Optimization Problems for Neural Networks

Chih-Jen Lin
National Taiwan University

Last updated: April 9, 2019
Outline

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

\[
\min_{\text{model}} (\text{training errors})
\]

- 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
For simplicity, let’s consider the model to be a vector $w$

That is, the decision function is

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

For any data, $x$, the predicted label is

$$\begin{cases} 1 & \text{if } w^T x \geq 0 \\ -1 & \text{otherwise} \end{cases}$$
The two-dimensional situation

\[ w^T x = 0 \]

This seems to be quite restricted, but practically \( x \) is in a much higher dimensional space
To characterize the training error, we need a loss function $\xi(w; y, x)$ for each instance $(y, x)$, where $y = \pm 1$ is the label and $x$ is the feature vector.

Ideally we should use 0–1 training loss:

$$\xi(w; y, x) = \begin{cases} 
1 & \text{if } yw^T x < 0, \\
0 & \text{otherwise}
\end{cases}$$
However, this function is discontinuous. The optimization problem becomes difficult

\[ \xi(w; y, x) \]

We need continuous approximations
Common Loss Functions

- Hinge loss (l1 loss)

\[
\xi_{L1}(\mathbf{w}; y, x) \equiv \max(0, 1 - yw^T x) \tag{1}
\]

- Logistic loss

\[
\xi_{LR}(\mathbf{w}; y, x) \equiv \log(1 + e^{-yw^T x}) \tag{2}
\]

- Support vector machines (SVM): Eq. (1).
- Logistic regression (LR): (2)

SVM and LR are two very fundamental classification methods.
Logistic regression is very related to SVM
Their performance is usually similar
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 $w$ vector so that it fits the data.
- To avoid overfitting we need a way to make $w$’s values less extreme.
- One idea is to make $w$ values closer to zero.
- We can add, for example,

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

...to the function that is minimized.
General Form of Linear Classification

- Training data \( \{y_i, x_i\}, x_i \in \mathbb{R}^n, i = 1, \ldots, l, y_i = \pm 1 \)
- \( l \): # of data, \( n \): # of features

\[
\min_w f(w), \quad f(w) \equiv \frac{w^T w}{2} + C \sum_{i=1}^{l} \xi(w; y_i, x_i)
\]

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

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

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

- \(K\): # of classes
- If \(x^i\) is in class \(k\), then

\[
y^i = [0, \ldots, 0, 1, 0, \ldots, 0]^T \in \mathbb{R}^K
\]

\(k-1\)
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 $m$th layer is

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

- $n_m$: # 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

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

$$s^m = W^m z^m,$$

$$z_{j}^{m+1} = \sigma(s_{j}^{m}), \quad j = 1, \ldots, n_{m+1},$$

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

\[
s^m = W^m z^m + b^m
\]
Activation function is an $R \rightarrow R$ transformation. As we are interested in optimization, let’s not worry about why it’s needed.

We collect all variables:

$$\theta = \begin{bmatrix} \text{vec}(W^1) \\ b^1 \\ \vdots \\ \text{vec}(W^L) \\ b^L \end{bmatrix} \in R^n$$
Operations Between Two Layers V

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

- The \( \text{vec}(\cdot) \) operator stacks columns of a matrix to a vector
We solve the following optimization problem,

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

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

$C$: regularization parameter

$z^{L+1}(\theta) \in R^{n_{l+1}}$: last-layer output vector of $x$.

$\xi(z^{L+1}; y, x)$: loss function. Example:

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

$$s^{m,i} = W^m z^{m,i},$$

$$z^{m+1,i}_j = \sigma(s^{m,i}_j), \quad j = 1, \ldots, n_{m+1},$$

This makes the training more complicated.
Outline

1. Regularized linear classification
2. Optimization problem for fully-connected networks
3. Optimization problem for convolutional neural networks (CNN)
4. Discussion
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 evaluated 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
Consider a $K$-class classification problem with training data

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

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

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

$$y^i = [0, \ldots, 0, 1, 0, \ldots, 0]^T \in \mathbb{R}^K.$$ 

$k-1$

CNN maps each image $Z^{1,i}$ to $y^i$
Typically, CNN consists of multiple convolutional layers followed by fully-connected layers.

Input and output of a convolutional layer are assumed to be images.
For the current layer, let the input be an image

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

\( a^{\text{in}} \): height, \( b^{\text{in}} \): width, and \( d^{\text{in}} \): \#channels.
Convolutional Layers II

The goal is to generate an output image

\[ Z_{\text{out},i} \]

of \( d_{\text{out}} \) channels of \( a_{\text{out}} \times b_{\text{out}} \) images.

- Consider \( d_{\text{out}} \) filters.
- Filter \( j \in \{1, \ldots, d_{\text{out}}\} \) has dimensions \( h \times h \times d_{\text{in}} \).

\[
\begin{bmatrix}
w_{1,1,1}^j & w_{1,h,1}^j \\
\vdots & \vdots \\
w_{h,1,1}^j & w_{h,h,1}^j
\end{bmatrix}
\cdots
\begin{bmatrix}
w_{1,1,d_{\text{in}}}^j & w_{1,h,d_{\text{in}}}^j \\
\vdots & \vdots \\
w_{h,1,d_{\text{in}}}^j & w_{h,h,d_{\text{in}}}^j
\end{bmatrix}
\]
To compute the $j$th channel of output, we scan the input from top-left to bottom-right to obtain the sub-images of size $h \times h \times d^\text{in}$
We then calculate the inner product between each sub-image and the $j$th filter.

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

$$\begin{bmatrix}
Z_{1,1,d}^i & \cdots & Z_{1,h,d}^i \\
\vdots & \ddots & \vdots \\
Z_{h,1,d}^i & \cdots & Z_{h,h,d}^i
\end{bmatrix}.$$
We then calculate

\[
\sum_{d=1}^{d_{\text{in}}} \left\langle \begin{bmatrix} z_{1,1,d}^i & \cdots & z_{1,h,d}^i \\ \vdots & \ddots & \vdots \\ z_{h,1,d}^i & \cdots & z_{h,h,d}^i \end{bmatrix}, \begin{bmatrix} w_{1,1,d}^j & \cdots & w_{1,h,d}^j \\ \vdots & \ddots & \vdots \\ w_{h,1,d}^j & \cdots & w_{h,h,d}^j \end{bmatrix} \right\rangle + b_j,
\]

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.
- 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:

\[
\begin{bmatrix}
z_{1+s,1,d}^i & \cdots & z_{1+s,h,d}^i \\
\vdots & \ddots & \vdots \\
z_{h+s,1,d}^i & \cdots & z_{h+s,h,d}^i
\end{bmatrix}.
\]
Convolutional Layers VII

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

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

(4)
Assume that vertically and horizontally we can move the filter \( a^{\text{out}} \) and \( b^{\text{out}} \) times, respectively.

\[
a^{\text{out}} = \left\lfloor \frac{a^{\text{in}} - h}{s} \right\rfloor + 1, \quad b^{\text{out}} = \left\lfloor \frac{b^{\text{in}} - h}{s} \right\rfloor + 1
\] (5)
For efficient implementations, we should conduct convolutional operations by matrix-matrix and matrix-vector operations.

We will go back to this issue later.
Let’s collect images of all channels as the input.

\[
Z_{\text{in},i} = 
\begin{bmatrix}
    z_{1,1,1}^i & z_{2,1,1}^i & \cdots & z_{a_{\text{in}},b_{\text{in}},1}^i \\
    \vdots & \vdots & \ddots & \vdots \\
    z_{1,1,d_{\text{in}}}^i & z_{2,1,d_{\text{in}}}^i & \cdots & z_{a_{\text{in}},b_{\text{in}},d_{\text{in}}}^i
\end{bmatrix}
\in \mathbb{R}^{d_{\text{in}} \times a_{\text{in}} b_{\text{in}}}.
\]
Let all filters

\[ W = \begin{bmatrix}
    w_{1,1,1} & w_{1,1,1} & \cdots & w_{h,h,d_{in}} \\
    \vdots & \vdots & \ddots & \vdots \\
    w_{d_{out},1,1} & w_{d_{out},1,1} & \cdots & w_{d_{out},h,h,d_{in}} \\
\end{bmatrix} \in \mathbb{R}^{d_{out} \times hh d_{in}} \]

be variables (parameters) of the current layer
Usually a bias term is considered

\[ b = \begin{bmatrix} b_1 \\ \vdots \\ b_{d_{out}} \end{bmatrix} \in \mathbb{R}^{d_{out} \times 1} \]

Operations at a layer

\[ S_{out,i} = W\phi(Z_{in,i}) + b_1^T a_{out} b_{out} \in \mathbb{R}^{d_{out} \times a_{out} b_{out}} \] (6)
where

$$\mathbf{1}_{a_{\text{out}} b_{\text{out}}} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \in \mathbb{R}^{a_{\text{out}} b_{\text{out}} \times 1}.$$ 

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

\[ \phi(Z_{\text{in},i}) = \begin{bmatrix}
Z_{1,1,1} & Z_{1}^i + s,1,1 \\
Z_{2,1,1} & Z_{2}^i + s,1,1 \\
\vdots & \vdots & \ddots \\
Z_{h,h,1} & Z_{h}^i + s,h,1 \\
\vdots & \vdots & \ddots \\
Z_{h,h,d_{\text{in}}} & Z_{h}^i + s,h,d_{\text{in}} \\
\end{bmatrix} \in \mathbb{R}^{hhd_{\text{in}} \times a_{\text{out}} b_{\text{out}}}
\]
Next, an activation function scales each element of $S^{\text{out},i}$ to obtain the output matrix $Z^{\text{out},i}$.

$$Z^{\text{out},i} = \sigma(S^{\text{out},i}) \in \mathbb{R}^{d_{\text{out}} \times a_{\text{out}} b_{\text{out}}}. \quad (7)$$

For CNN, commonly the following RELU activation function

$$\sigma(x) = \max(x, 0) \quad (8)$$

is used (reasons?)

Later we need that $\sigma(x)$ is differentiable, but the RELU function is not.
Past works such as Krizhevsky et al. (2012) assume

\[ \sigma'(x) = \begin{cases} 
1 & \text{if } x > 0 \\
0 & \text{otherwise} 
\end{cases} \]
Optimization problem for convolutional neural networks (CNN)

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

- In the matrix-matrix product
  
  $W \phi(Z^{in,i}),$

  each element is the inner product between a filter and a sub-image
- We need to represent $\phi(Z^{in,i})$ in an explicit form.
- This is important for subsequent calculation
- Clearly $\phi$ is a linear mapping, so there exists a 0/1 matrix $P^m_{\phi}$ such that

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

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

$$\text{vec}(M) = \left[ \begin{array}{c} M_{:,1} \\ \vdots \\ M_{:,b} \end{array} \right] \in \mathbb{R}^{ab \times 1}, \text{ where } M \in \mathbb{R}^{a \times b}$$

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

$$\text{mat}(v)_{a \times b} = \left[ \begin{array}{ccc} v_1 & \cdots & v_{(b-1)a+1} \\ \vdots & \ddots & \vdots \\ v_a & \cdots & v_{ba} \end{array} \right] \in \mathbb{R}^{a \times b}, \quad (10)$$

where

$$v \in \mathbb{R}^{ab \times 1}.$$
The Function $\phi(Z^{in,i})$

- $P_\phi$ is a huge matrix:

$$P_\phi \in \mathbb{R}^{hd^{in}a^{out}b^{out} \times d^{in}a^{in}b^{in}}$$

and

$$\phi : \mathbb{R}^{d^{in}a^{in}b^{in}} \rightarrow \mathbb{R}^{hd^{in}a^{out}b^{out}}$$

- Later we will check implementation details
- Past works using the form (9) include, for example, Vedaldi and Lenc (2015)
Optimization Problem 1

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

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

- The output of the last layer $L$ is a vector $z^{L+1,i}(\theta)$. 
Optimization Problem II

- Consider any loss function such as the squared loss
  \[ \xi_i(\theta) = \| z^{L+1,i}(\theta) - y^i \|^2. \]

- The optimization problem is
  \[ \min_{\theta} f(\theta), \]
  where
  \[ f(\theta) = \frac{1}{2C} \theta^T \theta + \frac{1}{l} \sum_{i=1}^{l} \xi(z^{L+1,i}(\theta); y^i, Z^{1,i}) \]

  \( C \): regularization parameter.
Optimization Problem III

- The formulation is almost the same as that for fully connected networks.
- Note that we divide the sum of training losses by the number of training data.
  Thus the second 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
Optimization Problem IV

- 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

An input image

\[
p \begin{cases} 0 \cdots 0 \\ \vdots \\ 0 \cdots 0 \end{cases} \quad \begin{cases} 0 \cdots 0 \\ \vdots \\ 0 \cdots 0 \end{cases}
\]

\[
\begin{cases} 0 \cdots 0 \\ \vdots \\ 0 \cdots 0 \end{cases} \quad \begin{cases} 0 \cdots 0 \\ \vdots \\ 0 \cdots 0 \end{cases}
\]

Chih-Jen Lin (National Taiwan Univ.)
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

$$P_{\text{pad}} \in \mathbb{R}^{d_{\text{out}} a_{\text{out}} b_{\text{out}} \times d_{\text{in}} a_{\text{in}} b_{\text{in}}}$$

so that the padding operation can be represented by

$$Z_{\text{out},i} \equiv \text{mat}(P_{\text{pad}} \text{vec}(Z_{\text{in},i}))_{d_{\text{out}} \times a_{\text{out}} b_{\text{out}}}. \quad (11)$$

Implementation details will be discussed later
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
Optimization problem for convolutional neural networks (CNN)

Pooling II

An example:

image A

\[
\begin{bmatrix}
2 & 3 & 6 & 8 \\
5 & 4 & 9 & 7 \\
1 & 2 & 6 & 0 \\
4 & 3 & 2 & 1 \\
\end{bmatrix}
\rightarrow
\begin{bmatrix}
5 & 9 \\
4 & 6 \\
\end{bmatrix}
\]

image B

\[
\begin{bmatrix}
3 & 2 & 3 & 6 \\
4 & 5 & 4 & 9 \\
2 & 1 & 2 & 6 \\
3 & 4 & 3 & 2 \\
\end{bmatrix}
\rightarrow
\begin{bmatrix}
5 & 9 \\
4 & 6 \\
\end{bmatrix}
\]
Pooling III

- B is derived by shifting A by 1 pixel in the horizontal direction.
- We split two images into four $2 \times 2$ 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 $h$. 
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

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

where

$$
a^{out} = \left[ \begin{array}{c} a^{in} \\ h \end{array} \right], \quad b^{out} = \left[ \begin{array}{c} b^{in} \\ h \end{array} \right], \quad d^{out} = d^{in}.
$$
Pooling VI

- Note that here we consider

\[ hh \times d^{\text{out}} a^{\text{out}} b^{\text{out}} \] rather than \( hh d^{\text{out}} \times a^{\text{out}} b^{\text{out}} \)

because we can then do a max operation on each column.

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

\[ M^i \in \mathbb{R}^{d^{\text{out}} a^{\text{out}} b^{\text{out}} \times hhd^{\text{out}} a^{\text{out}} b^{\text{out}}} \]

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

\[ Z^{\text{out},i} = \text{mat}\left( M^i \text{vec}\left( \phi\left( Z^{\text{in},i}\right) \right) \right)_{d^{\text{out}} \times a^{\text{out}} b^{\text{out}}} . \]  \hspace{1cm} (14)

A comparison with (6) shows that \( M^i \) is in a similar role to the weight matrix \( W \) though \( M^i \) is a constant.

By combining (12) and (14), we have

\[ Z^{\text{out},i} = \text{mat}\left( P_{\text{pool}}^i \text{vec}(Z^{\text{in},i}) \right)_{d^{\text{out}} \times a^{\text{out}} b^{\text{out}}} , \]  \hspace{1cm} (15)

where

\[ P_{\text{pool}}^i = M^i P_{\phi} \in R^{d^{\text{out}} a^{\text{out}} b^{\text{out}} \times d^{\text{in}} a^{\text{in}} b^{\text{in}}} . \]  \hspace{1cm} (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:

\[ Z_{m,i} \rightarrow \text{padding by (11)} \rightarrow \text{convolutional operations by (6), (7)} \rightarrow \text{pooling by (15)} \rightarrow Z_{m+1,i}, \quad (17) \]
where $Z^{m,i}$ and $Z^{m+1,i}$ are input and output of the $m$th layer, respectively.

- Let the following symbols denote image sizes at different stages of the convolutional layer.
  
  \[
  a^m, \ b^m : \text{size in the beginning} \\
  a^m_{\text{pad}}, \ b^m_{\text{pad}} : \text{size after padding} \\
  a^m_{\text{conv}}, \ b^m_{\text{conv}} : \text{size after convolution}.
  \]

- 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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Padding: (11)</td>
<td>$Z^{m,i}$</td>
<td>$\text{pad}(Z^{m,i})$</td>
</tr>
<tr>
<td>Convolution: (6)</td>
<td>$\text{pad}(Z^{m,i})$</td>
<td>$S^{m,i}$</td>
</tr>
<tr>
<td>Convolution: (7)</td>
<td>$S^{m,i}$</td>
<td>$\sigma(S^{m,i})$</td>
</tr>
<tr>
<td>Pooling: (15)</td>
<td>$\sigma(S^{m,i})$</td>
<td>$Z^{m+1,i}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>$a^{\text{in}}, b^{\text{in}}, d^{\text{in}}$</th>
<th>$a^{\text{out}}, b^{\text{out}}, d^{\text{out}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Padding: (11)</td>
<td>$a^m, b^m, d^m$</td>
<td>$a^\text{pad}, b^\text{pad}, d^m$</td>
</tr>
<tr>
<td>Convolution: (6)</td>
<td>$a^\text{pad}, b^\text{pad}, d^m$</td>
<td>$a^\text{conv}, b^\text{conv}, d^{m+1}$</td>
</tr>
<tr>
<td>Convolution: (7)</td>
<td>$a^\text{conv}, b^\text{conv}, d^{m+1}$</td>
<td>$a^{m+1}, b^{m+1}, d^{m+1}$</td>
</tr>
<tr>
<td>Pooling: (15)</td>
<td>$a^\text{conv}, b^\text{conv}, d^{m+1}$</td>
<td>$a^{m+1}, b^{m+1}, d^{m+1}$</td>
</tr>
</tbody>
</table>
Summary of a Convolutional Layer IV

- Let the filter size, mapping matrices and weight matrices at the $m$th layer be

\[ h^m, \ P^m_{\text{pad}}, \ P^m_{\phi}, \ P^m_{\text{pool}}, \ W^m, \ b^m. \]

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

\[ S^{m,i} = W^m \text{mat}(P^m_{\phi} P^m_{\text{pad}} \text{vec}(Z^m,i)) h^m h^m d^m \times a^m_{\text{conv}} b^m_{\text{conv}} + b^m_{1^T} a^m_{\text{conv}} b^m_{\text{conv}} \]

\[ Z^{m+1,i} = \text{mat}(P^m_{\text{pool}} \text{vec}(\sigma(S^{m,i}))) d^{m+1} \times a^{m+1} b^{m+1}, \]

(18)
Fully-Connected Layer I

- Input vector of the first fully-connected layer:

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

- 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:

\[
W^m = \begin{bmatrix}
  w_{11}^m & w_{12}^m & \cdots & w_{1n_m}^m \\
  w_{21}^m & w_{22}^m & \cdots & w_{2n_m}^m \\
  \vdots & \vdots & \ddots & \vdots \\
  w_{n_{m+1}1}^m & w_{n_{m+1}2}^m & \cdots & w_{n_{m+1}n_m}^m
\end{bmatrix}_{n_{m+1} \times n_m}
\]  \hspace{1cm} (19)

- Bias vector

\[
b^m = \begin{bmatrix}
  b_1^m \\
  b_2^m \\
  \vdots \\
  b_{n_{m+1}}^m
\end{bmatrix}_{n_{m+1} \times 1}
\]
Here $n_m$ and $n_{m+1}$ are the numbers of nodes in layers $m$ and $m + 1$, respectively.

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

\[
\begin{align*}
    s^{m,i} &= W^m z^{m,i} + b^m, \\
    z_{j}^{m+1,i} &= \sigma(s_{j}^{m,i}), \quad j = 1, \ldots, n_{m+1}.
\end{align*}
\]
Outline

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

- The objective function is non-convex. It may have many local minima.
- Traditionally 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 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


