50.039 – Deep Learning

Alex

Week 02/03: Classification by logistic regression

1 Logistic regression

Key takeaways:

- logistic regression is a classification algorithm
- output are scores in $[0, 1]$ which can be used as a probability for a label class
- logistic regression is linear mapping with the logistic sigmoid on top
- it is an exemplary model for a single neuron
- the cross entropy loss
- crossentropy loss can be derived by maximum likelihood on top of probabilities for a predicted class

Classification goal: Given an input space $\mathcal{X}$, the goal is to predict for every sample $x \in \mathcal{X}$ to which class it belongs. For 2 class classification, the goal is: to predict in the output space $\mathcal{Y} = \{-1, +1\}$ or $\mathcal{Y} = \{0, 1\}$.

Can be generalized to C classes, then $\mathcal{Y} = \{0, 1, 2, 3, \ldots, C-1\}$. We focus on 2 classes here. So much to the question of input and output spaces. Lets go over to the question of model.

1.1 Derivation of the logistic sigmoid

How to use a linear model ? Linear model has unbounded values, we need outputs in $\{-1, +1\}$ (or $\{0, 1\}$)

$$f(x) = w \cdot x + b$$

Start from this idea for a classification mapping $f(x) = sign(g(x))$.

So if $f(x) > 0$, then we predict the label $+1$, otherwise we predict -1 ($f(x) = 0$ is a tie case). What properties?

- The classification switches at the set of points x such that $f(x) = 0$.
- additional intuition from unbounded values:

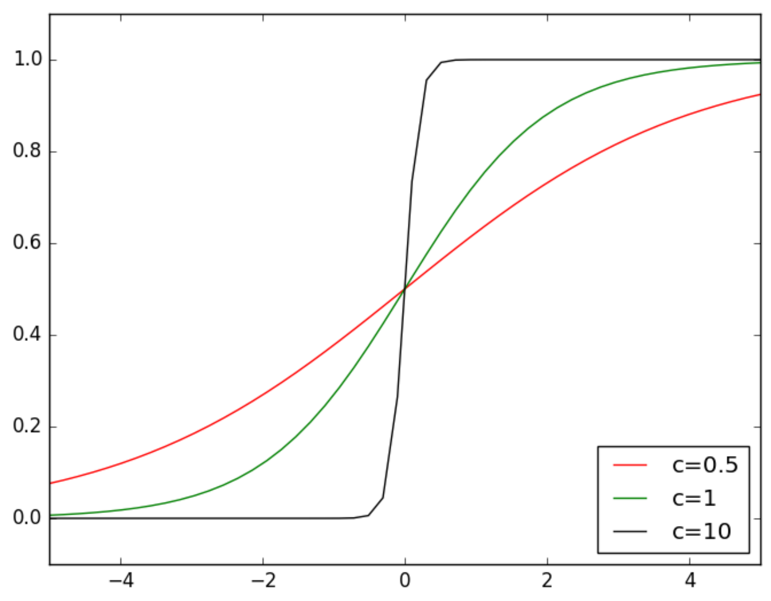
  for points x with $f(x) \approx 0$, we are not sure about the prediction.

  the larger $|f(x)|$, the more confident one should be to predict a class label.
- $f(x) >> 0$ large positive, we should be confident about the prediction $+1$
- $f(x) << 0$ large negative, we should be confident about the prediction -1

Can we encode this confidence as a probability?

- Lets map $(-\infty, +\infty)$ onto $(0, 1)$ such that 0 gets mapped onto 0.5.
- $+\infty$ should be mapped in the limit to 1
- $-\infty$ should be mapped in the limit to 0

$$\begin{aligned}
s(a) &= \frac{\exp(a)}{1+\exp(a)} = \frac{1}{\exp(-a)+1}\frac{\exp(a)}{\exp(a)} = \frac{1}{\exp(-a)+1} \\
\lim_{a\to-\infty} s(a) &= \frac{\exp(-\infty)}{1+\exp(-\infty)} = \frac{0}{1+0} = 0 \\
\lim_{a\to+\infty} s(a) &= \frac{1}{\exp(-\infty)+1} = \frac{1}{0+1} = 1
\end{aligned}$$



$s(ca) = \frac{\exp(ca)}{1+\exp(ca)} = \frac{1}{\exp(-ca)+1}$ for different values of c.

The convergence of $s(a \to -\infty) \to 0, s(a \to +\infty) \to 1$ shows that: $s(a)$ can be interpreted as a probability, expressing confidence for $P(Y = +1|X = x)$

Definition: Logistic sigmoid function

$$s(a) = \frac{\exp(a)}{1+\exp(a)} = \frac{1}{\exp(-a)+1}$$

Definition: Logistic regression model

The logistic regression model consists of plugging in the output of the linear/affine mapping $f_{w,b}(x) = w \cdot x + b$ into the logistic sigmoid function $s(a)$.

$$h(x) = s(f_{w,b}(x)) = \frac{\exp(w \cdot x + b)}{1 + \exp(w \cdot x + b)} = \frac{1}{\exp(-w \cdot x - b) + 1}$$

Its output $h(x) \in (0,1)$ can be interpreted for 2 classes as $P(Y = +1 | X = x)$ – the confidence/belief expressed as a probability that sample x has classification label $y = +1$.

Above establishes the logistic regression model. What we need is loss function and a way to optimize it.[1]
We need to define a loss function for this model. Model outputs a probability, so one can use maximum likelihood principle.

1.2 The cross-entropy loss for 2-class classification

Definition: cross-entropy loss for 2-class classification

Suppose we have for every sample x_i a function $p(x_i) = (p_1(x_i), p_2(x_i))$ which provides a probability $p_1(x_i) = P(Y = +1 | X = x_i)$ for the positive class and $p_2(x_i) = P(Y = -1 | X = x_i)$ for the negative class.

Furthermore let $\bar{y}_i = (y_i + 1)/2$ be the mapping of the labels $y_i \in \{-1, +1\}$ onto $\{0, +1\}$.
Then the crossentropy loss for a sample (x_i, y_i) is given as

$$\begin{aligned} L(p(x_i), y_i) &= \begin{cases} -\log(p_1(x_i)) & \text{if } y_i = +1 \\ -\log(p_2(x_i)) & \text{if } y_i = -1 \end{cases} \\ &= -\bar{y}_i \log(p_1(x_i)) - (1 - \bar{y}_i) \log(1 - p_1(x_i)) \end{aligned}$$

- Why these two definitions are equivalent?
- Does such a log-probability make sense as a loss?

Side note: the cross-entropy is actually defined between two probability distributions and related to the KL-divergence, which is a divergence measure between

[1] Why $s(a)$ encodes a conditional probability $P(Y = +1 | X = x)$ and not a joint probability $P(Y = +1, X = x)$?

two probability distributions.

The loss averaged over a dataset $\{(x_i, y_i), i = 1, \dots\}$ is:

$$\frac{1}{n}\sum_i -\bar{y_i}\log(p_1(x_i)) - (1-\bar{y_i})\log(1-p_1(x_i))$$

Plug in for $p_1(x_i)$ our model $h(x_i) = \frac{1}{\exp(-w\cdot x - b)+1}$. Then, the optimization problem to be solved is:

$$(w^*, b^*) = \text{argmin}_{w,b}\frac{1}{n}\sum_i -\bar{y_i}\log(h(x_i)) - (1-\bar{y_i})\log(1-h(x_i))$$

1.3 Derivation of the cross entropy loss

The roadmap of steps for the derivation:

1. the maximum likelihood principle
2. $h(x) = P(Y = +1|X = x)$, that is the logistic output $h(x)$ encodes $P(Y = +1|X = x)$
3. from there can obtain as an expression for both labels:
   $P(Y = \bar{y}|X = x) = h(x)^{\bar{y}}(1-h(x))^{1-\bar{y}}$ given that $\bar{y} \in \{0, +1\}$
   This holds because of
   $$h(x)^{\bar{y}}(1-h(x))^{1-\bar{y}} = \begin{cases} h(x) = P(Y = +1|X = x) & \text{if } \bar{y} = 1 \\ 1 - h(x) = P(Y = -1|X = x) & \text{if } \bar{y} = 0 \end{cases}$$
4. conditional independence:
   $P(Y_1 = y_1, Y_2 = y_2, \dots, Y_n = y_n|x_1, x_2, ..., x_n) =$
   $P(Y_1 = y_1|x_1) \cdot P(Y_2 = y_2|x_2) \cdot \ldots \cdot P(Y_n = y_n|x_n)$
5. plugging in above independence into the likelihood term $L(w) = P(Y_1 = y_1, Y_2 = y_2, \dots, Y_n = y_n|x_1, x_2, ..., x_n) = \prod_{i=1}^n P(Y_i = y_i|x_i)$
6. understanding the goal: want to maximize the probability of observing all labels y_i for given data points x_i, so maximize $L(w)$: $w^* = \text{argmax}_w L(w)$
7. applying neglog-transform: $w^* = \text{argmin}_w - \log(L(w))$

Recap: maximum likelihood:

Suppose one has a dataset $D_n = (z_1, \dots, z_n)$ and wants to fit *the parameter* w of a probability model P. P takes $(z_1, \dots, z_n)$ as input and produces a probability $P(z_1, \dots, z_n|w) \in (0, 1)$ for observing the dataset $(z_1, \dots, z_n)$.

The maximum likelihood principle states: Given D_n, choose the parameter w such that the probability of drawing/observing the data under $P = P_w$ is maximized, that is:

$$w^* = \text{argmax}_w P(z_1, \dots, z_n | w)$$

SideNote: maximum a-posteriori, fully bayesian methods offer often better ways to fit models.

Derivation of the loss function:

How to apply this idea? We want to maximize the joint probability of observing all the labels y_i given all the data points x_i:

$$w^* = \text{argmax}_w P(Y_1 = y_1, Y_2 = y_2, \dots, Y_n = y_n | x_1, x_2, ..., x_n)$$

How to arrive at this ?We have

- a dataset $D_n = ((x_1, y_1), \dots, (x_n, y_n))$.
- $h(x) = P(Y = +1 | X = x)$
- For likelihood we need an expression of type $P(Y = y | X = x)$. **From now on we will assume** $y_i \in \{0, 1\}$

$$h(x)^y (1 - h(x))^{1-y} = \begin{cases} h(x) = P(Y = +1 | X = x) & \text{if } y = 1 \\ 1 - h(x) = P(Y = -1 | X = x) & \text{if } y = 0 \end{cases}$$

so we have:

$$\begin{aligned} P(Y = y | X = x) &= P(Y = +1 | X = x)^y \cdot P(Y = 0 | X = x)^{1-y} \\ &= h(x)^y (1 - h(x))^{1-y} \quad \text{for } y \in \{0, 1\} \end{aligned}$$

We assume here a conditional likelihood, that is we fix our datapoints x_i, and optimize the parameter w such that the distribution over the labels y_i gets high probability. So we want to optimize:

$$P(Y_1 = y_1, Y_2 = y_2, \dots, Y_n = y_n | x_1, x_2, ..., x_n)$$

if we **assume that all samples are statistically independent**, that is $P((x_1, y_1), \dots, (x_n, y_n)) = P(x_1, y_1) \cdot P(x_2, y_2) \cdot \dots \cdot P(x_n, y_n)$, then we can get:

$$\begin{aligned}
L(w) =& P(Y_1 = y_1, Y_2 = y_2, \ldots, Y_n = y_n | x_1, x_2, ..., x_n) \\
=& \frac{P(Y_1 = y_1, Y_2 = y_2, \ldots, Y_n = y_n, x_1, x_2, ..., x_n)}{P(x_1, x_2, ..., x_n)} \\
=& \frac{P(Y_1 = y_1, x_1) P(Y_2 = y_2, x_2) \cdot \ldots \cdot P(Y_n = y_n, x_n)}{P(x_1, x_2, ..., x_n)} \\
=& \frac{P(Y_1 = y_1, x_1) P(Y_2 = y_2, x_2) \cdot \ldots \cdot P(Y_n = y_n, x_n)}{P(x_1) P(x_2) \cdot \ldots \cdot P(x_n)} \\
=& P(Y_1 = y_1 | x_1) \cdot P(Y_2 = y_2 | x_2) \cdot \ldots \cdot P(Y_n = y_n | x_n) \\
=& h(x_1)^{y_1} (1 - h(x_1))^{1-y_1} \cdot h(x_2)^{y_2} (1 - h(x_2))^{1-y_2} \cdot \ldots \cdot h(x_n)^{y_n} (1 - h(x_n))^{1-y_n} \\
=& \prod_{i=1}^{n} h(x_i)^{y_i} (1 - h(x_i))^{1-y_i}
\end{aligned}$$

Result: applying maximum likelihood, to find the parameter w, such that the probability $P(Y_1 = y_1, Y_2 = y_2, \ldots, Y_n = y_n | x_1, x_2, ..., x_n)$ is maximized is the same as maximizing the term

$$L(w) = \prod_{i=1}^{n} h(x_i)^{y_i} (1 - h(x_i))^{1-y_i}$$

with respect to parameter w.

This is a product –difficult to optimize using gradients (product rule), instead we can maximize the logarithm of this term.

- Logarithms turns products into sums.

$$log(\prod_i a_i) = \sum_i log(a_i)$$

- Logarithms preserve points minizing or maximizing a function: If a point w is a maximizer of a $L(w)$, then it is also a maximizer of $\log L(w)$, because a logarithm is a strictly monotonically growing function . That is:

$$\begin{gathered}
w^* = \text{argmax} L(w) \\
\text{so: } L(w^*) > L(w) \Leftrightarrow \log L(w^*) > \log L(w)
\end{gathered}$$

The final step: Maximizing a function f is same as minimizing -1 times this function f, so in the end: we can minimize the negative logarithm of above likelihood, in short: neg-log-likelihood. Thus our **loss function** will be the negative logarithm of the likelihood.

Goal:

$$\begin{aligned}
w^* &= \text{argmin}_w - \log\left(L(w)\right) \\
&= \text{argmin}_w - \log\left(\prod_{i=1}^{n} h(x_i)^{y_i}(1-h(x_i))^{1-y_i}\right) \\
&= \text{argmin}_w (-1)\cdot\sum_{i=1}^{n} \log\left(h(x_i)^{y_i}(1-h(x_i))^{1-y_i}\right) \\
&= \text{argmin}_w (-1)\cdot\sum_{i=1}^{n} \log\left(h(x_i)^{y_i}\right) + \log\left((1-h(x_i))^{1-y_i}\right) \\
&= \text{argmin}_w (-1)\cdot\sum_{i=1}^{n} y_i\log\left(h(x_i)\right) + (1-y_i)\log\left(1-h(x_i)\right)
\end{aligned}$$

This is exactly the sum of cross-entropy losses over our data. We only need to add an averaging term $\frac{1}{n}$ to be done. Note that this derivation shows another insight: the sum of losses over data samples is motivated by a conditional independence assumption in your set of training data $\{(x_i, y_i), i = 1, \ldots\}$.

1.4 Using the gradient of the loss

The gradient of the loss function is simple. We will use

$$\begin{aligned}
\frac{\partial log(s(a))}{\partial a} &= 1 - s(a) \\
\frac{\partial log(1-s(a))}{\partial a} &= -s(a)
\end{aligned}$$

and we know $h(x) = s(w \cdot x)$

The gradient with respect to w will be:

$$\begin{aligned}
\nabla_w L &= \nabla_w \left((-1)\cdot\sum_{i=1}^{n} y_i\log(s(w\cdot x_i)) + (1-y_i)\log\left(1-s(w\cdot x_i)\right)\right) \\
\nabla_w L &= \sum_{i=1}^{n} x_i(s(w\cdot x_i) - y_i) = \sum_{i=1}^{n} x_i(h(x_i) - y_i)
\end{aligned}$$

Observation 1:

The gradient has a nice interpretation: it is the sum of data points x_i weighted by differences between the predicted value $h(x_i) = s(w \cdot x_i)$ and the true value $y_i \in \{0, 1\}$.

Observation 2:
There is another effect that is important in the context of neural networks: the

gradient does not suffer from saturation of the logistic sigmoid. To understand this, consider the derivative of $s(a)$.

$$s'(a) = \frac{e^{-a}}{(1+e^{-a})^2} \le \begin{cases} e^{-a} & a \ge 0 \\ \frac{1}{1+e^{-a}} & a < 0 \end{cases}$$

for $|a|$ large, this derivative quickly gets exponentially small. The gradient of above loss does not.

Observation 3:
If both classes in the data can be perfectly separated by a linear mapping, then optimization will try to make the weights w to go to infinity. This is a source of instability during optimization time. Why?

- if $y_i = +1$, then the loss for the sample $(x_i, y_i = +1)$ is $\log h(x_i)$.
- $\log h(x_i) = 0$ if $h(x_i) = 1$. So trying to make the loss to zero for $y_i = 1$ means that one tries to make $h(x_i) = 1$.
- $h(x_i) = s(wx)$
- $s(z) = \frac{1}{1+e^{-z}}$ goes to 1 if $e^{-z} \to 0$, this happens if $z \to \infty$ (Aha!)
- $z \to \infty$ can be obtained by changing w!
- $h(x) = s(wx)$ converges to $0, +1$ for very large values of wx.

  So making $P(Y = y|X = x)$ to 1 for all data points – requires to push $f(x) = wx$ to very large values for all data points. This can be done by upscaling w to huge values such as $w \mapsto 10^{10}w$.

Why this does not happend when the training dataset is not linearly separable ?

- no matter how you place w, there will be always a sample x_i, y_i that is misclassified. Suppose $y_i + = 1$
- We can simulate $\|w\| \to \infty$ by using as weight $w := cw_*$ and let go $c \to \infty$. then for this sample we have

$$\text{if } y_i = +1 : \; l(x_i, y_i) = -\log h(x_i) \text{ and}$$
$$cwx_i + b < 0, \; cwx_i + b \xrightarrow{c\to\infty} -\infty$$
$$\Rightarrow h(x_i) = s(wx_i + b) \xrightarrow{c\to\infty} 0 \Rightarrow -\log h(x_i) \to -\log(0) = \infty$$

  By that the whole summed loss will go to infinity if the training dataset is not linearly separable For that reason the weight norm will not go infinity. instead it will go to some finite norm where gains from

- pushing $-\log(h(x_i))$ to small values by $h(x_i) \approx 1$ for correctly predicted samples with $y_i = +1$
- pushing $-\log(1 - h(x_i))$ to small values by $h(x_i) \approx 0$ for correctly predicted samples with $y_i = 0$

are offsetted against increasing losses by pushing $-\log(h(x_i))$, $-\log(1 - h(x_i))$ to larger values for misclassified samples

Observation 4:
What is missing here? A bias term.

$$f_{w,b}(x) = s(wx + b)$$

Can be approximately included by augmenting all your datapoints by adding an extra dimension which is constant :

$$x_i = (x_i^{(1)}, \ldots, x_i^{(d)}) \mapsto \hat{x_i} = (x_i^{(1)}, \ldots, x_i^{(d)}, 1)$$
$$(w, w^{(d+1)}) \cdot \hat{x_i} = w \cdot x_i + w^{(d+1)} \cdot 1 \sim wx + b$$

The only caveat here: any regularization term on w now has effect on the bias term, too. Zero-meaning the features makes this effect less harmful.

1.5 Solving the optimization problem

Apply the gradient descent algorithm from last lecture! Gradient is given.

Side question: why we did not exercise this for the case of linear model and hinge-loss?