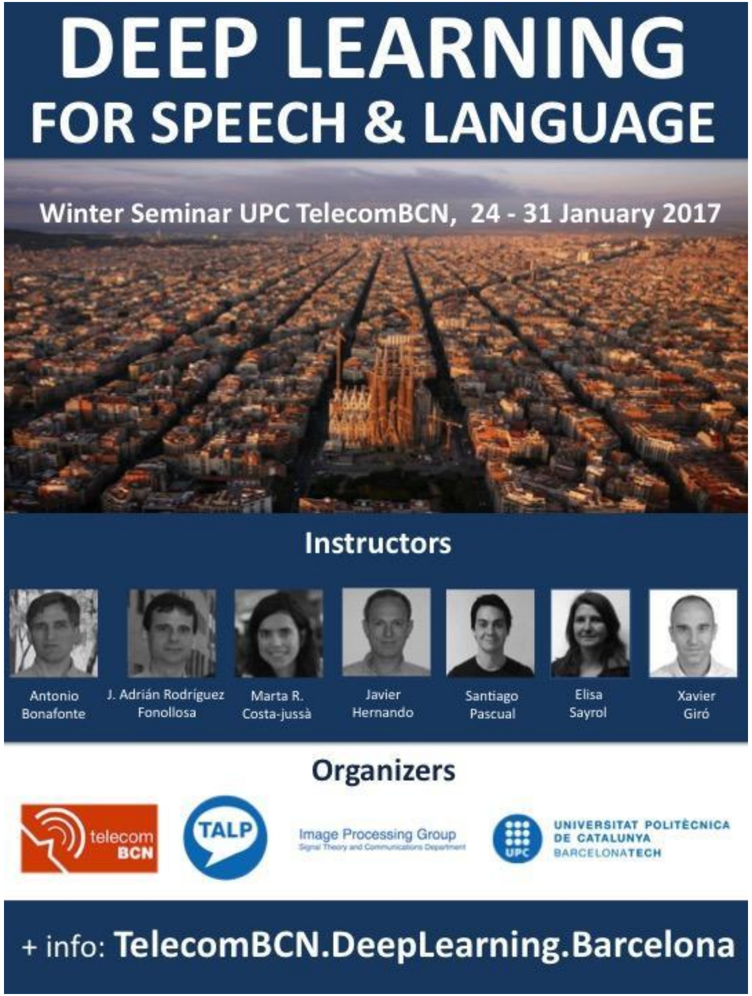


[course site]

Day 1 Lecture 6

Training Deep Networks

Elisa Sayrol

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Department of Signal Theory
and Communications
Image Processing Group

Overview

Training Set

Training

Overfitting/Underfitting

Regularization

Hyperparamenters

Slide credit: Eva Mohedano

Training set

$$\{(\mathbf{x}_i, y_i) : 1 \leq i \leq N\}$$

$$X = \underbrace{\begin{bmatrix} 2.1 & 3.2 & 4.8 & 0.1 & 0.0 & 2.6 \\ 3.1 & 1.4 & 2.5 & 0.2 & 1.0 & 2.0 \\ 1.0 & 2.3 & 3.2 & 9.3 & 6.4 & 0.3 \\ 2.0 & 5.0 & 3.2 & 1.0 & 6.9 & 9.1 \\ 9.0 & 3.5 & 5.4 & 5.5 & 3.2 & 1.0 \end{bmatrix}}_{D \text{ features (columns)}} \Bigg\} \; N \text{ training examples (rows)}$$

$\mathbf{x}_3^T$

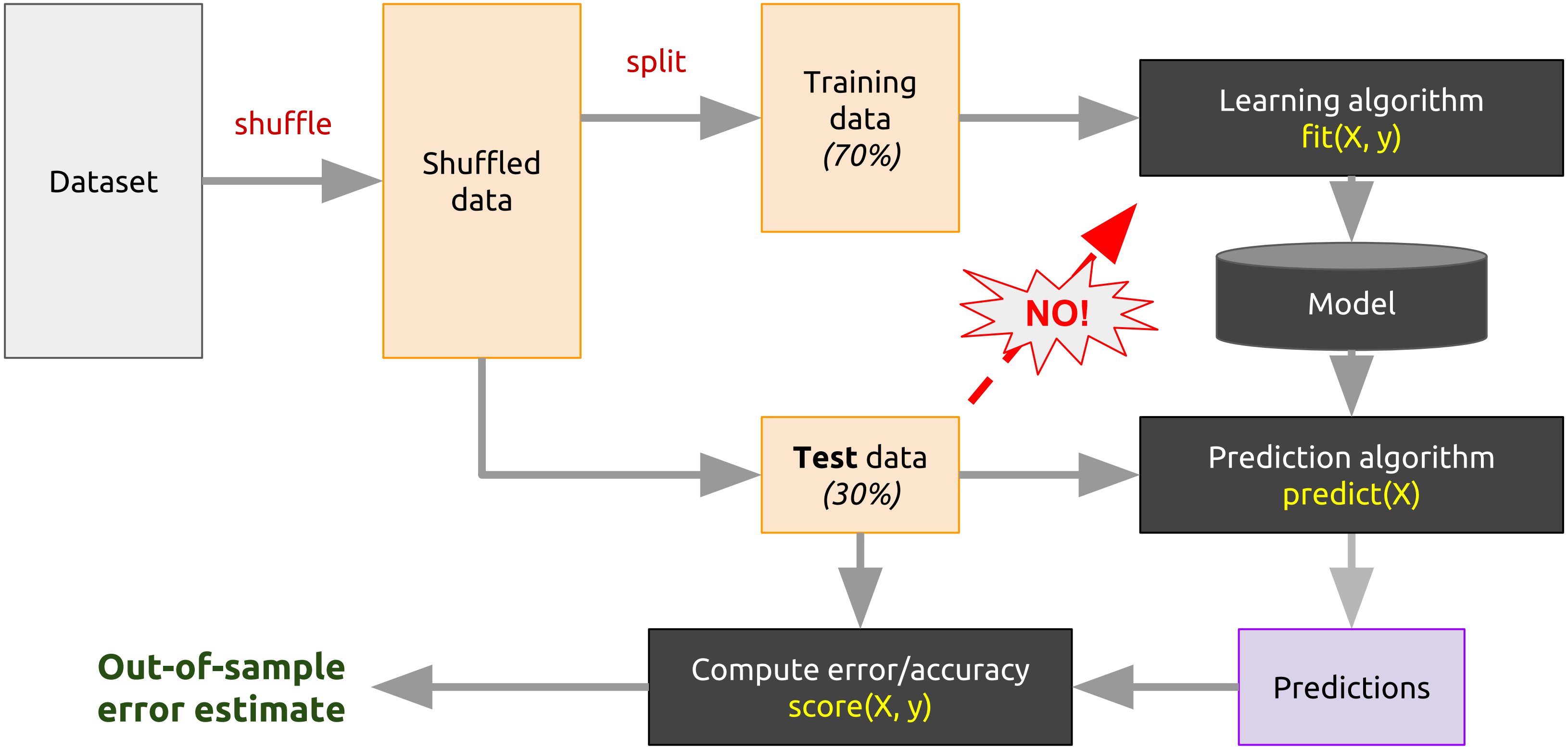
$$\mathbf{y} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \Bigg\} \; N$$

y_3

$$X \in \mathbb{R}^{N \times D}$$

$$\mathbf{y} \in \{0, 1\}^N$$

Training set: Train/Test Splits



Training set: Data hygiene

Split your dataset into train and test at the very start

Usually good practice to shuffle data (exception: time series)

Do not look at test data (data snooping)!

Lock it away at the start to prevent contamination

NB: Never ever train on the test data!

You have no way to estimate error if you do

Your model could easily overfit the test data and have poor generalization, you have no way of knowing without test data

Model may fail in production

Training: Goal

Given some paired training examples $\{(\mathbf{x}_i, y_i): \mathbf{x}_i \in \mathbf{X}, y_i \in \mathbf{Y}\}$ produce a function $y = f(\mathbf{x})$ such that $f(\mathbf{x})$ **generalizes well** to previously unseen data.

Examples

X are times of day, **Y** are light levels

X are light levels, **Y** are times of day

X are measurements from sensors (temp, humidity, brightness, etc.), **Y** is {rain, no rain}

X are words occurring in an email, **Y** is {spam, not spam}

X are vectors of image pixels, **Y** is {cat, dog, car, person, …}

X are recorded audio fragments, **Y** are words

Training: Remember Metrics/Loss function

Classification Metrics:

$$Accuracy = \frac{\boxed{TP+TN}}{\boxed{TP+TN+FP+FN}}$$

→ Not differenciable!

Example: Binary cross entropy:

$$L = -\frac{1}{N}\sum_{i=1}^{N} y_i \log f(\mathbf{x}_i) + (1-y_i)\log(1-f(\mathbf{x}_i))$$

Training: Monitoring progress

1. Split data into train, validation, and test sets

    Keep 10-30% of data for validation

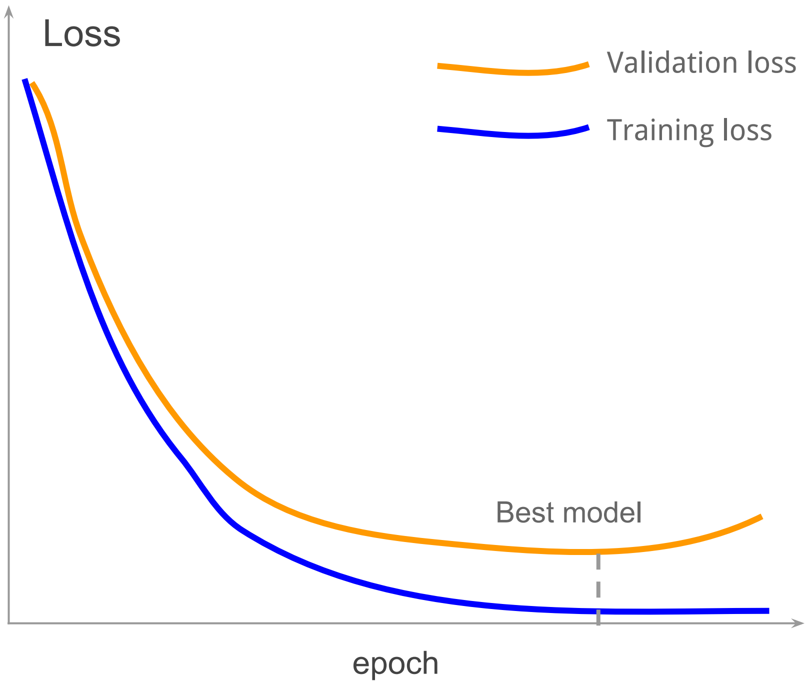
2. Fit model parameters on train set using SGD
3. After each epoch:

    Test model on validation set and compute loss

    Also compute whatever other metrics you are interested in, e.g. top-5 accuracy

    Save a snapshot of the model

4. Plot **learning curves** as training progresses
5. Stop when validation loss starts to increase
6. Use model with minimum validation loss



Overfitting

Symptoms:

Validation loss decreases at first, then starts increasing

Training loss continues to go down

Try:

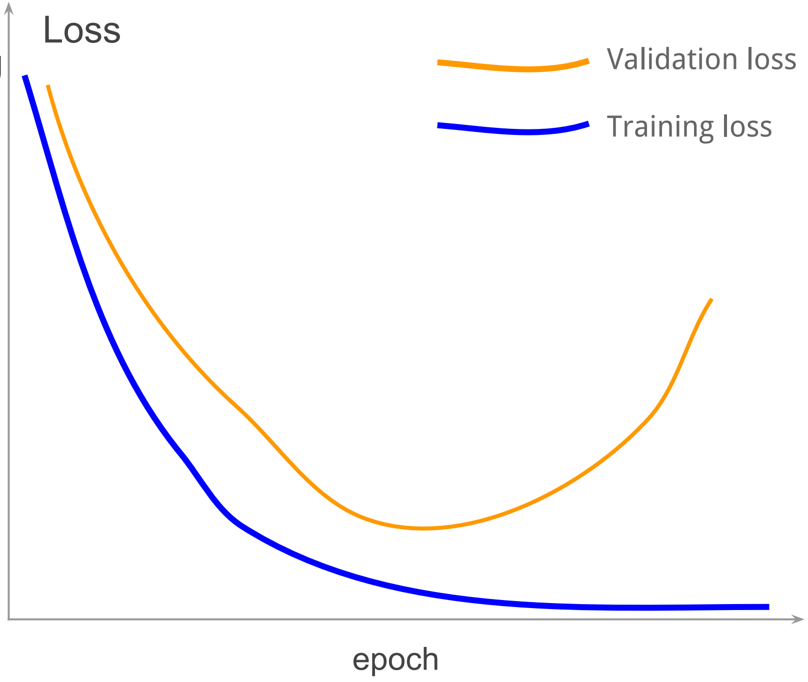
Find more training data

Add stronger regularization

dropout, drop-connect, L2

Data augmentation (flips, rotations, noise)

Reduce complexity of your model



Underfitting

Symptoms:

Training loss decreases at first but then stops

Training loss still high

Training loss tracks validation loss

Try:

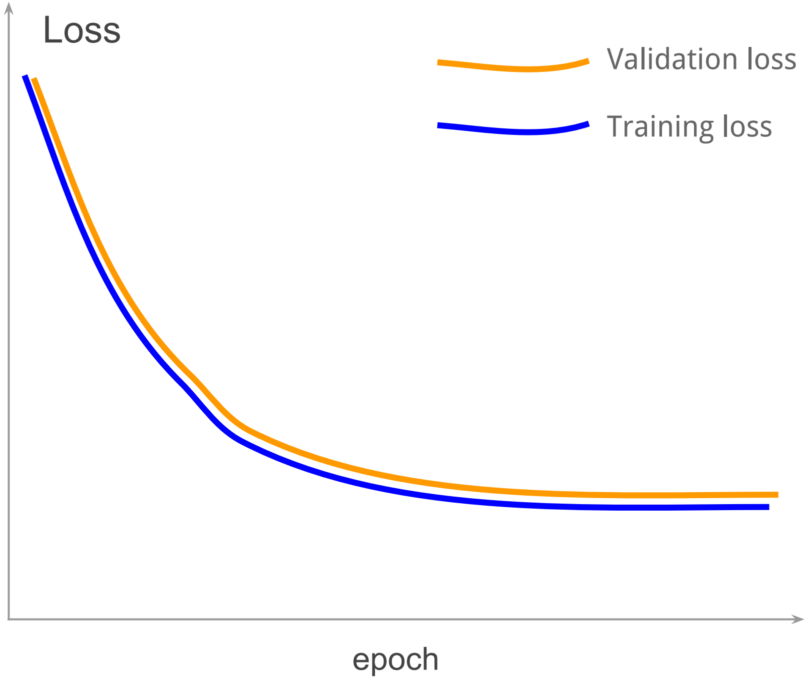
Increase model capacity

Add more layers, increase layer size

Use more suitable network architecture

E.g. multi-scale architecture

Decrease regularization strength



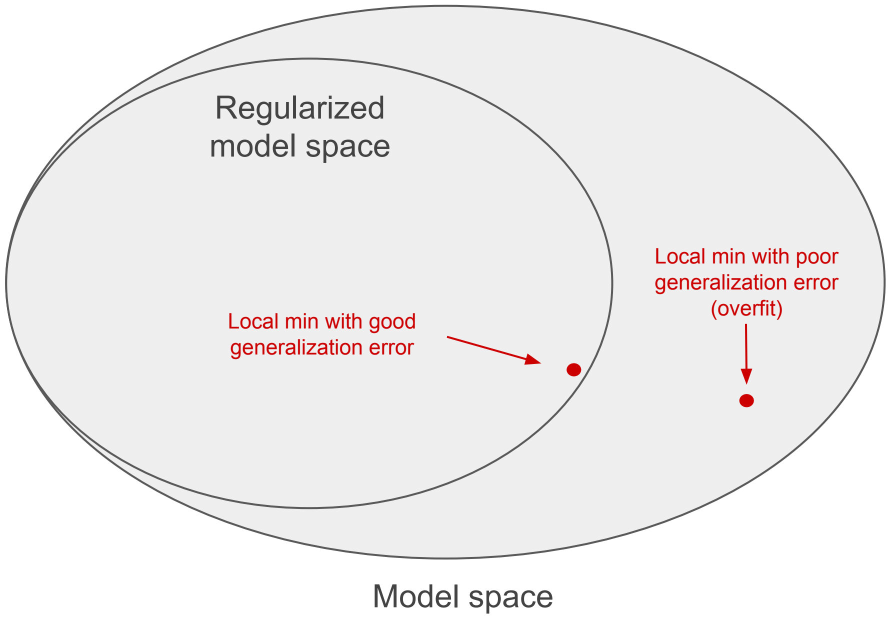
Regularization

Early stopping is a form of structural risk minimization

- Limits the space of models we explore to only those we expect to have good generalization error
- Helps prevent **overfitting**
- A type of **regularization**

Other regularization techniques:

- Weight constraints: e.g. **L2 regularization**
  - Aka. weight decay
- Dropout
- Transfer learning, pretraining



Regularization: Weight decay

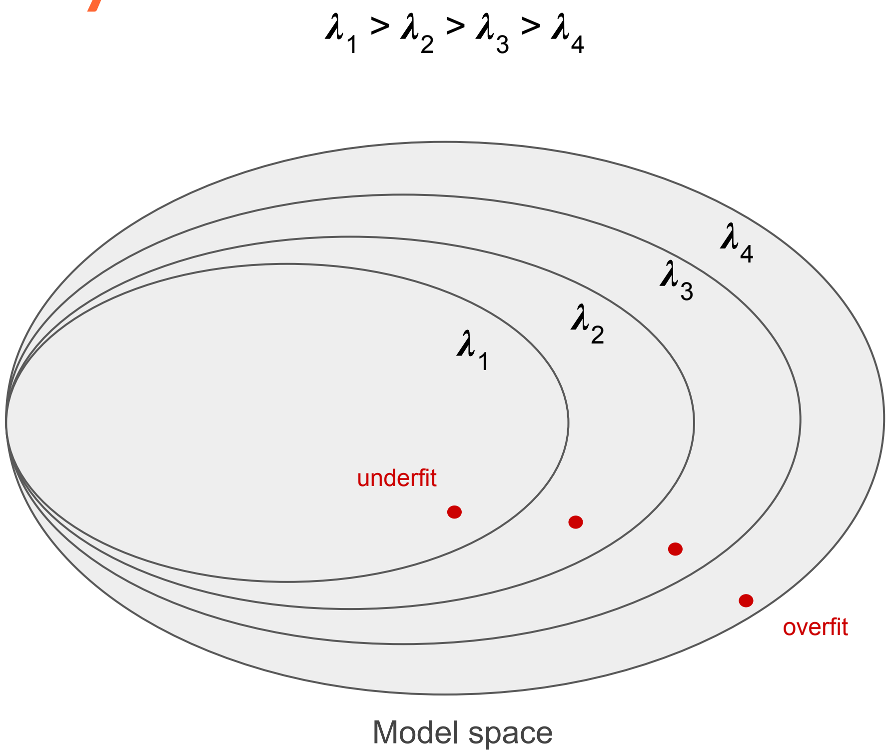
Add a penalty to the loss function for large weights

L2 regularization on weights

$$L = L_{\text{data}} + \frac{\lambda}{2}||W||_2^2$$

Differentiating, this translates to decaying the weights with each gradient descent step

$$w_{t+1} = w_t - \alpha\Delta_w L_{\text{data}} - \lambda w$$



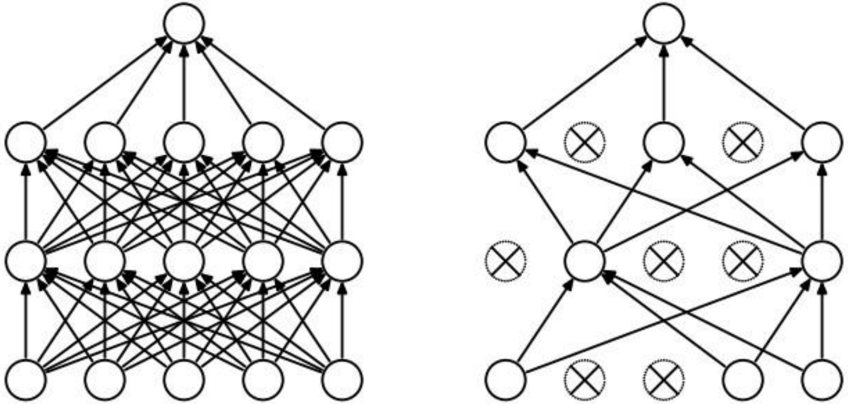
Regularization: Dropout

Modern regularization technique for deep nets

Used by many deepnets

Method:

- During training, outputs of a layer to zero randomly with probability p
  - Prevents units from co-adapting too much
  - Forces network to learn more robust features
- At test time, dropout is disabled and unit output is multiplied by p



(a) Standard Neural Net (b) After applying dropout.

Srivastava et al. **Dropout: A simple way to prevent neural networks from overfitting.** JRML 15(1), 2014, pp 1929-1958.

Hyperparameters

Can already see we have lots of **hyperparameters** to choose:

1. Learning rate
2. Regularization constant
3. Number of epochs
4. Number of hidden layers
5. Nodes in each hidden layer
6. Weight initialization strategy
7. Loss function
8. Activation functions
9. … :(

Choosing these is difficult, and a bit of an art.

There are some reasonable **heuristics**:

1. Try 0.1 for the learning rate. If this doesn't work, divide by 3. Repeat.
2. Multiply LR by 0.1 every 1-10 epochs.
3. Try ~ 0.00001 as regularization constant
4. Try an existing network architecture and adapt it for your problem
5. Start smallish, keep adding layers and nodes until you overfit too much

You can also do a **hyperparameter search** if you have enough compute:

Randomized search tends to work well

Summary

To produce a function that generalizes well to unseen data

It can be a challenging task: Overffitting/Underfitting

Regularization as a way of adding extra information during training for finding the good model.

Training Neural Networks requires to set a large number of parameters (hyperparameters). Choosing is difficult and requires experimentation (a bit of an art!).