50.039 – Theory and Practice of Deep Learning

Alex

Week 02: A note on generalization

[The following notes are compiled from various sources such as textbooks, lecture materials, Web resources and are shared for academic purposes only, intended for use by students registered for a specific course. In the interest of brevity, every source is not cited. The compiler of these notes gratefully acknowledges all such sources.]

Learning goals

- Give an explanation what is generalization in machine learning

1 the goal in regression (and machine learning in general!)

Find parameters w, b which generalize well to new, unseen data points.

The next sections deal all with one central problem: what do we want to achieve when we train any regression mapping, decision tree, neural net.
What does Generalize well to new, unseen data points mean???

This will need several steps

- We need a way to model *new, unseen data points*
- we define generalization in terms of an experiment in which we draw 100 or 1000 times a test dataset
- we define generalization in terms of an expected loss, where the probability used to compute the expectation of the loss is the one for drawing a test sample (x, y)

We need a way to model *new, unseen data points*

Assumption 1: We can draw test data sets T_n from your data source. A test data set consists of n pairs (x_i, y_i):

$$T_n = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} = \{(x_i, y_i), i = 1, \ldots, n\}$$

x_i is the input feature, y_i is the ground truth label to it (here: the regression value which x_i is expected to have.)

Assumption 2: We cannot predict which samples (x_i, y_i) we will obtain from your data source as new, unseen data points. Therefore, we model the uncertainty by drawing from a probability for the (x_i, y_i).

$$(x_i, y_i) \sim P_{test}.$$

An Example for such a generating probability
for input samples $x \in [0, 1] \times [0, 1] \subset \mathbb{R}^2$:

$$\begin{aligned} p(x) &= Unif([0, 1] \times [0, 1]) \\ p(y|x) &= Normal(\mu(x), \sigma^2 = 0.1) \\ \mu(x) &= 2x_1 - 3x_2 \\ P_{test}(x, y) &= p(x)p(y|x) \end{aligned}$$

Drawing in practice:

- draw $x \sim p(x)$
- draw $y \sim p(y|x)$
- this implies: have $(x, y) \sim p(x)p(y|x) = P_{test}(x, y)$

By drawing n times in this way, one can obtain a training dataset $D_n = \{(x_i, y_i), i = 1, \ldots, n\}$ or a test dataset T_n of independent samples.

Generalization for regression with l2-loss (short form)

Generalize well means in short form:

- we find parameters (w^*, b^*) defining a mapping f which gives gives predictions with low errors on new, unseen data.

In practice, the search for such parameters is done on a training dataset D_n, and involves minimizing a training loss $\hat{L}(f, D_n)$ computed on the training dataset. Usually the loss is more than just a square difference $(f(x) - y)^2$ and contains

regularization terms, or upper bounds on what we want to measure as error.

$$(w^*, b^*) = \text{argmin}_{(w,b)} \hat{L}(f, D_n) = \text{argmin}_{(w,b)} \frac{1}{n} \sum_{(x_i,y_i)\in D_n} \ell(f(x_i), y_i)$$
$$= \text{argmin}_{(w,b)} \frac{1}{n} \sum_{(x_i,y_i)\in D_n} (f(x_i) - y_i)^2$$

Generalization for regression with l2-loss – (long form: draws of test datasets)

Suppose you can draw from your data source K times test data sets $T_n^{(k)}, k = 1, \ldots, K$, each of them has n samples .

This allows to define a probability over draws of datasets in the sense: if I draw a test dataset T_n 100 times and compute a value $L(T_n)$ which depends on the test dataset , how often (among the 100 draws of test datasets) do I observe that this value is low or high?

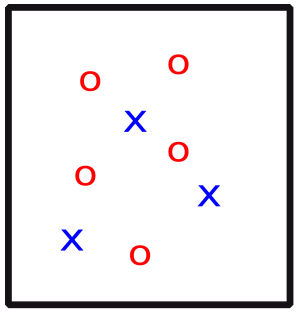
We have good generalization, if, whenever we draw a test dataset,

- with high probability (over repeating the drawing of test datasets T_n 100 times), the averaged loss $L(f, T_n^{(k)})$ on the test dataset T_n is low.

$$\hat{L}(f, T_n^{(k)}) = \frac{1}{n} \sum_{(x_i,y_i)\in T_n^{(k)}} \ell(f(x_i), y_i)$$
$$= \frac{1}{n} \sum_{(x_i,y_i)\in T_n^{(k)}} (f(x_i) - y_i)^2$$

Why is this good generalization? it means: the parameters we have learned for our predictor f give – for most tests datasets T_n – predictions which are for most samples (x_i, y_i) close to what they should be (the y_i).

Why is this hard?



In this setup we want to train a classifier which in thew whole space predicts constant label 0 (blue) or constantly label 1 (red) – only two classifiers to choose from.
In above setup, if we draw 7 training data points and 4 or more of them will be red/label 1, then we have chosen the worse performing predictor (why worse ?). For a sufficiently large test set, which error will we make on average when chosing to predict red/label 1 for the whole input space?

Which distribution governs the probability that we draw 7 training data points and 4 or more of them will be red/label 1 ? For small datasets, even for this simple problem, there is a high chance to estimate the label wrongly - due to randomness - in a way that the estimated label from the finite training data does not match well the true distribution with $P(y = 0) = 0.54$.

For any more complicated prediction problem, this will get more complex, e.g. in classification where the decision boundary can take any shape as defined by a neural network.

1.1 From draws of testsets to the expected loss

Above definition is different from what can be seen in most textbooks with some theory. Those make definitions in terms of an expected loss. Here comes the reasoning:
Above definition will hold, if we can find a mapping f^* (or parameters (w^*, b^*) which define a mapping f^*) which achieves a low **expected** loss under the probability P_{test} of the test samples.

$$E_{(x,y)\sim P_{test}}[\ell(f^*(x), y)] \rightarrow \text{low}$$

Why?

If the expectation $E_{(x,y)\sim P_{test}}[\ell(f^*(x), y)]$ is low, then by the **law of large numbers**, for most datasets of size n the averaged loss on the dataset will be close to that expectation,

$$\frac{1}{n} \sum_{(x_i,y_i)\in T_n} \ell(f^*(x_i), y_i) \approx E_{(x,y)\sim P_{test}}[\ell(f^*(x), y)]$$

and therefore low as well!

What means low ? How to define that the expected loss is **low**? One way is to compare it to the losses of other f taken from some class of functions $\mathcal{F}$:

A loss is low, when it is close to the lowest loss from all mappings f contained in a class $\mathcal{F}$ (all neural nets, all regression trees, all polynomials of degree at most 2 , and so on). This lowest possible loss can be written down as:

$$\min_{f\in\mathcal{F}} E_{(x,y)\sim P_{test}}[\ell(f(x), y)]$$

Example: if we consider $\mathcal{F}$ to be the class of all possible linear regression functions, how to describe $\mathcal{F}$ formally?

Above gives a simpler way to define generalization:

Generalization for any loss ℓ – (long form: expectation)

We have good generalization for our mapping f^*, if:

- the expectation of the under the probability P_{test} of the test samples is low. This can be described in the following way: the expected loss of f^* is close to the lowest possible loss over all mappings of some function class $\mathcal{F}$:

$$E_{(x,y)\sim P_{test}}[\ell(f^*(x), y)] \approx \min_{f\in\mathcal{F}} E_{(x,y)\sim P_{test}}[\ell(f(x), y)]$$

This does not use probabilities over draws of datasets.

It is simpler than imagining 100 draws of test datasets. However, for most real problems (e.g. classifying images which you receive from customers; classifying images which are uploaded by users on a server), **this expectation cannot be computed at all**, while 100 test datasets can be drawn.

One can express the approximation

$$E_{(x,y)\sim P_{test}}[\ell(f^*(x), y)] \approx \min_{f\in\mathcal{F}} E_{(x,y)\sim P_{test}}[\ell(f(x), y)]$$

more formally by requiring these two quantities to be ϵ-close:

$$|E_{(x,y)\sim P_{test}}[\ell(f^*(x), y)] - \min_{f\in\mathcal{F}} E_{(x,y)\sim P_{test}}[\ell(f(x), y)]| < \epsilon$$

2 How do these optimal functions look like ?

Need to make assumptions on $\mathcal{F}$, $P(x, y)$, the loss used.

2.1 The optimal solution in an exemplary regression setting

optimal...

- Assume the data source for (x, y) is such that there exists a function $g(x)$ such that we have:

  $$y = g(x) + \epsilon, \epsilon \sim N(0, \sigma^2)$$

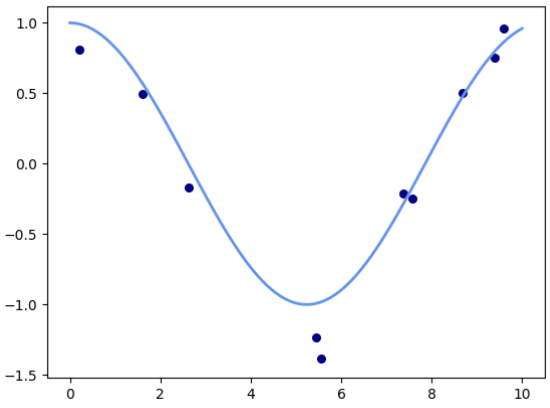
  , where $N(0, \sigma^2)$ is zero mean gaussian noise.
  Note that this specifies $P(y|x) = N(g(x), \sigma^2)$ but it makes no assumption on $p(x)$.

- assume that the loss function is MSE loss

- then the optimal function is the mean generator $g(\cdot)$:

  $$\operatorname*{argmin}_{\hat{y}} E_{y\sim P_{test}(y|x)}[(\hat{y} - y)^2] = g(x)$$
  $$\operatorname*{argmin}_{f} E_{(x,y)\sim P_{test}}[(f(x) - y)^2] = g(\cdot)$$

Note: g is not guaranteed to lie in the $\mathcal{F}$ which you are using for learning (e.g. the set of all piece-wise linear functions). Next an example of a toy problem.



The data was generated as:

$$y = \cos(0.6x) + 0.2\epsilon, \epsilon \sim N(0, 1)$$

For this case , the optimal solution that one wants to find by learning from data is $g(x) = \cos(0.6x)$, which is the blue line.

2.2 The optimal solution in an exemplary classification setting

optimal...

- Assume that the loss used is zero-one loss $1[f(x) \neq y]$. We want to find the predictor $f(x)$ of a class label for the problem

$$\min_{f(x)} E_{y \sim P_{test}(y|x)}[1[f(x) \neq y]]$$

- then the optimal solution in input point x is to predict the class label c^* defined as:

$$f(x) = c^* = \underset{c}{\operatorname{argmax}} P(y = c|X = x)$$

To see this, consider a two class problem. Let us consider what **expected loss** do we make in point x when we choose $f(x) = +1$ for the loss function $1[f(x) \neq y]$?

$P(y|x)$ is discrete: Y can take only $\{-1, +1\}$. We know

$$\begin{aligned} E_{(x,y)\sim P(x,y)}[I[\underbrace{+1}_{=f(x)} \neq y]|x] &= I[1 \neq 1]P(y = +1|x) \\ &\quad + I[1 \neq -1]P(y = -1|x) \\ &= 0 + P(y = -1|x) \end{aligned}$$

Result: For $f(x) = +1$ the loss will be $P(y = -1|x)$.

Let us consider what **expected loss** do we make in point x when we choose $f(x) = -1$ for the loss function $1[f(x) \neq y]$?

By the same steps as above, the expected loss will be

$$E[I[W - 1 \neq y]|x] = P(y = +1|x)$$

So:

$$E[I[f(x) \neq y]|x] = \begin{cases} P(y = -1|x) & \text{if } f(x) = +1 \\ P(y = +1|x) & \text{if } f(x) = -1 \end{cases}$$

How to minimize our loss ? The loss is either, $P(y = -1|x)$ or $P(y = +1|x)$. Therefore we choose as prediction

$$f_{opt}(x) = \text{argmax}_y P(y|x)$$

, because then the loss will be for the case of two classes $\text{argmin}_y P(y|x)$

- $f_{opt}(x) = \text{argmax}_y P(y|x)$ does not need to be a member of $\mathcal{F}$! – example are $\mathcal{F}$ = decision trees up to some depth for a non-aligned decision boundary
- Note: if $P(y = +1|x) \neq \{0, 1\}$, then we will always have a non-zero expected loss, no matter what prediction we choose. Loss zero under P is often impossible – thats why predictions need a human in the loop often.

takeaway I:

If we know the generating distribution $P(y|x)$, and optimize directly for it, then there is no overfitting. Overfitting occurs bcs we optimize for a finite training set D_n, which is does not contain all information about the generating distribution $P(y|x)$

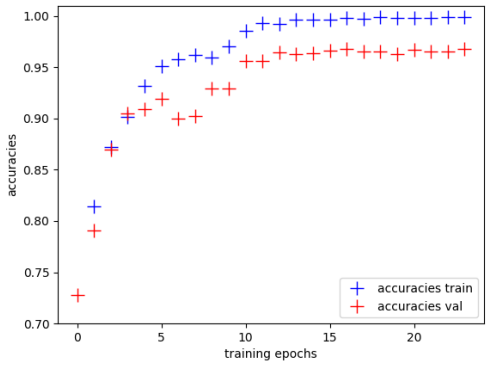
2.3 What is overfitting?

Phenomenological definition of overfitting

We have overfitting whenever we observe:

- training loss (on train set) $<$ validation loss (on val set)
- training score $>$ validation score

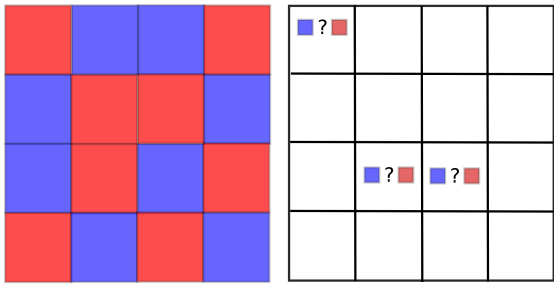
In systems that allow to observe training and validation performance over training steps, one can see a gap opening up over time



Accuracies. Blue: Training. Red: Validation. 102 Flowers Dataset training of a resnet

2.4 Why more data helps when the function class $\mathcal{F}$ gets more large and more complex?

Assume we have in the blue squares $P(y = 0|X = x) = 0.8$ and in the red squares $P(y = 1|X = x) = 0.8$, and the classifier is a decision tree which predicts a constant label in every square



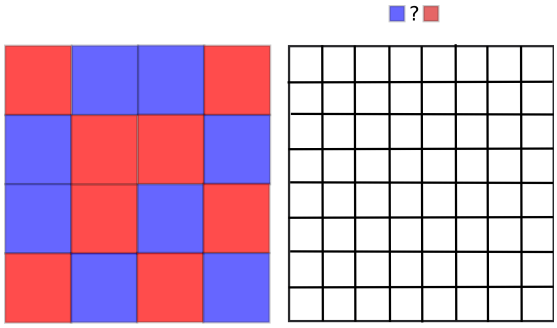
Then you may need in every square 50 data points to achieve a reliable estimate of the prediction label in every bin. Why?

If y_i are drawn i.i.d., then we know that

$$\frac{1}{n}\sum_{i=1}^{n} 1[y_i = 1] \approx N(p, \sigma^2 = \frac{p(1-p)}{n})$$

So for $n \approx 50$ the variance will be sufficiently small, and the normal approximation of sufficiently good quality.

Now consider the case when your classifier has more complexity as you need, and we increase the complexity: You use a decision tree with 64 bins instead of the necessary 16 bins.



Then: if the number of training data is keepts the same as for the case above with 16 bins, then – for this more complex classifier with 64 bins– you will learn in every bin to predict the label using only 1/4 of the data. Less data per bin to estimate ⇒ Higher probability to predict the wrong label in one of the bins. So more likely to overfit.

To obtain the same estimation quality in every bin, you will need 4 times more data.

takeaway II:

Suppose we are using the same number of data. Then more complex classifiers – have a higher chance of locally predicting a wrong label, as compared to more simple classifiers.

Alternatively: to keep overfitting withhin bounds, you will need more data for more complex classifiers.

This is because more complex classifiers need more data to estimate whether they predict correctly.