

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

### Tips for neural network training

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

### Advanced Gradient Descent methods

- Learning rate scheduling
- Nesterov momentum

- AdaGrad
- AdaDelta
- RMSprop
- Adam

# Gradient Descent

## Optimization Goal

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

## Weight update rule

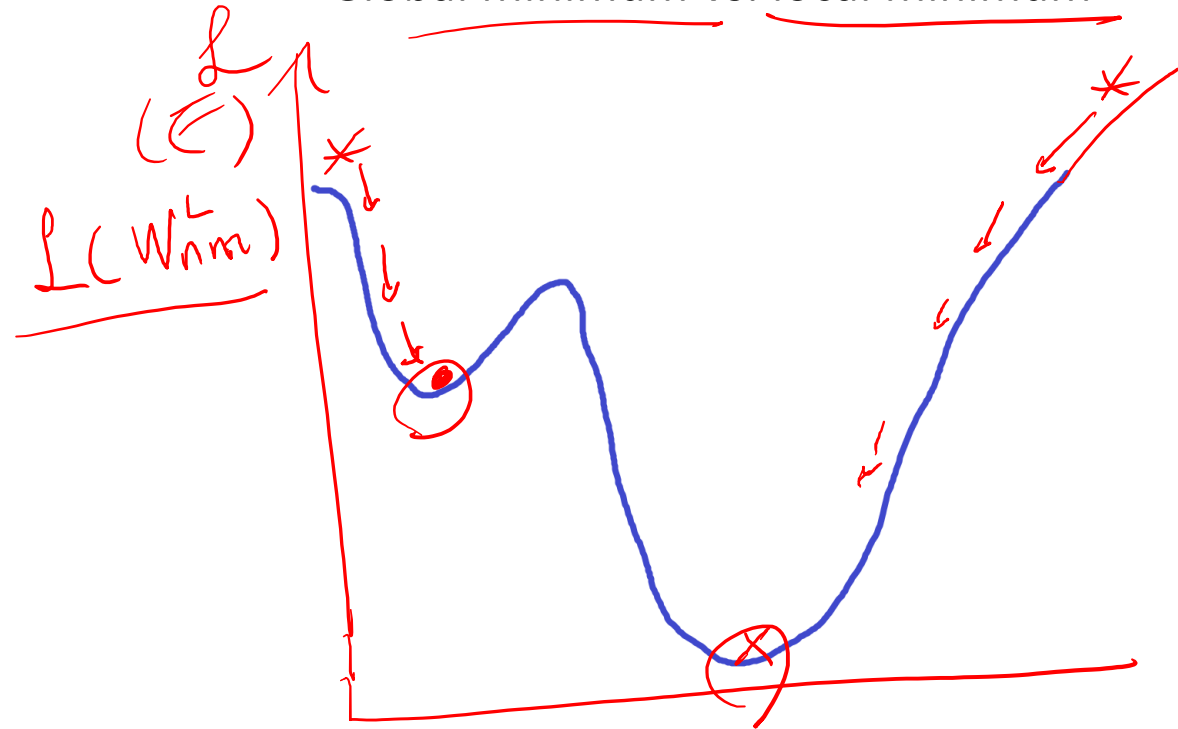
$$\dot{W}_{nm}^L \leftarrow W_{nm}^L - \alpha * \delta W_{nm}^L \frac{\partial \mathcal{L}}{\partial W_{nm}^L}$$

*Handwritten notes:*  $\delta W_{nm}^L$  is circled in red.  $\frac{\partial \mathcal{L}}{\partial W_{nm}^L}$  is written in red above the equation.  $n+1$  is written in red below the equation.

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

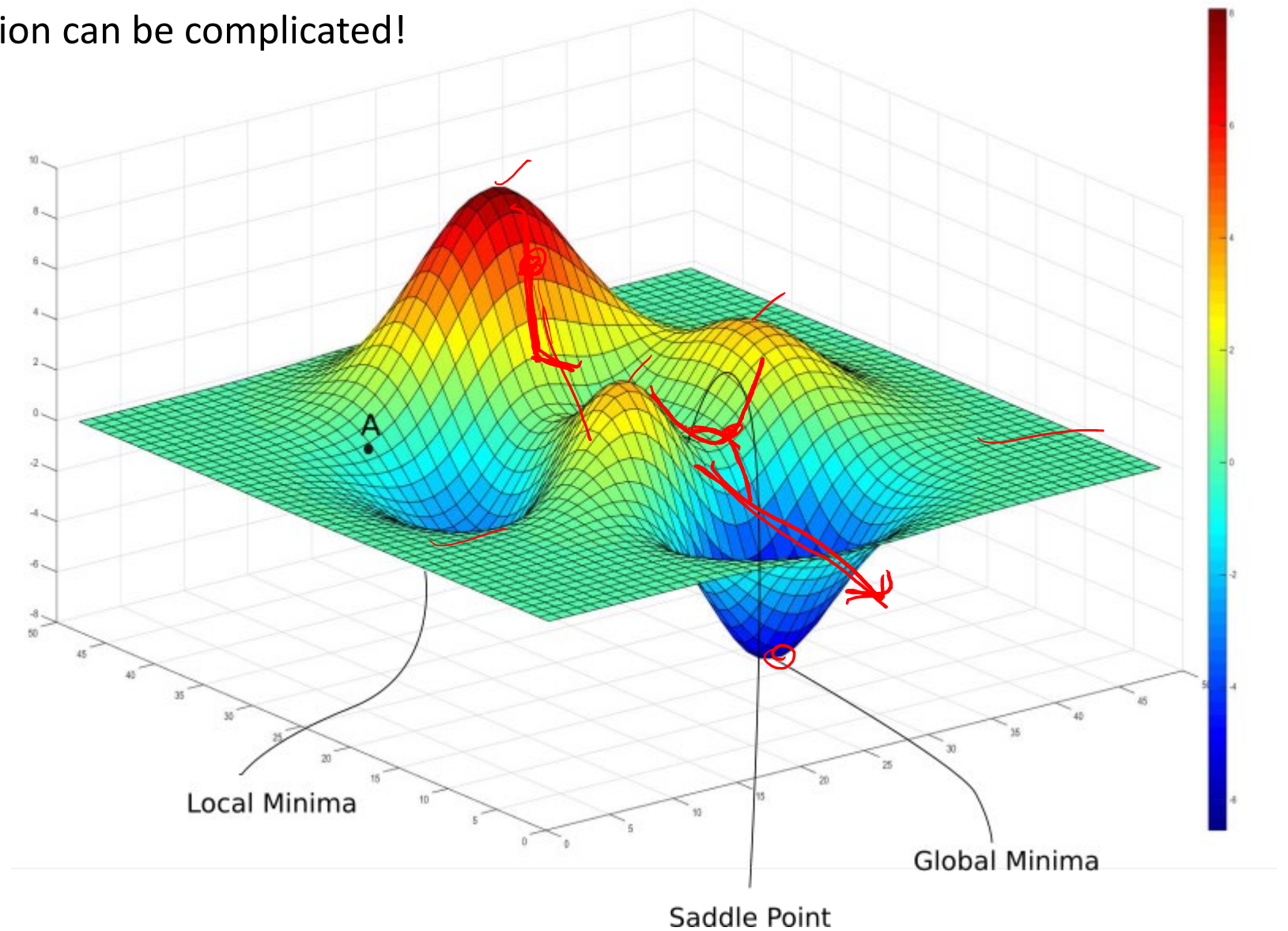
*Handwritten notes:* A red line is drawn under the equation.

## Global minimum vs. local minimum



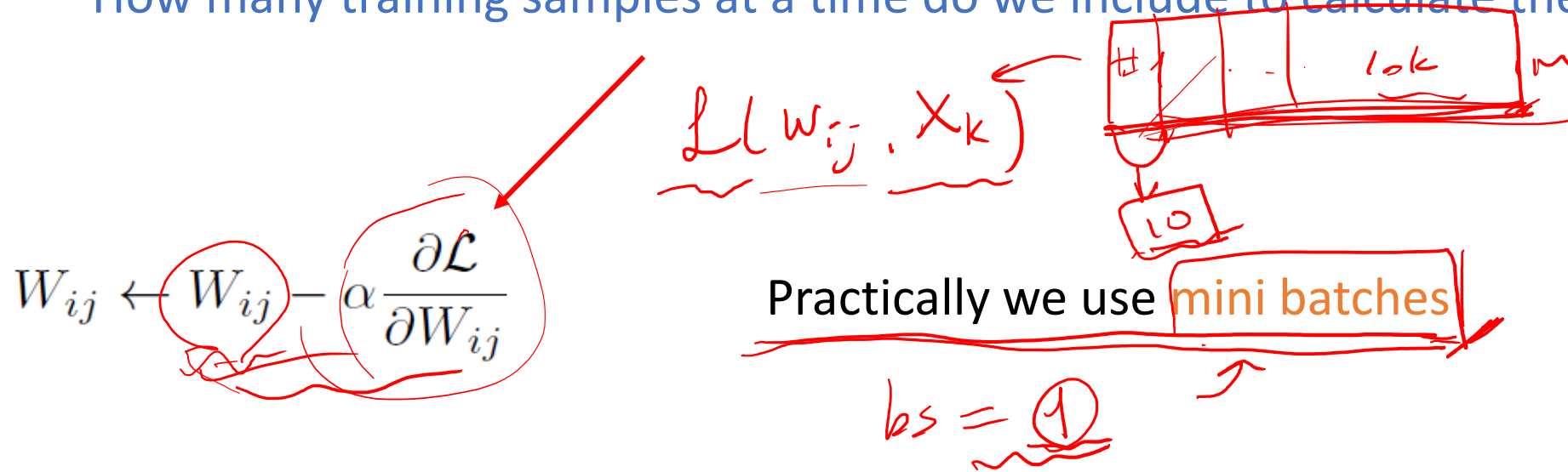
# Gradient Descent

Error surface in the multi dimension can be complicated!



# Stochastic Gradient Descent

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



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  $\epsilon_1, \epsilon_2, \dots$

**Require:** Initial parameter  $\theta \leftrightarrow w$

$k \leftarrow 1$

**while** stopping criterion not met **do**

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

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

Apply update:  $\theta \leftarrow \theta - \epsilon_k \hat{\mathbf{g}}$

$k \leftarrow k + 1$

**end while**

---

# Stochastic Gradient Descent with momentum

SGD with learning rate alone is slow to converge

Adding a momentum (moving average) can make it faster

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

warning- different notations used (from deeplearningbook.org)

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

# 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
- nesterov
- (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

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


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

# Stochastic Gradient Descent with decay

Learning rate scheduling using decay

For iteration  $k$  (epoch)





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

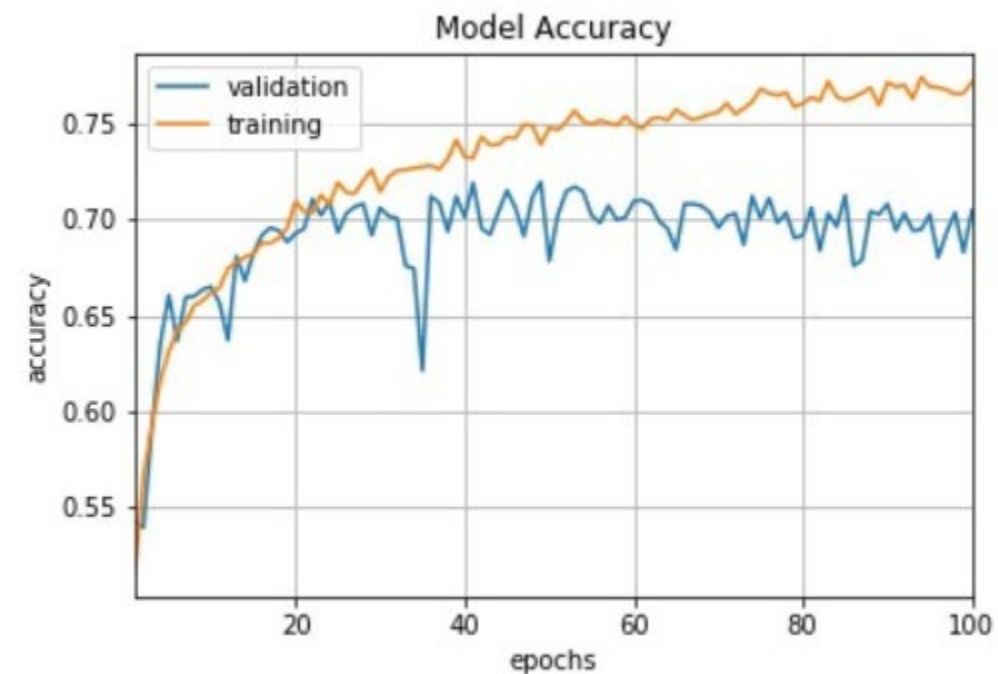
\*\* In the algorithm pseudocode  $k$  is for step (each mini batch),  
and decay learning rate by step,  
but normally we decrease learning rate each epoch


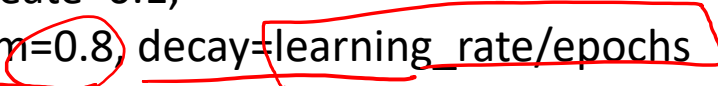
warning- different notations used (from deeplearningbook.org)

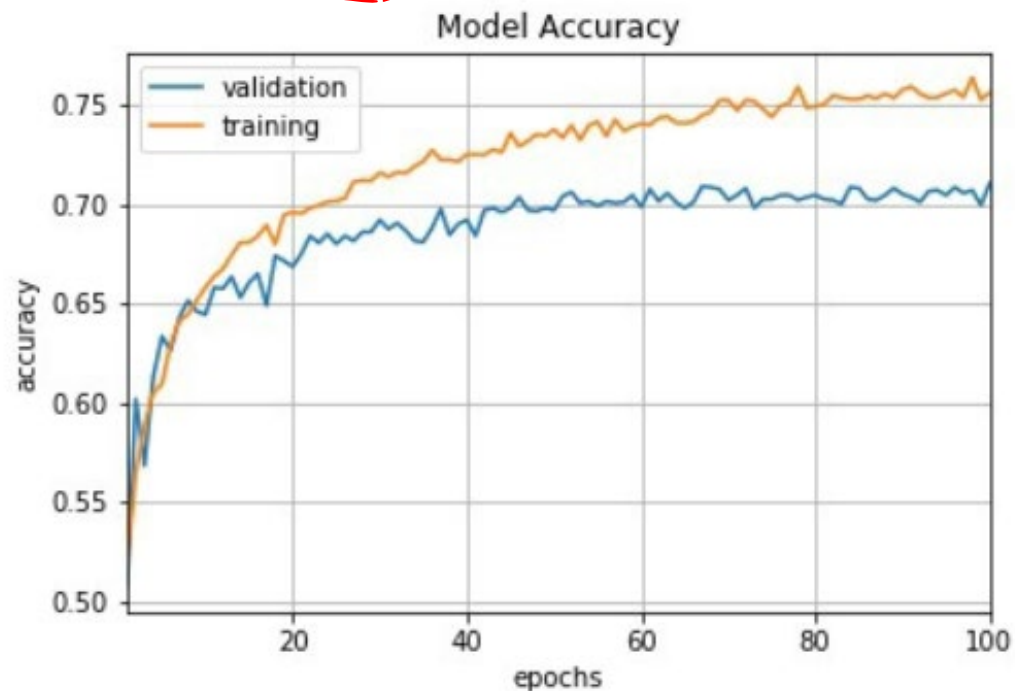
# Learning rate scheduling

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

learning\_rate=0.1,   
momentum=0, decay=0, nesterov=False 



learning\_reate=0.1,   
momentum=0.8, decay=learning\_rate/epochs 



# Learning rate scheduling (custom)

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

```
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
```

Ex2

drop lr by half every 10 epochs

```
lrate = LearningRateScheduler(step_decay)
```

```
# 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))
```

Ex1

# Learning rate scheduling (custom)

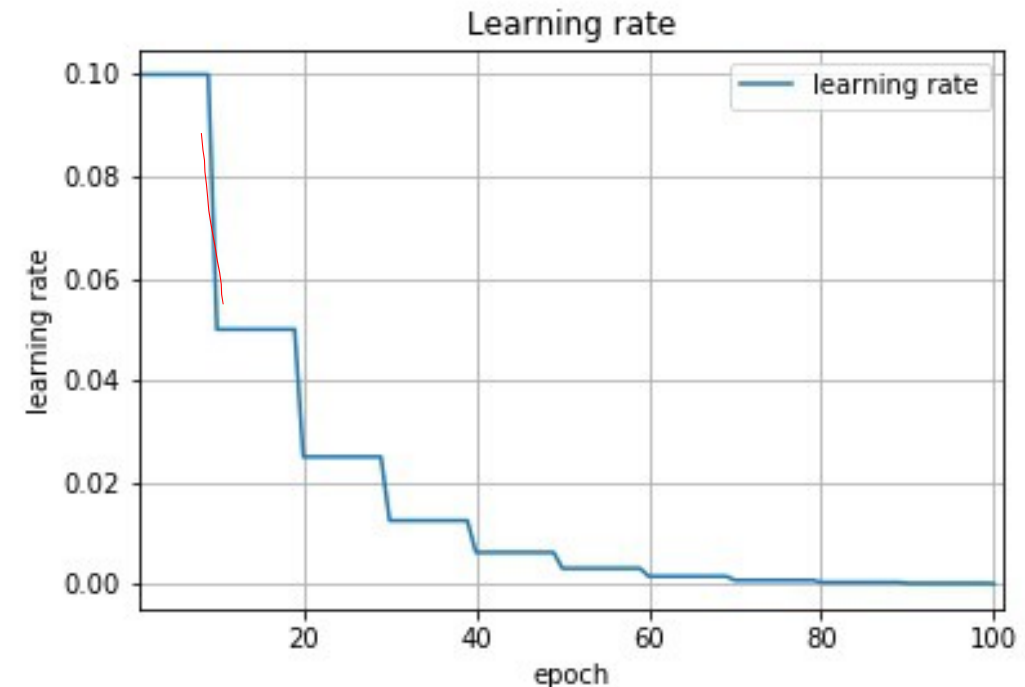
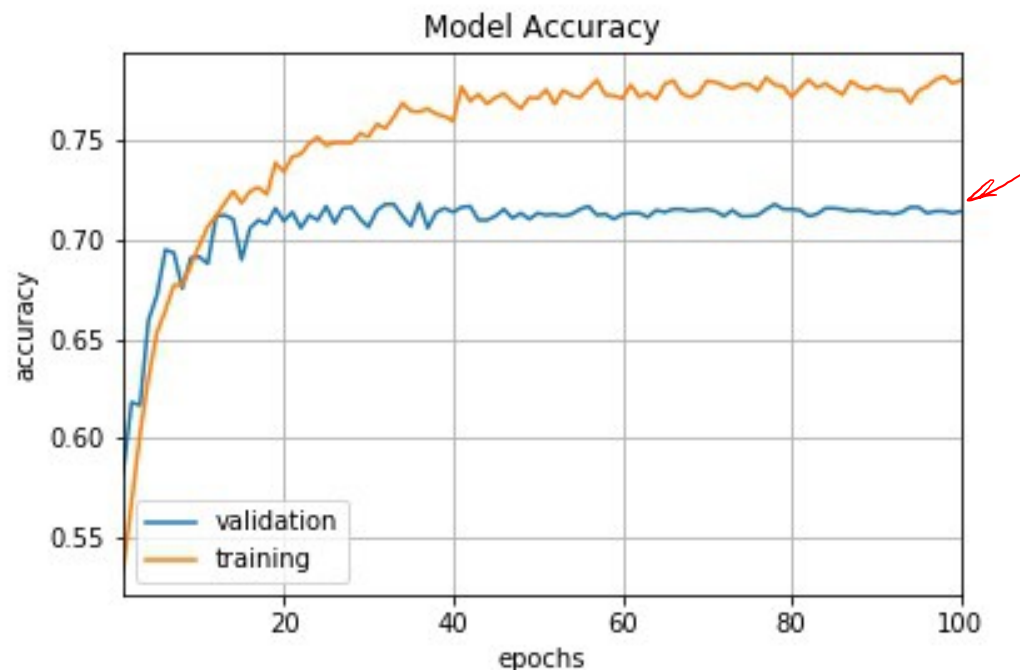
```
tf.keras.callbacks.LearningRateScheduler(  
    schedule, verbose=0  
)
```

```
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
```

Ex2

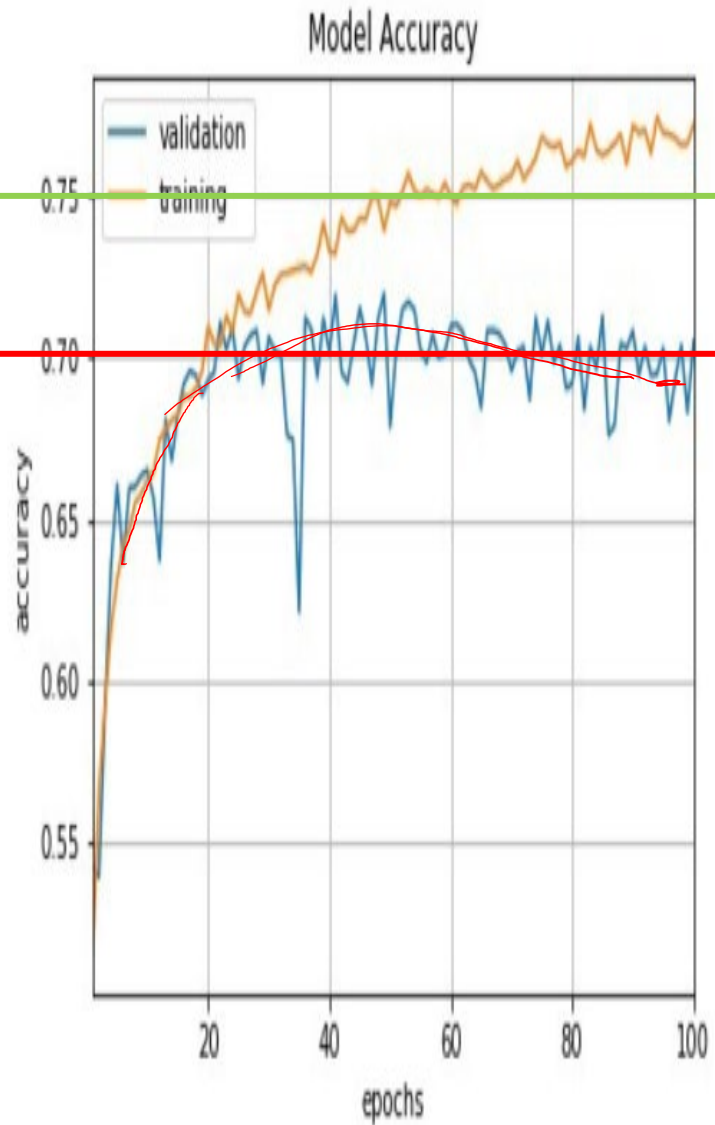
drop lr by half every 10 epochs

```
lrate = LearningRateScheduler(step_decay)
```

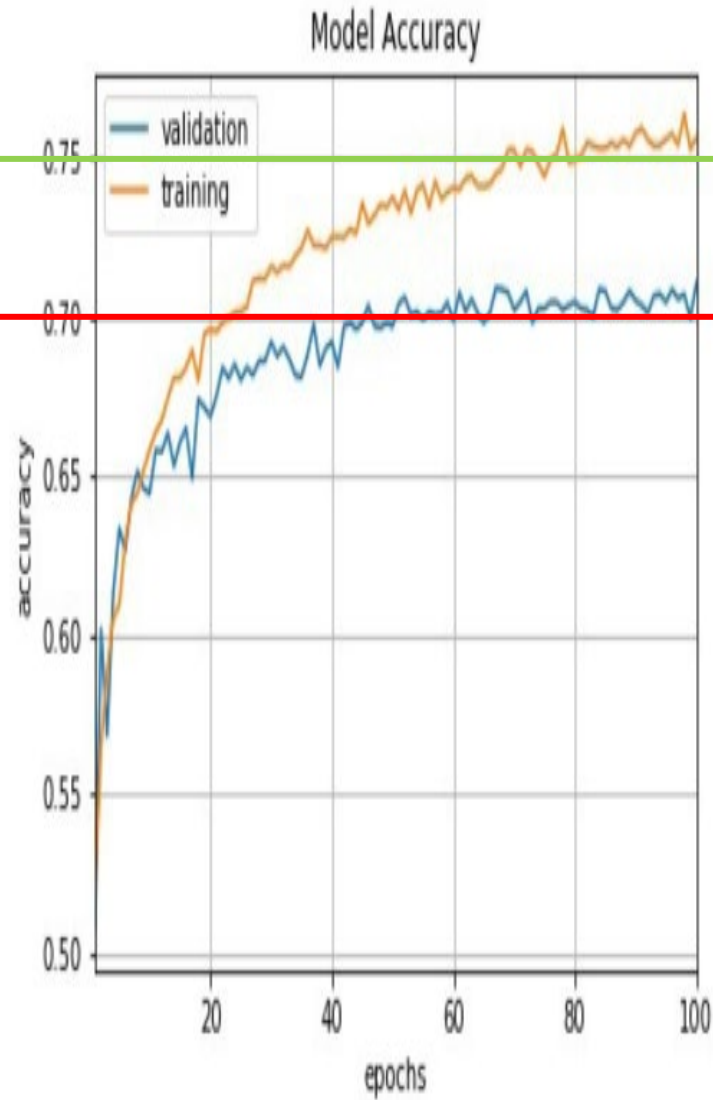


# comparison

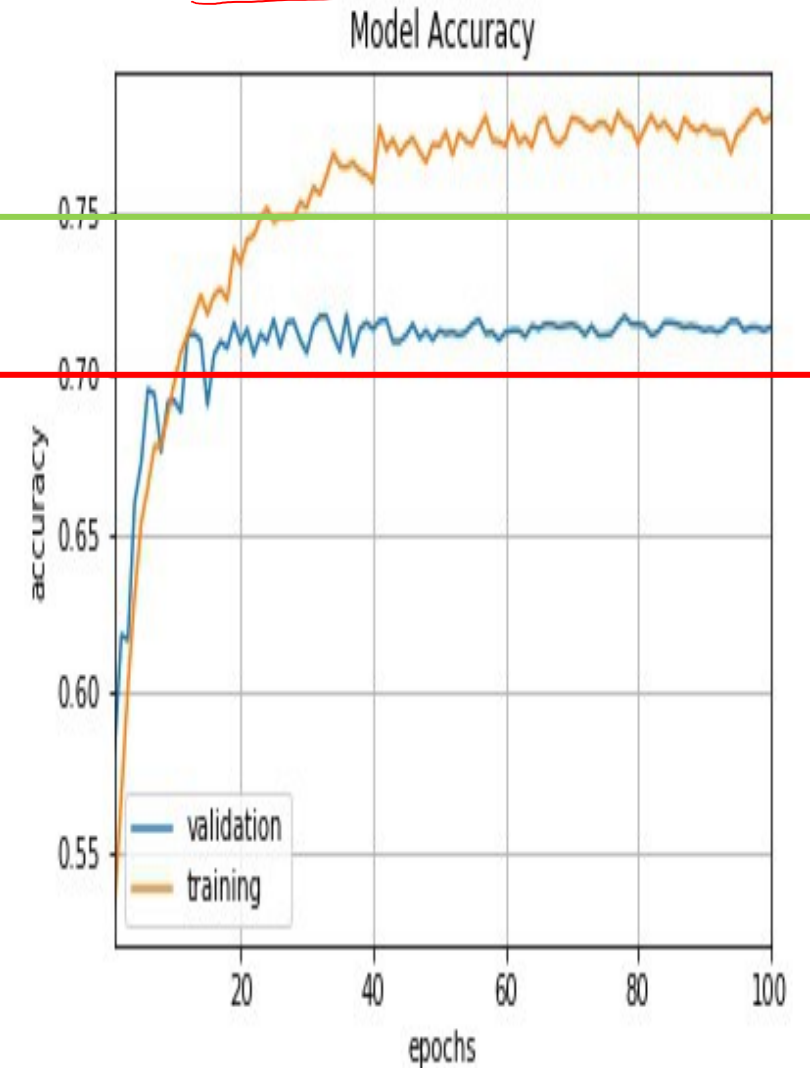
Base (fixed lr)



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

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

Nestrov momentum

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

## Adagrad

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

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



# Advanced optimization

## Adadelta

learning rate is normalized by the RMS of the gradient

Weight change is proportional to the RMS ratio

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

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

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

$$E[g^2]_t = \underbrace{0.9}_{\eta} E[g^2]_{t-1} + \underbrace{0.1}_{\eta} g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta g_t}{\sqrt{E[g^2]_t + \epsilon}}$$

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

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

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

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

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

*momentum*

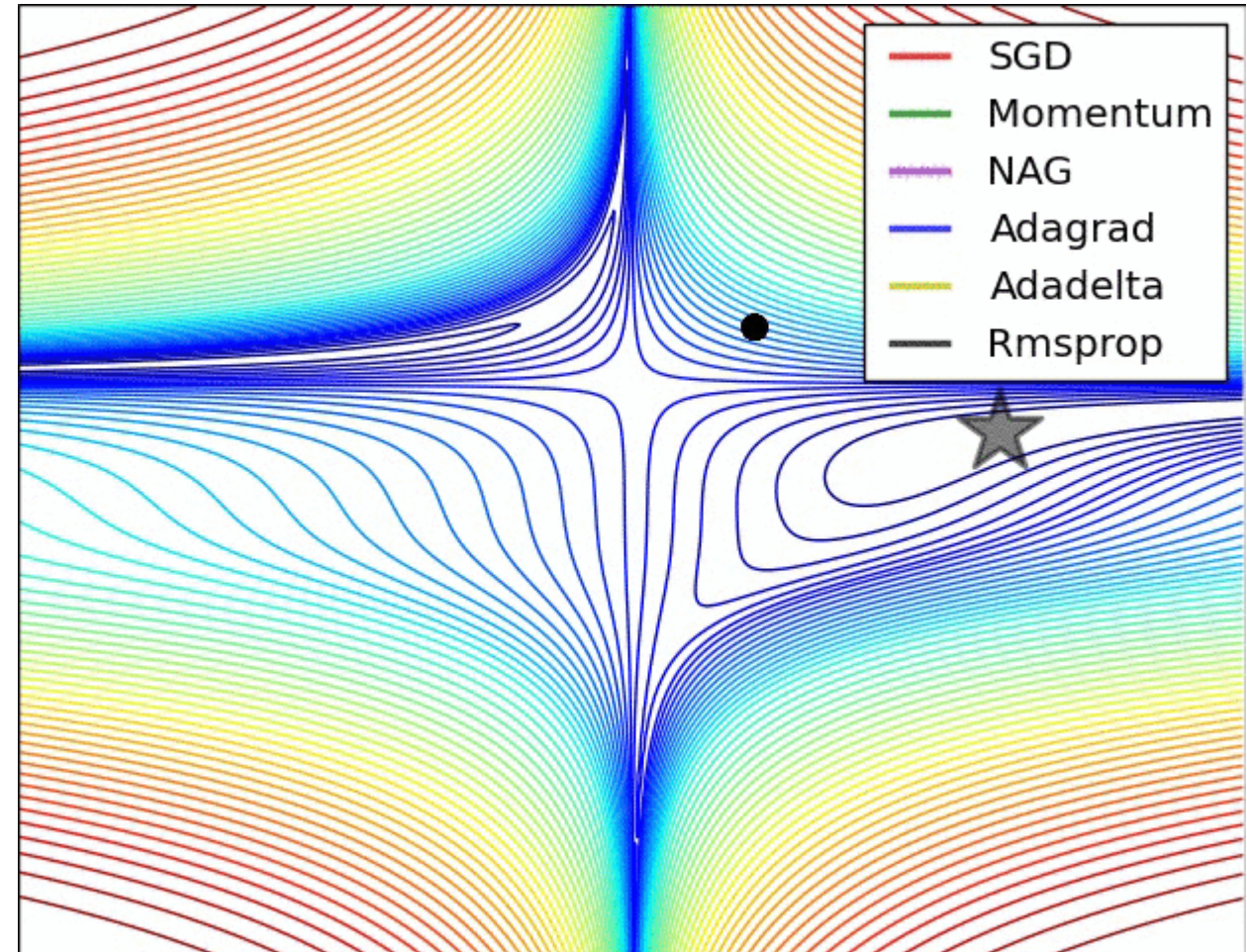
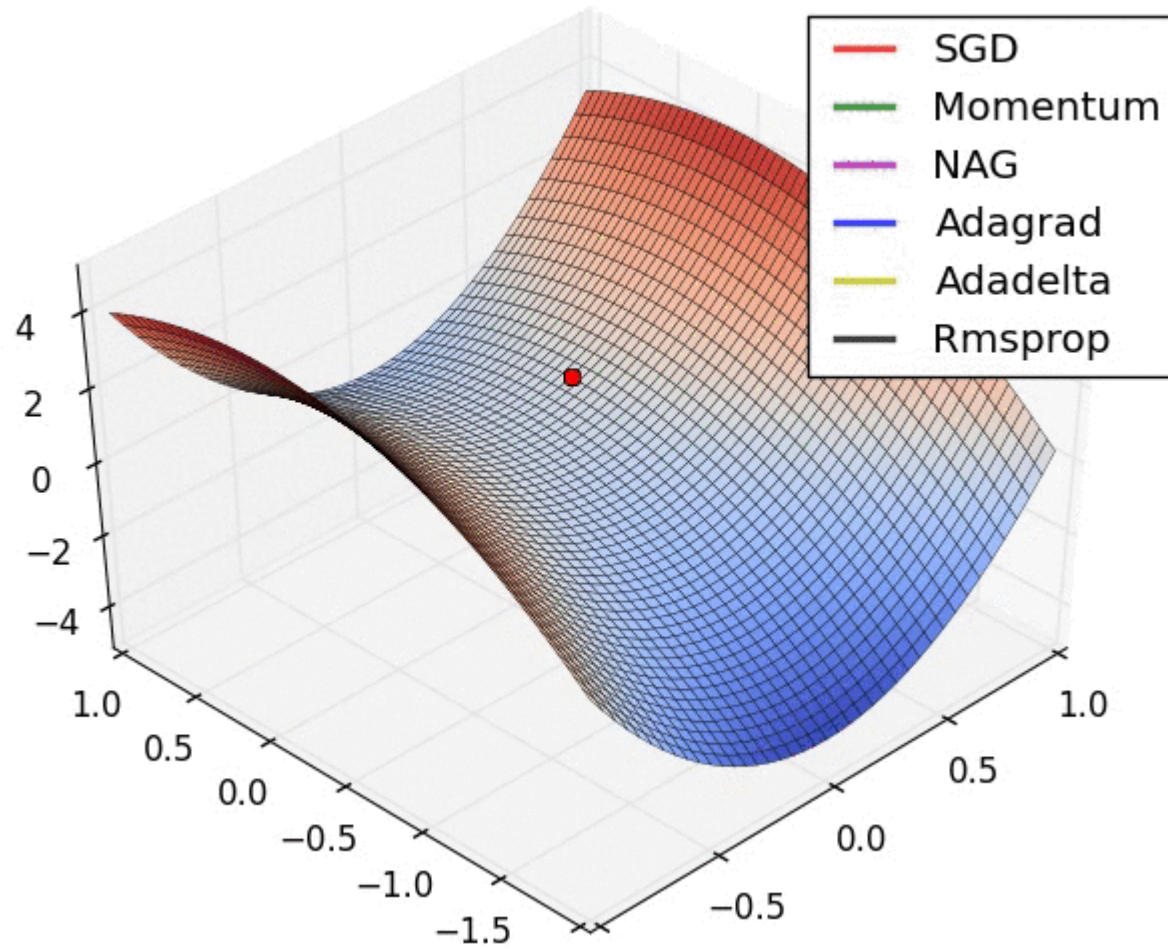
*RMS  $g$   $E(g^2)$*

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

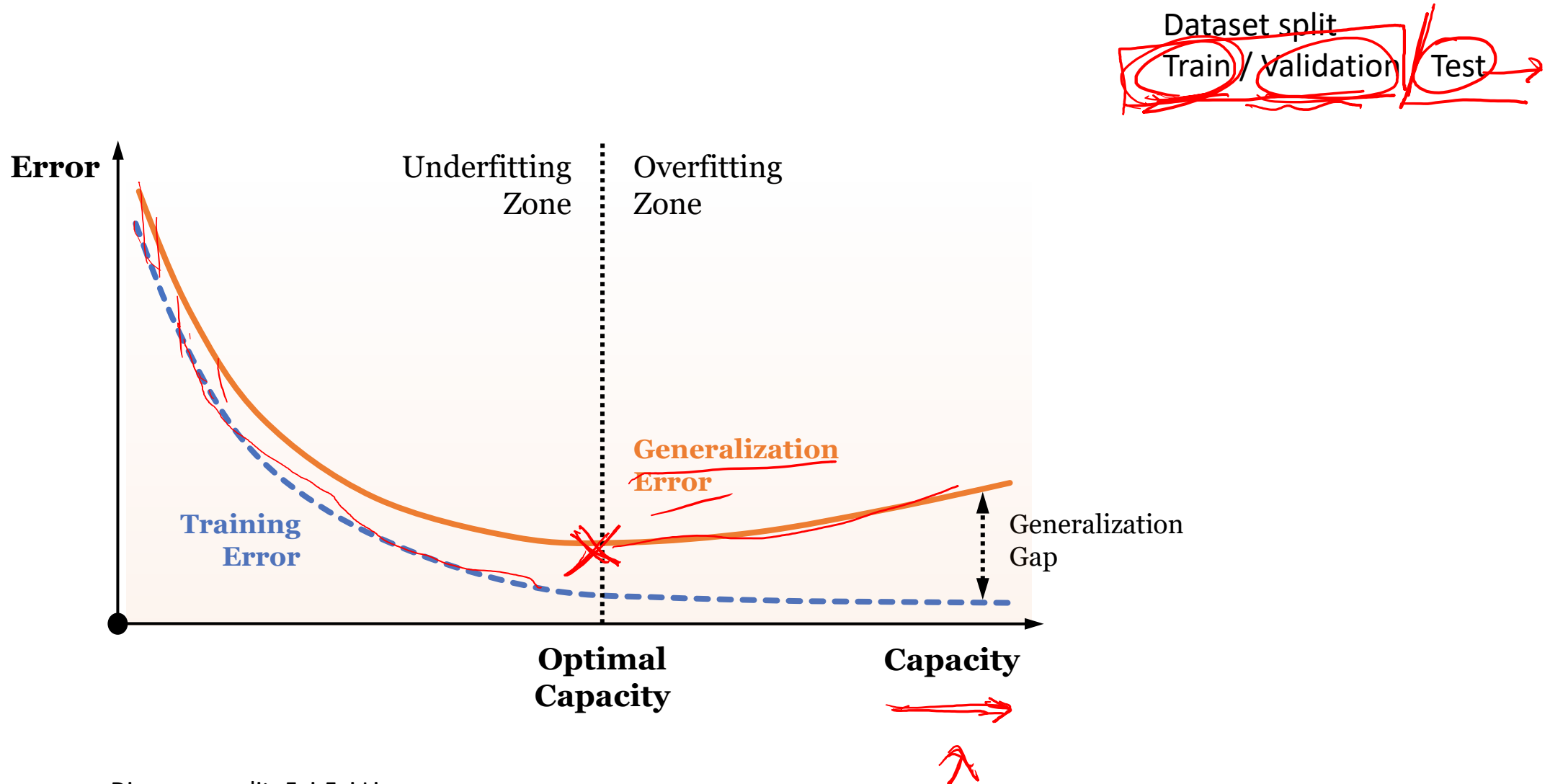
Try different optimization methods

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

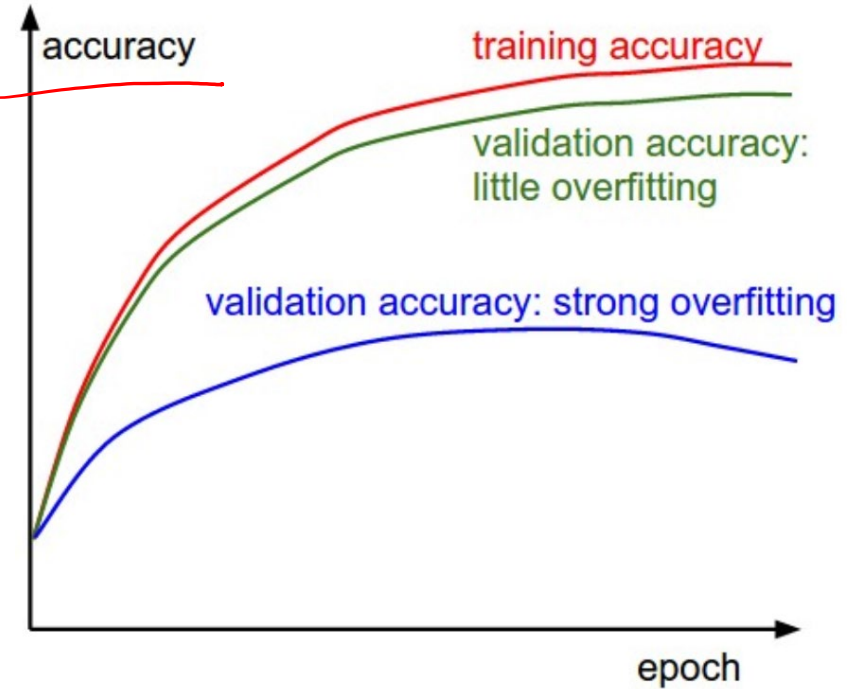
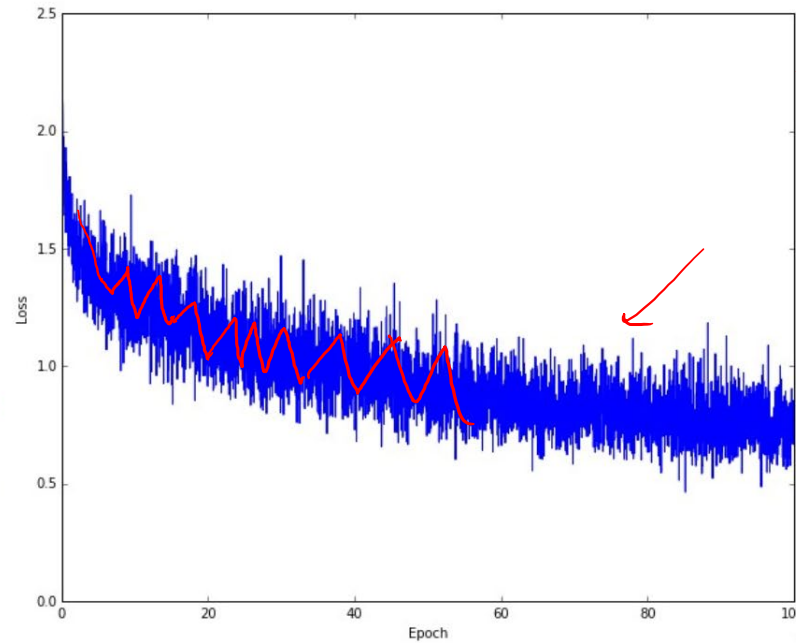
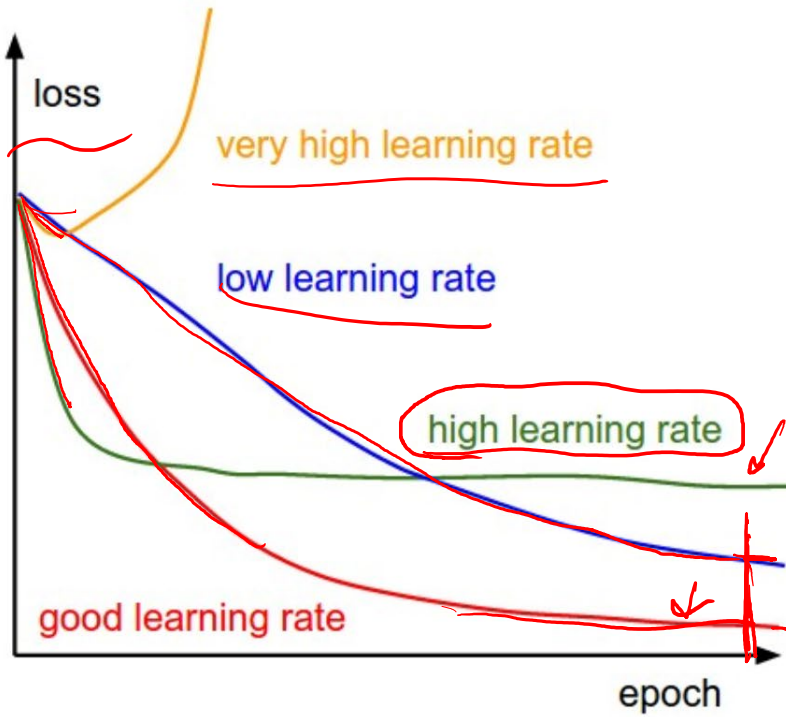
$$\sum |a| \rightarrow \sum a^2$$



# Monitoring Overfitting in Training

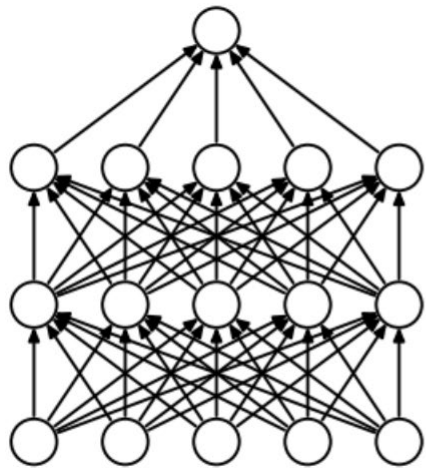


# Monitoring Overfitting in Training

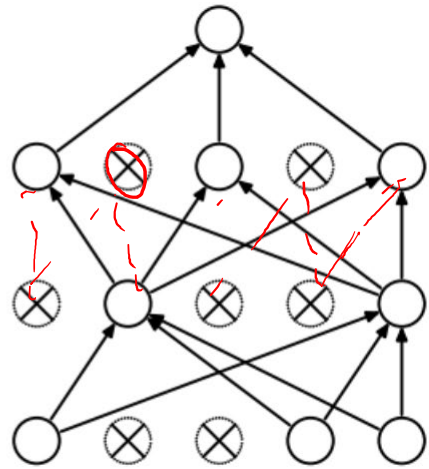


# Ways to reduce overfitting

## Dropout [1]

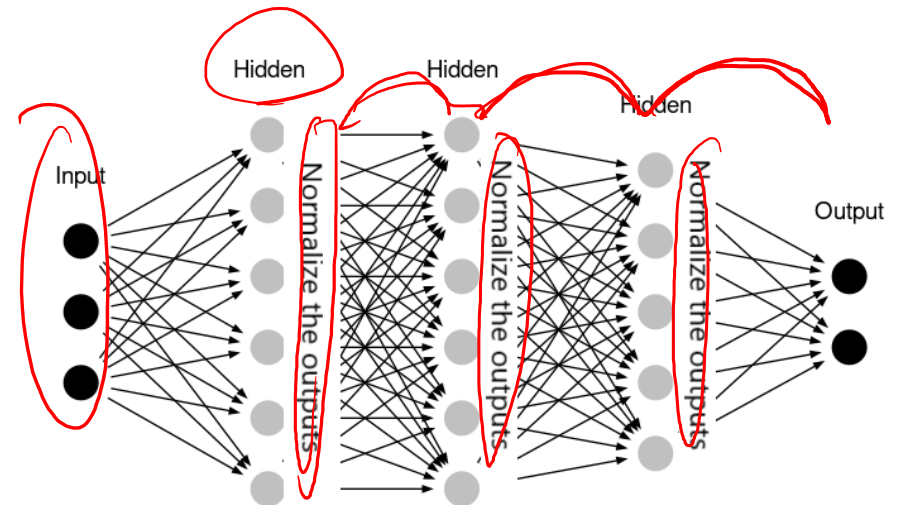


(a) Standard Neural Net



(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>