Some resources for NLP (given by Annalina)

- <https://github.com/sebastianruder/NLP-progress?utm_source=pocket_mylist> (état de l’art de nombreuses méthodes de NLP qui existent)

- <https://lena-voita.github.io/nlp_course.html?utm_source=pocket_mylist> (Cours de NLP)

- <https://spacy.io/> (librairie python pour le NLP)

- <https://www.nltk.org/> (librairie python pour le NLP)

- <https://radimrehurek.com/gensim/> (librairie python pour le NLP)

- <https://learning.oreilly.com/home/> (books online) -> <https://learning.oreilly.com/library/view/natural-language-processing>

Cours NLP: notes

**Word embeddings** are a type of word representation that allows words with similar meaning to have a similar representation. They are a distributed representation for text that is perhaps one of the key breakthroughs for the impressive performance of deep learning methods on challenging natural language processing problems.

Word embeddings are in fact a class of techniques where individual words are represented as real-valued vectors in a predefined vector space. The distributed representation is learned based on the usage of words. This allows words that are used in similar ways to result in having similar representations, naturally capturing their meaning.

It is common for researchers to make pre-trained word embeddings available for free, often under a permissive license so that you can use them on your own academic or commercial projects.

For example, both word2vec and GloVe word embeddings are available for free download.

These can be used on your project instead of training your own embeddings from scratch.

You have two main options when it comes to using pre-trained embeddings:

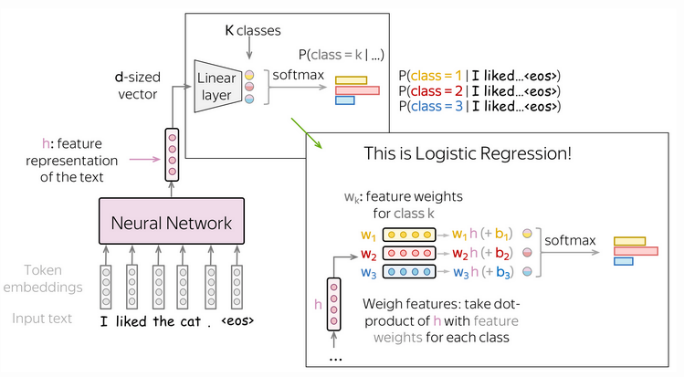
* Static, where the embedding is kept static and is used as a component of your model. This is a suitable approach if the embedding is a good fit for your problem and gives good results.
* Updated, where the pre-trained embedding is used to seed the model, but the embedding is updated jointly during the training of the model. This may be a good option if you are looking to get the most out of the model and embedding on your task.

<https://machinelearningmastery.com/what-are-word-embeddings/>

**Text Classification:** est-ce que j’aurai besoin de true labels ?

*Questions* : comment représenter le text : extract features from it ? puis une fois que les features sont extraites, on peut y appliquer un algo dessus. Les features formées à partir du texte peuvent dépendre de l’algo qu’on va appliquer ensuite.

On peut extraire les features manuellement et utiliser les algos : bayes classifieur, linear regression ou SVM ou alors utiliser les réseaux de neurones qui extraient les features tous seuls. Ce sont tous des méthodes supervisées.

The main idea of neural-network-based classification is that feature representation of the input text can be obtained using a neural network. In this setting, we feed the **embeddings** of the input tokens to a neural network, and this neural network gives us a vector representation of the input text. After that, this vector is used for classification.

*Image : qu’est-ce qu’il y a dans la boîte noire « neural network » ? Voir* **(\*)**

Neural classifiers are trained to predict probability distributions over classes. Intuitively, at each step we maximize the probability a model assigns to the correct class. Thus neural network are supervised learning so we need the true labels.

**(\*)**There is different ways to get a vector representation of an input text using neural networks :

- Bag of Embeddings (BOE) and Weighted BOE

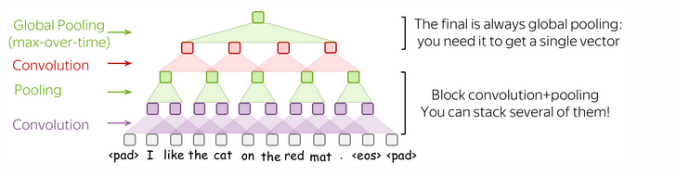
- Recurrent (RNN/LSTM/etc)

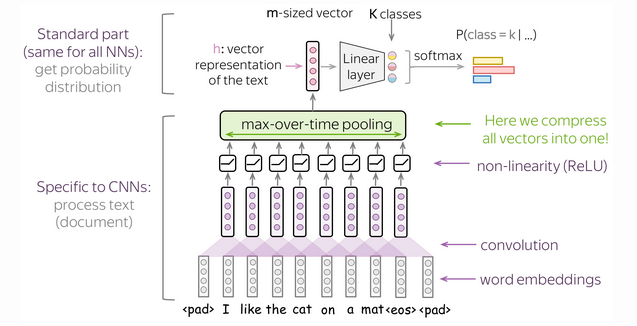
- Convolutional (CNN) : Convolution+Pooling Blocks

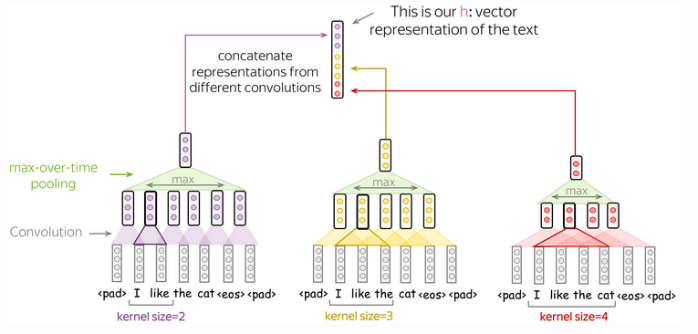
convolution: finds matches with patterns (as the cat head we saw above)

pooling: aggregates these matches over positions (either locally or globally)

To get a vector representation of an input text, a convolutional layer is applied to word embedding, which is followed by a non-linearity (usually ReLU) and a pooling operation.

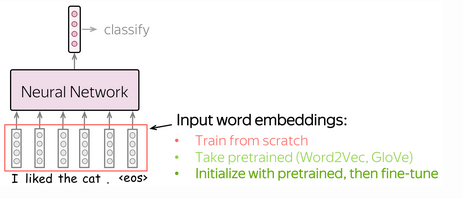
Intuitively, each filter in a convolution extracts a feature. One filter extracts a single feature. Usually, we want many features: for this, we have to take several filters. After a convolution extracted m features from each window, a pooling layer summarises the features in some region. Pooling layers are used to reduce the input dimension, and, therefore, to reduce the number of parameters used by the network.



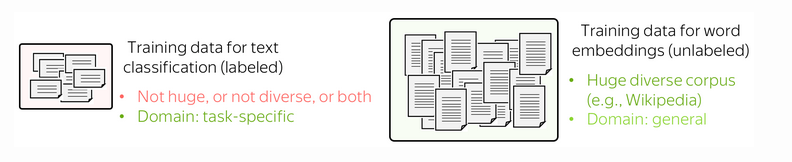


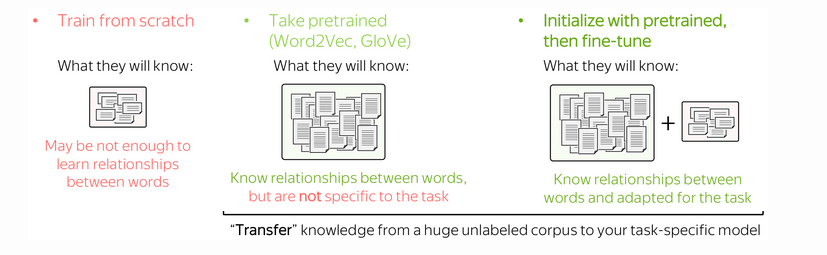
***Rq***: “Code from scratch" in the context of software development means writing the software from the beginning i.e. without using any code from any pre-existing software or any pre-existing components

**Word Embeddings: how to deal with them?**

Input for a network is represented by word embeddings. You have three options how to get these embeddings for your model:

* train from scratch as part of your model,
* take pretrained (Word2Vec, GloVe, etc) and fix them (use them as static vectors)
* initialize with pretrained embeddings and train them with the network ("fine-tune")

Training data for classification is labeled and task-specific, but labeled data is usually hard to get. Therefore, this corpus is likely to be not huge (at the very least), or not diverse, or both. On the contrary, training data for word embeddings is not labeled - plain texts are enough. Therefore, these datasets can be huge and diverse - a lot to learn from.

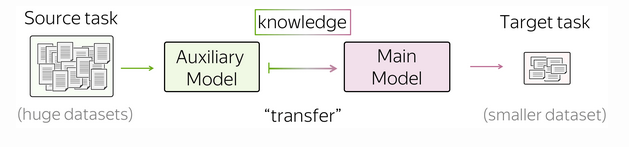
Now let us think what a model will know depending on what we do with the embeddings. If the embeddings are trained from scratch, the model will "know" only the classification data - this may not be enough to learn relationships between words well. But if we use pretrained embeddings, they (and, therefore, the whole model) will know a huge corpus - they will learn a lot about the world. To adapt these embeddings to your task-specific data, you can fine-tune these embeddings by training them with the whole network - this can bring gains in the performance (not huge though).

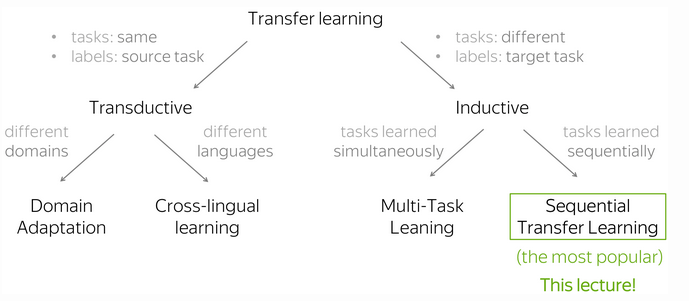
Il semblerait que la dernière image soit la meilleure méthode à adopter. Dernière image = **transfer learning**.

**Data augmentation** alters your dataset in different ways to get alternative versions of the same training example. Data augmentation can increase the amount of data (improve the quality of the model) diversity of data (model more robust).

**Transfer Learning** (ELMo, BERT)

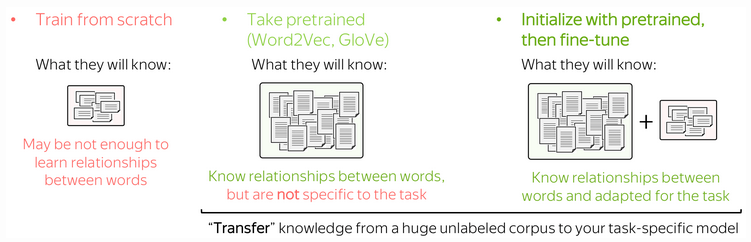
-> "transfer" knowledge from one task/model to another. For example, you don't have a huge amount of data for the task you are interested in (e.g., classification), and it is hard to get a good model using only this data. Instead, you can have data for some other task, which is easier to get.





***The Simplest Transfer: Word Embeddings*:** When talking about Text Classification, we already discussed that using pretrained word embeddings can help a lot.

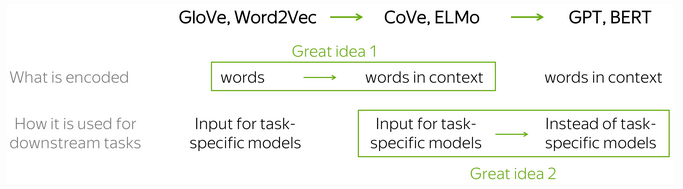
When we use pretrained embeddings, this is an example of transfer learning: through the embeddings, we "transfer" the knowledge of their training data to our task-specific model.



The idea of knowledge transfer we formulated for embeddings is general and stays the same when coming from word embeddings to pretrained models. As we will see a bit later, this won't always be the case.

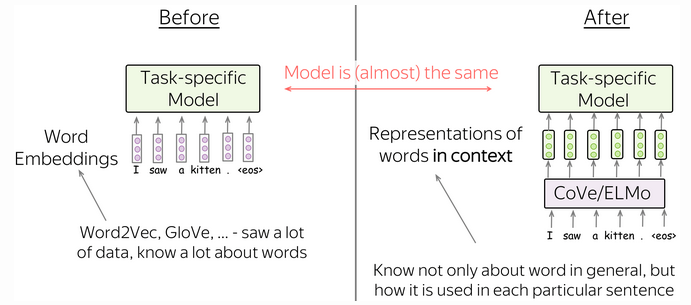
***The Two Great Ideas :***

In this part, we are going to see 4 models: CoVe, ELMo, GPT, BERT. Note that now there are lots of variations of these models: ranging from very small modifications (e.g., training data and/or setting) to rather prominent ones (e.g., different training objectives). However, roughly speaking, the transition from word embeddings to the current state-of-the-art models can be explained with just two ideas.



The two great ideas:

**Idea 1:** what is encoded: from words to words-in-context (the transition from Word2Vec/GloVe/etc. to Cove/ELMo): Instead of representing individual words, we can learn to represent words along with the context they are used in.



**Idea 2:** usage for downstream tasks: from replacing only word embeddings in task-specific models to replacing entire task-specific models (the transition from Cove/ELMo to GPT/BERT). CoVe/ELMo replace word embeddings, but GPT/BERT replace **entire models**.

* *Before: Specific model architecture for each downstream task :* Note that ELMo/CoVe representations were mainly used to replace the embedding layer, and kept task-specific model architectures almost intact. This means e.g. that for coreference resolution, one had to use a specific model designed for this task, for part-of-speech tagging - some other model, for question answering - another, very peculiar model, etc. For each of these tasks, researchers specializing in it kept improving the task-specific model architectures.
* *After: Single model which can be fine-tuned for all tasks : I*n contrast to previous models, GPT/BERT act not as a replacement for word embeddings, but as a replacement for task-specific models. In this new setting, a model is first pretrained using a huge amount of unlabeled data (plain texts). Then, this model is fine-tuned on each of the downstream tasks. What is important, now during fine-tuning you have to only use task-aware input transformations (i.e. feed the data in a certain way) instead of modifying model architecture.

**GPT:** In the fine-tuning stage, the model architecture stays the same except for the final linear layer. What changes is the input format for each task.

* Single sentence classification : To classify individual sentences, just feed the data as in training and predict the label from the final representation of the last input token.
  + Examples of tasks: SST-2 - binary sentiment classification (the one we saw in the Text Classification lecture); CoLA (Corpus of Linguistic Acceptability) - say whether a sentence is linguistically acceptable.
* Sentence pairs classification : To classify sentence pairs, feed the two fragments with a special token-separator (e.g. delim). Then, predict the label from the final representation of the last input token.
  + Examples of tasks: SNLI - entailment classification. Given a pair of sentences, say if the second is an entailment, contradiction or neutral); QQP (Quora Question Pairs) - given two questions say if they are semantically equivalent; STS-B - given two sentences return a similarity score from 1 to 5.

**BERT**: (state-of-the art)

BERT's model architecture is very simple and you already know how it works: it's just the Transformer's encoder. What is new, is the training objectives and the way BERT is used for downstream tasks. It uses a bidirectional encoder.

**Transformer’s encoder:** see lesson **seq2seq models**.

La j’avoue j’ai un peu décroché…

**“Fine-tuning**" means that you take a pretrained model and train in for the task you are interested in (e.g., sentiment classification) with a rather small learning rate.

Other NLP resources

* Sentiment analysis in Java : <https://blogs.oracle.com/javamagazine/post/java-sentiment-analysis-stanford-corenlp> and <https://blogs.oracle.com/javamagazine/post/java-sentiment-analysis-multisentence-text-block> (\*)

**Textblob**

* Leçon simple (2h) sur NLP (les étapes pour aborder un sujet. Très bien fait). <https://www.youtube.com/watch?v=xvqsFTUsOmc> (2) Parle de data cleaning + mise en forme des data + data visualisation + sentiment analysis + topic modelling
* Sentiment analysis with textblob: <https://www.youtube.com/watch?v=ujId4ipkBio>

**BERT**

* Sentiment analysis with BERT (YT): <https://www.youtube.com/watch?v=szczpgOEdXs>

(use already petrained bert) ou <https://www.youtube.com/watch?v=Osj0Z6rwJB4> + <https://www.youtube.com/watch?v=8N-nM3QW7O0> (use basic bert and construct your own sentiment classifier)

* Unlabelled training with bert: <https://nlpiation.medium.com/is-it-possible-to-do-sentiment-analysis-on-unlabeled-data-using-bert-feat-vader-experiment-357bba53768c>
* Fonctionnement de bert (YT): <https://www.youtube.com/watch?v=xI0HHN5XKDo>
* Fonctionnement transformers (un peu compliqué) (YT) : <https://www.youtube.com/watch?v=TQQlZhbC5ps>
* BERT tuto (webpage) <https://towardsdatascience.com/part-1-data-cleaning-does-bert-need-clean-data-6a50c9c6e9fd>
* BERT pre-processing (webpage) : <https://analyticsindiamag.com/a-guide-to-text-preprocessing-using-bert/>
* RoBERTa (YT): <https://www.youtube.com/watch?v=QpzMWQvxXWk>
* BERT with long texts (LSTM) <https://medium.com/@armandj.olivares/using-bert-for-classifying-documents-with-long-texts-5c3e7b04573d> + <https://stackoverflow.com/questions/58636587/how-to-use-bert-for-long-text-classification>
* **Fined-tuned BERT**:
* <https://skimai.com/fine-tuning-bert-for-sentiment-analysis/>
* <http://mccormickml.com/2019/07/22/BERT-fine-tuning/#2-loading-cola-dataset>
* <https://medium.com/nerd-for-tech/fine-tuning-pretrained-bert-for-sentiment-classification-using-transformers-in-python-931ed142e37>

**Vader:**

* <https://www.youtube.com/watch?v=QpzMWQvxXWk>

**Text visualization/cleaning**

* **Gensim** : biblio python pour tokenization
* Textual data visualization: (YT) <https://www.youtube.com/watch?v=HVBk2Ge_Q98>
* Data analysis text: (YT) <https://www.youtube.com/watch?v=dmRb_CDDH-s>
* Clean text (YT) easy recap: <https://www.youtube.com/watch?v=pvUiCMCLgBA> + <https://www.youtube.com/watch?v=KhXU7KOxQcg>

**Librairies**:

* Differences between NLTK and SPACY library for NLP (YT): <https://www.youtube.com/watch?v=h2kBNEShsiE>
* Spacy for NLP (YT): <https://www.youtube.com/watch?v=dIUTsFT2MeQ>

**Text features extraction :**

* <https://www.analyticsvidhya.com/blog/2021/04/a-guide-to-feature-engineering-in-nlp/>
* <https://medium.com/swlh/nlp-all-them-features-every-feature-that-can-be-extracted-from-text-7032c0c87dee>
* <https://www.geeksforgeeks.org/feature-extraction-techniques-nlp/>
* YT : (bien fait il me semble) <https://www.youtube.com/watch?v=7YacOe4XwhY>

**Datasets : (3 or more labels)**

* <https://www.kaggle.com/datasets/mathurinache/reddit-machine-learning-posts> (pb pas de labels pour la polarité)
* ~~AndrewNg Machine Learning Tweets :~~ [~~https://www.kaggle.com/datasets/gauravsahani/andrewng-machine-learning-tweets~~](https://www.kaggle.com/datasets/gauravsahani/andrewng-machine-learning-tweets)
* Amazon Kindle Book Review for Sentiment Analysis: OK <https://www.kaggle.com/datasets/meetnagadia/amazon-kindle-book-review-for-sentiment-analysis>
* Product sentiment analysis: <https://www.kaggle.com/datasets/arbazkhan971/product-sentiment-analysis/code>
* Twitter and Reddit Sentimental analysis Dataset: <https://www.kaggle.com/datasets/cosmos98/twitter-and-reddit-sentimental-analysis-dataset>
* Sentiment Analysis for Financial News (only head of the papers): <https://www.kaggle.com/datasets/ankurzing/sentiment-analysis-for-financial-news>
* SemEval-2014 Task 4 (laptops & restaurants reviews) use only laptops?: <https://alt.qcri.org/semeval2014/task4/>
* Yelp dataset: <https://www.yelp.com/dataset> ce que ça contient: <https://www.kaggle.com/code/ambarish/a-very-extensive-data-analysis-of-yelp/report> (je crois que c’est des revues sur des restaurants)

**Datasets : (2 labels)**

- Amazon Review Full Score Dataset (**score from 1 to 5**) already downloaded

Can find dataset here: <https://drive.google.com/drive/folders/0Bz8a_Dbh9Qhbfll6bVpmNUtUcFdjYmF2SEpmZUZUcVNiMUw1TWN6RDV3a0JHT3kxLVhVR2M?resourcekey=0-TLwzfR2O-D2aPitmn5o9VQ>

Name of the file to download: amazon\_review\_full\_csv.tar.gz

- Amazon Kindle Book Review for Sentiment Analysis (**score from 1 to 5**) already downloaded

Can find dataset here: <https://www.kaggle.com/datasets/meetnagadia/amazon-kindle-book-review-for-sentiment-analysis>

- IMDB dataset (Sentiment analysis) in CSV format (**score 0,1**)

Can fond dataset here: <https://www.kaggle.com/datasets/columbine/imdb-dataset-sentiment-analysis-in-csv-format>

Remarques textes:

* Il faudra sans doute supprimer les sauts de lignes
* “Il y a parfois des dates dans le texte + le commentaire dessous et tout ceci est placé dans une seule ligne. Faut-il découper ce texte selon la date ? et ajouter des lignes pour chaque date ou est-ce un cas marginal ?
* Comme le dit le document (\*) pour attribuer un poids à chaque phrase, il faudra décider si le commentaire (+ aussi penser aux délivrables) est construit comme une histoire ou une review.

Même si pas mal de description comme les histoires, je dirai review pour être sûr de prendre en compte la 1ere phrase aussi où il y a parfois des appréciations importantes.

* I am not sure if the meeting notes will have sentiment change within the same meeting, if you noticed that sentiment changes within the same meeting....you can do sentence segmentation first before sentiment analysis (peut être c’est plus pertinent)
* Textblob: très simple à utiliser mais ne fait pas une analyse très poussée (donc attention aux résultats). Donne pour chaque texte un score de sentiment : positive ou négative et un score de subjectivité : subjective ou facts.
* Sentiment analysis over time (see video (2)). Mais sinon, juste prendre le sentiment de chaque meeting (la date du meeting contient déjà le temps).
* Supervised or unsupervised method ?
* J’ai lu tous vos notebooks sur le NLP et je ne suis pas sure de comprendre à quoi servent les techniques du dossier NLP2 pour l’extraction de features. Est-ce que ça va servir dans mon contexte ?

Remarques BERT :

* Bert and many other pre-trained language models use the Wordpiece algorithm so stemming is not necessary. Preprocessing is not needed when using pre-trained language representation models like BERT. In particular, it uses all of the information in a sentence, even punctuation and stop-words, from a wide range of perspectives by leveraging a multi-head self attention - mechanism.
* Au lieu de s’embêter à préprocesser le texte avec python, on peut directement utiliser un algo de BERT-preprocessing adapté à notre cas : sentiment analysis.

=> saw how easily the word embedding can be implemented using BERT pre-processing modules. All the traditional pre-processing steps are included in the BERT pre-processing modules which saves a lot of time while building the NLP-based model.

Sentiment analysis to try:

* BERT without heavy cleaning (apparently better then Bert with heavy cleaning)
* BERT with heavy cleaning
* word embedding can be easily implemented using BERT pre-processing modules.
* Textblob from nltk (naïve approach), needs heavy text cleaning
* VADER sentiment scoring (uses nltk’s *SentimentIntensityAnalyzer*): un peu comme textBlob : ne prend pas en compte les relations entre les mots (pas de contexte)
* RoBERTa pretareined model from huggingFace (import from transformers)