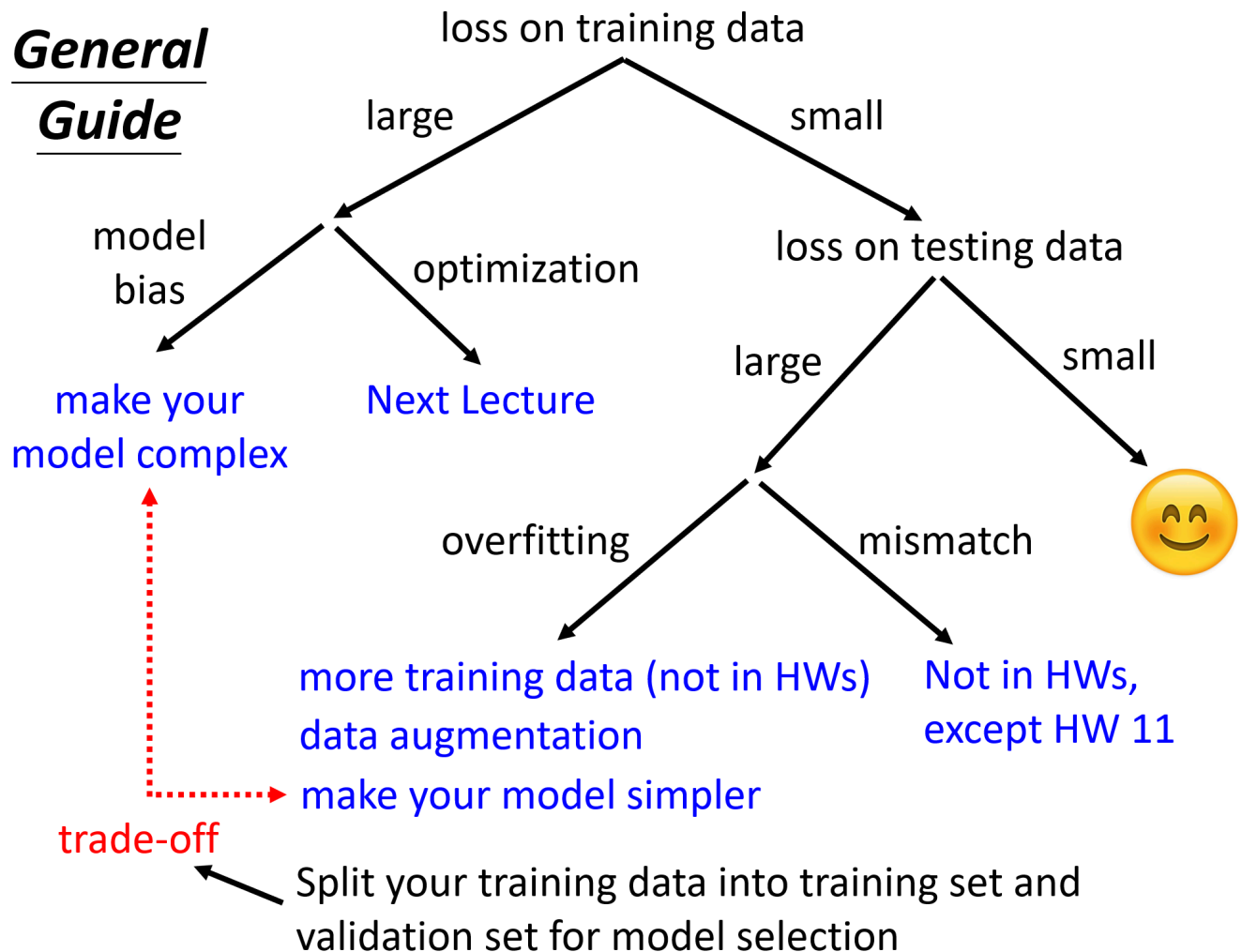


Roadmap of Improving Model

General Guide



After training, we may find that...

Loss is large

The model doesnot perform well even in the traning set.

Two possibilities: Model Bias or Optimization issues?

Model Bias: the model is too simple

See the training process as finding a function in a function set. Model bias means that all functions in the function set cannot reduce the loss to the desired level.

Solution: redesign the model to make it more flexible

- more features
- (in DL) more layers

Optimization

development of optimizers




The optimization approach cannot find the way to the lowest Loss

1. **Case1 :critical point.** use *Second Partial Test* to confirm

- local minima (very rare when the model is complex)
- saddle point

Solutions to escape from the critical point:

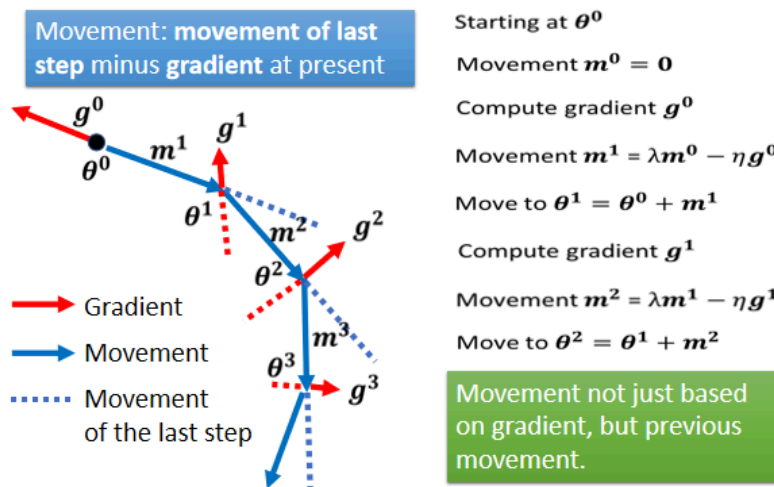
1. use Batch Gradient Descent

	Small	Large
Speed for one update (no parallel)	Faster	Slower
Speed for one update (with parallel)	Same	Same (not too large)
Time for one epoch	Slower	Faster 
Gradient	Noisy	Stable
Optimization	Better 	Worse
Generalization	Better 	Worse

Small Batch v.s. Large Batch

Batch size is a hyperparameter you have to decide.

2. use Momentum (adjust the optimization direction with the last optimization)



λ is the momentum hyperparameter

2. **Case2: skip over (the learning rate is so high)**

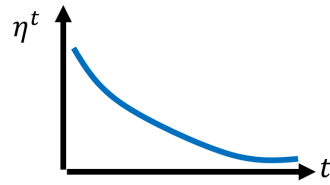
Solution: Adaptive Learning Rate

- Adagrad: use Root Mean Square of *all gradients in history* to update the learning rate.
 - if the RMS is big, decrease the learning rate
 - if the RMS is small, increase the learning rate

- RMSProp: optimize Adagrad by adding a hyperparameter α which weights the current gradient's effect on learning rate

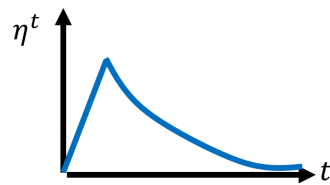
The most popular optimizer Adam = RMSProp + Momentum

- Learning Rate Scheduling
 - Learning Rate Decay
 - Warm Up (useful in training BERT)



Learning Rate Decay

As the training goes, we are closer to the destination, so we reduce the learning rate.



Warm Up

Increase and then decrease?

How to identify Model Bias / Optimization issue

Start from smaller / shallower networks, which are easier to optimize.

If **deeper networks do not obtain smaller loss on training data**, then there is optimization issue.

Loss is small

Loss on training data is small, which is good.

But when it turns to testing data, if the Loss is large, there are two possibilities:

Overfitting or Mismatch?

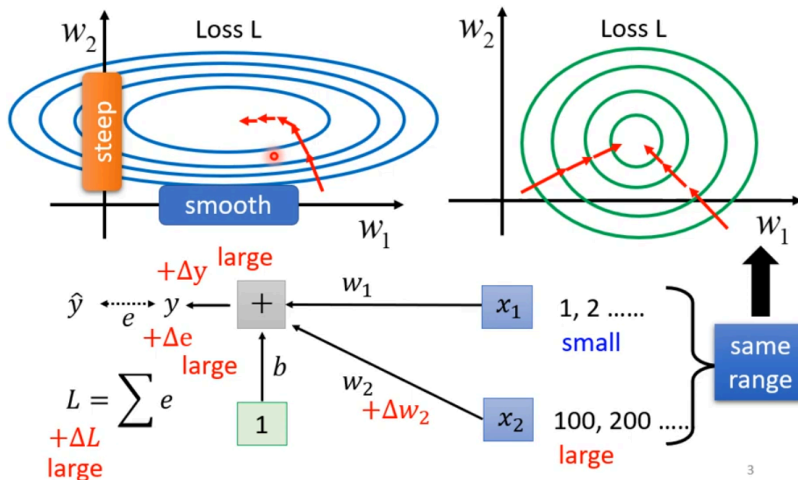
Overfitting

Solution:

- More training data
 - data augmentation e.g. flip the picture
- Simplify the model
 - Less parameters / neurons, sharing parameters(e.g. CNN's conv kernel)
 - Less features
- Regularization
 - L1, L2 regularization
 - add penalties to large parameters
- Early stopping

- stop the training when the loss cannot decrease
- Dropout
 - discard random neurons to penalize the complexity of the network structure
- Batch Normalization

Changing Landscape



- normalization: scaling the features to a standard range to improve training efficiency
- batch: we scale a batch of features instead of all features in DL, **but the batch should be large enough to approximate the whole set**
- internal covariate shift ...

Mismatch

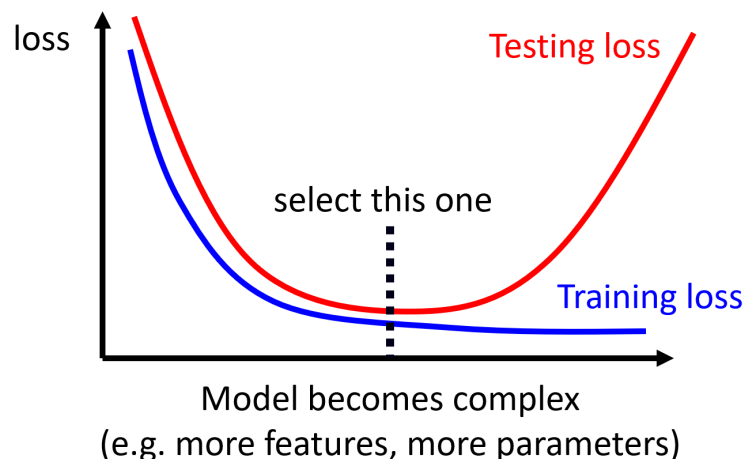
The training data and testing data have different distributions

...

Bias-Complexity Trade-off

Low complexity --> Model Bias

High complexity --> Overfitting



so it need to balance manually by **Cross Validation**:

Split the dataset into *training set* and *validation set*. The Loss on validation set measures the result.