Artificial Neural Network Techniques

Dr. Mongkol Ekpanyapong

Gradient Descent

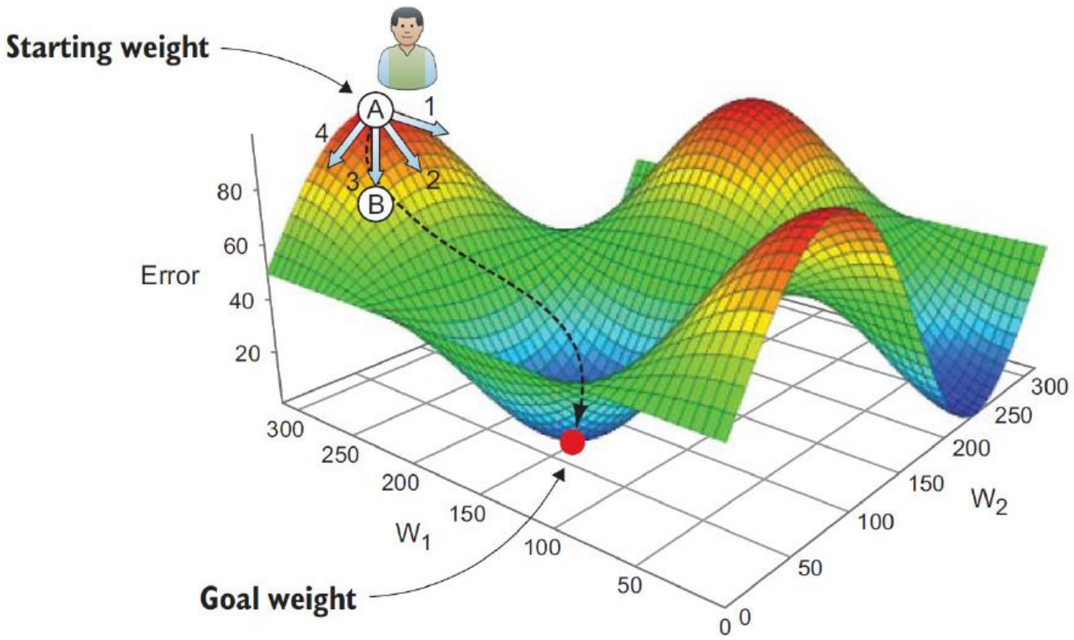
- Weight function

$$\Delta w_i = -\alpha \frac{dE}{dw_i}$$

- Weight update

$$w_{next-step} = w_{current} + \Delta w$$

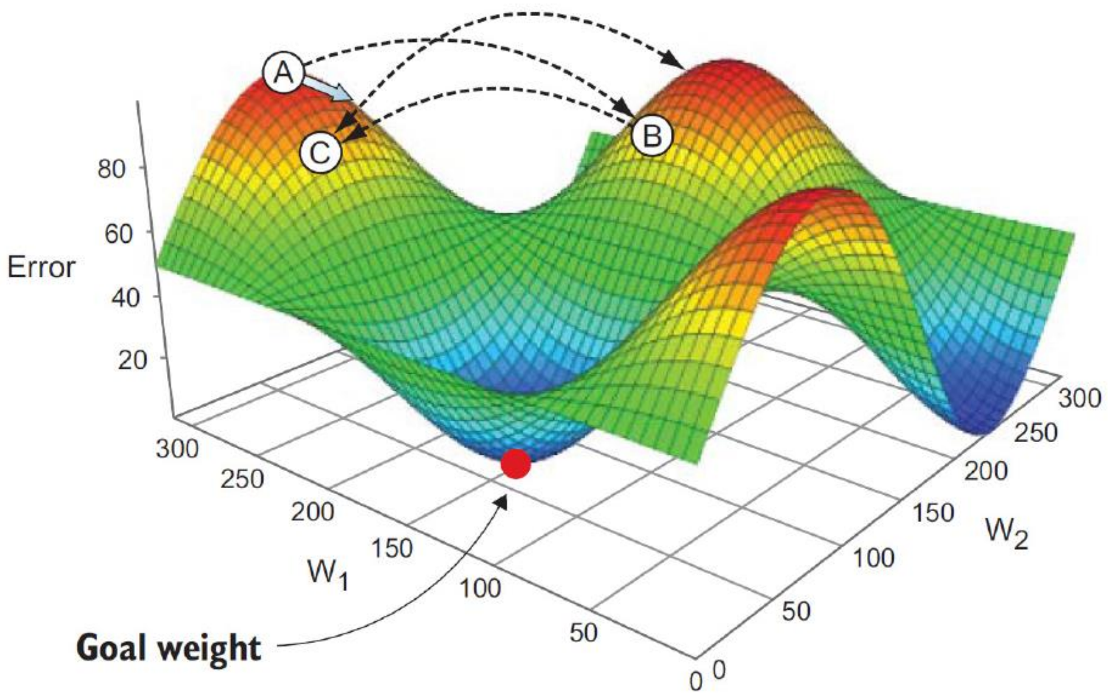
Example



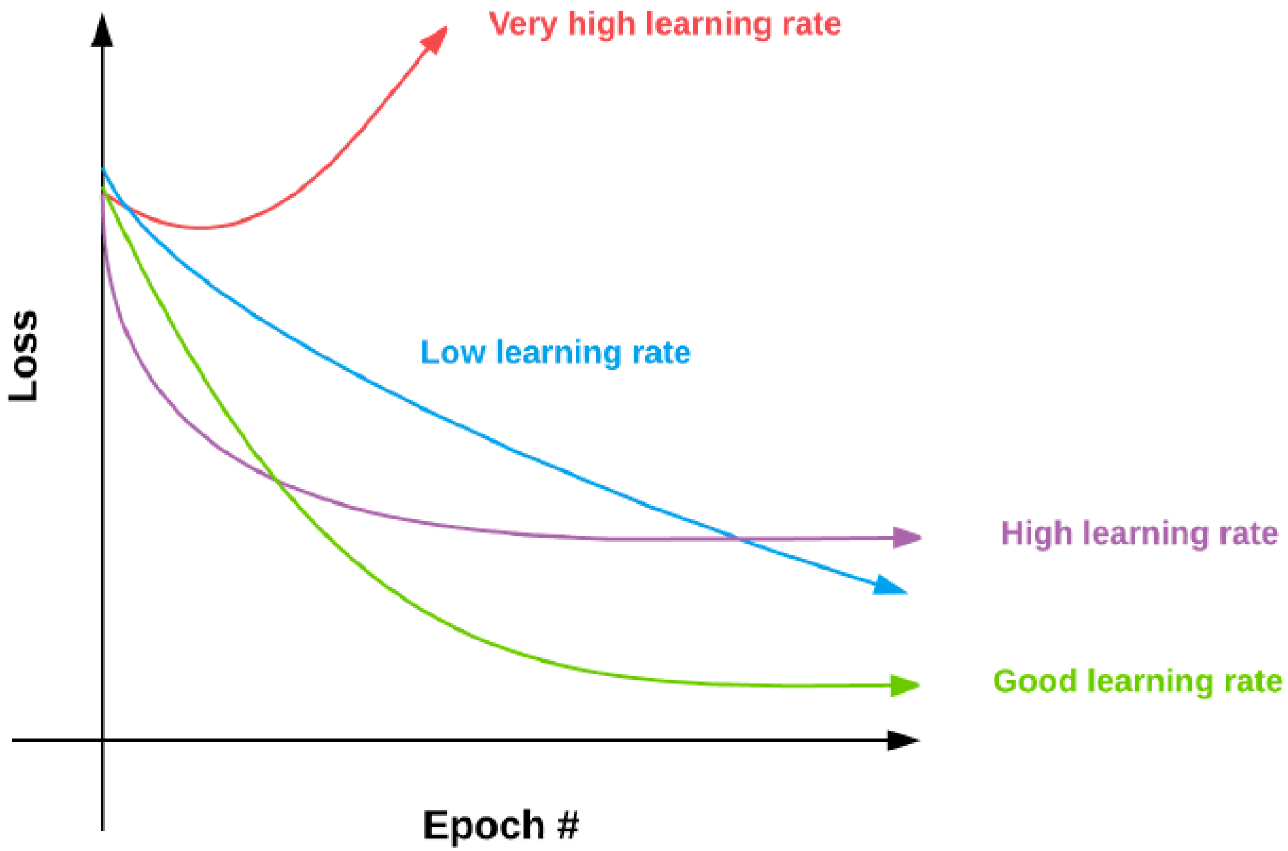
Starting weight
A
1
2
3
4
B
80
60
Error
40
20
300
250
200
150
W_1
100
50
0
0
50
100
150
200
250
300
W_2
Goal weight

The step size

- Impact of large step size



Effects of Learning Rates



Default SGD weight update

- SGD weight update

```
W += -lr * dW
```

W is weight matrix
lr is learning rate
dW is the gradient of W

Adaptive Learning Rate?

Adagrad

- Previously, we performed an update for all parameters using the same learning rate
- Adagrad adapts the learning rate to the network parameters. Larger updates are performed on parameters that change infrequently. While small updates are done on parameters changes frequently
- Cache is the variable maintains the per-parameter sum of square gradient

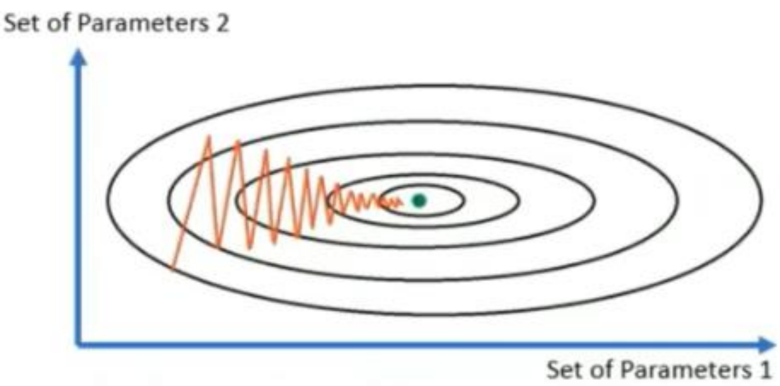
```
cache += (dW ** 2)
W += -lr * dW / (np.sqrt(cache) + eps)
```

RMSprop

- Using exponential weighted moving average

```
cache = decay_rate * cache + (1 - decay_rate) * (dW ** 2)
W += -lr * dW / (np.sqrt(cache) + eps)
```

RMSprop



We want to **minimize** the oscillation in the vertical direction
We want to **increase** the speed in the horizontal direction

Denote Set of Parameters 1 as ***dw***
Denote Set of Parameters 2 as ***db***

dw is small (has a small variation)
db is large (has a large variation)

dw^2 and db^2 are element-wise squared

$$s_{dw} = \beta s_{dw} + (1-\beta)dw^2 \quad \text{small}$$
$$s_{db} = \beta s_{db} + (1-\beta)db^2 \quad \text{large}$$

$(small\ number)^2 \rightarrow$ Becomes Smaller
$(large\ number)^2 \rightarrow$ Becomes Larger

ϵ is a very small number to prevent dividing by zero

$$w_{new} = w_{old} - \frac{\alpha}{\sqrt{s_{dw}} + \epsilon} dw$$

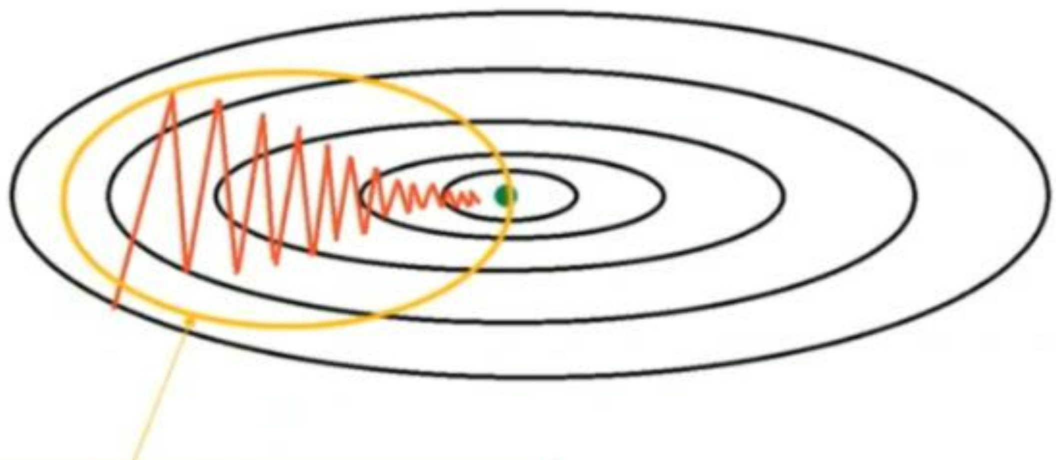
$$b_{new} = b_{old} - \frac{\alpha}{\sqrt{s_{db}} + \epsilon} db$$

Dividing by a small number → gets larger

Dividing by a large number → gets smaller

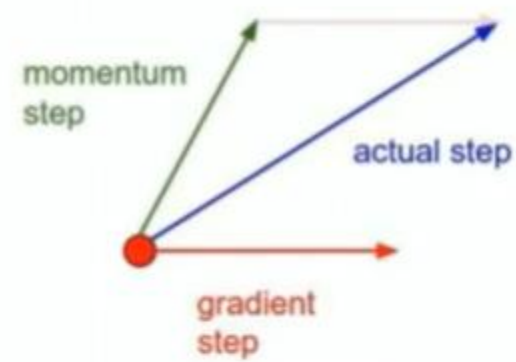
Assume w is parameter 1 and b is parameter 2

Momentum



β *is set to* 0.9 (*robust*)

We can use exponentially moving average to reduce the oscillations in the vertical direction and speed it up in the horizontal direction!



From $W = W - lr * dW$

Momentum weight : $v = \beta v + (1- \beta) dW$

$W = W – lr * v$

Adam

- Adaptive Moment Estimation (Adam) is proposed by Kingman and Ba in 2014
- It is similar to RMSprop with momentum added

```
m = beta1 * m + (1 - beta1) * dW
v = beta2 * v + (1 - beta2) * (dW ** 2)
W += -lr * m / (np.sqrt(v) + eps)
```

SGD, Adam, and RMSprop

“The choice of which algorithm to use, at this point, seems to depend largely on the user’s familiarity with the algorithm” –Goodfellow

-Adam and RMSprop provides faster training time

- SGD is well studied

Most used deep learning algorithm

1. Adam
2. SGD
3. RMSprop

Recommendation: try Adam, then SGD with momentum. Then, RMSprop

Training Cifar using Keras

```
from keras import layers
from keras import models
from keras.datasets import cifar10
from sklearn.preprocessing import LabelBinarizer

model = models.Sequential()
model.add(layers.Conv2D(64, (5, 5), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (5, 5), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(10, activation='softmax'))
model.summary()
```

```python
print("[INFO] loading CIFAR-10 data...")
((trainX, trainY), (testX, testY)) = cifar10.load_data()
trainX = trainX.astype("float") / 255.0
testX = testX.astype("float") / 255.0
lb = LabelBinarizer()
trainY = lb.fit_transform(trainY)
testY = lb.transform(testY)

# initialize the label names for the CIFAR-10 dataset
labelNames = ["airplane", "automobile", "bird", "cat", "deer",
        "dog", "frog", "horse", "ship", "truck"]

model.compile(optimizer='Adam',
        loss='categorical_crossentropy',
        metrics=['accuracy'])

H = model.fit(trainX, trainY, validation_data=(testX, testY),
        batch_size=250, epochs=100, verbose=1)
```

Homework:

- Please run CIFAR-10 with ADAM, RMSPROP and SGD and compare the performance. Try to get the best performance out of each optimization algorithm

Learning Rate Schedulers

Remember alpha parameter for Gradient Descent algorithm:

- If alpha is too high, we can have overshoot case
- If alpha is too low, it will take a long time to reach the optimal value

What’s about if the rate can be modified adaptively?

Adaptive Learning Rate Schedule

- Finding a set of reasonably “good” weights early in the training process with a higher learning rate
- Tuning these weights later in the process to find more optimal weights using a smaller learning rate

Two Approaches for Learning Rate Scheduler

1. Learning rate schedules that decreases gradually based on the epoch number

2. Learning rate schedulers that drop based on specific epoch

Keras Implemenation

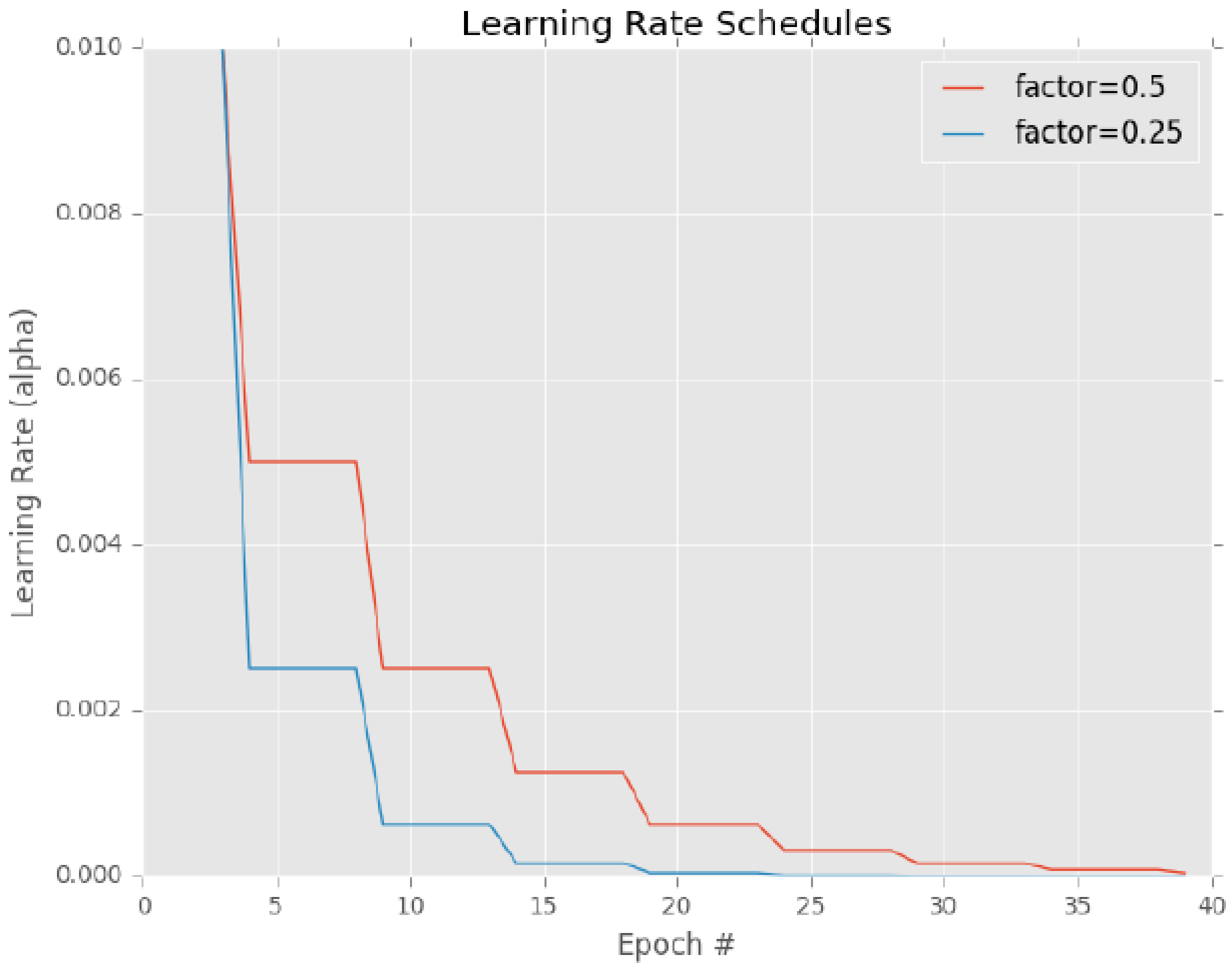
- Keras applies the following learning rate schedule to adjust the learning rate after every batch update

```
lr = init_lr * (1.0 / (1.0 + decay * iterations))
```

Alpha = 0.01 and decay = 0.01/40

Epoch	Learning Rate (α)
0	0.01000
1	0.00836
2	0.00719
...	...
37	0.00121
38	0.00119
39	0.00116

Step-based Decay



Advanced Optimization

Apart from SGD, there are other optimization methods:

- Reduce the amount of time (number of epochs) to obtain reasonable classification accuracy
- Make the network more “well-behaved” for a larger range of hyperparameters other than the learning rate
- Obtain higher accuracy

Spotting Underfitting and Overfitting

- What are underfitting?

- What are overfitting?

You need to be highly concerned with both underfitting and overfitting

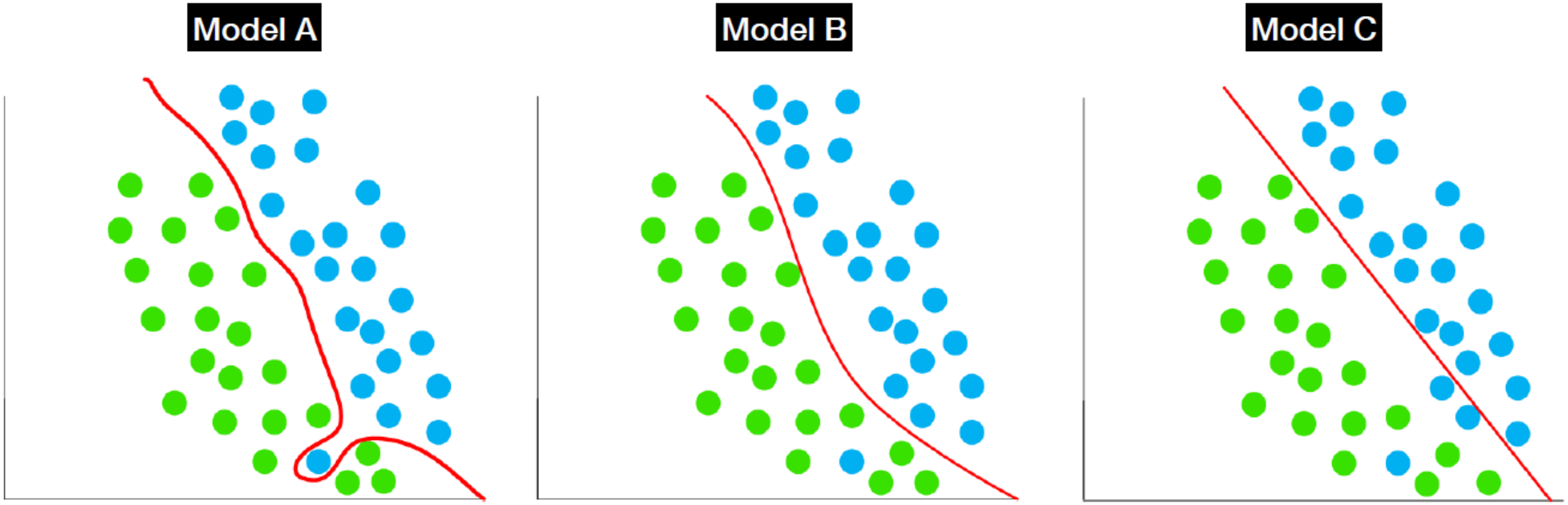
Underfitting

- Underfitting occurs when your model cannot obtain sufficiently low loss on the training set

Overfitting

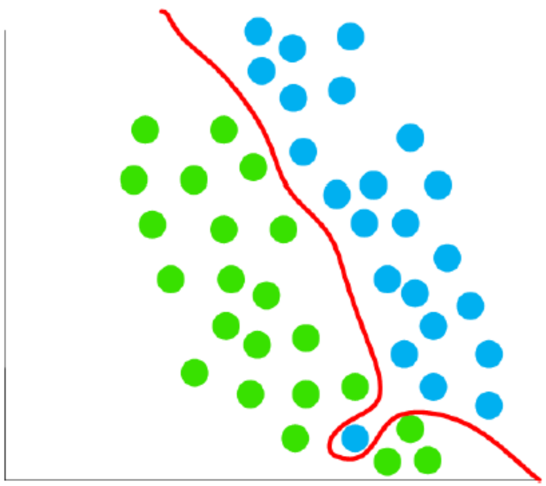
- Overfitting occurs when your model predicts the training data so well and fails to generalize to your validation data

Which model works better?



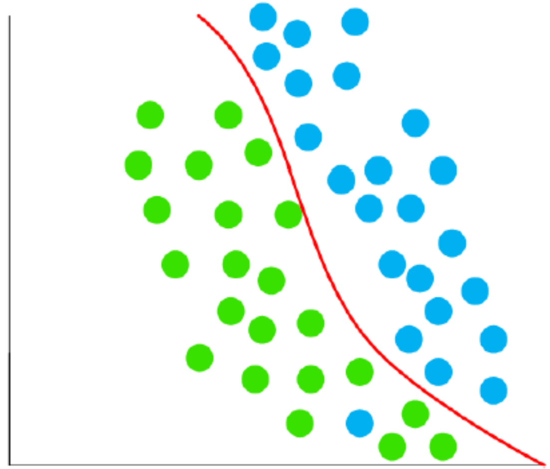
Model B

Overfit

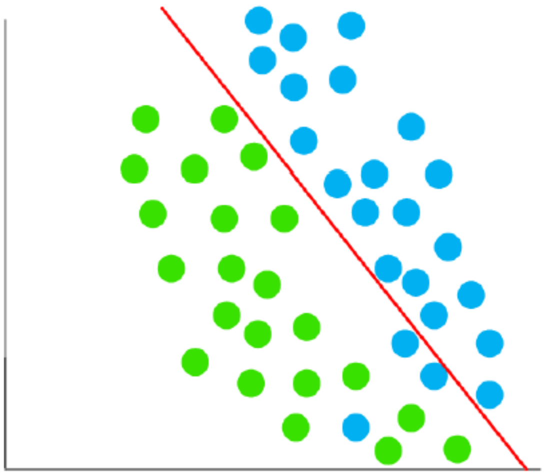


- High Variance
- Fits Noise

Generalised Well

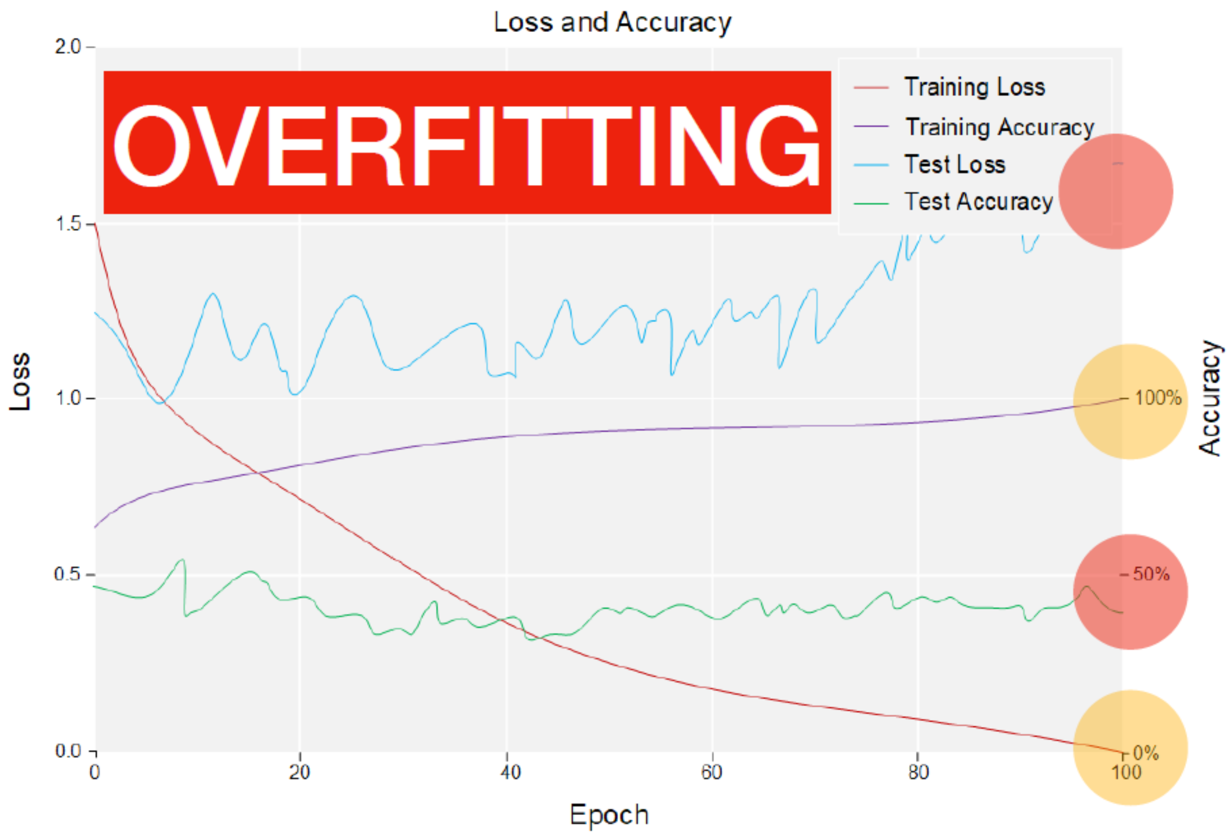


Under-fit



- Low Variance

Training accuracy vs. Test accuracy



Regularization

- Is a technique to generalize the model

- Is a technique used as an attempt to reduce overfitting

Regularization Techniques

- L1 & L2 Regularization
- Early Stopping
- Data Augmentation
- Drop Out
- Batch Normalization

L1 & L2 Regularization

- The idea is to force parameters (weights and biases) to take small values

- This helps reduce a node with big weights, but having many nodes with small weights instead

L1 Regularization

$$\text{Loss Function} + \lambda \sum_{j=1}^{p} |\beta_j|$$

- β are our weights and λ is a controlled parameter (the value is less than 1)

L2 Regularization

$$LossFunction + \lambda \sum_{j=1}^{p} \beta_j^2$$

- β are our weights and λ is a controlled parameter (the value is less than 1)

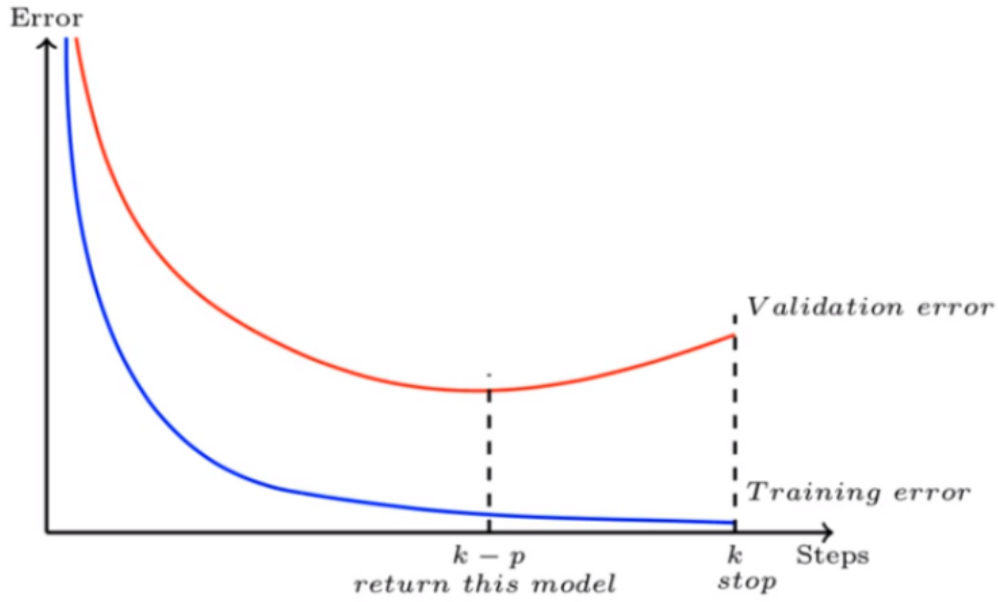
Differences between L1 and L2

- Both try to shrink the weight toward 0
- L2 penalizes large weights more
- λ is used to control the weight:
  - Small means we prefer to minimize the original lost function
  - Large means prefers small weights

Early Stopping

- At some points during training process, our validation loss may stop decreasing
- At this point, we should not continue training
- This method is called early stopping

How Early Stopping reduces overfitting



- Early stopping ensures that our model does not learn pattern of noise during training

Question?