# Design patterns for Machine Learning

Exploration report by Szymon Jessa

2017-10-23, mlgdansk

*Best practices?*

# ~~Design patterns~~ for Machine Learning

Exploration report by Szymon Jessa

2017-10-23, mlgdansk

# Reference

**Explored:**

- Leslie N. Smith, ***Best Practices for Applying Deep Learning to Novel Applications***, https://arxiv.org/abs/1704.01568
- Andrew Ng, ***Structuring Machine Learning Projects***, https://www.coursera.org/learn/machine-learning-projects/home/welcome
- Andrew Ng, ***Machine Learning***, https://www.coursera.org/learn/machine-learning
- Martin Zinkevich, ***Rules of Machine Learning: Best Practices for ML Engineering***, http://martin.zinkevich.org/rules_of_ml/rules_of_ml.pdf
- Robert Chu, David Duling, Wayne Thompson, ***Best Practices for Managing Predictive Models in a Production Environment***, http://www2.sas.com/proceedings/forum2007/076-2007.pdf

**Not yet explored:**

- More than 10 other papers/pages/lessons
- **DO YOU HAVE ANY RECOMMENDATIONS?**

## *Design patterns? But I know everything already...*

**ML Superhero:** My great model achieved 90% accuracy.

**CEO:** Nice, but we need 95%. What can we do?

**ML Superhero:** I have plenty of ideas what we could do:

- so we can address the **bias**
- try a **bigger network**
- or maybe we **overfit**
- we could ask Martin for **more data**
- or do **data augmentation**
- use **regularization**
- maybe **dropout**
- check the alternative **features extraction**
- review some **recent papers** from competition
- try the **novel idea** I have...

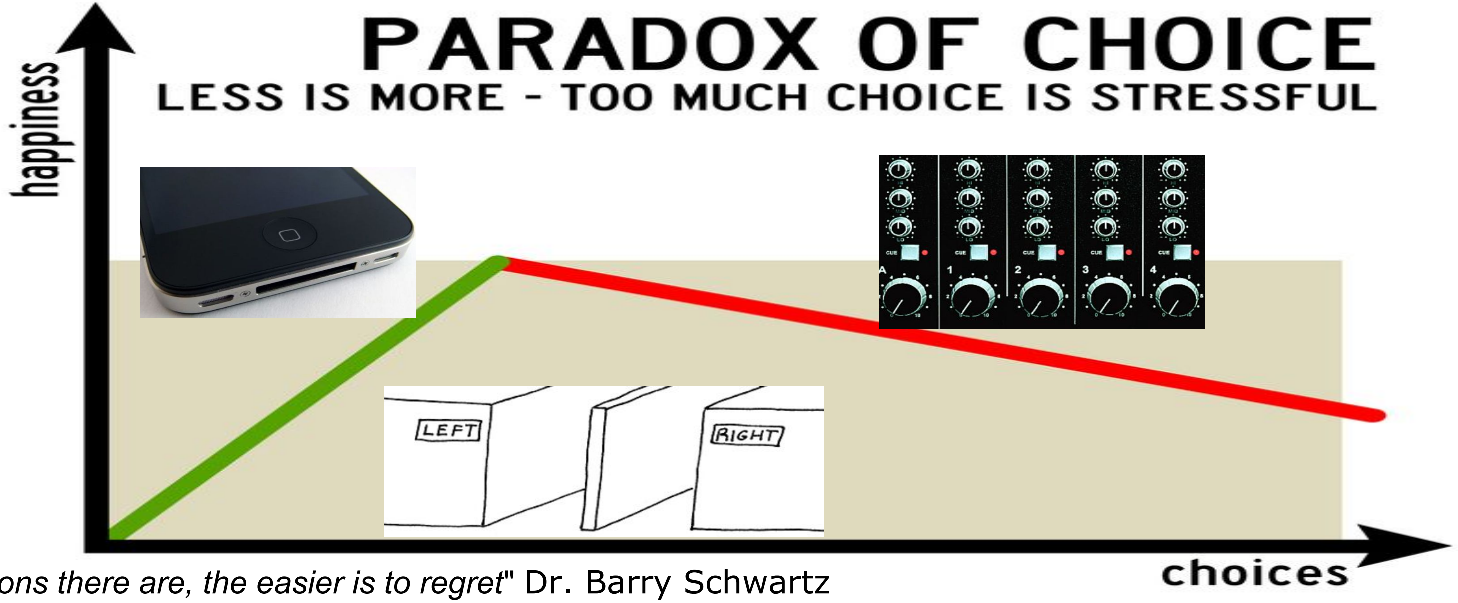
**CEO:** Nice, but we have **no time** to check everything, so which of these makes **most sense** to do now?

**ML Superhero:** Ok... Let me check in the "***Design patterns for Machine Learning***" presentation...

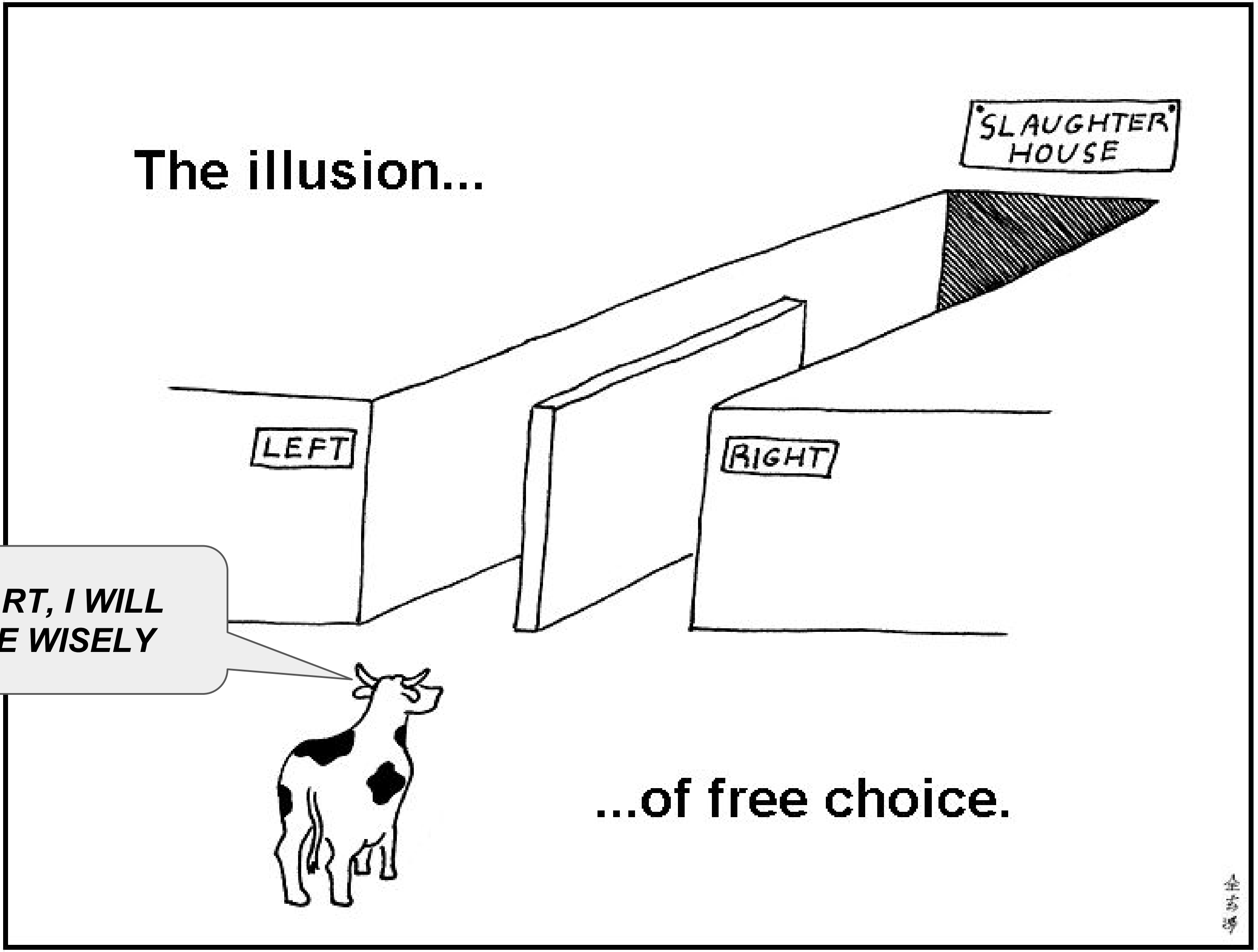
## Root cause? *I'm just too smart...?*

TOO
**~~SO~~ many options**

**Single variable/parameter affects multiple things**



"*The more options there are, the easier is to regret*" Dr. Barry Schwartz
https://www.ted.com/talks/barry_schwartz_on_the_paradox_of_choice/transcript


The illusion...
SLAUGHTER HOUSE
LEFT
RIGHT
I'M SMART, I WILL CHOOSE WISELY
...of free choice.

## Orthogonalization

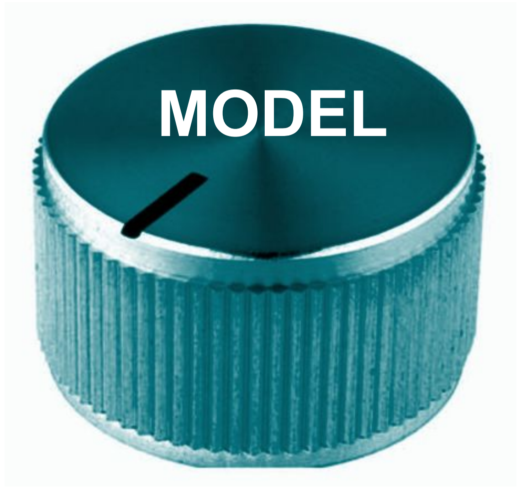
**MAKE SINGLE KNOB RESPONSIBLE FOR CHANGING SINGLE THING**

**MAKE IT EASY TO UNDERSTAND**

*ML EXPERTS KNOW WHICH KNOB TO USE TO MAKE THE DESIRED CHANGE*

## Orthogonalization on high level

Define the metric for what you aim to achieve



Build a model that will fit the target

## Building an application: General advices

Build your first system **quickly**, then **iterate**.

Build **solid** end to end **pipeline** (complete system)

Start with **reasonable objective**

Add **common sense features** in a simple way

Ensure the **pipeline stays solid**

*And... do you **really need** to use Machine Learning for your project at all?*

## Building an application: General advices

If machine learning is not **absolutely required** for your product, don't use it until you have data.

## Building an application: General advices

If you think that machine learning will give you a 100% boost,
then a heuristic will get you 50% of the way there.

***But...***

## Building an application: General advices

If you think that machine learning will give you a 100% boost,
then a heuristic will get you 50% of the way there.

***But...***

Choose machine learning over a complex heuristic:

A complex heuristic is **unmaintainable** - -learned model is easier to update and maintain.

Once you have data and a basic idea of the goal, move on to machine learning.

## Building an application: General advices

Use bias, variance and error analysis to prioritize next steps.

If there are already solutions for your problem - use them,
*but don't make it more complex than necessary*!

## Building an application: Steps

Step 1. Define goal and set evaluation metric
Step 2. Prepare data
Step 3. Baseline model
Step 4. Environment - visualization, debugging
Step 5. Fine tune model
Step 6. Increase complexity

## Define goal and set evaluation metric

- Define what success look like (either if done by human or computer)
- What is human performance and how the state-of-the art compares to it?
- Make the network's task as easy as possible:
  eg. face detection, face recognition but not face detection + face recognition
  (the easier the job the networks must perform, the easier it will be to train and the better the performance)
- Remember: The scarcest resource is human time so let the network learn its representations rather than require any fixed, manual preprocessing

## Single number evaluation metric

*WHICH MODEL IS BETTER?*

*DID WE IMPROVED WITH MODEL B OR NOT?*

| Model | Precision | Recall |
|---|---|---|
| A | 91% | 87% |
| B | 94% | 80% |

$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Recall} = \frac{tp}{tp + fn}$$

## Single number evaluation metric

*When ONE is MORE than TWO*

| **Model** | **Precision** | **Recall** | **F-measure** |
|---|---|---|---|
| **A** | 91% | 87% | **89%** |
| B | 94% | 80% | **86%** |

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

*So model A was better... did you noticed?*

## Single number evaluation metric

*When ONE is MORE than TWO*

| Model | Precision | Recall | F-measure |
|---|---|---|---|
| A | 91% | 87% | **89%** |
| B | 94% | 80% | **86%** |

**ONE... BUT WHICH ONE!?**

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

*So model A was better... did you noticed?*

## Single number evaluation metric

Don’t overthink which objective to choose to optimize.

*Early in the machine learning process, you will notice all the metrics going up, even those that you do not directly optimize.*

Keep it simple until you can still easily increase all the metrics.

Note: Often you don't know the true objective. The ML objective should be something that is easy to measure and is a proxy for the “true” objective.

## Human-level performance: *coz we are the best...*

**Why it is good to know the human-level performance?**

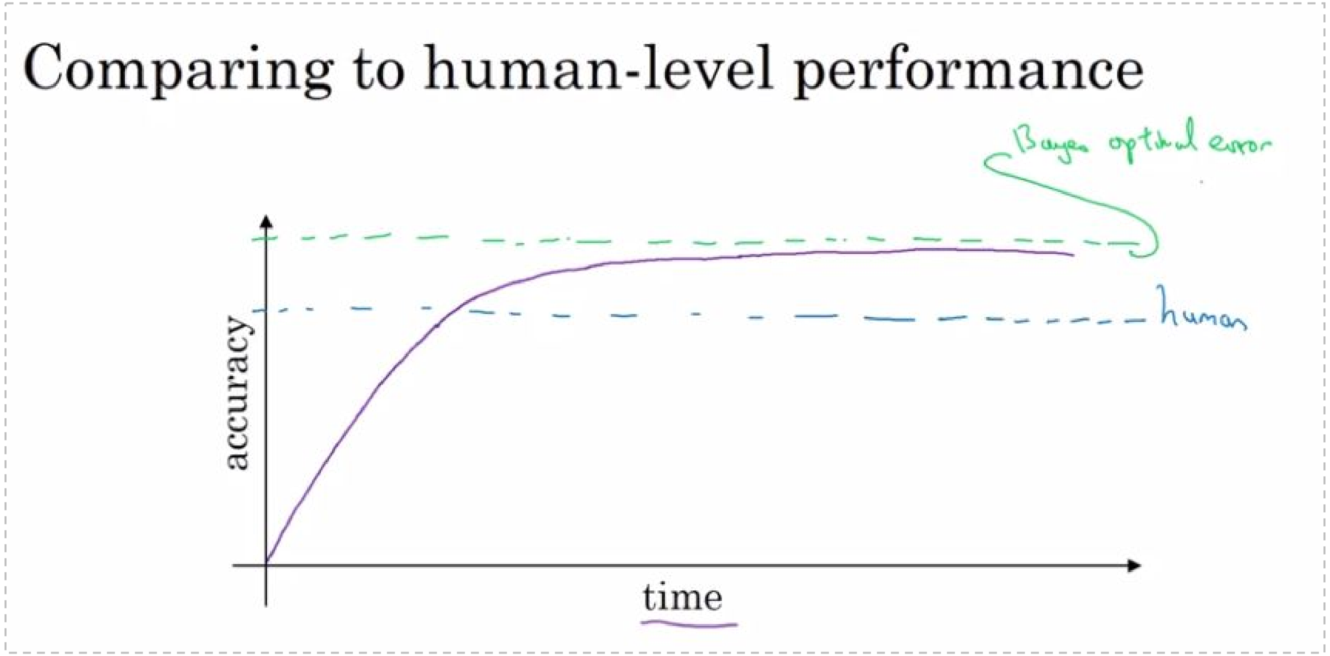
If the model performs below or at human-level, you can:

- get manually labeled data
- gain insight from manual error analysis (why did a person get this right?)
- analyse the bias/variance better
- use human-level error as a proxy for Bayes error

Knowing human-level performance, you know how well but not too well you want your algorithm to do on the training set.

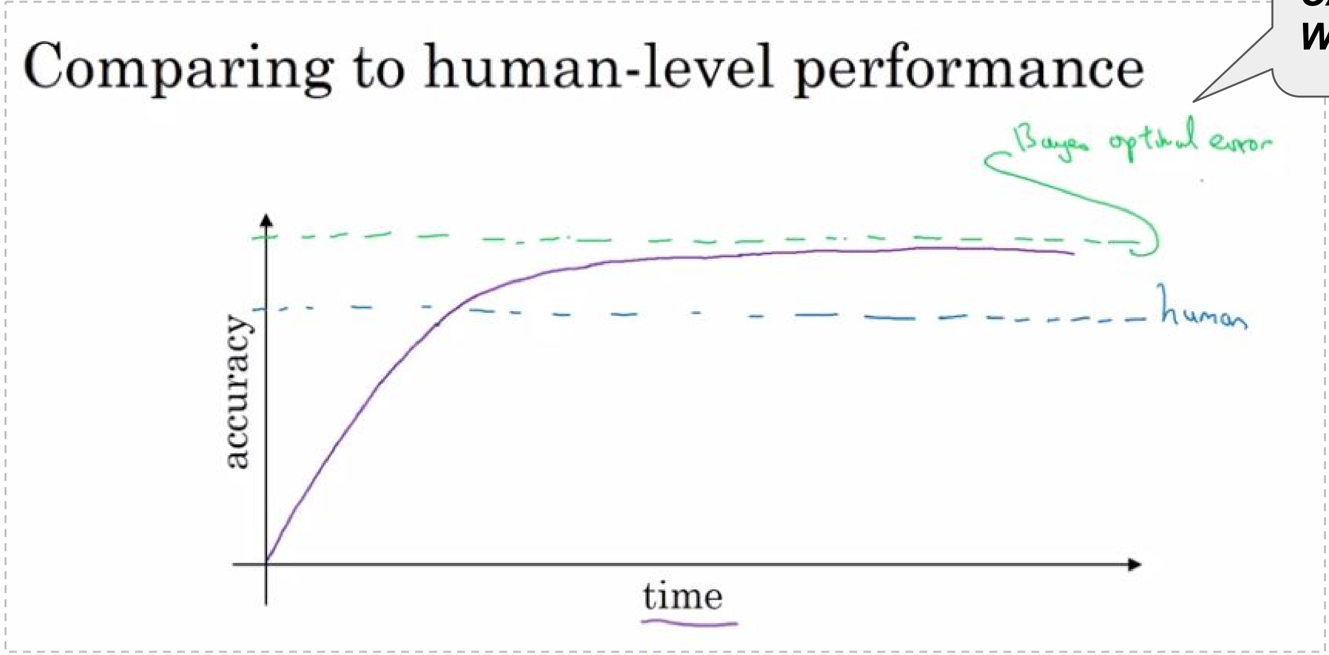
# Human-level performance vs Bayes error rate

**Bayes error rate** is the lowest possible error rate for any classifier of a random outcome and is analogous to the irreducible error



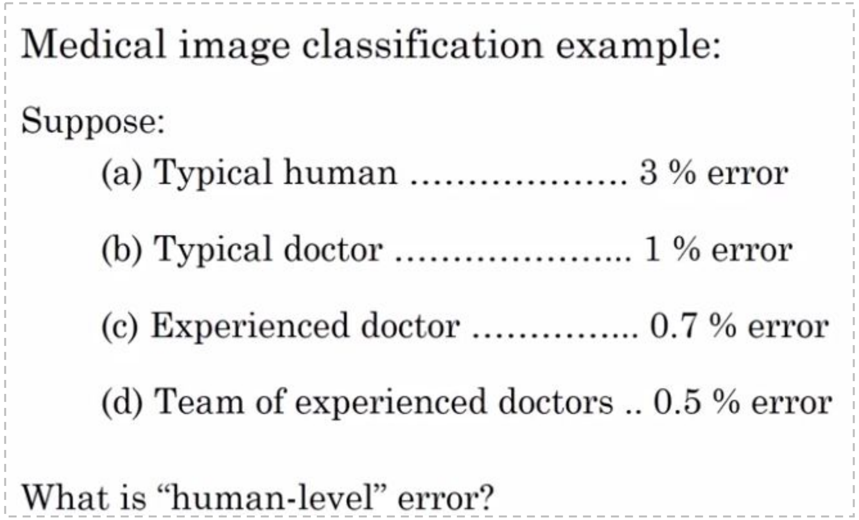
## Human-level performance vs Bayes error rate

**Bayes error rate** is the lowest possible error rate for any classifier of a random outcome and is analogous to the irreducible error



*CAN YOU RECOGNIZE WHO WROTE THAT?*

## Human-level performance: *which one?*



Medical image classification example:

Suppose:

| | |
|---|---|
| (a) Typical human | 3 % error |
| (b) Typical doctor | 1 % error |
| (c) Experienced doctor | 0.7 % error |
| (d) Team of experienced doctors | 0.5 % error |

What is "human-level" error?

**QUESTION**: *WHAT IS THE HUMAN-LEVEL ERROR IN THIS EXAMPLE?*

**QUESTION**: *WHAT IS THE BAYES ERROR?*

## Step 2. Prepare data

### Make the data wait for YOU - and not the opposite

Before formalizing what your machine learning system will do, track as much as possible in your current system, eg. gain permission from the system's users earlier. If you think that something might be a concern in the future, it is better to get historical data now.

### Ensure you are ALLOWED to use the data

Identify legal issues: certain variables might be good predictors, they may result in unwanted risk exposure for an organization when used in a predictive model. For example, there can be legal reasons why variables such as age, gender, and/or race might not be used in the model-building process.

## Step 2. Prepare data

### Data completeness

- Ensure the data is relevant to the goal
- Ensure that it is diverse enough that it covers the whole problem space
- Check if the classes are balanced

### Make the data easy to learn

- Zero mean / normalize the data (removes the need of learning the mean)
- Use a priori knowledge or known heuristics to lower the dimensionality of the data

## Step 2. Prepare data

Not enough data?

- Consider transfer learning
- Download datasets that are closest to your data to use for pre-training
- Consider creating synthetic data

## Split the data: set up training, dev and test sets

**General workflow / data sets use:**

- Try a lot of ideas, train different models on the **training set**,
- and then use the **dev set** to evaluate the different ideas and pick the best one.
- And you keep innovating to improve **dev set** performance until, finally, you have one model that you're happy with (according to the **evaluation metric**),
- that you then evaluate on your **test set**.

## Split the data: size of train, dev and test sets

*YOU ALL KNOW THIS ALREADY, RIGHT?*

| TRAINING SET | DEV SET | TEST SET |
|---|---|---|
| 60% | 20% | 20% |

and *without the test set:*

| TRAINING SET | DEV SET |
|---|---|
| 80% | 20% |
| 70% | 30% |

# Split the data: size of train, dev and test sets

*YOU ALL KNOW THIS ALREADY, RIGHT?*

| TRAINING SET | DEV SET | TEST SET |
|---|---|---|
| 60% | 20% | 20% |

and *without the test set:*

| TRAINING SET | DEV SET |
|---|---|
| [illegible]% | 20% |
| 70% | 30% |

**NOT REALLY**

## Split the data: size of train, dev and test sets

IT IS OK FOR ~100-10k EXAMPLES, BUT WHAT ABOUT 1MLN EXAMPLES?

MAKE THE DEV/TEST BIG ENOUGH TO VERIFY THE OVERALL PERFORMANCE BUT NOT BIGGER

| TOTAL NUMBER OF EXAMPLES | TRAINING SET | | DEV SET | | TEST SET | |
|---|---|---|---|---|---|---|
| 10 000 | 60% | 6 000 | 20% | **2 000** | 20% | **2 000** |
| 1 000 000 | 98% | 980 000 | 1% | **10 000** | 1% | **10 000** |

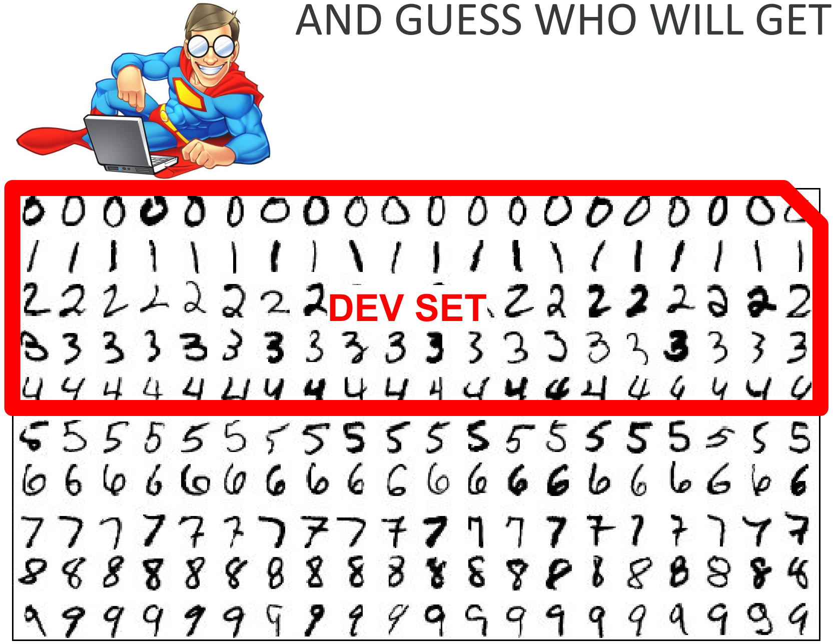
HAVING 1M EXAMPLES, WITH 1% WE STILL GET ENOUGH! ...

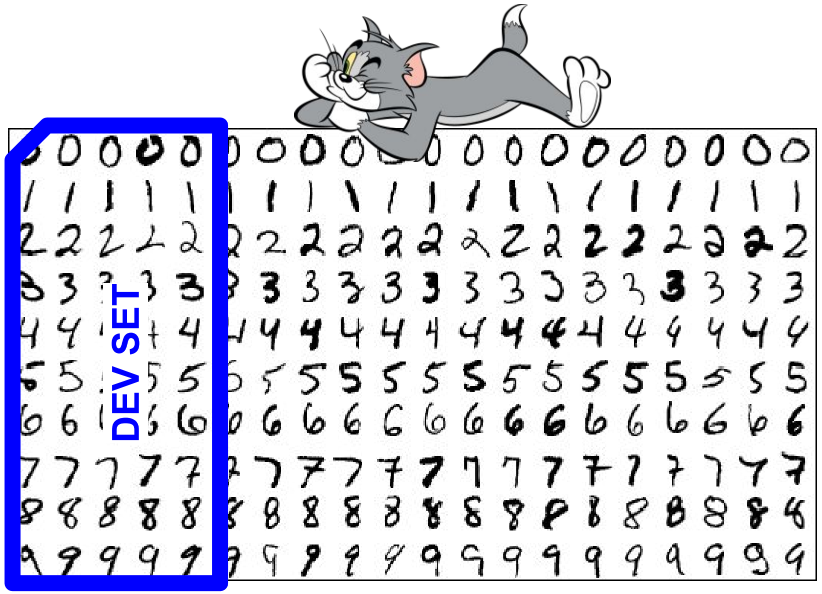
... AND WE WILL ENJOY THE TRAINING DATA IN DEEP LEARNING

## Split the data: in dev/test sets balance is the key

IMAGINE: BOTH **YOU** AND **TOM** ACHIEVED 100% ACCURACY ON YOUR **DEV SETS**.

AND GUESS WHO WILL GET MORE CUSTOMER COMPLAINS?



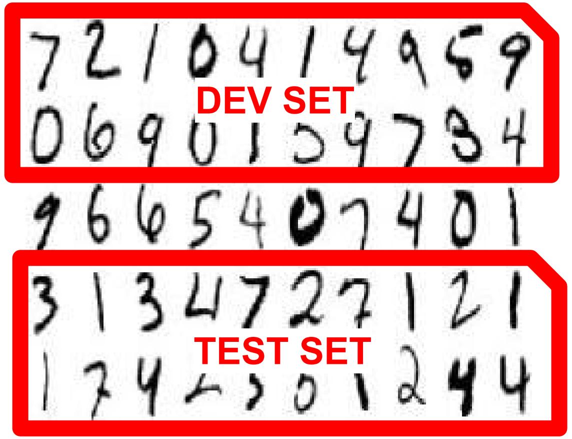


*DID YOU TARGETED THIS SAME GOAL?*

## Split the data: in dev/test sets balance is the key

**GUIDELINE:** ENSURE DEV AND TEST SETS REFLECT THE DATA (INC. DISTRIBUTION) EXPECTED IN THE TARGET ENVIRONMENT

**BEST PRACTICE:** SHUFFLE THE DATA AND SPLIT IT THEN INTO DEV AND TEST SET



## Cleaning data: concerns with incorrect labels

What to do with incorrectly labelled examples in **training data set**?

Deep learning algorithms are **robust to random errors**

(***SOME*** white dogs labelled as cat)**,**

but are **prone to systematic errors**

(***ALL*** white dogs labelled as cat)**.**

# Cleaning data: concerns with incorrect labels

What to do with incorrectly labelled examples in **dev or test set**?

When do we need to fix them?

Do error analysis:

- get 100 incorrectly classified examples
- count how many of the incorrect classifications are due to an incorrect label
- calculate how much they affect the dev set error (eg. 6 wrong labels in 100 errors = 6%; 6% of 10% => 0.6%)

| | **PROJECT ALPHA** | **PROJECT BETA** |
|---|---|---|
| DEV SET ERROR | 10% | 2% |
| ERRORS DUE TO INCORRECT LABELS | 0.6% | 0.6% |
| **IS IT WORTH FIXING LABELS?** | ***Do you know?*** | |

## Cleaning data: concerns with incorrect labels

What to do with incorrectly labelled examples in **dev or test set**?

When do we need to fix them?

**GUIDELINE:** Fix labels, only if it makes significant difference

| | **PROJECT ALPHA** | **PROJECT BETA** |
|---|---|---|
| DEV SET ERROR | 10% | 2% |
| ERRORS DUE TO INCORRECT LABELS | 0.6% => 9.4% | 0.6% => 1.4% |
| **IS IT WORTH FIXING LABELS?** | **MAYBE NOT YET?** | **YES!** |

# Cleaning data: concerns with incorrect labels

What to do with incorrectly labelled examples in **dev or test set**?

When do we need to fix them?

**GUIDELINE:** Fix labels, only if it makes significant difference

**BUT WAIT! WHY DO WE NEED TO FIX THEM AT ALL?**

| | **PROJECT ALPHA** | **PROJECT BETA** |
|---|---|---|
| DEV SET ERROR | 10% | 2% |
| ERRORS DUE TO INCORRECT LABELS | 0.6% => 9.4% | 0.6% => 1.4% |
| **IS IT WORTH FIXING LABELS?** | **MAYBE NOT YET?** | **YES!** |

## Cleaning data: concerns with incorrect labels

BUT WAIT! WHY DO WE NEED TO FIX THEM AT ALL?

| | **MODEL ALPHA** | **MODEL BETA** |
|---|---|---|
| DEV SET ERROR | 10% | 5% |
| | | |
| | ***CAN YOU TELL WHICH ONE IS BETTER?*** | |

## Cleaning data: concerns with incorrect labels

**BUT WAIT! WHY DO WE NEED TO FIX THEM AT ALL?**

| | **MODEL ALPHA** | **MODEL BETA** |
|---|---|---|
| DEV SET ERROR | 10% | 5% |
| ERRORS DUE TO INCORRECT LABELS | 80% => 80% * 10% = **8%** | 20% => 20% * 5% = **1%** |
| | ***CAN YOU TELL WHICH ONE IS BETTER?*** | |

## Cleaning data: concerns with incorrect labels

**RECALL:** The **dev set** is used to compare models (MODEL ALPHA or MODEL BETA?).

If due to incorrectly labelled examples the error is too high, the **dev set** error won't be a valid way to compare models.

**BUT WAIT! WHY DO WE NEED TO FIX THEM AT ALL?**

| | **MODEL ALPHA** | **MODEL BETA** |
|---|---|---|
| DEV SET ERROR | 10% | 5% |
| ERRORS DUE TO INCORRECT LABELS | 80% => **8% (best: 2%)** | 20% => **1% (best: 4%)** |
| | ***CAN YOU TELL WHICH ONE IS BETTER?*** | |

## Step 3. Baseline model

**Create a simple baseline model**

- **Keep the first model simple**: The first model provides the biggest boost to your product, so it doesn't need to be fancy.
- Some teams aim for a **“neutral” first launch**: a first launch that explicitly de-prioritizes machine learning gains, to avoid getting distracted.
- Use **one framework** and **one computer language** to minimize complexity
- Find similar application or paper and **replicate the results**
- Use a **smaller architecture** than you anticipate you might need
- Use only **part of the training data**
- Get only the **basic functionality** now
- Adopt some of the practices of **agile** software methodologies

## Step 3. Baseline model

**Use simple features**

Start with directly observed and reported features as opposed to learned features.

**AND EVERYONE KNOWS WHAT A "LEARNED FEATURE" IS...**

## Step 3. Baseline model

**Use simple features**

Start with directly observed and reported features as opposed to learned features.

**AND EVERYONE KNOWS WHAT A "LEARNED FEATURE" IS...**

***A learned feature is a feature generated either by an external system (such as an unsupervised clustering system) or by the learner itself (e.g. via deep learning).***

## Step 3. Baseline model

**Use simple features**

Start with directly observed and reported features as opposed to learned features.

**AND EVERYONE KNOWS WHAT A "LEARNED FEATURE" IS...**

***A learned feature is a feature generated either by an external system (such as an unsupervised clustering system) or by the learner itself (e.g. via deep learning).***

**Turn heuristics into features**

transition to a machine learned system will be smoother

usually those rules contain a lot of the intuition about the system

**... BUT HOW?**

## Step 3. Baseline model

### Turn heuristics into features

**Preprocess using the heuristic**

spam filter: if the sender has already been blacklisted, don’t try to relearn what “blacklisted” means

**Create a feature directly from the heuristic**

use a heuristic to compute a relevance score for a query result as the value of a feature

**Decompose a complex heuristic**

If there is a heuristic for apps that combines the number of installs, the number of characters in the text, and the day of the week, then consider pulling these pieces apart, and feeding these inputs into the learning separately.

***But...*** Using old heuristics in new machine learning algorithm can help to create a smooth transition, but think about whether there is a simpler way to accomplish the same effect.

## Step 3. Baseline model

**Choose the algorithm with respect to your business needs**

The trade-off between interpretability and prediction accuracy must be carefully considered at this stage.

Certain models such as decision trees produce results that are easy to be interpreted. They enable you to answer questions like “*Why was this customer denied credit?*” Other models such as neural networks do not provide this sort of simple interpretation and hence might be inappropriate for certain applications.

## Step 4. Environment - visualization, debugging

- Your level of **understanding** what is happening in the model will affect the success of your project
- **Visualize** your architecture and measure internal entities to understand why you are getting the results you are obtaining
- Set up visualizations so you can **monitor** as much as possible while the architecture evolves
- Create **unit tests** for all of your code modifications

# Step 4. Environment - visualization, debugging

**Test the infrastructure independently from the machine learning**

Make sure that the infrastructure is testable, and that the learning parts of the system are encapsulated so that you can test everything around it. Machine learning has an element of unpredictability, so make sure that you have tests for the code for creating examples in training and serving, and that you can load and use a fixed model during serving

**Audit validation processes: Every step of the validation process needs to be logged**

For example: who imported what model when; who selected what model as the champion model, when and why; who reviewed the champion model for compliance purposes; and who published the champion to where and when.

**Monitor the model**

A champion predictive model is deployed for a period of time in production environments. A champion model needs to retire when its performance degradation hits a certain threshold. Typically, model performance degrades over time.

**Track input and output variable distribution shifts**

## Step 4. Environment - visualization, debugging

### Watch for silent failures

Suppose that a particular table that is being joined is no longer being updated. The machine learning system will adjust, and behavior will continue to be reasonably good, decaying gradually. Sometimes tables are found that were months out of date, and a simple refresh improved performance.  For example, the coverage of a feature may change due to implementation changes: for example a feature column could be populated in 90% of the examples, and suddenly drop to 60% of the examples. Play once had a table that was stale for 6 months, and refreshing the table alone gave a boost of 2% in install rate. If you track statistics of the data, as well as manually inspect the data on occassion, you can reduce these kinds of failures.

### Give feature column owners and documentation

If the system is large, and there are many feature columns, know who created or is maintaining each feature column.

## Step 5. Fine tune model

- This will take the most time - you will **experiment extensively**
- Apart from factors you believe will improve the result - try changing every factor to **learn what happens** when it changes
- The objectives you defined in step 1 should guide how far you want to pursue the **performance improvements**.
- During this phase, all of the **metrics should still be rising.**
- Or you may **revise the objectives** your defined earlier.

# Step 5. Fine tune model

## Use existing features to create new features in human- understandable ways

### Discretization

Taking a continuous feature and creating many discrete features from it.
Example: Consider a continuous feature such as age. You can create a feature which is 1 when age is less than 18, another feature which is 1 when age is between 18 and 35 etc. Don’t overthink the boundaries of these histograms: basic quantiles will give you most of the impact.

### Cross combination of two or more feature columns

A feature column, in TensorFlow's terminology, is a set of homogenous features, (e.g. {male, female}, {US, Canada, Mexico}, etc.). A cross is a new feature column with features in, for example, {male, female} × {US,Canada, Mexico} . This new feature column will contain the feature (male, Canada).
Note that it takes massive amounts of data to learn models with crosses of three, four, or more base feature columns. Crosses that produce very large feature columns may overfit.

## Step 5. Fine tune model

**Use very specific features when you can**

With tons of data, it is simpler to learn millions of simple features than a few complex features. Don’t be afraid of groups of features where each feature applies to a very small fraction of your data, but overall coverage is above 90%. You can use regularization to eliminate the features that apply to too few examples.

**Avoid feedback loops with positional features**

The position of content dramatically affects how likely the user is to interact with it. If you put an app in the first position it will be clicked more often, and you will be convinced it is more likely to be clicked

## Step 5. Fine tune model

**Clean up features you are no longer using**

Unused features create technical debt. If you find that you are not using a feature, and that combining it with other features is not working, then drop it out of your infrastructure.

**Keep coverage in mind when considering what features to add or keep**

How many examples are covered by the feature? Some features may punch above their weight.
For example, if you have a feature which covers only 1% of the data, but 90% of the examples that have the feature are positive, then it will be a great feature to add.

## Step 5. Fine tune model

The number of features you can learn in a linear model is roughly proportional to the amount of data you have

**1 000 examples -> a dozen features**

**10 000 000 examples -> hundred thousand features**

**billions or hundreds of billions of examples -> 10 million features**

## Ceiling analysis: *what to do next?*

Iterate from the first to the last component in the pipeline:

- assume that the component gets 100% accuracy -> and evaluate the system.

OCR pipeline: text detection, character segmentation, character recognition.

| Step | System accuracy | Change |
|---|---|---|
| Actual system performance | 72% | |
| + Perfect text detection | | |
| + Perfect character segmentation | | |
| + Perfect character recognition | | |

## Ceiling analysis: *what to do next?*

Iterate from the first to the last component in the pipeline:

- assume that the component gets 100% accuracy -> and evaluate the system.

OCR pipeline: **text detection**, character segmentation, character recognition.

| Step | System accuracy | Change |
|---|---|---|
| Actual system performance | 72% | |
| **+ Perfect text detection** | 89% | +17% |
| + Perfect character segmentation | | |
| + Perfect character recognition | | |

## Ceiling analysis: *what to do next?*

Iterate from the first to the last component in the pipeline:

- assume that the component gets 100% accuracy -> and evaluate the system.

OCR pipeline: text detection, **character segmentation**, character recognition.

| Step | System accuracy | Change |
|---|---|---|
| Actual system performance | 72% | |
| + Perfect text detection | 89% | +17% |
| **+ Perfect character segmentation** | 90% | +1% |
| + Perfect character recognition | | |

## Ceiling analysis: *what to do next?*

Iterate from the first to the last component in the pipeline:

- assume that the component gets 100% accuracy -> and evaluate the system.

**Look for the biggest change in accuracy.**

| Step | System accuracy | Change |
|---|---|---|
| Actual system performance | 72% | |
| + Perfect text detection | 89% | +17% |
| + Perfect character segmentation | 90% | +1% |
| **+ Perfect character recognition** | 100% | +10% |

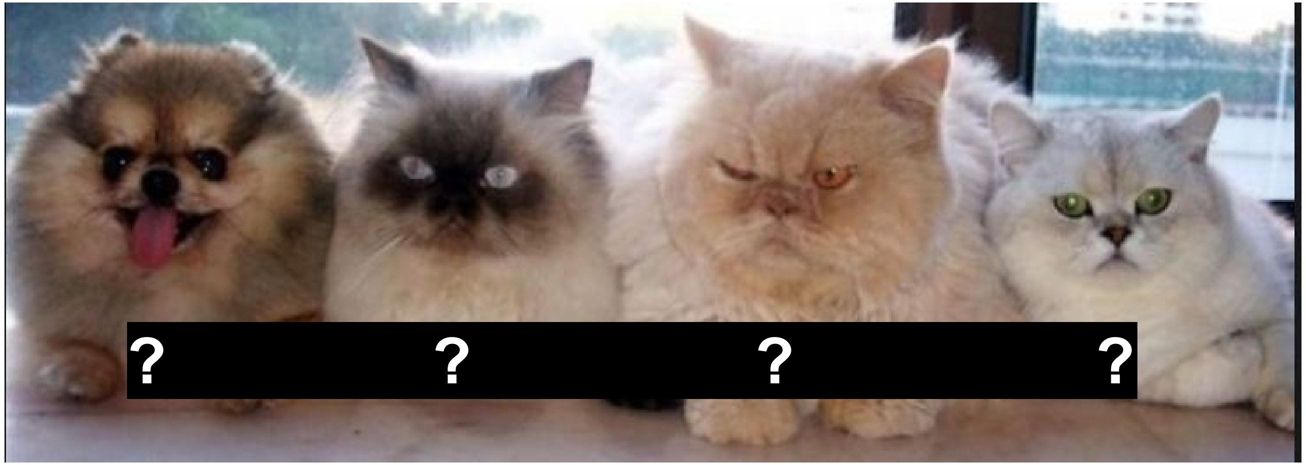
## Ceiling analysis: *what to do next?*

It shows which component significantly impacts the performance (text detection).
And which component - even if 100% accurate - won't help much (segmentation).

| Step | System accuracy | Change |
|---|---|---|
| Actual system performance | 72% | |
| + Perfect text detection | 89% | +17% |
| + Perfect character segmentation | 90% | +1% |
| + Perfect character recognition | 100% | +10% |

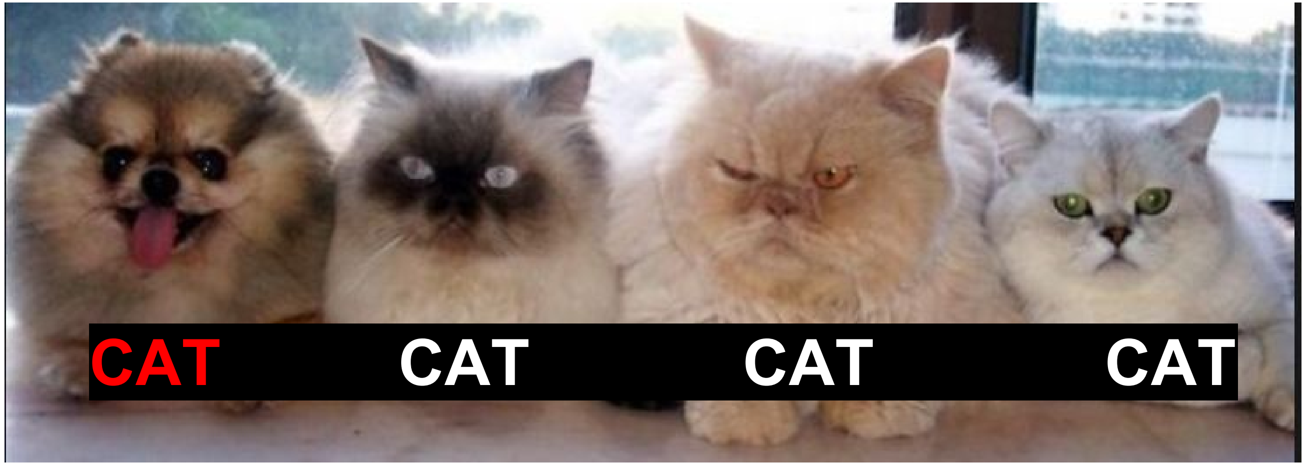
# Manual error analysis

Detect cats



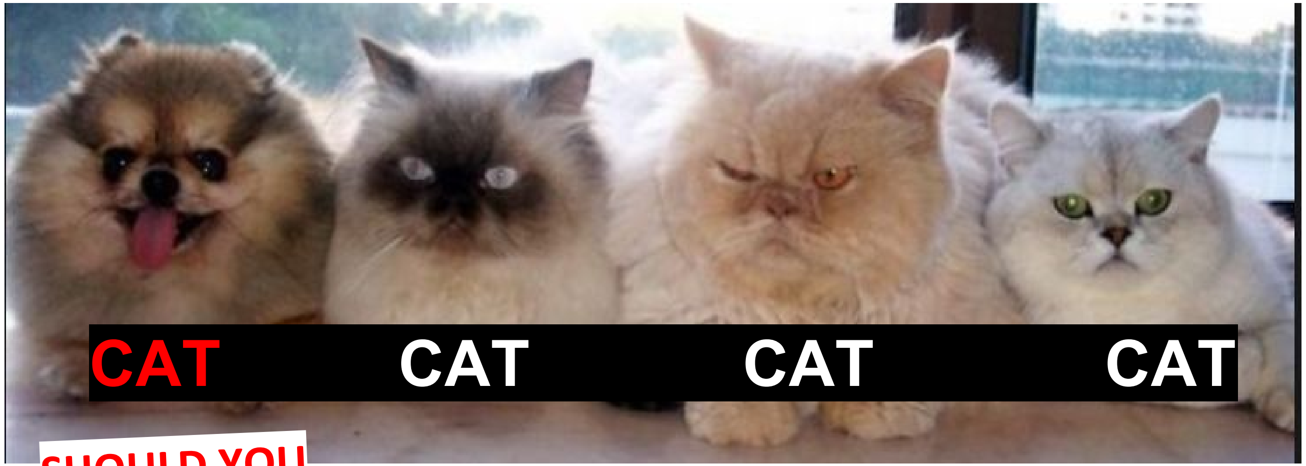
## Manual error analysis

Detect cats



## Manual error analysis

Detect cats



SHOULD YOU "FIX THIS CAT"?

# Manual error analysis

Get insight examining mistakes **manually**.

1. Get ~100 misrecognized dev set examples
2. Count how many cats are actual dogs
3. Evaluate the impact on the performance

| **EXAMPLE** | |
|---|---|
| Error rate | 10% |
| Number of dogs in the 100 misrecognized dev set examples | 5<br>(5 in 100 = 5%) |
| Estimated error rate in best case after improvement | 9.5%<br>(5% of 10% = **0.5%**) |

## Manual error analysis

### Try to quantify observed undesirable behavior

The general rule is “measure first, optimize second”

Don't fix just because someone in your team says it is important. Let the numbers speak.

### Look for patterns in the measured errors and create new features

If you give the model a feature that allows it to fix the error, the model will try to use it.

If you try to create a feature based upon examples the system doesn’t see as mistakes, the feature will be ignored.

If you are going to add post length, don’t try to guess what long means - let the model figure it out.

## Manual error analysis

You are not a typical end user - While a change which is obviously bad should not be used, anything that looks reasonably near production should be tested further, either by paying laypeople to answer questions on a crowdsourcing platform, or through a live experiment on real users. There are two reasons for this:

- ○ you are too close to the code. You may be looking for a particular aspect of the posts, or you are simply too emotionally involved (confirmation bias)
- ○ your time is too valuable. Consider the cost of 9 engineers sitting in a one hour meeting, and think of how many contracted human labels that buys on a crowdsourcing platform

- Be aware that identical short-term behavior does not imply identical long-term behavior.

## Human-level performance: *coz we are the best...*

Human-level performance tells you how well your algorithm may fit the training set

(and we know that is achievable - because humans can do it)

Human-level error serves as an approximation for Bayes error rate (the minimal error)

**Human-level error**

Avoidable bias

**Training error**

Variance

**Dev error**

## Reduce bias or variance: *what to do next?*

| | **Project Alpha** | **Project Beta** |
|---|---|---|
| Training set error | 8% | 8% |
| Dev set error | 10% | 10% |

So, chief, should I work on reducing bias or variance in these projects?

## Reduce bias or variance: *what to do next?*

| | **Project Alpha** | **Project Beta** |
|---|---|---|
| Training set error | 8% | 8% |
| Dev set error | 10% | 10% |
| **Human-level error** | **1%** | **7,5%** |

**Two fundamental assumptions of supervised learning:**

You can fit the training set pretty well (via working on reducing avoidable bias)

The training set performance generalizes pretty well to the dev/test set (via working on reducing variance)

## Reduce bias or variance: *what to do next?*

| | **Project Alpha** | **Project Beta** |
|---:|:---:|:---:|
| Training set error | 8% | 8% |
| Dev set error | 10% | 10% |
| **Human-level error** | **1%** | **7,5%** |

Try reducing **bias** (training set error)

Try reducing **variance** (dev set error, generalization)

## Reduce bias or variance: *but how?*

**To reduce avoidable bias (human-level error vs training set error)**

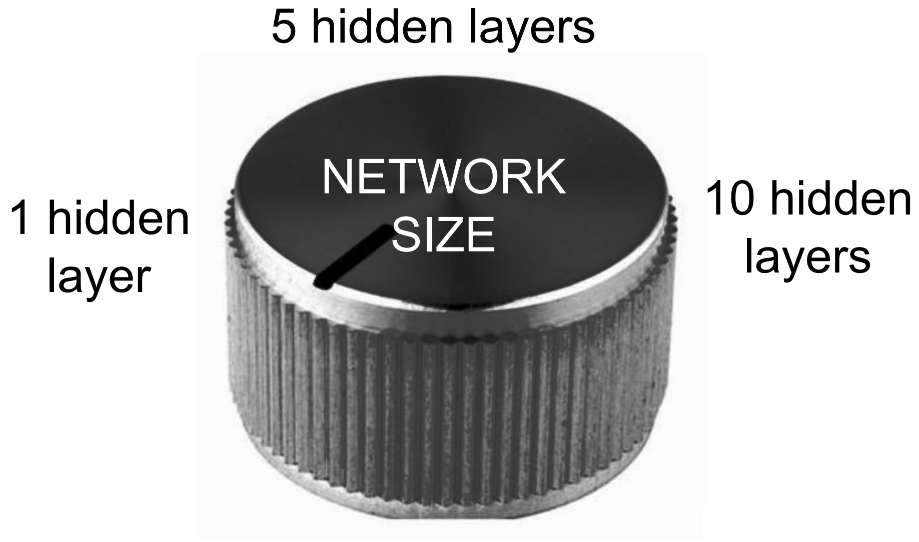
- Train bigger/higher dimensional model
- Use another optimization algorithm (Momentum, RMSProp, Adam etc.)
- Try another NN architecture/change hyperparameters (RNN, CNN, LSTM etc.)

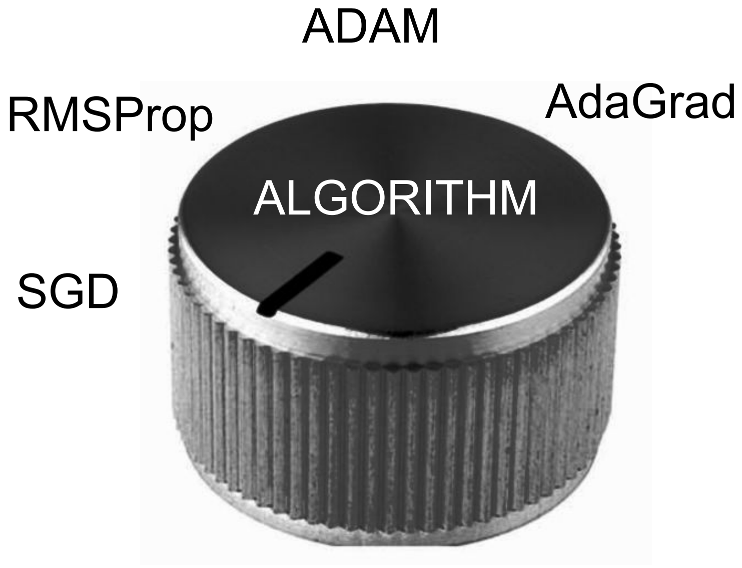
**To reduce variance (training error vs dev set error)**

- Get more data (to generalize better)
- Use regularization (L2, dropout, data augmentation - add noise)
- Try another NN architecture/change hyperparameters

# Orthogonalization in supervised learning

To fit training set well (eg. achieve human-level performance)

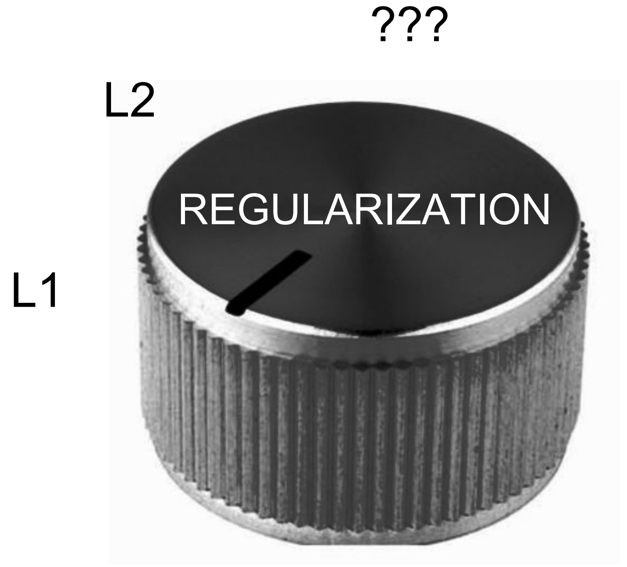




# Orthogonalization in supervised learning

Fit dev set (model fits the training set well but fails to generalize)

1k examples
10 examples
TRAINING SET SIZE
1M examples



## Orthogonalization in supervised learning

Fit test set (model fits train and dev sets, but fails on test set)

*you probably overfit on the dev set, try to add more examples to it*

## Orthogonalization in supervised learning

Your model fits the training, dev and test set,
but doesn't perform well in real world

Does the data (distribution)
reflect the real world distribution?

Does the metric correctly
reflect the real goal?

## Step 6. Increase complexity

If you have the time and budget, you can investigate more complex methods and there are worlds of complexities that are possible.

Two common methods to consider:

**end-to-end training** and **ensembles**

## Step 6. Increase complexity

DO YOU HAVE IT?

If you have the time and budget, you can investigate more complex methods and there are worlds of complexities that are possible.

Two common methods to consider:

**end-to-end training** and **ensembles**

## Step 6. Increase complexity

***SORRY, NO TIME***

***WORK IN PROGRESS!***

***COME BACK LATER :)***

## Conclusion?

**THEY DAY IS TOO SHORT**