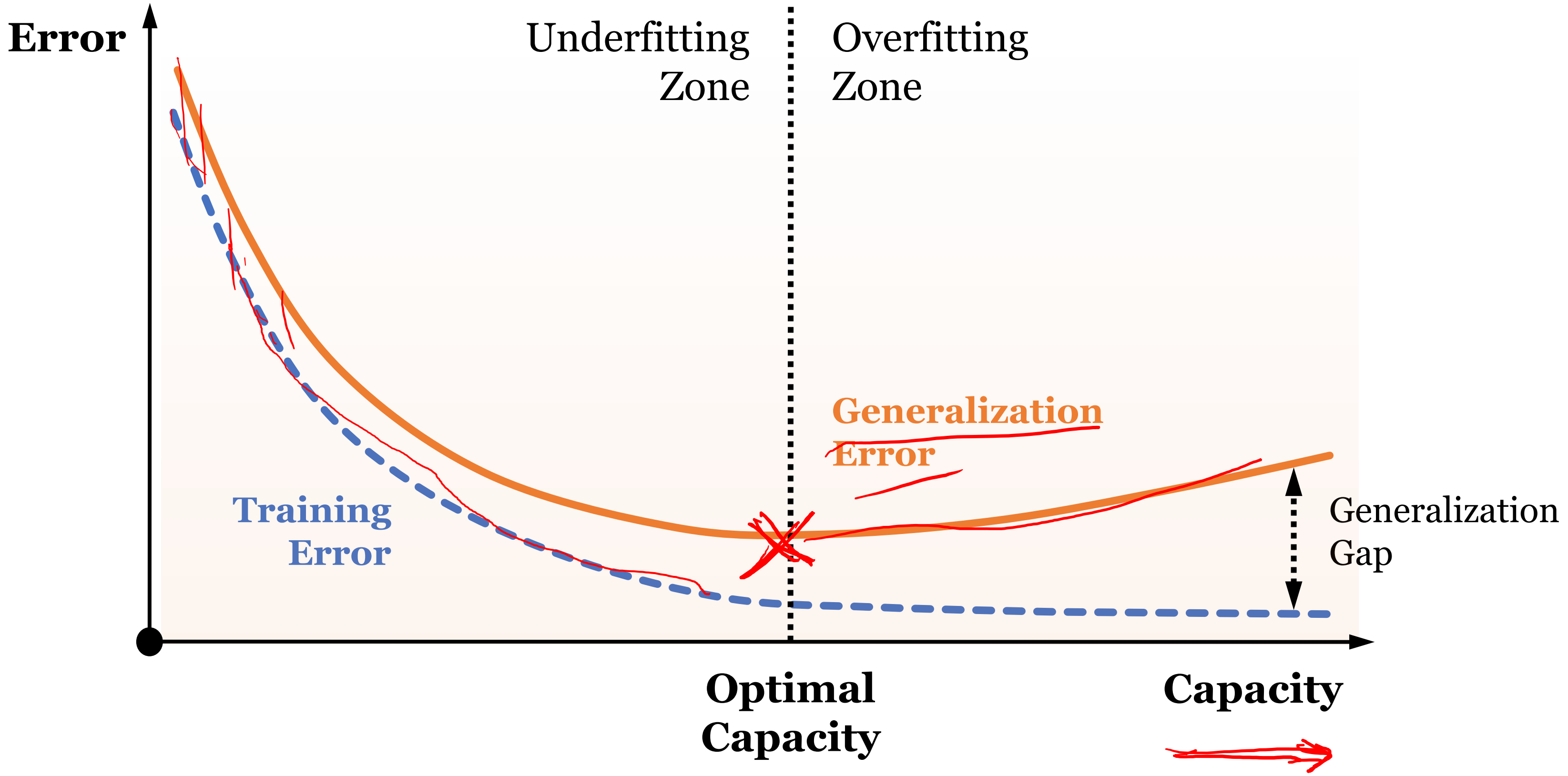
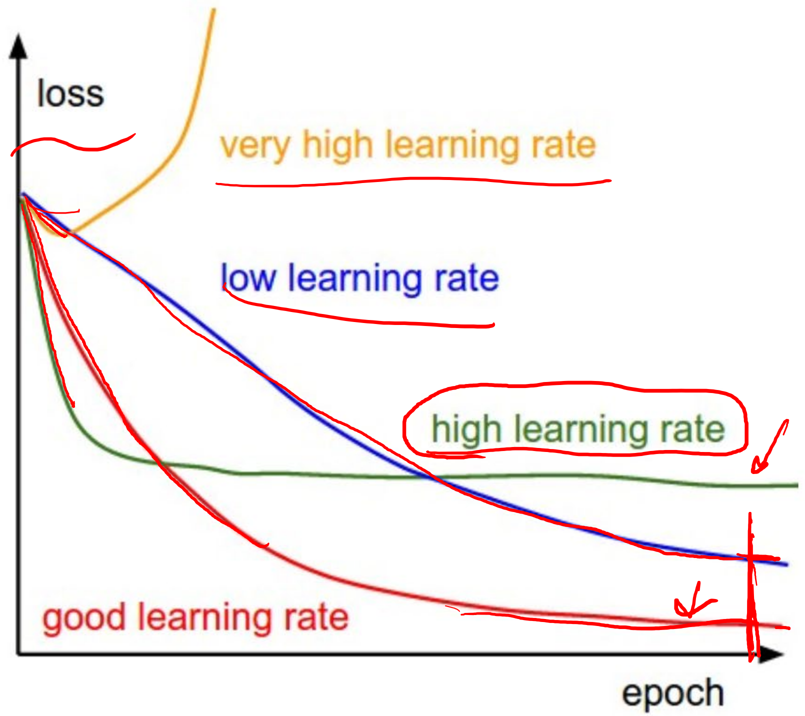
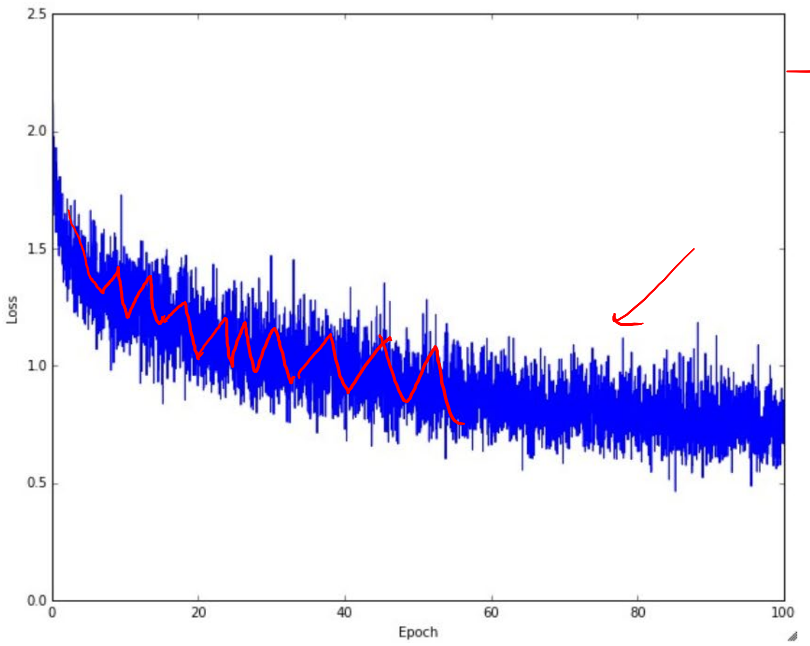
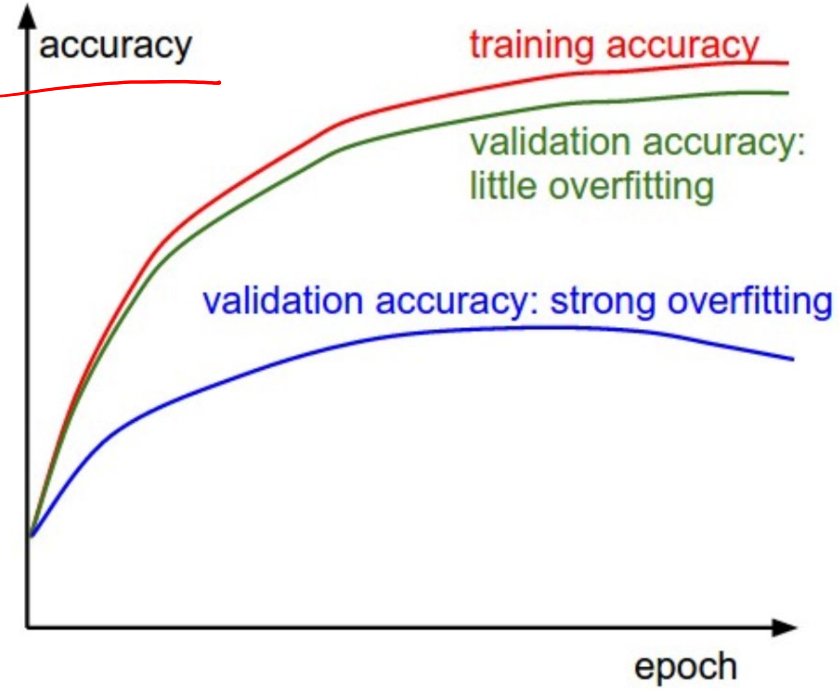


Diagram credit: Fei-Fei Li

Monitoring Overfitting in Training



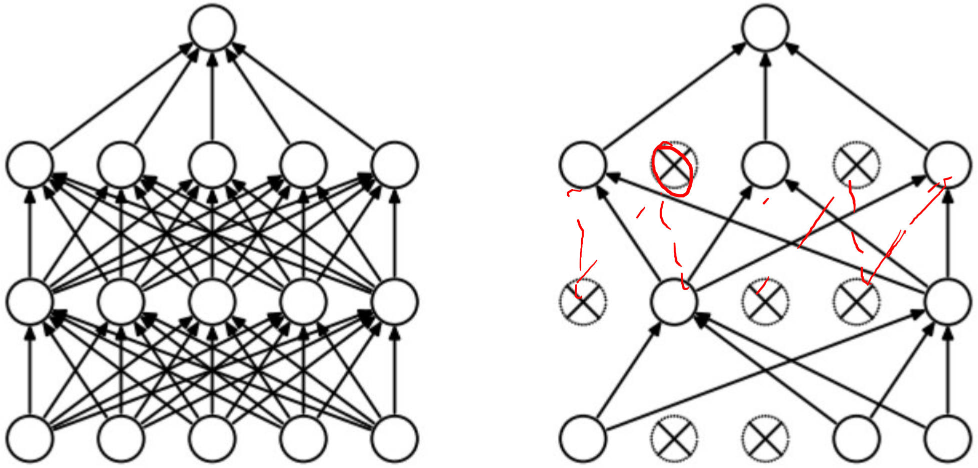




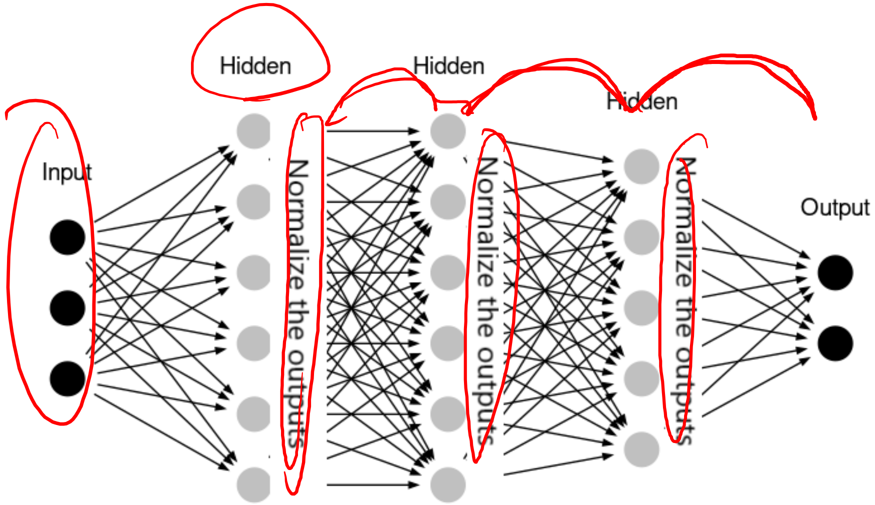
http://cs231n.github.io/neural-networks-3/

Ways to reduce overfitting

Dropout [1]


(a) Standard Neural Net (b) After applying dropout.

Batch normalization [2]



[1] http://jmlr.csail.mit.edu/papers/volume15/srivastava14a/srivastava14a.pdf

[2] https://arxiv.org/pdf/1502.03167.pdf