Essential mathematics in Neural Networks

Richard Xu

October 11, 2022

1 Introduction

I have been teaching people about Neural Networks and Convolutional Neural Networks for many years. So I developed my own style of explaining things. Although there are thousands of blogs and YouTube videos explaining what neural networks do, I've always wanted to write ”my own” teaching notes on this topic to show the essential mathematics about neural networks and convolutional neural networks. The only prior knowledge the reader needs to have is just basic calculus.

I started off by showing the reader the three different models at the last output layer: logistic regression (binary classification), multinomial regression (multi-class classification) and linear regression (well, true regression!). Then I showed people the concept of gradient descent algorithm. Then I moved on to the main part, to show the reader what a basic fully connected neural network is, and then I end this note by showing people what a convolutional neural network is.

To make things easier, I also try to keep expression in terms of scalars as much as possible in this note.

2 Logistic regression

Although Logistic Regression has the word “regression” in it, it's actually used for classification. The same goes for Multinomial regression.

2.1 motivation

Assume there are **two classes** of data that the classifier (with parameter θ) need to classify:

1. data x_1 has label $y_1 = 1$.
2. data x_2 has label $y_2 = 0$
3. data x_3 has label $y_3 = 1$
4. data x_4 has label $y_4 = 0$
5. data x_5 has label $y_5 = 1 \dots$

So basically, we want to build a classifier $f(x, \theta)$, such that:

1. $(y_1 = 1) \implies f(x_1, \theta)$ should be as close to 1 as possible, e.g., $f(x_1, \theta) = 0.99$
2. $(y_2 = 0) \implies f(x_2, \theta)$ should be as close to 0 as possible, e.g., $f(x_2, \theta) = 0.005$
3. $(y_3 = 1) \implies f(x_3, \theta)$ should be as close to 1 as possible, e.g., $f(x_3, \theta) = 0.92$
4. $(y_4 = 0) \implies f(x_4, \theta)$ should be as close to 0 as possible, e.g., $f(x_4, \theta) = 0.1$
5. $(y_5 = 1) \implies f(x_5, \theta)$ should be as close to 1 as possible, e.g., $f(x_5, \theta) = 0.93$
6. $\dots$

2.1.1 property of $f(x, \theta) \in (0, 1)$

It is obvious that $f(x_i, \theta) \in (0, 1)$, i.e., it should be closer to 1 when $y_i = 1$ and close to zero otherwise. So let's find such $f(x, \theta) \in (0, 1)$ that will make it happen.

Compared with previous slides, I have interchangeably used:

$$\hat{y}_\theta(x) \equiv f(x, \theta) \tag{1}$$

Confused? Let's use linear regression idea to illustrate.

2.1.2 thinking about linear regression

Linear regression is a lot easier to understand, so let me use it to illustrate:

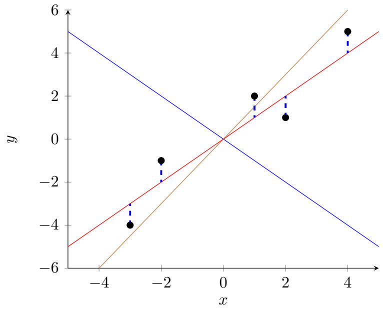


Figure 1: linear regression: the red line is the best fit

so for linear regression, your algorithm is to find the best line (the red line) that minimize the sum of residuals (the blue dash lines). So basically, for a linear regression model you need define two things:

1. the model $f(\mathbf{x}_i, \theta^*)$. In linear regression, it is a line $y_i = \mathbf{m}^\top \mathbf{x}_i + b$
2. a way to measure the loss between predicted and real label: $\ell(\hat{y}_i, y_i)$. In linear regression it is $\ell(\hat{y}_i, y_i) \equiv (\hat{y}_i - y_i)^2$

So the overall loss will be:

$$\mathcal{L} = \sum_{i=1}^{N} \ell(\hat{y}_i, y_i) \quad \text{or} \quad \mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} \ell(\hat{y}_i, y_i) \tag{2}$$

So the question is how do do the same for Logistic Regression?

2.1.3 corresponding diagrammatic representation

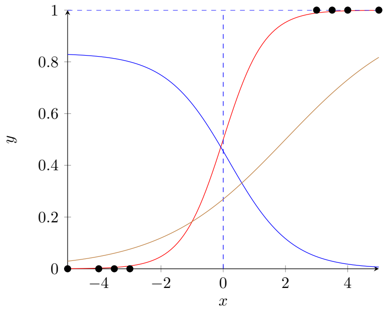


Figure 2: logistic regression: the red line is the best fit

then the question is what is:

1. the model $f(\mathbf{x}_i, \theta^*)$
2. a way to measure the loss between predicted and real label: $\ell(\hat{y_i}, y_i)$?

2.2 Sigmoid function σ

Well, we found a squashing (activation) function to map any x, i.e., $f(x, \theta)$ to be a value between $(0 \dots 1)$:

$$\begin{aligned} \sigma(t) &= \frac{1}{1+\exp(-t)} \\ &= \frac{\exp(t)}{\exp(t)+1} \end{aligned} \tag{3}$$

One of the best ways to see the properties of this function is to plug the extreme points into it. Since we are allowing $-\infty \leq x_i \leq \infty$, then why not just to test them? We can very easily find:

$$\begin{cases} \sigma(t=-\infty) & = \frac{1}{1+\exp(\infty)} \\ & = 0 \\ \sigma(t=+\infty) & = \frac{1}{1+\exp(-\infty)} \\ & = 1 \end{cases} \tag{4}$$

and let's see what this function looks like:

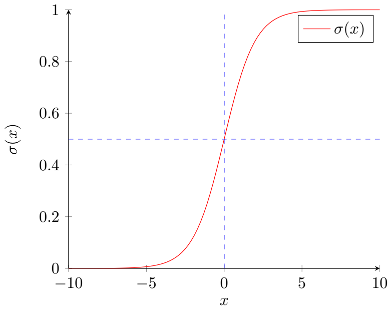


Figure 3: σ function

this is why it's called a "squashing" function, since $\sigma(x)$ squashes any number between $-\infty, \dots, \infty$ to be between $0, \dots, 1$.

2.3 parameters

Remember, our goal is to "learn" the best model in terms of its parameter θ to fit the data. However, so far, just by looking at $\sigma(t) = \frac{1}{1+\exp(-t)}$, there is no θ for us to learn. So we need to insert a "learnable" θ into this equation. This can be easily achieved by having:

$$t = \mathbf{x}_i^\top \boldsymbol{\theta} + b \tag{5}$$

then our equation with parameter becomes:

$$\sigma(\mathbf{x}_i^\top \theta) = \frac{1}{1 + \exp\big(- (\mathbf{x}_i^\top \boldsymbol{\theta} + b)\big)} \tag{6}$$

2.3.1 what is $\mathbf{x}^\top \boldsymbol{\theta}$?

the $^\top$ operation is called the inner product:

$$\begin{aligned} \mathbf{x} &= [x_1, \dots x_n] \\ \boldsymbol{\theta} &= [\theta_1, \dots, \theta_n] \\ \implies \mathbf{x}^\top \boldsymbol{\theta} &= \sum_{i=1}^{n} x_i \theta_i \end{aligned} \tag{7}$$

Basically, it is just to have a sum of product between two vectors's elements. For example:

$$\begin{aligned} \mathbf{x} &= [2, 3, 2] \\ \boldsymbol{\theta} &= [1, 3, 4] \\ \implies \mathbf{x}^\top \boldsymbol{\theta} &= 2 \times 1 + 3 \times 3 + 2 \times 4 \\ &= 19 \end{aligned} \tag{8}$$

However, without loss of generality, you may view both $\mathbf{x}_i$ and $\boldsymbol{\theta}$ as 1-D number, then $\mathbf{x}^\top \boldsymbol{\theta}$ becomes a product of two (scalar) number, which is also a (scalar) number.

2.3.2 Role of θ in sigmoid function

to demonstrate, we have used $\theta = 0.5$ and $\theta = 2.0$ therefore their corresponding σ function becomes:

$$\sigma(0.5x) = \frac{1}{1+\exp\big(-0.5x)\big)} \qquad \sigma(2x) = \frac{1}{1+\exp\big(-2x)\big)} \tag{9}$$

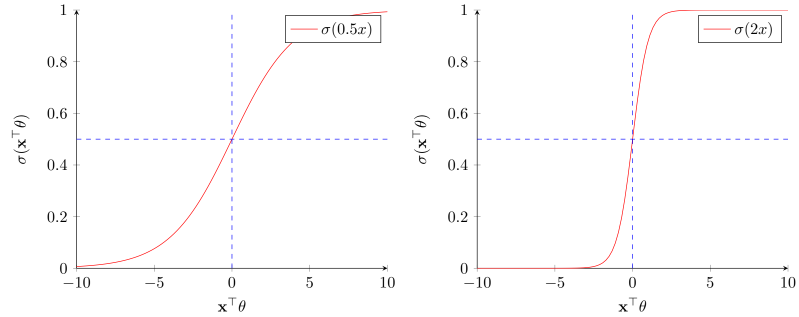


Figure 4: $\sigma(\mathbf{x}^\top\theta)$ on the left: we have $\sigma(0.5x)$ (i.e., $\theta = 0.5$). On the right, we have $\sigma(2x)$ (i.e., $\theta = 2$)

You can see that the right hand side figure is much more steep as we are using larger $\boldsymbol{\theta}$. You should also notice that without the b term, the zero value at the x-axis has σ function value occur at 0.5

what if we negate $\boldsymbol{\theta}$? Please see the following plots, where we have $\boldsymbol{\theta} = -0.5$ and $\boldsymbol{\theta} = -2$ respectively:

$$\sigma(0.5x) = \frac{1}{1+\exp\big(0.5x)\big)} \qquad \sigma(2x) = \frac{1}{1+\exp\big(2x)\big)} \tag{10}$$

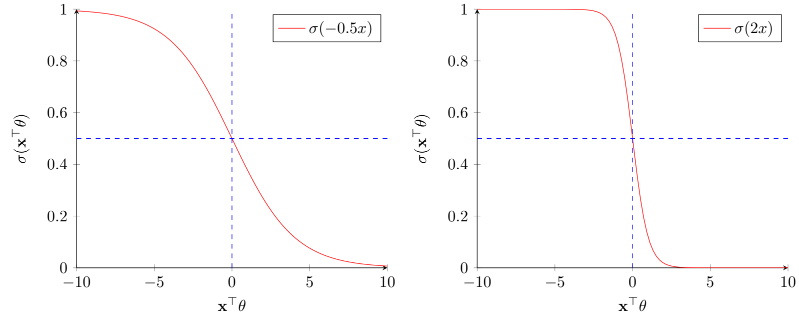


Figure 5: $\sigma(-\mathbf{x}^\top\theta)$ on the left: we have $\sigma(-0.5x)$ (i.e., $\theta = -0.5$). On the right, we have $\sigma(-2x)$ (i.e., $\theta = -2$)

2.3.3 Role of b

let's plot the following cases where $b = -1$ and $b = 1$:

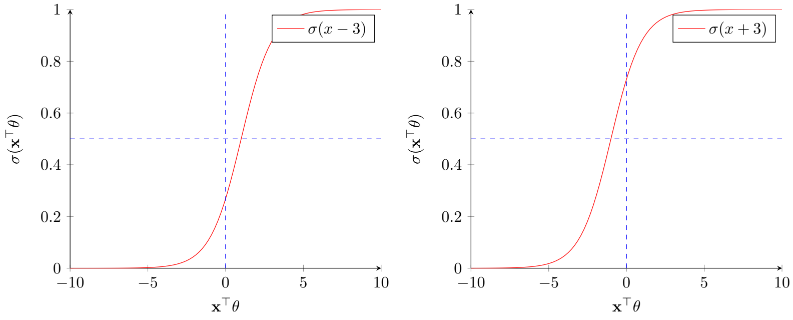


Figure 6: $\sigma(\mathbf{x}^\top\theta)$ on the left: we have $\sigma(x-3)$ (i.e., $b = 3$). On the right, we have $\sigma(x+3)$ (i.e., $b = -3$)

.

Notice that on both cases, the plot was shifted by b amount horizontally. You should also notice that the b value at the x-axis has a corresponding σ function value of 0.5.

2.3.4 simplifying $\mathbf{x}_i^\top\boldsymbol{\theta} + b$

It is much easier to write everything in the form of inner project rather than having a summation term in the end. Therefore, $\mathbf{x}_i^\top\boldsymbol{\theta} + b$ can be simplified by:

$$\begin{cases} \begin{bmatrix} \mathbf{x}_i \\ 1 \end{bmatrix} & \longrightarrow \tilde{\mathbf{x}}_i \\ \\ \begin{bmatrix} \boldsymbol{\theta} \\ b \end{bmatrix} & \longrightarrow \tilde{\boldsymbol{\theta}} \end{cases} \tag{11}$$

you can see that by doing so:

$$\begin{aligned} \tilde{\mathbf{x}}_i^\top\tilde{\boldsymbol{\theta}} &= \mathbf{x}_i^\top\theta + 1 \times b \\ &= \mathbf{x}_i^\top\theta + b \end{aligned} \tag{12}$$

Therefore, without loss of generality, when we have (part of) equation in the form $\mathbf{x}_i^\top\theta$, we can assume that b term is in it.

2.3.5 Properties of sigmoid function

One of the most important property to remember for $\sigma(t)$ function is that:

$$\frac{\mathrm{d}\sigma(t)}{\mathrm{d}t} = \sigma(t)(1-\sigma(t)) \qquad\qquad \text{it's always positive} \tag{13}$$

Note that $\frac{\mathrm{d}\sigma(t)}{\mathrm{d}t}$ is the derivative operator, which is used to calculate the rate of change of a curve (or function). Although it is not necessary to understand how differentiation is calculated mathematically. But students need to understand what a derivative is, because understanding it is important for neural network's back-propagation operation.

Also it's important to remember that the largest value of $\frac{\mathrm{d}\sigma(t)}{\mathrm{d}t}$ is 0.25!

$$\begin{aligned} 1-\sigma(t) &= 1-\frac{1}{1+\exp(-t)} \\ &= \frac{1+\exp(-t)-1}{1+\exp(-t)} \\ &= \frac{\exp(-t)}{1+\exp(-t)} \\ &= \sigma(-t) \end{aligned} \tag{14}$$

2.3.6 computing $\frac{\mathbf{d}\sigma(t)}{\mathbf{d}t}$

$$\begin{aligned} \frac{\mathrm{d}\sigma(t)}{\mathrm{d}t} &= \frac{\mathrm{d}\left(\frac{1}{1+\exp(-t)}\right)}{\mathrm{d}t} \\ &= \frac{\exp(-t)}{(1+\exp(-t))^2} \\ &= \left(\frac{1}{1+\exp(-t)}\right)\left(\frac{\exp(-t)}{1+\exp(-t)}\right) \\ &= \sigma(t)(1-\sigma(t)) \qquad \text{it's always positive} \end{aligned} \tag{15}$$

Question can you compute the largest value of $\frac{\mathrm{d}\sigma(t)}{\mathrm{d}t}$ without $\frac{\mathrm{d}^2\sigma(t)}{\mathrm{d}t^2} = 0$?

We can also compute the derivative of $\frac{\mathrm{d}\sigma(-t)}{\mathrm{d}t}$:

$$\begin{aligned} \frac{\mathrm{d}\sigma(-t)}{\mathrm{d}t} &= \frac{\mathrm{d}\left(\frac{1}{1+\exp(t)}\right)}{\mathrm{d}t} = \frac{-\exp(t)}{(1+\exp(t))^2} \\ &= -\underbrace{\left(\frac{1}{1+\exp(t)}\right)}_{\sigma(-t) \text{ or } 1-\sigma(t)}\underbrace{\left(\frac{\exp(t)}{1+\exp(t)}\right)}_{\sigma(t)} \\ &= -\sigma(t)(1-\sigma(t)) \qquad \text{it's always negative} \end{aligned} \tag{16}$$

2.4 Another squashing (activation) tanh function:

Obviously it's inappropriate for logistic regression, as tanh maps values $(-1,\ldots,1)$. However, sometimes you may want the labels to be between $(-1,\ldots,1)$

$$\begin{aligned} \tanh(x) &= \frac{\exp^x-\exp^{-x}}{\exp^x+\exp^{-x}} \\ &= \frac{\exp^{2x}-1}{\exp^{2x}+1} \qquad \times\exp^x \\ &= \frac{1-\exp^{-2x}}{1+\exp^{-2x}} \qquad \times\exp^{-x} \end{aligned} \tag{17}$$

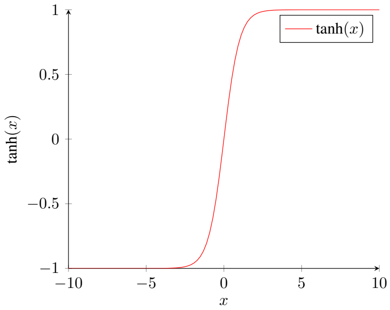


Figure 7: tanh function

2.4.1 tanh derivatives

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{\exp^x - \exp^{-x}}{\exp^x + \exp^{-x}}\right) &= \frac{1}{(\exp^x + \exp^{-x})}\frac{\mathrm{d}}{\mathrm{d}x}\left(\exp^x - \exp^{-x}\right) - \frac{\exp^x - \exp^{-x}}{(\exp^x + \exp^{-x})^2}\frac{\mathrm{d}}{\mathrm{d}x}\left(\exp^x + \exp^{-x}\right)\\ &= \frac{\exp^x + \exp^{-x}}{(\exp^x + \exp^{-x})^2}\frac{\mathrm{d}}{\mathrm{d}x}\left(\exp^x - \exp^{-x}\right) - \frac{\exp^x - \exp^{-x}}{(\exp^x + \exp^{-x})^2}\frac{\mathrm{d}}{\mathrm{d}x}\left(\exp^x + \exp^{-x}\right)\\ &= \frac{\left(\exp^x + \exp^{-x}\right)\left(\exp^x + \exp^{-x}\right)}{(\exp^x + \exp^{-x})^2} - \frac{\left(\exp^x - \exp^{-x}\right)\left(\exp^x - \exp^{-x}\right)}{(\exp^x + \exp^{-x})^2}\\ &= \frac{\left(\exp^x + \exp^{-x}\right)^2 - \left(\exp^x - \exp^{-x}\right)^2}{(\exp^x + \exp^{-x})^2}\\ &= 1 - \left(\frac{\exp^x - \exp^{-x}}{\exp^x + \exp^{-x}}\right)^2\\ &= 1 - \tanh(x)^2\end{aligned} \tag{18}$$

2.5 Logistic Regression

now that we know how to define $\hat{y}$, we also need to define $\ell(\hat{y}, y)$. So we resort to our favorite Cross Entropy function:

2.5.1 formal revision on Cross Entropy

Remember from last topic, you have studied **Entropy** formally in which you have learned Entropy of a distribution p is:

$$H(p) = -\sum_{x \in \mathcal{S}} p(X = x)\log(p(X = x)) \tag{19}$$

in the case where sample space $\mathcal{S}$ only takes on two values, say $\{0, 1\}$ it becomes:

$$H(p) = -\Big(p(X = 0)\log(p(X = 0)) + p(X = 1)\log(p(X = 1))\Big) \tag{20}$$

looking very closely, the formula for **cross entropy** requires just a small change from the formula of **Entropy**. Let's still look at the simple case where both q and q has the same sample space $\{0, 1\}$:

$$H(p, q) = -\Big(p(X = 0)\log(q(X = 0)) + p(X = 1)\log(q(X = 1))\Big) \tag{21}$$

2.5.2 example of Cross Entropy

It's time we substitute some numbers into Cross Entropy:

1. Cross Entropy between distribution:

$$\begin{aligned} &\begin{cases} p(X=0) &= 0.3 \\ p(X=1) &= 0.7 \end{cases} \qquad \begin{cases} q(X=0) &= 0.7 \\ q(X=1) &= 0.3 \end{cases} \\ \implies H(p,q) &= -\Big(0.3\log(0.7)+0.7\log(0.3)\Big) \\ &= 0.949 \end{aligned} \tag{22}$$

2. Cross Entropy between distribution:

$$\begin{aligned} &\begin{cases} p(X=0) &= 0.3 \\ p(X=1) &= 0.7 \end{cases} \qquad \begin{cases} q(X=0) &= 0.3 \\ q(X=1) &= 0.7 \end{cases} \\ \implies H(p,q) &= -\Big(0.3\log(0.3)+0.7\log(0.7)\Big) \\ &= 0.610 \end{aligned} \tag{23}$$

   note that even two distributions are identical. Their cross Entropy is not 0

3. Cross Entropy between distribution:

$$\begin{aligned} &\begin{cases} p(X=0) &= 1 \\ p(X=1) &= 0 \end{cases} \qquad \begin{cases} q(X=0) &= 0.3 \\ q(X=1) &= 0.7 \end{cases} \\ \implies H(p,q) &= -\Big(0.3\log(0.3)+0.7\log(0.7)\Big) \\ &= 1.203 \end{aligned} \tag{24}$$

4. Cross Entropy between distribution:

$$\begin{aligned} &\begin{cases} p(X=0) &= 1 \\ p(X=1) &= 0 \end{cases} \qquad \begin{cases} q(X=0) &= 0.9 \\ q(X=1) &= 0.1 \end{cases} \\ \implies H(p,q) &= -\Big(0.3\log(0.3)+0.7\log(0.7)\Big) \\ &= 0.105 \end{aligned} \tag{25}$$

   compare the last two, it's obvious that the last one should have much smaller cross entropy!

2.5.3 back to Logistic regression

Let me give you the logistic regression objective function directly!

Given a set of points and their labels $(\mathbf{x}_i, y_i)$, where $y_i \in \{0,1\}$, we need to find $\boldsymbol{\theta}$ that minimize the following objective function:

$$\begin{aligned} \mathcal{C}(\boldsymbol{\theta}) &= -\Big(\sum_{i=1}^{n} y_i \log \sigma(\mathbf{x}_i^\top \boldsymbol{\theta}) + (1-y_i)\log\big(1-\sigma(\mathbf{x}_i^\top \boldsymbol{\theta})\big)\Big) \\ &= -\Big(\sum_{i=1}^{n} y_i \log\Big[\frac{1}{1+\exp(-\mathbf{x}_i^\top \boldsymbol{\theta})}\Big] + (1-y_i)\log\Big[1-\frac{1}{1+\exp(-\mathbf{x}_i^\top \boldsymbol{\theta})}\Big]\Big) \end{aligned} \tag{26}$$

can you spot why the above is the Cross Entropy? where is p and q? Let's looking at just the term inside the summation and drop data index i:

by letting $q_i \equiv \sigma(\mathbf{x}_i^\top \boldsymbol{\theta})$, the last expression is in the form of:

$$\begin{aligned} &-\Big(\underbrace{y}_{p_1}\log\Big[\underbrace{\frac{1}{1+\exp(-\mathbf{x}^\top\boldsymbol{\theta})}}_{q_1}\Big] + \underbrace{(1-y)}_{p_2}\log\Big[\underbrace{1-\frac{1}{1+\exp(-\mathbf{x}^\top\boldsymbol{\theta})}}_{q_2}\Big]\Big) \\ &= -\Big(p_1\log(q_1) + p_2\log(q_2)\Big) \end{aligned} \tag{27}$$

Hopefully, now you can see despite the complex form of Eq.(26), this is in fact **Cross Entropy** between two distributions which you studied in Topic 4!

It is easy to see that Eq.(26) and equation Eq.(28) below are equivalent:

$$\mathcal{C}(\boldsymbol{\theta}) = \begin{cases} -\sum_{i:y_i=1}\log\Big[\frac{1}{1+\exp(-\mathbf{x}_i^\top\boldsymbol{\theta})}\Big] \\ \\ -\sum_{i:y_i=0}\log\Big[1-\frac{1}{1+\exp(-\mathbf{x}_i^\top\boldsymbol{\theta})}\Big] \end{cases} \tag{28}$$

You may optimize the above using gradient descend. We will talk about them together with the Multinomial regression.

2.5.4 Logistic regression from a Bernoulli probability perspective

Remember all the biased coin examples you learned in the stats course?

$$\begin{cases} \Pr(Y=1) & = t \\ \Pr(Y=0) & = 1-t \end{cases} \tag{29}$$

You can combine both lines into a single equation: Bernoulli probability distribution:

$$\Pr(Y=y\,|t) = t^y\ (1-t)^{(1-y)} \qquad y \in \{0,1\} \tag{30}$$

You are half-way there, we can now change $t \rightarrow \sigma(x_i^\top\boldsymbol{\theta})$, so we can actually define a Bernoulli probability for a random label Y_i:

$$\begin{cases} \Pr(Y_i=1) & = \sigma(x_i^\top\boldsymbol{\theta}) \\ \Pr(Y_i=0) & = 1-\sigma(x_i^\top\boldsymbol{\theta}) \end{cases} \tag{31}$$

Then we can simply have $\mathbf{x}_i^\top\boldsymbol{\theta}$, we can maximize the joint likelihood probability:

$$\begin{aligned} \Pr(\mathbf{Y}|\mathbf{X},\boldsymbol{\theta}) &= \prod_{i=1}^{n}\left[\frac{1}{1+\exp(-x_i^\top\boldsymbol{\theta})}\right]^{y_i}\left[1-\frac{1}{1+\exp(-x_i^\top\boldsymbol{\theta})}\right]^{1-y_i} \\ &= \prod_{i=1}^{n} t_i^{y_i}(1-t_i)^{1-y_i} \qquad \text{let } t_i \equiv \frac{1}{1+\exp(-x_i^\top\boldsymbol{\theta})} \end{aligned} \tag{32}$$

note that we are writing $\Pr(\mathbf{Y}|\mathbf{X},\boldsymbol{\theta}) = \prod_{\mathbf{i=1}}^{\mathbf{n}}\Pr(\mathbf{y_i})$ and then perform a maximum likelihood estimation (MLE):

$$\boldsymbol{\theta}_{\text{MLE}} = \arg\,\max_{\boldsymbol{\theta}}\Big(\prod_{i=1}^{n}\Big[\frac{1}{1+\exp(-x_i^\top\boldsymbol{\theta})}\Big]^{y_i}\Big[1-\frac{1}{1+\exp(-x_i^\top\boldsymbol{\theta})}\Big]^{1-y_i}\Big) \tag{33}$$

However, this is different to what you have learnt previously, for example, Gaussian MLE:

$$\theta_{\text{MLE}} = \arg\ \max_\theta \prod_{i=1}^{n} \mathcal{N}(y_i; \mu, \sigma) \tag{34}$$

This time, instead of having each y_i be distributed from an independent and identical (IID) distribution, instead, in our case, each y_i is distributed from an independent but non-identical distribution. They all share the same $\boldsymbol{\theta}$, but each $\mathbf{x}_i$ is specific only to their respective y_i.

2.5.5 How can we reconcile the two?

here is the watershed moment where we can minimize the $-\log(\Pr)$:

$$\begin{aligned}\mathcal{C}(\boldsymbol{\theta}) &= -\log[p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta})] \\ &= -\Big(\prod_{i=1}^{n} \Big[\frac{1}{1+\exp(-x_i^\top \boldsymbol{\theta})}\Big]^{y_i} \Big[1 - \frac{1}{1+\exp(-x_i^\top \boldsymbol{\theta})}\Big]^{1-y_i}\Big)\end{aligned} \tag{35}$$

2.5.6 food for thought about Bernoulli

basically we need to have a way to combine the following two statements into a single line:

$$\begin{cases}\Pr(Y=1) & = t \\ \Pr(Y=0) & = 1-t\end{cases} \tag{36}$$

and there can be many different choices, for example:

$$\begin{aligned}&\Pr_t(Y=y) = (1-t)^{(1-y)} t^y \quad \text{or :} \\ &\Pr_t(Y=y) = (1-t)(1-y) + py\end{aligned} \tag{37}$$

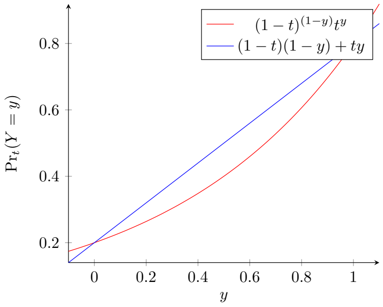


Figure 8: different functions to combine Bernoulli statements

2.6 Softmax function

Assume there are four classes that the classifier is designed for:

1. if data x_1 has label $y_1 = 1$, then y_1 can be written as $[1, 0, 0, 0]^\top$
2. if data x_2 has label $y_2 = 2$, then y_2 can be written as $[0, 1, 0, 0]^\top$

3. if data x_3 has label $y_3 = 2$, then y_3 can be written as $[0, 1, 0, 0]^\top$
4. if data x_4 has label $y_4 = 3$, then y_4 can be written as $[0, 0, 1, 0]^\top$
5. if data x_5 has label $y_5 = 4$, then y_5 can be written as $[0, 0, 0, 1]^\top$
6. ...

So basically, we want to build a classifer $f(x, \theta)$, such that:

1. $(y_1 = 1) \implies f(x_1, \theta)$ be as close to $[1, 0, 0, 0]^\top$ as possible, e.g., $f(x_1, \theta) = [0.99, 0.005, 0.005, 0]^\top$
2. $(y_2 = 2) \implies f(x_2, \theta)$ be as close to $[0, 1, 0, 0]^\top$ as possible, e.g., $f(x_2, \theta) = [0.005, 0.99, 0.005, 0]^\top$
3. $(y_3 = 2) \implies f(x_3, \theta)$ be as close to $[0, 1, 0, 0]^\top$ as possible, e.g., $f(x_3, \theta) = [0.005, 0.99, 0.005, 0]^\top$
4. $(y_4 = 3) \implies f(x_4, \theta)$ be as close to $[0, 0, 1, 0]^\top$ as possible, e.g., $f(x_4, \theta) = [0.005, 0.005, 0.99, 0]^\top$
5. $(y_5 = 4) \implies f(x_5, \theta)$ be as close to $[0, 0, 0, 1]^\top$ as possible, e.g., $f(x_5, \theta) = [0.005, 0.005, 0, 0.99]^\top$
6. ...

Therefore, we need to have a function $f(\boldsymbol{\theta}, \mathbf{x})$ that can transform $\mathbf{x}$ to $[\pi_1, \pi_2, \pi_3, \pi_4]^\top$, such that $\sum_{i=1}^{4} \pi_i = 1$

2.6.1 Softmax function expression

This is a beautiful function called "softmax" function. You may find a survey paper I wrote about some of the interesting research about this function:

`https://github.com/roboticcam/machine-learning-notes/blob/master/files/softmax.pdf`

A softmax function of $k = 3$ classes are (and you can generalize it to arbitrary k classes easily)

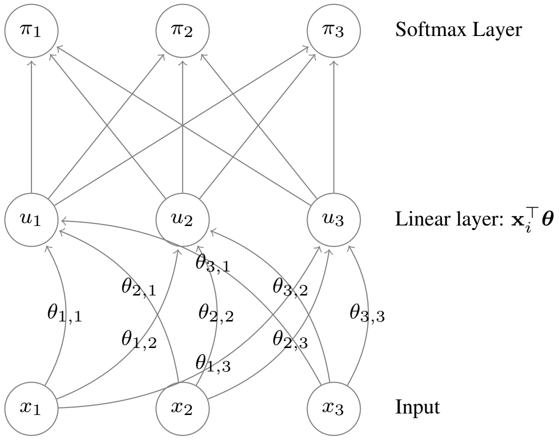
$$f(\boldsymbol{\theta}, \mathbf{x}_i) = \left(\frac{\exp(\mathbf{x}_i^\top \boldsymbol{\theta}_1)}{\sum_{k=1}^{3} \exp(\mathbf{x}_i^\top \boldsymbol{\theta}_k)}, \frac{\exp(\mathbf{x}_i^\top \boldsymbol{\theta}_2)}{\sum_{k=1}^{3} \exp(\mathbf{x}_i^\top \boldsymbol{\theta}_k)}, \frac{\exp(\mathbf{x}_i^\top \boldsymbol{\theta}_3)}{\sum_{k=1}^{3} \exp(\mathbf{x}_i^\top \boldsymbol{\theta}_k)} \right) \tag{38}$$

where $\boldsymbol{\theta} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3\}$

2.6.2 Diagrammatic representation

Although you can write Softmax function down in a single step, I thought it may still be useful to express them step by step in terms of "layers". In Neural networks, layers represent the different steps of a composite function:

$$f_3 \odot f_2 \odot f_1(x) \equiv \underbrace{f_1\big(\underbrace{f_2\big(\underbrace{f_1(x)}_{\text{layer 1}}\big)}_{\text{layer 2}}\big)}_{\text{layer 3}} \tag{39}$$



from this diagram, you can tell:

1. a single data $\mathbf{x} \equiv (x_1, x_2, x_3)$
2. parameter $\boldsymbol{\theta} \equiv \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3\}$, where

$$\boldsymbol{\theta}_1 = \{\theta_{1,1}, \theta_{1,2}, \theta_{1,3}\} \quad \boldsymbol{\theta}_2 = \{\theta_{2,1}, \theta_{2,2}, \theta_{2,3}\} \quad \boldsymbol{\theta}_3 = \{\theta_{3,1}, \theta_{3,2}, \theta_{3,3}\} \tag{40}$$

Let's look at each of layers, so we introduce auxiliary variables u_1, u_2, u_3:

1. Linear Layer

$$\begin{aligned} u_1 &= \mathbf{x}^\top \boldsymbol{\theta}_1 = \sum_{i=1}^{3} x_i \theta_{1,i} \\ u_2 &= \mathbf{x}^\top \boldsymbol{\theta}_2 = \sum_{i=1}^{3} x_i \theta_{2,i} \\ u_3 &= \mathbf{x}^\top \boldsymbol{\theta}_3 = \sum_{i=1}^{3} x_i \theta_{3,i} \end{aligned} \tag{41}$$

   each of the $\mathbf{x}^\top \boldsymbol{\theta}_i$ is called logit

2. Softmax Layer

$$\begin{aligned} \pi_1 \equiv \Pr(y_i = 1 | \mathbf{x}_i, \boldsymbol{\theta}) &= \frac{\exp(u_1)}{\sum_{k=1}^{3} \exp(u_k)} = \frac{\exp(\mathbf{x}_i^\top \boldsymbol{\theta}_1)}{\sum_{k=1}^{3} \exp(\mathbf{x}_i^\top \boldsymbol{\theta}_k)} \\ \pi_2 \equiv \Pr(y_i = 2 | \mathbf{x}_i, \boldsymbol{\theta}) &= \frac{\exp(u_2)}{\sum_{k=1}^{3} \exp(u_k)} = \frac{\exp(\mathbf{x}_i^\top \boldsymbol{\theta}_2)}{\sum_{k=1}^{3} \exp(\mathbf{x}_i^\top \boldsymbol{\theta}_k)} \\ \pi_3 \equiv \Pr(y_i = 3 | \mathbf{x}_i, \boldsymbol{\theta}) &= \frac{\exp(u_3)}{\sum_{k=1}^{3} \exp(u_k)} = \frac{\exp(\mathbf{x}_i^\top \boldsymbol{\theta}_3)}{\sum_{k=1}^{3} \exp(\mathbf{x}_i^\top \boldsymbol{\theta}_k)} \end{aligned} \tag{42}$$

of course, it is easy to check that:

$$\pi_1 + \pi_2 + \pi_3 = 1 \tag{43}$$

one may easily extend them to multiple K arbitrary classes

2.6.3 Relationship between 2-class Softmax and sigmoid function

$$\begin{aligned}\pi_1 \equiv \pi(y_i = 1|\mathbf{x}_i, \boldsymbol{\theta}) &= \frac{\exp(\mathbf{x}^\top\boldsymbol{\theta}_1)}{\exp(\mathbf{x}^\top\boldsymbol{\theta}_1)+\exp(\mathbf{x}^\top\boldsymbol{\theta}_2)} \\ &= \frac{1}{1+\exp\left(\mathbf{x}^\top(\boldsymbol{\theta}_2-\boldsymbol{\theta}_1)\right)} \\ &= \frac{1}{1+\exp\left(\mathbf{x}^\top(-\boldsymbol{\theta})\right)} \\ &= \frac{\exp\left(\mathbf{x}^\top\boldsymbol{\theta}\right)}{\exp\left(\mathbf{x}^\top\boldsymbol{\theta}\right)+1}\end{aligned} \tag{44}$$

which is identical to sigmoid function

2.7 Logistic Regression and Multinomial Regression

An perfect $\boldsymbol{\theta}$ that gives zero error for all data obviously impossible. We need to find a $\boldsymbol{\theta}$ which minimize the sum of cost instead. We talk about both the two class (Logistic regression) and the k-class Multinomial Regression (and confusingly, sometimes it is also called Softmax Regression).

We also follow what is said about the relationship between Bernoulli MLE and Logistic Regression. Let's draw the relationship between Multinomial MLE and Multinomial regression!

$$\begin{aligned}\mathcal{C}(\boldsymbol{\theta}) &= -\sum_{i=1}^{N}\sum_{k=1}^{K}\underbrace{y_{i,k}}_{p}\underbrace{\log\left[\frac{\exp(\mathbf{x}_i^\top\boldsymbol{\theta}_k)}{\sum_{l=1}^{K}\exp(\mathbf{x}_i^\top\boldsymbol{\theta}_l)}\right]}_{\log(q)} \\ &= -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{i,k}\Big[\log\exp(\mathbf{x}_i^\top\boldsymbol{\theta}_k) - \log\sum_{l=1}^{K}\exp(\mathbf{x}_i^\top\boldsymbol{\theta}_l)\Big]\end{aligned} \tag{45}$$

it is easy to see that the above can be expressed as:

$$-\sum_{i=1}^{n}\sum_{k=1}^{K} y_{i,k}\log q_{i,k} \tag{46}$$

this is the **Cross Entropy** between the one-hot vector $[y_{i,1} \quad \ldots \quad y_{i,K}]$ and $[q_{i,1} \quad \ldots \quad q_{i,K}]$. The above can be alternatively expressed as (when we have K different classes):

$$\mathcal{C}(\boldsymbol{\theta}) = \begin{cases} -\sum_{i:y_i=[1,\ldots,0]}\log\left[\frac{\exp(\mathbf{x}_i^\top\boldsymbol{\theta}_1)}{\sum_{l=1}^{K}\exp(\mathbf{x}_i^\top\boldsymbol{\theta}_l)}\right] \\ \qquad\qquad\vdots \\ -\sum_{i:y_i=[0,\ldots,1]}\log\left[\frac{\exp(\mathbf{x}_i^\top\boldsymbol{\theta}_K)}{\sum_{l=1}^{K}\exp(\mathbf{x}_i^\top\boldsymbol{\theta}_l)}\right]\end{cases} \tag{47}$$

2.7.1 from Multinomial distribution perspective

given we have a probability distribution of K possible outcomes, we have t_i being the probability of the i^{th} component. Therefore, we have a probability vector $\mathbf{t} = [t_1 \quad \ldots \quad t_K]$, such that $\sum_{t_i} = 1$, and our random categories Y is expressed as one-hot vector:

$$\begin{cases} \Pr(Y = \begin{bmatrix}1 & 0 & \ldots & 0\end{bmatrix}) & = t_1 \\ \Pr(Y = \begin{bmatrix}0 & 1 & \ldots & 0\end{bmatrix}) & = t_2 \\ \vdots \\ \Pr(Y = \begin{bmatrix}0 & 0 & \ldots & 1\end{bmatrix}) & = t_K \end{cases} \tag{48}$$

using a single line expression, we should have:

$$
\begin{aligned}
\Pr(Y = y \mid \mathbf{t}) &= \prod_{k=1}^{K} {t_k}^{y_k} \qquad y \text{ is a one-hot vector} \\
\Pr(Y_i = y_i \mid \mathbf{t}) &= \prod_{k=1}^{K} {t_{i,k}}^{y_{i,k}} \qquad \text{add data index } i
\end{aligned} \tag{49}
$$

Just like the Logistic regression case, if we substitute $t_{i,k}$ for one of the component of the softmax function, i.e.,:

$$
t_{i,k} = \frac{\exp(\mathbf{x}_i^\top \boldsymbol{\theta}_k)}{\sum_{l=1}^{K} \exp(\mathbf{x}_i^\top \boldsymbol{\theta}_l)} \tag{50}
$$

then we have the joint Likelihood density (remember they are independent and non-identical):

$$
\Pr(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta}) = \prod_{i=1}^{N} \prod_{k=1}^{K} \Big[\frac{\exp(\mathbf{x}_i^\top \boldsymbol{\theta}_k)}{\sum_{l=1}^{K} \exp(\mathbf{x}_i^\top \boldsymbol{\theta}_l)} \Big]^{y_{i,k}} \tag{51}
$$

now if we take the $-\log(\Pr)$:

$$
\begin{aligned}
\mathcal{C}(\boldsymbol{\theta}) = -\log\Big(\Pr(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta})\Big) &= -\log\Big(\prod_{i=1}^{N} \prod_{k=1}^{K} \Big[\frac{\exp(\mathbf{x}_i^\top \boldsymbol{\theta}_k)}{\sum_{l=1}^{K} \exp(\mathbf{x}_i^\top \boldsymbol{\theta}_l)} \Big]^{y_{i,k}}\Big) \\
&= -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{i,k} \log \Big[\frac{\exp(\mathbf{x}_i^\top \boldsymbol{\theta}_k)}{\sum_{l=1}^{K} \exp(\mathbf{x}_i^\top \boldsymbol{\theta}_l)} \Big]
\end{aligned} \tag{52}
$$

2.7.2 How to find θ^*?

How may we find $\boldsymbol{\theta}^*$, such that:

$$
\begin{aligned}
\boldsymbol{\theta}^* &= \arg\min_{\boldsymbol{\theta}} \mathcal{C}(\boldsymbol{\theta}) \\
&= \arg\min_{\boldsymbol{\theta}} \Big(-\sum_{i=1}^{N} \sum_{k=1}^{K} y_{i,k} \log \Big[\frac{\exp(\mathbf{x}_i^\top \boldsymbol{\theta}_k)}{\sum_{l=1}^{K} \exp(\mathbf{x}_i^\top \boldsymbol{\theta}_l)} \Big] \Big)
\end{aligned} \tag{53}
$$

We need to solve it using Gradient descend algorithm

2.8 Gradient Descend algorithm

It is an iterative method of finding maxima or minima for $f(\mathbf{x})$, when you can not solve it analytically, i.e., solve for $\nabla f(\mathbf{x}) = 0$:

$$
\mathbf{x}_{n+1} = \mathbf{x}_n - \eta_n \nabla f(\mathbf{x}_n), \quad n \geq 0 \tag{54}
$$

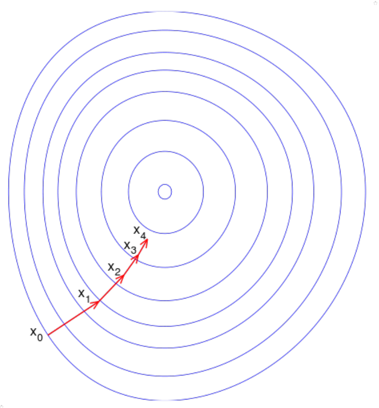


Figure 9: gradient descend in 2D

A descriptive explanation is that when a person wants to go from an arbitrary location in a mountain to a valley (minimum) in the dark, if the person is only equipped with a flashlight and can only illuminate the vicinity, then the person goes along the steepest direction. So he/she walks in one step by η length. When the person reaches a new location, he/she finds a new steepest direction and takes another step with η length (both step size do not need to equal). By doing this repeatedly, the person hopes to reach the valley.

The question remains, what is the definition of the steepest direction? It turns out that it is the gradient vector $\nabla f(\mathbf{x}_n)$ estimated at the location $\mathbf{x}_n$.

so you can see that in order to use gradient descend, you must find the expression of the derivative $\nabla f(\mathbf{x})$.

2.8.1 1-D gradient descend example

Let us use:

$$\begin{aligned} f(x) &= x^2 \\ \implies \nabla_x f(x) &= 2x \end{aligned} \tag{55}$$

Let's pretend we don't know how to find the minimum of $f(x) = x^2$, we need to use gradient descent to find it:

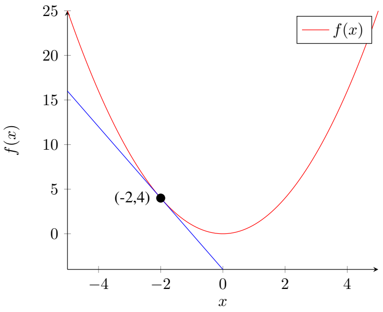


Figure 10: gradient descend in 1-D, starting at $x_1 = -2$

as the above diagram shows, we let $\mathbf{x}_1 = -2$, then, the first iteration would be, if we let $\eta_n = 0.1$:

$$\begin{aligned} \mathbf{x}_{n+1} &= \mathbf{x}_n - \eta_n \nabla f(\mathbf{x}_n) \\ \implies \mathbf{x}_2 &= \mathbf{x}_1 - \eta \times 2x_1 \\ &= -2 - 0.1 \times 2 \times (-2) \\ &= -1.6 \end{aligned} \tag{56}$$

now that we have $\mathbf{x}_2 = -1.6$, so the gradient need to computed at $\mathbf{x} = -1.6$:

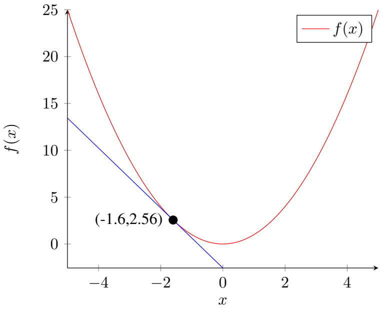


Figure 11: gradient descend in 1-D, this iteration starting at $\mathbf{x}_2 = -1.6$

You can iteratively apply the gradient descend to obtain $\mathbf{x}_3$ by

$$\begin{aligned} \mathbf{x}_{n+1} &= \mathbf{x}_n - \eta_n \nabla f(\mathbf{x}_n) \\ \implies \mathbf{x}_3 &= \mathbf{x}_2 - \eta \times 2x_2 \\ &= -1.6 - 0.1 \times 2 \times (-1.6) \\ &= -1.28 \end{aligned} \tag{57}$$

you repeat the process until some stopping criteria is met. A common stopping criteria may be when $|\mathbf{x}_{n+1} - \mathbf{x}_n| \leq \delta$.

2.8.2 starting from $\mathbf{x}_1 = 2$

The same applies if we start from $\mathbf{x}_1 = 2$:

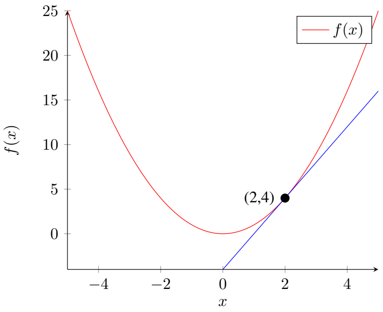


Figure 12: gradient descend in 1-D, starting at $\mathbf{x}_1 = 2$

as the above diagram shows, we let $\mathbf{x}_1 = 2$, then, the first iteration would be, if we let $\eta_n = 0.1$:

$$
\begin{aligned}
\mathbf{x}_{n+1} &= \mathbf{x}_n - \eta_n \nabla f(\mathbf{x}_n) \\
\implies \mathbf{x}_2 &= \mathbf{x}_1 - \eta \times 2x_1 \\
&= 2 - 0.1 \times 2 \times (2) \\
&= 1.6
\end{aligned}
\tag{58}
$$

now that we have $\mathbf{x}_2 = 1.6$, so the gradient need to computed at $\mathbf{x} = 1.6$, i.e., $\nabla_x f(1.6)$:

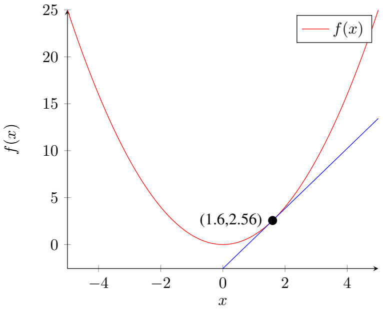


Figure 13: gradient descend in 1-D, this iteration starting at $\mathbf{x}_2 = 1.6$

You can iteratively apply the gradient descend to obtain $\mathbf{x}_3$ by

$$
\begin{aligned}
\mathbf{x}_{n+1} &= \mathbf{x}_n - \eta_n \nabla f(\mathbf{x}_n) \\
\implies \mathbf{x}_3 &= \mathbf{x}_2 - \eta \times 2x_2 \\
&= 1.6 - 0.1 \times 2 \times (1.6) \\
&= 1.28
\end{aligned}
\tag{59}
$$

you repeat the process until some stopping criteria is met. A common stopping criteria may be when $|\mathbf{x}_{n+1} - \mathbf{x}_n| \leq \delta$.

2.9 Gradiend descend for Multinomial regression

Therefore, in the case of Multinomial regression, we just need to find the gradient, in order to apply Graident Descend algorithm. This time, we are not pretending we don't know how to solve for $\nabla_{\boldsymbol{\theta}} \mathcal{C}(\boldsymbol{\theta})$!

$$
\begin{aligned}
\nabla_{\boldsymbol{\theta}} \mathcal{C}(\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \Big(- \sum_{i=1}^{N} \sum_{k=1}^{K} y_{i,k} \Big[\log \Big(\frac{\exp(\mathbf{x}_i^\top \boldsymbol{\theta}_k)}{\sum_{l=1}^{K} \exp(\mathbf{x}_i^\top \boldsymbol{\theta}_l)} \Big) \Big] \Big) \\
&= \begin{bmatrix} \frac{\exp(\mathbf{x}_i^\top \boldsymbol{\theta}_1)}{\sum_{l=1}^{K} \exp(\mathbf{x}_i^\top \boldsymbol{\theta}_l)} \\ \vdots \\ \frac{\exp(\mathbf{x}_i^\top \boldsymbol{\theta}_K)}{\sum_{l=1}^{K} \exp(\mathbf{x}_i^\top \boldsymbol{\theta}_l)} \end{bmatrix} - y_i \\
&\equiv \Big[\frac{\exp(\mathbf{x}_i^\top \boldsymbol{\theta})}{\sum_{l=1}^{K} \exp(\mathbf{x}_i^\top \boldsymbol{\theta}_l)} - y_i \Big]^\top
\end{aligned}
\tag{60}
$$

remember here y_i is a one-hot vector, and we let $\frac{\exp(\mathbf{x}_i^\top \boldsymbol{\theta})}{\sum_{l=1}^{K} \exp(\mathbf{x}_i^\top \boldsymbol{\theta}_l)}$ to represent the softmax vector

2.9.1 put in Gradient Descend iteration

so the gradient descend algorithm then becomes:

$$\boldsymbol{\theta}^{n+1} = \boldsymbol{\theta}^{n} - \eta_n \left(\sum_{i=1}^{N} \mathbf{x}_i \Big[\frac{\exp(\mathbf{x}_i^\top \boldsymbol{\theta})}{\sum_{l=1}^{K} \exp(\mathbf{x}_i^\top \boldsymbol{\theta}_l)} - y_i \Big]^\top \right), \; n \geq 0 \tag{61}$$

in many cases, we need to add a regulariser to the objective function:

$$\mathcal{C}(\boldsymbol{\theta}) = -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{i,k} \log \left[\frac{\exp(\mathbf{x}_i^\top \boldsymbol{\theta}_k)}{\sum_{l=1}^{K} \exp(\mathbf{x}_i^\top \boldsymbol{\theta}_l)} \right] + \frac{\lambda}{2} \sum_{k=1}^{K} \sum_{j=1}^{m} \theta_{k,j}^2 \tag{62}$$

then, the corresponding gradient descend algorithm becomes:

$$\boldsymbol{\theta}^{n+1} = \boldsymbol{\theta}^{n} - \eta_n \left(\sum_{i=1}^{N} \mathbf{x}_i \Big[\frac{\exp(\mathbf{x}_i^\top \boldsymbol{\theta})}{\sum_{l=1}^{K} \exp(\mathbf{x}_i^\top \boldsymbol{\theta}_l)} - y_i \Big]^\top + \lambda \boldsymbol{\theta}^{n} \right), \; n \geq 0 \tag{63}$$

What's this $\frac{\lambda}{2} \sum_{k=1}^{K} \sum_{j=1}^{m} \theta_{k,j}^2$ business? It's the **regulariser** we added. Regulizer is beyond the scope of this topic, I will not discuss here.

2.10 Find derivatives of Multinomial regression

Let $Z_k = \mathbf{x}_i^\top \boldsymbol{\theta}_k$, we have,

$$\begin{aligned}
\frac{\partial \mathcal{C}(\boldsymbol{\theta})}{\partial Z_k} &= \sum_{i=1}^{N} \frac{\partial \left(-\sum_{k=1}^{K} y_{i,k} \left[\log \left(\frac{\exp(Z_k)}{\sum_{l=1}^{K} \exp(Z_l)} \right) \right] \right)}{\partial Z_k} \\
&= \sum_{i=1}^{N} \left[\frac{\exp(Z_k)}{\sum_{l=1}^{K} \exp(Z_l)} - y_{i,k} \right] \qquad \text{see next page} \\
\implies \frac{\partial \mathcal{C}(\boldsymbol{\theta})}{\partial Z} &= \sum_{i=1}^{N} \begin{bmatrix} \frac{\exp(Z_1)}{\sum_{l=1}^{K} \exp(Z_l)} - y_{i,1} \\ \cdots \\ \frac{\exp(Z_K)}{\sum_{l=1}^{K} \exp(Z_l)} - y_{i,K} \end{bmatrix} = \sum_{i=1}^{N} \left(\frac{\exp(Z)}{\sum_{l=1}^{K} \exp(Z_l)} - y_i \right)
\end{aligned} \tag{64}$$

If we were to differentiate with respect to $\boldsymbol{\theta}$:

$$\frac{\partial \mathcal{C}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \frac{\partial \mathcal{C}(\boldsymbol{\theta})}{\partial Z} \frac{\partial Z}{\partial \boldsymbol{\theta}} = \sum_{i=1}^{N} \mathbf{x}_i \left[\frac{\exp(\mathbf{x}_i^\top \boldsymbol{\theta})}{\sum_{l=1}^{K} \exp(\mathbf{x}_i^\top \boldsymbol{\theta}_l)} - y_i \right]^\top \tag{65}$$

let:

$$p_k = \frac{\exp^{z_k}}{\sum_l \exp^{z_l}} \qquad \mathcal{C} = -\sum_k y_k \log p_k, \qquad \text{where } \sum_k y_k = 1 \tag{66}$$

1. when $k = i$:

$$\frac{\partial p_k}{\partial z_k} = \frac{\partial \left(\frac{\exp^{z_k}}{\sum_l \exp^{z_l}}\right)}{\partial z_k} = \frac{\partial \left(\frac{\exp^{z_k}}{\exp^{z_k} + \sum_{l \neq k} \exp^{z_l}}\right)}{\partial \exp(z_k)} \times \frac{\partial \exp^{z_k}}{\partial z_k} \tag{67}$$

We know the identity:

$$\frac{\partial \frac{x}{x+c}}{\partial x} = \frac{\partial x(x+c)^{-1}}{\partial x} = (x+c)^{-1} - x(x+c)^{-2} = \frac{(x+c) - x}{(x+c)^2} = \frac{c}{(x+c)^2} \tag{68}$$

Therefore,

$$\begin{aligned} \frac{\partial p_k}{\partial z_k} &= \frac{\partial \left(\frac{\exp^{z_k}}{\exp^{z_k} + \sum_{l \neq k} \exp^{z_l}}\right)}{\partial \exp(z_k)} \times \frac{\partial \exp^{z_k}}{\partial z_k} = \frac{\sum_{l \neq k} \exp^{z_l}}{\left(\sum_{l=1}^{K} \exp^{z_l}\right)^2} \times \exp^{z_k} \\ &= \frac{\sum_{l \neq k} \exp^{z_l}}{\left(\sum_{l=1}^{K} \exp^{z_l}\right)} \times \frac{\exp^{z_k}}{\left(\sum_{l=1}^{K} \exp^{z_l}\right)} = p_k(1 - p_k) \end{aligned} \tag{69}$$

2. when $k \neq i$,
   We know the identity:

$$\frac{\partial \frac{y}{z+c}}{\partial z} = \frac{\partial y(z+c)^{-1}}{\partial z} = -\frac{y}{(z+c)^2} \tag{70}$$

$$\begin{aligned} \frac{\partial p_k}{\partial z_i} &= \frac{\left(\partial \frac{\exp^{z_k}}{\exp^{z_i} + \sum_{l \neq i} \exp^{z_l}}\right)}{\partial \exp(z_i)} \times \frac{\partial \exp^{z_i}}{\partial z_i} \\ &= -\frac{\exp^{z_k}}{\left(\sum_{l=1}^{K} \exp^{z_l}\right)^2} \times \exp^{z_i} \\ &= -\frac{\exp^{z_i}}{\left(\sum_{l=1}^{K} \exp^{z_l}\right)} \times \frac{\exp^{z_k}}{\left(\sum_{l=1}^{K} \exp^{z_l}\right)} = -p_i p_k \end{aligned} \tag{71}$$

3. combine the two

$$\frac{\partial p_k}{\partial z_i} = \begin{cases} p_i(1 - p_i), & i = k \\ -p_i p_k, & i \neq k \end{cases} \tag{72}$$

$$
\begin{aligned}
\frac{\partial \mathcal{C}}{\partial z_i} &= -\sum_k y_k \frac{\partial \log p_k}{\partial z_i} \\
&= -\sum_k y_k \frac{1}{p_k} \frac{\partial p_k}{\partial z_i} \\
&= -y_i(1-p_i) - \sum_{k \neq i} y_k \frac{1}{p_k}(-p_k p_i) \\
&= -y_i(1-p_i) + \sum_{k \neq i} y_k (p_i) \\
&= -y_i + y_i p_i + \sum_{k \neq i} y_k (p_i) \\
&= p_i \left(\sum_k y_k \right) - y_i \\
&= p_i - y_i
\end{aligned}
\tag{73}
$$

3 summary of models

Now is a good time summarize what we have studied so far. The objective is to minimize the overall (average across all data) cost:

$$
\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(\mathbf{x}_i, \boldsymbol{\theta})) \tag{74}
$$

looking at above, there are three elements in the above equation: y_i, $f(\mathbf{x}_i, \boldsymbol{\theta})$ and $\ell\big(y_i, f(\mathbf{x}_i, \boldsymbol{\theta})\big)$. Defining them gives us a unique model to solve. For example:

1. logistic regression:

$$
\begin{aligned}
y_i &\in \{0, 1\} \\
f(\mathbf{x}_i, \boldsymbol{\theta}) &\equiv \sigma(\mathbf{x}_i^\top \boldsymbol{\theta}) \\
\ell\big(y_i, f(\mathbf{x}_i, \boldsymbol{\theta})\big) &\equiv \text{Cross Entropy}\big(y_i, f(\mathbf{x}_i, \boldsymbol{\theta})\big)
\end{aligned}
\tag{75}
$$

2. multinomial regression:

$$
\begin{aligned}
y_i &\in \text{one-hot vector} \\
f(\mathbf{x}_i, \boldsymbol{\theta}) &\equiv \text{softmax}(\mathbf{x}_i, \boldsymbol{\theta}) \\
\ell\big(y_i, f(\mathbf{x}_i, \boldsymbol{\theta})\big) &\equiv \text{Cross Entropy}\big(y_i, f(\mathbf{x}_i, \boldsymbol{\theta})\big)
\end{aligned}
\tag{76}
$$

3. linear regression:

$$
\begin{aligned}
y_i &\in \mathbb{R} \\
f(\mathbf{x}_i, \boldsymbol{\theta}) &\equiv \mathbf{x}_i^\top \boldsymbol{\theta} \\
\ell\big(y_i, f(\mathbf{x}_i, \boldsymbol{\theta})\big) &\equiv \big(y_i - f(\mathbf{x}_i, \boldsymbol{\theta})\big)^2
\end{aligned}
\tag{77}
$$

You may notice that the function $\ell(y, \hat{y})$ usually contains no parameter

4 Neural Networks

Neural networks, despite the hype, are expressed in an extremely simple way:

$$\mathbf{a} = \sigma(\mathbf{W}\mathbf{x}) \tag{78}$$

where $\mathbf{a}$ is called neuron, and σ is a nonlinear activation function. Yes, σ has been widely used for a long time as σ! $\mathbf{W}$ is a matrix. $\mathbf{x}$ is the input. Yes, that is it! So from now on, I am just going to change σ to σ function. However, please keep in mind that other non-linear activation function can also be used.

So it looks like the neural network is nothing more than a linear transformation $\mathbf{W}$, followed by a non-linear transformation σ, but what's so magical about it? Let's look at an inspiring example: suppose you decide to fit a linear regression model to the following data: $f(\theta, \mathbf{x}) \equiv \mathbf{U}^\top \mathbf{x}$. Obviously, the fitting is very inaccurate, since $\mathbf{x}$ and y do not form a linear relationship.

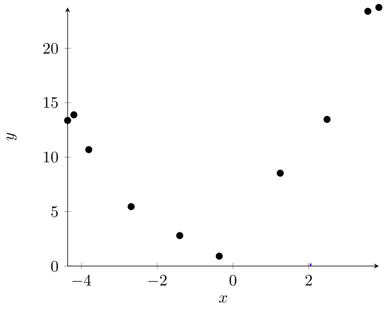


Figure 14: example of x vs y, note they do not form a linear relationship

Then instead of regressing $\mathbf{x}$ directly on y, we now transform $\mathbf{x} \rightarrow \sigma(\mathbf{W}\mathbf{x})$ where $\sigma(\mathbf{W}\mathbf{x})$ is a neural network with activation function σ. By doing this, it has better model flexibility, which allows us to match $\hat{y} \equiv \mathbf{U}^\top \sigma(\mathbf{W}\mathbf{x})$ with y, so we end up with the following:

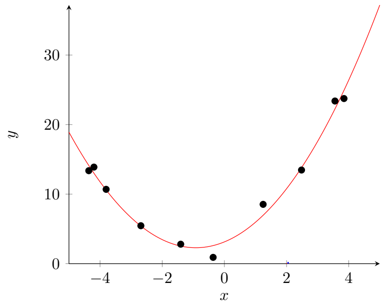


Figure 15: with the help of neural network transformation $\sigma(\mathbf{W}\mathbf{x})$, $\hat{y}$ is fitted much closer to y

This can be viewed equivalently to the situation after we transform $\mathbf{x} \rightarrow \big(\mathbf{a} = \sigma(\mathbf{W}\mathbf{x})\big)$, y is now forming a linear relationship with $\mathbf{a} = \sigma(\mathbf{W}\ x)$, We can illustrate this in the diagram below:

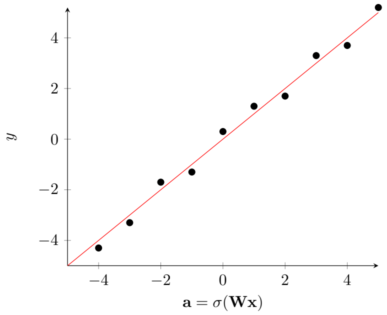


Figure 16: now the neurons $\mathbf{a} = \sigma(\mathbf{W}\mathbf{x})$ forms a linear relationship with y

Therefore, the neural network (here, it is $\sigma(\mathbf{W}\mathbf{x})$) has a good data representation/projection ability, and can transform $\mathbf{x}$ into $\mathbf{a}$, so that $\mathbf{a}$ will form an input/output relationship with the label $\boldsymbol{y}$ indicated by the output layer ($\mathbf{U}^\top \mathbf{a}$ in this case).

4.1 Feed-forward Neural Network in a nutshell

In a nutshell, a neural network operation consists of the following modules:

1. Feed-forward

   this is essentially defining the neural network based model:

$$f(\boldsymbol{\theta}) = f_L(\dots f_2(f_1(\mathbf{x};\theta_1);\theta_2)\dots),\theta_L) \tag{79}$$

   Remember we discussed the idea of having layers to represent composite functions in Eq.(39). Therefore, each function f_i with associated parameter θ_i is the i^{th} layer (neural network + output layer). and:

$$\boldsymbol{\theta} = \{\theta_1, \theta_2, \dots \theta_L\} \tag{80}$$

2. Back-propagation

   Once we have defined our feed-forward model, we need to put it in a gradient descent framework, i.e.,

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \nabla_{\boldsymbol{\theta}} f_L(\boldsymbol{\theta}_t) \tag{81}$$

   so we must know how to compute $\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_t)$, through a process called Back-propagation

4.1.1 Something to remember about derivatives

We know all about it already from high school mathematics:

1. **Product rule**:

$$\frac{\partial f}{\partial \boldsymbol{\theta}}\left(f\left(\boldsymbol{\theta}\right)g\left(\boldsymbol{\theta}\right)\right) = f\left(\boldsymbol{\theta}\right)\frac{\partial}{\partial \boldsymbol{\theta}}g\left(\boldsymbol{\theta}\right) + \frac{\partial}{\partial \boldsymbol{\theta}}f\left(\boldsymbol{\theta}\right)g\left(\boldsymbol{\theta}\right)$$

2. **Derivative of sums**:

$$\frac{\partial f}{\partial \boldsymbol{\theta}}\left(f\left(\boldsymbol{\theta}\right)+g\left(\boldsymbol{\theta}\right)\right)=\frac{\partial}{\partial \boldsymbol{\theta}}f\left(\boldsymbol{\theta}\right)+\frac{\partial}{\partial \boldsymbol{\theta}}g\left(\boldsymbol{\theta}\right)$$

3. **Chain rule**

$$\begin{aligned}\frac{\partial f}{\partial \boldsymbol{\theta}_l} &= \frac{\partial}{\partial \boldsymbol{\theta}_l} f_L(\dots f_2(f_1(\mathbf{x};\boldsymbol{\theta}_1);\boldsymbol{\theta}_2)\dots),\boldsymbol{\theta}_L) \\ &= \frac{\partial f_L}{\partial f_{L-1}}\frac{\partial f_{L-1}}{\partial f_{L-2}}\cdots\frac{\partial f_{l+2}}{\partial f_l}\frac{\partial f_l}{\partial \boldsymbol{\theta}_l}\end{aligned} \tag{82}$$

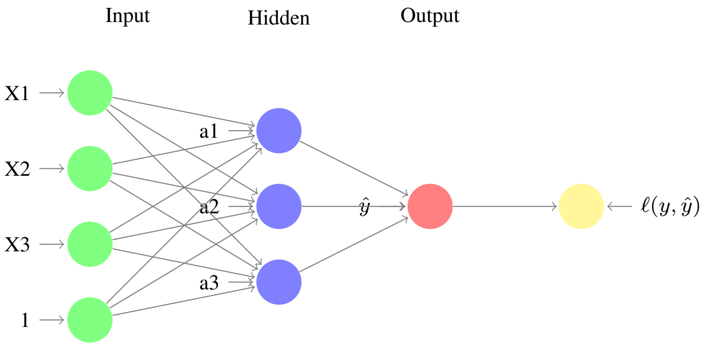
4.2 Look at Neural network systematically

Let's try something new! For this illustrative example, let's use Linear Regression for the last layer. So instead of having the last layer output $\hat{y}$ to be σ function, or softmax function, let's make $\hat{y} \in \mathbb{R}$ instead (easy!):

Furthermore, in order to distinguish the parameters related to the last layer from those related to the rest of the neural network, i.e., instead of using the generic $\boldsymbol{\theta}$ for all the parameters, we now refer the last layer (linear regression) parameters as $\mathbf{U}$ and the neural network parameters $\mathbf{W}$ and b.

$$\begin{aligned}\hat{y} &= \mathbf{U}^\top \sigma(\mathbf{W}\mathbf{x}+b) \\ &= \mathbf{U}^\top \underbrace{\sigma(\underbrace{\mathbf{W}\mathbf{x}+b}_{z})}_{\mathbf{a}} = \mathbf{U}^\top \mathbf{a}\end{aligned} \tag{83}$$

let $\hat{y}$ be a **scalar** as we are dealing with a regression output layer. However, if you are given a classification problem, then **softmax** needs to be used.



$$\hat{y} = \mathbf{U}^\top \sigma\left(\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33}\end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3\end{bmatrix}+\begin{bmatrix} b_1 \\ b_2 \\ b_3\end{bmatrix}\right) \tag{84}$$

then we should have, let $\hat{y} = \mathbf{U}^\top \sigma(\cdot)$:

$$\begin{aligned}\ell(y,\hat{y}) &\equiv \frac{1}{2}(y-\hat{y})^2 \\ \frac{\partial \ell}{\partial \hat{y}} &= y-\hat{y}\end{aligned} \tag{85}$$

4.2.1 the accompanying example

Instead of just throwing symbols at you, I think it would be helpful to attach some numerical examples:

$$\begin{aligned}\mathbf{X} &= \begin{bmatrix}2 & 1 & 2\end{bmatrix} \qquad y = 1\\ \mathbf{W} &= \begin{bmatrix}1 & 0 & 1\\ 1 & -1 & 0\\ 0 & 0 & 2\end{bmatrix}\\ \mathbf{b} &= \begin{bmatrix}1 & 1 & 1\end{bmatrix}\\ \mathbf{U} &= \begin{bmatrix}-1 & 1 & 1\end{bmatrix}\end{aligned} \tag{86}$$

we will use this example to illustrate every stage of the feedforward pipeline:

4.3 Feed Forward layers

the feed forward layer operates in a pipleline moving forward, so let's have a look at it in each forward operations:

4.3.1 computing z

Let **z** be an auxiliary variable, the value of the linear layer:

$$\begin{aligned}z_1 = \mathbf{W}_{1,:}X + b_1 &= \sum_i W_{1,i}x_i + b_1\\ &= W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1\\ z_2 = \mathbf{W}_{2,:}X + b_2 &= \sum_i W_{2,i}x_i + b_2\\ &= W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2\\ z_3 = \mathbf{W}_{3,:}X + b_3 &= \sum_i W_{3,i}x_i + b_3\\ &= W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3\end{aligned} \tag{87}$$

where $\mathbf{W}_{i,:}$ meaning the i^{th} row of the matrix $\mathbf{W}$. For example, using our example:

$$\begin{aligned}\mathbf{W}_{1,:} &= \begin{bmatrix}1 & 0 & 1\end{bmatrix}\\ \mathbf{W}_{2,:} &= \begin{bmatrix}1 & -1 & 0\end{bmatrix}\\ \mathbf{W}_{3,:} &= \begin{bmatrix}0 & 0 & 2\end{bmatrix}\end{aligned} \tag{88}$$

by substituting our examples and compute **z**:

$$\begin{aligned}z_1 &= 1 \times 2 + 0 \times 1 + 1 \times 2 + 1 = 5\\ z_2 &= 1 \times 2 + (-1) \times 1 + 0 \times 2 + 1 = 2\\ z_3 &= 0 \times 2 + 0 \times 1 + 2 \times 2 + 1 = 5\end{aligned} \tag{89}$$

Looking at $W_{i,j}$, we can realize a few properties:

1. altering the value of $W_{i,j}$, only z_i will be affected.
2. altering the value of x_i, all $\{z_i\}$ will be affected.

4.3.2 computing $\mathbf{a}$

$f(\cdot)$ is the non-linear layer, which applies to each of the z_i individually:

$$\mathbf{a} = \sigma(\mathbf{z}) = \begin{cases} \sigma(z_1) \\ \sigma(z_2) \\ \sigma(z_3) \end{cases} \tag{90}$$

meaning that if we alter the value of z_1, $a_2 = \sigma(z_2)$ and $a_3 = \sigma(z_3)$ will not be affected!

The remaining question is which function f can be suitable? Yes, you guessed right! It's our famous sigmoid function from Eq.(3), (or less commonly, tanh function):

$$\sigma(t) = \frac{1}{1 + \exp(-t)} \tag{91}$$

by substituting our examples and compute $\mathbf{a}$:

$$\sigma(\mathbf{z}) = \begin{cases} \sigma(z_1) \\ \sigma(z_2) \\ \sigma(z_3) \end{cases} = \begin{cases} \sigma(5) \\ \sigma(2) \\ \sigma(5) \end{cases} = \begin{cases} \frac{1}{1+\exp(-5)} \\ \frac{1}{1+\exp(-2)} \\ \frac{1}{1+\exp(-5)} \end{cases} = \begin{cases} 0.993 \\ 0.880 \\ 0.993 \end{cases} \tag{92}$$

4.3.3 computing $\hat{y}$

$$\begin{aligned} \hat{y} &= \mathbf{U}^\top \mathbf{a} \\ &= \sum_i U_i \times a_i \end{aligned} \tag{93}$$

by substituting our examples and compute $\hat{y}$:

$$\hat{y} = (-1) \times 0.993 + 1 \times 0.880 + (-1) \times 0.993 = 0.880 \tag{94}$$

4.3.4 computing $\ell(\hat{y}, y)$

for linear regression layer, we use the square difference loss:

$$\ell(\hat{y}, y) = \big(\hat{y} - y\big)^2 \tag{95}$$

by substituting our examples and compute ℓ:

$$\ell = \big(0.880 - 1\big)^2 = 0.0144 \tag{96}$$

this means that using neural network with parameters $\mathbf{W} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & -1 & 0 \\ 0 & 0 & 2 \end{bmatrix}$ $b = [1 \quad 1 \quad 1]$ $\mathbf{U} = [-1 \quad 1 \quad 1]$ will give and error of 0.0144 when it is tested with input $\mathbf{X} = [2 \quad 1 \quad 2]$ and output $y = 1$.

Obviously the goal is to minimize the average loss across all your data points, i.e.,:

$$\mathcal{L} = \sum_{i=1}^{N} \ell(y_i, \hat{y}_i) \tag{97}$$

Therefore, this will become the job of the gradient descend.

4.3.5 two layer neural network

In many data problems, one layer of neural networks may not provide all the flexibility, so we need to add another. Now we have two layers, so the feed-forward process can be written as:

$$\begin{aligned}
\mathbf{a}^{(0)} &= \mathbf{x} && \text{(input)} \\
\mathbf{z}^{(1)} &= \mathbf{W}^{(1)}\mathbf{a}^{(0)} + \mathbf{b}^{(1)} && \text{(linear)} \\
\mathbf{a}^{(1)} &= \sigma\big(\mathbf{z}^{(1)}\big) && \text{(non-linear)} \\
\mathbf{z}^{(2)} &= \mathbf{W}^{(2)}\mathbf{a}^{(1)} + \mathbf{b}^{(2)} && \text{(linear)} \\
\mathbf{a}^{(2)} &= \sigma\big(\mathbf{z}^{(2)}\big) && \text{(non-linear)} \\
\hat{y} &= \mathbf{U}^{\top} a^{(2)} && \text{(output)} \\
\ell &= \frac{1}{2}(y-\hat{y})^2 && \text{(loss)}
\end{aligned} \tag{98}$$

4.4 Neural network: Backpropagation of single layer

Note that the mathematical derivations in this section will not be assessed. However, students still need to understand the general mechanics of Backpropagation, as it can be assessed on the exam.

$$\begin{aligned}
\hat{y} &= \mathbf{U}^{\top}\sigma(\mathbf{W}\mathbf{x}+b) \\
&= \mathbf{U}^{\top}\,\underbrace{\sigma(\underbrace{\mathbf{W}\mathbf{x}+b}_{z})}_{\mathbf{a}} = \mathbf{U}^{\top}\mathbf{a} \\
\ell(y,\hat{y}) &= \frac{1}{2}(y-\hat{y})^2
\end{aligned} \tag{99}$$

Careful of their dimensions:

$$\begin{aligned}
\underbrace{\frac{\partial \ell}{\partial \mathbf{W}}}_{\text{matrix}} &= \underbrace{\frac{\partial \ell}{\partial \hat{y}}}_{\text{scalar}} \times \underbrace{\frac{\partial \hat{y}}{\partial \mathbf{a}}\frac{\partial \mathbf{a}}{\partial \mathbf{z}}}_{\text{column vector}} \times \underbrace{\frac{\partial \mathbf{z}}{\partial \mathbf{W}}}_{\text{row vector}} \\
&= \underbrace{(y-\hat{y})}_{\text{scalar}}\underbrace{\big(U \odot f'(\mathbf{z})\big)}_{\text{column vector}} \times \underbrace{\mathbf{x}}_{\text{row vector}}
\end{aligned} \tag{100}$$

4.4.1 Backpropagation for $W_{i,j}$

$$\hat{y} = \mathbf{U}^{\top}\,\underbrace{\sigma(\underbrace{\mathbf{W}\mathbf{x}+b}_{z})}_{a} = \mathbf{U}^{\top}a \tag{101}$$

If the dimension of the derivative with respect of $\mathbf{W}$ is too difficult to see, then we take the derivative with respect to $W_{i,j}$ for an element at that time. By noticing which $\mathbf{a}$ (and hence $\mathbf{z}$), and $\mathbf{x}$ element that concern $\mathbf{W}_{i,j}$, we can tell that only a_i, z_i and x_j were involved in $W_{i,j}$. In summary, by looking at the two indices of W:

$$W_{(\text{index of } \mathbf{a}, \text{index of } \mathbf{x})} \tag{102}$$

If we were to compute $\frac{\partial \hat{y}}{W_{i,j}}$ (much easier, as everything is scalar):

$$\begin{aligned}
\frac{\partial \ell}{\partial \mathbf{W}} &= \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{a}} \frac{\partial \mathbf{a}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{W}} \\
\implies \frac{\partial \ell}{\partial W_{i,j}} &= \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_i} \frac{\partial a_i}{\partial z_i} \frac{\partial z_i}{\partial W_{i,j}}
\end{aligned} \tag{103}$$

so what are the derivative of each of the terms?

$$\begin{aligned}
\frac{\partial \ell}{\partial \hat{y}} &= y - \hat{y} \\
\frac{\partial \hat{y}}{\partial a_i} &= \frac{\partial \mathbf{U}^\top \mathbf{a}}{\partial a_i} \\
&= \frac{\partial \sum_j u_j a_j}{\partial a_i} \\
&= U_i \\
\frac{\partial a_i}{\partial z_i} &= \sigma'(z_i) \\
\frac{\partial z_i}{\partial W_{i,j}} &= \frac{\partial \mathbf{W}_{i,:} \mathbf{x} + b_i}{\partial W_{i,j}} \\
&= \frac{\partial \big(\sum_j W_{i,j} x_j + b_i \big)}{\partial W_{i,j}} \\
&= x_j
\end{aligned} \tag{104}$$

4.4.2 putting them together

$$\begin{aligned}
\frac{\partial \ell}{\partial W_{i,j}} &= \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_i} \frac{\partial a_i}{\partial z_i} \frac{\partial z_i}{\partial W_{i,j}} \\
&= (y - \hat{y}) \underbrace{U_i \sigma'(z_i)}_{\delta_i} x_j \\
&= (y - \hat{y}) \underbrace{U_i \sigma'(\mathbf{W}_{i,:} \mathbf{x} + b_i)}_{\delta_i} x_j
\end{aligned} \tag{105}$$

4.4.3 now we try $\frac{\partial \hat{y}}{\partial \mathbf{W}}$

$$\begin{aligned}
\delta &= \begin{bmatrix} U_1 \sigma'(\mathbf{W}_{1,:} \mathbf{x} + b_1) \\ U_2 \sigma'(\mathbf{W}_{2,:} \mathbf{x} + b_2) \\ U_3 \sigma'(\mathbf{W}_{3,:} \mathbf{x} + b_3) \end{bmatrix} \\
&= \mathbf{U} \odot \sigma'(\mathbf{W} \mathbf{x} + \mathbf{b})
\end{aligned} \tag{106}$$

where $\mathbf{U} \odot \mathbf{x}$ is called element-wise times

$$\mathbf{U} \odot \mathbf{x} = \begin{bmatrix} U_1 \\ U_2 \\ U_3 \end{bmatrix} \odot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} U_1 x_1 \\ U_2 x_2 \\ U_3 x_3 \end{bmatrix} \tag{107}$$

$$\begin{aligned}
\frac{\partial \hat{y}}{\partial \mathbf{W}} &= \begin{bmatrix} \delta_1 x_1 & \delta_1 x_2 & \delta_1 x_3 \\ \delta_2 x_1 & \delta_2 x_2 & \delta_2 x_3 \\ \delta_3 x_1 & \delta_3 x_2 & \delta_3 x_3 \end{bmatrix} \\
&= \delta \mathbf{x}^\top \\
\frac{\partial \ell}{\partial \mathbf{W}} &= (y - \hat{y}) \delta \mathbf{x}^\top
\end{aligned} \tag{108}$$

4.4.4 Something about δ

$$\delta = \begin{bmatrix} U_1 \sigma'(W_{1,:}^\top x + b_1) \\ U_2 \sigma'(W_{2,:}^\top x + b_2) \\ U_3 \sigma'(W_{3,:}^\top x + b_3) \end{bmatrix} \\ = \mathbf{U} \odot \sigma'(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{109}$$

δ is the error signal, i.e., $\frac{\partial \hat{y}}{\partial \mathbf{z}}$

4.4.5 Backpropagation for b

$$\frac{\partial \hat{y}}{\partial b_i} = \underbrace{U_i \sigma'(W_{i,:}^\top X + b_i)}_{\delta_i} \\ = \delta_i \\ \frac{\partial \ell}{\partial b_i} = (y - \hat{y})\delta_i \tag{110}$$

4.4.6 Gradient Descend algorithm

remember, the reason why we are doing this is so that we can put them into our gradient descend algorithm:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t) \tag{111}$$

In the case of one-layer neural networks configuration, we have:

$$\boldsymbol{\theta} = [W_{1,1} \quad \dots \quad W_{3,3} \quad b_1 \quad \dots \quad b_3 \quad U_1 \quad \dots \quad U_3]^\top \\ \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial W_{1,1}} & \dots & \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial W_{3,3}} & \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial b_1} & \dots & \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial b_3} & \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial U_1} & \dots & \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial U_3} \end{bmatrix}^\top \tag{112}$$

where

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2} \sum_i \ell\big(y_i, \hat{y}_i(\mathbf{x}_i, \boldsymbol{\theta})\big) \\ = \frac{1}{2} \sum_i \big(y_i - \hat{y}_i(\mathbf{x}_i, \boldsymbol{\theta})\big)^2 \tag{113}$$

4.4.7 Backpropagation for x_j

Note that each x_j is contributed by all $\{a_i\}$. It might be weird to actually perform $\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial x}$ because $\mathbf{x}$ is the data, there's no need to do that because it's not used for gradient descent. However, this can be useful in many cases, as we might want to find out which part of the data is causing the most errors.

$$
\begin{aligned}
\frac{\partial \hat{y}}{\partial x_j} &= \sum_{i=1}^{3} \frac{\partial \hat{y}}{\partial a_i} \frac{\partial a_i}{\partial x_j} \\
&= \sum_{i=1}^{3} \frac{\partial \mathbf{U}^\top a}{\partial a_i} \frac{\partial a_i}{\partial x_j} \\
&= \sum_{i=1}^{3} U_i \frac{\partial \sigma(\mathbf{W}_{i,:} x + \mathbf{b})}{\partial x_j} \\
&= \sum_{i=1}^{3} \underbrace{U_i \sigma'(\mathbf{W}_{i,:} x + \mathbf{b})}_{\delta_i} \frac{\partial \mathbf{W}_{i,:}\mathbf{x}}{\partial x_j} \\
&= \sum_{i=1}^{3} \delta_i W_{i,j} = \delta^\top \mathbf{W}_{:,j}
\end{aligned} \tag{114}
$$

4.5 Backpropagation for two layers

Similar to the single layer section, the mathematical derivations in this section will not be assessed. However, students still need to understand the general mechanics of Backpropagation, as it can be assessed on the exam.

Let's set aside $\ell = \frac{1}{2}(y - \hat{y})^2$ for now:

$$
\hat{y} = \mathbf{U}^\top f\big(\mathbf{W}^{(2)} \sigma\big(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\big) + \mathbf{b}^{(2)}\big) \tag{115}
$$

4.5.1 Backpropagation with respect to $\mathbf{W}^{(2)}$

this is actually identical to the backpropgation of $\mathbf{W}$ in one-layer neural network:

$$
\begin{aligned}
\frac{\partial \hat{y}}{\partial W_{i,j}} &= \underbrace{U_i \sigma'(z_i)}_{\delta_i} x_j \qquad \text{(one layer case)} \\
\implies \frac{\partial \hat{y}}{\partial W^{(2)}_{i,j}} &= \underbrace{U_i \sigma'(z^{(2)}_i)}_{\delta^{(2)}_i} a^{(1)}_j \\
\implies \frac{\partial \hat{y}}{\partial \mathbf{W}^{(2)}} &= \delta^{(2)} \mathbf{a}^{(1)\top} \qquad \text{where } \delta^{(2)} = \mathbf{U} \odot \sigma'(\mathbf{z}^{(2)})
\end{aligned} \tag{116}
$$

4.5.2 Backpropagation with respect to $\mathbf{W}^{(1)}$

$$
\begin{aligned}
\hat{y} &= \mathbf{U}^\top \sigma\big(\mathbf{W}^{(2)} \sigma\big(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\big) + \mathbf{b}^{(2)}\big) \\
&= \mathbf{U}^\top \sigma\Bigg(\underbrace{\mathbf{W}^{(2)} \sigma\big(\underbrace{\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}}_{\mathbf{z}^{(1)}}\big) + \mathbf{b}^{(2)}}_{\mathbf{z}^{(2)}} \Bigg)
\end{aligned} \tag{117}
$$

There is some abuse of notation and I am treating everything as scalars (without thinking of their dimensions):

$$
\begin{aligned}
\frac{\partial \hat{y}}{\partial \mathbf{W}^{(1)}} &= \underbrace{\mathbf{U}\sigma'\left(\mathbf{W}^{(2)}\sigma\left(\mathbf{W}^{(1)}\mathbf{x}+\mathbf{b}^{(1)}\right)+\mathbf{b}^{(2)}\right)}_{\frac{\partial \hat{y}}{\partial \mathbf{a}^{(2)}}\frac{\partial a^{(2)}}{\partial \mathbf{z}^{(2)}}}\underbrace{\mathbf{W}^{(2)}}_{\frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{a}^{(1)}}}\underbrace{\sigma'\left(\mathbf{W}^{(1)}\mathbf{x}+\mathbf{b}^{(1)}\right)}_{\frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{z}^{(1)}}}\ \underbrace{\mathbf{x}}_{\frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{W}^{(1)}}} \\
&= \underbrace{\underbrace{\frac{\partial \hat{y}}{\partial \mathbf{a}^{(2)}}\frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{z}^{(2)}}}_{\delta^{(2)}}\frac{\partial z^{(2)}}{\partial \mathbf{a}^{(1)}}\frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{z}^{(1)}}}_{\delta^{(1)}}\ \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{W}^{(1)}} \\
&= \underbrace{\delta^{(2)}\mathbf{W}^{(2)}\sigma'(\mathbf{z}^{(1)})}_{\delta^{(1)}}\,\mathbf{x}
\end{aligned}
\tag{118}
$$

the last line tells us the recursion, where $\delta^{(l)}$ can be written in terms of $\delta^{(l+1)}$.

4.5.3 Gradient vanishing

did you see in Eq.(118), we have two derivative of activation function of both layers multiply together $\frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{z}^{(2)}} \times \frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{z}^{(1)}}$.

What if we choose the activation function to be the σ-function, and remember that its maximum derivative is only 0.25 from section (2.3.5)? So if we have a lot of layers, we'll have a lot of very small numbers multiply together. So we may end up with the total gradient numerically close to zero!

Remember gradient descend algorithm is $\mathbf{x}_{n+1} = \mathbf{x}_n - \eta_n \nabla f(\mathbf{x}_n), \quad n \geq 0$, therefore, having gradient $\nabla f(\mathbf{x}_n)$, the parameters are not updating. So your algorithm is not "learning".

4.6 Backpropagation of an arbitrary layered neural networks

From the back-propagation of two-layer neural networks, we now put them into the "correct" form of the matrix operations and generalize:

$$
\begin{aligned}
&= \underbrace{\left(\mathbf{W}^{(2)^\top}\delta^{(2)}\right)\odot\sigma'(\mathbf{z}^{(1)})}_{\delta^{(1)}}\,\mathbf{x}^\top \\
&\implies \delta^{(1)} = \left(\mathbf{W}^{(2)^\top}\delta^{(2)}\right)\odot\sigma'(\mathbf{z}^{(1)}) \\
&\implies \delta^{(l)} = \left(\mathbf{W}^{(l)^\top}\delta^{(l+1)}\right)\odot\sigma'(\mathbf{z}^{(l)})
\end{aligned}
\tag{119}
$$

To apply back-propagation to any-layers neural networks:

$$
\delta^{(l)} = \left(\mathbf{W}^{(l)^\top}\delta^{(l+1)}\right)\odot\sigma'(\mathbf{z}^{(l)}) \tag{120}
$$

1. compute δ of the last layer L:

$$
\delta^{(L)} = \mathbf{U}\odot \underbrace{\sigma'(\mathbf{z}^{(L)})}_{\text{from feed-forward}} \tag{121}
$$

2. Generate the whole sequence of $\{\delta^{(l)}\}_1^L$

$$
\delta^{(l)} = \left(\mathbf{W}^{(l)^\top}\delta^{(l+1)}\right)\odot\sigma'(\mathbf{z}^{(l)}) \tag{122}
$$

3. Compute gradients at each layers $\left\{\frac{\partial \hat{y}}{\partial \mathbf{W}^{(l)}}\right\}_{1}^{(L-1)}$:

$$\frac{\partial \hat{y}}{\partial \mathbf{W}^{(l)}} = \delta^{(l+1)} \mathbf{a}^{(l)\top} \tag{123}$$

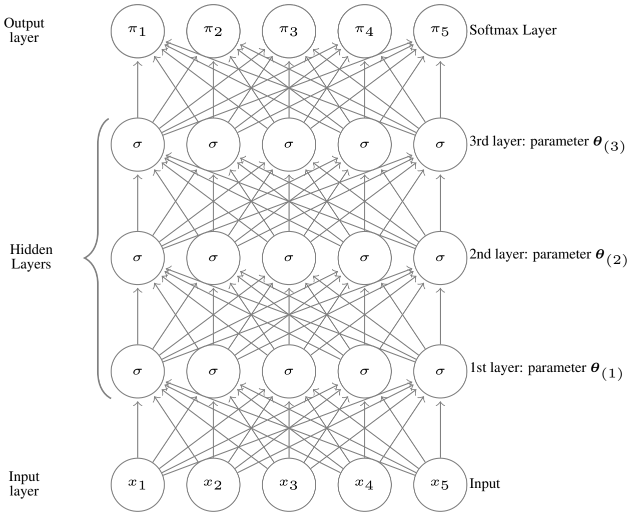
Note that $\frac{\partial \hat{y}}{\partial \mathbf{W}^{(l)}}$ can be obtained as soon as $\delta^{(l+1)}$ becomes available.
In the end we need to apply the loss function (depends the task ℓ and $\hat{y}$), so we will have:

$$\frac{\partial \ell}{\partial \mathbf{W}^{(l)}} = \frac{\partial \ell}{\partial \hat{y}} \delta^{(l+1)} \mathbf{a}^{(l)\top} \tag{124}$$

5 Deep Learning: Convolution Neural Network

5.1 drawback of the traditional “fully-connected” neural networks

if we look at a classification problem:


Output layer
π_1
π_2
π_3
π_4
π_5
Softmax Layer
Hidden Layers
σ
3rd layer: parameter $\boldsymbol{\theta}_{(3)}$
2nd layer: parameter $\boldsymbol{\theta}_{(2)}$
1st layer: parameter $\boldsymbol{\theta}_{(1)}$
Input layer
x_1
x_2
x_3
x_4
x_5
Input

What could be the problem? Well, for complex problems like image or speech recognition, one hidden layer might **not** be enough. However, since each layer may have its own parameter $\boldsymbol{\theta}_{(l)}$, many parameters are involved in total. To make matters worse, when the ”width” of the neural network becomes large (i.e. there are many neurons in a single layer), the weight matrix $\mathbf{W}$ needs to have many elements. Let $m_{l-1} = |\mathbf{a}^{l-1}|$ and $m_l = |z^l|$, where $|\cdot|$ represent the cardinality of a vector or matrix, i.e., the number of elements. Then, the number of elements of the weight matrix of the l^{th} layer is:

$$|\mathbf{W}^{(l)}| = m_{l-1} \times m_l \tag{125}$$

For example, when we’re dealing with an image that might have 1 million pixels, assuming each layer of the neural network has the same number or neurons, you’ll end up with $10^6 \times 10^6$ for each $\mathbf{W}^{(l)}$ elements to learn for just one layer. Therefore, the algorithm has too many parameters to learn!

5.2 Convolution Neural Networks

So how may we reduce the number of parameters from the fully connected network?. The keyword of the day: **parameter sharing**

5.2.1 1-d discrete convolution

it is best to illustrate convolution from a discrete convolution perspective. Let's start with a 1-d convolutions. Basically you just repetitively shift vector g on place to the right and then perform the inner product between the corresponding part of f and g:

$$\begin{aligned} f &= [1 \quad 0 \quad -1 \quad 2 \quad 1 \quad 2] \\ g &= [1 \quad 0 \quad 1] \end{aligned} \tag{126}$$

therefore, after convolution $(f \circledast g)$, you have:

$$\begin{aligned} (f \circledast g)(1) &= [1 \quad 0 \quad -1]\,[1 \quad 0 \quad 1]^\top \\ &= 1 \times 1 + 0 \times 0 + (-1) \times 1 = 0 \\ (f \circledast g)(2) &= [0 \quad -1 \quad 2]\,[1 \quad 0 \quad 1]^\top \\ &= 0 \times 1 + (-1) \times 0 + 2 \times 1 = 2 \\ (f \circledast g)(3) &= [-1 \quad 2 \quad 1]\,[1 \quad 0 \quad 1]^\top \\ &= (-1) \times 1 + 2 \times 0 + 1 \times 1 = 0 \\ (f \circledast g)(4) &= [2 \quad 1 \quad 2]\,[1 \quad 0 \quad 1]^\top \\ &= 2 \times 1 + 1 \times 0 + 2 \times 1 = 5 \end{aligned} \tag{127}$$

final answer is:

$$f \circledast g = [0 \quad 2 \quad 0 \quad 5]^\top \tag{128}$$

5.2.2 2-d discrete convolution

let me show you a 2-dimensional discrete convolution:

$$f = \begin{bmatrix} 1 & 0 & -1 & 2 \\ 2 & -1 & 2 & 1 \\ -1 & 2 & 0 & 1 \\ 1 & -1 & 1 & 1 \end{bmatrix} \qquad g = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{129}$$

let's defining vec$(\mathcal{A})$ as the operation of converting a matrix $\mathcal{A}$ into a vector, for example:

$$\text{vec}\Big(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \Big) = [1 \quad 0 \quad 0 \quad 1]^\top \tag{130}$$

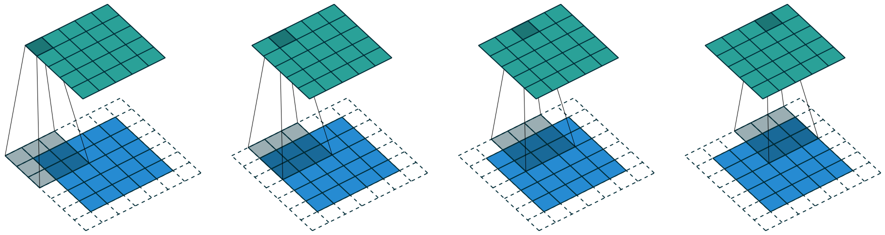
therefore, the 2-d convolution becomes:

$$\begin{aligned}
(f \circledast g)(1,1) &= \operatorname{vec}\Big(\begin{bmatrix}1 & 0\\ 2 & -1\end{bmatrix}\Big)^\top \circledast \operatorname{vec}\Big(\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\Big) = 0\\
(f \circledast g)(1,2) &= \operatorname{vec}\Big(\begin{bmatrix}0 & -1\\ -1 & 2\end{bmatrix}\Big)^\top \circledast \operatorname{vec}\Big(\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\Big) = 2\\
(f \circledast g)(1,3) &= \operatorname{vec}\Big(\begin{bmatrix}-1 & 2\\ 2 & 1\end{bmatrix}\Big)^\top \circledast \operatorname{vec}\Big(\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\Big) = 0\\
(f \circledast g)(2,1) &= \operatorname{vec}\Big(\begin{bmatrix}2 & -1\\ -1 & 2\end{bmatrix}\Big)^\top \circledast \operatorname{vec}\Big(\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\Big) = 4\\
(f \circledast g)(2,2) &= \operatorname{vec}\Big(\begin{bmatrix}-1 & 2\\ 2 & 0\end{bmatrix}\Big)^\top \circledast \operatorname{vec}\Big(\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\Big) = -1\\
(f \circledast g)(2,3) &= \operatorname{vec}\Big(\begin{bmatrix}2 & 1\\ 0 & 1\end{bmatrix}\Big)^\top \circledast \operatorname{vec}\Big(\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\Big) = 3\\
(f \circledast g)(3,1) &= \operatorname{vec}\Big(\begin{bmatrix}-1 & 2\\ 1 & -1\end{bmatrix}\Big)^\top \circledast \operatorname{vec}\Big(\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\Big) = -2\\
(f \circledast g)(3,2) &= \operatorname{vec}\Big(\begin{bmatrix}2 & 0\\ -1 & 1\end{bmatrix}\Big)^\top \circledast \operatorname{vec}\Big(\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\Big) = 3\\
(f \circledast g)(3,3) &= \operatorname{vec}\Big(\begin{bmatrix}0 & 1\\ 1 & 1\end{bmatrix}\Big)^\top \circledast \operatorname{vec}\Big(\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\Big) = 1
\end{aligned} \tag{131}$$

therefore:

$$f \circledast g = \begin{bmatrix}1 & 0 & -1 & 2\\ 2 & -1 & 2 & 1\\ -1 & 2 & 0 & 1\\ 1 & -1 & 1 & 1\end{bmatrix} \circledast \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} = \begin{bmatrix}0 & 2 & 0\\ 4 & -1 & 3\\ -2 & 3 & 1\end{bmatrix} \tag{132}$$

Let me further illustrate discrete convolution through a 2-d example of Convolving a 3×3 kernel over a 5×5 input image with padding 1 (of the first four steps), there are still 12 steps left:



5.2.3 Convolution of Continuous signals

it is a common technique used in signal processing. In continuous functions:

$$(f \circledast g)(t) \overset{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau = \int_{-\infty}^{\infty} f(t-\tau)\, g(\tau)\, d\tau \tag{133}$$

For a continuous signal, it is impossible to show you the ”shift” operation, because there will be an infinite number of shift operations. However, the same approach applies: if you were to get $(f \circledast g)(t')$ for any $t' \neq t$, then it would be:

$$(f \circledast g)(t') \overset{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)\, g(t'-\tau)\, d\tau = \int_{-\infty}^{\infty} f(t'-\tau)\, g(\tau)\, d\tau \tag{134}$$

5.2.4 properties of convolution

1. Exploiting a strong spatially (can be temporal as well if we are dealing with stock data, sound signals) local correlation present in natural data.

2. Assume in a layer, when you apply kernel of size 5×5, if we have 128 channels in both layer $l-1$ and l, then we have something like $128 \times 128 \times 5 \times 5 = 409600$ parameters. This is much less than using fully connected neural networks. Notice the number parameters are depend only on the number of channels as well as the size of the kernel. It does not depends on the width of the network, i.e., number of neurons in an image.

5.2.5 Convolution is a linear operator

as matter of fact, in 1-d discrete convolution, one may replace convolution by a matrix multiplication:

$$
\begin{aligned}
\mathbf{z}^l = W_{\text{conv}} \circledast \mathbf{a}^{l-1} &= [w_1 \quad w_2 \quad \dots w_m] \circledast \mathbf{a}^{l-1} \\
&= \begin{bmatrix}
w_1 & 0 & \dots & 0 & 0 \\
w_2 & w_1 & \dots & \vdots & \vdots \\
w_3 & w_2 & \dots & 0 & 0 \\
\vdots & w_3 & \dots & w_1 & 0 \\
w_{m-1} & \vdots & \dots & w_2 & w_1 \\
w_m & w_{m-1} & \vdots & \vdots & w_2 \\
0 & w_m & \dots & w_{m-2} & \vdots \\
0 & 0 & \dots & w_{m-1} & w_{m-2} \\
\vdots & \vdots & \vdots & w_m & w_{m-1} \\
0 & 0 & 0 & \dots & w_m
\end{bmatrix}
\begin{bmatrix}
a_1^{l-1} \\
a_2^{l-1} \\
a_3^{l-1} \\
\vdots \\
a_n^{l-1}
\end{bmatrix}
\end{aligned}
\tag{135}
$$

you can see the rows of the matrix is just shifting one place (to the right) at the time. Compare this with a fully connected neural network with:

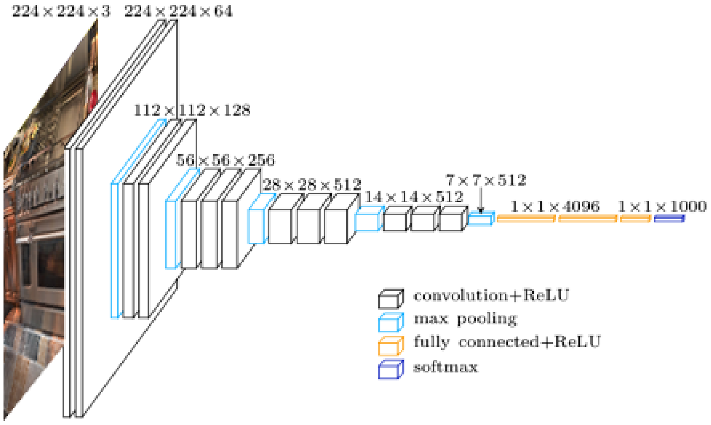
$$
\mathbf{z}^l = \mathbf{W}\mathbf{a}^{l-1} \tag{136}
$$

there is a lot less distinct parameters. Therefore, convolution delivers parameter sharing.

5.3 Look closely at CNN for image classification

firstly, let me show you a real convolution neural network:

Figure 17: VGG neural network structure



This is the famous Oxford's VGG Neural network structure. I will go through this with you in class in detail. Also please note that the word "kernel", "filter", "weight (matrix)" are used interchangeably in Deep Learning literature.

It's better if I just illustrate the steps in its order:

$$\text{Convolution} \rightarrow \text{ReLU} \rightarrow \text{max-pooling.}$$

1. **Convolution**

   You have seen a single convolution operation. However, I will discuss how to perform the overall convolution operation from one layer to the next.

   (a) between input and layer 1

   Let $\mathbf{x}$ be the input data, since it's an image, now this is a Tensor (picture it as 3-d matrix) with size $224 \times 224 \times 3$. The reason to $\times 3$ is because it is a color image and it has 3 channels (R, G, B). Let's call each of the channels $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$, each $\mathbf{x}_i$ is just a one-channel image of size 224×224. Let's call first pre-activation layer set $\{\mathbf{z}_i^{(1)}\}$, since it has 64 channels, we have $\mathbf{z}_1^{(1)}, \ldots \mathbf{z}_{64}^{(1)}$. Each of the channel is a 112×112 image.

   Then We have filters set $\mathbf{W}_{1,1}^{(1)}, \mathbf{W}_{1,2}^{(2)}, \ldots \mathbf{W}_{3,64}^{(1)}$, such chat:

$$\begin{aligned}
\mathbf{z}_1^{(1)} &= \mathbf{W}_{1,1}^{(1)} \circledast \mathbf{x}_1 + \mathbf{W}_{1,2}^{(1)} \circledast \mathbf{x}_2 + \mathbf{W}_{1,3}^{(1)} \circledast \mathbf{x}_3 \\
\mathbf{z}_2^{(1)} &= \mathbf{W}_{2,1}^{(1)} \circledast \mathbf{x}_1 + \mathbf{W}_{2,2}^{(1)} \circledast \mathbf{x}_2 + \mathbf{W}_{2,3}^{(1)} \circledast \mathbf{x}_3 \\
&\vdots \\
\mathbf{z}_{64}^{(1)} &= \mathbf{W}_{64,1}^{(1)} \circledast \mathbf{x}_1 + \mathbf{W}_{64,2}^{(1)} \circledast \mathbf{x}_2 + \mathbf{W}_{64,3}^{(1)} \circledast \mathbf{x}_3
\end{aligned} \tag{137}$$

   (b) between layer 1 and 2:

   Let me show one more example, between layer 1 and 2, i.e., $112 \times 112 \times 64 \rightarrow 112 \times 112 \times 128$

   Then We have filters set $\mathbf{W}_{1,1}^{(2)}, \mathbf{W}_{1,2}^{(2)}, \ldots \mathbf{W}_{64,128}^{(2)}$, such chat:

$$
\begin{aligned}
\mathbf{z}_1^{(2)} &= \mathbf{W}_{1,1}^{(2)} \circledast \mathbf{a}_1^{(1)} + \mathbf{W}_{1,2}^{(2)} \circledast \mathbf{a}_2^{(1)} + \mathbf{W}_{1,64}^{(2)} \circledast \mathbf{a}_{64}^{(1)} \\
\mathbf{z}_2^{(2)} &= \mathbf{W}_{2,1}^{(2)} \circledast \mathbf{a}_1^{(1)} + \mathbf{W}_{2,2}^{(2)} \circledast \mathbf{a}_2^{(1)} + \mathbf{W}_{2,64}^{(2)} \circledast \mathbf{a}_{64}^{(1)} \\
&\vdots \\
\mathbf{z}_{128}^{(2)} &= \mathbf{W}_{128,1}^{(2)} \circledast \mathbf{a}_1^{(1)} + \mathbf{W}_{128,2}^{(2)} \circledast \mathbf{a}_2^{(1)} + \mathbf{W}_{128,64}^{(2)} \circledast \mathbf{a}_{64}^{(1)}
\end{aligned}
\tag{138}
$$

2. **Re**ctified **L**inear **U**nit: **the winning ingredient**

   Remember when we discussed the vanishing gradient problem by choosing σ-function as the activation function because its gradient was too small? So how do we choose a nonlinear activation function whose gradient is neither too large nor too small? Here is the solution:

$$
\text{ReLU}(z) = \max(0, z) \tag{139}
$$

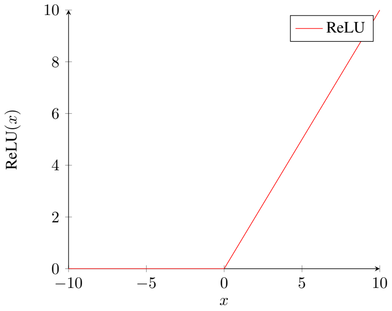


Figure 18: ReLU function

   this is just using High school mathematics, important property is that **its derivative can only be** $\{0, 1\}$. This is useful to prevent gradient vanishing and exploding. This is much better than using σ-function!

   In terms of speed, they do not require any exponential computation (such as those required in sigmoid). Reported around 6X speed over existing activation functions. Think about, all you need is some **IF** statements.

   Back to the Figure (5.3), we have:

$$
a_{i,j}^{(l)} = \text{ReLU}(z_{i,j}^{(l)}) \tag{140}
$$

3. **max pooling operation**

   this is just simply selecting the largest value, for example if we have 2×2 max pooling operation, then:

$$
\begin{bmatrix} 2.5 & 3.5 \\ 1.5 & 3.2 \end{bmatrix} \rightarrow 3.5 \qquad \begin{bmatrix} 3.5 & 9.5 & 1.7 & 6.2 \\ 1.5 & 2.5 & 1.4 & 3.2 \\ 7.5 & 6.5 & 1.5 & 5.2 \\ 2.5 & 10.0 & 2.4 & 0.1 \end{bmatrix} \rightarrow \begin{bmatrix} 9.5 & 6.2 \\ 10.0 & 5.2 \end{bmatrix} \tag{141}
$$

So you may wonder why in Figure (5.3), as you go through the layers, the image sizes becomes smaller, for example:

$$\begin{aligned} (224, 224) &\rightarrow (112, 112) \\ (112, 112) &\rightarrow (56, 56) \end{aligned} \tag{142}$$

this is because of the max-pooling operation.

5.4 CNN equation

5.4.1 Feed forward equation

let's just examine one layer only: say when we are at layer $(l-1)$: we have neurons of the previous layer $\mathbf{a}^{(l-1)}$ having size $N \times N$ neurons. Assuming that the filter $\mathbf{W}^l$ we applied is size $m \times m$.

When stride 0 is used, $\mathbf{z}^{(l)}$ will be of size $(N-m+1) \times (N-m+1)$. For simpler illustration and without the loss of generality, we shift the center by $\left(\frac{m}{2}, \frac{m}{2}\right)$, so when $(0,0)$ corner of $\mathbf{W}$ is coincides with the (i,j) position of $\mathbf{a}^{(l-1)}$, then the inner product result is placed at (i,j) position of $\mathbf{z}^{(l)}$. This means that $\mathbf{z}^{(l)}$ indices will only run between $\{0, \ldots (N-m+1)\}$ for both directions. This is illustrated in the figure below:

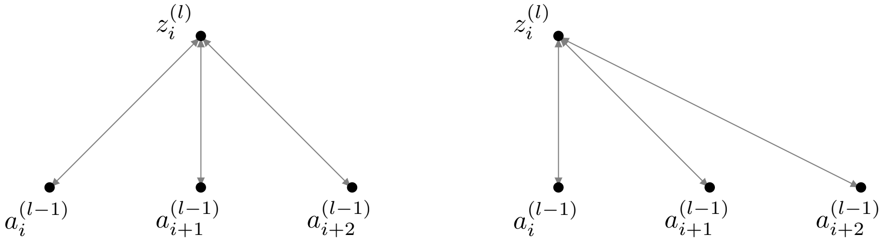


Figure 19: instead of LHS, we now use RHS

$$\begin{aligned} z_{i,j}^{(l)} &= \sum_{u=0}^{m-1} \sum_{v=0}^{m-1} W_{u,v}^{(l)} \times a_{i+u,j+v}^{(l-1)} \\ a_{i,j}^{(l)} &= \sigma(z_{i,j}^{(l)}) \end{aligned} \tag{143}$$

5.4.2 CNN back-propagation

we need four derivative quantities:

$$\begin{aligned} \frac{\partial a_{i,j}^{(l)}}{\partial z_{i,j}^{(l)}} &= \sigma'(z_{i,j}^{(l)}) \qquad & \frac{\partial z_{i,j}^{(l)}}{\partial W_{u,v}^{(l)}} &= a_{i+u,j+v}^{(l-1)} \\ \frac{\partial z_{i,j}^{(l)}}{\partial a_{i+u,j+u}^{(l-1)}} &= W_{u,v}^{(l)} \quad \implies & \frac{\partial z_{i-u,j-v}^{(l)}}{\partial a_{i,j}^{(l-1)}} &= W_{u,v}^{(l)} \end{aligned} \tag{144}$$

assume we already computed the derivative with respect to neurons at layer l : $\frac{\partial \hat{y}}{\partial a_{i,j}^{(l)}}$, then how may we proceed to compute derivative with respect to neurons at layer $l-1$?

1. find $\frac{\partial \hat{y}}{\partial a^{(l-1)}_{i,j}}$: it involves a “filter-window” of $z^{(l)}_{i,j}$

$$\begin{aligned}
\frac{\partial \hat{y}}{\partial a^{(l-1)}_{i,j}} &= \sum_{u=0}^{m-1}\sum_{v=0}^{m-1} \frac{\partial \hat{y}}{\partial z^{(l)}_{i-u,j-v}} \frac{\partial z^{(l)}_{i-u,j-v}}{\partial a^{(l-1)}_{i,j}} \\
&= \sum_{u=0}^{m-1}\sum_{v=0}^{m-1} \frac{\partial \hat{y}}{\partial a^{(l)}_{i-u,j-v}} \sigma'\left(z^{(l)}_{i-u,j-v}\right) W^{(l)}_{u,v}
\end{aligned} \tag{145}$$

   notice the above is to sum over a filter window

2. find $\frac{\partial \hat{y}}{\partial W^{(l)}_{u,v}}$: it involves entire $z^{(l)}_{i,j}$

$$\begin{aligned}
\frac{\partial \hat{y}}{\partial W^{(l)}_{u,v}} &= \sum_{i=0}^{N-m+1}\sum_{j=0}^{N-m+1} \frac{\partial \hat{y}}{\partial z^{(l)}_{i,j}} \times \frac{\partial z^{(l)}_{i,j}}{\partial W^{(l)}_{u,v}} \\
&= \sum_{i=0}^{N-m+1}\sum_{j=0}^{N-m+1} \frac{\partial \hat{y}}{\partial a^{(l)}_{i,j}} \times \sigma'\left(z^{(l)}_{i,j}\right) \times a^{(l-1)}_{i+u,j+v}
\end{aligned} \tag{146}$$

   not this is sum over an entire image.

It is important to know that although $\sigma'\left(z^{(l)}_{i,j}\right)$ appear in both terms and they can be zero. However, as long as one term in the summation is non-zero, then it will not be zero.

the rest of the back-propagation is very similar to the fully connected architecture.

5.4.3 automatic differentiation software

The good news is that nowadays, even if you don’t know how to calculate derivatives and the explicit formula for calculating backpropagation, you can still write deep learning algorithms using automatic differentiation (auto-diff) software, such as:

1. TensorFlow (Google)
2. PyTorch (Facebook)
3. Mxnet (Amazon)

With these software, all the user needs to do is specify a feed-forward deep learning architecture, then back propagation and the subsequent gradient descent is handled automatically by the software.