50.021 – AI

Week 03: Overfitting

[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.]

1 Supervised Discriminative Machine Learning

Learning goals for today

- being able to reproduce the basic philosophy of discriminative machine learning
- being able to execute the steps needed to set up a discriminative machine learning task
- Discriminative Learning problems consist usually of a least four basic components
  1. Input space $\mathcal{X}$ and output space $\mathcal{Y}$
  2. the prediction mapping connecting these $f : \mathcal{X} \to \mathcal{Y}$
  3. a loss function
  4. a method to select a prediction mapping f / or its parameters from a dataset
- the data to be expected to be seen is modeled by a probability distribution
- training and testing data are finites samples from it
- Learning $\approx$ finding a good predictive mapping f of an input space to an output space.
- *good* mapping means: one that minimizes an expectation of a loss function. The expectation of the loss is based on a probability of the data expected to be seen at application time.
- that probability distribution is unknown, but the expected loss can be approximated using training and testing data sets
- occams razor for selecting models

Lets look at some examples to understand about Input space $\mathcal{X}$ and output space $\mathcal{Y}$

- Classification of Images

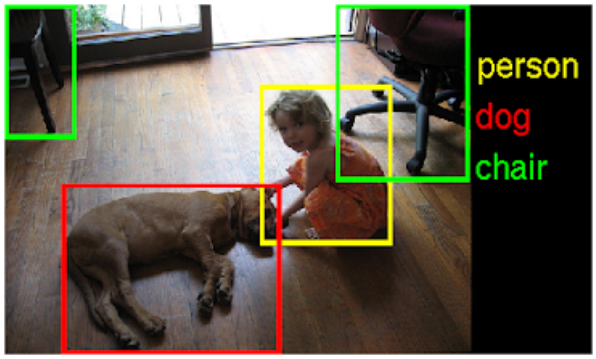
```
top 10 classes and their scores: 670 | n03791053 motor scoo 0.636128
top 10 classes and their scores: 880 | n04509417 unicycle,  0.261106
top 10 classes and their scores: 665 | n03785016 moped 0.0432479
top 10 classes and their scores: 573 | n03444034 go-kart 0.0156194
top 10 classes and their scores: 444 | n02835271 bicycle-bu 0.00917528
top 10 classes and their scores: 920 | n06874185 traffic li 0.00490618
top 10 classes and their scores: 518 | n03127747 crash helm 0.00412505
top 10 classes and their scores: 822 | n04311174 steel drum 0.0024649
top 10 classes and their scores: 733 | n03976657 pole 0.00223075
top 10 classes and their scores: 557 | n03355925 flagpole,  0.00199626
```

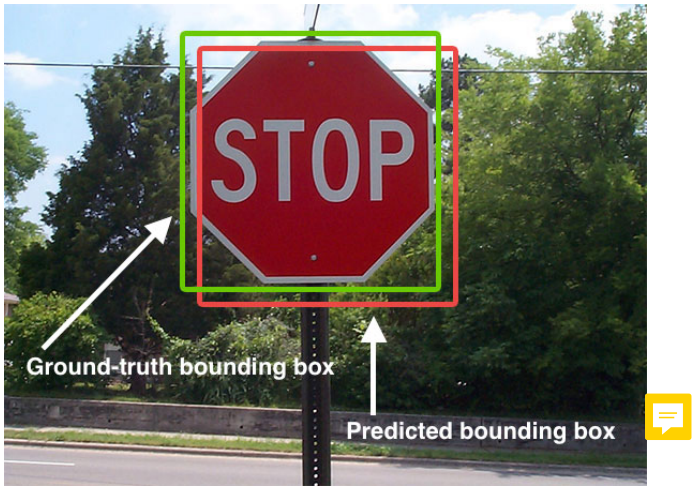
  - input: image (how to represent it?)
  - output: concept label

- Classification of sounds
  - input: sound recording (how to represent it?)
  - output: concept label

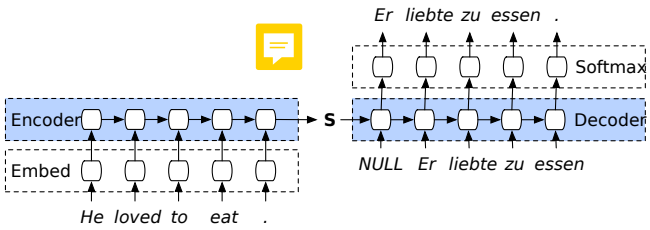
- Regression of blood pressure based on amount of food, percentage of fibers, sugars,fats, proteins
  - input: ?
  - output: ?

- Detection of a objects in images
  - input: image



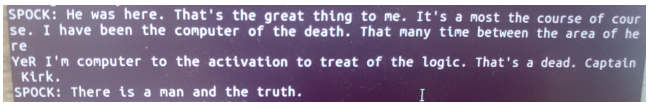


  - output: ?
- Machine Translation
  - input: (how to represent it?)



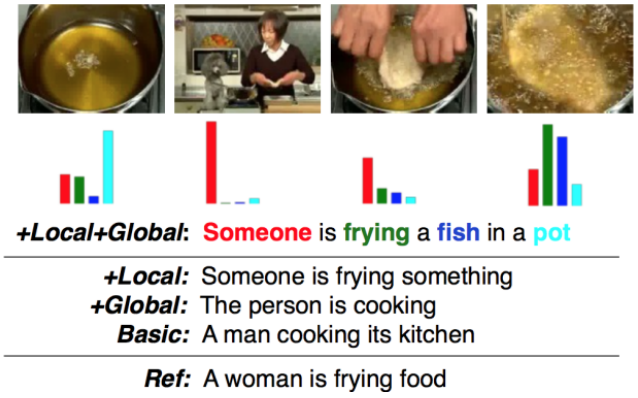
(from smerity.com)

  - output: ?
- Text Summarization
  - input: sequence of words $(w_1, \ldots, w_K)$ same as above



  - output: ?

- Video captioning
  - input: ?



(from nvidia)

  - output: ?

Always the same structure:

- x – data sample, from an input space $\mathcal{X}$ get mapped onto an output y in some space $\mathcal{Y}$
- Prediction by a mapping $f : \mathcal{X} \to \mathcal{Y}$, $f(x) = y \in \mathcal{Y}$.

Usually the first task in designing any learning system is to think how to describe the input and output spaces $\mathcal{X}, \mathcal{Y}$

Assumptions for discriminative learning setups

Assumption 1: Supervised Learning
Have a dataset for which we know for every input sample what prediction is desired:

- x – an input sample
- y – its ground truth prediction .
- a data sample/point comes as a pair (x, y)

Assumption 2: Loss function
able to define/measure a loss $l(f(x), y)$ for how wrong our prediction $f(x)$ is relative to its ground truth y.
(is groundtruth always unique??)

Example for a loss function – classification with C classes.
Output space $\mathcal{Y} = \{0, \ldots, C-1\}$.

Classification is wrong on (x, y) if $f(x) \neq y$

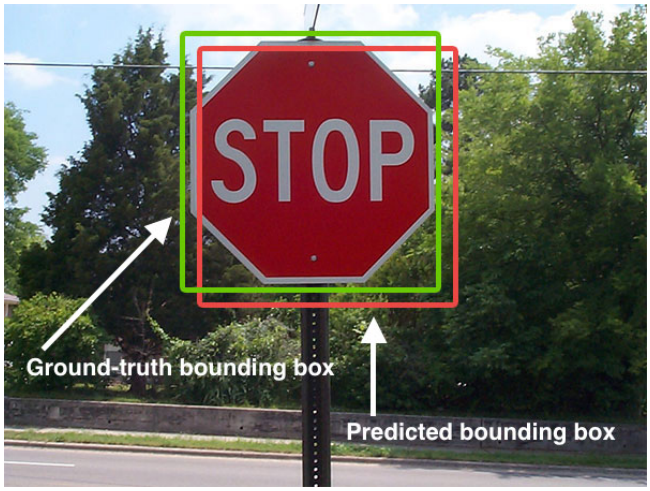
$$L(f(x), y) = 1[f(x) \neq y] = \begin{cases} 0 & \text{if } f(x) == y \\ 1 & \text{if } f(x) \neq y \end{cases}$$

Example for a loss function – Regression

$$L(f(x), y) = (f(x) - y)^2$$

is 0 if prediction exactly matches the regression groundtruth.

Example for a loss function – Detection:



Intersection over union

$$IoU(gt, pred) = \frac{Area(gt \cap pred)}{Area(gt \cup pred)}$$

Area of intersection of predicted vs true bounding box, divided by area of union of those.

1.1 What goal to achieve for the prediction? Part 1

We have

- know how to describe input $\mathcal{X}$ and output $\mathcal{Y}$ spaces
- have a class of functions $f : \mathcal{X} \to \mathcal{Y}$, $f(x) = y$.
- loss function $L(f(x), y)$ for measuring disagreement $f(x) \leftrightarrow y$ (prediction vs ground truth)

How to select/learn a good mapping f?

When we deploy the mapping f at test time after training/research, it should *perform well* when deployed.

- The unseen data (x, y) is uncertain. Model uncertainty by assume data coming from a probability distribution:

$$\text{Test/Deployment time: } (x, y) \sim P_{test}$$

- Now reformulate goal:
  want a predictor which is good on average on data $(x, y) \sim P_{test}$,

  therefore: seek a mapping f^*

$$f^* \approx \underset{f \in \mathcal{F}}{\operatorname{argmin}} E_{(x,y)\sim P_{test}}[L(f(x), y)]$$

The goal of discriminative learning

Choose the model f^* such that it has low loss $L(f(x), y)$ between predictions $f(x)$ and ground truth y on average under the test time/deployment time probability distribution.

This average under the test probability distribution is the expectation $E_{(x,y)\sim P_{test}}[\cdot]$:

$$E_{(x,y)\sim P_{test}}[L(f(x), y)] \rightarrow \text{ minimal}$$
$$f^* \approx \operatorname{argmin}_f E_{(x,y)\sim P_{test}}[L(f(x), y)]$$

recap Expectation:

expectation is an average of function values over all your data

$E_{X\sim P}[t(X)] = \sum_{x \in Values} t(x)p(x)$ if X takes countable (infinite or finite) number of values

$E_{X\sim P}[t(X)] = \int t(x)f(x)dx$ if P follows a density defined by $f(x)$

A problem

usually we do not have access to a definition of P_{test}!
Cannot compute $E_{(x,y)\sim P_{test}}[L(f(x), y)]$

We have an approximate measurement of the above by the averaged error on a validation or test set T_l:

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

The Approximation becomes better as l gets larger.

Why this works? Law of large numbers / Central Limit Theorem. If we have drawn an i.i.d. dataset of samples $T_n = \{(x_i, y_i), i = 1, \ldots, n\}$ **independently and identically distributed** from P_{test}, then

$$\frac{1}{n} \sum_{(x_i,y_i)\in T_n}^{n} L(f(x_i), y_i) \overset{n\to\infty}{\longrightarrow} E_{(x,y)\sim P_{test}}[L(f(x), y)]$$

We are able to compute approximations on test data T_l and training data D_n

$$R(f, L, T_l) = \frac{1}{l} \sum_{(x_i,y_i)\in T_l}^{n} L(f(x_i), y_i)$$

$$R(f, L, D_n) = \frac{1}{n} \sum_{(x_i,y_i)\in D_n}^{n} L(f(x_i), y_i)$$

basic idea for machine learning

- draw a training dataset D_n from the test time distribution P_{test}
- select a mapping f^* such that

$$f^* = \text{argmin}_f \frac{1}{n} \sum_{(x_i,y_i)\in D_n}^{n} L(f(x_i), y_i) = \text{argmin}_f R(f, L, D_n)$$

This is called empirical risk minimization

After we have selected a mapping f, we need to report its performance.

1.2 How to measure the performance of the selected mapping?

Use a evaluation/testing dataset T_n **disjoint (!!!) from the training data** D_n . Measure empirical risk $R(f, L, T_n)$.

It is important: one cannot reliably evaluate the performance on the data D_n used for training. T_n must be disjoint from D_n for reliable performance estimates.

performance measurement for machine learning

- draw a test dataset T_n from the test time distribution P_{test} **disjoint** to the training dataset D_n
- report for the selected mapping f^*

$$R(f^*, L, T_n)$$

It should be noted here: sometimes one uses a different loss L for training than what is used to report performance at test time. This is because some criteria used at test time are not differentiable, and therefore cannot be used with optimization toolboxes.

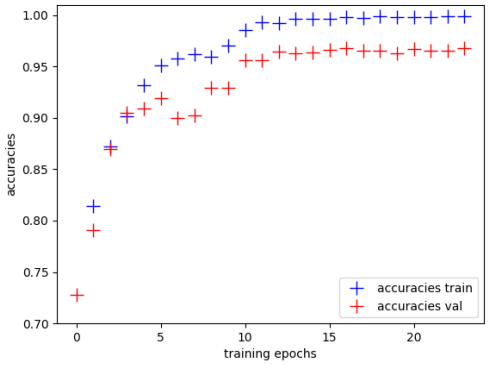
1.3 What is overfitting?

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 Aspects of the nature of overfitting

In order to understand what overfitting is, we need to know more precise how a loss minizing solution looks like – in simplified settings. We will consider in two case studies what are desirable solutions for predictors.

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:

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

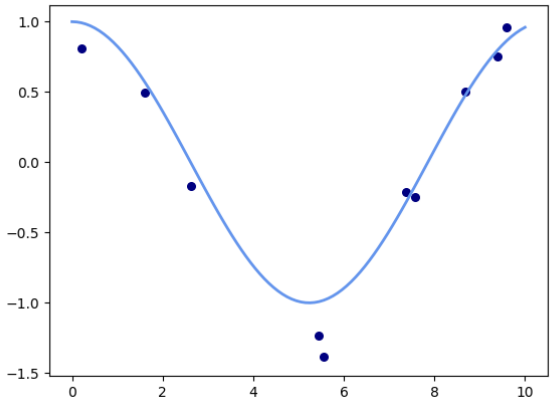
, where $N(0, \sigma^2)$ is zero mean gaussian noise.
Note that this specifies $P(y|x) = N(g(x), \epsilon)$ but it makes no assumption on $p(x)$.

- assume that the loss function is MSE loss
- then the optimal function is the mean generator $g(\cdot)$:

$$\underset{\hat{y}}{\text{argmin}}\, E_{y \sim P_{test}(y|x)}[(\hat{y} - y)^2] = g(x)$$

$$\underset{f}{\text{argmin}}\, E_{(x,y) \sim P_{test}}[(f(x) - y)^2] = g(\cdot)$$

Here an example of a toy problem.



The data was generated as:

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

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

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

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

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

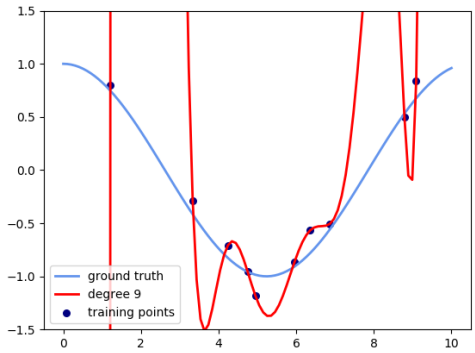
takeaway I:

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 Case study: regression – the impact of function class complexity

We see here one fundamental reason for overfitting: the expected loss is measured on the whole space $X \times Y$, while the regression predictor is learned over a finite set of training data points.

You do not need much noise in the data to overfit – when the function class makes poor assumptions how to transfer knowledge learned from the finite set of training data D_n to the whole rest of the space $X \times Y$.



9th order polynomial. zero training set loss. $\epsilon \sim N(0, 0.1^2)$

takeaway II:

Overfitting can occur due to: A. a bit of noise and B. a function class which makes poor assumptions on how to extrapolate knowledge learned from the finite set of training data D_n to the whole rest of the space $X \times Y$.

How to measure the proneness of a function class to overfitting ? Measure complexity. Complexity here: How fast the members of the function class $\mathcal{F}$ = { order 9 polynomials} change around the training data points. Example of complexity measures are:

- the polynomial order for polynomials
- $\int_x |f''(x)|dx$ – averaged absolute value of second order derivature

2.4 Case study: classification – the impact of noise in your training data

Consider at first very simple case:

$$X = [0,1] \times [0,1]$$
$$P(Y = 1 | x = x) = 0.75 \; \forall x$$

We consider a very simple classifier model: $f(x) = 1$ everywhere or $f(x) = 0$ everywhere.

The optimal prediction is $f(x) = 1$, which results in an average error of 0.25.
Lets see how much is the probability to learn a wrong prediction, when we have different number of training samples?

takeaway III:

Overfitting in classification can mean to predict the wrong label (locally) based on what one sees in the finite training data set.

- Suppose we have only one data sample, then the probability to learn the wrong prediction is 0.25.
- If we have 3 data samples, then the probability to learn the wrong prediction is the probability to draw at least 2 out of 3 training data samples. This is binomially distributed with probability

$$P(\sum_{i=1}^{3} 1[y_i = 0] \geq 2) = (1 - 0.75)^3 + \binom{3}{2}(1 - 0.75)^2 = 0.25^3 + 3 \cdot 0.25^2$$
$$= \frac{13}{64} < \frac{16}{64} = 0.25$$

  The message is: given 3 data points it is less likely to learn the wrong prediction than with 1.

- What happens for more training data points, for a larger size of n data points? We make a wrong prediction if

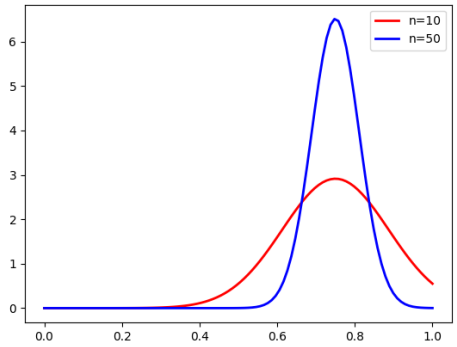
$$\frac{1}{n} \sum_{i=1}^{n} 1[y_i = 1] < 0.5$$

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

$$\frac{1}{n} \sum_{i=1}^{n} 1[y_i = 1] \approx N(0.75, \sigma^2 = \frac{0.75(1 - 0.75)}{n})$$

In general the frequencies $\frac{1}{n}\sum_{i=1}^{n} 1[y_i = 0]$ of a binomial distribution with success probability p converge to $N(p, \sigma^2 = \frac{p(1-p)}{n})$

This is a normal distribution, where the variance $\frac{0.75(1-0.75)}{n}$ decreases to 0 as n increases.



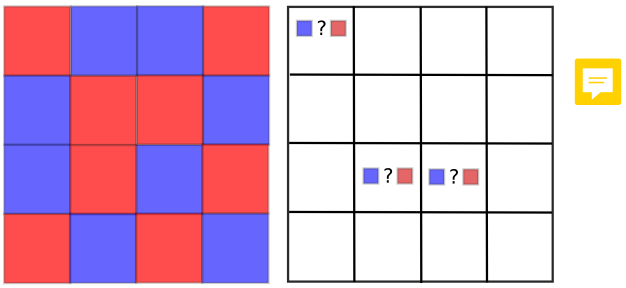
Therefore the probability mass will be concentrated around 0.75 (which is larger than 0.5) and the probability in the tail of $P(X < 0.5), X \sim N(0.75, \sigma^2 = \frac{0.75(1-0.75)}{n})$ goes straight to zero (see Figure above)

Can also consider the example of two gaussians, and data drawn from the lower half of one and the upper half of the other ... this may suggest a wrong classifier in the absence of label noise – when only of spatial sampling randomness is present.

takeaway IV:

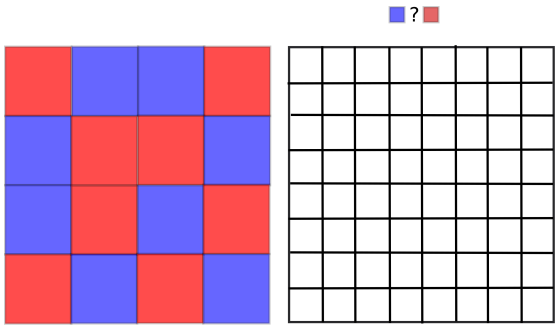
Takeaway: under noise, more data helps to reduce the probability of overfitting.

The same reasoning applies locally for more complex decision boundaries.
Below 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 need in every square 50 data points to achieve a similarly realiable estimate as in the case of one square before, so you need 16 times more data.

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 keeps 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 $\Rightarrow$ Higher probability to predict the wrong label in one of the bins. So more likely to overfit.

takeaway V:

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.

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

2.5 Why we have to evaluate the performance on a testset which is disjoint from the training set?

We have seen above: we can get a low loss on training data, but high loss on validation/test data. Losses on training data are an unreliable measurement because we optimize for the same training data (and the result can be not representative for unseen data). This explains the golden why train/val/test set must be strictly disjoint.

2.6 How to reduce overfitting?

A million of options.

Model selection: Different function classes with increasing complexity and model selection on a validation set for the best one.

- e.g. polynomials try out: polynomials up to degree 2, polynomials up to degree 3, ... polynomials up to degree n, everytime measure validation performance. select best model on the validation dataset **(not the testset !!)**
- decision trees: up to 4 leaves, 16 leaves, 64 leaves, 255 leaves ... everytime measure validation performance. select best model on the validation dataset

- general neural nets: learning rate, batchsize, different optimizers + data augmentation parameters
- Recurrent neural nets: dimensionality of the hidden state vector
- convolutional neural nets: number of output channels, spatial kernel size. lower choices for both result in a model with less parameters, which thus tends to overfit less

The example with the polynomials gives one weak principle of selecting prediction models, known as Occam's razor `https://simple.wikipedia.org/wiki/William_of_Ockham`.

Occam's razor as a rule of thumb / weak guideline

Among two hypotheses, which both explain an observation well, the simpler is preferable.

The idea behind: the more assumptions, the more likely one of them does not apply.

Weight regularization: Keep weights small (+ model selection how much to penalize).

- ridge regression: different regularization constants λ and model selection for the best λ:

$$\min_w ||wX - y||^2 + \lambda ||w||_2^2$$

- LASSO regression: different regularization constants λ and model selection for the best λ:

$$\min_w ||wX - y||^2 + \lambda ||w||_1^1$$

- SVM: different regularization constants C and model selection for the best C

$$\begin{aligned} &\operatorname*{argmin}_w \frac{1}{2}||w||_2^2 + C \sum_i \max(0, 1 - y_i(w \cdot x_i + b)) \\ =&\operatorname*{argmin}_w \frac{1}{2C}||w||_2^2 + \sum_i \max(0, 1 - y_i(w \cdot x_i + b)) \end{aligned}$$

- neural networks: weight decay (towards zero) by β

$$\begin{aligned} w_{t+1} = \quad & w_t - \eta \nabla_w L(w_t) \\ \rightsquigarrow w_{t+1} = \quad & w_t(1 - \beta) - \eta \nabla_w L(w_t) \end{aligned}$$

Look at subsets of the dimensions or data(+ model selection how much to ...).

A common strategy when training a decision forest, a set of decision trees

- train n models, each on a random subset of dimensions of the input features
- average those n models
- Aim: to reduce spurious correlations.

Add noise (+ model selection how much to ...).

Inject noise or randomly set inputs to a constant. Aim: to reduce spurious correlations.

- add gaussian noise to input features $\tilde{x}_d := x_d + \epsilon, \epsilon \sim N(0, \sigma^2)$ or to intermediate feature maps in neural networks
- dropout layer in neural networks (more later): set feature map dimensions randomly to zero during training (compare to decision trees above)

Neural nets: improve gradient flow.

- Batch Normalization
- Residual connections
- recurrent nets: use LSTM, Gated recurrent units, temporal convolutional NNs

Will be shown in another lecture.

Neural nets: use good initializations from other datasets: transfer learning.

Will be shown in another lecture.

Neural nets and many other predictors: use data augmentation at training time (and at validation / test time).

Will be shown in another lecture.

Neural nets: Early stopping of training + model selection on validation.

Simple idea: save model every n-th epoch. The model selection parameter is the number of epochs the model was trained.

More data is the best

This is what you usually will never get.

2.7 Machine learning steps in practice:

In practice one often has not only to learn a mapping but also select from a set of hyperparameters such as data preprocessing choices or any kind of algorithm parameters which are not part of the main training process. This makes it necessary to split the available data into training data, validation data and testing data.

Machine learning steps in practice

1– get clarity on input space and output space

2– choose a suitable loss function L (predicted outputs vs ground truth outputs)

3– choose a class of mappings used to predict outputs from inputs (e.g. recurrent neural networks for sequential inputs)

4– choose an algorithm to select a mapping based on training data, that is a way how to compute $f_h = \text{argmin}_f R(f, L, D_n)$

Often one starts not with a given train and test dataset, but just with one dataset, which must be splitted into training, validation and test set.

5– split your dataset into training D_n/validation V_m/test data T_l (use nested crossvalidation if too small sample sizes)

6– get clarity what hyperparameters h will need to be optimized (hyperparameters can be for example regularization parameters, or choices of preprocessing steps or loss functions)

7– for every choice h of hyperparameters:

7–1– train on your training dataset D_n: $f_h = \text{argmin}_f R(f, L, D_n)$

7–2– measure performance on validation dataset $R(f_h, L, V_m)$

Now we have for every choice h of hyperparameters a trained mapping f_h and its performance on validation data $R(f_h, L, V_m)$

8– select best choice of hyperparameters h^* on the validation dataset: $h^* = \text{argmin}_h R(f_h, L, D_n)$

9– train final model on training dataset D_n or on united train and validation dataset $D_n \cup V_m$ using the best choice of hyperparameters h^*: $f^*_{h^*} = \text{argmin}_{f_{h^*}} R(f_{h^*}, L, D_n \cup V_m)$

10– report performance of final model on test set $R(f^*_{h^*}, L, T_l)$

Pitfalls:

- inner/outer crossvalidation for small sample sizes
- only evaluate the final model on test set, never optimize hyperparameters on test set T_l - why?

- what can happen if one class has only 1% abundance, and you do a random split 70-30? - **think what statistical properties to ensure when splitting data**

two main challenges in machine learning

- the training dataset comes from a different distribution, not from the test time distribution P_{test}
- initially, during research, the training dataset comes the test time distribution P_{test}, but P_{test} changes over time