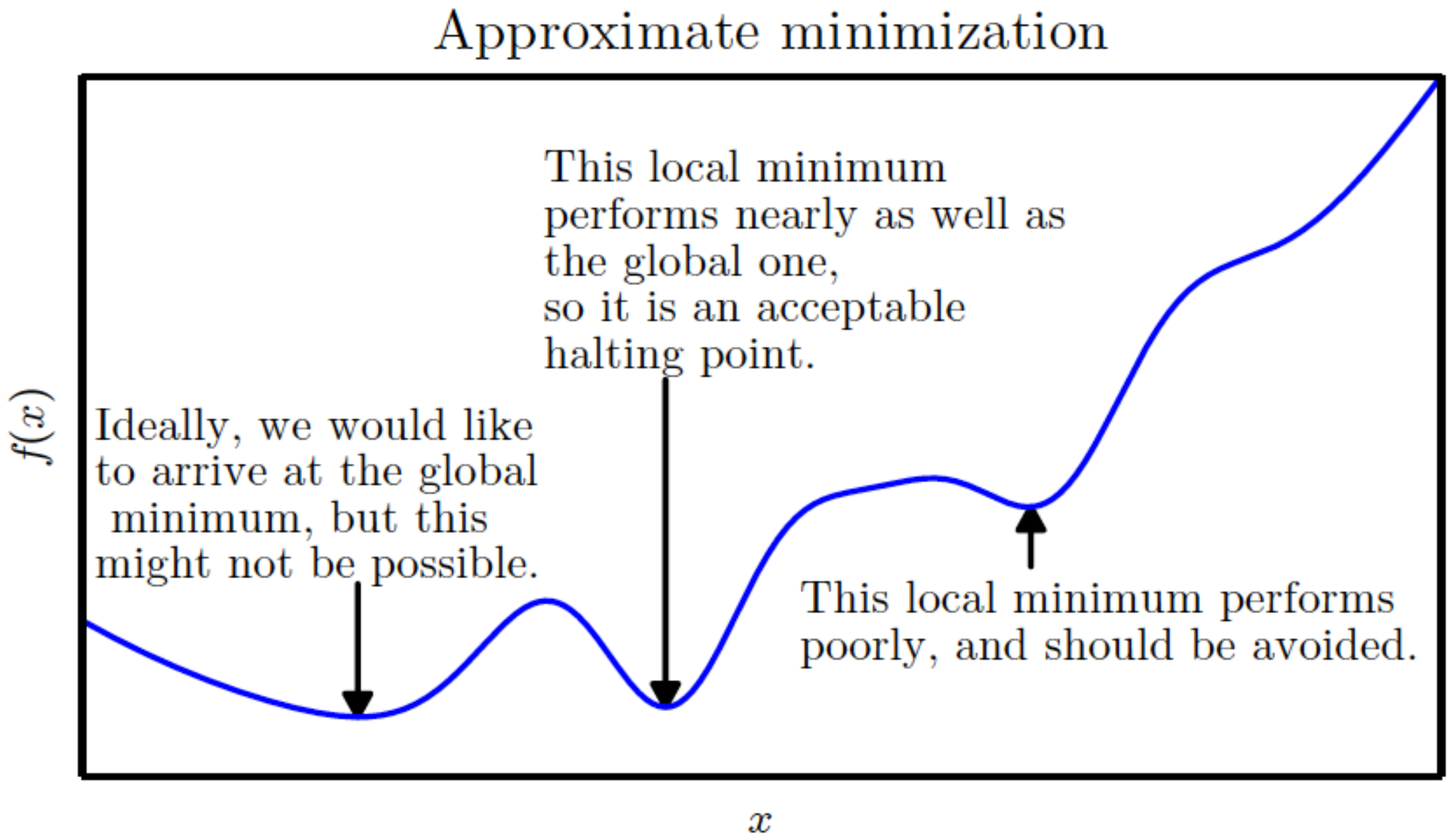
Deep Learning

Lecture 03: Optimization and Regularization
优化与正则化

Wanxiang Che

Optimization

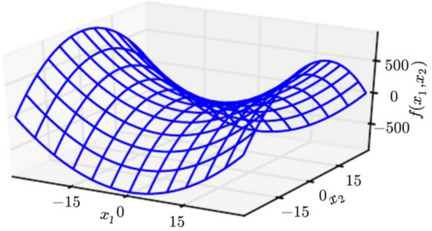
Challenges in NN Optimization

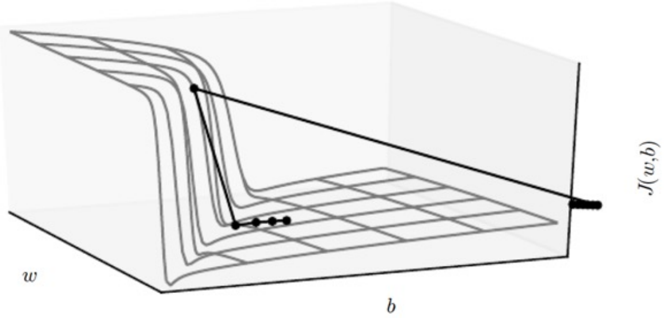


Challenges in NN Optimization

[Goodfellow et al., 2016. Section 8.2]

- Local Minima
- Plateaus, **Saddle Points** and Other Flat Regions
- Cliffs and Exploding Gradients
- Learning **Long-Term Dependencies**
  - Exploding or Vanishing Product Jacobians
- Inexact Gradients
- Theoretical Limits of Optimization





Optimizing (Learning) Algorithms

- (Batch) Gradient Descent

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \epsilon \nabla_{\boldsymbol{\theta}} \sum_t L(f(\boldsymbol{x}^{(t)}; \boldsymbol{\theta}), \boldsymbol{y}^{(t)}; \boldsymbol{\theta})$$

- Stochastic Gradient Descent (SGD)
  - Online GD (m = 1), Minibatch SGD (m > 1)

Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ϵ_k.
Require: Initial parameter $\boldsymbol{\theta}$
  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: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$
  end while

Batch or Minibatch?

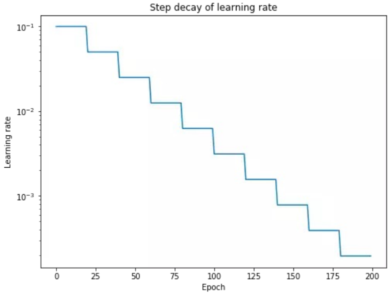
- Batch
  - Slow to estimate gradient
  - More exactly
- Minibatch
  - Fast to estimate gradient
  - Standard error of a mean estimated from *n* samples $\hat{\sigma}/\sqrt{n}$
    - there are less than linear returns to using more examples to estimate the gradient
    - 100 examples vs. 10,000 examples

Factors of Minibatch Sizes

- Larger batches provide a more accurate estimate of the gradient, but with less than linear returns
- Multicore architectures are usually underutilized by extremely small batches
- When using GPU, it is common for power of 2 batch sizes (32-256) to offer better runtime

How to Choose Learning Rate (学习率)?

- Too high → Oscillation; Too low → Slow
- Gradually decrease the learning rate
  - Step decay



  - Exponential decay: $\alpha = \alpha_0 e^{-kt}$
  - 1/t decay: $\alpha = \alpha_0 / (1 + kt)$

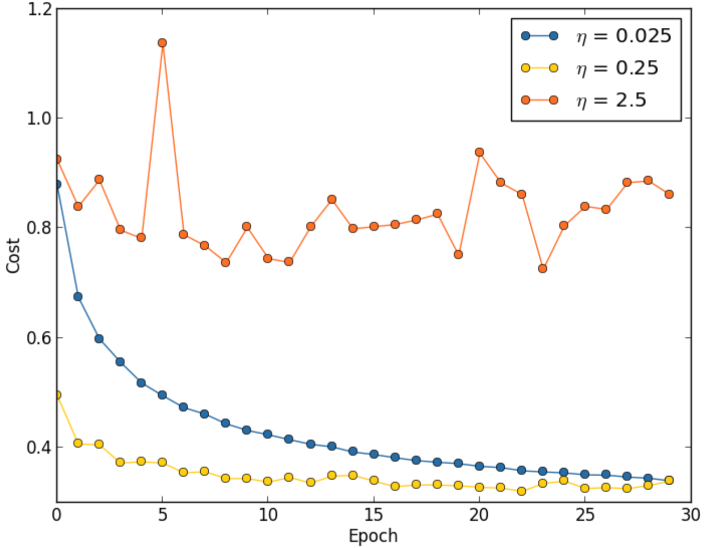
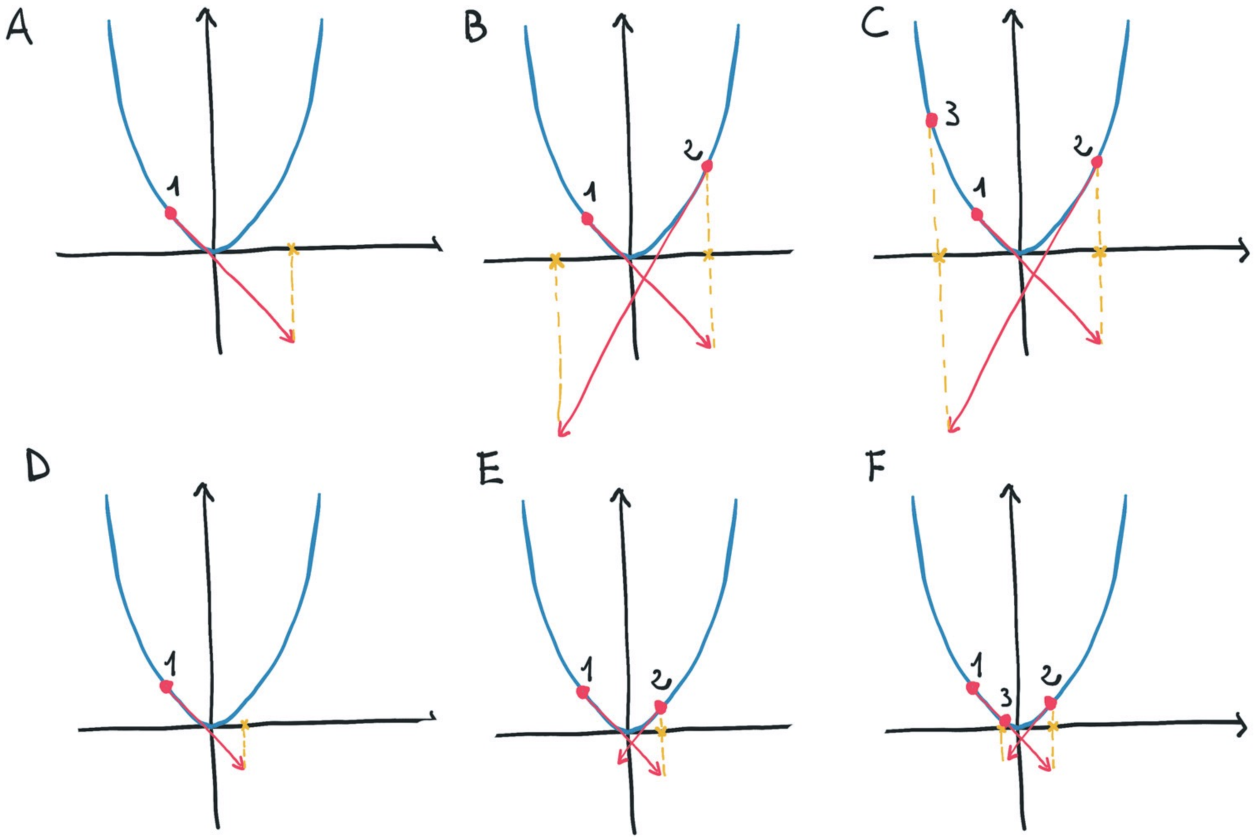


Illustration of different learning rates



Momentum

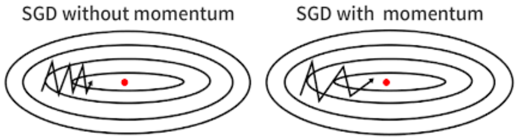
- SGD sometimes can be slow
  - high curvature, small but consistent gradients, or noisy gradients
- The method of momentum Polyak (1964) is designed to **accelerate learning**
- Intuition
  - Derived from a physical interpretation of the optimization process

Momentum

- A variable *v* that plays the role of velocity (or momentum) that accumulates gradient

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

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




```
sgd = keras.optimizers.SGD(lr=0.01, momentum=0.9)
```

Acc: ~98%

Nesterov Momentum (Sutskever *et al*. 2013)

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

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

$\Rightarrow$

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

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

Algorithm 8.3 Stochastic gradient descent (SGD) with Nesterov momentum

Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
  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 labels $\boldsymbol{y}^{(i)}$.
    Apply interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
    Compute gradient (at interim point): $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while

```
sgd = keras.optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
```

Acc: ~98%

AdaGrad

- **Individually** adapts the learning rates of all model parameters by scaling them inversely proportional to an accumulated sum of squared partial derivatives over all training iterations
- Empirically, AdaGrad results in a premature and excessive decrease in the effective learning rate

Algorithm 8.4 The Adagrad algorithm

Require: Global learning rate η,
Require: Initial parameter $\boldsymbol{\theta}$
  Initialize gradient accumulation variable $\boldsymbol{r} = \mathbf{0}$,
  while Stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$.
    Set $\boldsymbol{g} = \mathbf{0}$
    for $i = 1$ to m **do**
      Compute gradient: $\boldsymbol{g} \leftarrow \boldsymbol{g} + \nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    end for
    Accumulate gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g}^2$ (square is applied element-wise)
    Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\eta}{\sqrt{\boldsymbol{r}}}\boldsymbol{g}$. % ($\frac{1}{\sqrt{\boldsymbol{r}}}$ applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}_t$
  end while

RMSprop (Hinton, 2012)

- Addresses the deficiency of AdaGrad by changing the gradient accumulation into an **exponentially** weighted moving average

Algorithm 8.5 The RMSprop algorithm

Require: Global learning rate η, decay rate ρ.
Require: Initial parameter $\boldsymbol{\theta}$
  Initialize accumulation variables $\boldsymbol{r} = 0$
  while Stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$.
    Set $\boldsymbol{g} = \mathbf{0}$
    for $i = 1$ to m **do**
      Compute gradient: $\boldsymbol{g} \leftarrow \boldsymbol{g} + \nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    end for
    Accumulate gradient: $\boxed{\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g}^2}$
    Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\eta}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$.   % ($\frac{1}{\sqrt{\boldsymbol{r}}}$ applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
  end while

Algorithm 8.6 RMSprop algorithm with Nesterov momentum

Require: Global learning rate η, decay rate ρ, momentum coefficient α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
  Initialize accumulation variable $\boldsymbol{r} = \mathbf{0}$
  while Stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$.
    Compute interim update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
    Set $\boldsymbol{g} = \mathbf{0}$
    for $i = 1$ to m **do**
      Compute gradient: $\boldsymbol{g} \leftarrow \boldsymbol{g} + \nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    end for
    Accumulate gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g}^2$
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\eta}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$.   % ($\frac{1}{\sqrt{\boldsymbol{r}}}$ applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while

Adam (Kingma and Ba, 2014)

- As a variant on RMSprop + momentum with a few important distinctions
  - Momentum is incorporated directly as an estimate of the first order moment (with exponential weighting) of the gradient
  - Adam includes bias corrections to the estimates of both the first-order moments (the momentum term) and the (uncentered) second order moments to account for their initialization at the origin

Algorithm 8.7 The Adam algorithm

Require: Step-size α
Require: Decay rates ρ_1 and ρ_2, constant ϵ
Require: Initial parameter $\boldsymbol{\theta}$
  Initialize 1st and 2nd moment variables $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{0}$,
  Initialize timestep $t = 0$
  while Stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$.
    Set $\boldsymbol{g} = \boldsymbol{0}$
    for $i = 1$ to m **do**
      Compute gradient: $\boldsymbol{g} \leftarrow \boldsymbol{g} + \nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    end for
    $t \leftarrow t + 1$
    Get biased first moment: $\boxed{\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}}$
    Get biased second moment: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g}^2$
    Compute bias-corrected first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t}$
    Compute bias-corrected second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1-\rho_2^t}$
    Compute update: $\Delta\boldsymbol{\theta} = -\alpha \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}}+\epsilon} \boldsymbol{g}$ % (operations applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while

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

Acc: >98%

AdaDelta

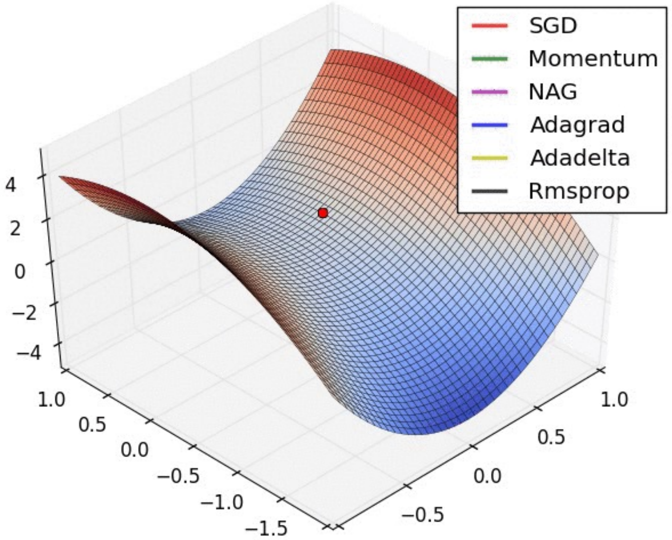
- Another recently introduced optimization algorithm that seeks to directly address problems with AdaGrad, while incorporating some second-order gradient information

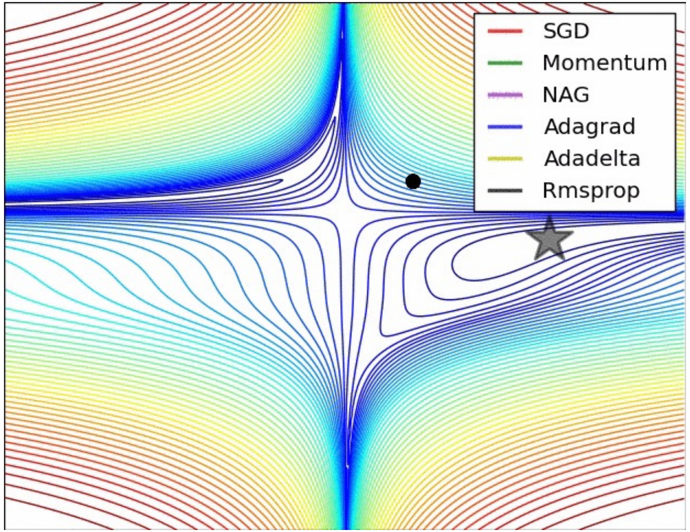
Algorithm 8.8 The Adadelta algorithm

Require: Decay rate ρ, constant ϵ
Require: Initial parameter $\boldsymbol{\theta}$
  Initialize accumulation variables $\boldsymbol{r} = \mathbf{0}$, $\boldsymbol{s} = \mathbf{0}$,
  while Stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$.
    Set $\boldsymbol{g} = \mathbf{0}$
    for $i = 1$ to m **do**
      Compute gradient: $\boldsymbol{g} \leftarrow \boldsymbol{g} + \nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    end for
    Accumulate gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g}^2$
    Compute update: $\Delta \boldsymbol{\theta} = -\frac{\sqrt{\boldsymbol{s} + \epsilon}}{\sqrt{\boldsymbol{r} + \epsilon}} \boldsymbol{g}$ % (operations applied element-wise)
    Accumulate update: $\boxed{\boldsymbol{s} \leftarrow \rho \boldsymbol{s} + (1 - \rho) [\Delta \boldsymbol{\theta}]^2}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
  end while

Which algorithm should one choose?

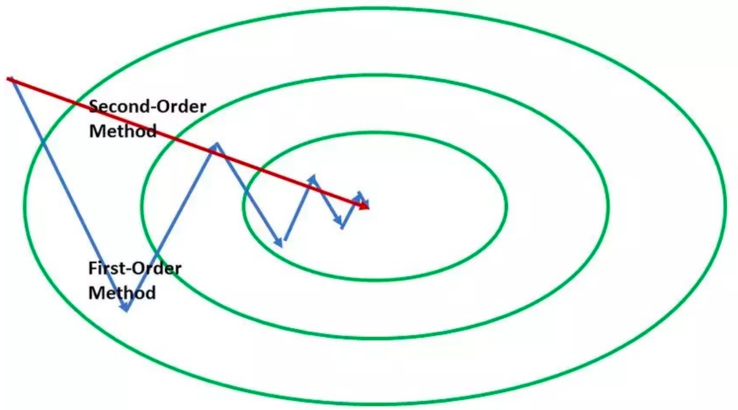
- Unfortunately, there is currently no consensus
- Depend on the users familiarity with the algorithm (for ease of hyperparameter tuning)





Approximate Second-Order Methods

- Newton's Method
- Conjugate Gradients
- BFGS (Broyden – Fletcher – Goldfarb – Shanno)
  - L-BFGS (Limited Memory BFGS)



Optimization Strategies

- Batch Normalization
- Initialization Strategies
- Supervised Pretraining
- Designing Models to Aid Optimization

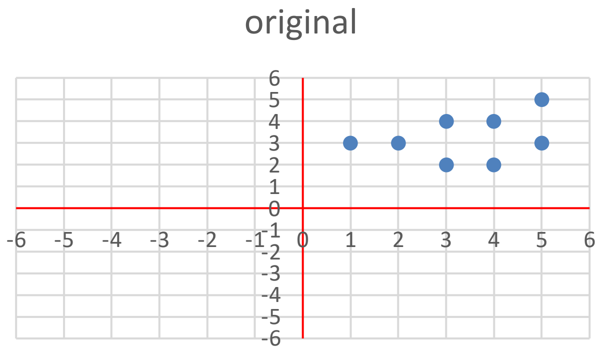
Optimization Strategies

- Batch Normalization
- Initialization Strategies
- Supervised Pretraining
- Designing Models to Aid Optimization

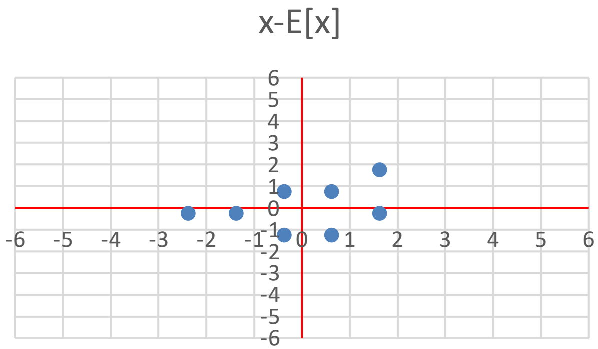
Batch Normalization

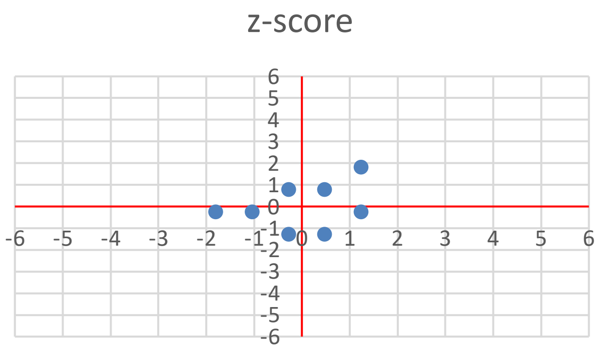
- Sergey Ioffe, Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. International Conference on Machine Learning (ICML). 2015.
- One of the most exciting recent innovations in optimizing deep neural networks
  - Faster training
  - Better generalization

Preprocess (Normalization)



$$x' = x - \mathrm{E}[x]$$



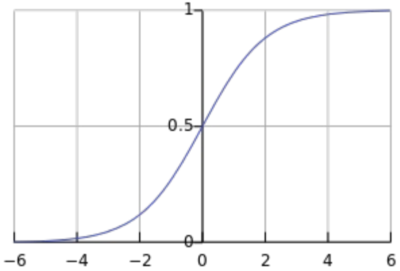


$$x' = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]}}$$

Fixed Distribution

$$z = g(W\mathrm{u} + \mathrm{b})$$

$$g(x) = \frac{1}{1+\exp(-x)} \qquad \mathrm{x} = W\mathrm{u}+\mathrm{b}$$



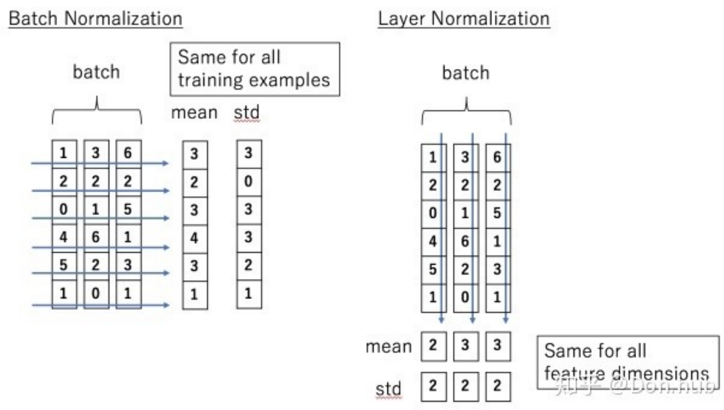
- As |x| increase, g’(x) tends to zero
- Changes to those parameters W and b during training will likely move **many dimensions** of x into the **saturated regime** of the nonlinearity and **slow down the convergence**
- Solution: ReLU activation function
- Other: Ensure that the distribution of nonlinearity inputs **remains more stable**

Batch Normalization

- $x^{(k)}$ represent a dimension of input
- **Simply normalizing** each input of a layer may change what the layer can represent
- $\gamma^{(k)}, \beta^{(k)}$ are learned along with the original model parameters, and **restore the representation power of the network**.

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}.$$



How to use BN at test time?

Advantages

- Batch Normalization enables higher learning rates
- Batch Normalization makes training more resilient to the parameter scale
- Batch Normalization regularizes the model

```
model = keras.Sequential()
# model.add(keras.layers.Dense(512, activation='relu', input_shape=(784,)))
model.add(keras.layers.Dense(512, input_shape=(784,)))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Activation('relu'))
model.add(keras.layers.Dense(num_classes, activation='softmax'))
```

Optimization Strategies

- Batch Normalization
- Initialization Strategies
- Supervised Pretraining
- Designing Models to Aid Optimization

Initialization Strategies

- Most algorithms are strongly affected by the choice of initialization
- “break symmetry” ➔ random initialization
- Too large or too small are both bad

$$\mathrm{W}_{i,j} \sim U\left(-\frac{6}{\sqrt{m+n}}, \frac{6}{\sqrt{m+n}}\right)$$

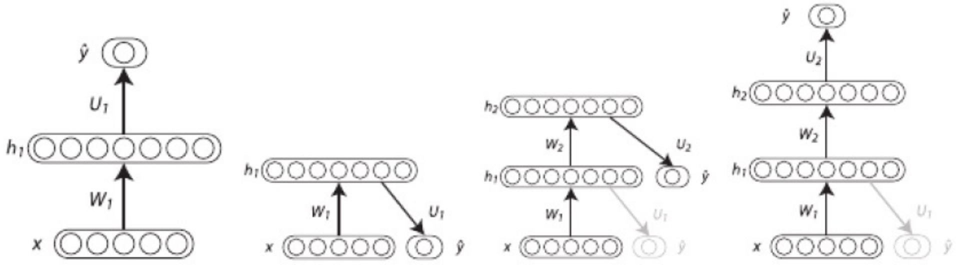
$$W^{[l]} \sim \mathcal{N}\left(\mu = 0, \sigma^2 = \frac{1}{n^{[l-1]}}\right)$$
$$b^{[l]} = 0$$

Xavier initialization

Optimization Strategies

- Batch Normalization
- Initialization Strategies
- Supervised Pretraining
- Designing Models to Aid Optimization

Supervised Pretraining (Greedy Pretraining)



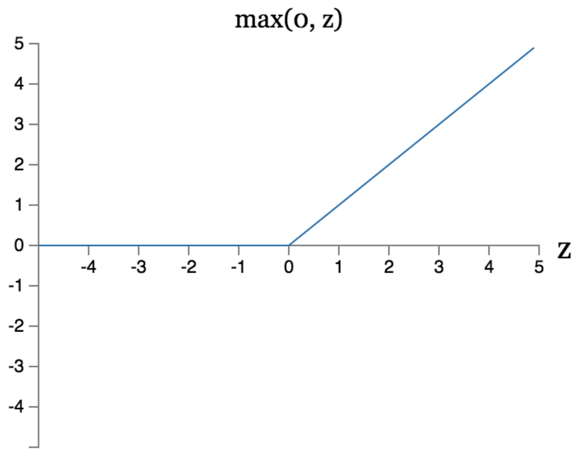
- Some related work
  - Transfer learning
  - *FitNets* (Romero *et al*. 2015)

Optimization Strategies

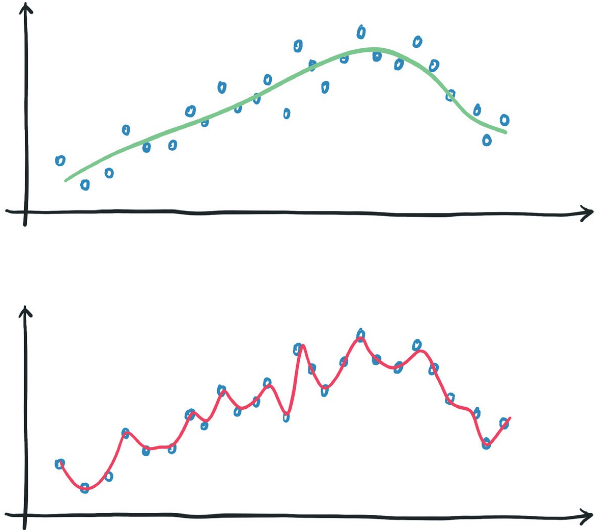
- Batch Normalization
- Initialization Strategies
- Supervised Pretraining
- Designing Models to Aid Optimization

Designing Models to Aid Optimization

- Designing models to be easier to optimize, rather than improve the optimization algorithm
- For example
  - LSTM, rectified linear units (max(0, z))
  - Linear paths or skip connections between layers (Srivastava *et al*. 2015)



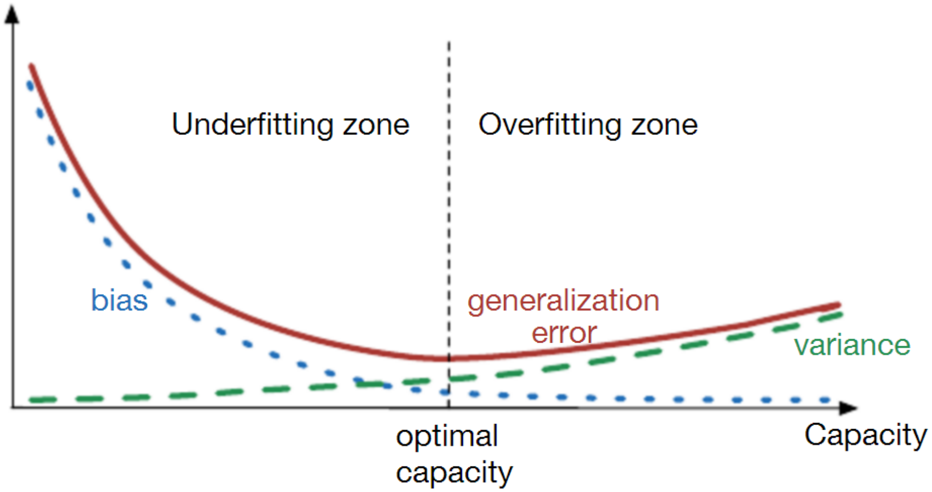
Regularization



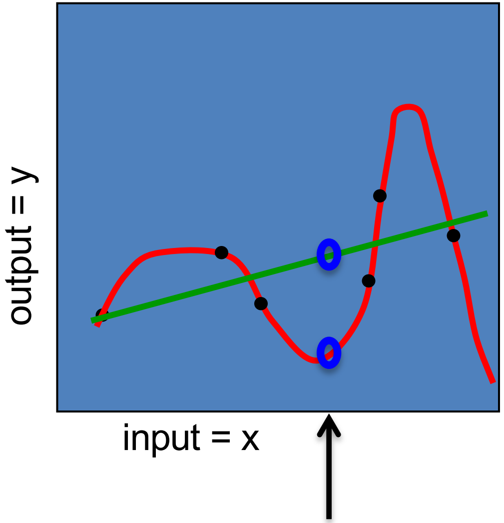
General Purpose of Regularization

- Make machine learning algorithm not only perform well on training data, but also on new inputs. [**minimizing generalization error**]
- Regularization
  - <u>strategies</u> to **reduce test error**, possibly at expense of increase training error.
- Functions of regularization
  - encode specified knowledge
  - express a generic preference of simple model
  - (ensemble method) combine multiple hypotheses

Generalization Error in Term of: Fitting Data Generation Process



A Simple Example of Overfitting



Which output value should
you predict for this test input?

- Which model do you trust?
  - The complicated model fits the data better.
  - But it is not economical.
- A model is convincing when it fits a lot of data surprisingly well.
  - It is not surprising that a complicated model can fit a small amount of data well.

Examples of Regularization

General form

$$\tilde{\mathcal{J}}(\theta) = \mathcal{J}(\theta) + \alpha\Omega(\theta)$$

- Parameter norm penalty
  - $\tilde{\mathcal{J}}(\theta) = \mathcal{J}(\theta) + \alpha\left||w|\right|^2$
- Entropy regularizer (Bengio 2005)
  - $\tilde{\mathcal{J}}(\theta) = \mathcal{J}(\theta) + \alpha H(z_i)$
- Generalized expectation (Mann and McCallum, 2008)
  - $\tilde{\mathcal{J}}(\theta) = \mathcal{J}(\theta) + \alpha KL(y||\hat{y})$

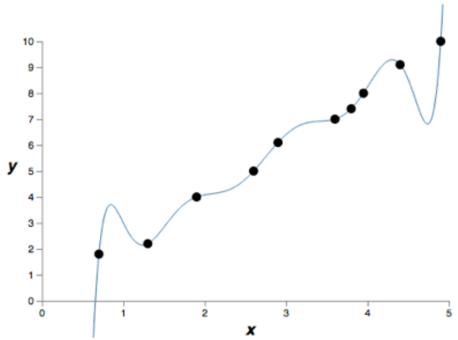
Norm (范数) Penalty in Classical Perspective

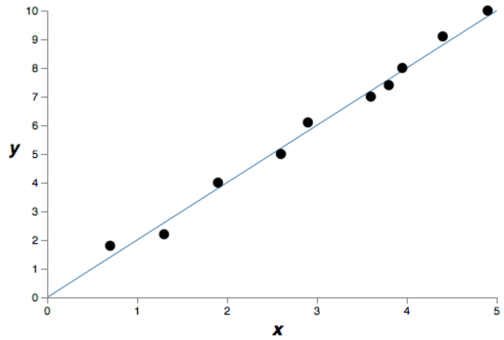
- L2 regularizer: $\Omega(\theta) = \frac{1}{2}\sum_i |\theta_i^2|$
- L1 regularizer: $\Omega(\theta) = \sum_i |\theta_i|$

```
model = keras.Sequential()
model.add(keras.layers.Dense(512, kernel_regularizer=keras.regularizers.l2(1e-4), activation='relu', input_shape=(784,)))
model.add(keras.layers.Dense(num_classes, kernel_regularizer=keras.regularizers.l2(1e-4), activation='softmax'))
```

Intuition 1 (on L2)

- L2 regularizer control θ and penalize on large θ





Intuition 2 (on L2): Single Update Step View

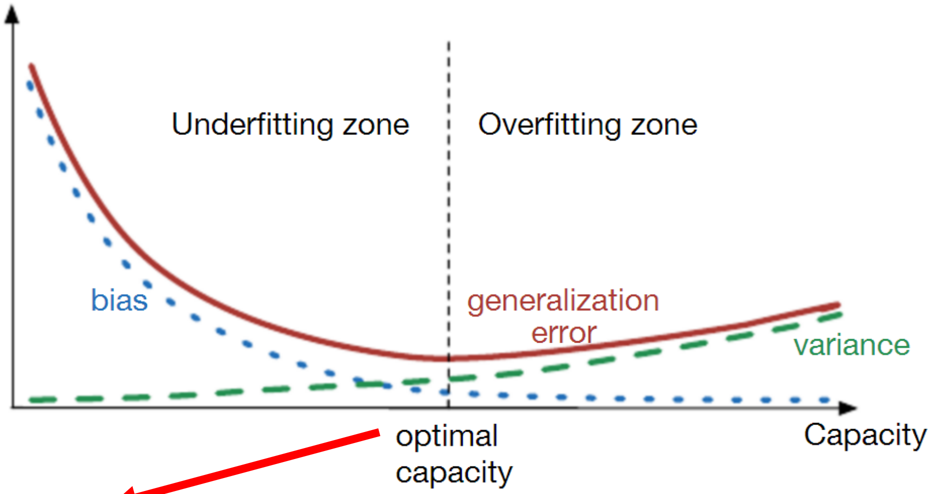
- gradient:
  - $\nabla_\theta \tilde{\mathcal{J}}(\theta) = \nabla_\theta \mathcal{J}(\theta) + \alpha\theta$
- one gradient update
  - $\theta^{new} = \theta^{old} - \epsilon\left(\alpha\theta^{old} + \nabla_\theta \mathcal{J}\left(\theta^{old}\right)\right)$
- write it in other way
  - $\theta^{new} = \boxed{(1 - \alpha\epsilon)\theta^{old}} - \epsilon\nabla_\theta \mathcal{J}\left(\theta^{old}\right)$

at each step, make θ a little smaller

Dataset Argument

- Improve generalization error in other way
  - feed more data
- However, lack data
- Solution1: making fake data
  - Can not distinguish between “b” and “d”
- Solution 2
  - Impose random noise to input data

Early Stopping



Decide the best generalization error model by a validation set
tune the hyperparameter of iteration number

Early Stopping (cont.)

- Early stopping:
  - for iteration in max_iteration:
    - learn model on training data
    - valuate_score_of_current_model = evaluate on validation data
    - if valuate_score_of_current_model > best_validation_score:
      - best_validation_score = valuate_score_of_current_model
      - save current model
      - **patient = 0**
    - **else: patient += 1**
    - **if patient > max_patient:**
      - **break**

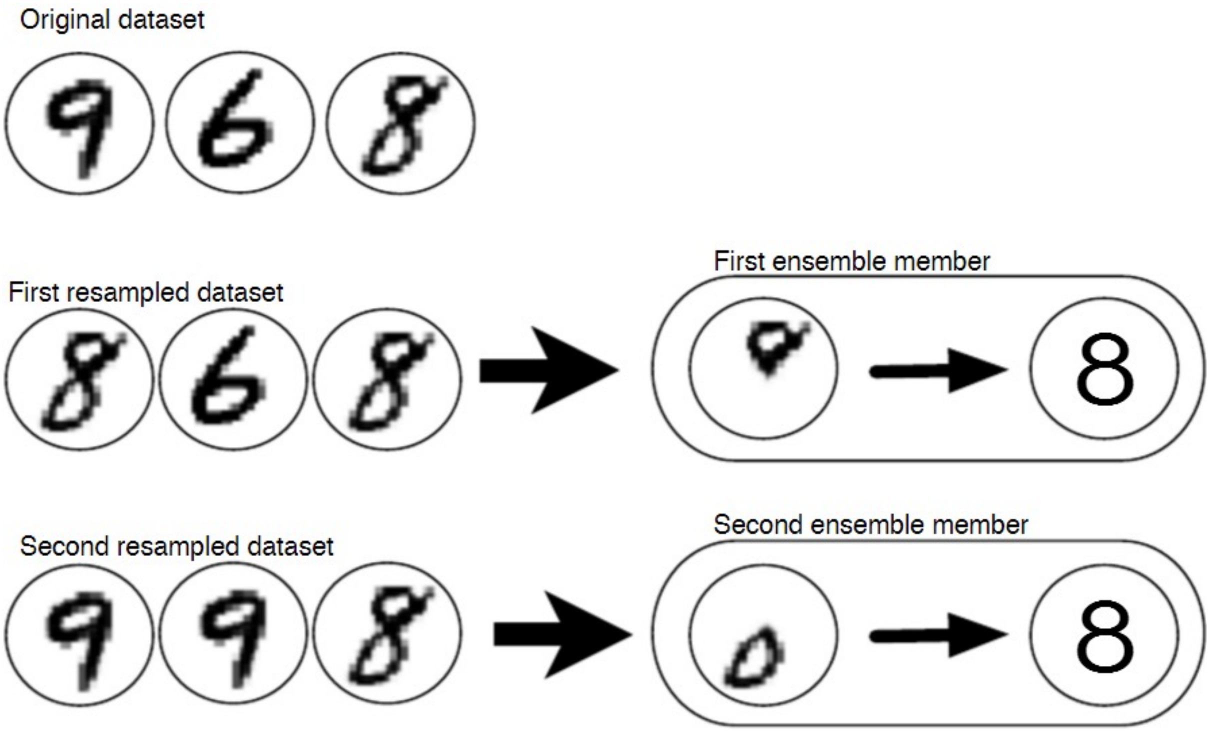
Early Stopping (cont.)

- To make use of the validation set
  - Second pass training
    - tune the best iteration *BEST*
    - train on the merge of {train, validation} set with *BEST* iteration.
  - Continuous training
    - continue training on the merge of {train, validation}
    - stop when $\mathcal{J}$ on validation set reach a small value

Bagging in Practice (9-fold)

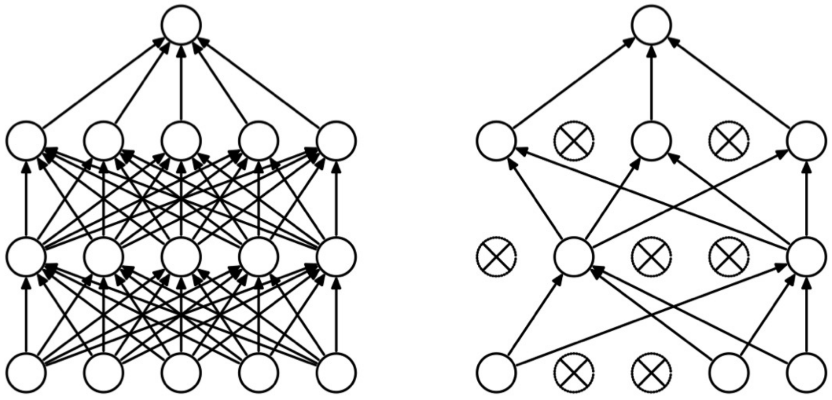
- Considering a binary classification problem on training set D.
- We randomly sample 9 subsets: $\{D_1, D_2, \dots, D_9\}$
  - maybe $|D_i| = 0.63|D|$
- Obtain 9 models on different subset: $\{M_1, M_2, \dots, M_9\}$
- On test phase, these 9 models give 9 results:
  - if 5 or more positive, we got positive
  - otherwise, negative

Cartoon Depiction of How Bagging Works



Dropout

- Hinton et al. 2012
- Proved a powerful tool for training neural network
- Description: during learning phase, for an input $v \in \mathbb{R}^n$, randomly generate a vector $d \in \mathbb{R}^n$, where $d_i = 0$ if rand() < threshold



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

Dropout as Bagging

- Dropout like bagging
  - many different models are trained on different subsets of the data
- Dropout dislike bagging
  - each model is trained for only one step and all of the models share parameters

```
model = keras.Sequential()
# model.add(keras.layers.Dense(512, activation='relu', input_shape=(784,)))
model.add(keras.layers.Dense(512, input_shape=(784,)))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Activation('relu'))
model.add(keras.layers.Dropout(0.2))
model.add(keras.layers.Dense(num_classes, activation='softmax'))
```

How to Choose Hyper-parameters?

1. Reducing the size of training data and network structure to get rapid insight
2. Repeatedly tuning one of hyper-parameters to get better results
   – Learning rate
   – Regularization parameter
   – Mini-batch size
   – ……
3. More training data and complicated network structure, goto 2

- More of an art than a science
- Automated techniques: grid search

Summary

- Optimization
  - SGD → Momentum → Nesterov Momentum
  - AdaGrad → RMSprop → Adam → AdaDelta
  - Other optimization strategies
- Regularization
  - L1/L2, Early stopping, Dropout