**Week 1 – Notes**

**Introduction to ML Strategy**

**Why ML Strategy**

A strategy is needed because when you want to improve the performance of your NN you can try a lot of things, but you can wander around for many months by choosing a wrong path

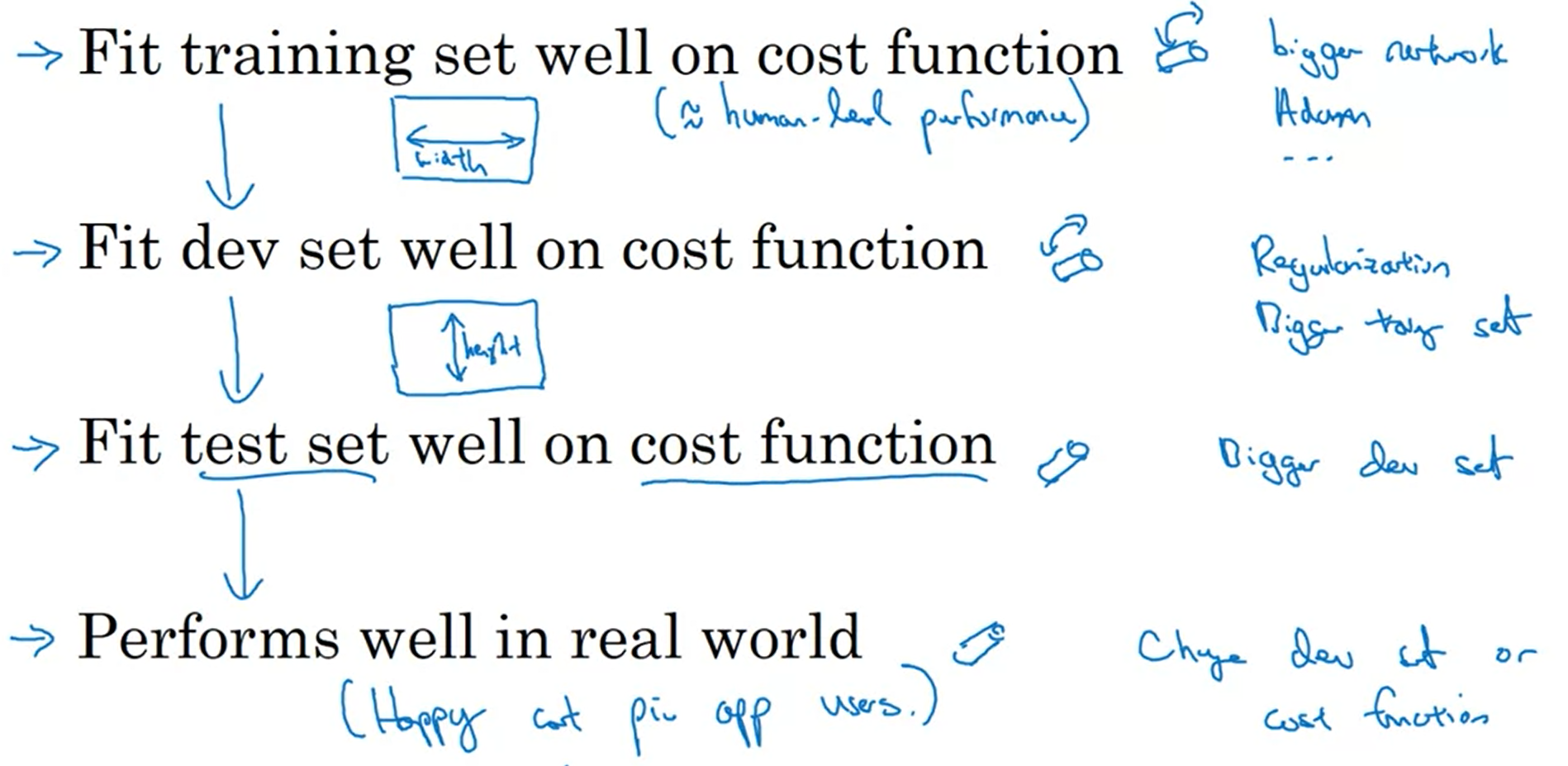
**Orthogonalization**

Is the process of tuning a NN by separating the techniques used, so that we don’t influence other things except the thing we are tuning

For example, an analog TV has several buttons to adjust the image, but each knob do not influence other settings

If there had been a single knob that changes every setting in a specific proportion, the process would had been harder

Exactly like that for the DL, there is a chain of assumptions



Early stopping represents a technique that interfere in the same time with the performance on the train and dev sets

**Setting Up your Goal**

**Single Number Evaluation Metric**

If you deal with a classifier for which you have computed the precision and the recall, it would be really hard to pick the best model considering multiple metrics

Precision – of the examples recognized as 1, what % actually are 1?

Recall – what % of actual 1s are correctly recognized?

It’s better to use only one metric, for example the F1 score, which is the harmonic mean of P and R

In order to evaluate a model you need a dev set and a single (real) number evaluation metric

Another example: if you have multiple models with metrics computed on different regions, pick the best model based on the mean of the performance across these regions

A picture containing text, screenshot, font, number

Description automatically generated

**Satisficing and Optimizing Metric**

You cannot optimize more metrics at once

For example, if you have several classifiers for which you measure the accuracy and the running time, only the accuracy has to be maximized and the running time to be satisfied (under one threshold)

If you have N metrics, you optimize one and you satisfy N-1

**Train/Dev/Test Distributions**

The dev set + the metric represent the target you want to hit

It’s really important to pick these 2 correctly, because otherwise, if the dev and test sets have different distributions, then you can have good results for dev, but the results on test are rubbish

Dev and test sets should be randomly picked; pick them so that they reflect the data you expect to get in the future and consider important to do well on

The training set and how you choose it affects how well you hit that target



**Size of the Dev and Test Sets**

Traditionally, the dev and test sets represent a large portion for the whole data (like 30% - 40% in total) and that’s fine when you deal with small data sets (<10,000 examples)

For large data (>1,000,000 examples), it’s ok to leave only 0.1% or 1% for dev / test

The size of the test set should be big enough to give high confidence in the overall performance of your system

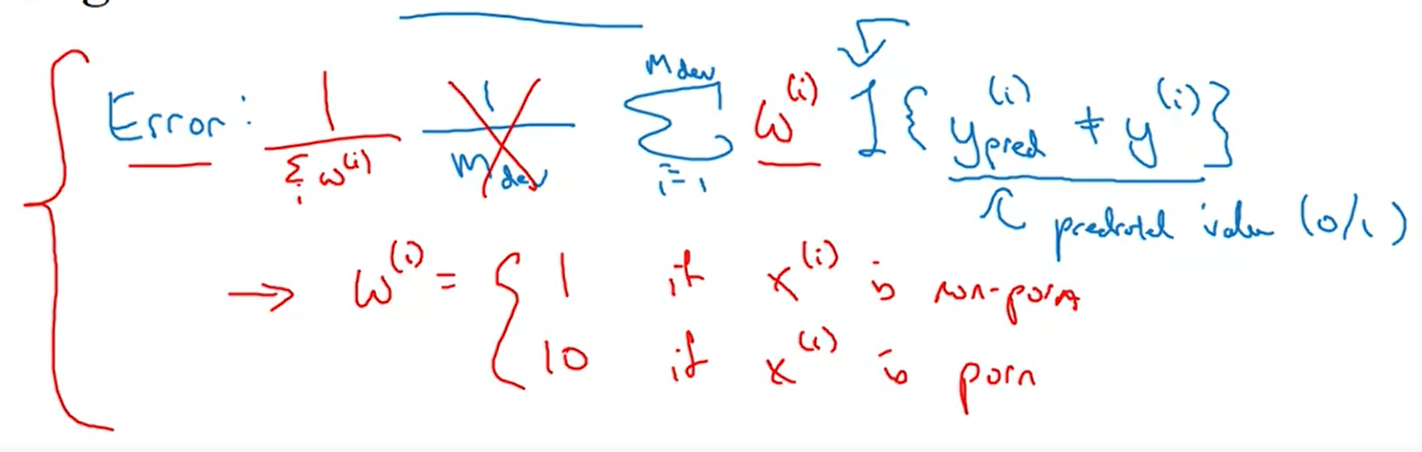
If you are ok with a biased estimate on the dev set, you can avoid having a test set, but if you have only 2 sets, name them train and dev, not train and test

**When to Change Dev/Test Sets and Metrics?**

Let’s say you have trained 2 classifiers: A with 3% error and B with 5% error

The problem is that the better classifier predicts pictures you don’t want your user to see, so you’re not satisfied with the results, even though it has the better metric (you prefer B)

This is a sign that you should change the metric / loss function, so that it will fit your preferences; in this case you just penalize when an unwanted picture (porn one) is predicted



The orthogonalization process:

1. define the metric that fits your preferences

2. only then focus on how to do well on this metric

If you have a model that performs well on the dev and test data, but it does bad in production (on data from users), then this is a sign that you have to change the dev / test sets to match the real data

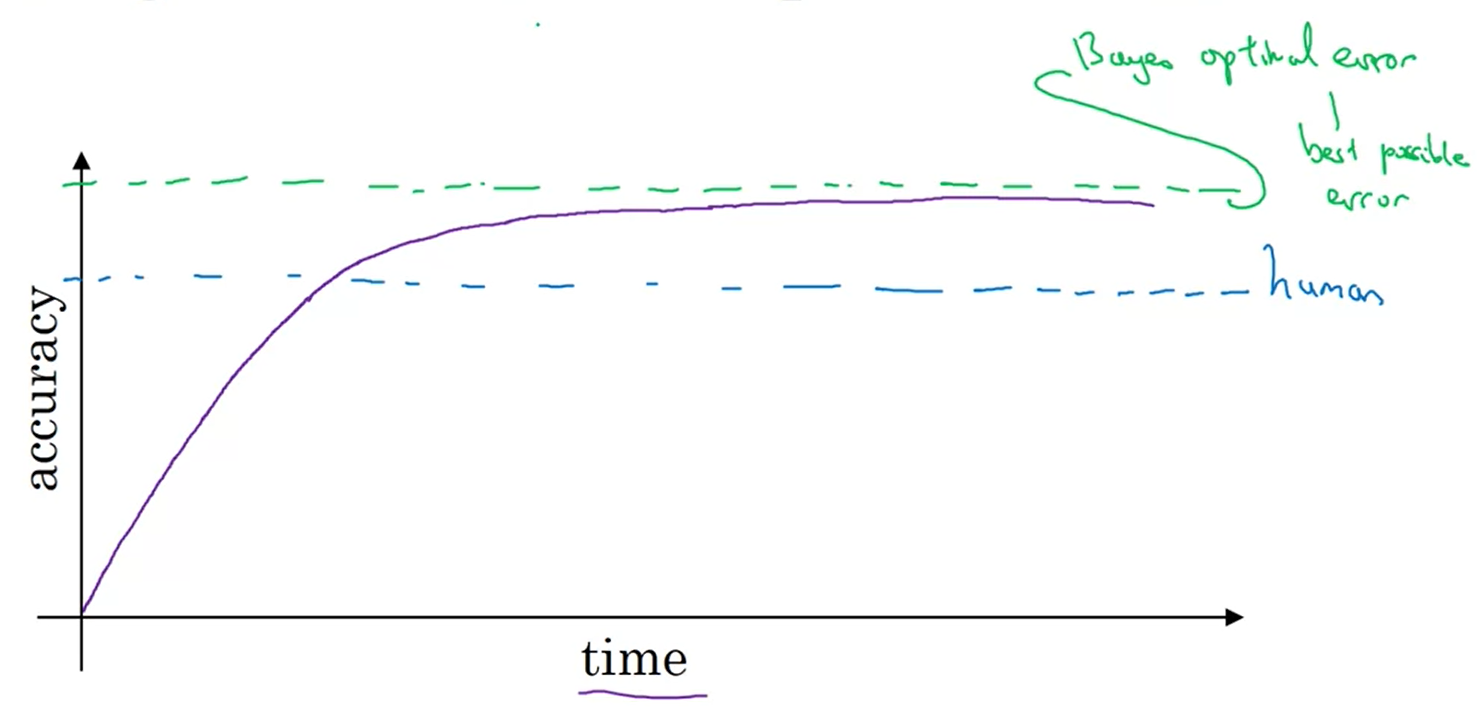


If doing well on your metric + dev / test data sets does not correspond to doing well on your application, change your metric and / or your dev / test set

**Comparing to Human-level Performance**

**Why Human-level Performance?**

There are two performance thresholds we want to consider: the human one and the Bayes optimal error, which is the best possible error (it isn’t feasible to reach)



Over time, is easier to reach the human performance, but then it’s extremely hard to increase it towards Bayesian optimal error

It’s easier to reach the human level because you can get labeled data from humans, gain insights from manual error analysis (why did a person get this right), better analysis of bias / variance

**Avoidable Bias**

The human-level error is a proxy for Bayes error

Compare 2 differences: human-level error and training error and training error and dev error

If the first difference is bigger, we focus on the avoidable bias, else we try to reduce the variance

A picture containing text, handwriting, font, calligraphy

Description automatically generated

**Understanding Human-level Performance**

A picture containing text, screenshot, font, algebra

Description automatically generated

We know that the bayes error <= 0.5%, because is could not be reached in real life

It’s reasonable to pick the human-level error as the one of the typical doctor

It’s important which error rate is picked as the proxy for Bayes, because if you have a really low training error (lower than the Bayes one), it’s almost impossible to know what to improve (avoidable variance or the bias)

A picture containing handwriting, text, font, calligraphy

Description automatically generated

In the case above, the Bayes error as 0.7% is almost useless, but if we were to pick the 0.5%, then we would know that both the bias and the variance can be improved (even though 0.1% and 0.2% are extremely low, they are similar and compared to the Bayes error can be improved)

Initially, to check if our model is biased, we compared the training error with 0%, which is many cases is not a feasible human-level error; better, pick a realistic Bayes error

**Surpassing Human-level Performance**

If the human-level performance is 0.5% and the training error is 0.3%, dev error is 0.4% you don’t know if the actual bayes error is lower than 0.3% or if your model overfits by 0.2%; additionally, you don’t know if you have a bias or variance problem

Usually, the human-level performance is easier to surpass when solving problems defined based on structural data, with a lot of data and without natural perception (such as image recognition)

Examples:

A picture containing text, font, screenshot, white

Description automatically generated

However, in the last decade, ML systems have become increasingly better at speech recognition, medical problems and some image recognition tasks

**Improving your Model Performance**

There are fundamental assumptions of the supervised learning:

You can fit the training set well (avoidable bias)

The training set performance generalizes well to the dev / test set (variance)

Techniques that can be used to solve the avoidable bias problem:

A picture containing text, font, line, screenshot

Description automatically generated

Techniques that that can be used to fix the variance problem:

A close-up of a document

Description automatically generated with low confidence