Jinchao Xu

PSU-PKU Joint Course (497): An Introduction to Deep Learning

Summer 2019

Contributors:

This set of notes are based on contributions from many of graduate students, post-doctoral fellows and other collaborators. Here is a partial list:

Juncai He, Li Jiang, Shaobo Liang, Pengfei Yin, Qian Zhang.....

Contents

1

An Introduction to Machine Learning and DNN

1.1 Notes for 1st day of the first week

The most content of this section is on the slides, here is just some supplement for the slides.

1. Heaviside function.

$$H(x) = \begin{cases} 1, & x > 0, \\ 0, & x \leq 0. \end{cases} \tag{1.1}$$

2.

$$s(x) = \frac{1}{1 + e^{-x}} \to \begin{cases} 0, & x \to -\infty, \\ 1, & x \to +\infty. \end{cases} \tag{1.2}$$

3. Rectified linear unit.

$$\mathrm{ReLU}(x) = \max\{0, x\} = \begin{cases} x, & x \geq 0, \\ 0, & x < 0. \end{cases} \tag{1.3}$$

Notice that

$$\frac{d}{dx}\mathrm{ReLU}(x) = H(x). \tag{1.4}$$

Given a vector x, we define two kinds of operations. One is linear operation which maps x to $wx + b$, while another operation is activation which maps x to $\sigma(x)$ with an activation function σ.

Deep neural network can be regarded as compositions and combinations of the above two operations.

1.2 Notes for 2nd day of the first week

1.2.1 Notations

We use $\mathbb{R}^d$ to denote the d-dimensional real vector space, and use $\mathbb{R}^{k\times d}$ to denote the $(k \times d)$-dimensional real matrix space. For example,

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_d \end{pmatrix} \in \mathbb{R}^d. \tag{1.5}$$

$$W = \begin{pmatrix} w_1 \\ w_2 \\ \cdots \\ w_d \end{pmatrix} = \begin{pmatrix} w_{11}\ w_{12}\ \cdots\ w_{1d} \\ w_{21}\ w_{22}\ \cdots\ w_{2d} \\ \cdots \\ w_{k1}\ w_{k2}\ \cdots\ w_{kd} \end{pmatrix} \in \mathbb{R}^{k\times d}. \tag{1.6}$$

where $w_i = (w_{i1}\ w_{i2}\ \cdots\ w_{id}) \in R^{1\times d}$.
Given $W \in \mathbb{R}^{k\times d}$ and $b \in \mathbb{R}^d$, a linear function $f : \mathbb{R}^d \to \mathbb{R}^k$ can be defined as

$$f(x) = Wx + b = \begin{pmatrix} w_1 x + b_1 \\ w_2 x + b_2 \\ \cdots \\ w_k x + b_k \end{pmatrix} \in \mathbb{R}^k, \quad \forall x \in \mathbb{R}^d. \tag{1.7}$$

In $\mathbb{R}^d$, given $w \in \mathbb{R}^d, b \in \mathbb{R}$, the point set $\{x : wx + b = 0\}$ is called a hyperplane, or $(d-1)$-dimensional hyperplane. A hyperplane is a point in a 1-d space, a straight line in a 2-d space and a plane in a 3-d space.

1.2.2 Linearly separable sets

Definition 1. *The two sets A_1, $A_2 \subset \mathbb{R}^d$ are linearly separable if and only if there exists a hyperplane*

$$H_0 = \{x : wx + b = 0\}, \tag{1.8}$$

such that $wx + b > 0$ if $x \in A_1$ and $wx + b < 0$ if $x \in A_2$.

A intuitive explanation of the definition is that two sets are linearly separable iff they can be separated by a hyperplane. But when it comes to the multi-class case, we don't have such intuitive explanation.

Definition 2 (Linearly Separable). *A collection of subsets $A_1, ..., A_k \subset \mathbb{R}^d$ are linearly separable if there exists*

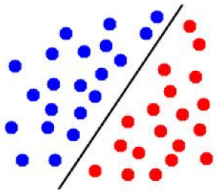


Fig. 1.1. Two linearly separable sets

$$W = \begin{pmatrix} w_1 \\ \vdots \\ w_k \end{pmatrix} \in \mathbb{R}^{k\times d}, b = \begin{pmatrix} b_1 \\ \vdots \\ b_k \end{pmatrix} \in \mathbb{R}^k, \tag{1.9}$$

such that, for each $1 \le i \le k$ *and* $j \ne i$

$$(Wx + b)_i > (Wx + b)_j, \ \forall x \in A_i, \tag{1.10}$$

or

$$w_i x + b_i > w_j x + b_j, \ \forall x \in A_i. \tag{1.11}$$

Lemma 1. *Assume that* $A_1, ..., A_k$ *are linearly separable and* $\boldsymbol{\theta} = (W, b)$ *where* $W \in \mathbb{R}^{k\times n}$ *and* $b \in \mathbb{R}^k$ *satisfy* (1.11)*. Define*

$$\Gamma_i(\boldsymbol{\theta}) = \{x \in \mathbb{R}^n : (Wx + b)_i > (Wx + b)_j, \ \forall j \ne i\} \tag{1.12}$$

Then for each i,

$$A_i \subset \Gamma_i(\boldsymbol{\theta}) \tag{1.13}$$

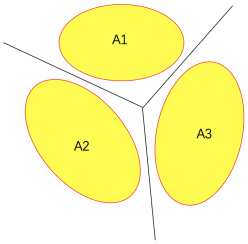


Fig. 1.2. linearly separable sets in 2-d space while k = 3

We can obtain many facts of linearly separable sets in this lemma.

1. Denotes $\partial\Gamma_i(\boldsymbol{\theta})$ as the boundary of $\Gamma_i(\boldsymbol{\theta})$, and the union of all boundaries $\partial\Gamma$ as

$$\partial\Gamma = \bigcup_{i=1}^{k} \partial\Gamma_i(\boldsymbol{\theta}) \tag{1.14}$$

then we have

$$\Gamma_1(\boldsymbol{\theta}) \cup \Gamma_2(\boldsymbol{\theta}) \cup \cdots \cup \Gamma_k(\boldsymbol{\theta}) \cup \partial\Gamma = \mathbb{R}^d, \tag{1.15}$$

where $\Gamma_i(\boldsymbol{\theta}) \cap \Gamma_j(\boldsymbol{\theta}) = \emptyset$ and $\partial\Gamma$ is a set of measure zero.
This means that most points of $\mathbb{R}^d$ will be attached to a unique class unless a set of measure zero.
2. Each $\Gamma_i(\boldsymbol{\theta})$ is a polygon whose boundary consists of hyperplanes

$$H_{ij} = \{(w_i - w_j) \cdot x + (b_i - b_j) = 0\}, \quad \forall j \neq i. \tag{1.16}$$

3. Different W, bs can generate the same classifier. The truly effective parameters are $(w_i - w_j)$ and $(b_i - b_j)$ for all $i \neq j$ which means if we add a same vector to w_i and a same scalar to b_i for all $1 \leq i \leq k$, it won't affect the result of clssification.

1.2.3 Other notions of linearly separable sets

Definition 3 (All-vs-One Linearly Separable). *A collection of subsets $A_1, ..., A_k \subset \mathbb{R}^d$ is all-vs-one linearly separable if for each $i = 1, ..., k$, A_i and $\cup_{j \neq i} A_j$ are linearly separable.*

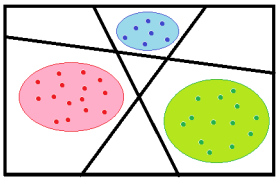


Fig. 1.3. All-vs-One linearly separable sets

This kind of linearly separable is intuitive. That means for each class we have a hyperplane to separate it from the rest. Easy to observe that it's equivelant that there exists a W, b such that

$$w_i x + b_i > 0, w_j x + b_j < 0, \forall x \in A_i, j \neq i, i = 1, \cdots, k. \tag{1.17}$$

Definition 4 (Pairwise Linearly Separable). *A collection of subsets $A_1, ..., A_k \subset \mathbb{R}^d$ is pairwise linearly separable if for each pair of indices $1 \leq i < j \leq k$, A_i and A_j are linearly separable.*

This notion is also easy to understand. It means each two different classes can be linearly separated. Linearly separable sets is a special type of pairwise linearly separable sets because it requires some consistency with the parameters. Mathematically, we can write this consistency as

$$w_{ij} = w_{ik} + w_{kj}, b_{ij} = b_{ik} + b_{kj} \ \ \forall k \neq i, j. \tag{1.18}$$

In 3-class case in 2-d space, that means we need the three lines to intersect at one point. We can use this to construct some counter-examples which are pairwise linearly separable but not linearly separable.

Lemma 2. *When $k = 2$, those three kinds of linearly separable are equivalent.*

Proof. Please prove it yourself as an exercise. □

Lemma 3. *Genarally, all-vs-one linearly separable ⇒ linearly separble ⇒ pairwise linearly separable.*

Proof. Please prove it yourself as an exercise. □

1.2.4 Logistic regression

Assume that we are given k linearly separable sets $A_1, A_2, \cdots, A_k \in \mathbb{R}^d$, we define the set of classifiable weights as

$$\boldsymbol{\Theta} = \{\boldsymbol{\theta} = (W, b) : w_i x + b_i > w_j x + b_j, \ \forall x \in A_i, j \neq i, i = 1, \cdots, k\} \tag{1.19}$$

which means those (W, b) can separate $A_1, A_2, \cdots, A_k$ absolutely correctly. And our assumption implies that $\boldsymbol{\Theta} \neq \emptyset$.
Now we know the existence of linearly classifiable weights. But how can we find one element in $\boldsymbol{\Theta}$?

Definition 5 (soft-max). *Given parameter $\boldsymbol{\theta} = (W, b)$, a soft-max mapping $\boldsymbol{p} : A \to \mathbb{R}^k$ is a mapping with the following fomulation*

$$\boldsymbol{p}(\boldsymbol{\theta}, x) = \frac{e^{Wx+b}}{e^{Wx+b} \cdot \mathbf{1}} = \frac{1}{\sum_{i=1}^{k} e^{w_i x + b_i}} \begin{pmatrix} e^{w_1 x + b_1} \\ e^{w_2 x + b_2} \\ \cdots \\ e^{w_k x + b_k} \end{pmatrix} \tag{1.20}$$

where $Wx + b = \begin{pmatrix} e^{w_1 x + b_1} \\ e^{w_2 x + b_2} \\ \cdots \\ e^{w_k x + b_k} \end{pmatrix}$, $\mathbf{1} = (1\ 1 \cdots 1)^T \in \mathbb{R}^k$, *and the i-th component* $p_i(\boldsymbol{\theta}, x) = (\sum_{i=1}^{k} e^{w_i x + b_i})^{-1} e^{w_i x + b_i}$.

The soft-max mapping have several important properties.

1. $0 < p_i(\boldsymbol{\theta}, x) < 1,\ \sum_i p_i(\boldsymbol{\theta}, x) = 1$.
   This implies that $\boldsymbol{p}(\boldsymbol{\theta}, x)$ can be regarded as a probability distribution of data points which means given $x \in \mathbb{R}^d$, we have $x \in A_i$ with probability $p_i(\boldsymbol{\theta}, x)$, $i = 1, \cdots, k$.
2. $p_i(\boldsymbol{\theta}, x) > p_j(\boldsymbol{\theta}, x) \Leftrightarrow w_i x + b_i > w_j x + b_j$.
   This implies that the linearly classifiable weights have an equivalent discription as

$$\boldsymbol{\Theta} = \{\boldsymbol{\theta} : p_i(\boldsymbol{\theta}, x) > p_j(\boldsymbol{\theta}, x),\ \forall x \in A_i, j \neq i, i = 1, \cdots, k\} \tag{1.21}$$

3. We usually use the max-out method to do classification. For a given data point x, we first use a soft-max mapping to map it to $\boldsymbol{p}(\boldsymbol{\theta}, x)$, then we attached x to the class $i = \arg\max_j p_i(\boldsymbol{\theta}, x)$.
   In probability, that means we pick the label i as the class of x such that $x \in A_i$ has the biggest probability $p_i(\boldsymbol{\theta}, x)$.

We denote the function

$$P(\boldsymbol{\theta}) = \prod_{i=1}^{k} \prod_{x \in A_i} p_i(\boldsymbol{\theta}, x) \tag{1.22}$$

which is called maximum likelihood function in Statistics. We'll show that can use this function to help us find some linearly classifiable weights.

Theorem 1. *Assume that the sets $A_1, A_2, \cdots, A_k$ are linearly separable. Then we have*

$$\{\theta :\ P(\boldsymbol{\theta}) > \frac{1}{2}\} \subset \boldsymbol{\Theta}. \tag{1.23}$$

Proof. This theorem means that if $P(\boldsymbol{\theta}) > \frac{1}{2}$, then we must have $\boldsymbol{\theta} \in \boldsymbol{\Theta}$. It is equivalent to the proposition that if $\boldsymbol{\theta} \notin \boldsymbol{\Theta}$, we must have $P(\boldsymbol{\theta}) \leq \frac{1}{2}$. So we only need to prove the second proposition.

For any $\boldsymbol{\theta} \notin \boldsymbol{\Theta}$. There must exist an i_0 ,an $x_0 \in A_{i_0}$ and a $j_0 \neq i_0$ such that

$$w_{i_0} x + b_{i_0} \leq w_{j_0} + b_{j_0}. \tag{1.24}$$

Then we have

$$p_{i_0}(\boldsymbol{\theta}, x) \leq \frac{e^{w_{i_0} x + b_{i_0}}}{e^{w_{i_0} x + b_{i_0}} + e^{w_{j_0} x + b_{j_0}}} \leq \frac{1}{2}. \tag{1.25}$$

Notice that $p_i(x, \theta) < 1$ for all $i = 1, \cdots, k$, $x \in D$.
So

$$P(\boldsymbol{\theta}) < p_{i_0}(\boldsymbol{\theta}, x) \leq \frac{1}{2}. \tag{1.26}$$

□

Lemma 4. *If $\boldsymbol{\theta} \in \boldsymbol{\Theta}$, we have*

$$\lim_{\alpha \to +\infty} p_i(\alpha\boldsymbol{\theta}) = 1 \Leftrightarrow x \in A_i. \tag{1.27}$$

Proof.
$\Leftarrow$. Easy to observe that for all $x \in A_i$,

$$\lim_{\alpha \to +\infty} p_i(\theta, x) = \lim_{\alpha \to +\infty} \frac{1}{1 + \sum\limits_{j \neq i} e^{\alpha[(w_j x + b_j) - (w_i x + b_i)]}} = 1. \tag{1.28}$$

$\Rightarrow$. Easy to observe that for all $x \notin A_i$,

$$p_i(\boldsymbol{\alpha\theta}, x) = \frac{1}{1 + \sum\limits_{j \neq i} e^{\alpha[(w_j x + b_j) - (w_i x + b_i)]}} \leq \frac{1}{2}. \tag{1.29}$$

This implies that if $x \notin A_i$, $\lim_{\alpha \to \infty} p_i(\alpha\boldsymbol{\theta}) \neq 1$ which is equivalent to the proposition that if $\lim_{\alpha \to \infty} p_i(\alpha\boldsymbol{\theta}) = 1$, then $x \in A_i$. □

Theorem 2.

$$\boldsymbol{\Theta} = \{\boldsymbol{\theta} : \lim_{\alpha \to +\infty} P(\alpha\boldsymbol{\theta}) = 1\}. \tag{1.30}$$

Proof.
$\Rightarrow$. If $\boldsymbol{\theta} \in \boldsymbol{\Theta}$, we have $\lim_{\alpha \to +\infty} p_i(\alpha\boldsymbol{\theta}) = 1$ for all $x \in A_i$. So

$$\lim_{\alpha \to +\infty} P(\alpha\boldsymbol{\theta}) = \lim_{\alpha \to +\infty} \prod_{i=1}^{k} \prod_{x \in A_i} p_i(\boldsymbol{\theta}, x) = \prod_{i=1}^{k} \prod_{x \in A_i} \lim_{\alpha \to +\infty} p_i(\boldsymbol{\theta}, x) = 1. \tag{1.31}$$

$\Leftarrow$. If $\lim\limits_{\alpha \to +\infty} P(\alpha\boldsymbol{\theta}) = 1$, there must exist one $\alpha_0 > 0$ such that $P(\alpha_0\boldsymbol{\theta}) > \frac{1}{2}$. From Theorem 1, we have $\alpha_0\boldsymbol{\theta} \in \boldsymbol{\Theta}$, which means $\boldsymbol{\theta} \in \boldsymbol{\Theta}$. □

1.2.5 Introduction to Logistic regression

Theorem 1 implies that if we can obtain a classifiable weight through maximizing $P(\boldsymbol{\theta})$, while theorem 2 tells us that $P(\boldsymbol{\theta})$ won't have a minimum actually.

The design of logistic regression is that maximize $P(\boldsymbol{\theta})$ is equivalent to miniimize $\log P(\boldsymbol{\theta})$ which the second one is more convenient to evaluate the gradient. Meanwhile, we add a regularization term $R(\boldsymbol{\theta})$ to the objective function which makes the optimization problem has a unique solution.

Mathematically, we can formulate Logistic regression as

$$\min_{\boldsymbol{\theta}} -\log P(\boldsymbol{\theta}) + \lambda R(\boldsymbol{\theta}), \tag{1.32}$$

where $R(\boldsymbol{\theta})$ usually equals to $\|\boldsymbol{\theta}\|_F^2 = \sum_{i,j} \theta_{ij}^2$.

Lemma 5. *$-logP(\boldsymbol{\theta})$ is a convex function.*

Lemma 6. *If $A_1, A_2, \cdots, A_k$ are linearly separable, $-\log P(\boldsymbol{\theta})$ has no global minimum.*

Theorem 3. *$-\log P(\boldsymbol{\theta}) + \lambda R(\boldsymbol{\theta})$ has a global minimizer for sufficiently small $\lambda > 0$.*

1.3 Notes for 3rd day of the first week

1.3.1 Logistic Regression

Recall that the definition of *linear separable sets*.

Definition 6. *A collection of subsets $A_1, \dots, A_k \subset \mathbb{R}^d$ are linearly separable if there exists*

$$W = \begin{pmatrix} w_1 \\ \vdots \\ w_k \end{pmatrix} \in \mathbb{R}^{k\times d}, b = \begin{pmatrix} b_1 \\ \vdots \\ b_k \end{pmatrix} \in \mathbb{R}^{k\times 1} \tag{1.33}$$

such that for given $1 \le i \le k$ and each $j \ne i$

$$(Wx + b)_i > (Wx + b)_j, \forall x \in A_i \tag{1.34}$$

i.e.,

$$i = \arg\max_j (Wx + b)_j, \forall x \in A_i \tag{1.35}$$

The probability density function of the distribution over the k categories should satisfy

$$0 < p_i(\theta, x) < 1 \quad \text{and} \quad \sum_{i=1}^{k} p_i(\theta, x) = 1. \tag{1.36}$$

Here, $p_i(\theta, x)$ is the probability of x in the i-th category, i.e., $x \in A_i$. Then, we introduce a popular choice of the probability density function in classification problems.

Definition 7. *The standard softmax function $\sigma : \mathbb{R}^k \to \mathbb{R}^k$ is defined by the formula*

$$(\sigma(z))_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \tag{1.37}$$

for $i = 1, \dots, k$ and $z = (z_1, \dots, z_k)^\top \in \mathbb{R}^k$

Let $z = Wx + b$, we obtain

$$p(\theta, x) = \frac{e^{Wx+b}}{e^{Wx+b} \cdot \mathbf{1}} = \frac{1}{e^{Wx+b} \cdot \mathbf{1}} \begin{pmatrix} e^{w_1 x+b_1} \\ \vdots \\ e^{w_k x+b_k} \end{pmatrix} = \begin{pmatrix} p_1(\theta, x) \\ \vdots \\ p_k(\theta, x) \end{pmatrix} \tag{1.38}$$

where $e^z = (e^{z_1}, \dots, e^{z_k})^\top$ for any $z \in \mathbb{R}^k$ and $\mathbf{1} = (1, \dots, 1)^\top$ with appropriate dimension.

Note that $A_1, \ldots, A_k$ are linearly separable if and only if $p_i(\theta, x) > p_j(\theta, x)$ for any $x \in A_i$, $i \neq j$ since the exponential function is monotone increasing. Hence logistic regression will do a "prefect" job.

Denote the set of classifiable weights

$$\Theta = \left\{\theta = (W, b) : p_i(\theta, x) > p_j(\theta, x), x \in A_i, i \neq j\right\}.$$

We proved the following results:

1. $\theta \in \Theta$ if and only if $\lim_{\alpha\to\infty} p(\alpha\theta) = 1$. We note that $0 < p_i(\theta) < 1$ for any θ, so $p(\theta)$ has no global maximum.
2. $\left\{\theta : p(\theta) \geq \frac{1}{2}\right\} \subset \Theta$.

In previous section, we already show that we can construct a function with global maximum by adding a regularization term.

Lemma 7. *Assume that $R(\theta) \geq 0$ is an non-negative function such that*

$$\Theta^*(\lambda) = \operatorname{argmax} p(\theta)e^{-\lambda R(\theta)} \neq \emptyset, \quad \forall \lambda > 0 \tag{1.39}$$

then for sufficiently small $\lambda > 0$,

$$\Theta^*(\lambda) \subset \left\{\theta : p(\theta) \geq \frac{1}{2}\right\} \subset \Theta.$$

Proof. Given $\theta_0 \in \Theta$. Let $\alpha > 0$ be such that

$$p(\alpha\theta_0) \geq \frac{2}{3}$$

Let $\lambda > 0$ be sufficiently small such that

$$e^{-\lambda R(\theta)} > \frac{3}{4}$$

For any $\theta \in \Theta^*$

$$p(\theta)e^{-\lambda R(\theta)} \geq p(\alpha\theta_0)e^{-\lambda R(\alpha\theta_0)} > \frac{2}{3} \cdot \frac{3}{4} = \frac{1}{2}$$

Then $p(\theta) > \frac{1}{2}$. □

Since $-\log(\cdot)$ is strictly decreasing, we can transform the maximal problem to a minimal problem

$$\theta^* = \operatorname{argmin}\left\{-\log p(\theta)e^{-\lambda R(\theta)}\right\}$$

Recall that

$$\operatorname{argmin} g(\theta) = \{\theta^* : g(\theta^*) = \min g(\theta)\}, \tag{1.40}$$

that is, the set of all minimizers. And note that

$$\operatorname{argmin}\{-\log p(\theta)\} \neq \emptyset$$

The definition of **Likelihood function**

$$p(\theta) = \Pi_{i=1}^{k} \Pi_{x \in A_i} p_i(\theta, x).$$

and **logarithmic likelihood function**

$$\begin{aligned} -\log p(\theta) &= -\log\left(\Pi_{i=1}^{k} \Pi_{x \in A_i} p_i(\theta, x)\right) \\ &= \sum_{i=1}^{k} \sum_{x \in A_i} (-\log p_i(\theta, x)) \\ &= \sum_{i=1}^{k} \sum_{x \in A_i} \left(\log(e^{(Wx+b)\cdot \mathbf{1}}) - (w_i x + b_i)\right) \end{aligned} \tag{1.41}$$

Now we consider the "training set" containing all **labelled** data:

$$\cup_{i=1}^{k} A_i = \{x_j \in \mathbb{R}^d,\ 1 \leq j \leq N\} \tag{1.42}$$

For $x_j \in A_i$, y_j is the corresponding label

$$y_j = e_i \tag{1.43}$$

where e_i is the k dimensional vector with the ith component being 1 and the others being 0. In the classical cat-dog-rabbit problem,

$$\begin{aligned} \text{cat} &\leftarrow (1, 0, 0)^\top \\ \text{dog} &\leftarrow (0, 1, 0)^\top \\ \text{rabbit} &\leftarrow (0, 0, 1)^\top \end{aligned} \tag{1.44}$$

For $x_j \in A_i$, the label $y_j = e_i$, then $w_i x_j + b_i = (W x_j + b) \cdot y_j$, then the logarithmic likelihood function (1.41) can be written as

$$-\log p(\theta) = \sum_{j=1}^{N} \log(e^{(Wx_j+b)\cdot \mathbf{1}} - (W x_j + b) \cdot y_j, \tag{1.45}$$

which is so-called **cross-entropy loss function**.

Logistic regression is to find

$$\operatorname{argmin}\ \left\{L(\theta, \lambda) = -\log\left(p(\theta) e^{-\lambda R(\theta)}\right)\right\} \tag{1.46}$$

and

$$L(\theta) = L(\theta, \lambda) = \sum_{j=1}^{m} l(\theta, x_j) + \lambda R(\theta) \tag{1.47}$$

where $l(\theta, x) = \log(e^{Wx+b} \cdot \mathbf{1}) - (Wx + b) \cdot y(x)$

1.3.2 Training Algorithm: Gradient Descent Based Methods

We will introduce **stochasic gradient descent method** in next part. Let us start with a calculus question. *What is the fastest descent direction of a function?*

A function $f : \mathbb{R}^n \to \mathbb{R}^1$. For $x \in \mathbb{R}^n$

$$f(x) = f(x_1, \ldots, x_n)$$

The partial derivative w.r.t. x_1

$$\frac{\partial f}{\partial x_1} = \lim_{h \to 0} \frac{f(x_1 + h, x_2, \ldots, x_n) - f(x_1, x_2, \ldots, x_n)}{h} = \lim_{h \to 0} \frac{f(x + he_1) - f(x)}{h}$$

so the partial derivative w.r.t. x_i

$$\frac{\partial f}{\partial x_1} = \lim_{h \to 0} \frac{f(x + he_1) - f(x)}{h}$$

The gradient of f w.r.t x

$$\nabla f = (\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n})^\top.$$

Consider $f(x + tl)$ where $t \in \mathbb{R}^1$ and $l \in \mathbb{R}^n$ with $\|l\|_2 = 1$.

$$\frac{\partial}{\partial t} f(x + tl)\Big|_{t=0} \begin{cases} > 0 \ f \text{ ascends along direction } l \\ < 0 \ f \text{ descends along direction } l \end{cases} \tag{1.48}$$

By chain rule, the derivative

$$\frac{\partial}{\partial t} f(x + tl) = \sum_{i=1}^{n} \frac{\partial f}{\partial x_i} \cdot l_i.$$

The fastest descend direction is $l = -\frac{\nabla f}{\|\nabla f\|}$. (Hint for proof: Cauchy-Schwarz inequality)

Consider an optimization problem: find min $f(x)$. We solve it by iterative method. Starting from initial point x^0. How to get x^{m+1} from current point x^m?

Gradient descent method (steepest descend)

$$x^{m+1} = x^m - \eta_m \nabla f(x^m), \quad m = 0, 1, 2, \ldots \tag{1.49}$$

Here η_m is called learning rate.

A machine learning problem

$$f(x) = \frac{1}{N} \sum_{j=1}^{N} f_j(x)$$

where N is too large (e.g. 10^6) to compute a full gradient.

Stochastic gradient descent (SGD) method. Pick $j_m \in \{1, 2, \ldots, N\}$*randomly*, and

$$x^{m+1} = x^m - \eta_m \nabla f_{j_m}(x^m).$$

Mini-batch method. Pick a index set $\{i_1, i_2, \ldots, i_k\} \subset \{1, 2, \ldots, N\}$, and

$$x^{m+1} = x^m - \eta_m \tilde{\nabla} f(x^m)$$

where $\tilde{\nabla} f(x^m) = \frac{1}{k} \sum_{j=1}^{k} \nabla f_{i_j}(x^m)$.

1.4 Notes for 4th day of the first week

For Logistic regression, let us consider the data set as

$$D = \{x_j, y_j\}, \quad j = 1 : N.$$

Here we assume that there are k-class:

$$\{x_j : j = 1 : N\} = \cup_{i=1}^{k} A_k,$$

where

$$x_j \in A_j \Rightarrow y_j = y(x_j) = e_i.$$

Here y_j is called the label of data x_j.

The final problem for a machine learning model is:

$$\min_{\theta} L(\theta),$$

with data set can be split as **training data**, **validation data** and **test data**. The test accuracy is the criterion for the performance of the machine learning model and algorithm.

1.4.1 Gradient Descent (GD) Method

Let us consider the objective (loss) function as:

$$L(\theta),$$

if we apply some iterative methods, we will have

$$\theta^0, \theta^1, \cdots, \theta^m, \theta^{m+1}, \cdots.$$

We know that $L(\theta^m + \alpha \boldsymbol{l})$ descent most rapidly along $-\nabla L(\theta^m)$, i.e.

$$\theta^{m+1} = \theta^m - \eta_m \nabla L(\theta^m),$$

where η_m is called the step size or learning rate.

1.4.2 Understand GD from the Viewpoint of Dynamic System

Generally speaking, we hope to find the stationary point for the loss function, i.e.

$$\nabla L(\theta^*) = 0. \tag{1.50}$$

Consider a dynamical problem:

$$\dot{\theta}(t) = -\nabla L(\theta), \tag{1.51}$$

which is a simple ordinary differential equation (ODE). Then the optimality condition (1.50) is understood as the steady state solution of the above ODE, i.e.

$$\theta^* = \lim_{t\to\infty} \theta(t).$$

That is to say,

$$0 = \dot{\theta}^* = -\nabla L(\theta^*),$$

thus we have

$$\nabla L(\theta^*) = 0,$$

which is the optimality condition in (1.50).

Numerical ODE

The fundamental numerical scheme for solve the above ODE (1.51) is the forward Euler scheme. In that scheme, we approximate the time derivative with difference:

$$\dot{\theta}(t^m) \approx \frac{\theta(t^m + \eta) - \theta(t^m)}{\eta}, \quad (\eta \to 0). \tag{1.52}$$

Take this into the above ODE (1.51), we have

$$\frac{\theta(t^m + \eta) - \theta(t^m)}{\eta} \approx -\nabla L(\theta(t^m)).$$

This leads to the same scheme of gradient descent:

$$\theta^{m+1} = \theta^m - \eta_m \nabla L(\theta^m).$$

1.4.3 Stochastic Gradient Descent (SGD)

Let us recall that the loss function of a general machine learning problem:

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(\theta; x_i, y_i),$$

for example:

$$\ell(\theta; x, y) = \log(e^{wx+b} \cdot \mathbf{1}) - (wx + b) \cdot y + \lambda R(\theta).$$

From the consideration of computation cost, we can compute the gradient with some approximation scheme $\tilde{\nabla} L(\theta)$. So, the final scheme can be:

$$\theta^{m+1} = \theta^m - \eta_m \tilde{\nabla} L(\theta). \tag{1.53}$$

Generally speaking, we have the next two forms of SGD:

1. SGD with replacement.
   Randomly pick $B_m \subset D$,
   $$B_m = \{x_{i_1}, \cdots, x_{i_s}\},$$
   we can compute the approximated gradient as
   $$\tilde{\nabla} L(\theta^m) = \frac{1}{s} \sum_{x \in B_m} \nabla \ell(\theta^m; x).$$
   Then we do the SGD as (1.53).
2. SGD without replacement (shuffle SGD).
   In each epoch, we shuffle the data set D first, and then split this set into $\frac{m}{s}$ mini-batch in order, i.e.
   $$D = \cup_{l=1}^{\frac{N}{s}} B^l.$$
   Then the SGD in this epoch is applied as:
   $$\theta \leftarrow \theta - \eta \frac{1}{s} \sum_{x \in B^l} \nabla L(\theta; x), \quad \text{for} \quad l = 1, 2, \cdots, \frac{N}{s}.$$

Remark 1. Let us make some brief statements about SGD method.

1. SGD is not "consistent".
   The goal of a general optimization algorithm is to find the next stationary point,
   $$\theta^m \to \theta^* = \arg\min L(\theta),$$
   where we also have
   $$\nabla L(\theta^*) = 0.$$
   Thus, if we have $\theta^m = \theta^*$, we call the "consistence" as that we can have $\theta^{m+1} = \theta*$.
   - Full GD.
     If $\theta^m = \theta^*$, we have $\nabla L(\theta^m) = \nabla L(\theta^*) = 0$. Thus, we have
     $$\theta^{m+1} = \theta^m - \eta_m \nabla L(\theta^m) = \theta^*.$$
   - SGD. If $\theta^m = \theta^*$, we have $\tilde{\nabla} L(\theta^m) = \tilde{\nabla} L(\theta^*) \neq 0$. Thus, if we want
     $$\theta^{m+1} = \theta^m - \eta_m \tilde{\nabla} L(\theta^m) \to \theta^*,$$
     we need to make $\eta_m \to 0$.
2. The size of the mini-batch $s = |B|$.
   a) $s = N$, this is the full GD which is not good because of its poor generalization accuracy.
   b) $s = 1$, this will cause x_i-outlier which means the noise or incorrect data may cause bad effect.
   c) s should be chosen with an appropriate size.

1.4.4 Non-linearly Separable

If $D \subset \mathbb{R}^d$ is not linearly separable, but "separable" namely, there exists a continuous function:

$$\phi : \mathbb{R}^d \mapsto \mathbb{R}^n,$$

such that

$$\{\tilde{A}_i = \phi(A_i) : i = 1 : k\},$$

are linearly separable. Here ϕ is often called feature mapping.

For general assumptions, we assume that

$$D \subset \Omega,$$

which is a bounded set and note $\bar{\Omega}$ as the closure of Ω. For simplicity, we can also assume that $\Omega = (0, 1)^d$.

Let us consider ϕ is continuous on $\bar{\Omega}$ ($\phi \in C(\bar{\Omega})$ = set of all continuous functions on $\bar{\Omega}$). Now, there is an natural question that

Is there a "good" function space V such that for any $\phi \in C(\bar{\Omega})$ we have

$$\lim_{n\to\infty} \|\phi - \phi_n\|_{C(\bar{\Omega})} = 0,$$

for $\phi_n \in V$.

Here

$$\|\phi - \phi_n\|_{C(\bar{\Omega})} = \max_{x\in\bar{\Omega}} |\phi(x) - \phi_n(x)|.$$

The first example of V is

$$V_{\text{poly}} = \{\text{all polynomials on } \bar{\Omega} \text{ in } \mathbb{R}^d\}.$$

Theorem 4 (Weierstrass-Stone Theorem). *For any $\phi \in C(\bar{\Omega})$, there exists a sequence of polynomials $p_n \in V_{\text{poly}}$ such that*

$$\lim_{n\to\infty} \|\phi - p_n\|_{C(\bar{\Omega})} = 0.$$

That is to say

$$\bar{V}_{\text{poly}} = C(\bar{\Omega}).$$

However, we can see that for any polynomial $p_n(x)$ of degree n for $x \in \mathbb{R}^d$ the number of coefficient of p_n is

$$\binom{n+d}{n} = C^n_{d+n},$$

which is huge if $d \gg 1$. This is also known as the curse of dimensionality.

1.4.5 Deep Neural Network (DNN) Function

From now on, we are going to talk about models of deep learning. The first model that we will introduce is the deep neural network (which is also called forward neural network or fully connected neural network).

Actually, there are only two main components of DNN models:

1. Linear mapping:

$$Wx + b : \mathbb{R}^d \mapsto \mathbb{R}^{n_0},$$

   as $W \in \mathbb{R}^{n_0 \times d}$ and $b \in \mathbb{R}^{n_0}$.
2. Activation function:

$$\sigma : \mathbb{R}^1 \mapsto \mathbb{R}^1,$$

   and $\sigma(x) = (\sigma(x_1), \cdots, \sigma(x_d))^\top$.

Thus, we can construct a function from $\mathbb{R}^d$ to $\mathbb{R}^n$ as:

$$\phi(x) = W^4\sigma\left\{W^3\sigma\left\{W^2\sigma\left\{W^1\sigma\left\{W^0x + b^0\right\} + b^1\right\} + b^2\right\} + b^3\right\} + b^4.$$

This is called a 4-th layer DNN model.

1.5 Notes for 5th day of the first week

1.5.1 Recall for Last Lecture

Feature Mapping

There exists a continuous function:

$$\phi : \mathbb{R}^d \mapsto \mathbb{R}^n,$$

such that

$$\{\tilde{A}_i = \phi(A_i) : i = 1 : k\},$$

are linearly separable. Here ϕ is often called feature mapping.

Weierstrass-Stone Theorem

That is to say, what we need is to approximate this continuous feature mapping ϕ. First of all, let us recall the Weierstrass-Stone theorem.

Theorem 5 (Weierstrass-Stone Theorem). *For any $\phi \in C(\bar{\Omega})$ (i.e. ϕ is continuous on a compact (bounded and closed) set in $\mathbb{R}^d$), there exists a sequence of polynomials $p_n \in V_{\text{poly}}$ such that*

$$\lim_{n\to\infty} \|\phi - p_n\|_{C(\bar{\Omega})} = 0.$$

That is to say

$$\bar{V}_{\text{poly}} = C(\bar{\Omega}).$$

There is an constructive proof by the Bernstein polynomial for any $f(x) \in C([0,1])$:

$$B_n(f) = \sum_{k=0}^{n} f(\frac{k}{n})\binom{n}{k}x^k(1-x)^{n-k} \to f(x), \quad (n \to \infty).$$

Homework 6 *Take some continuous functions $f \in C([0,1])$, then plot f and $B_n(f)$ for some n.*

1.5.2 Deep Neural Network Functions

Just as what we mentioned before, there are two main components of DNN models:

1. Linear mapping:

$$Wx + b : \mathbb{R}^d \mapsto \mathbb{R}^{d_0},$$

   as $W \in \mathbb{R}^{d_0 \times d}$ and $b \in \mathbb{R}^{d_0}$.
2. Activation function:

$$\sigma : \mathbb{R}^1 \mapsto \mathbb{R}^1,$$

   and $\sigma(x) = (\sigma(x_1), \cdots, \sigma(x_d))^\top$.

Then we can define a DNN function as:

1. $$W^0 x + b^0 \in \mathbb{R}^{d_0},$$
2. $$x^1 = \sigma(W^0 x + b^0) \in \mathbb{R}^{d_0},$$
3. $$W^1 x^1 + b^1 \in \mathbb{R}^{d_1},$$
4. $$x^2 = \sigma(W^1 x^1 + b^1) \in \mathbb{R}^{d_1},$$
5. $$\dots$$
6. $$\phi(x) = W^L x^L + b^L \in \mathbb{R}^{d_L} (= \mathbb{R}^n).$$

We call this as a L-th layer DNN function.

1.5.3 Activation Functions

- An general activation function(must be nonlinear) is
$$\sigma : \mathbb{R} \to \mathbb{R}.$$
- The Heaviside function is
$$H(x) = \begin{cases} 0 & \text{if } x \le 0, \\ 1 & \text{if } x > 0. \end{cases}$$
The biggest problem for this activation function is that this function is not continuous which will cause huge difficult in training phase.
- The sigmoid function:
$$s(x) = \frac{1}{1 + e^{-x}} \to \begin{cases} 0, & x \to -\infty, \\ 1, & x \to +\infty. \end{cases} .$$
This function can be seen as the smooth approximation of Heaviside function. This activation function was very popular in shallow neural network in about 1990s. Now, this activation function is also often used in RNN or some NLP tasks.
- Currently, the most used activation function in DNN and CNN is "Rectified Linear Unit" (ReLU):
$$\mathrm{ReLU}(x) = \max(0, x).$$
There are many interesting properties of ReLU function:

1. ReLU is a piecewise linear function. Thus, DNN with this activation function is always a piecewise linear function.
2. The connection of ReLU and Heaviside.

$$\frac{d}{dx}\mathrm{ReLU}(x) = H(x). \tag{1.54}$$

3. Recently, there are huge research works about the approximation properties of DNN with ReLU activation function see [1, 2, 3].

Here is a simple diagram for a general DNN structure:

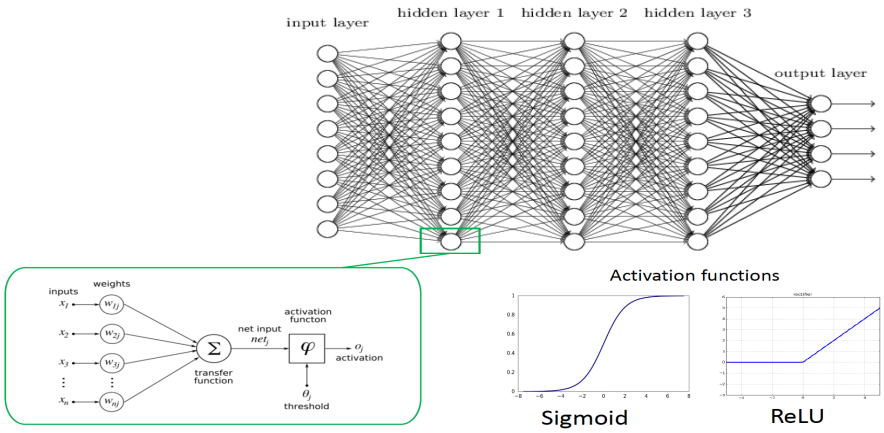


Fig. 1.4. A General Structure of DNN

1.5.4 Special Case: 1-hidden Layer

First, let us define the so-called 1-hidden layer (shallow) neural network.

Definition 8. *The 1-hidden layer (shallow) neural network is defined as:*

$$\mathrm{DNN}_1 = \{\phi \ : \ \phi(x) = \sum_{i=1}^{N} a_i\sigma(w_i x + b_i) + c, \quad N \in \mathbb{N}^+\}.$$

To consistent with above notation, we can write it as:

$$\phi = W^1 x^1 + b^1 = (a_1, \cdots, a_N)\sigma(W^0 x + b^0) + c \in \mathrm{DNN}_1.$$

The first question about DNN_1 is about the approximation properties for any continuous functions. Here we have the next theorem:

Theorem 7 (Universal Approximation Property of Shallow Neural Networks). *Let Ω be bounded, if any $f \in C(\bar{\Omega})$, there exists a sequence $\phi_n \in \mathrm{DNN}_1$ such that*

$$\max_{x\in\bar{\Omega}} |\phi_n(x) - f(x)| \to 0, \quad n \to \infty.$$

This provide that σ is not a polynomial. On the other hand, let σ be a non-polynomial Riemann integrable function and $\sigma \in L^\infty_{loc}(\mathbb{R})$ then we have

$$\overline{\mathrm{DNN}}_1 = C(\bar{\Omega}).$$

Before the proof, let us give some notation. We use $\mathbb{P}_m(\mathbb{R}^d)$ to define the polynomials of d-variables of degree less than m. Let $\alpha = (\alpha_1, \cdots, \alpha_d)$(α_i non-negative integers), we note $|\alpha| = \sum_{i=1}^d \alpha_i$ and

$$x^\alpha = x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_d^{\alpha_d}.$$

Lemma 8. *Let $\sigma \in C^\infty(\Omega)$ (i.e. σ is infinitely differentiable) and is not a polynomial, then for any $k \geq 0$ there exists $t_k \in \mathbb{R}$ such that*

$$\sigma^{(k)}(t_k) \neq 0.$$

Now we are going to give the proof of

Proof. If σ is a polynomial, say $\sigma \in \mathbb{P}_m(\mathbb{R})$, then we have that

$$\mathrm{DNN}_1 \subset \mathbb{P}_m(\mathbb{R}^d).$$

Thus, DNN_1 cannot approximate polynomial of degree bigger than $m+1$. This implies that σ cannot be polynomial if DNN_1 has the approximation property.

Now we prove that $\overline{\mathrm{DNN}}_1 = C(\bar{\Omega})$ if σ is not a polynomials.

Case 1 First, let us assume that $\sigma \in C^\infty(\Omega)$.

Fact 1: We have the next relation:

$$\frac{\partial}{\partial[w]_i}(\sigma(wx+b))\,|_{w=0} = \sigma'(wx+b)\frac{\partial}{\partial[w]_i}(wx+b) = \sigma'(wx+b)x_i|_{w=0}.$$

That is to say,

$$\frac{\partial}{\partial[w]_i}(\sigma(wx+b))\,|_{w=0} = \sigma'(b)x_i.$$

By the lemma 8, there exists a $b \in \mathbb{R}$ such that

$$\sigma'(b) \neq 0.$$

Fact 2: By using the definition of derivative, we have

$$\frac{\partial}{\partial[w]_i}(\sigma(wx+b))\,|_{w=0} = \lim_{n\to\infty} \frac{\sigma((0+\frac{1}{n}e_i)\cdot x + b) - \sigma(b)}{\frac{1}{n}} = \lim_{n\to\infty} \phi_n(x),$$

where

$$\phi_n(x) = n\left(\sigma(\frac{1}{n}e_i \cdot x + b) - \sigma(b)\right) \in \mathrm{DNN}_1.$$

This leads to the result that

$$\sigma'(b)x_i = \lim_{n\to\infty} \phi_n(x) \in \overline{\mathrm{DNN}}_1,$$

because of the definition that

$$\overline{\mathrm{DNN}}_1 = \mathrm{DNN}_1 \cup \{f \ : \ f = \lim_{n\to\infty} \phi_n(x), \phi_n \in \mathrm{DNN}_1\}.$$

Follow there facts, we know that

$$\sigma'(b)x_i \in \mathrm{DNN}_1,$$

this leads to

$$x_i \in \overline{\mathrm{DNN}}_1,$$

because $[\sigma'(b)]^{-1}\phi_n(x) \to x_i$.

Thus we have

$$\frac{\partial^2}{\partial[w]_1\partial[w]_2}\left(\sigma(wx+b)\right)|_{w=0} = \sigma^{(2)}(b)x_1x_2.$$

Using the Lemma 8 again, there exists a $b \in \mathbb{R}$ such that

$$\sigma^{(2)}(b) \neq 0.$$

This leads to

$$x_1x_2 \in \overline{\mathrm{DNN}}_1.$$

Similarly, we can prove that

$$x_1^{\alpha_1}x_2^{\alpha_2}\cdots x_d^{\alpha_d} \in \overline{\mathrm{DNN}}_1.$$

This proves that DNN_1 can approximate any polynomials. Combine with the Weierstrass theorem, DNN_1 can approximate any continuous functions.

Then we will finish the proof for any σ as a non-polynomial Riemann integrable function and $\sigma \in L^\infty_{loc}(\mathbb{R})$.

□

1.6 Notes for 6th day of the first week

1.6.1 Recall for Last Lecture

Deep Neural Network

An example of DNN functions can be written as

$$\phi(x,\theta) = W^3\sigma(W^2\sigma(W^1\sigma(W^0x + b^0) + b^1) + b^2) + b^3$$

$$\theta = (W^0, b^0, W^1, b^1, W^2, b^2, W^3, b^3)$$

The DNN functions with one hidden layer is defined as

$$\mathrm{DNN}_1(\sigma) = \{\phi : \phi = \sum_{i=1}^{N} a_i\sigma(w_ix + b_i)\}$$

Theorem 8. $\mathrm{DNN}_1(\sigma)$ *is dense in* $C([0,1]^d) \Leftrightarrow \sigma$ *is NOT a polynomial.*

We need to proof:
If σ is not a polynomial. For any continuious function $f \in C(\bar{\Omega})$, there exists a sequence $\{\phi_n\} \subset \mathrm{DNN}_1$, such that:

$$\lim_{n\to\infty} \max_{x\in\bar{\Omega}} |\phi_n(x) - f(x)| = 0$$

Idea:

case1 $\sigma \in C^\infty(\mathbb{R})$.
case2 σ is Riemann integrable.

Properties(Homework)

1. It reproduces identity map. (i.e. $x \in \mathrm{DNN}_1(\sigma)$)
2. $\mathrm{DNN}_l(\sigma) \subset \mathrm{DNN}_L(\sigma)$, for $l \le L$
3. Functions in $\mathrm{DNN}_l(\sigma)$ are continuous and piecewise linear.
4. Any continuous piecewise linear function can be written as a DNN function.
   For $d = 2$, one may plot some examples, for $\mathbb{R}^d = \bigcup \bar{D}_i$, D_i is polyhedron and ϕ is linear on each D_i.

1.6.2 Application to image classification

Given $\{A_i\}_{i=1}^k \subset \mathbb{R}^d$, We look for a feature map $\phi \in \mathrm{DNN}_l$, with the coefficient θ, such that $\tilde{A}_i = \{\phi(A_i, \theta)\}$ are linear separable.

DNN + Logistic Regression $\longrightarrow$ Nonlinear models(non-polynomial) $\xrightarrow{\text{"training"}}$ Non-convex optimization problem.

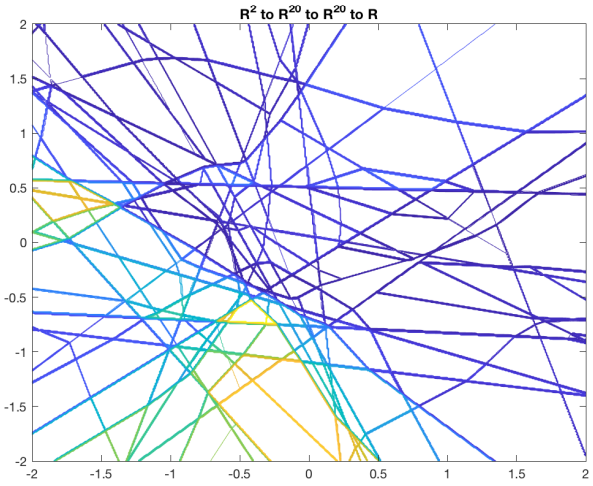


Fig. 1.5. A piecewise linear function obtained from DNN_2

1.6.3 Image Classification

In general, we have two kinds of images, gray image and color image.
Gray image: $g : [0,1]^2 \to \mathbb{R}^1$, $0 \le g \le 255$
Color image(R.G.B.): $g : [0,1]^2 \to \mathbb{R}^3$, $0 \le g \le 255$
After discretizing the gray picture, we get a 2-d discrete function:

$$\{g_{i,j} : 1 \le i \le m, 1 \le j \le m\}$$

and for color image, we we are going to get three of these functions.

Image as a matrix or tensor

The tensor transformed from image has three dimension (m, n, c). c is called channel.

$$c = \begin{cases} 1, & \text{gray image: } g = (g_{i,j}) \subset \mathbb{R}^{m\times n} \\ 3, & \text{colored image: } g = (g_{i,j,k}) \subset \mathbb{R}^{m\times n\times c} \end{cases} \tag{1.55}$$

(1.56)

Edges and Convolution

We are supposed to find a linear mapping $\theta(g) = Wg + b$ which can distinguish edges from images, and here is an simple example:

1. $[\theta(g)]_{i,j} = g_{i,j+1} - g_{i,j} \forall i, j$
2.

$$[\theta(g)]_{i,j} = g_{i+1,j} - g_{i,j} = \begin{cases} 0, & (i,j) \text{ is away from the bar} \\ \pm 1, & (i,j) \text{ is at or near the bar} \end{cases} \tag{1.57}$$

(1.58)

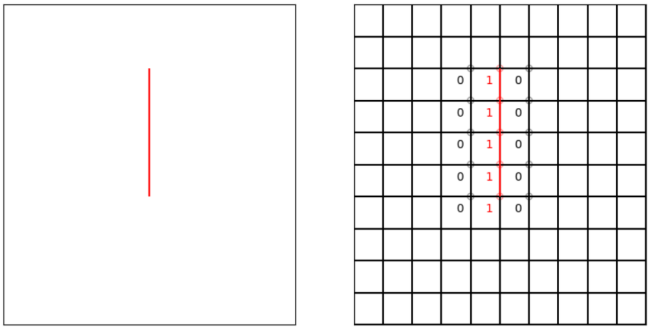


Fig. 1.6. The Edge

(a) This is a linear mapping.
(b) This is not fully connected.
(c) This will use specail convolution filter.

Let see an example for a 3×3 convolution.
kernel

$$K = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix},$$

then

$$(K * g)_{i,j} = 4g_{i,j} - g_{i-1,j} - g_{i+1,j} - g_{i,j-1} - g_{i,j+1}.$$

Here is a simple example about the index.

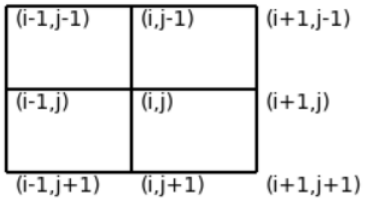


Fig. 1.7. The subscript

Generally we have, if

$$K = \begin{pmatrix} k_{-1,-1} & k_{0,-1} & k_{1,-1} \\ k_{-1,0} & k_{0,0} & k_{1,0} \\ k_{-1,1} & k_{0,1} & k_{1,1} \end{pmatrix},$$

then we have

$$(K * g)_{i,j} = \sum_{-1 \le s,t \le 1} g_{i+s,j+t} k_{s,t}.$$

Remark 2. If (i, j) is on the boundary, we need to "padding", which means to extend g outside of the image. There are three common methods:
(1) zero padding (2) periodic padding (3) reflection padding

You are supposed to implement a CNN algorithm.

$$\begin{cases} g^0 & = \theta^0 * g, \\ \text{for} & i = 1 : n, \\ & g^i = \theta^i \circ \sigma \circ g^{i+1}. \end{cases}$$

2

Supplemental Material: Convolution Filters

In this chapter, we will give a brief description how convolution operations are used for image processing.

One useful description can be found in the following link:
http://aishack.in/tutorials/image-convolution-examples/

2.1 Examples of convolution

Convolutions is a technique for general signal processing. People studying electrical/electronics will tell you the near infinite sleepless nights these convolutions have given them. Entire books have been written on this topic. And the questions and theorems that need to be proved are [insurmountable]. But for computer vision, we'll just deal with some simple things.

A convolution lets you do many things, like calculate derivatives, detect edges, apply blurs, etc. A very wide variety of things. And all of this is done with a "convolution kernel".

2.2 Calculation with convolutions

The most direct way to compute a convolution would be to use multiple for loops. But that causes a lot of repeated calculations. And as the size of the image and kernel increases, the time to compute the convolution increases too (quite drastically).

Techniques haves been developed to calculate convolutions rapidly. One such technique is using the Discrete Fourier Transform. It converts the entire convolution operation into a simple multiplication. Fortunately, you don't need to know the math to do this in OpenCV. It automatically decides whether to do it in frequency domain (after the DFT) or not.

2.3 Image convolution examples

A convolution is very useful for signal processing in general. There is a lot of complex mathematical theory available for convolutions. For digital image processing, you don't have to understand all of that. You can use a simple matrix as an image convolution kernel and do some interesting things!

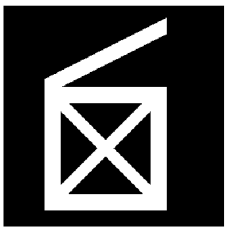
2.3.1 Simple box blur

[1] Here's a first and simplest. This convolution kernel has an averaging effect. So you end up with a slight blur. The image convolution kernel is:

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Note that the sum of all elements of this matrix is 1.0. This is important. If the sum is not exactly one, the resultant image will be brighter or darker.

Here's a blur that I got on an image:



(a) image

$$\begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix}$$

(b) filter

(c) result

[1] The following examples are from the website, http://aishack.in/tutorials/image-convolution-examples/

(d) image

$$\begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix}$$

(e) filter

(f) result

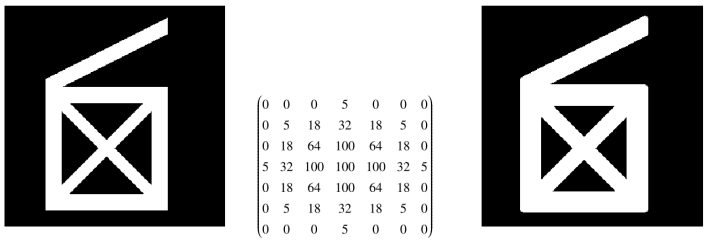
Fig. 2.1. A simple blur done with convolutions

2.3.2 Gaussian blur

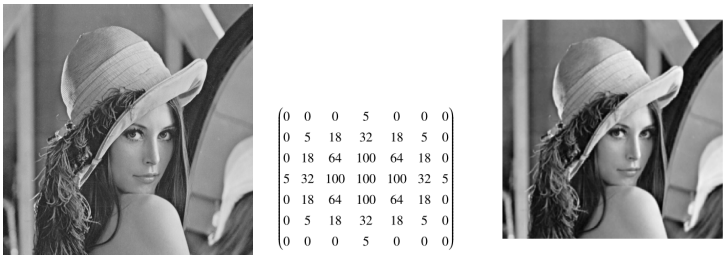
Gaussian blur has certain mathematical properties that makes it important for computer vision. And you can approximate it with an image convolution. The image convolution kernel for a Gaussian blur is:

0	0	0	5	0	0	0
0	5	18	32	18	5	0
0	18	64	100	64	18	0
5	32	100	100	100	32	5
0	18	64	100	64	18	0
0	5	18	32	18	5	0
0	0	0	5	0	0	0

Here's a result that I got:



(a) image (b) filter (c) result



(d) image (e) filter (f) result

Fig. 2.2. A Gaussian blur done with convolutions

2.3.3 Line detection with image convolutions

With image convolutions, you can easily detect lines. Here are four convolutions to detect horizontal, vertical and lines at 45 degrees:

-1	-1	-1
2	2	2
-1	-1	-1

Horizontal lines

-1	2	-1
-1	2	-1
-1	2	-1

Vertical lines

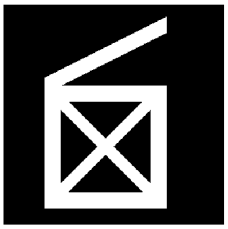
-1	-1	2
-1	2	-1
2	-1	-1

45 degree lines

2	-1	-1
-1	2	-1
-1	-1	2

135 degree lines

Here's 0,90,45,135 lines detection that I got on an image:



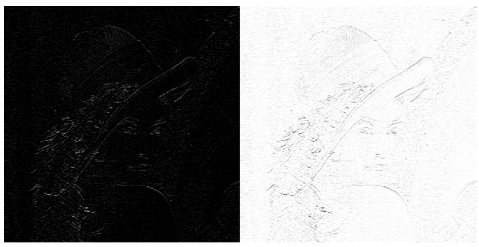
(a) image

$$\begin{pmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{pmatrix}$$

(b) filter

(c) result

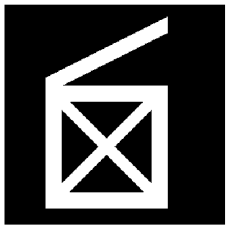
(d) image



(e) result

Fig. 2.3. A horizontal line detection done with convolutions

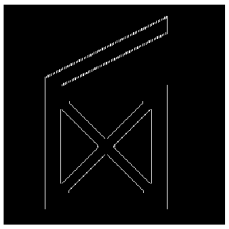
In Lena,the black background is the original result, the white background is obtained by subtracting the original result from 255, the same below.



(a) image

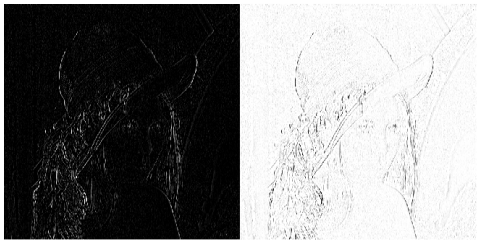
$$\begin{pmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{pmatrix}$$

(b) filter



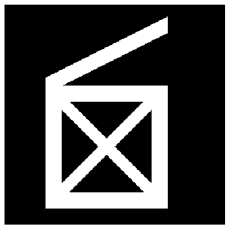
(c) result

(d) image



(e) result

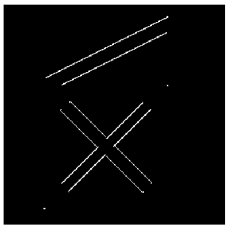
Fig. 2.4. A vertical line detection done with convolutions



(a) image

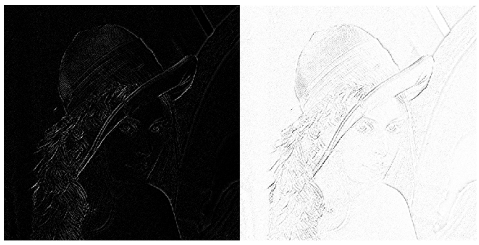
$$\begin{pmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{pmatrix}$$

(b) filter



(c) result

(d) image



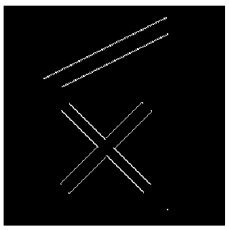
(e) result

Fig. 2.5. A 45 degress line detection done with convolutions

(a) image

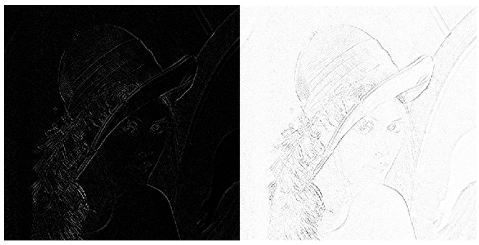
$$\begin{pmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{pmatrix}$$

(b) filter



(c) result

(d) image



(e) result

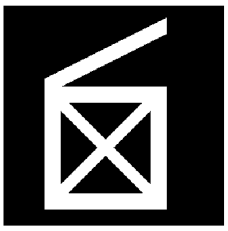
Fig. 2.6. A 135 degress line detection done with convolutions

2.3.4 Edge detection

The above kernels are in a way edge detectors. Only thing is that they have separate components for horizontal and vertical lines. A way to ”combine” the results is to merge the convolution kernels. The new image convolution kernel looks like this:

-1	-1	-1
-1	8	-1
-1	-1	-1

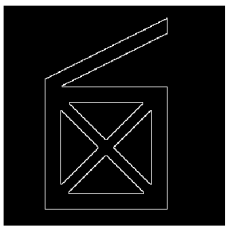
Below result I got with edge detection:



(a) image

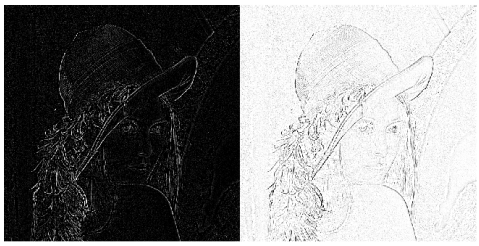
$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

(b) filter



(c) result

(d) image



(e) result

Fig. 2.7. A edge detection done with convolutions

2.3.5 The Sobel Edge Operator

The above operators are very prone to noise. The Sobel edge operators have a smoothing effect, so they're less affected to noise. Again, there's a horizontal component and a vertical component.

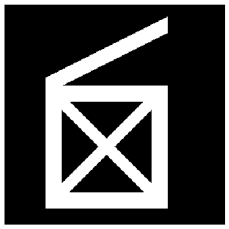
-1	-2	-1
0	0	0
1	2	1

Horizontal

-1	0	1
-2	0	2
-1	0	1

Vertical

On applying horizontal component in image , the result was:



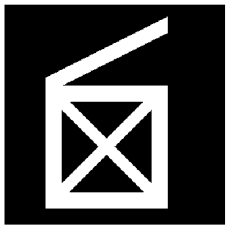
$$\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

(a) image (b) filter (c) result

(d) image (e) result

Fig. 2.8. A horizontal sobel edge operator done with convolutions

On applying vertical component in image , the result was:



(a) image

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

(b) filter

(c) result

(d) image

(e) result

Fig. 2.9. A vertical sobel edge operator done with convolutions

2.3.6 The laplacian operator

The laplacian is the second derivative of the image. It is extremely sensitive to noise, so it isn't used as much as other operators. Unless, of course you have specific requirements.

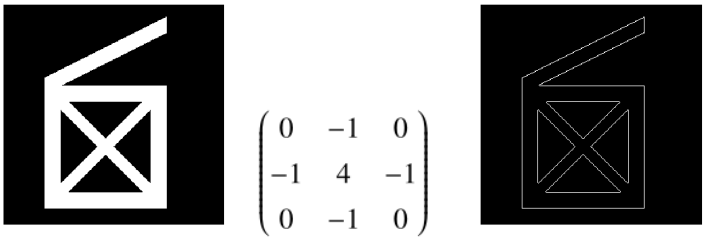
0	-1	0
-1	4	-1
0	-1	0

The laplacian operator

-1	-1	-1
-1	8	-1
-1	-1	-1

The laplacian operator (include diagonals)

Here's the result with the convolution kernel without diagonals:



$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

(a) image (b) filter (c) result

(d) image (e) result

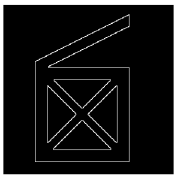
Fig. 2.10. A laplace operator done with convolutions

The result with the convolution kernel with diagonals:

(a) image

$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

(b) filter



(c) result

(d) image

(e) result

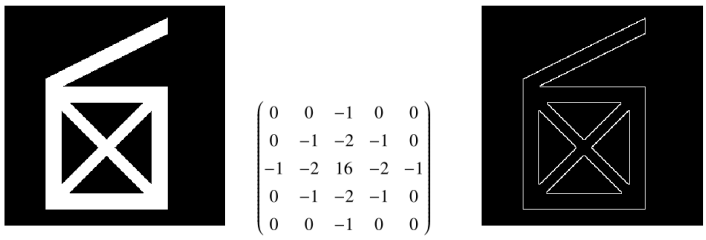
Fig. 2.11. A laplace operator include diagonals done with convolutions

2.3.7 The Laplacian of Gaussian

The laplacian alone has the disadvantage of being extremely sensitive to noise. So, smoothing the image before a laplacian improves the results we get. This is done with a 5x5 image convolution kernel.

0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

The result on applying this image convolution was:



(a) image (b) filter (c) result

(d) image (e) result

Fig. 2.12. A Laplacian of Gaussian operator done with convolutions

2.3.8 Summary

You got to know about some important operations that can be approximated using an image convolution. You learned the exact convolution kernels used and also saw an example of how each operator modifies an image. I hope this helped!

References

[1] J. He, L. Li, J. Xu, and C. Zheng. Relu deep neural networks and linear finite elements. *arXiv preprint arXiv:1807.03973*, 2018.

[2] Q. Wang and W. E. Exponential convergence of the deep neural network approximation for analytic functions. *arXiv preprint arXiv:1807.00297*, 2018.

[3] D. Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 94:103–114, 2017.