Optimization Problems for Neural Networks

Chih-Jen Lin
National Taiwan University

Last updated: May 25, 2020

Outline

Outline

1. Regularized linear classification
2. Optimization problem for fully-connected networks
3. Optimization problem for convolutional neural networks (CNN)
4. Discussion

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

Minimizing Training Errors (Cont'd)

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

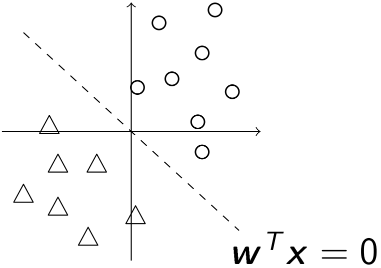
$$\operatorname{sgn}(\boldsymbol{w}^T \boldsymbol{x})$$

- 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

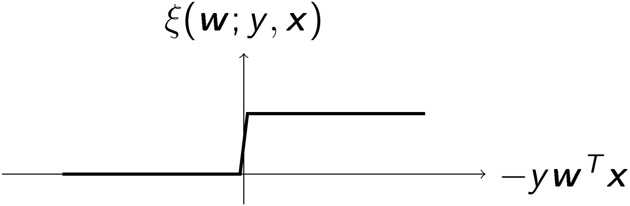
  $y = \pm 1$ is the label and $\boldsymbol{x}$ is the feature vector

- 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)

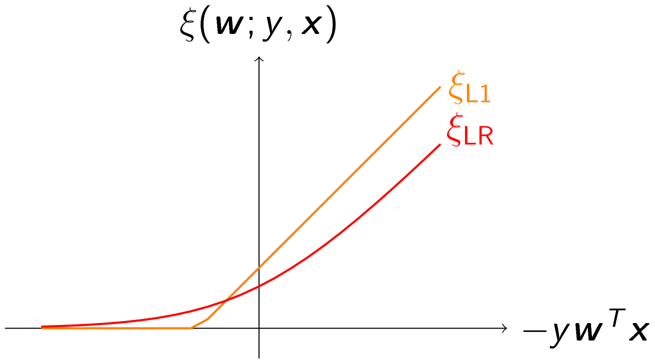
$$\xi_{\mathrm{L1}}(\boldsymbol{w}; y, \boldsymbol{x}) \equiv \max(0, 1 - y\boldsymbol{w}^T\boldsymbol{x}) \qquad (1)$$

- Logistic loss

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

- 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

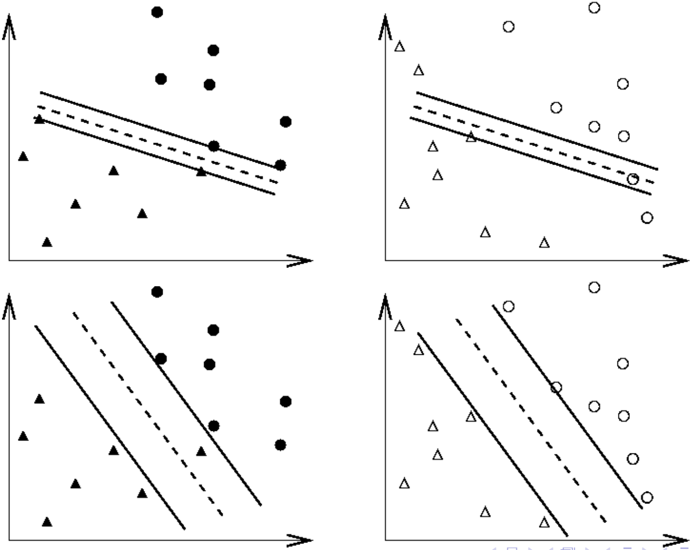
- See the illustration in the next slide
- For classification,

  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

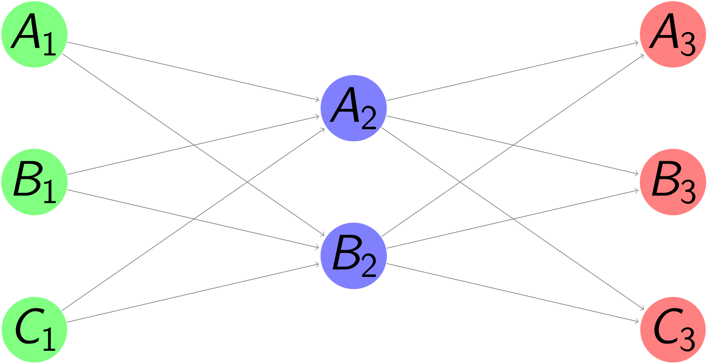
$$\boldsymbol{y}^i = [\underbrace{0, \ldots, 0}_{k-1}, 1, 0, \ldots, 0]^T \in R^K$$

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 $\boldsymbol{z}^m$ be the input of the mth layer, $\boldsymbol{z}^1 = \boldsymbol{x}$ and $\boldsymbol{z}^{L+1}$ be the output
- From mth layer to $(m+1)$th layer

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

$\sigma(\cdot)$ 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

$$R \to R$$

  transformation. As we are interested in optimization, let's not worry about why it's needed
- We collect all variables:

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

Operations Between Two Layers V

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,

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

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

  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

Why CNN? I

- There are many types of neural networks
- They are suitable for different types of problems
- While deep learning is hot, it's not always better than other learning methods
- For example, fully-connected networks were evalueated for general classification data (e.g., data from UCI machine learning repository)
- They are not consistently better than random forests or SVM; see the comparisons (Meyer et al., 2003; Fernández-Delgado et al., 2014; Wang et al., 2018).

Why CNN? II

- We are interested in CNN because it's shown to be significantly better than others on image data
- That's one of the main reasons deep learning becomes popular
- To study optimization algorithms, of course we want to consider an "established" network
- That's why CNN was chosen for our discussion
- However, the problem is that operations in CNN are more complicated than fully-connected networks
- Most books/papers only give explanation without detailed mathematical forms

Why CNN? III

- To study the optimization, we need some clean formulations
- So let's give it a try here

Convolutional Neural Networks I

- 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

- 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 $\boldsymbol{y}^i$

Convolutional Neural Networks II

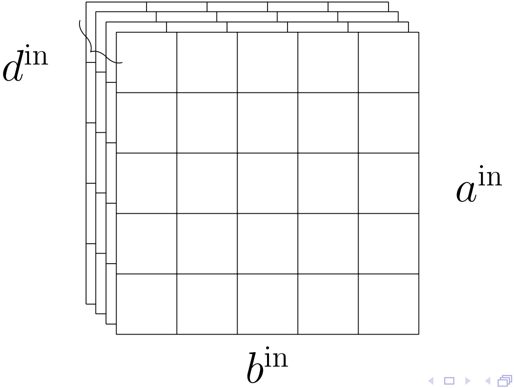
- Typically, CNN consists of multiple convolutional layers followed by fully-connected layers.
- Input and output of a convolutional layer are assumed to be images.

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.

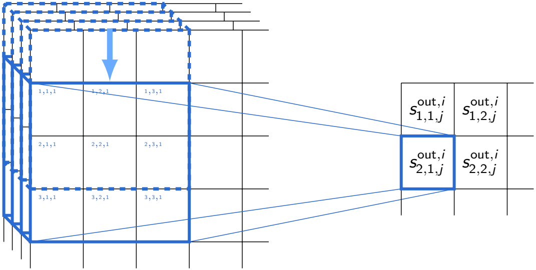
- Consider d^{out} filters.
- Filter $j \in \{1, \ldots, d^{\text{out}}\}$ has dimensions

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

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

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
- For example, if we start from the upper left corner of the input image, the first sub-image of channel d is

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

Convolutional Layers V

We then calculate

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

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

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

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 Operations I

- For efficient implementations, we should conduct convolutional operations by matrix-matrix and matrix-vector operations

  We will go back to this issue later

Matrix Operations II

- Let's collect images of all channels as the input

$$
\begin{aligned}
&Z^{\text{in},i} \\
=&\begin{bmatrix} z^i_{1,1,1} & z^i_{2,1,1} & \cdots & z^i_{a^{\text{in}},b^{\text{in}},1} \\ \vdots & \vdots & \ddots & \vdots \\ z^i_{1,1,d^{\text{in}}} & z^i_{2,1,d^{\text{in}}} & \cdots & z^i_{a^{\text{in}},b^{\text{in}},d^{\text{in}}} \end{bmatrix} \\
\in& R^{d^{\text{in}} \times a^{\text{in}} b^{\text{in}}}.
\end{aligned}
$$

Matrix Operations III

- Let all filters

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

  be variables (parameters) of the current layer

Matrix Operations IV

- 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 Operations V

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 Operations VI

Specifically,

$$
\phi(Z^{\text{in},i}) =
\begin{bmatrix}
z^i_{1,1,1} & z^i_{1+s,1,1} & & z^i_{1+(a^{\text{out}}-1)s,1+(b^{\text{out}}-1)s,1} \\
z^i_{2,1,1} & z^i_{2+s,1,1} & & z^i_{2+(a^{\text{out}}-1)s,1+(b^{\text{out}}-1)s,1} \\
\vdots & \vdots & \dots & \vdots \\
z^i_{h,h,1} & z^i_{h+s,h,1} & & z^i_{h+(a^{\text{out}}-1)s,h+(b^{\text{out}}-1)s,1} \\
\vdots & \vdots & & \vdots \\
z^i_{h,h,d^{\text{in}}} & z^i_{h+s,h,d^{\text{in}}} & & z^i_{h+(a^{\text{out}}-1)s,h+(b^{\text{out}}-1)s,d^{\text{in}}}
\end{bmatrix}
\in R^{hhd^{\text{in}} \times a^{\text{out}} b^{\text{out}}}
$$

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

- We need to represent $\phi(Z^{\text{in},i})$ in an explicit form.
- This is important for subsequent calculation
- 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 $\text{vec}(M)$

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

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

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

- Later we will check implementation details
- Past works using the form (9) include, for example, Vedaldi and Lenc (2015)

Optimization Problem I

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

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

- 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

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

C: regularization parameter.

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

Optimization Problem III

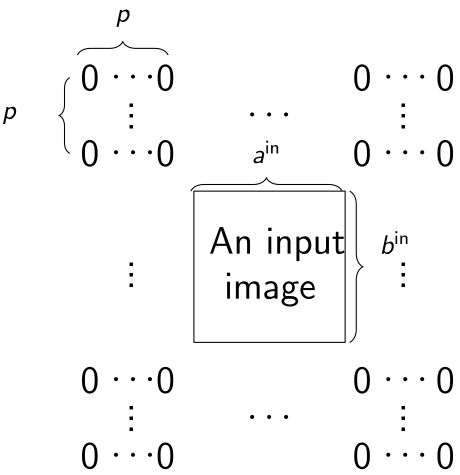
- Note that we divide the sum of training losses by the number of training data

  Thus the secnd term becomes the average training loss
- With the optimization problem, there is still a long way to do a real implementation
- Further, CNN involves additional operations in practice
  - padding
  - pooling
- We will explain them

Zero Padding I

- To better control the size of the output image, before the convolutional operation we may enlarge the input image to have zero values around the border.
- This technique is called zero-padding in CNN training.
- An illustration:

Zero Padding II



Zero Padding III

- The size of the new image is changed from

$$a^{\text{in}} \times b^{\text{in}} \text{ to } (a^{\text{in}} + 2p) \times (b^{\text{in}} + 2p),$$

  where p is specified by users
- The operation can be treated as a layer of mapping an input $Z^{\text{in},i}$ to an output $Z^{\text{out},i}$.
- Let

$$d^{\text{out}} = d^{\text{in}}.$$

Zero Padding IV

- There exists a 0/1 matrix

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

  so that the padding operation can be represented by

$$Z^{\text{out},i} \equiv \text{mat}(P_{\text{pad}} \text{vec}(Z^{\text{in},i}))_{d^{\text{out}} \times a^{\text{out}} b^{\text{out}}}. \tag{11}$$

- Implementation details will be discussed later

Pooling I

- To reduce the computational cost, a dimension reduction is often applied by a pooling step after convolutional operations.
- Usually we consider an operation that can (approximately) extract rotational or translational invariance features.
- Examples: average pooling, max pooling, and stochastic pooling,
- Let's consider max pooling as an illustration

Pooling II

- An example:

$$\text{image A} \quad \left[\begin{array}{cc|cc} 2 & 3 & 6 & 8 \\ 5 & 4 & 9 & 7 \\ \hline 1 & 2 & 6 & 0 \\ 4 & 3 & 2 & 1 \end{array}\right] \rightarrow \begin{bmatrix} 5 & 9 \\ 4 & 6 \end{bmatrix}$$

$$\text{image B} \quad \left[\begin{array}{cc|cc} 3 & 2 & 3 & 6 \\ 4 & 5 & 4 & 9 \\ \hline 2 & 1 & 2 & 6 \\ 3 & 4 & 3 & 2 \end{array}\right] \rightarrow \begin{bmatrix} 5 & 9 \\ 4 & 6 \end{bmatrix}$$

Pooling III

- B is derived by shifting A by 1 pixel in the horizontal direction.
- We split two images into four 2×2 sub-images and choose the max value from every sub-image.
- In each sub-image because only some elements are changed, the maximal value is likely the same or similar.
- This is called translational invariance
- For our example the two output images from A and B are the same.

Pooling IV

- For mathematical representation, we consider the operation as a layer of mapping an input $Z^{\text{in},i}$ to an output $Z^{\text{out},i}$.
- In practice pooling is considered as an operation at the end of the convolutional layer.
- We partition every channel of $Z^{\text{in},i}$ into non-overlapping sub-regions by $h \times h$ filters with the stride $s = h$
- Because of the disjoint sub-regions, the stride s for sliding the filters is equal to h.

Pooling V

- This partition step is a special case of how we generate sub-images in convolutional operations.
- By the same definition as (9) we can generate the matrix

$$\phi(Z^{\text{in},i}) = \text{mat}(P_\phi \text{vec}(Z^{\text{in},i}))_{hh \times d^{\text{out}} a^{\text{out}} b^{\text{out}}}, \qquad (12)$$

where

$$a^{\text{out}} = \lfloor \frac{a^{\text{in}}}{h} \rfloor, \ b^{\text{out}} = \lfloor \frac{b^{\text{in}}}{h} \rfloor, \ d^{\text{out}} = d^{\text{in}}. \qquad (13)$$

Pooling VI

- This is the same from the calculation in (5) as

$$\lfloor \frac{a^{\text{in}} - h}{h} \rfloor + 1 = \lfloor \frac{a^{\text{in}}}{h} \rfloor$$

- Note that here we consider

$$hh \times d^{\text{out}} a^{\text{out}} b^{\text{out}} \text{ rather than } hhd^{\text{out}} \times a^{\text{out}} b^{\text{out}}$$

  because we can then do a max operation on each column

Pooling VII

- To select the largest element of each sub-region, there exists a 0/1 matrix

$$M^i \in R^{d^{\text{out}} a^{\text{out}} b^{\text{out}} \times hhd^{\text{out}} a^{\text{out}} b^{\text{out}}}$$

so that each row of M^i selects a single element from $\text{vec}(\phi(Z^{\text{in},i}))$.

- Therefore,

$$Z^{\text{out},i} = \text{mat}\left(M^i \text{vec}(\phi(Z^{\text{in},i}))\right)_{d^{\text{out}} \times a^{\text{out}} b^{\text{out}}}. \quad (14)$$

Pooling VIII

- A comparison with (6) shows that M^i is in a similar role to the weight matrix W
- While M^i is $0/1$, it is not a constant. It's positions of 1's depend on the values of $\phi(Z^{\text{in},i})$
- By combining (12) and (14), we have

$$Z^{\text{out},i} = \text{mat}\left(P^i_{\text{pool}} \text{vec}(Z^{\text{in},i})\right)_{d^{\text{out}} \times a^{\text{out}} b^{\text{out}}}, \quad (15)$$

where

$$P^i_{\text{pool}} = M^i P_\phi \in R^{d^{\text{out}} a^{\text{out}} b^{\text{out}} \times d^{\text{in}} a^{\text{in}} b^{\text{in}}}. \quad (16)$$

Summary of a Convolutional Layer I

- For implementation, padding and pooling are (optional) part of the convolutional layers.
- We discuss details of considering all operations together.
- The whole convolutional layer involves the following procedure:

$$
\begin{aligned}
Z^{m,i} &\rightarrow \text{padding by (11)} \rightarrow \\
&\quad \text{convolutional operations by (6), (7)} \\
&\rightarrow \text{pooling by (15)} \rightarrow Z^{m+1,i}, \qquad (17)
\end{aligned}
$$

Summary of a Convolutional Layer II

where $Z^{m,i}$ and $Z^{m+1,i}$ are input and output of the mth layer, respectively.

- Let the following symbols denote image sizes at different stages of the convolutional layer.

$$
\begin{aligned}
a^m,\ b^m &: \text{ size in the beginning} \\
a^m_{\text{pad}},\ b^m_{\text{pad}} &: \text{ size after padding} \\
a^m_{\text{conv}},\ b^m_{\text{conv}} &: \text{ size after convolution.}
\end{aligned}
$$

- The following table indicates how these values are $a^{\text{in}}, b^{\text{in}}, d^{\text{in}}$ and $a^{\text{out}}, b^{\text{out}}, d^{\text{out}}$ at different stages.

Summary of a Convolutional Layer III

Operation	Input	Output
Padding: (11)	$Z^{m,i}$	$\text{pad}(Z^{m,i})$
Convolution: (6)	$\text{pad}(Z^{m,i})$	$S^{m,i}$
Convolution: (7)	$S^{m,i}$	$\sigma(S^{m,i})$
Pooling: (15)	$\sigma(S^{m,i})$	$Z^{m+1,i}$

Operation	$a^{\text{in}}, \ b^{\text{in}}, \ d^{\text{in}}$	$a^{\text{out}}, \ b^{\text{out}}, \ d^{\text{out}}$
Padding: (11)	$a^m, \ b^m, \ d^m$	$a^m_{\text{pad}}, \ b^m_{\text{pad}}, \ d^m$
Convolution: (6)	$a^m_{\text{pad}}, \ b^m_{\text{pad}}, \ d^m$	$a^m_{\text{conv}}, \ b^m_{\text{conv}}, \ d^{m+1}$
Convolution: (7)	$a^m_{\text{conv}}, \ b^m_{\text{conv}}, \ d^{m+1}$	$a^m_{\text{conv}}, \ b^m_{\text{conv}}, \ d^{m+1}$
Pooling: (15)	$a^m_{\text{conv}}, \ b^m_{\text{conv}}, \ d^{m+1}$	$a^{m+1}, \ b^{m+1}, \ d^{m+1}$

Summary of a Convolutional Layer IV

- Let the filter size, mapping matrices and weight matrices at the mth layer be

$$h^m,\ P^m_{\text{pad}},\ P^m_\phi,\ P^{m,i}_{\text{pool}},\ W^m,\ \boldsymbol{b}^m.$$

- From (11), (6), (7), (15), all operations can be summarized as

$$S^{m,i} = W^m \text{mat}(P^m_\phi P^m_{\text{pad}} \text{vec}(Z^{m,i}))_{h^m h^m d^m \times a^m_{\text{conv}} b^m_{\text{conv}}} + \boldsymbol{b}^m \mathbb{1}^T_{a^{\text{conv}} b^{\text{conv}}}$$

$$Z^{m+1,i} = \text{mat}(P^{m,i}_{\text{pool}} \text{vec}(\sigma(S^{m,i})))_{d^{m+1} \times a^{m+1} b^{m+1}}, \tag{18}$$

Fully-Connected Layer I

- Assume L^C is the number of convolutional layers
- Input vector of the first fully-connected layer:

$$z^{m,i} = \text{vec}(Z^{m,i}), \ i = 1, \ldots, l, \ m = L^c + 1.$$

- In each of the fully-connected layers ($L^c < m \leq L$), we consider weight matrix and bias vector between layers m and $m+1$.

Fully-Connected Layer II

- Weight matrix:

$$W^m = \begin{bmatrix} w^m_{11} & w^m_{12} & \cdots & w^m_{1n_m} \\ w^m_{21} & w^m_{22} & \cdots & w^m_{2n_m} \\ \vdots & \vdots & \vdots & \vdots \\ w^m_{n_{m+1}1} & w^m_{n_{m+1}2} & \cdots & w^m_{n_{m+1}n_m} \end{bmatrix}_{n_{m+1} \times n_m} \tag{19}$$

- Bias vector

$$\boldsymbol{b}^m = \begin{bmatrix} b^m_1 \\ b^m_2 \\ \vdots \\ b^m_{n_{m+1}} \end{bmatrix}_{n_{m+1} \times 1}$$

Fully-Connected Layer III

Here n_m and n_{m+1} are the numbers of nodes in layers m and $m+1$, respectively.

- If $\boldsymbol{z}^{m,i} \in R^{n_m}$ is the input vector, the following operations are applied to generate the output vector $\boldsymbol{z}^{m+1,i} \in R^{n_{m+1}}$.

$$\boldsymbol{s}^{m,i} = W^m \boldsymbol{z}^{m,i} + \boldsymbol{b}^m, \tag{20}$$

$$z_j^{m+1,i} = \sigma(s_j^{m,i}), \ j = 1, \ldots, n_{m+1}. \tag{21}$$

Outline

Challenges in NN Optimization

- The objective function is non-convex. It may have many local minima
- It's known that global optimization is much more difficult than local minimization
- The problem structure is very complicated
- In this course we will have first-hand experiences on handling these difficulties

Formulation I

- We have written all CNN operations in matrix/vector forms
- This is useful in deriving the gradient
- Are our representation symbols good enough? Can we do better?
- You can say that this is only a matter of notation, but given the wide use of CNN, a good formulation can be extremely useful

References I

M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15:3133–3181, 2014.

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.

D. Meyer, F. Leisch, and K. Hornik. The support vector machine under test. *Neurocomputing*, 55:169–186, 2003.

A. Vedaldi and K. Lenc. MatConvNet: Convolutional neural networks for matlab. In *Proceedings of the 23rd ACM International Conference on Multimedia*, pages 689–692, 2015.

C.-C. Wang, K.-L. Tan, C.-T. Chen, Y.-H. Lin, S. S. Keerthi, D. Mahajan, S. Sundararajan, and C.-J. Lin. Distributed Newton methods for deep learning. *Neural Computation*, 30(6): 1673–1724, 2018. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/dnn/dsh.pdf`.