CSCI4622 Machine Learning

2020 Spring

Optimization and Tips for Neural Network Training

Geena Kim

*Some of the slide/diagram adopted from CMU deep learning course

Gradient Descent

Optimization Goal

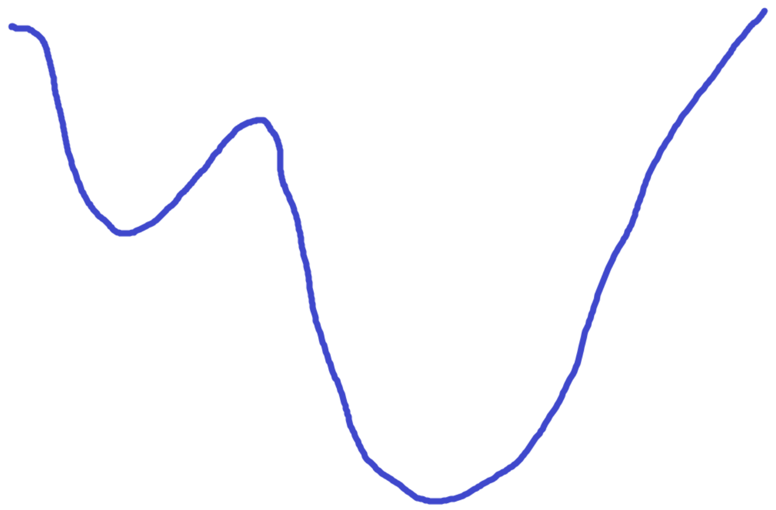
Find a set of (optimized) weights which minimize the error (or loss function) at the output

Weight update rule

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

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

Global minimum vs. local minimum



Stochastic Gradient Descent

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

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

Practically we use mini batches

lep

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 $\boldsymbol{\theta}$
$k \leftarrow 1$
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: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$
$k \leftarrow k + 1$
end while

warning- different notations used (from deeplearningbook.org)

Stochastic Gradient Descent

(notations)

learning rate — α , η , ε

momentum : ν , v

weights : w , θ

loss ft : $\mathcal{L}$, J , E

warning- different notations used (from deeplearningbook.org)

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

$w \leftarrow w - \alpha \nabla \ell$

0

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

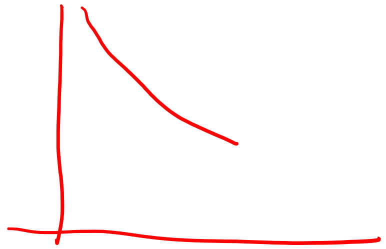
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}.$$

** 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)

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

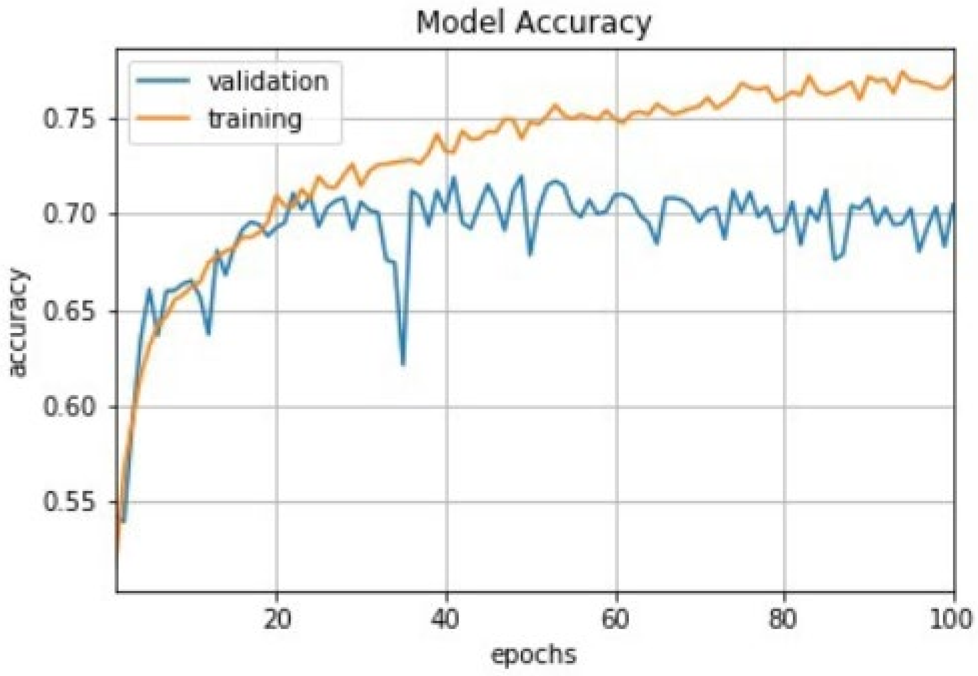
** In the algorithm pseudocode k is for step (each mini batch),
and decay learning rate by step,
but normally we decrease learning rate each epoch

warning- different notations used (from deeplearningbook.org)

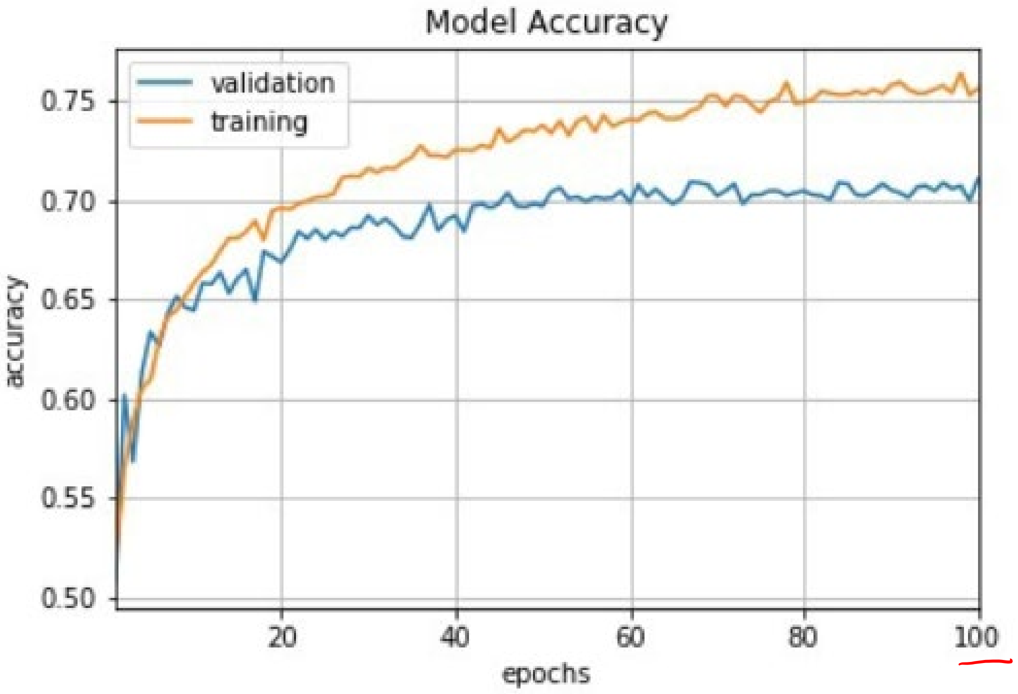
Learning rate scheduling

```
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,
0.001
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))
```

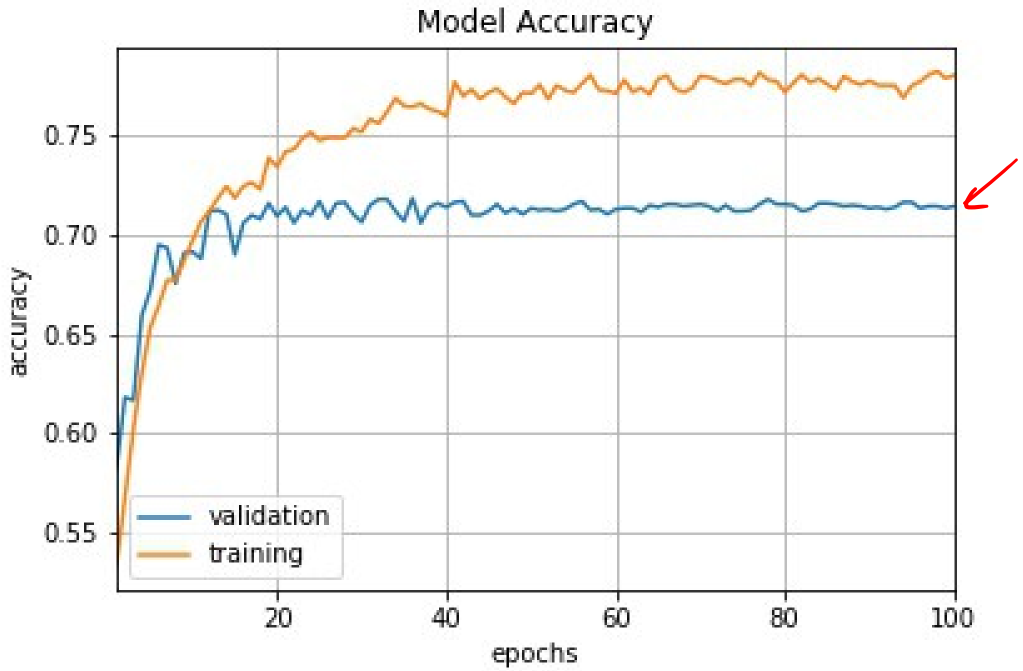
Learning rate scheduling (custom)

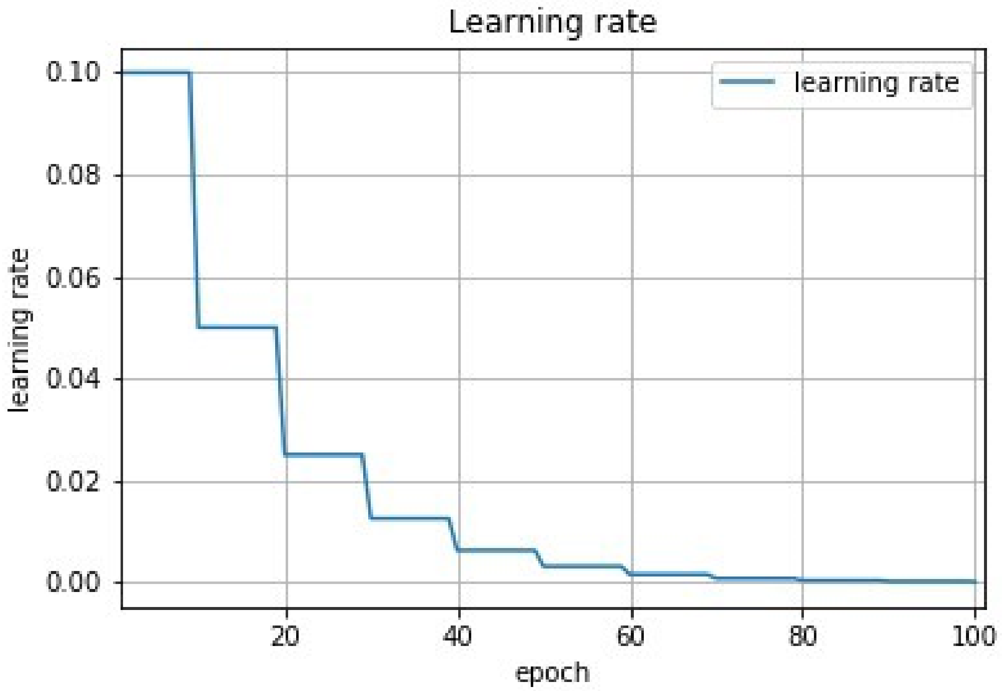
```
tf.keras.callbacks.LearningRateScheduler(
    schedule, verbose=0
)
```

Ex2

drop lr by half every 10 epochs

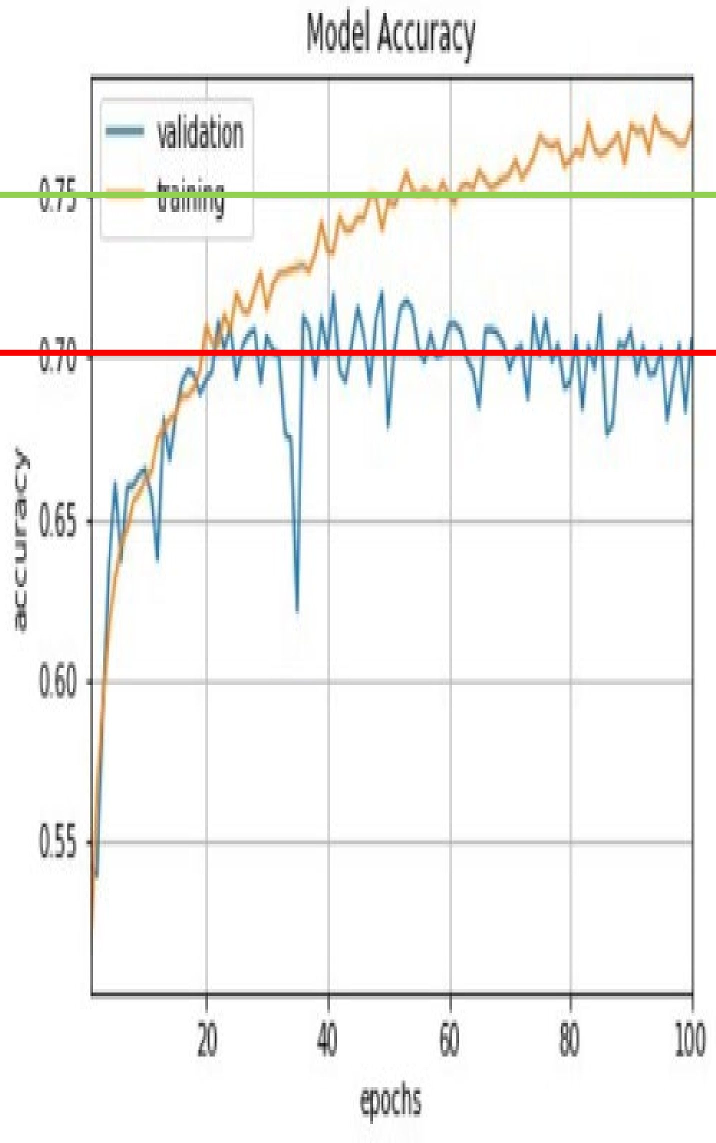
```
def step_decay(epoch):
   initial_lrate = 0.1
   drop = 0.5
   epochs_drop = 10.0
   lrate = initial_lrate * math.pow(drop,
           math.floor((1+epoch)/epochs_drop))
   return lrate

lrate = LearningRateScheduler(step_decay)
```



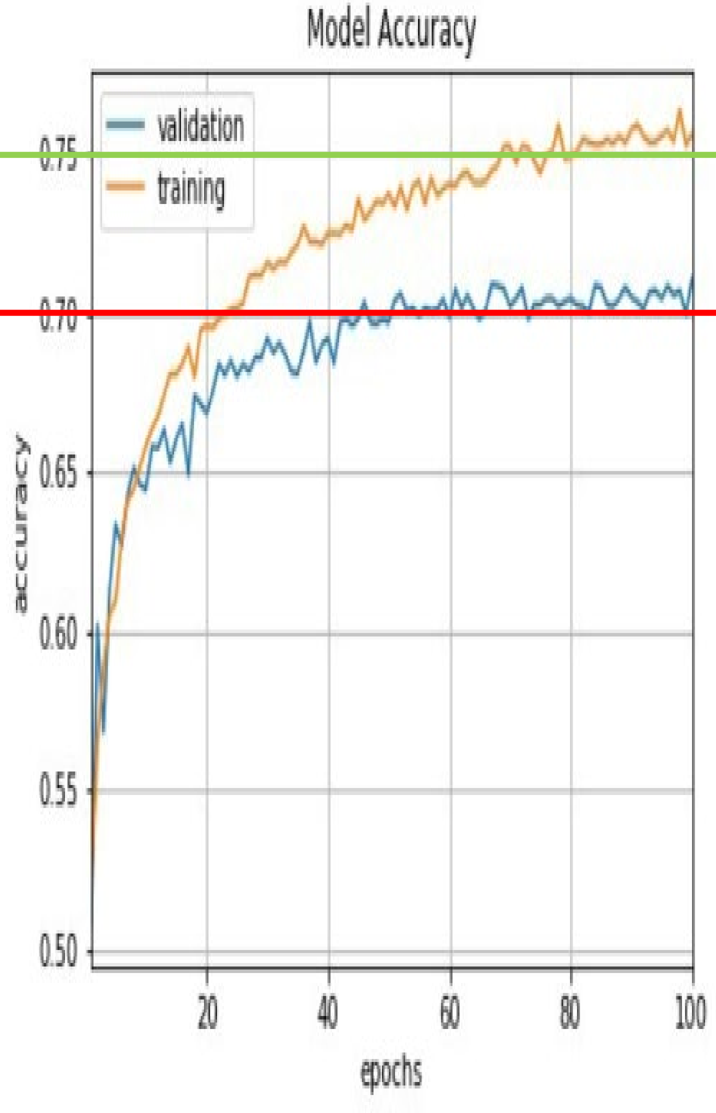


comparison

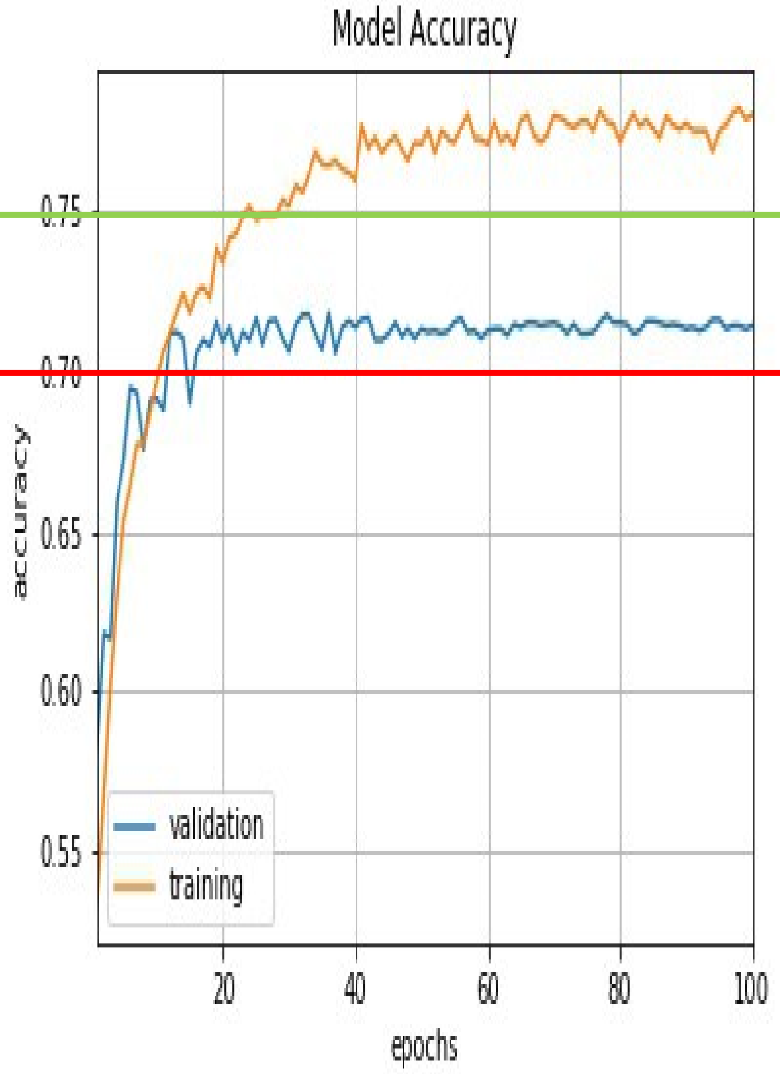
Base (fixed lr)



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

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

$$\theta \leftarrow \theta + v.$$

Nestrov momentum

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

$$\theta \leftarrow \theta + v,$$

warning- different notations used (from deeplearningbook.org)

Advanced optimization

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

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

$$\sqrt{\sum g^2/m}$$

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

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

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

Adaptive Moment Estimation (Adam)

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

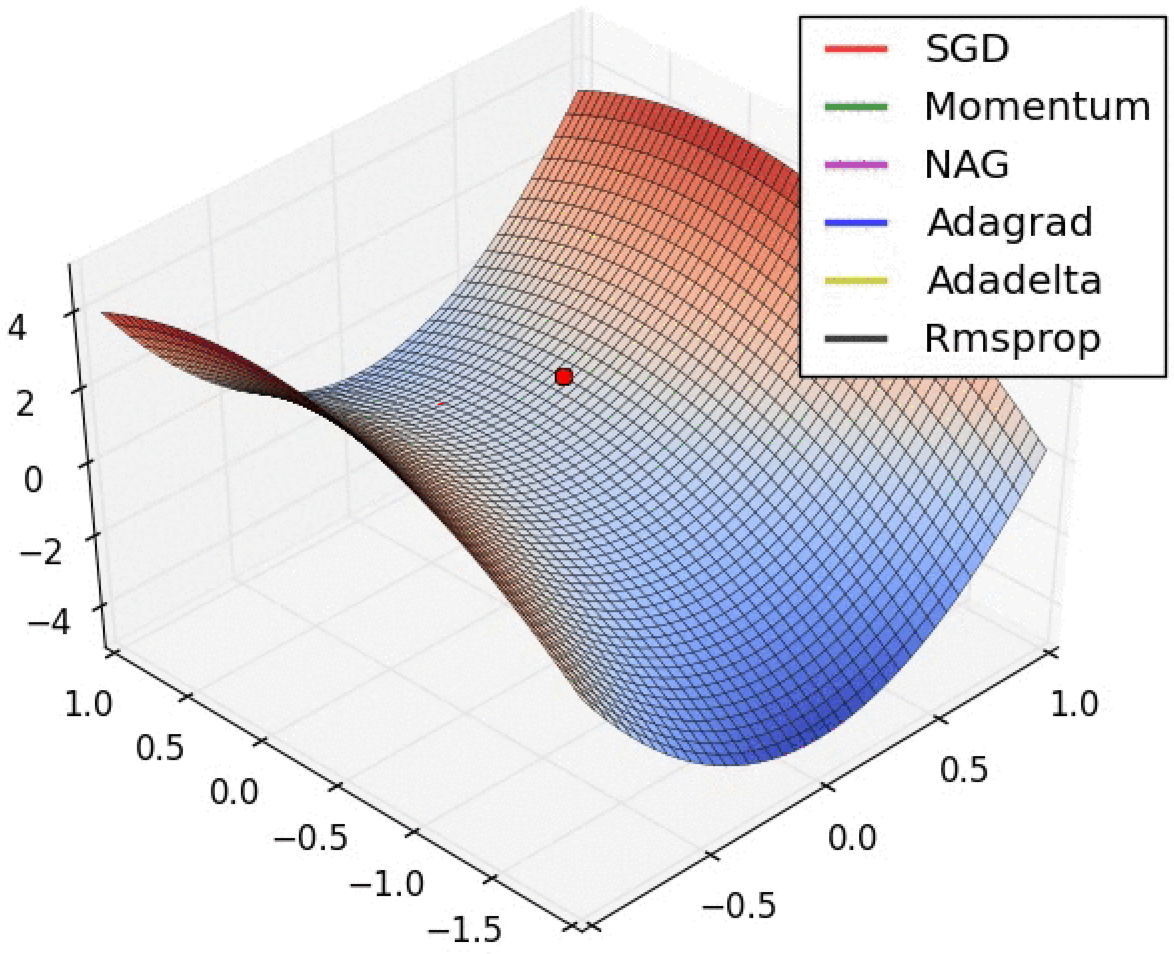
$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \end{aligned}$$

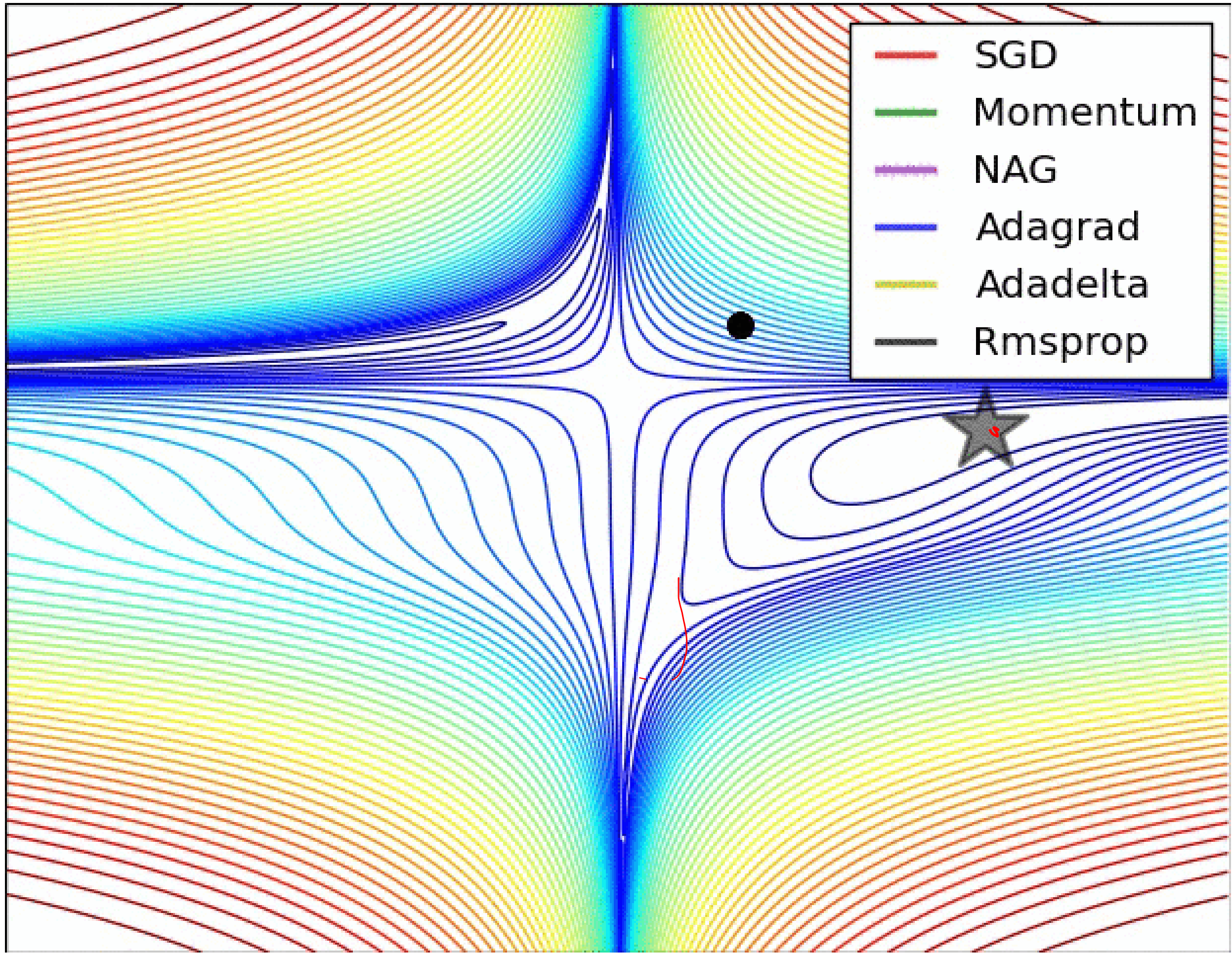
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

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

warning- different notations used

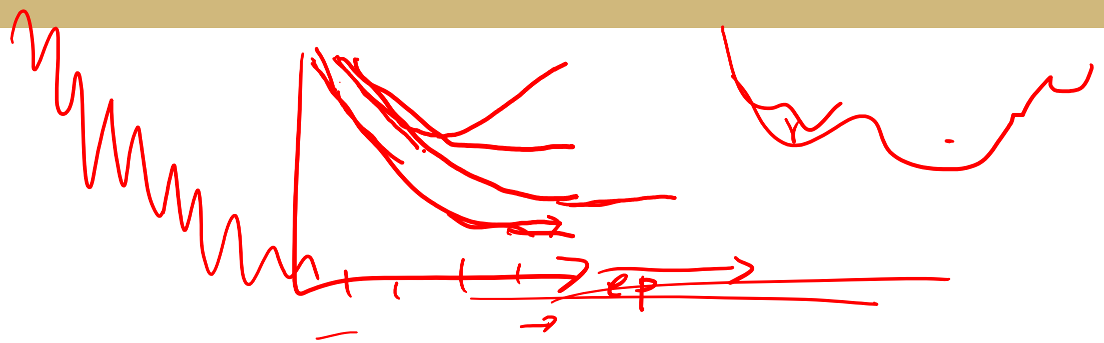
Advanced optimization





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

Monitoring Overfitting in Training

Dataset split
Train / Validation / Test

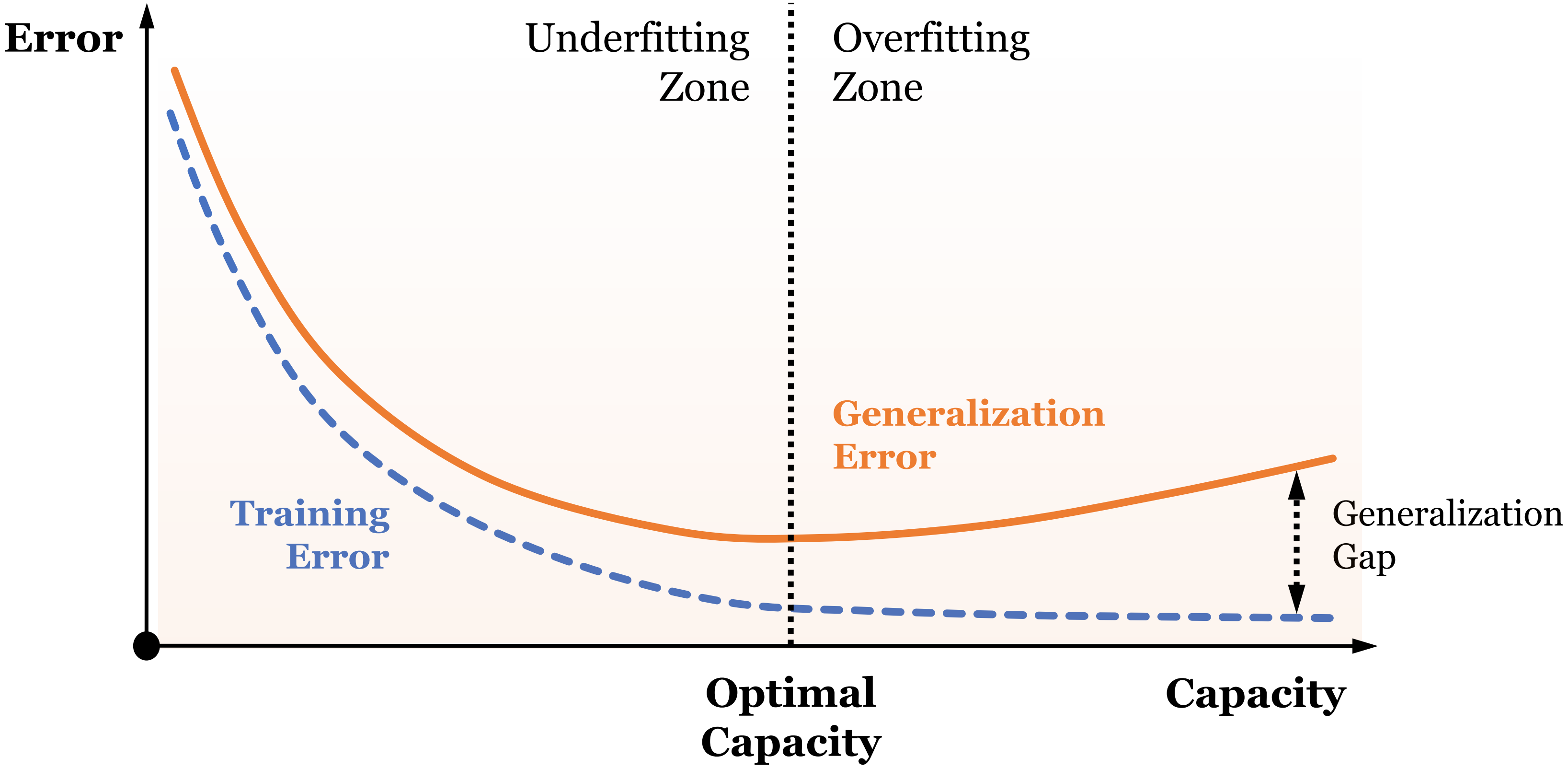
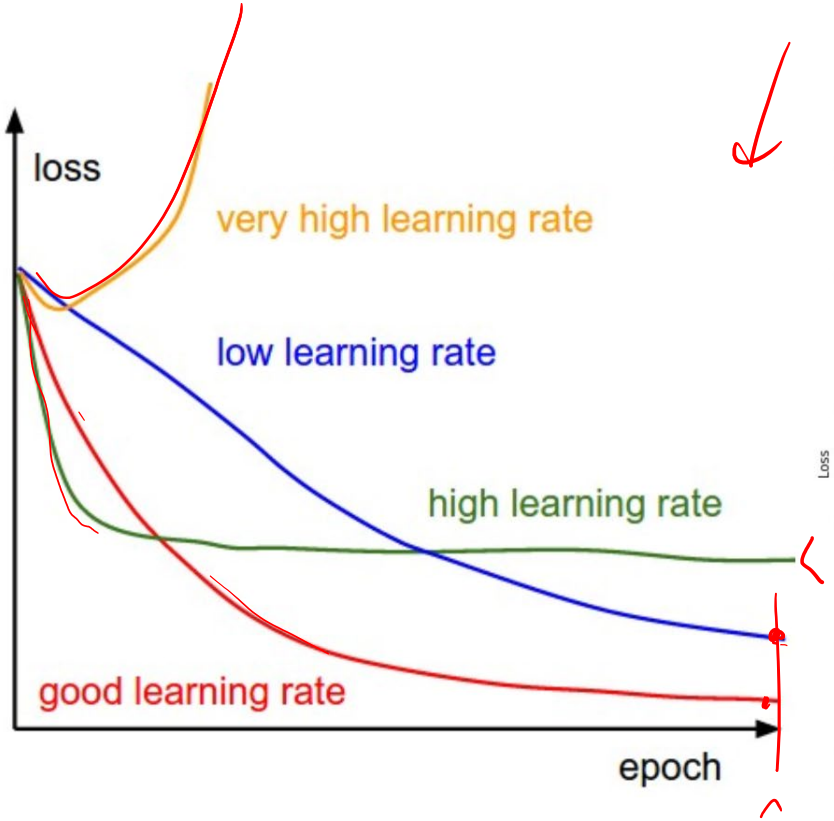
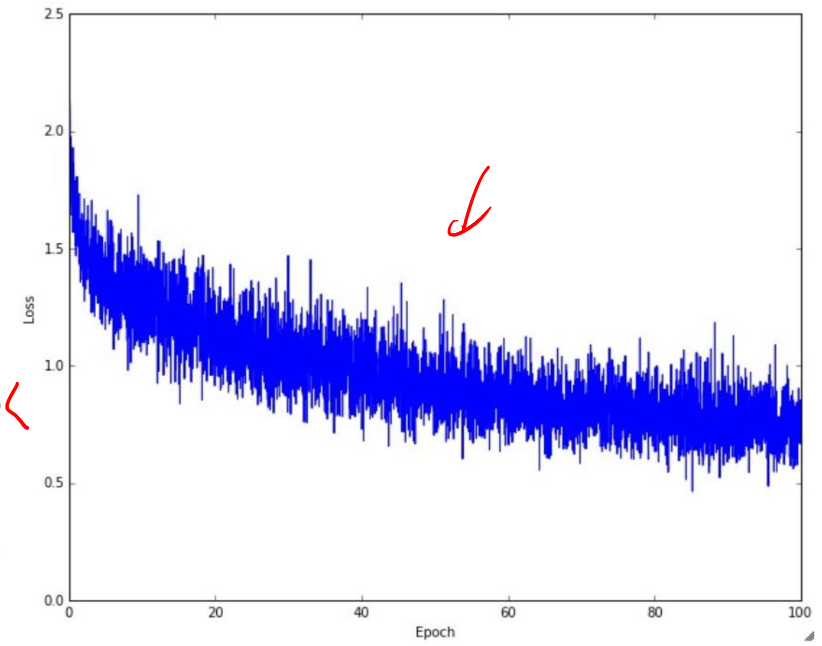
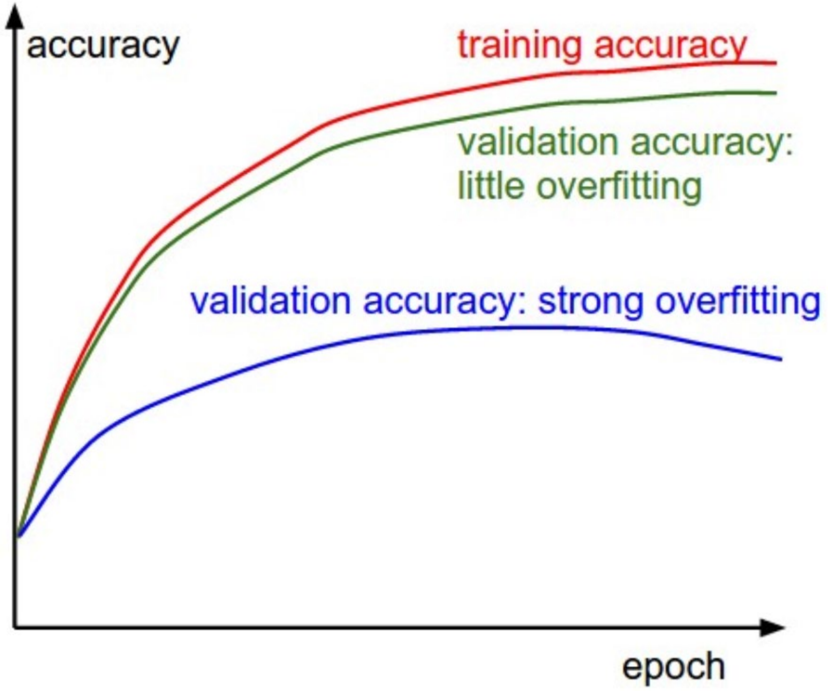


Diagram credit: Fei-Fei Li

Monitoring Overfitting in Training



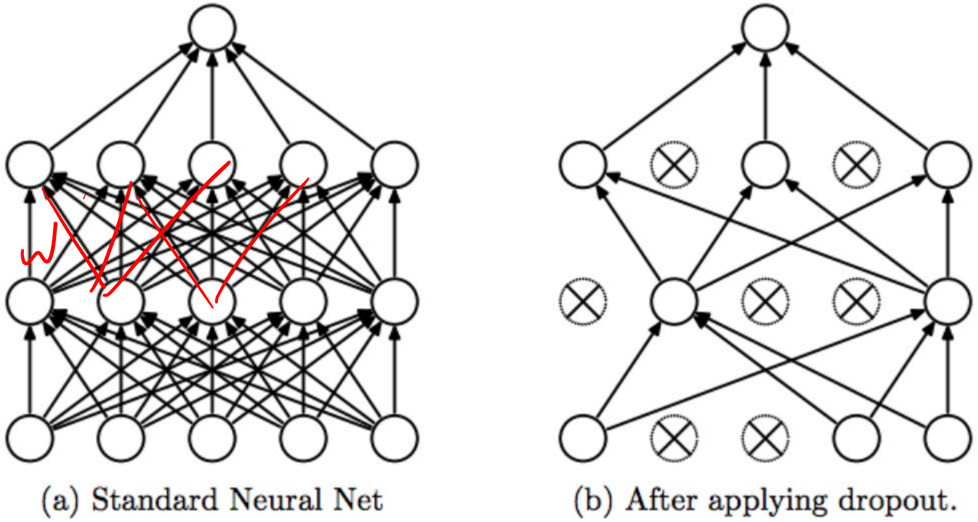




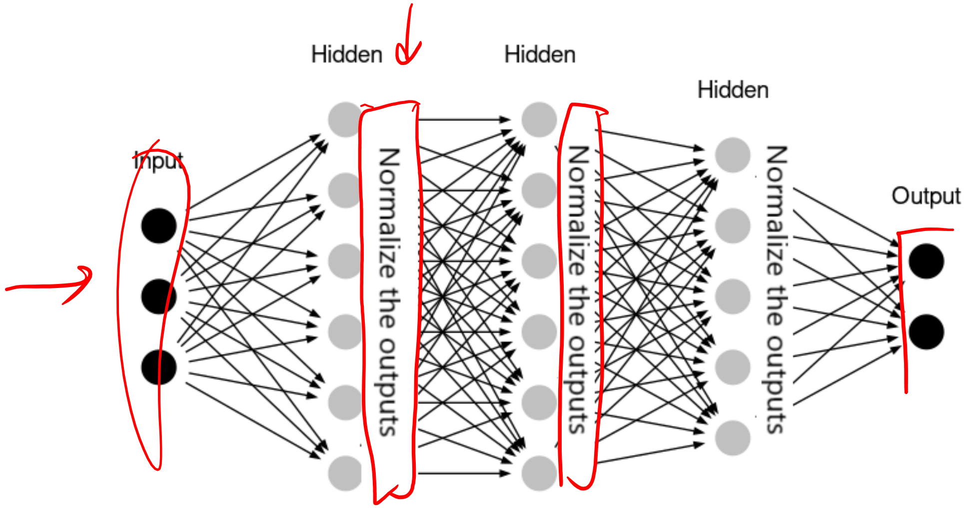
http://cs231n.github.io/neural-networks-3/

Ways to reduce overfitting

Dropout



Batch normalization



https://www.kaggle.com/c/cub-csci-4622-kaggle-2-2020/overview

To participate please check the Piazza post for the invitation link.