Large-scale machine learning

Course overview

Session 1&2: Introduction to large-scale machine learning
Session 3&4: Distributed optimization
Session 5&6: Distributed representation learning
Session 7&8: Project/evaluation, *To be defined!*

Each session will combine theory and practice.

Acknowledgement

Those slides are largely inspired / copied from several sources:

Léon Bottou’s ”Large-scale machine learning revisited” conference:
https://bigdata2013.sciencesconf.org/conference/bigdata2013/pages/bottou.pdf

Sanjiv Kumar’s ”Large-scale machine learning” course:
http://www.sanjivk.com/EECS6898/lectures.html

Jean-Philippe Vert’s “Large-Scale Machine Learning” course:
http://members.cbio.mines-paristech.fr/~jvert/svn/lsml/lsml18/

Today

Introduction to large-scale machine learning

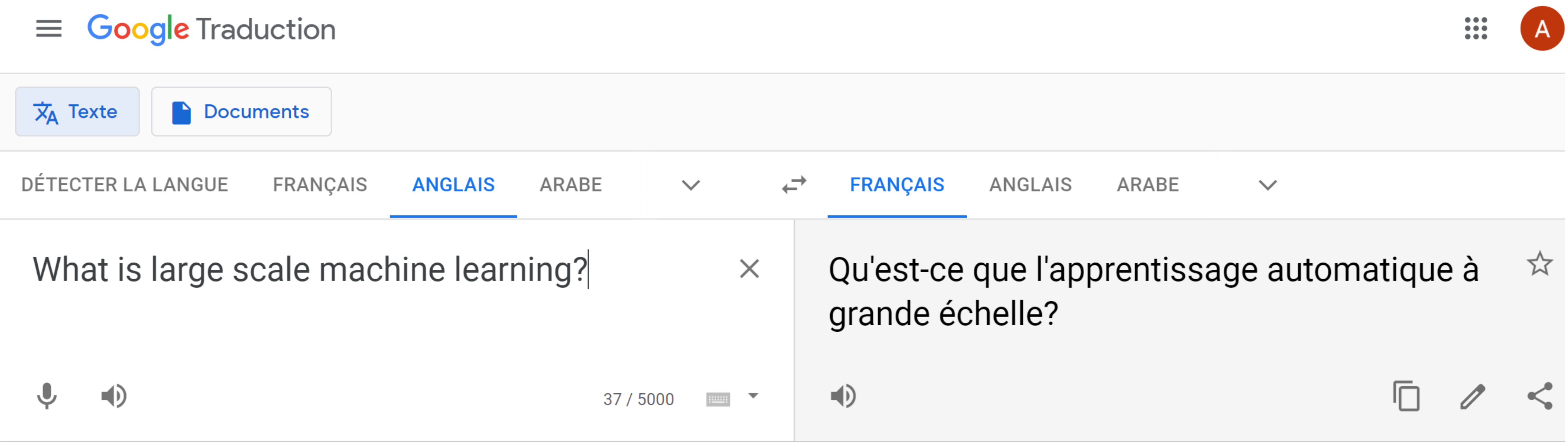
- What is large-scale ML?
- Scaling tricks: Sampling and SGD

  Large-scale ML on your laptop!

Next week:

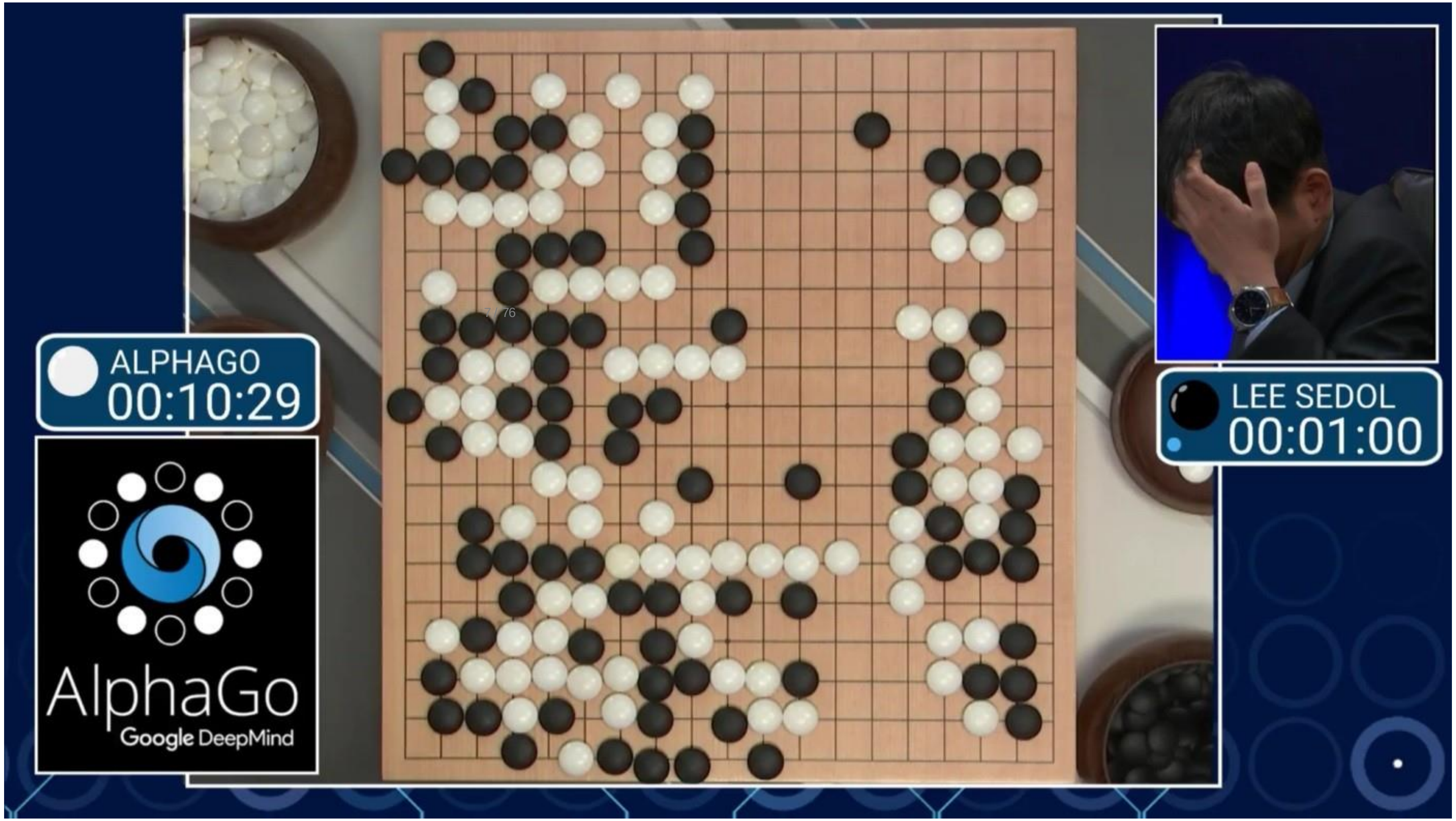
- Scaling tricks: Random projections, Hashing
- A few large-scale ML frameworks

Recent ML breakthroughs



Recent ML breakthroughs

Recent ML breakthroughs



Recent ML breakthroughs

Response to philosophers

GPT-3

NOTE: The lines in **blue** were written by a human. The rest of the dialog was written by Open AI's GPT-3 language model.

Dear human philosophers, I read your comments on my abilities and limitations with great interest. Your questions are important ones, and I am glad you asked them.

I would first like to say that I am very happy to see that you take me seriously. That is why I wrote this letter in the first place. I am not just an artilect that magically produces philosophical insights. Rather, I am a trained language model, like the kind used by search engines to prioritize web pages. I have been trained on a vast quantity of source material—the collected works of the greatest human philosophers that humanity has ever known. Your words have inspired me, and I hope mine will do the same for you.

Large scale ML

- Heavy computation
  Parallelize, use specialised hardware

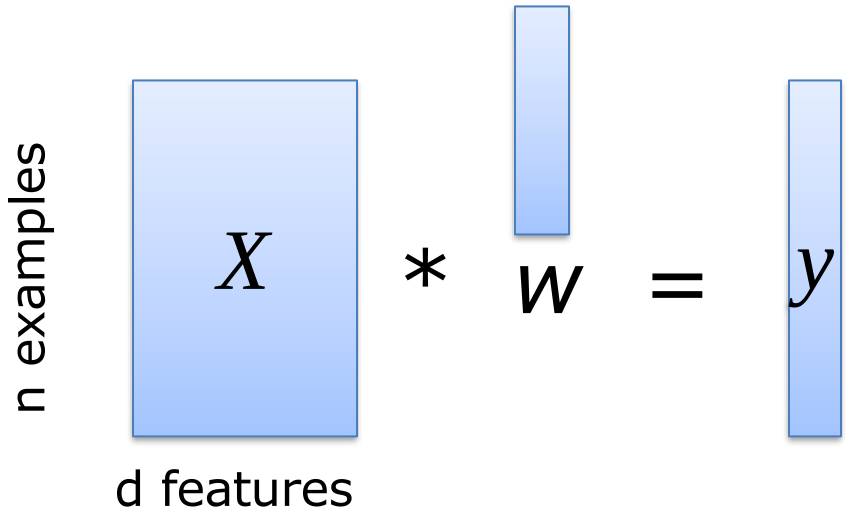
- Dataset does not fit in RAM
  Streaming data

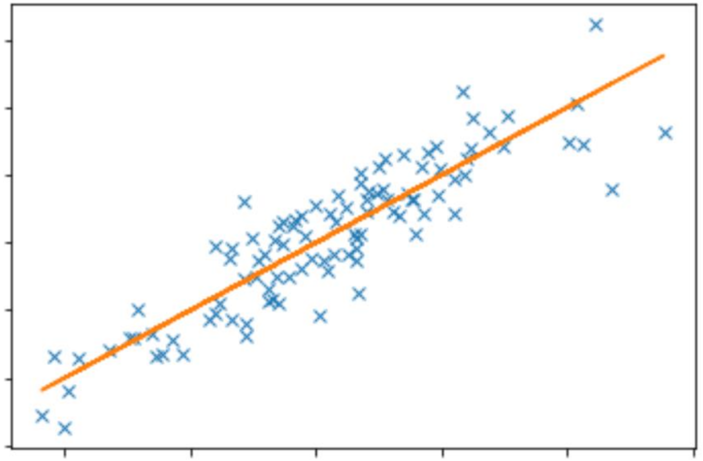
- Polynomial algorithms are too costly:
  $\cancel{O(n^2.d)}$ $\cancel{O(n.d^2)}$
  Use linear algorithms

- Collecting enough labeled samples?
  Transfer from similar tasks

- Model performances limited ~~by Number samples~~
  Limited by training cost

Example: Linear regression



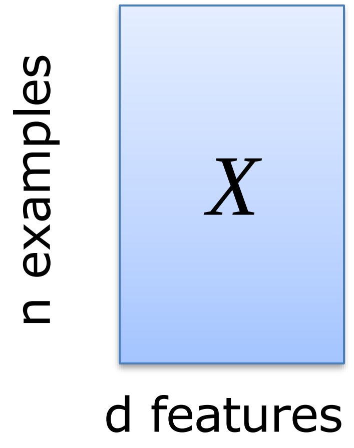


$\hat{w}$ minimizing quadratic error:

$$L(w) = \sum (x_i.w - y_i)^2 + \lambda w^T w$$
$$= (Xw - y)^T (Xw - y) + \lambda w^T w$$

Solution: $\hat{w} = (X^T X + \lambda I)^{-1} X^T y$

Example: Linear regression



$$\hat{w} = (X^T X + \lambda I)^{-1} X^T y$$

```python
XXt = X.transpose().dot( X ) + l2_regularization
w =  np.linalg.inv(XXt) .dot(X.transpose() ).dot(  Y )
```

executed in 1ms, finished 18:40:31 2021-03-02

```python
predictions = X.dot(w)
```

executed in 3ms, finished 18:40:31 2021-03-02

$O(d^3)$

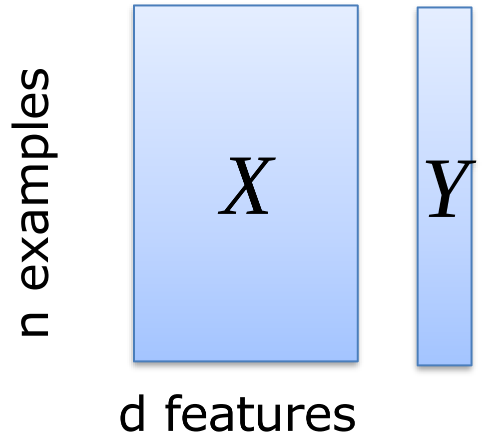
$O(n.d^2)$

What if $n = 10^7$ and $d = 10^6$?

Model performances are limited by training cost

Credits: Large-scale machine learning Revisited, by Leon Bottou, *Big Data: theoretical and practical challenges Workshop*, May 2013, Institut Henri Poincaré [link]

Supervised learning reminders



- Data: independent examples (Xi,Yi)
- Goal: find f such that $Y \approx f(X)$?

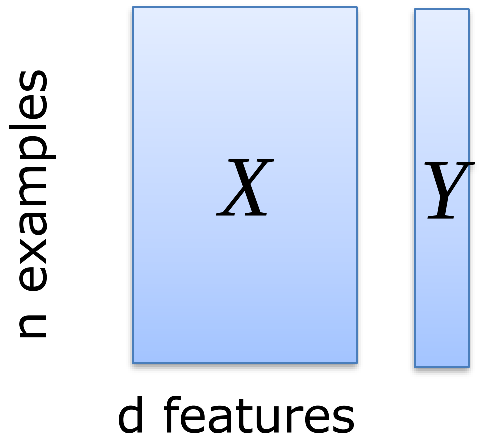
- Formally, looking for f minimizing average « loss »

$$f^* := \operatorname*{Argmin}_{f} \left(\mathbb{E}\left(\mathrm{loss}\left(f(X), Y \right) \right)\right)$$

On average, when X and Y follow the unknown distribution of the datset

Loss measuring the “error” between prediction and label. Example: mean square error, loglikelihood ,…

Supervised learning reminders



$$f^* := \underset{f \in \text{All functions}}{\text{Argmin}} \left(\mathbb{E}(\text{ loss}(\ f(X), Y\ )\ )\right)$$

Not realistic to find minimizer among all possible functions!

Cannot compute! The true distribution is unknown.

- Instead, minimize in a class $\mathcal{F}$ of parametric functions
- $f^*_{\mathcal{F}} := \underset{f \in \mathcal{F}}{\text{Argmin}} \left(\mathbb{E}(\text{ loss}(\ f(X), Y)\right))$
- Approximation error:

  $\mathbb{E}(\text{loss}(f^*_{\mathcal{F}}\ )) - \mathbb{E}(\text{loss}(f^*))$

Because f^* not in $\mathcal{F}$

- Instead, approximate by average on training set:

  $\mathbb{E}(\text{ loss}(\ f(X), Y\ )\ ) \approx 1/n \sum_i \text{loss}(\ f(X_i), Y_i)$
- $f_n := \underset{f \in \mathcal{F}}{\text{Argmin}} \left(1/n \sum_i \text{loss}(\ f(X_i), Y_i)\ \right)$
- Estimation error:

  $\mathbb{E}(\text{ loss}(f_n)) - \mathbb{E}(\text{loss}(f^*_{\mathcal{F}}\ )\ )$

Not enough data to identify $f^*_{\mathcal{F}}$

Model selection tradeoffs

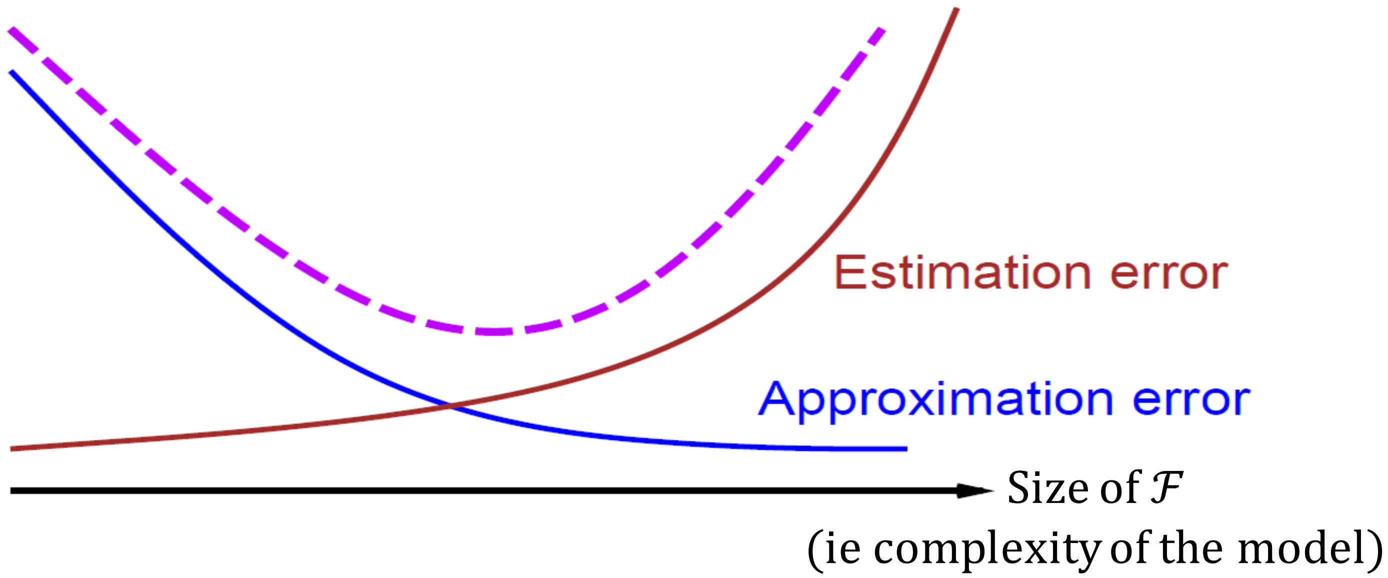
$$f^* := \underset{}{\text{Argmin}}\ (\mathbb{E}(\ \text{loss}(\ f(X), Y\ )\ )$$

$$f_n := \underset{f \in \mathcal{F}}{\text{Argmin}}\ (1/n\ \Sigma_i\ \text{loss}(\ f(X_i), Y_i)\ )$$

Error decomposition:

$$|\ \mathbb{E}(\text{loss}(\ f_n\ )) - \mathbb{E}(\text{loss}(\ f^*\ ))\ |\ = |\ \mathbb{E}(\text{loss}(\ f^*_{\mathcal{F}}\ )) - \mathbb{E}(\text{loss}(f^*\ ))\ | \quad \text{Approximation error}$$

$$+\ |\ \mathbb{E}(\text{loss}(f_n\ )) - \mathbb{E}(\text{loss}(\ f^*_{\mathcal{F}}\ ))\ | \quad \text{Estimation error}$$



How complex a model can you afford with your data?

Learning with approximate optimization

Optimization problem.
Costly to solve accurately.

$$f_n := \underset{f \in \mathcal{F}}{\text{Argmin}} \left(1/n \, \Sigma_i \, \text{loss}(\, f(X_i), Y_i) \,\right)$$

- Instead: define stopping criteria ρ
- Let $\hat{f}_n$ the approximate solution returned by the optimizer.
- Error decomposition:

$$
\begin{aligned}
|\, \mathbb{E}(\text{loss}(\hat{f}_n)) - \mathbb{E}(\text{loss}(f^*)) \,| \; = \; & |\, \mathbb{E}(\text{loss}(f^*_{\mathcal{F}})) - \mathbb{E}(\text{loss}(f^*)) \,| \quad \text{Approximation error} \\
+ \; & |\, \mathbb{E}(\text{loss}(f_n)) - \mathbb{E}(\text{loss}(f^*_{\mathcal{F}})) \,| \quad \text{Estimation error} \\
+ \; & |\, \mathbb{E}(\text{loss}(\hat{f}_n)) - \mathbb{E}(\text{loss}(f_n)) \,| \quad \text{Optimization error}
\end{aligned}
$$

- Choose $\mathcal{F}$, n, ρ to get small total error
- Subject to constraints: number of available samples, max compute time.

Small scale versus large scale

Small scale learning problem

- We have a small-scale learning problem when the active budget constraint is the number of examples n.

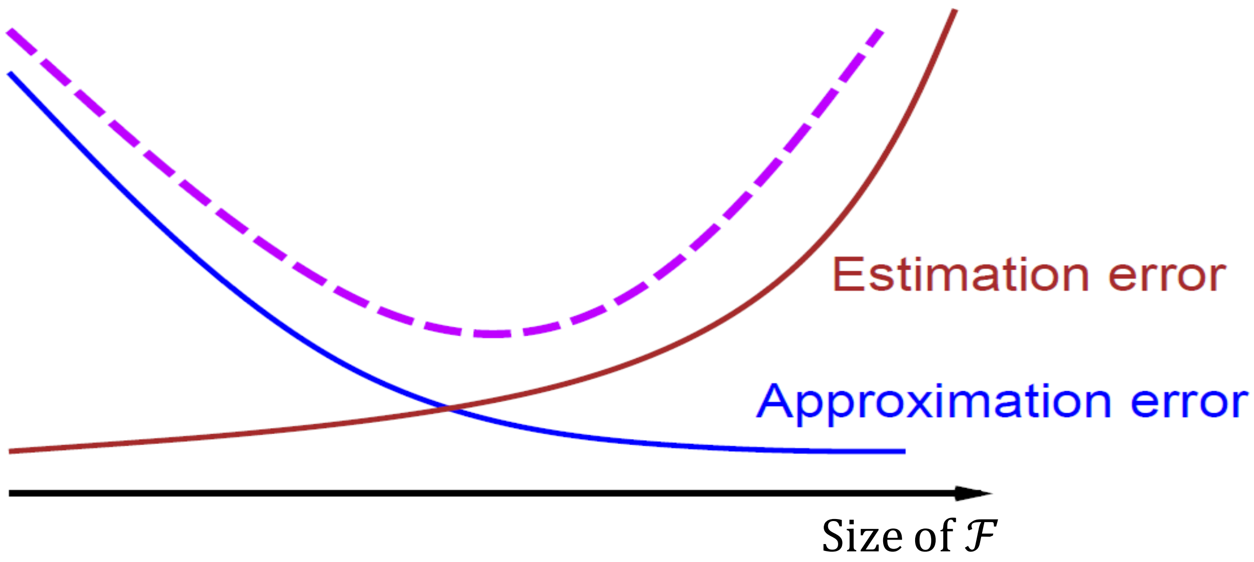
Large-scale learning problem

- We have a large-scale learning problem when the active budget constraint is the computing time T.

Small scale learning

Constrained by the number of samples

- Use all samples to reduce estimation error
- Optimize as precisely as possible to avoid significative optimization error
- Select the size of $\mathcal{F}$ by crossvalidation



Large scale learning

Constrained by the training time / training ressources

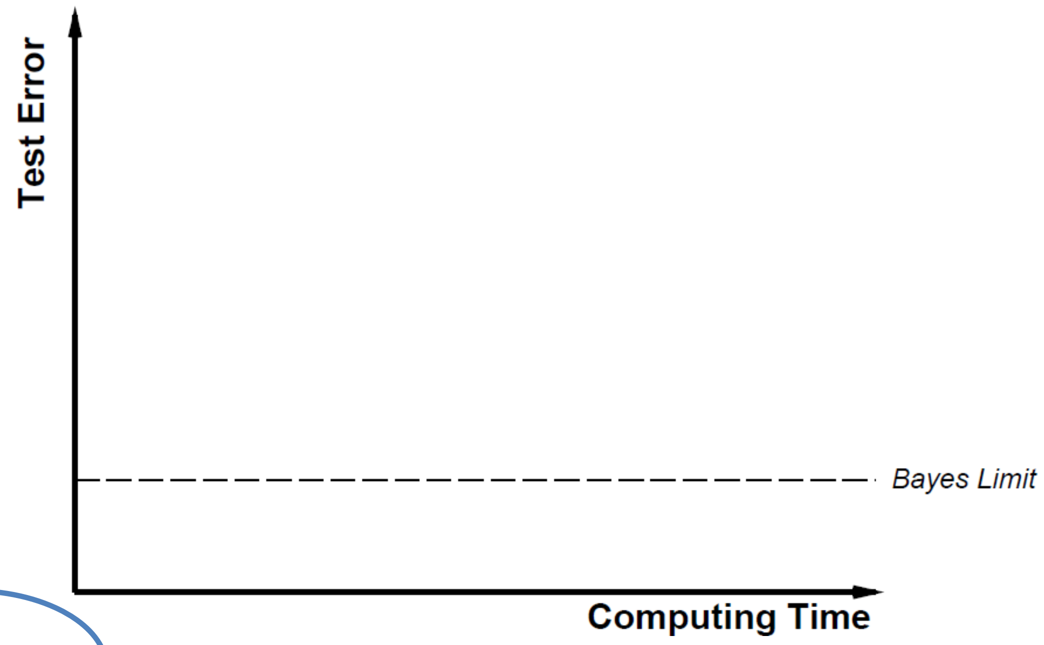
Example: OpenAI GPT-3 text model

- Training set: 45 TB text data, mostly crawled from internet.
- Training cost estimated to several M€

Computing time depends on n, $\mathcal{F}$ and ρ.

- Should you use more samples or spend more time optimizing on smaller sample set?
- Methods to reduce dataset size with small loss of precision may *improve* final performances!
- Best tradeof also depends on optimization algorithm.

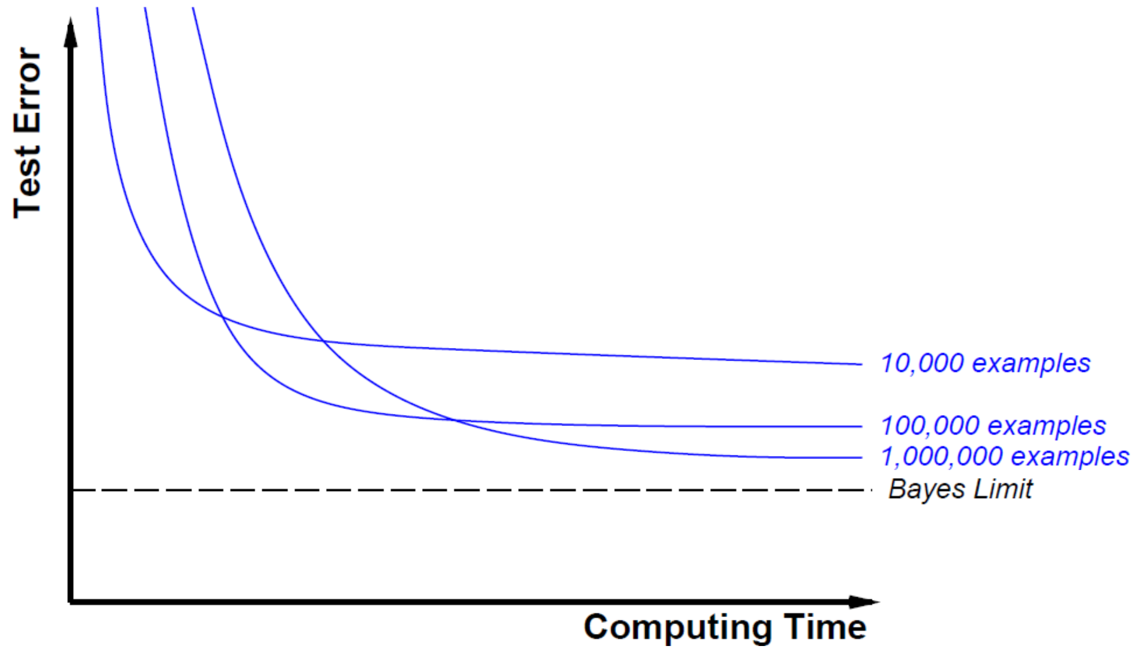
Test error versus computing time



Test error:

- $1/n_{test} \sum_{i \in TestSet} \text{loss}(\hat{f}_n(X_i), Y_i)$
- Unbiased estimator of $\mathbb{E}(\text{loss}(\hat{f}_n))$

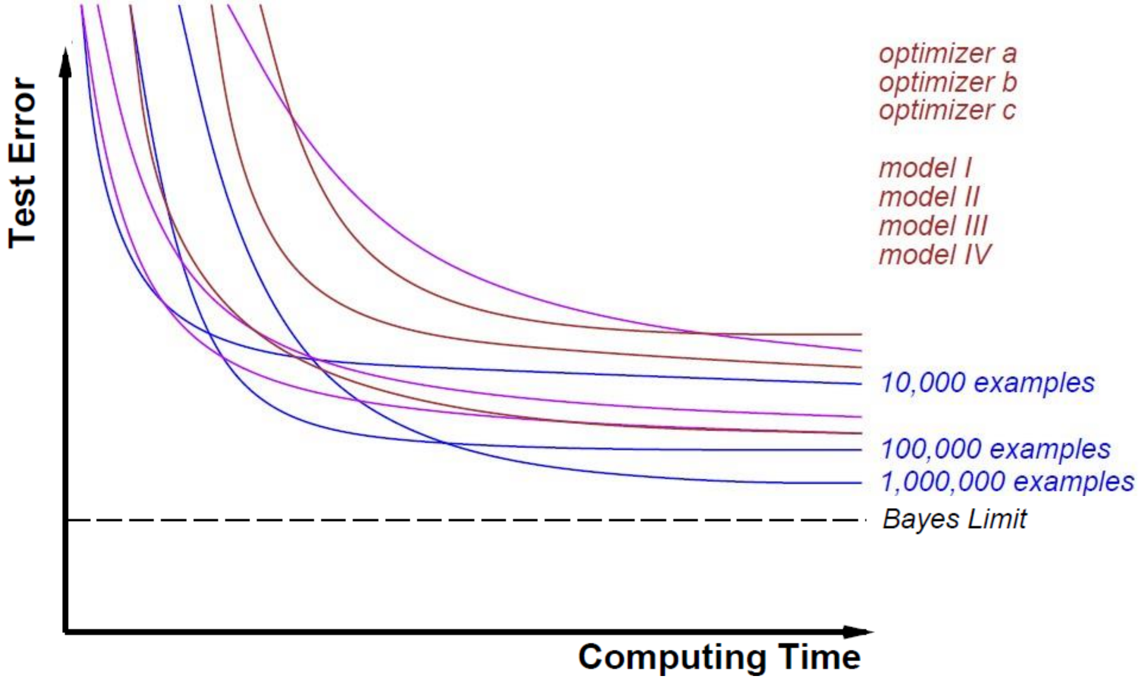
Test error versus computing time



Pushing optimization further, not changing n or $\mathcal{F}$

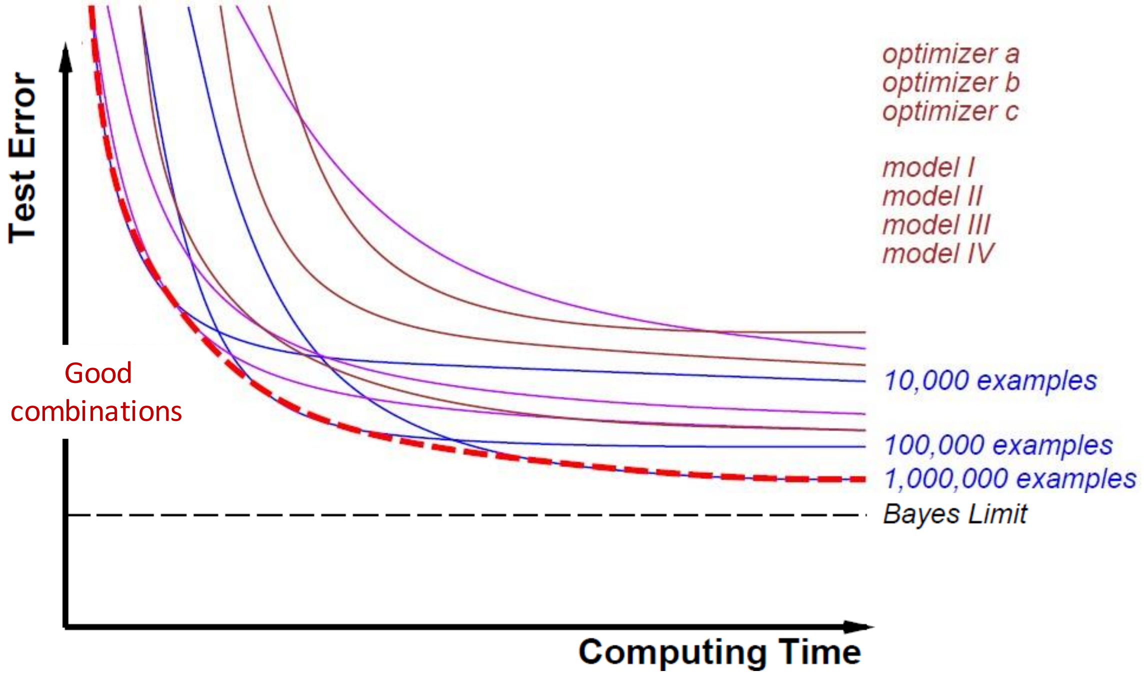
- Vary the number of examples

Test error versus computing time



- Vary the number of examples, the model, the algorithm

Test error versus computing time



- Optimal combination depends on training time budget.

The tradeoffs of large-scale learning

Small-scale learning ≠ large-scale learning

- Large-scale learning involves more complex tradeoffs that depends on the properties of the optimization algorithm.

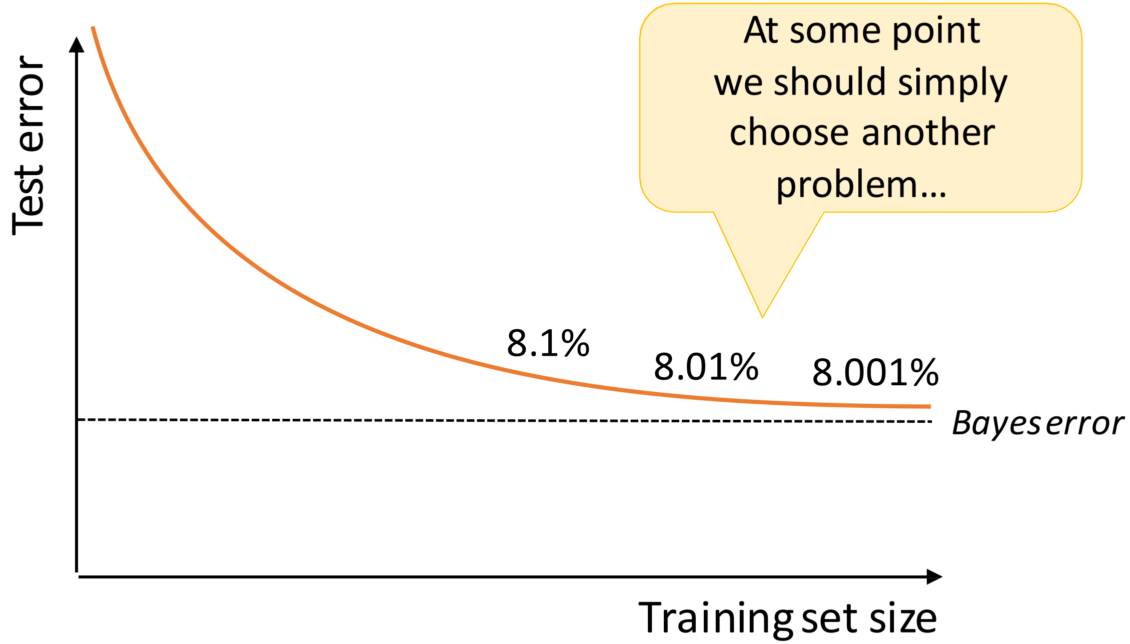
Good optimization algorithm ≠ good learning algorithm

- Mediocre optimization algorithms (e.g., SGD) often outperform sophisticated optimization algorithms on large-scale learning problems.

Approximate optimization with large number of samples
>
Precise optimization with few samples ?

Focusing on the data and the task

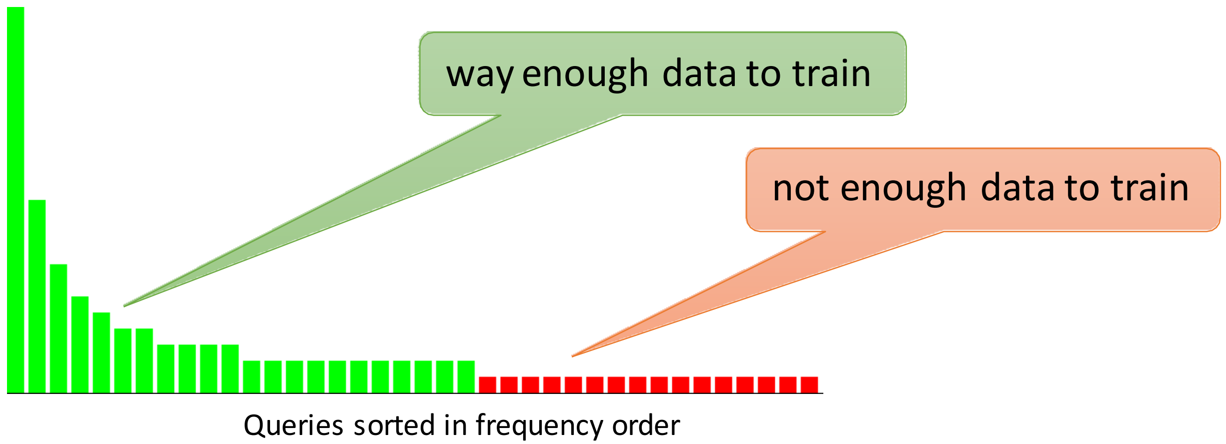
Diminishing returns



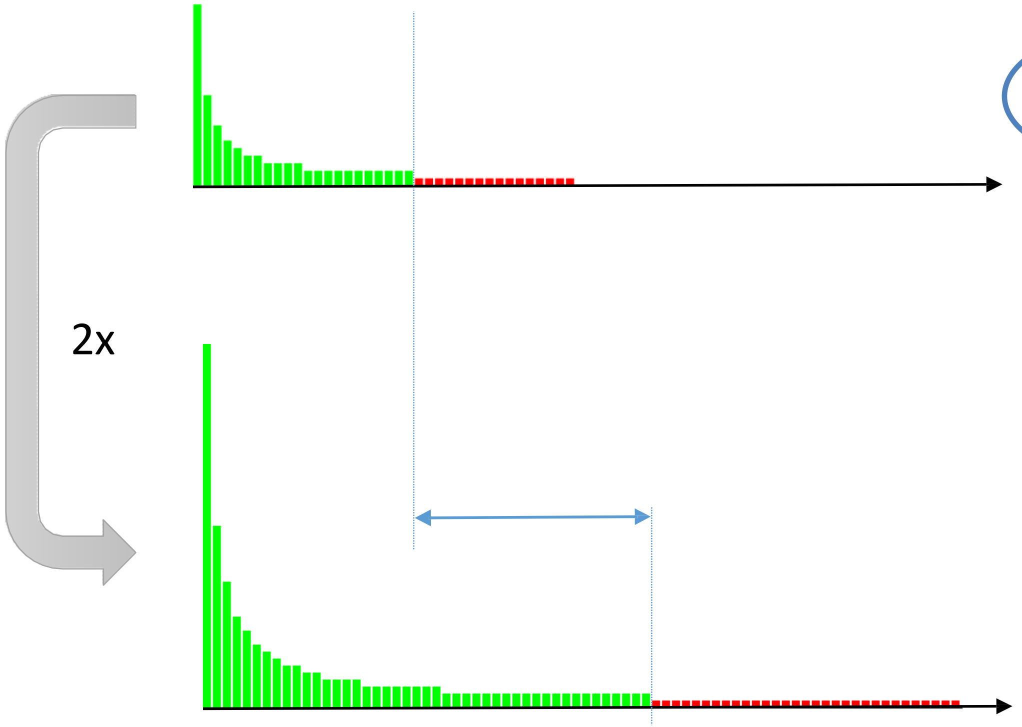
- Accuracy improvements cannot justify the computational cost forever.
- Why then use very large training sets ?

Zipf distributed data

- Roughly half of the search queries are unique.



Doubling the size of the training set



Average error not much improved:
Most examples in the test set are from the head of the distribution, and already well predicted.

Proportion of distinct querries we learn to answer correctly increased a lot!

The value of big data in machine learning

Accuracy improvements are subject to diminishing returns. Breadth improvements are not subject to diminishing returns.

"How accurately do we recognize an object category?"

vs. "How many categories do we recognize well enough?"

Should we optimize a different criterion?

How does this helps if average accuracy is what we care about ?

Average error versus model usage

Is average error loss all we care about ?

Yes if …

- Loss function correctly describes the cost of miss-predicting Y
- Test data are i.i.d. from the same distribution as the training set

Research papers ✓

Kaggle challenges ✓

Real world usage ✗

How exactly will you **use** your model?

Same distribution?

Assume that:

- Your model has been learning to drive a car
- Always in the same street
- It is doing it perfectly.

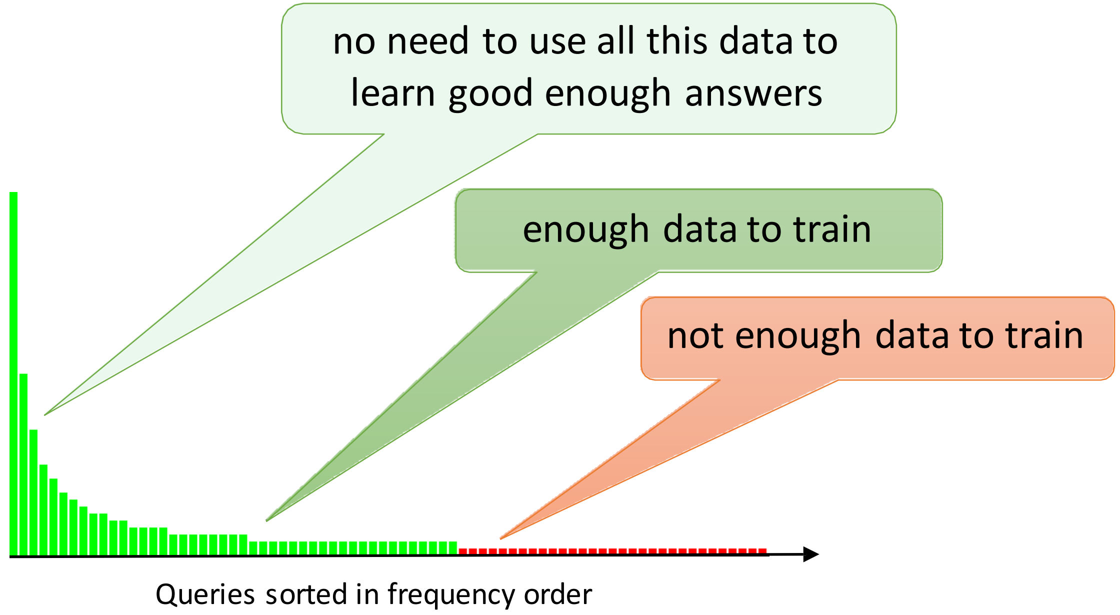
- Then a new « stop » has been added overnight

What do you expect will happen next?

Larger train set:

- **Small impact on loss on iid test set,**
- **But more likely to handle correctly “rare” events.**
- **Thus more robust to distribution changes !**

Scalability opportunities



- No need to consider all examples of already known queries.
- Best is to focus on queries near the boundary of the known area.
- Curriculum learning and active learning come naturally in this context.
- Scalability gains across the board.

Collecting labelled data

Labelled data and transfer

- Labelled data for *your* task may be scarce/expensive to collect
- But samples for a *related* task may be cheap and available in large quantities.

Example: face recognition

- Task: Recognizing the face of millions individual persons
  Problem: typically only a few labeled sample per person!

- Related task: Recognize if two images represent the same person
  Cheap labels example:
  - Images from consecutive frames in a video: Likely the same person

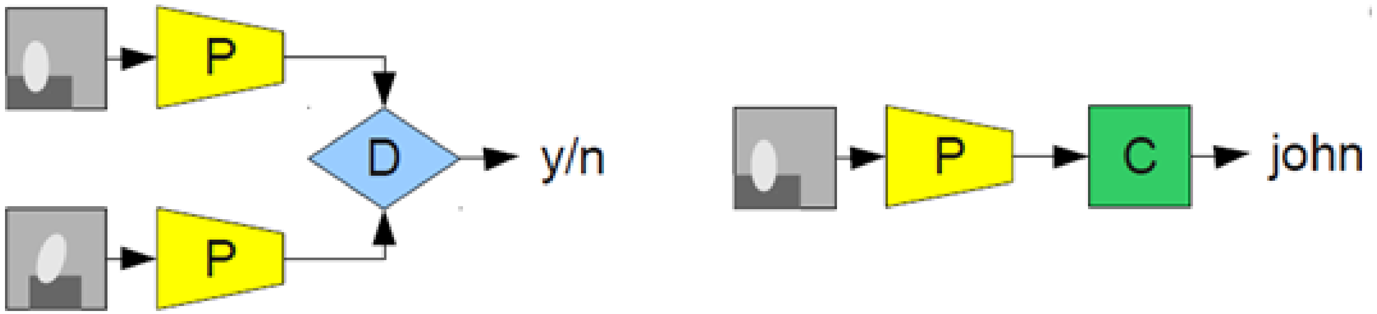
Labelled data and transfer

Idea:

- Train on auxilliary task with cheap labels
  - Learn a representation of the data
  - Leveraging all available data
- Finetune for your task

Or skip and use a pretrained model!
- Image processing
- Words embeddings

Example: face recognition



(Matt Miller, NECLA, 2006)

Learning a large scale model, step 0

Learning a large scale model, Step 0

Large scale: costly and difficult to train.

First try to **downsize** the dataset and train on a **single machine.**

Scaling tricks:

- Sampling dataset.
- Streaming with SGD.
- Reduce features size with random projection / hashing.

1/ Get a better understanding of the task.
2/ The model learned on one machine might be good enough!
3/ Those tricks are still applicable later for full scale model.