# Do we always have to start from scratch?

## Or is there a better way?

Levente Szabados

Frankfurt School of Finance and Management

AI Partners

# Motivation:

"- We would like to have a chatbot!"

"- Sure! Do you have enough data?"

"- Absolutely, we are overwhelmed!'

```
(Turns out to be 150 emails...)
```

"- OOOK, wel, then, let's see what we can do?"

**Options:**

1. Try to learn a whole language with deep meaning from 150 emails?
2. Give it up and go for a walk?
3. ????

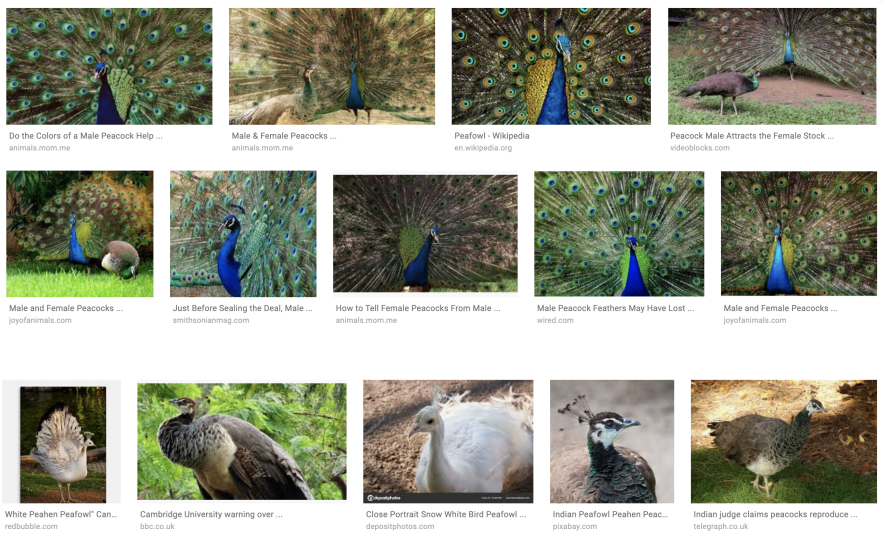
## Sidenote:

It is in itself a more nuanced problem, if we have enough data or not, see:

“Do I have enough data for Machine Learning?” — Um… maybe…? (https://medium.com/@haomiao/do-i-have-enough-data-for-machine-learning-um-maybe-d45f41234d2d)

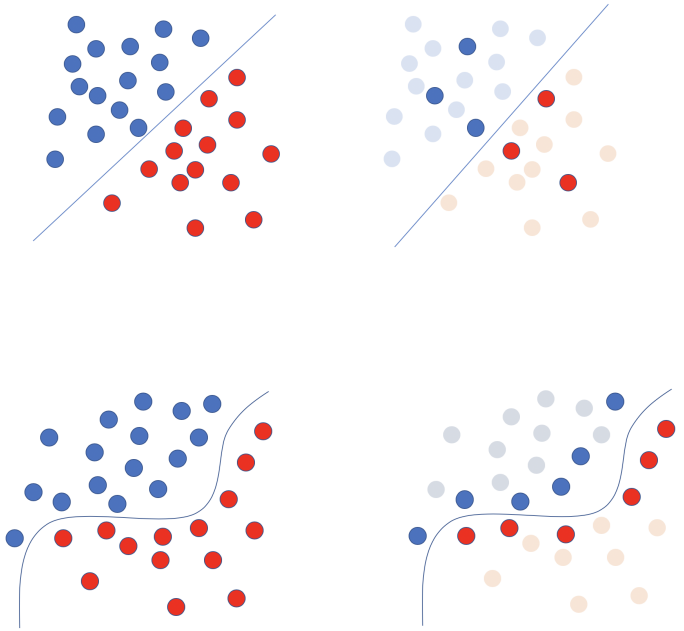
The answer is: **"It depends."**

And it does **not** just depend on dimensionality.



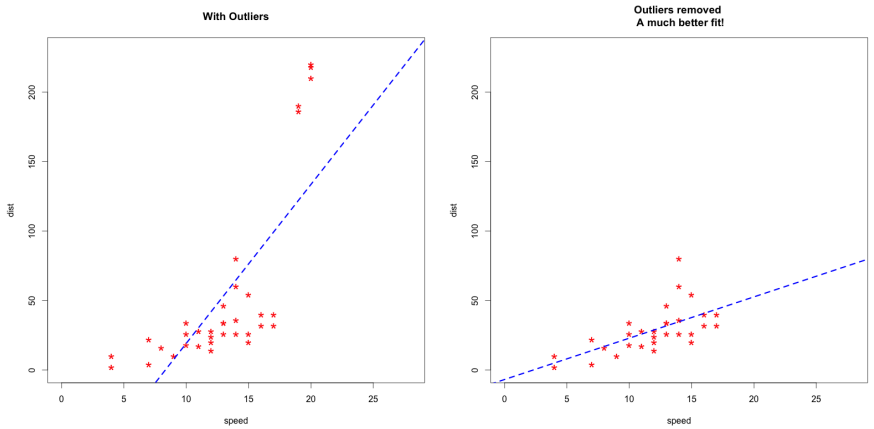

You can simply **distinguish peacocks from peahen** by measuring the **amount of green** on the picture - even though the picture is of high dimensionality (as many as number of pixels).

The amount of data is roughly proportionate on **how difficult decision you have to make**.



Which in many cases can be only found out by trying.

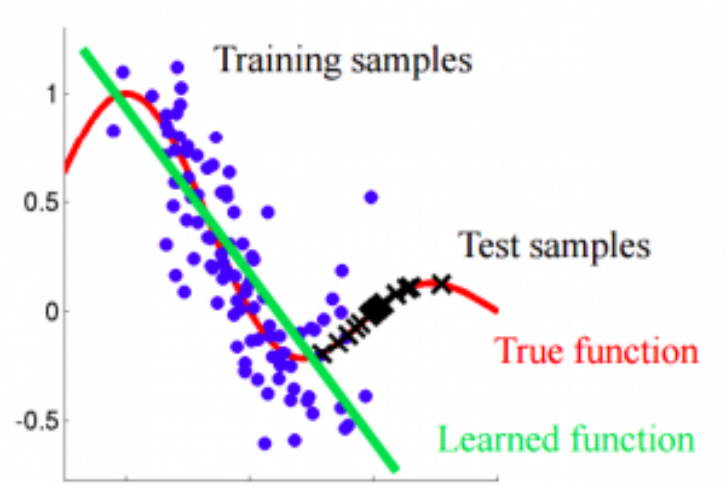
It is important to note that **all learning is a form of memorization**.




We would like to "distill" the essence, remember what is the **defining characteristic** of the data, and **disregard "noise"**, eg. outliers.

This in turn has strong connections with **overfitting**, since if we "memorize" also the **"unimportant", random properties** of the data, we will have a **bad generalization** performance out of sample

In turn, good generalization is crucial in case of **"covariate shift"**, when the data distribution changes **out of sample**.

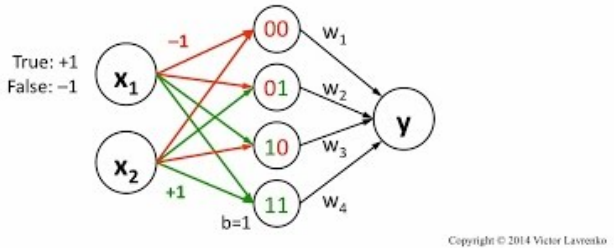



That is: ALWAYS :-)

So if we have learned the right function, we can still generalize.

## Representation Power of NNs

- NN with 1 hidden layer can represent:
  - any bounded continuous function (to arbitrary ε)
    - Universal Approximation Theorem [Cybenko 1989]
  - any Boolean function (exactly)





And since Deep Learning models have **huge memory capacity**, we would like to see if they can memorize some **generally useful patterns** across domains!

That is: train a model on an abundant, general dataset, (preferredly unsupervised), and apply it to the task at hand.

The other answer to the enough data question is:

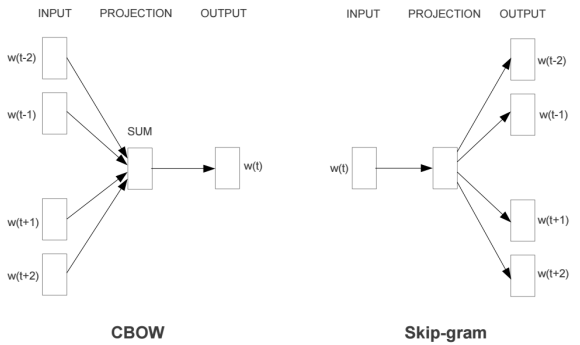
**"Look at what others did, maybe you get some help!"**

This latter strategy we will follow. :-)

# 1. Static representation transfer:

Learn a shallow representation, build a model on it.

Example: word2vec + SVM

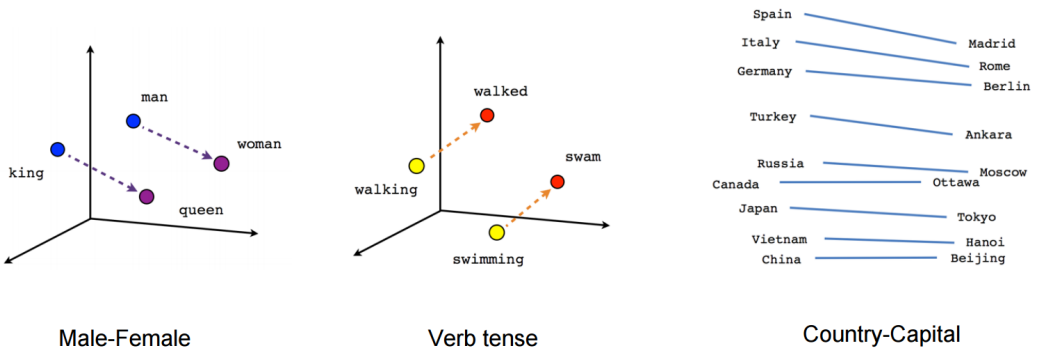



Mikolov et. al. substituted the large matrix decomposition task in language representation learning to a local context prediction task.

The main aim was: **learning a good representation** by solving an (uninteresting) unsupervised problem.

# Representation

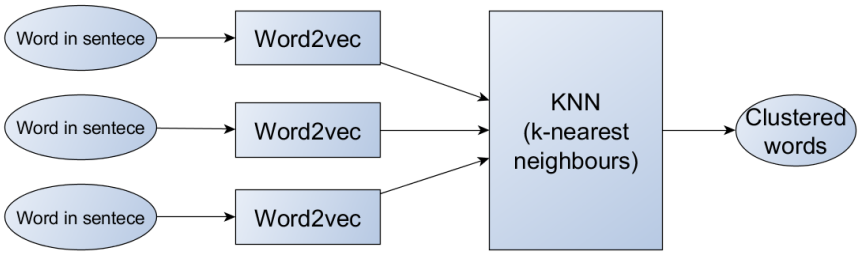
**Word2vec** as a quasi side-effect of the prediction task learns systematic mapping of word syntax and semantics to a dense vector space.




Male-Female Verb tense Country-Capital

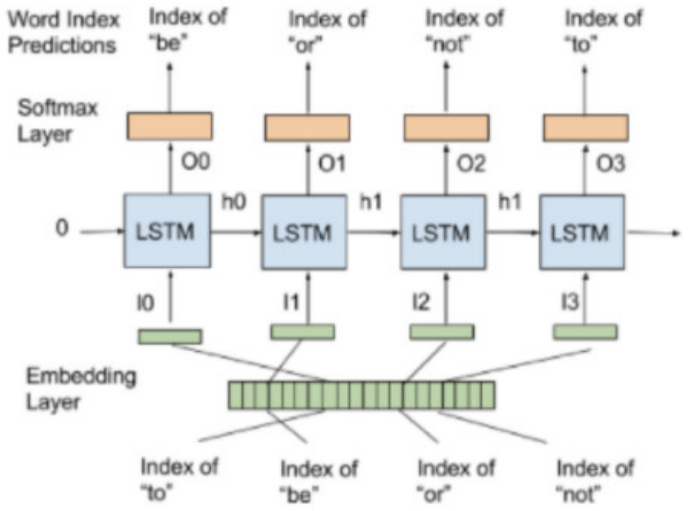
A stable decision boundary in this space is good possible, eg. with a large margin classifier like SVM.

In this approach, representation is passively transferred, it is not modified during the learning of the target task.



# 2. End-to-end training with transferred representations

Example: Initialize embedding layer of a deep network with word2vec, then let it train further



**Pro:**

- Easy to implement
- Gives visible boost to models

**Con:**

- Only "shallow" knowledge has been transferred
- With initial updates some of the information gets destroyed

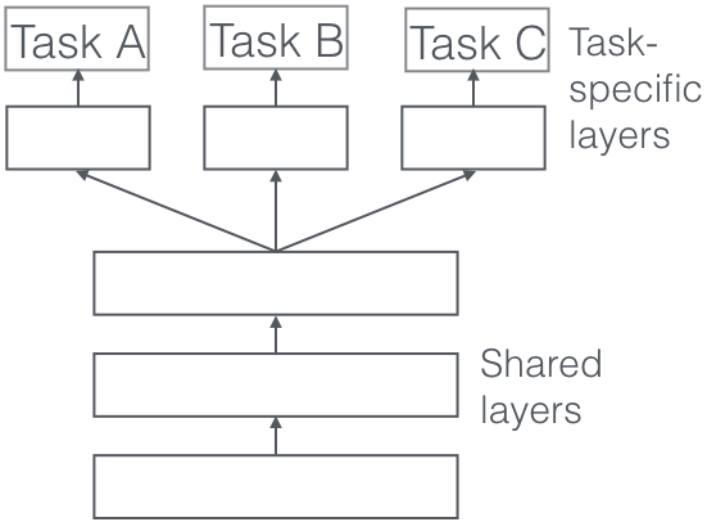
**Forgetting starts to pop up!**

# 3. Deep representation transfer

The general approach on deep representation transfer capitalizes on the fact, that learned representations for neural models are typically having two somewhat distinguishable structural elements:

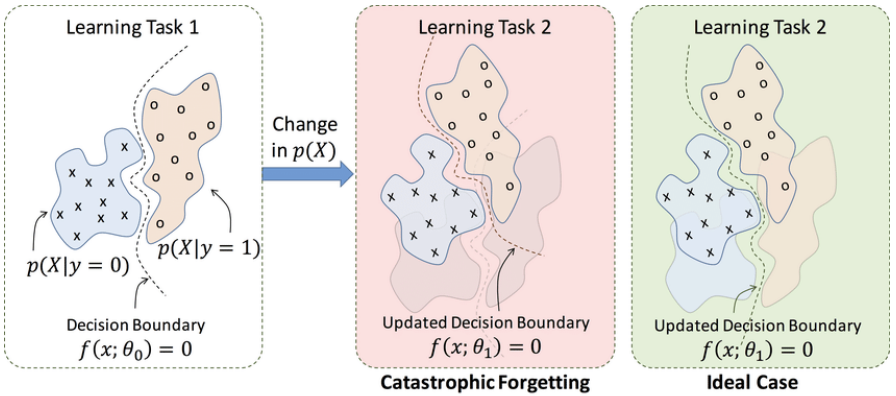
- **the first** (MANY!) **layers are for representation learning** ("embedding")
- **the last** (typically 1 or so) **layers are for classification** ("cut", ie. a linear classifier)

With this in mind, we can try to transfer a complete representation "stack", and only **replace the last decision layer**.




(Please observe, that nothing prevents us from parallel, multi-task learning at this point! Hence the picture above.)

# The problem: Catastrophic forgetting

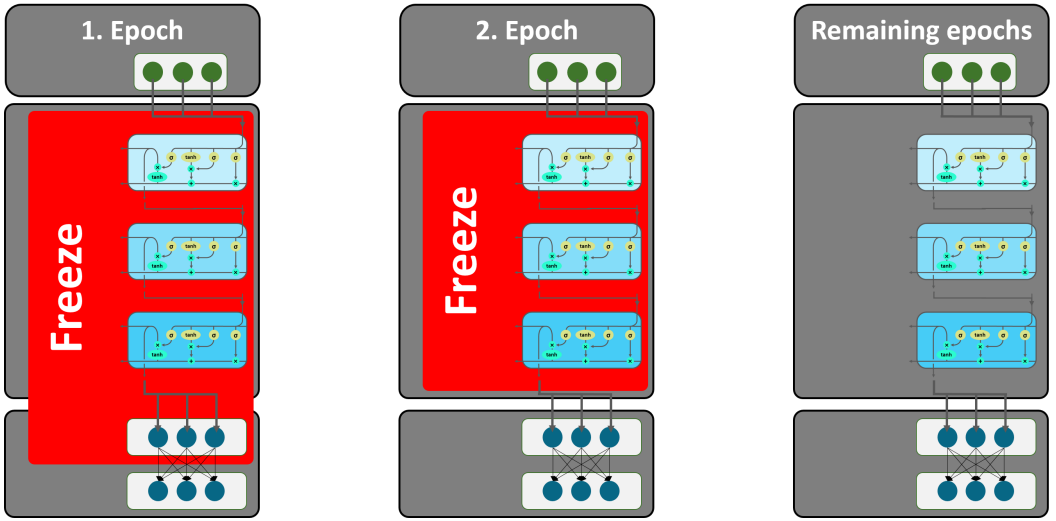



Even in the "natural" case, things can be easily forgotten.

Combine this with the possibility, that the **early gradient updates cause much noise** since the last classification layers can be newly initialized for the task and you will get pretty weak benefit from transfer!

# One solution: Gradual unfreezing

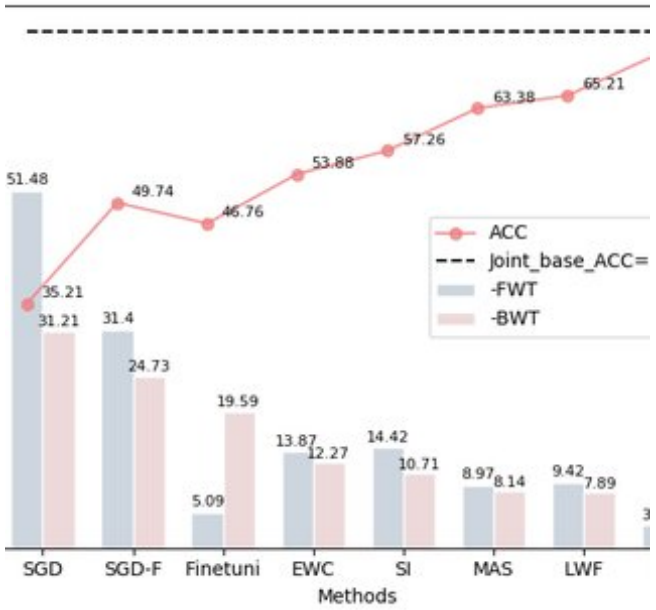
The idea is pretty simple: for the early part of the transfer learning based training do not allow updates on weights for the majority of the network, just **gradually unfreeze** the layers later on.




See a detailed analysis of Ruder and co.'s UMLFIT here (https://medium.com/explorations-in-language-and-learning/transfer-learning-in-nlp-2d09c3dfaeb6).

## Multiple other approaches: Verdict is still out

The development of more effective transfer learning methods is far from finished, there are quite recent papers (https://arxiv.org/abs/1812.01640) which offer a good survey:



Recently a more **systematic analysis of transfer learning with BERT model** has been carried out in context of sentiment classification.

Multiple approaches are being studied:

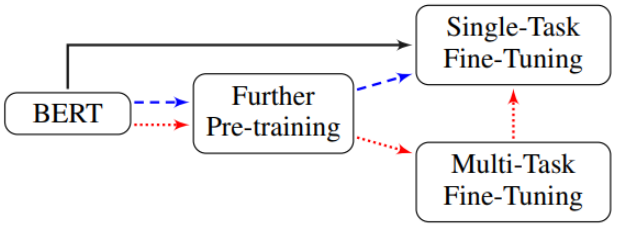



Figure 1: Three general ways for fine-tuning BERT, shown with different colors.

# It became critical

The widespread success of Deep Learning is more and more based on the fact, that:

1. *Huge* pre-trained models are available
2. Integration work became more easy, methods standardizing.

In fact, a whole "cottage industry" has spawn for making the adaptation work easy.

Some examples / infrastructure:

- Transfer learning with TensorfLow Hub (https://www.tensorflow.org/tutorials/images/transfer_learning_with_hub)
- BERT Text Classification in 3 Lines of Code Using Keras (https://towardsdatascience.com/bert-text-classification-in-3-lines-of-code-using-keras-264db7e7a358)
- Pre-trained models for speech recognition (https://www.kaggle.com/c/tensorflow-speech-recognition-challenge/discussion/43576)

...

# Conclusion:

**No. Typically we won't start from scratch!**

"Oh, only 150 emails? No problem! I import pre-trained BERT in 3 lines, and we are good to go!" :-)