Empirical Risk Minimization: Basic Concepts and Optimization Techniques

Chih-Jen Lin

National Taiwan Univ.

MBZUAI

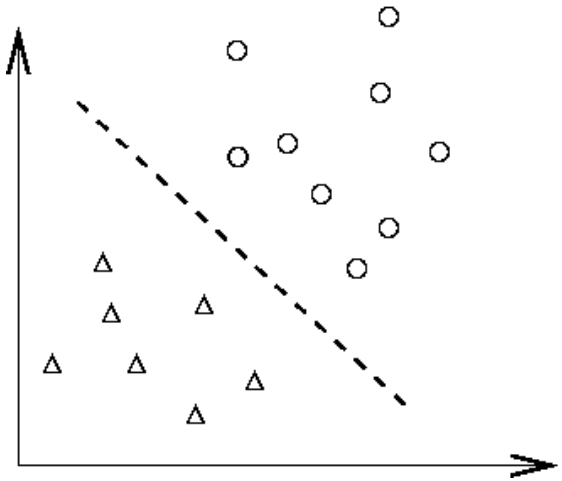
August 2023

Outline

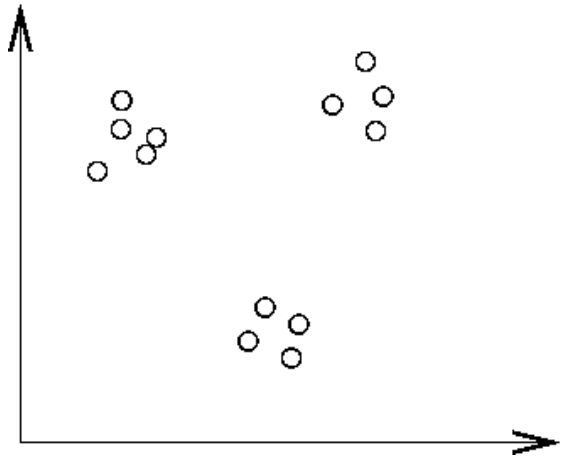
Outline

What is Machine Learning?

- Extract knowledge from data



Classification



Clustering

- We focus on classification. From data with labels, we build a model for prediction

Data Classification

- Given training data in different classes (labels known)

  Predict test data (labels unknown)
- Classic example
  1. Find a patient's blood pressure, weight, etc.
  2. After several years, know if he/she recovers
  3. Build a machine learning model
  4. New patient: find blood pressure, weight, etc
  5. Prediction
- Two main stages: training and testing

Outline

Outline

Minimizing Training Errors

- Basically a classification method starts with minimizing the training errors

$$\min_{\text{model}} \quad (\text{training errors})$$

- That is, all or most training data with labels should be correctly classified by our model
- A model can be a decision tree, a neural network, or other types
- This is called empirical risk minimization

Minimizing Training Errors (Cont'd)

- For simplicity, let's consider the model to be a vector $\boldsymbol{w}$
- That is, the decision function is

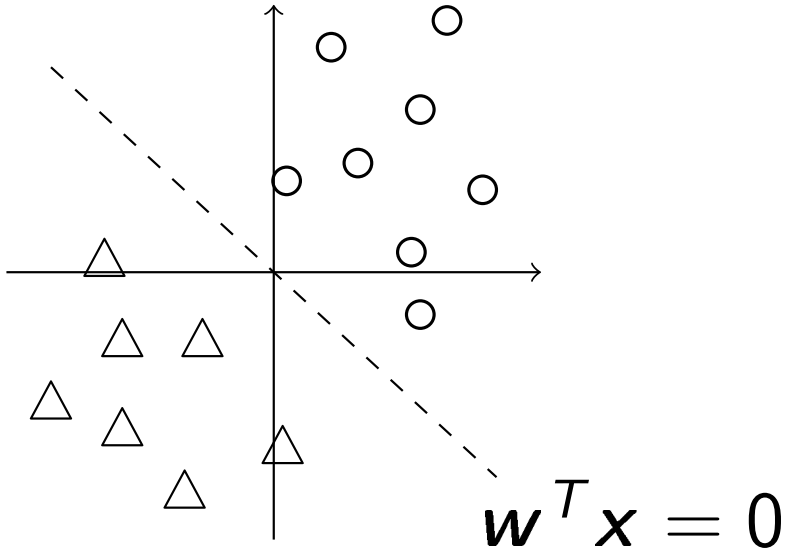
$$\text{sgn}(\boldsymbol{w}^T \boldsymbol{x})$$

- For any data, $\boldsymbol{x}$, the predicted label is

$$\begin{cases} 1 & \text{if } \boldsymbol{w}^T \boldsymbol{x} \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Minimizing Training Errors (Cont'd)

- The two-dimensional situation



- This seems to be quite restricted, but practically x is in a much higher dimensional space

Minimizing Training Errors (Cont'd)

- To characterize the training error, we need a loss function $\xi(\boldsymbol{w}; y, \boldsymbol{x})$ for each instance $(y, \boldsymbol{x})$, where

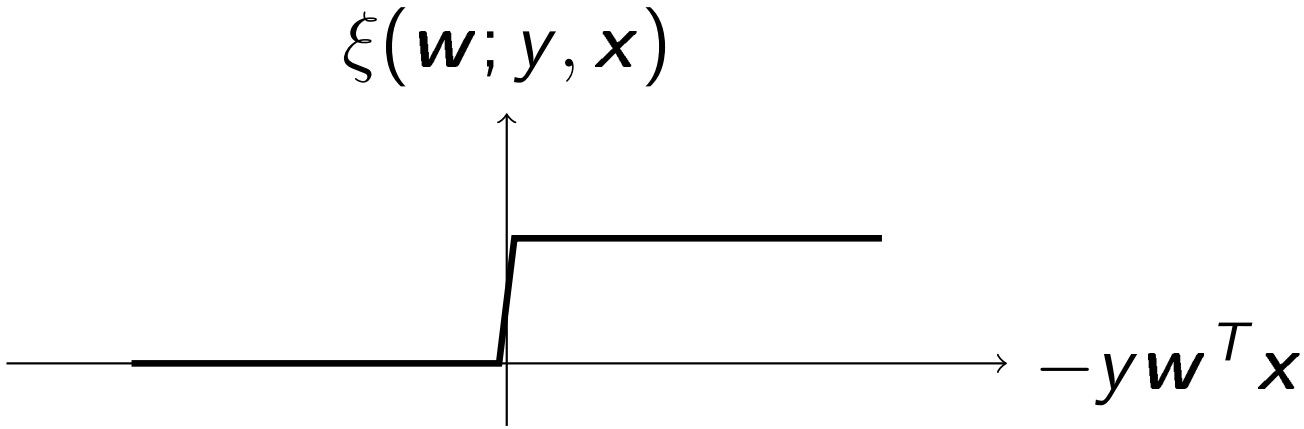
  $y = \pm 1$ is the label and $\boldsymbol{x}$ is the feature vector

- Ideally we should use 0–1 training loss:

$$\xi(\boldsymbol{w}; y, \boldsymbol{x}) = \begin{cases} 1 & \text{if } y\boldsymbol{w}^T\boldsymbol{x} < 0, \\ 0 & \text{otherwise} \end{cases}$$

Minimizing Training Errors (Cont'd)

- However, this function is discontinuous. The optimization problem becomes difficult



- We need continuous approximations

Common Loss Functions

- Hinge loss (l1 loss)

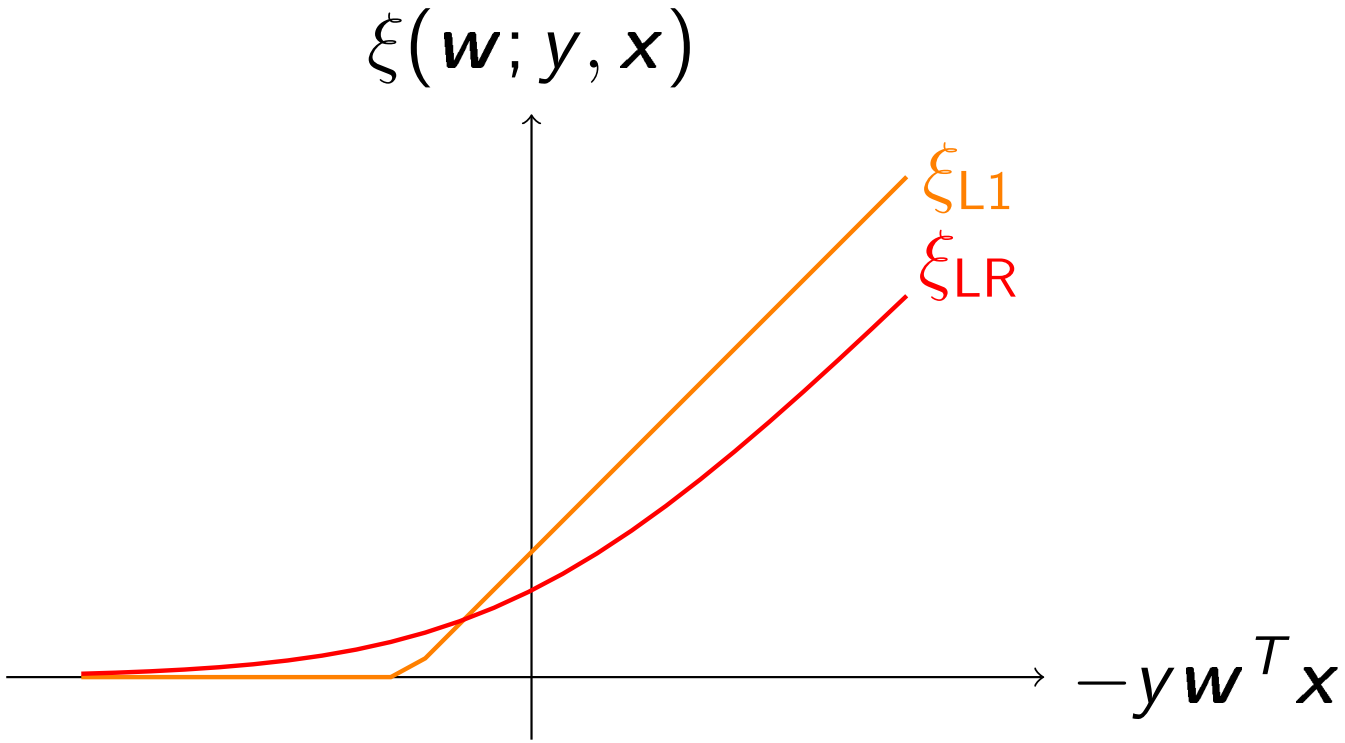
$$\xi_{\text{L1}}(\boldsymbol{w}; y, \boldsymbol{x}) \equiv \max(0, 1 - y\boldsymbol{w}^T\boldsymbol{x}) \tag{1}$$

- Logistic loss

$$\xi_{\text{LR}}(\boldsymbol{w}; y, \boldsymbol{x}) \equiv \log(1 + e^{-y\boldsymbol{w}^T\boldsymbol{x}}) \tag{2}$$

- Support vector machines (SVM): Eq. (1). Logistic regression (LR): (2)
- SVM and LR are two very fundamental classification methods

Common Loss Functions (Cont'd)



- Logistic regression is very related to SVM
- Their performance is usually similar

Common Loss Functions (Cont'd)

- However, minimizing training losses may not give a good model for future prediction
- Overfitting occurs

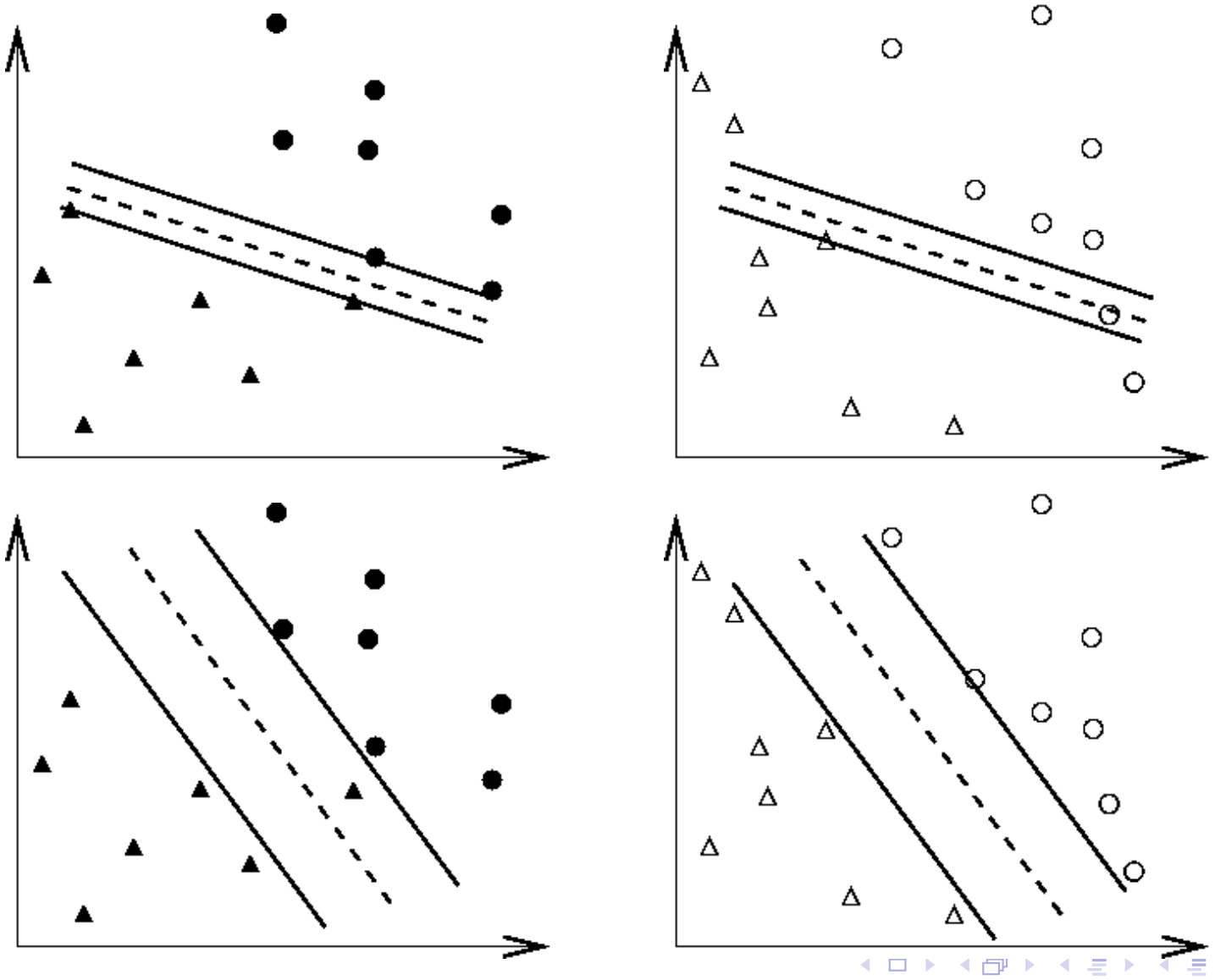
Overfitting

- See the illustration in the next slide
- For classification, you can easily achieve 100% training accuracy
- This is useless
- When training a data set, we should

  Avoid underfitting: small training error

  Avoid overfitting: small testing error

● and ▲: training; ○ and △: testing



Regularization

- To minimize the training error we manipulate the $\boldsymbol{w}$ vector so that it fits the data
- To avoid overfitting we need a way to make $\boldsymbol{w}$'s values less extreme.
- One idea is to make $\boldsymbol{w}$ values closer to zero
- We can add, for example,

$$\frac{\boldsymbol{w}^T\boldsymbol{w}}{2} \quad \text{or} \quad \|\boldsymbol{w}\|_1$$

to the function that is minimized

General Form of Linear Classification

- Training data $\{y_i, \boldsymbol{x}_i\}, \boldsymbol{x}_i \in R^n, i = 1, \ldots, l, y_i = \pm 1$
- l: # of data, n: # of features

$$\min_{\boldsymbol{w}} f(\boldsymbol{w}), \quad f(\boldsymbol{w}) \equiv \frac{\boldsymbol{w}^T \boldsymbol{w}}{2} + C \sum_{i=1}^{l} \xi(\boldsymbol{w}; y_i, \boldsymbol{x}_i)$$

- $\boldsymbol{w}^T \boldsymbol{w}/2$: regularization term
- $\xi(\boldsymbol{w}; y, \boldsymbol{x})$: loss function
- C: regularization parameter (chosen by users)

Outline

Multi-class Classification I

- Our training set includes $(\boldsymbol{y}^i, \boldsymbol{x}^i)$, $i = 1, \ldots, l$.
- $\boldsymbol{x}^i \in R^{n_1}$ is the feature vector.
- $\boldsymbol{y}^i \in R^K$ is the label vector.
- As label is now a vector, we change (label, instance) from

$$(y_i, \boldsymbol{x}_i) \text{ to } (\boldsymbol{y}^i, \boldsymbol{x}^i)$$

- K: # of classes
- If $\boldsymbol{x}^i$ is in class k, then

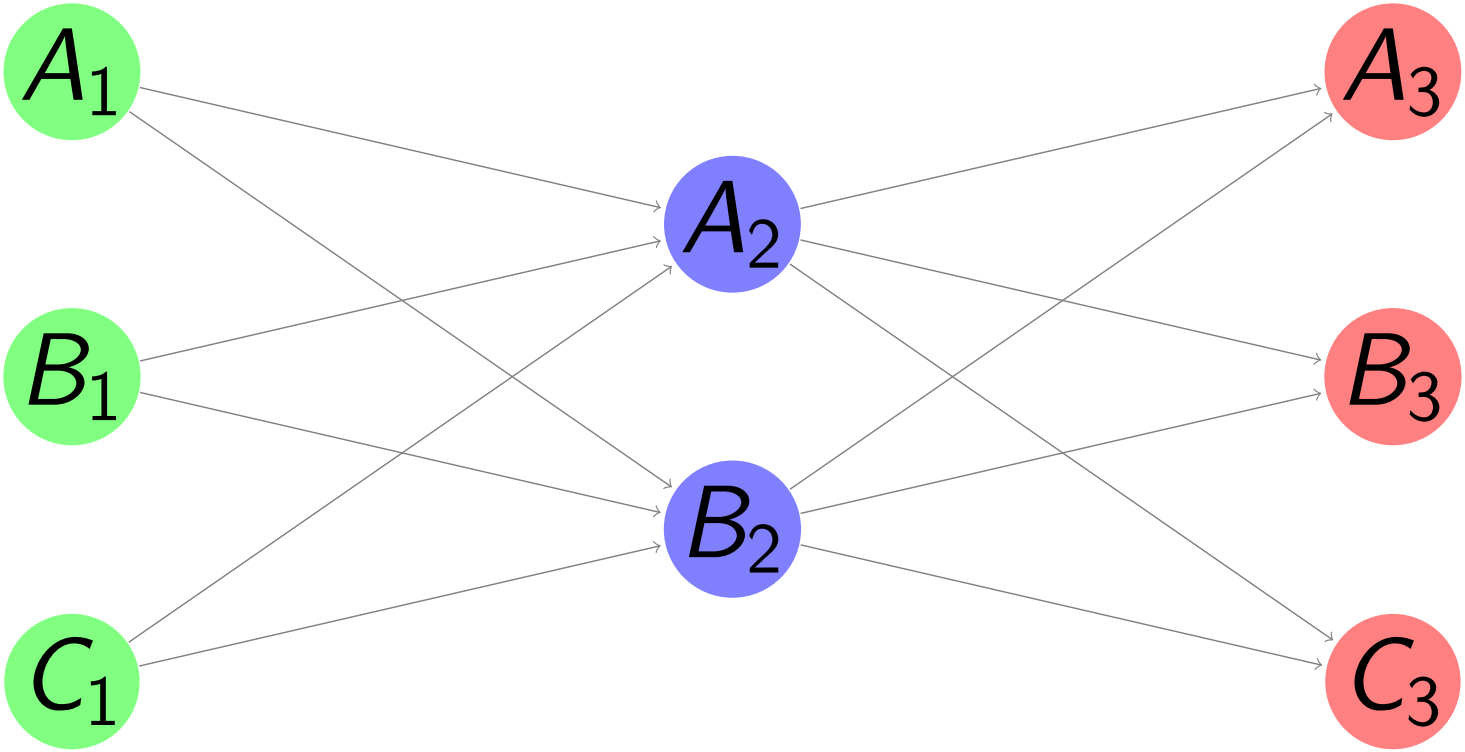
$$\boldsymbol{y}^i = [\underbrace{0, \ldots, 0}_{k-1}, 1, 0, \ldots, 0]^T \in R^K$$

Multi-class Classification II

- A neural network maps each feature vector to one of the class labels by the connection of nodes.

Fully-connected Networks

- Between two layers a weight matrix maps inputs (the previous layer) to outputs (the next layer).



Operations Between Two Layers I

- The weight matrix W^m at the mth layer is

$$W^m = \begin{bmatrix} w^m_{11} & w^m_{12} & \cdots & w^m_{1n_m} \\ w^m_{21} & w^m_{22} & \cdots & w^m_{2n_m} \\ \vdots & \vdots & \vdots & \vdots \\ w^m_{n_{m+1}1} & w^m_{n_{m+1}2} & \cdots & w^m_{n_{m+1}n_m} \end{bmatrix}_{n_{m+1} \times n_m}$$

- n_m : # input features at layer m
- n_{m+1} : # output features at layer m, or # input features at layer $m+1$
- L: number of layers

Operations Between Two Layers II

- $n_1 = \#$ of features, $n_{L+1} = \#$ of classes
- Let z^m be the input of the mth layer, $z^1 = x$ and z^{L+1} be the output
- From mth layer to $(m+1)$th layer

$$s^m = W^m z^m,$$
$$z_j^{m+1} = \sigma(s_j^m),\ j = 1, \dots, n_{m+1},$$

$\sigma(\cdot)$ is the activation function.

Operations Between Two Layers III

- Usually people do a bias term

$$\begin{bmatrix} b_1^m \\ b_2^m \\ \vdots \\ b_{n_{m+1}}^m \end{bmatrix}_{n_{m+1}\times 1},$$

so that

$$\boldsymbol{s}^m = W^m \boldsymbol{z}^m + \boldsymbol{b}^m$$

Operations Between Two Layers IV

- Activation function is usually an

$$R \to R$$

  non-linear transformation.
- There are various reasons of using an activation function. An important one is to introduce the non-linearity.

Operations Between Two Layers V

- If without an activation function, all

$$W^L \cdots W^2 W^1$$

becomes a single matrix and we end up with having only a linear mapping from the input feature to the output layer

Operations Between Two Layers VI

- We collect all variables:

$$\boldsymbol{\theta} = \begin{bmatrix} \text{vec}(W^1) \\ \boldsymbol{b}^1 \\ \vdots \\ \text{vec}(W^L) \\ \boldsymbol{b}^L \end{bmatrix} \in R^n$$

n : total # variables $= (n_1+1)n_2+\cdots+(n_L+1)n_{L+1}$

- The vec$(\cdot)$ operator stacks columns of a matrix to a vector

Optimization Problem I

- We solve the following optimization problem,

$$\min_{\theta} \quad f(\boldsymbol{\theta}), \quad \text{where}$$

$$f(\boldsymbol{\theta}) = \frac{1}{2}\boldsymbol{\theta}^T\boldsymbol{\theta} + C\sum_{i=1}^{l} \xi(\boldsymbol{z}^{L+1,i}(\boldsymbol{\theta}); \boldsymbol{y}^i, \boldsymbol{x}^i).$$

C: regularization parameter

- $\boldsymbol{z}^{L+1}(\boldsymbol{\theta}) \in R^{n_{L+1}}$: last-layer output vector of $\boldsymbol{x}$.
$\xi(\boldsymbol{z}^{L+1}; \boldsymbol{y}, \boldsymbol{x})$: loss function. Example:

$$\xi(\boldsymbol{z}^{L+1}; \boldsymbol{y}, \boldsymbol{x}) = ||\boldsymbol{z}^{L+1} - \boldsymbol{y}||^2$$

Optimization Problem II

- The formulation is same as linear classification
- However, the loss function is more complicated
- Further, it's non-convex
- Note that in the earlier discussion we consider a single instance
- In the training process we actually have for $i = 1, \ldots, l$,

$$\boldsymbol{s}^{m,i} = W^m \boldsymbol{z}^{m,i},$$
$$z_j^{m+1,i} = \sigma(s_j^{m,i}),\ j = 1, \ldots, n_{m+1},$$

This makes the training more complicated

Outline

Convolutional Neural Networks I

- There are many types of neural networks suitable for different types of problems
- We select CNN to discuss details because it is a very useful network for image classification
- Consider a K-class classification problem with training data

$$(\boldsymbol{y}^i, Z^{1,i}), \quad i = 1, \ldots, l.$$

$\boldsymbol{y}^i$: label vector $\qquad$ $Z^{1,i}$: input image

Convolutional Neural Networks II

- If $Z^{1,i}$ is in class k, then

$$\boldsymbol{y}^i = [\underbrace{0, \ldots, 0}_{k-1}, 1, 0, \ldots, 0]^T \in R^K.$$

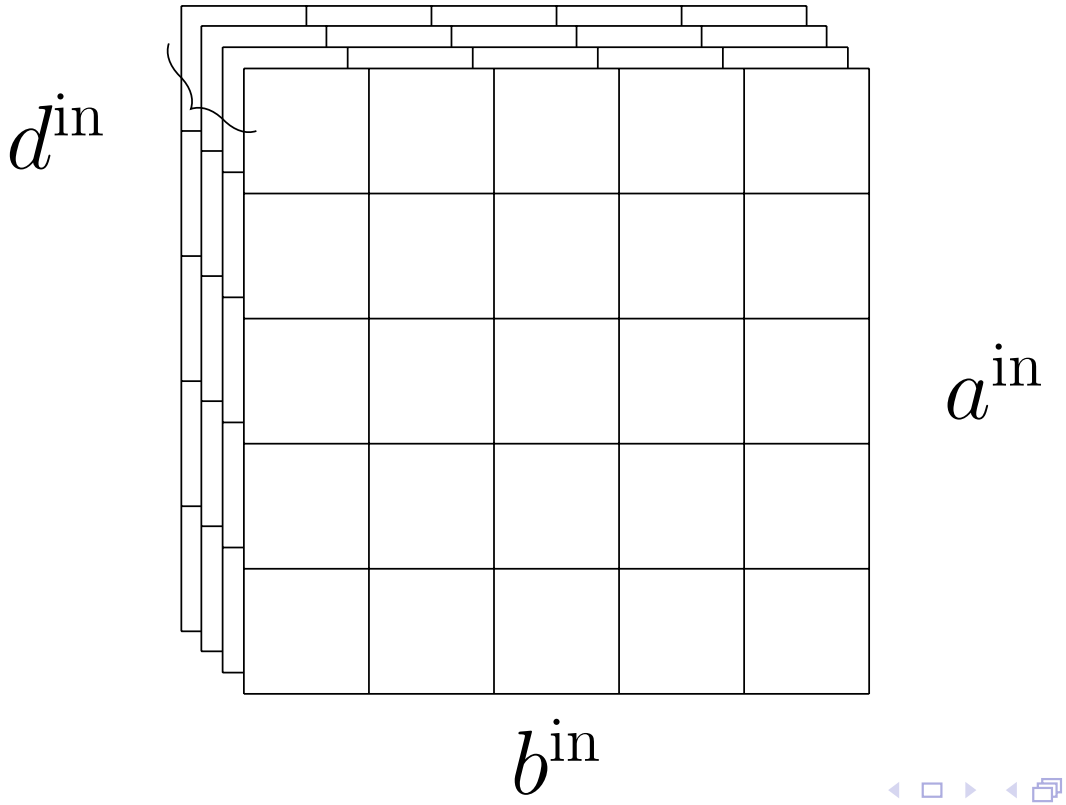
- CNN maps each image $Z^{1,i}$ to $\boldsymbol{y}^i$
- Typically, CNN consists of multiple convolutional layers followed by fully-connected layers.
- Input and output of a convolutional layer are assumed to be images.

Convolutional Layers I

- For the current layer, let the input be an image

$$Z^{\text{in}} : a^{\text{in}} \times b^{\text{in}} \times d^{\text{in}}.$$

a^{in}: height, b^{in}: width, and d^{in}: #channels.



Convolutional Layers II

The goal is to generate an output image

$$Z^{\text{out},i}$$

of d^{out} channels of $a^{\text{out}} \times b^{\text{out}}$ images.

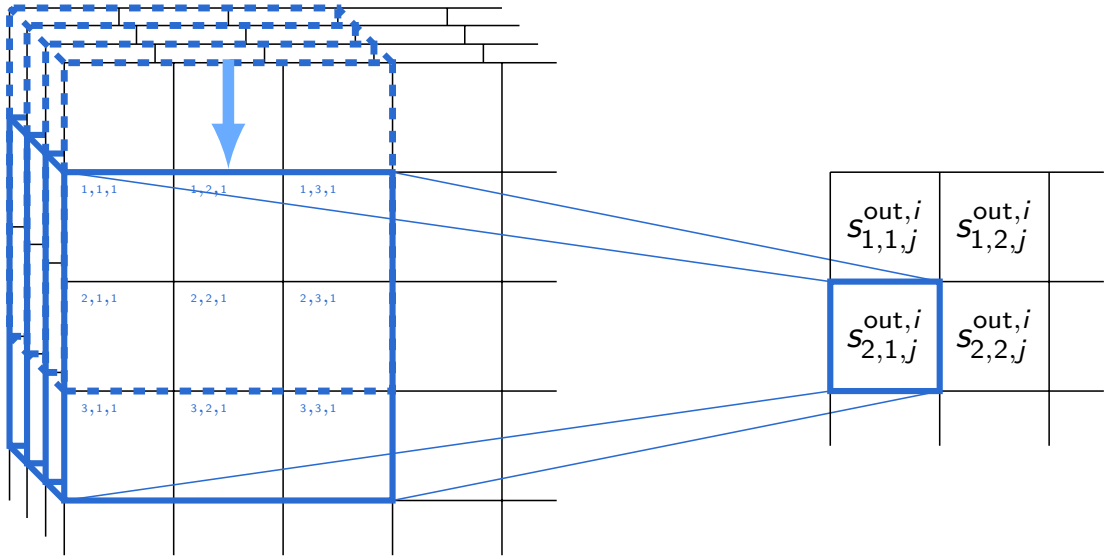
- Consider d^{out} filters.
- Filter $j \in \{1, \ldots, d^{\text{out}}\}$ has dimensions

$$h \times h \times d^{\text{in}}.$$

$$\begin{bmatrix} w^j_{1,1,1} & & w^j_{1,h,1} \\ & \ddots & \\ w^j_{h,1,1} & & w^j_{h,h,1} \end{bmatrix} \cdots \begin{bmatrix} w^j_{1,1,d^{\text{in}}} & & w^j_{1,h,d^{\text{in}}} \\ & \ddots & \\ w^j_{h,1,d^{\text{in}}} & & w^j_{h,h,d^{\text{in}}} \end{bmatrix}.$$

Convolutional Layers III

h: filter height/width (layer index omitted)



- To compute the jth channel of output, we scan the input from top-left to bottom-right to obtain the sub-images of size $h \times h \times d^{\text{in}}$

Convolutional Layers IV

- We then calculate the inner product between each sub-image and the jth filter
- The idea is that this inner product may extract local information of the sub-image
- For example, if we start from the upper left corner of the input image, the first sub-image of channel d is

$$\begin{bmatrix} z^i_{1,1,d} & \cdots & z^i_{1,h,d} \\ & \ddots & \\ z^i_{h,1,d} & \cdots & z^i_{h,h,d} \end{bmatrix}.$$

Convolutional Layers V

We then calculate

$$\sum_{d=1}^{d^{\text{in}}} \left\langle \begin{bmatrix} z^i_{1,1,d} & \cdots & z^i_{1,h,d} \\ & \ddots & \\ z^i_{h,1,d} & \cdots & z^i_{h,h,d} \end{bmatrix}, \begin{bmatrix} w^j_{1,1,d} & \cdots & w^j_{1,h,d} \\ & \ddots & \\ w^j_{h,1,d} & \cdots & w^j_{h,h,d} \end{bmatrix} \right\rangle + b_j, \tag{3}$$

where $\langle \cdot, \cdot \rangle$ means the sum of component-wise products between two matrices.

- This value becomes the $(1, 1)$ position of the channel j of the output image.

Convolutional Layers VI

- Next, we use other sub-images to produce values in other positions of the output image.
- Let the stride s be the number of pixels vertically or horizontally to get sub-images.
- For the $(2, 1)$ position of the output image, we move down s pixels vertically to obtain the following sub-image:

$$\begin{bmatrix} z^i_{1+s,1,d} & \cdots & z^i_{1+s,h,d} \\ & \ddots & \\ z^i_{h+s,1,d} & \cdots & z^i_{h+s,h,d} \end{bmatrix}.$$

Convolutional Layers VII

- The $(2, 1)$ position of the channel j of the output image is

$$\sum_{d=1}^{d^{\text{in}}} \left\langle \begin{bmatrix} z^i_{1+s,1,d} & \cdots & z^i_{1+s,h,d} \\ & \ddots & \\ z^i_{h+s,1,d} & \cdots & z^i_{h+s,h,d} \end{bmatrix}, \begin{bmatrix} w^j_{1,1,d} & \cdots & w^j_{1,h,d} \\ & \ddots & \\ w^j_{h,1,d} & \cdots & w^j_{h,h,d} \end{bmatrix} \right\rangle + b_j. \tag{4}$$

Convolutional Layers VIII

- The output image size a^{out} and b^{out} are respectively numbers that vertically and horizontally we can move the filter

$$a^{\text{out}} = \lfloor \frac{a^{\text{in}} - h}{s} \rfloor + 1, \quad b^{\text{out}} = \lfloor \frac{b^{\text{in}} - h}{s} \rfloor + 1 \quad (5)$$

- Rationale of (5): vertically last row of each sub-image is

$$h, h + s, \ldots, h + \Delta s \leq a^{\text{in}}$$

Convolutional Layers IX

Thus

$$\Delta = \lfloor \frac{a^{\text{in}} - h}{s} \rfloor$$

Matrix Representation I

- We may use a matrix form to represent the convolutional operation. This helps to easily derive the Gradient calculation
- Let's collect images of all channels as the input

$$Z^{\text{in},i}$$

$$= \begin{bmatrix} z^i_{1,1,1} & z^i_{2,1,1} & \cdots & z^i_{a^{\text{in}},b^{\text{in}},1} \\ \vdots & \vdots & \ddots & \vdots \\ z^i_{1,1,d^{\text{in}}} & z^i_{2,1,d^{\text{in}}} & \cdots & z^i_{a^{\text{in}},b^{\text{in}},d^{\text{in}}} \end{bmatrix}$$

$$\in \mathbf{R}^{d^{\text{in}} \times a^{\text{in}} b^{\text{in}}}.$$

Matrix Representation II

- Let all filters

$$W = \begin{bmatrix} w^1_{1,1,1} & w^1_{2,1,1} & \cdots & w^1_{h,h,d^{\text{in}}} \\ \vdots & \vdots & \ddots & \vdots \\ w^{d^{\text{out}}}_{1,1,1} & w^{d^{\text{out}}}_{2,1,1} & \cdots & w^{d^{\text{out}}}_{h,h,d^{\text{in}}} \end{bmatrix} \in \mathbf{R}^{d^{\text{out}} \times hhd^{\text{in}}}$$

be variables (parameters) of the current layer

Matrix Representation III

- Usually a bias term is considered

$$\boldsymbol{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_{d^{\text{out}}} \end{bmatrix} \in R^{d^{\text{out}} \times 1}$$

- Operations at a layer

$$\begin{aligned} S^{\text{out},i} &= W\phi(Z^{\text{in},i}) + \boldsymbol{b}\mathbb{1}^T_{a^{\text{out}}b^{\text{out}}} \\ &\in R^{d^{\text{out}} \times a^{\text{out}}b^{\text{out}}}, \end{aligned} \tag{6}$$

Matrix Representation IV

where

$$\mathbb{1}_{a^{\text{out}}b^{\text{out}}} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \in R^{a^{\text{out}}b^{\text{out}} \times 1}.$$

- $\phi(Z^{\text{in},i})$ collects all sub-images in $Z^{\text{in},i}$ into a matrix.

Matrix Representation V

Specifically,

$$\phi(Z^{\text{in},i}) =$$

$$\begin{bmatrix} z^i_{1,1,1} & z^i_{1+s,1,1} & & z^i_{1+(a^{\text{out}}-1)s,1+(b^{\text{out}}-1)s,1} \\ z^i_{2,1,1} & z^i_{2+s,1,1} & & z^i_{2+(a^{\text{out}}-1)s,1+(b^{\text{out}}-1)s,1} \\ \vdots & \vdots & \cdots & \vdots \\ z^i_{h,h,1} & z^i_{h+s,h,1} & & z^i_{h+(a^{\text{out}}-1)s,h+(b^{\text{out}}-1)s,1} \\ \vdots & \vdots & & \vdots \\ z^i_{h,h,d^{\text{in}}} & z^i_{h+s,h,d^{\text{in}}} & & z^i_{h+(a^{\text{out}}-1)s,h+(b^{\text{out}}-1)s,d^{\text{in}}} \end{bmatrix}$$

$$\in \mathbf{R}^{hhd^{\text{in}} \times a^{\text{out}} b^{\text{out}}}$$

Activation Function I

- Next, an activation function scales each element of $S^{\text{out},i}$ to obtain the output matrix $Z^{\text{out},i}$.

$$Z^{\text{out},i} = \sigma(S^{\text{out},i}) \in R^{d^{\text{out}} \times a^{\text{out}} b^{\text{out}}}. \tag{7}$$

- For CNN, commonly the following RELU activation function

$$\sigma(x) = \max(x, 0) \tag{8}$$

  is used
- Later we need that $\sigma(x)$ is differentiable, but the RELU function is not.

Activation Function II

- Past works such as Krizhevsky et al. (2012) assume

$$\sigma'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

The Function $\phi(Z^{\text{in},i})$ I

- In the matrix-matrix product

$$W\phi(Z^{\text{in},i}),$$

each element is the inner product between a filter and a sub-image

- Clearly ϕ is a linear mapping, so there exists a 0/1 matrix P_ϕ such that

$$\phi(Z^{\text{in},i}) \equiv \text{mat}\left(P_\phi \text{vec}(Z^{\text{in},i})\right)_{hhd^{\text{in}} \times a^{\text{out}} b^{\text{out}}}, \ \forall i, \quad (9)$$

The Function $\phi(Z^{\text{in},i})$ II

- $\text{vec}(M)$: all M's columns concatenated to a vector $\boldsymbol{v}$

$$\text{vec}(M) = \begin{bmatrix} M_{:,1} \\ \vdots \\ M_{:,b} \end{bmatrix} \in R^{ab\times 1}, \text{ where } M \in R^{a\times b}$$

- $\text{mat}(\boldsymbol{v})$ is the inverse of $\text{vec}(M)$

$$\text{mat}(\boldsymbol{v})_{a\times b} = \begin{bmatrix} v_1 & \cdots & v_{(b-1)a+1} \\ \vdots & & \vdots \\ v_a & & v_{ba} \end{bmatrix} \in R^{a\times b}, \quad (10)$$

The Function $\phi(Z^{\text{in},i})$ III

where

$$\boldsymbol{v} \in R^{ab \times 1}.$$

- P_ϕ is a huge matrix:

$$P_\phi \in R^{hhd^{\text{in}} a^{\text{out}} b^{\text{out}} \times d^{\text{in}} a^{\text{in}} b^{\text{in}}}$$

and

$$\phi : R^{d^{\text{in}} \times a^{\text{in}} b^{\text{in}}} \rightarrow R^{hhd^{\text{in}} \times a^{\text{out}} b^{\text{out}}}$$

- The representation of using P_ϕ makes some subsequent derivations easier

Optimization Problem I

- We collect all weights to a vector variable $\boldsymbol{\theta}$.

$$\boldsymbol{\theta} = \begin{bmatrix} \text{vec}(W^1) \\ \boldsymbol{b}^1 \\ \vdots \\ \text{vec}(W^L) \\ \boldsymbol{b}^L \end{bmatrix} \in R^n, \quad n : \text{total } \# \text{ variables}$$

- The output of the last layer L is a vector $\boldsymbol{z}^{L+1,i}(\boldsymbol{\theta})$.
- Consider any loss function such as the squared loss

$$\xi_i(\boldsymbol{\theta}) = ||\boldsymbol{z}^{L+1,i}(\boldsymbol{\theta}) - \boldsymbol{y}^i||^2.$$

Optimization Problem II

- The optimization problem is

$$\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}),$$

where

$$f(\boldsymbol{\theta}) = \frac{1}{2C}\boldsymbol{\theta}^T\boldsymbol{\theta} + \frac{1}{l}\sum_{i=1}^{l} \xi(\boldsymbol{z}^{L+1,i}(\boldsymbol{\theta}); \boldsymbol{y}^i, Z^{1,i})$$

C: regularization parameter.

- The formulation is almost the same as that for fully connected networks

Optimization Problem III

- Note that we divide the sum of training losses by the number of training data
- Thus the second term becomes the average training loss
- This form helps our explanation of stochastic gradient methods later

Other Operations in CNN I

- CNN involves additional operations in practice
  - padding
  - pooling
- We omit details

Outline

NN Optimization Problem I

- Recall that the NN optimization problem is

$$\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$$

where

$$f(\boldsymbol{\theta}) = \frac{1}{2C}\boldsymbol{\theta}^T\boldsymbol{\theta} + \frac{1}{l}\sum_{i=1}^{l} \xi(\boldsymbol{z}^{L+1,i}(\boldsymbol{\theta}); \boldsymbol{y}^i, Z^{1,i})$$

- Let's simplify the loss part

$$f(\boldsymbol{\theta}) = \frac{1}{2C}\boldsymbol{\theta}^T\boldsymbol{\theta} + \frac{1}{l}\sum_{i=1}^{l} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, Z^{1,i})$$

- The issue now is how to do the minimization

Gradient Descent I

- This is one of the most used optimization method
- First-order approximation

$$f(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) \approx f(\boldsymbol{\theta}) + \nabla f(\boldsymbol{\theta})^T \Delta\boldsymbol{\theta},$$

where

$$\nabla f(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial f(\boldsymbol{\theta})}{\partial \theta_1} \\ \vdots \\ \frac{\partial f(\boldsymbol{\theta})}{\partial \theta_n} \end{bmatrix}$$

is the gradient of $f(\boldsymbol{\theta})$

Gradient Descent II

- Solve

$$\begin{aligned} \min_{\Delta\boldsymbol{\theta}} \quad & \nabla f(\boldsymbol{\theta})^T \Delta\boldsymbol{\theta} \\ \text{subject to} \quad & \|\Delta\boldsymbol{\theta}\| = 1 \end{aligned} \tag{11}$$

  to find a direction $\Delta\boldsymbol{\theta}$
- The constraint $\|\Delta\boldsymbol{\theta}\| = 1$ is needed. Otherwise, the above sub-problem goes to $-\infty$
- The solution of (11) is

$$\Delta\boldsymbol{\theta} = -\frac{\nabla f(\boldsymbol{\theta})}{\|\nabla f(\boldsymbol{\theta})\|} \tag{12}$$

Gradient Descent III

- This is called the steepest descent direction
- However, because we only consider an approximation

$$f(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) \approx f(\boldsymbol{\theta}) + \nabla f(\boldsymbol{\theta})^T \Delta\boldsymbol{\theta}$$

  we may not have the strict decrease of the function value
- That is,

$$f(\boldsymbol{\theta}) < f(\boldsymbol{\theta} + \Delta\boldsymbol{\theta})$$

  may occur

Gradient Descent IV

- But in general we need the descent property to get the convergence
- We have

$$f(\boldsymbol{\theta} + \alpha\Delta\boldsymbol{\theta}) = f(\boldsymbol{\theta}) + \alpha\nabla f(\boldsymbol{\theta})^T\Delta\boldsymbol{\theta} + \frac{1}{2}\alpha^2\Delta\boldsymbol{\theta}^T\nabla^2 f(\boldsymbol{\theta})\Delta\boldsymbol{\theta} + \cdots,$$

where

$$\nabla^2 f(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial^2 f}{\partial\theta_1\partial\theta_1} & \cdots & \frac{\partial^2 f}{\partial\theta_1\partial\theta_n} \\ \vdots & & \vdots \\ \frac{\partial^2 f}{\partial\theta_n\partial\theta_1} & \cdots & \frac{\partial^2 f}{\partial\theta_n\partial\theta_n} \end{bmatrix}$$

Gradient Descent V

is the Hessian of $f(\boldsymbol{\theta})$

- If

$$\nabla f(\boldsymbol{\theta})^T \Delta\boldsymbol{\theta} < 0,$$

then a small enough α can ensure

$$f(\boldsymbol{\theta} + \alpha\Delta\boldsymbol{\theta}) < f(\boldsymbol{\theta})$$

- Thus in optimization for any direction (not necessarily the steepest descent direction), it is called a descent direction if

$$\nabla f(\boldsymbol{\theta})^T \Delta\boldsymbol{\theta} < 0$$

Gradient Descent VI

- The direction chosen in (12) is a descent direction:

$$-\nabla f(\boldsymbol{\theta})^T \frac{\nabla f(\boldsymbol{\theta})}{\|\nabla f(\boldsymbol{\theta})\|} < 0.$$

Line Search I

- We have seen that we need a step size α such that

$$f(\boldsymbol{\theta} + \alpha\Delta\boldsymbol{\theta}) < f(\boldsymbol{\theta})$$

- In optimization this is called a line search procedure
- Exact line search

$$\min_{\alpha} f(\boldsymbol{\theta} + \alpha\Delta\boldsymbol{\theta})$$

  This is a one-dimensional optimization problem
- In practice, people use backtracking line search

Line Search II

- We check

$$\alpha = 1, \beta, \beta^2, \dots$$

with $\beta \in (0, 1)$ until

$$f(\boldsymbol{\theta} + \alpha\Delta\boldsymbol{\theta}) < f(\boldsymbol{\theta}) + \nu\nabla f(\boldsymbol{\theta})^T(\alpha\Delta\boldsymbol{\theta})$$

- Here

$$\nu \in (0, \frac{1}{2})$$

- The convergence is well established.

Line Search III

- That is, we can reach a limit point $\bar{\boldsymbol{\theta}}$ with

$$\nabla f(\bar{\boldsymbol{\theta}}) = \mathbf{0}$$

- We then get a stationary point of a non-convex problem for deep learning

Estimation of the Gradient I

- Recall the function is

$$f(\boldsymbol{\theta}) = \frac{1}{2C}\boldsymbol{\theta}^T\boldsymbol{\theta} + \frac{1}{l}\sum_{i=1}^{l} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, Z^{1,i})$$

- The gradient is

$$\frac{\boldsymbol{\theta}}{C} + \frac{1}{l}\nabla_{\boldsymbol{\theta}} \sum_{i=1}^{l} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, Z^{1,i})$$

$$= \frac{\boldsymbol{\theta}}{C} + \frac{1}{l}\sum_{i=1}^{l} \nabla_{\boldsymbol{\theta}} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, Z^{1,i})$$

Estimation of the Gradient II

- Going over all data is time consuming
- If data are from the same distribution

$$E(\nabla_\theta \xi(\boldsymbol{\theta}; \boldsymbol{y}, Z^1)) = \frac{1}{l} \sum_{i=1}^{l} \nabla_\theta \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, Z^{1,i})$$

then we may just use a subset S (often called a batch)

$$\frac{\boldsymbol{\theta}}{C} + \frac{1}{|S|} \nabla_\theta \sum_{i:i \in S} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, Z^{1,i})$$

Stochastic Gradient Algorithm I

1: Given an initial learning rate η.
2: **while do**
3: Choose $S \subset \{1, \ldots, l\}$.
4: Calculate

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\left(\frac{\boldsymbol{\theta}}{C} + \frac{1}{|S|}\nabla_{\boldsymbol{\theta}} \sum_{i:i\in S} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, Z^{1,i})\right)$$

5: May adjust the learning rate η
6: **end while**

- It's known that deciding a suitable learning rate is difficult

Stochastic Gradient Algorithm II

- Too small learning rate: very slow convergence
- Too large learning rate: the procedure may diverge

Stochastic Gradient "Descent" I

- In comparison with gradient descent you see that we don't do line search
- Indeed we cannot. Without the full gradient, the sufficient decrease condition may never hold.

$$f(\boldsymbol{\theta} + \alpha\Delta\boldsymbol{\theta}) < f(\boldsymbol{\theta}) + \nu\nabla f(\boldsymbol{\theta})^T(\alpha\Delta\boldsymbol{\theta})$$

- Therefore, we don't have a "descent" algorithm here
- It's possible that

$$f(\boldsymbol{\theta}^{\text{next}}) > f(\boldsymbol{\theta})$$

- Though people frequently use "SGD," it's unclear if "D" is suitable in the name of this method

Momentum I

- Because we use a subset of data to get an approximate gradient, the resulting directions may be noisy
- We consider a moving average of sub-gradients
- A new vector $\boldsymbol{v}$ and a parameter $\alpha \in [0, 1)$ are introduced

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \eta\left(\frac{\boldsymbol{\theta}}{C} + \frac{1}{|S|}\nabla_{\boldsymbol{\theta}} \sum_{i:i\in S} \xi(\boldsymbol{\theta}; \boldsymbol{y}_i, \boldsymbol{x}_i)\right) \quad (13)$$

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

Momentum II

- However, the rule in (13) may be biased toward the initial value
- Thus we need bias correction, which will be discussed later
- So far the learning rate η is the same for every component of the sub-gradient

AdaGrad I

- Scaling learning rates inversely proportional to the square root of sum of past gradient squares (Duchi et al., 2011)
- Update rule:

$$\begin{aligned} \boldsymbol{g} &\leftarrow \frac{\boldsymbol{\theta}}{C} + \frac{1}{|S|}\nabla_{\boldsymbol{\theta}} \sum_{i:i\in S} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, Z^{1,i}) \\ \boldsymbol{r} &\leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g} \\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \frac{\epsilon}{\sqrt{\boldsymbol{r}} + \delta} \odot \boldsymbol{g} \end{aligned}$$

- $\boldsymbol{r}$: sum of past gradient squares

AdaGrad II

ϵ and δ are given constants

- $\odot$: Hadamard product (element-wise product of two vectors/matrices)
- A large $\boldsymbol{g}$ component

  $\Rightarrow$ a larger $\boldsymbol{r}$ component

  $\Rightarrow$ fast decrease of the learning rate
- Conceptual explanation from Duchi et al. (2011):
  - frequently occurring features $\Rightarrow$ low learning rates
  - infrequent features $\Rightarrow$ high learning rates

AdaGrad III

"the intuition is that each time an infrequent feature is seen, the learner should take notice."

- But how is this explanation related to $\boldsymbol{g}$ components?
- Let's consider linear classification. Recall our optimization problem is

$$\frac{\boldsymbol{w}^T\boldsymbol{w}}{2} + C\sum_{i=1}^{l}\xi(\boldsymbol{w}; y_i, \boldsymbol{x}_i)$$

AdaGrad IV

- For methods such as SVM or logistic regression, the loss function can be written as a function of $\boldsymbol{w}^T\boldsymbol{x}$

$$\xi(\boldsymbol{w}; y, \boldsymbol{x}) = \hat{\epsilon}(\boldsymbol{w}^T\boldsymbol{x})$$

Then the gradient is

$$\boldsymbol{w} + C\sum_{i=1}^{l} \hat{\epsilon}'(\boldsymbol{w}^T\boldsymbol{x}_i)\boldsymbol{x}_i$$

- Thus the gradient is related to the density of features

AdaGrad V

- The above analysis is for linear classification
- But now we have a non-convex neural network!
- Empirically, people find that the sum of squared gradient since the beginning causes too fast decrease of the learning rate

RMSProp I

- The original reference seems to be the lecture slides at `https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`
- Idea: they think AdaGrad's learning rate may be too small before reaching a locally convex region
- That is, OK to sum all past gradient squares in convex, but not non-convex
- Thus they do "exponentially weighted moving average"

RMSProp II

- Update rule

$$\begin{aligned} \boldsymbol{r} &\leftarrow \rho \boldsymbol{r} + (1-\rho)\boldsymbol{g} \odot \boldsymbol{g} \\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g} \end{aligned}$$

- AdaGrad:

$$\begin{aligned} \boldsymbol{r} &\leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g} \\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \frac{\epsilon}{\sqrt{\boldsymbol{r}} + \delta} \odot \boldsymbol{g} \end{aligned}$$

RMSProp III

- Somehow the setting is a bit heuristic and the reason behind the change (from AdaGrad to RMSProp) is not really that strong

Adam (Adaptive Moments) I

- The update rule (Kingma and Ba, 2015)

$$
\begin{aligned}
\boldsymbol{g} &\leftarrow \frac{\boldsymbol{\theta}}{C} + \frac{1}{|S|}\nabla_{\boldsymbol{\theta}} \sum_{i:i\in S} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, Z^{1,i}) \\
\boldsymbol{s} &\leftarrow \rho_1 \boldsymbol{s} + (1-\rho_1)\boldsymbol{g} \\
\boldsymbol{r} &\leftarrow \rho_2 \boldsymbol{r} + (1-\rho_2)\boldsymbol{g} \odot \boldsymbol{g} \\
\hat{\boldsymbol{s}} &\leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t} \\
\hat{\boldsymbol{r}} &\leftarrow \frac{\boldsymbol{r}}{1-\rho_2^t} \\
\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \frac{\epsilon}{\sqrt{\hat{\boldsymbol{r}}} + \delta} \odot \hat{\boldsymbol{s}}
\end{aligned}
$$

Adam (Adaptive Moments) II

- t is the current iteration index
- Roughly speaking, Adam is the combination of
  - Momentum
  - RMSprop
- From Goodfellow et al. (2016),

$$\frac{\epsilon}{\sqrt{\hat{\boldsymbol{r}}} + \delta} \odot \hat{\boldsymbol{s}}$$

(i.e., the use of momentum combined with rescaling) "does not have a clear theoretical motivation"

Adam (Adaptive Moments) III

- How about Adam's practical performance?
- From Goodfellow et al. (2016), "generally regarded as being fairly robust to the choice of hyperparmeters, though the learning rate may need to be changed from the default"
- However, some[1] said that: "The original paper ... showing huge performance gains in terms of speed of training. However, after a while people started noticing, that in some cases Adam actually finds worse solution than stochastic gradient"

Adam (Adaptive Moments) IV

- One example of showing the above is Wilson et al. (2017)

[1] https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c

Bias Correction in Adam I

- The two steps in Adam

$$\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t}$$

$$\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1-\rho_2^t}$$

  are called "bias correction"
- Why do we need this "bias correction"?
- Note that $\boldsymbol{s}$ is the direction used to update $\boldsymbol{\theta}$.

Bias Correction in Adam II

- We hope that its expectation is similar to the expected gradient

$$E[\boldsymbol{s}_t] = E[\boldsymbol{g}_t]$$

and

$$E[\boldsymbol{r}_t] = E[\boldsymbol{g}_t \odot \boldsymbol{g}_t],$$

where t is the iteration index

- The problem is that due to the moving average, the vector is biased toward the initial value
- Note that our initial $\boldsymbol{s}$ is $\mathbf{0}$

Bias Correction in Adam III

- For $\boldsymbol{s}_t$, we have

$$\begin{aligned}
\boldsymbol{s}_t &= \rho_1 \boldsymbol{s}_{t-1} + (1-\rho_1)\boldsymbol{g}_t \\
&= \rho_1(\rho_1 \boldsymbol{s}_{t-2} + (1-\rho_1)\boldsymbol{g}_{t-1}) + (1-\rho_1)\boldsymbol{g}_t \\
&= (1-\rho_1)\sum_{i=1}^{t} \rho_1^{t-i}\boldsymbol{g}_i
\end{aligned}$$

Bias Correction in Adam IV

- Then

$$\begin{aligned} E[\boldsymbol{s}_t] &= E[(1-\rho_1)\sum_{i=1}^{t}\rho_1^{t-i}\boldsymbol{g}_i] \\ &= E[\boldsymbol{g}_t](1-\rho_1)\sum_{i=1}^{t}\rho_1^{t-i} \end{aligned}$$

- Note that we assume

$$E[\boldsymbol{g}_i], \forall i \geq 1$$

are the same

Bias Correction in Adam V

- Next,

$$\begin{aligned}&(1-\rho_1)\sum_{i=1}^{t}\rho_1^{t-i}\\ =&(1-\rho_1)(1+\cdots+\rho_1^{t-1})\\ =&1-\rho_1^t\end{aligned}$$

- Thus

$$E[\boldsymbol{s}_t] = E[\boldsymbol{g}_t](1-\rho_1^t)$$

and they do

$$\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t}$$

Bias Correction in Adam VI

- The above derivation on bias correction partially follows from `https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimiz`
- The situation for $E[\boldsymbol{g}_t \odot \boldsymbol{g}_t]$ is similar

The Importance of Bias Correction I

- An interesting story is that BERT (Devlin et al., 2019), an important NLP technique using Adam, forgot to do bias correction
- This seems to cause lengthy iterations
- See Zhang et al. (2021) for discussing this issue

Weight Decay I

- Recall in our earlier description, the simple stochastic gradient update is

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\left(\frac{\boldsymbol{\theta}}{C} + \frac{1}{|S|}\nabla_{\boldsymbol{\theta}} \sum_{i:i\in S} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, Z^{1,i})\right)$$

- In this rule,

$$\frac{\boldsymbol{\theta}}{C}$$

comes from the regularization term $\boldsymbol{\theta}^T\boldsymbol{\theta}/(2C)$ in $f(\boldsymbol{\theta})$

Weight Decay II

- The use of regularization follows from standard machine learning settings
- However, in the area of neural networks, this term may come from a setting called weight decay (Hanson and Pratt, 1988)

$$\boldsymbol{\theta} \leftarrow (1-\lambda)\boldsymbol{\theta} - \eta\left(\frac{1}{|S|}\nabla_{\boldsymbol{\theta}} \sum_{i:i\in S} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, Z^{1,i})\right)$$

  where λ is the rate of weight decay
- In fact, Hanson and Pratt (1988) did not give good reasons for decaying the weight of $\boldsymbol{\theta}$

Weight Decay III

- Clearly, if

$$\lambda = \frac{\eta}{C}$$

  then weight decay is the same as regularization
- However, as pointed out in Loshchilov and Hutter (2019), the equivalence does not hold if adaptive learning rate is used

Weight Decay IV

- For example, in AdaGrad, the update rule is

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\epsilon}{\sqrt{\boldsymbol{r}} + \delta} \odot \left(\frac{1}{|S|} \nabla_{\boldsymbol{\theta}} \sum_{i:i \in S} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, Z^{1,i})\right)$$
$$- \frac{\epsilon}{\sqrt{\boldsymbol{r}} + \delta} \odot \frac{\boldsymbol{\theta}}{C}$$

  so the regularization term is scaled in a component-wise way

- Loshchilov and Hutter (2019) advocate to decouple the weight decay step

Weight Decay V

- For example, for the momentum algorithm

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \eta\left(\frac{\boldsymbol{\theta}}{C} + \frac{1}{|S|}\nabla_{\boldsymbol{\theta}} \sum_{i:i\in S} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, Z^{1,i})\right)$$

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

they prefer the following equivalent form

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \eta\left(\frac{1}{|S|}\nabla_{\boldsymbol{\theta}} \sum_{i:i\in S} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, Z^{1,i})\right)$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v} - \eta\frac{\boldsymbol{\theta}}{C}$$

Weight Decay VI

- Based on this, Loshchilov and Hutter (2019) proposed AdamW

AdamW I

$$
\begin{aligned}
\boldsymbol{g} &\leftarrow \frac{1}{|S|}\nabla_\theta \sum_{i:i\in S} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, Z^{1,i}) \\
\boldsymbol{s} &\leftarrow \rho_1 \boldsymbol{s} + (1-\rho_1)\boldsymbol{g} \\
\boldsymbol{r} &\leftarrow \rho_2 \boldsymbol{r} + (1-\rho_2)\boldsymbol{g} \odot \boldsymbol{g} \\
\hat{\boldsymbol{s}} &\leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t} \\
\hat{\boldsymbol{r}} &\leftarrow \frac{\boldsymbol{r}}{1-\rho_2^t} \\
\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \frac{\epsilon}{\sqrt{\hat{\boldsymbol{r}}}+\delta} \odot \hat{\boldsymbol{s}} - \epsilon \frac{\boldsymbol{\theta}}{C}
\end{aligned}
$$

AdamW II

- This is not equivalent to Adam because in Adam, $\boldsymbol{\theta}/C$ has been used in calculating $\boldsymbol{g}$ and then scaled after
- Why is the decoupled setting better? Some discussions are in Section 3 of Loshchilov and Hutter (2019)

Choosing Stochastic Gradient Algorithms

- From Goodfellow et al. (2016), "there is currently no consensus"
- Further, "the choice ... seemed to depend on the user's familiarity with the algorithm"

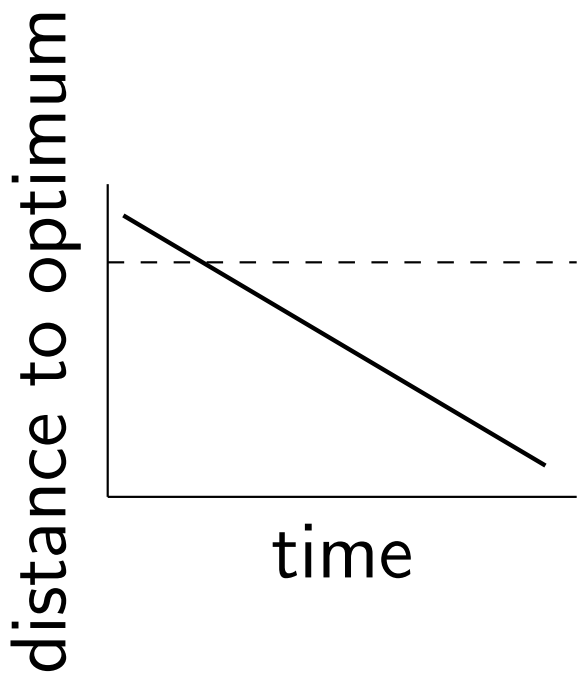
Why Stochastic Gradient Widely Used? I

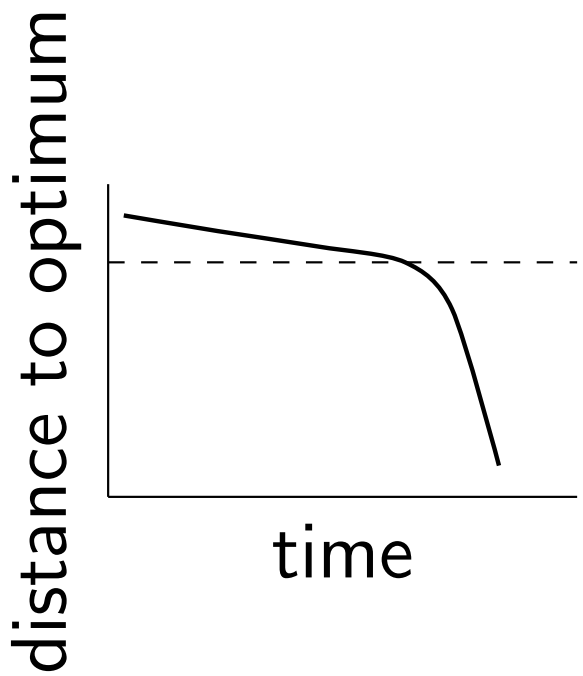
- In machine learning fast final convergence may not be important
  - An optimal solution $\boldsymbol{\theta}^*$ may not lead to the best model
  - Further, we don't need a point close to $\boldsymbol{\theta}^*$. Suppose the decision value at $\boldsymbol{\theta}^*$ is $0.3 > 0$ and a positive label is predicted. Then an approximate decision value of 0.29 makes no difference

    A not-so-accurate $\boldsymbol{\theta}$ may be good enough

Why Stochastic Gradient Widely Used? II

Thus a method with slow final convergence may be efficient enough





Slow final convergence Fast final convergence

Why Stochastic Gradient Widely Used? III

This illustration is modified from Tsai et al. (2014)

- The special property of data classification is essential

$$E(\nabla_\theta \xi(\boldsymbol{\theta}; \boldsymbol{y}, \boldsymbol{x}) = \frac{1}{l} \nabla_\theta \sum_{i=1}^{l} \xi(\boldsymbol{\theta}; \boldsymbol{y}_i, \boldsymbol{x}^i)$$

- We can cheaply get a good approximation of the gradient
- Easy implementation. It's simpler than methods using, for example, second derivative

Why Stochastic Gradient Widely Used? IV

- Now gradient is calculated by automatic differentiation
- We draw a network and the gradient can be calculated
- Non-convexity plays a role
  - For convex, other methods may possess advantages to more efficiently find the global minimum
  - But for non-convex, efficiency to reach a stationary point is less useful

Why Stochastic Gradient Widely Used? V

- A global minimum usually gives a good model (as loss is minimized), but for a stationary point we are less sure
- All these explain why SG is popular for deep learning

Conclusions

- In this talk, we only touch some aspects of machine learning such as empirical risk minimization and stochastic gradient methods
- These basic concepts are important for machine learning practitioners

References I

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT*, pages 4171–4186, 2019. doi: 10.18653/v1/n19-1423.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

I. J. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. The MIT Press, 2016.

S. Hanson and L. Pratt. Comparing biases for minimal network construction with back-propagation. In *Advances in Neural Information Processing Systems*, volume 1, 1988.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2015.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. 2012.

I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *Proceedings of International Conference on Learning Representations*, 2019.

References II

C.-H. Tsai, C.-Y. Lin, and C.-J. Lin. Incremental and decremental training for linear classification. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/ws/inc-dec.pdf`.

A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, pages 4148–4158, 2017.

T. Zhang, F. Wu, A. Katiyar, K. Q. Weinberger, and Y. Artzi. Revisiting few-sample BERT fine-tuning. In *Proceedings of International Conference on Learning Representations*, 2021.