NLP 3 - Neural networks, Transformers

SICSS Paris 2023

Julien Boelaert

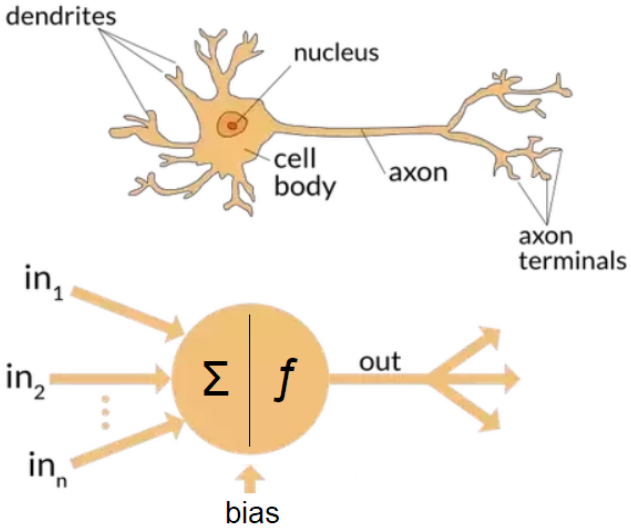
CERAPS, Université de Lille

27/6/2023

Outline

1.1/ Multilayer perceptrons

The basic building block of all NNs is the **artificial neuron**



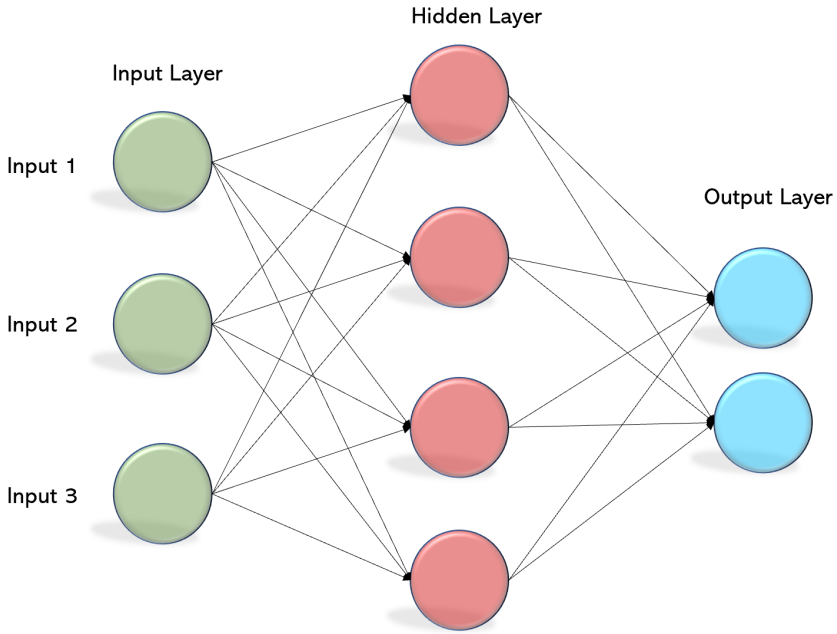
- loose biological inspiration
- transforms inputs to outputs, by
  1. **weighted sum** of the inputs
  2. and **activation function** f

  $\mathbf{out} = f\left(\alpha_0 + \sum_i \alpha_i \mathbf{in_i}\right)$
- With f = identity, linear regression (or perceptron, Rosenblatt 1958)
- With $f(x) = \frac{1}{1+exp(-x)}$, logistic regression

Fitting (*training*) a linear/logistic model: find the best weights α that minimize a cost function over the training data (MSE / cross-entropy).

1.1/ Multilayer perceptrons

A **multilayer perceptron** is a composition of such neurons (known since the 1960s, popularized 1985 by Rumelhart, Hinton and Williams with the **backpropagation** algorithm)



- each arrow (connection) has a weight
- layers: input, hidden, output
- information flows left to right: **feedforward** network

$$\mathbf{out_k} = g\left(\beta_{0k} + \sum_j \beta_{jk} f\left(\alpha_{0j} + \sum_i \alpha_{ij}\mathbf{in_i}\right)\right)$$

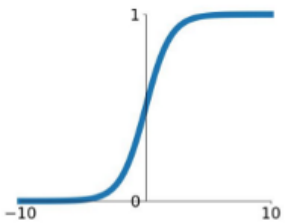
1.1/ Multilayer perceptrons

Activation functions for hidden layer:

- if f is linear, an MLP reduces to a linear model
- common choices of nonlinear activations:
  sigmoid or tanh (1980s), ReLU and variants (2000s)

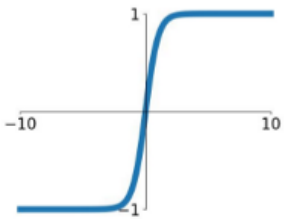
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



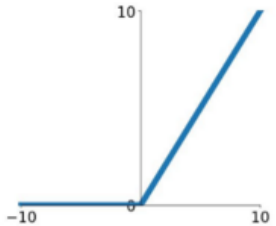
tanh

$$\tanh(x)$$



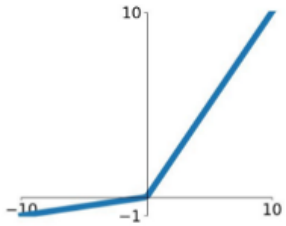
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

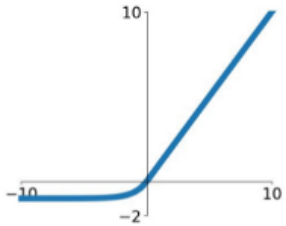


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



1.1/ Multilayer perceptrons

Common choices of **output layer activations**:

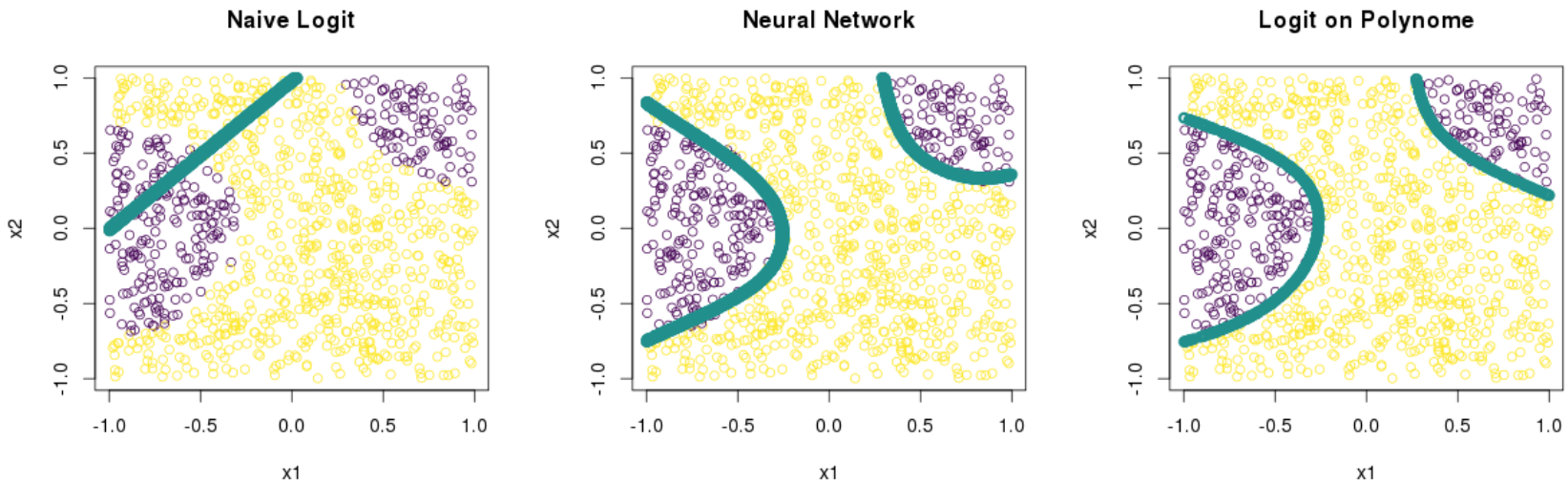
- regression: identity, or tanh (where outputs are first normalized to (-1,1))
- binary classification: sigmoid (as in logistic regression)
- multiple classification: softmax

$$softmax(out_k) = \frac{exp(out_k)}{\sum_l exp(out_l)}$$

1.1/ Multilayer perceptrons

Why use MLPs / ANNs?

- **universal approximation**: MLP theoretically capable of approximating *any* function, if enough neurons in hidden layer
- theoretically removes the need for hand-crafted **features** to feed the model: automatically constructed during training.

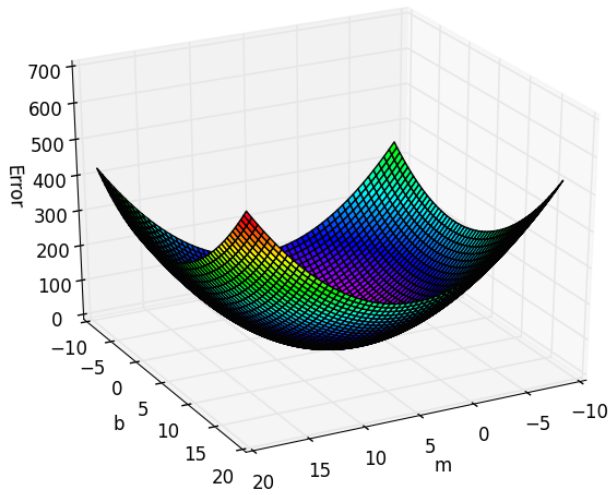


Example: binary classification on two predictors

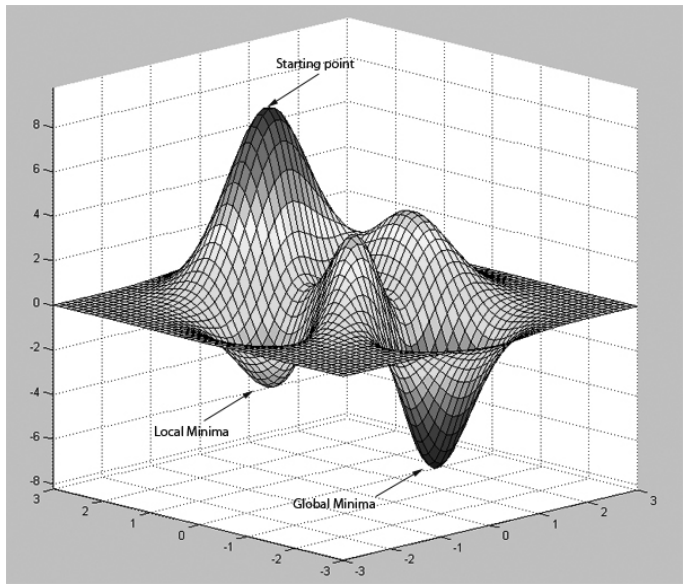
1.1/ Multilayer perceptrons

Training: adjusting the weights so that the predicted outputs correspond to the observed outputs

- in linear regression, exact closed-form solution
- in logistic regression, algorithmic solution through convex optimization (solution is existing and unique)
- in MLPs, the **error surface** is not convex $\rightarrow$ minimum can only be approached algorithmically, with no guarantee to find the global minimum



Convex error surface

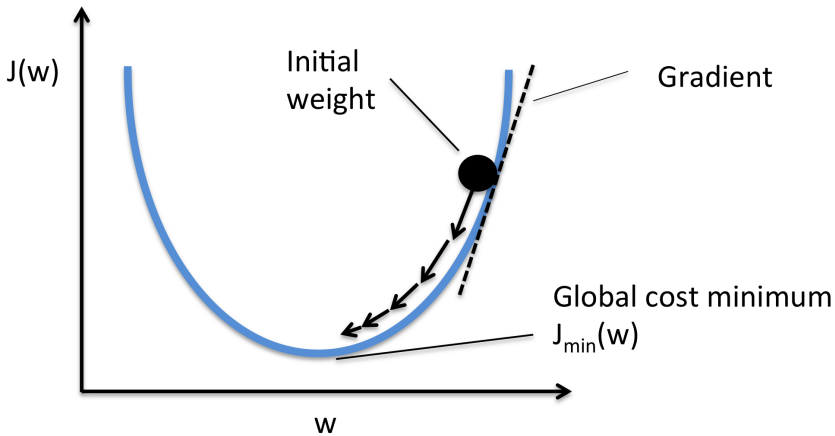


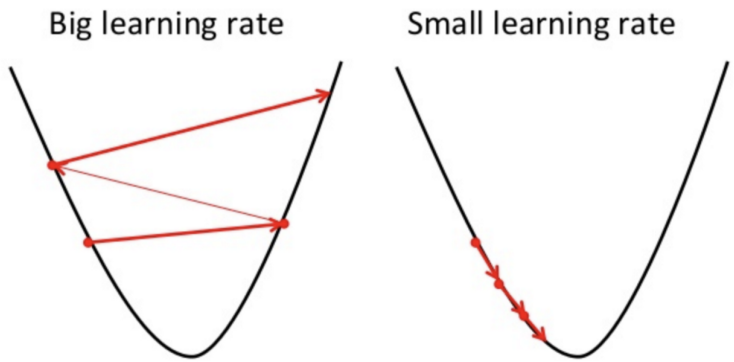
Nonconvex error surface

1.1/ Multilayer perceptrons

Training: **(stochastic) gradient descent**, aka **backpropagation**.

- weights are initialized randomly
- for each observation (several **epochs**, *ie* runs over the complete training set):
  1. compute the output error
  2. compute the gradient of the error wrt weights
  3. move the weights in the opposite direction of the gradient (with a small **step size** / **learning rate**)
- possible because error function differentiable wrt weights



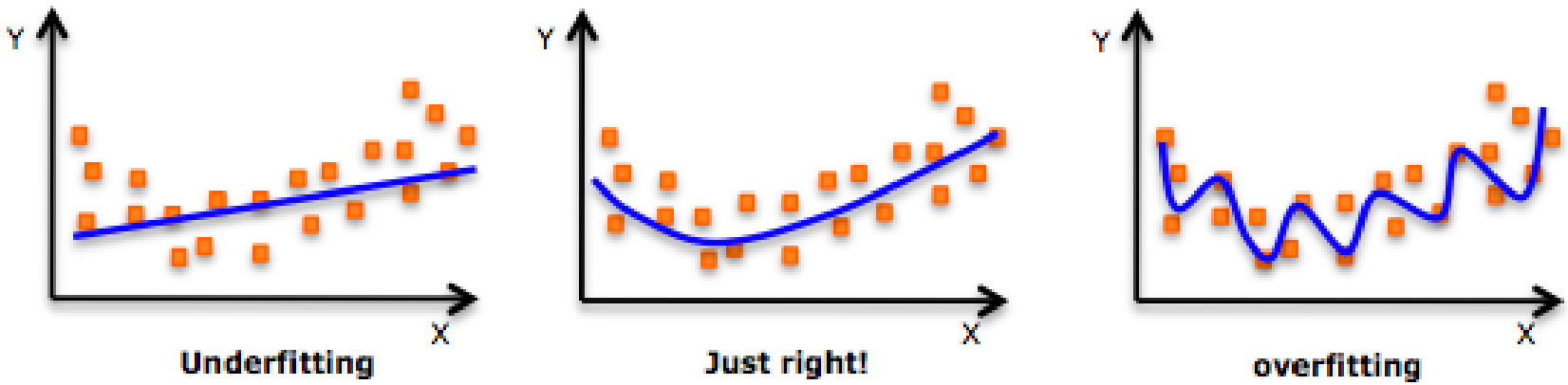


- today, the *de facto* standard is Adam (Kingma and Ba, 2015), adaptively adjusts the step size for each individual weight.

1.1/ Multilayer perceptrons

Because MLPs are so flexible, they are prone to **overfitting**

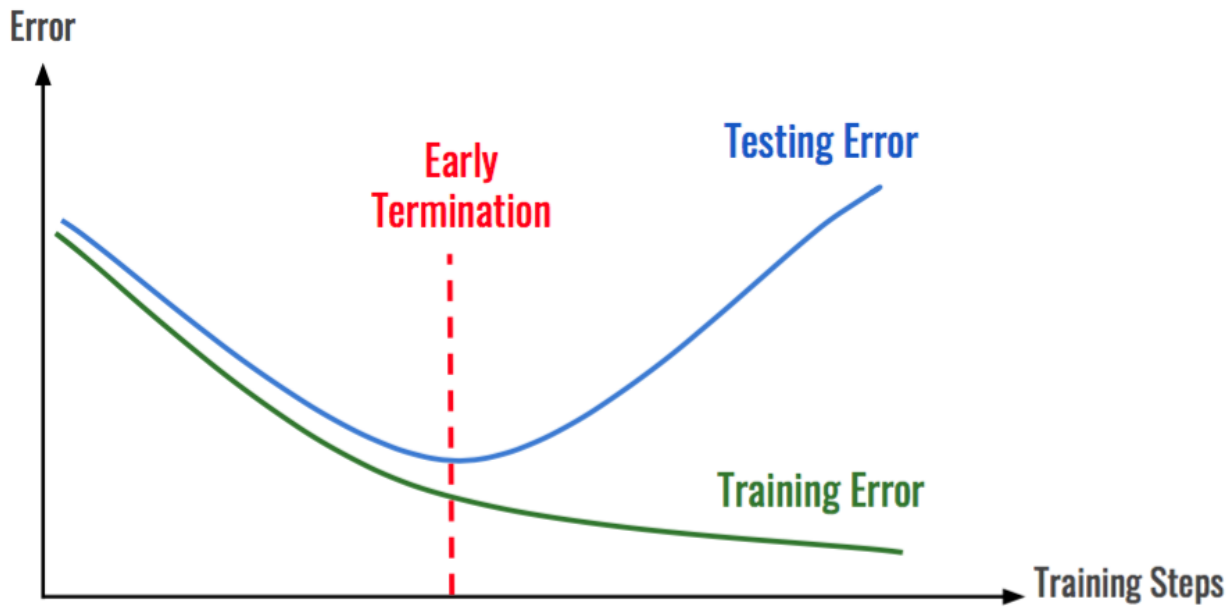
- overfitting = learning noise instead of only signal
- results in perfect fit on training data, but bad **generalization** (*ie* prediction on out-of-sample data)



1.1/ Multilayer perceptrons

To measure and control overfitting, dataset divided into 3 subsets

- **training** set: observations used for SGD
- **development** / validation set: not used for SGD, but for monitoring during training, and deciding when to stop
- **test** set: used *only* at the end, measure **generalization error**



1.1/ Multilayer perceptrons

To prevent overfitting, use **regularization** techniques:

- add a regularization term to the error function:
  $Err^*(\mathbf{w}) = Err(\mathbf{w}) + \sum_j |w_j|$ (L1-norm)
  $Err^*(\mathbf{w}) = Err(\mathbf{w}) + \sum_j w_j^2$ (L2-norm, "**weight decay**")
  $\rightarrow$ prevents weights from becoming too big, "simpler" model
- **dropout**: randomly drop units (along with their connections) from the neural network during training, preventing units from co-adapting too much.
  (N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R., 2014, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", *JMLR*, 15(56))
- many more: mini batch, gradient clipping, normalization layers such as BatchNorm (2015) or LayerNorm (2016), ...

In any case, though, neural nets are notoriously **difficult to train**!

1.1/ Multilayer perceptrons

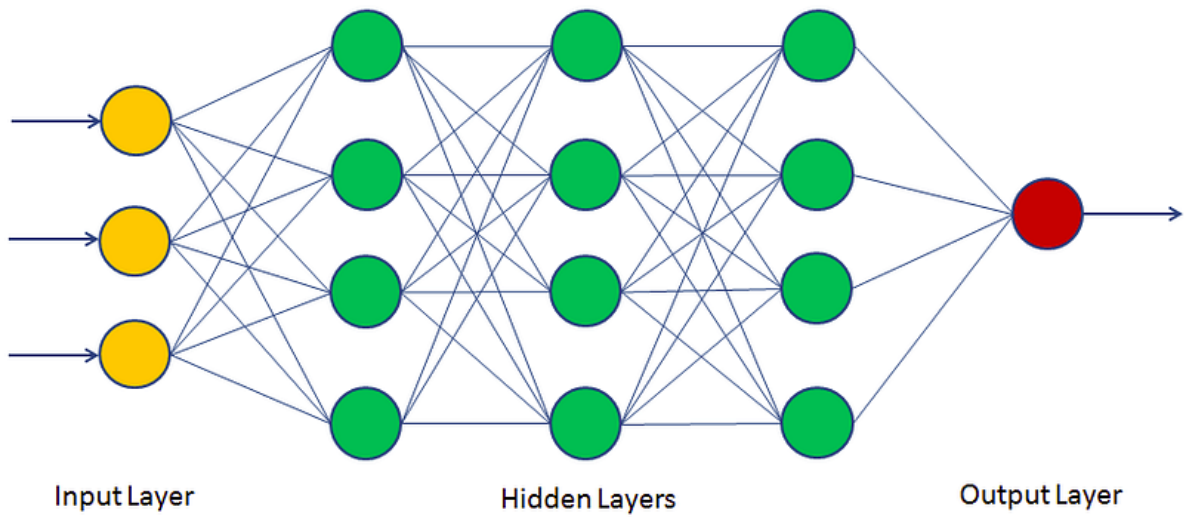
MLP for text analysis?

- using text as predictors in MLPs is possible but awkward: fixed text length, and each weight applies only to a certain position in the sequence
- Bengio et al. (2003) used an MLP for *language modeling* (predicting the next word, based on the k previous ones): starting point of word embeddings in the NN literature, and more generally of neural language modeling.

1.2/ Deep learning

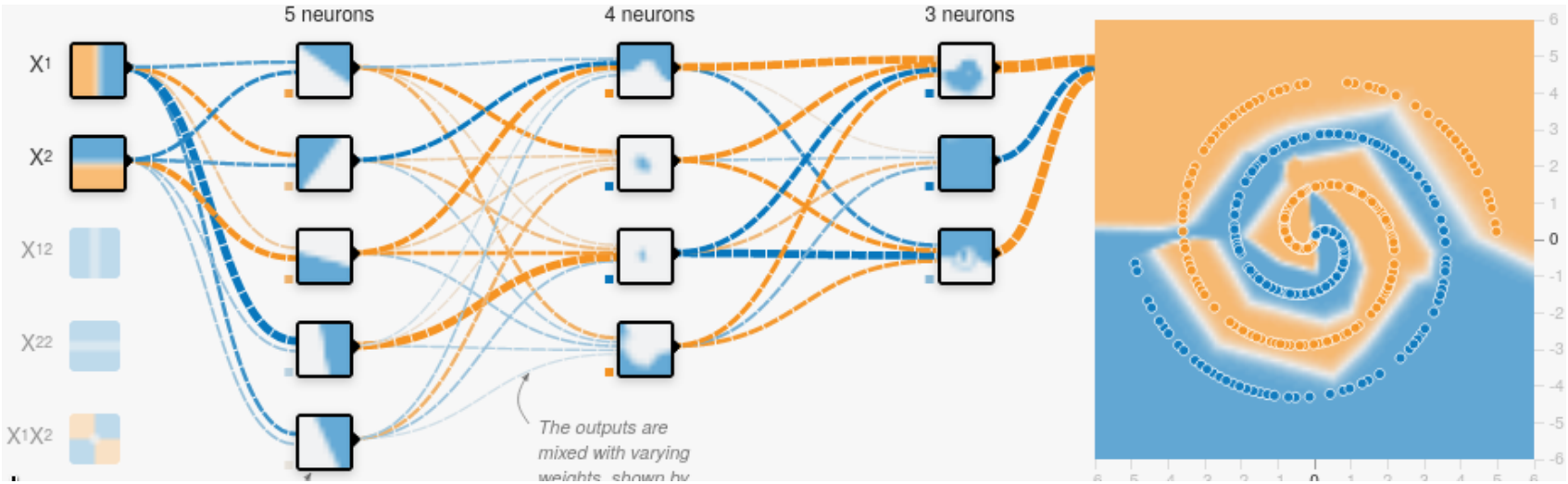
Deep learning started gaining traction in the late 2000s:

- more hidden layers
- bigger training datasets
- a few tricks: ReLU activation, better initialization schemes (*eg.* Xavier Glorot), dropout regularization
- diverse architectures: feedforward, convolutional, recurrent
- faster computation with GPUs (esp. convnets)



1.2/ Deep learning

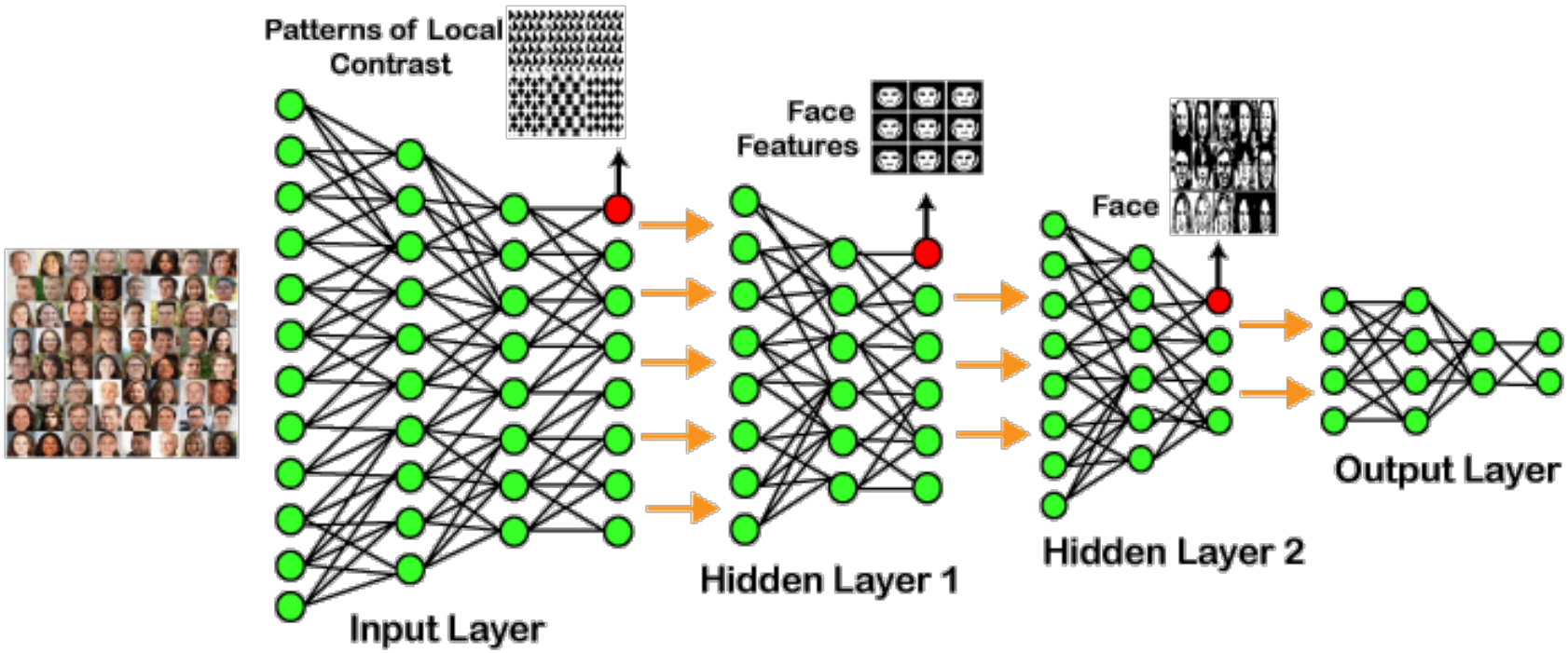
Main intuition: successive layers build **levels of abstraction**, by recombining the features of the previous layers



Try for yourself at `http://playground.tensorflow.org`

1.2/ Deep learning

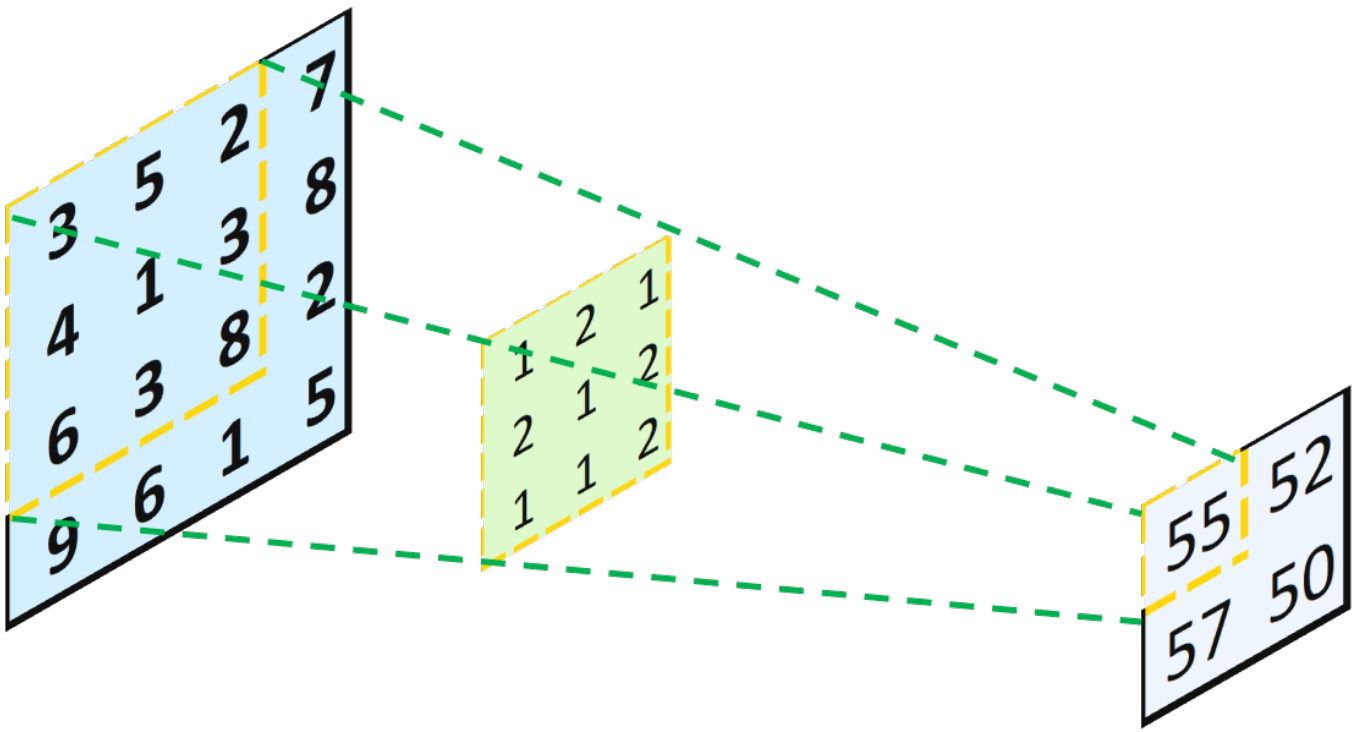
Example: facial features to faces



1.3/ Convolutional networks

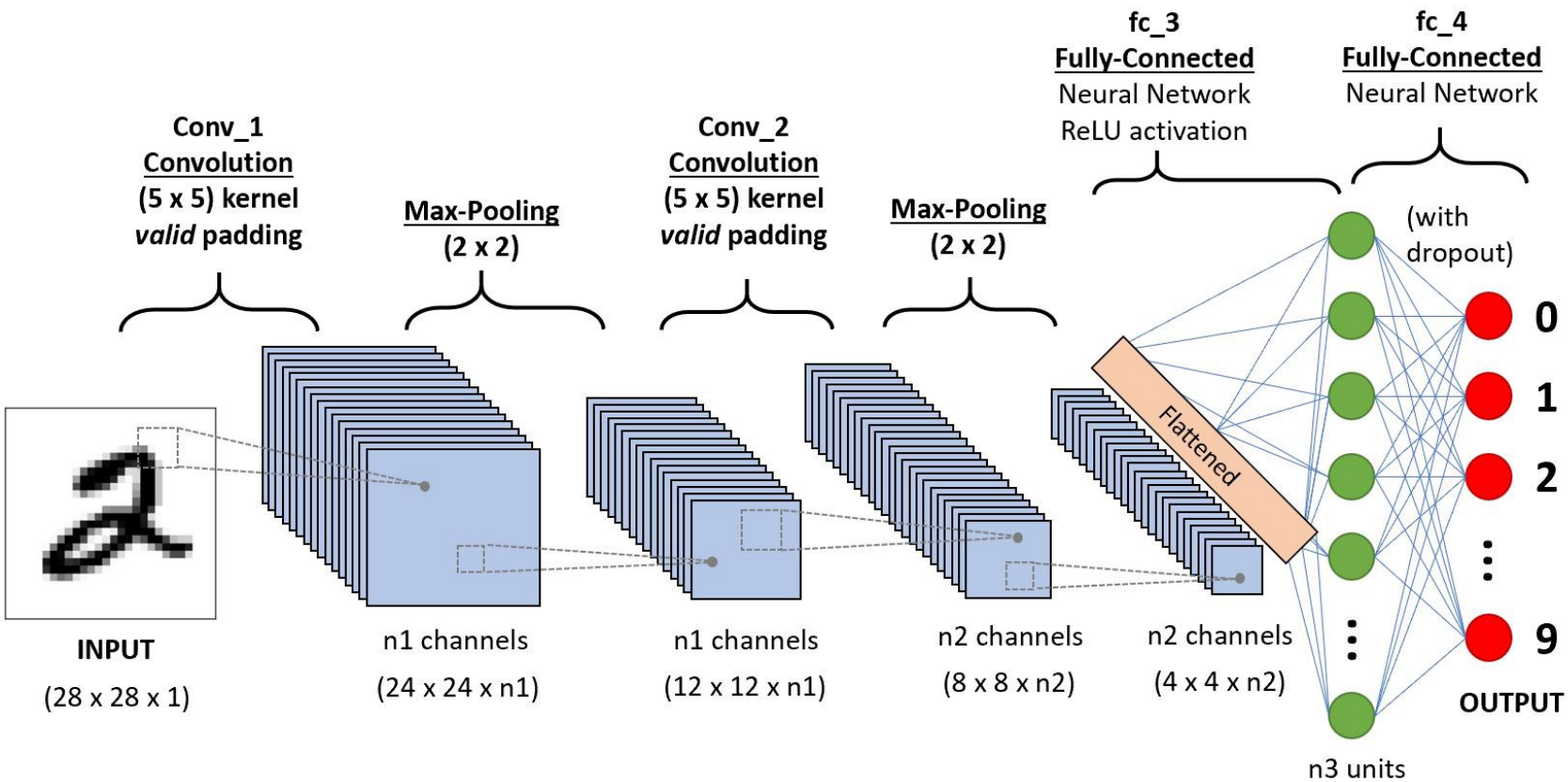
Convolutional neural networks (aka convnets, CNN) are a special architecture, well suited for structured data such as images or text. Invented in 1980s ("Neocognitron" by Fukushima 1980, CNNs by LeCun late 1980s), became huge in the 2000s.

Main building block: a convolution "filter" that slides over the input.



1.3/ Convolutional networks

Deep convnets typically alternate convolution and pooling layers, until a final few MLP layers



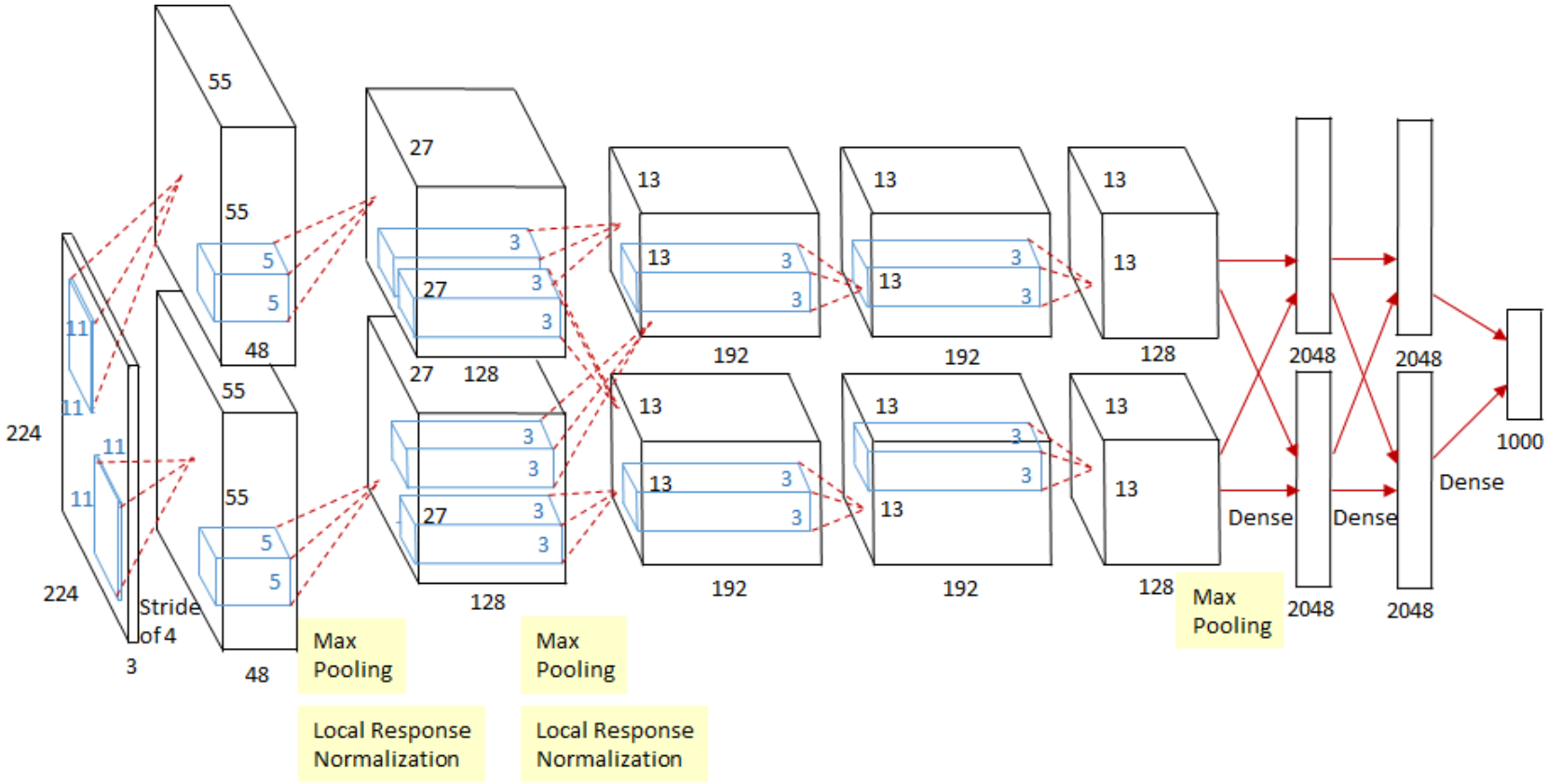
1.3/ Convolutional networks

Perks of convolutional networks:

- make explicit use of the structure (images 2D, text 1D)
- every filter is applied to every region of the input data
- deep convnets: higher layers have access to large regions
- efficient computation with GPUs
- trained end-to-end with backpropagation, no need for hand-crafted features.

1.3/ Convolutional networks

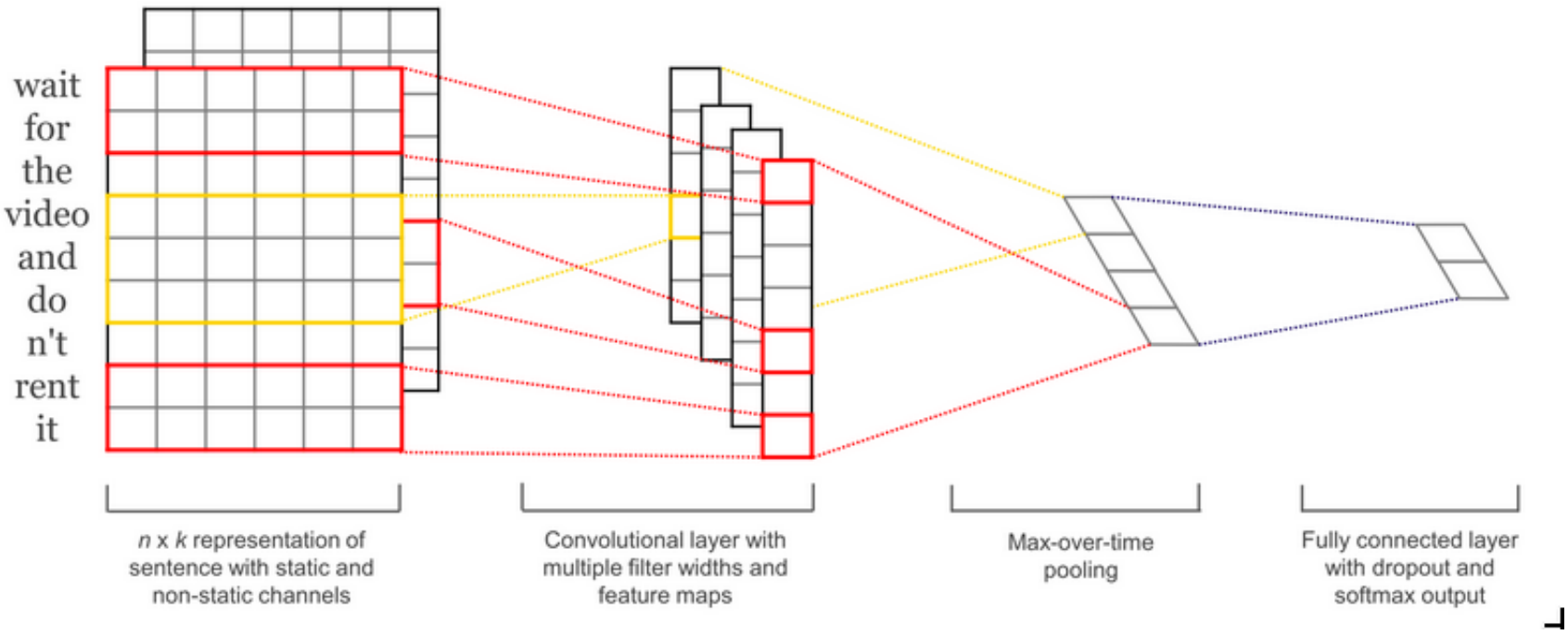
Example convnet: Alexnet (2012) achieved state-of-the-art image recognition, and started the deep learning hype.



A. Krizhevsky, I. Sutskever, G. Hinton, 2017, "ImageNet classification with deep convolutional neural networks", *Communications of the ACM*, 60 (6)

1.3/ Convolutional networks

Convnets for text analysis: 1D convolutions and pooling, possibly on pre-trained word embeddings.

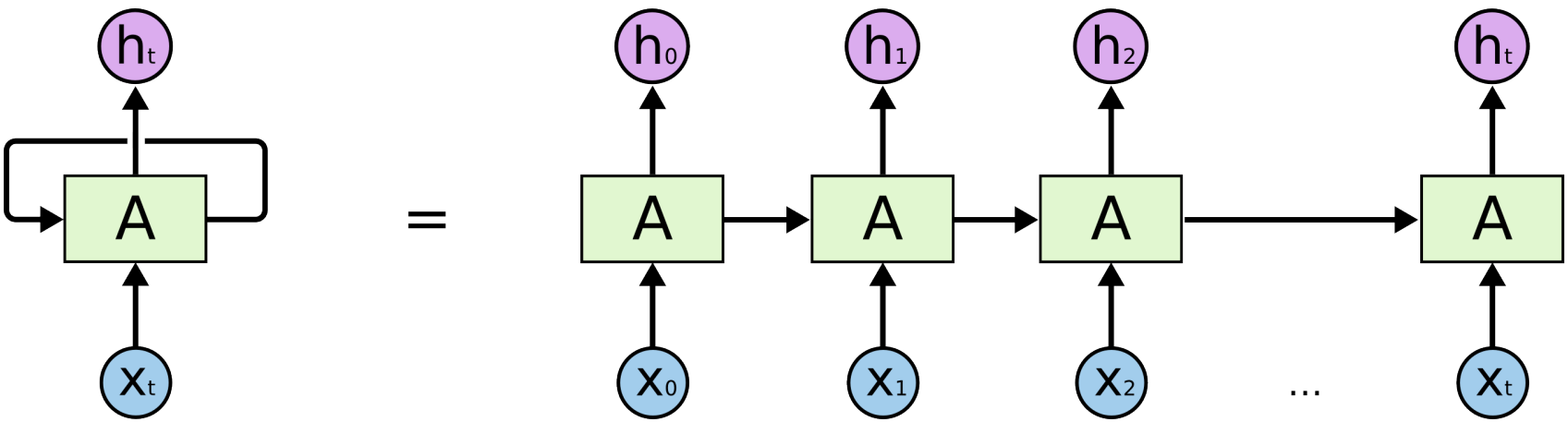


(Y. Kim, 2014, "Convolutional Neural Networks for Sentence Classification")

1.4/ Recurrent networks

Recurrent neural networks (RNN):

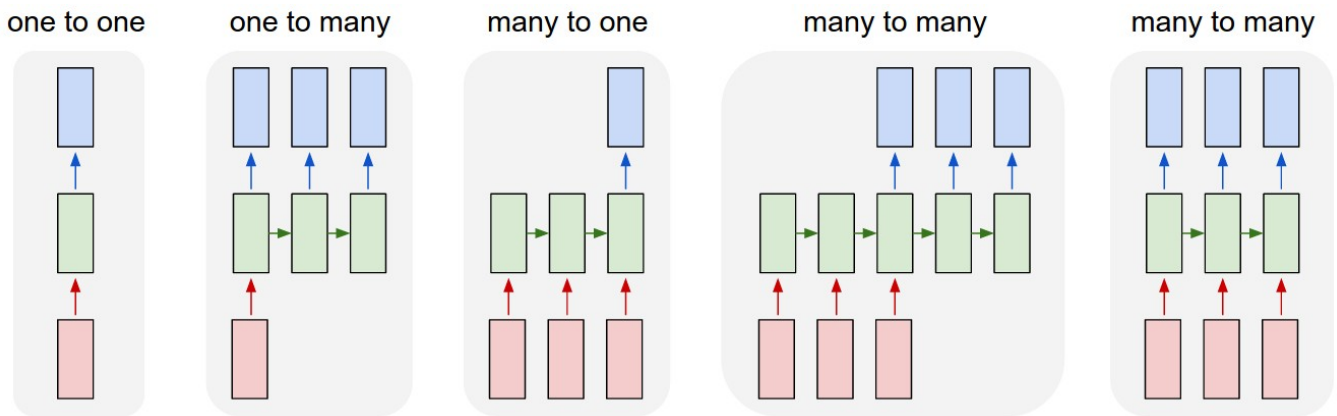
- a single neural network with **memory**
- processes the inputs as a sequence (one after the other)
- at each time step t, the network processes input t and its own previous state $(t-1)$
- trained by *backpropagation through time*



1.4/ Recurrent networks

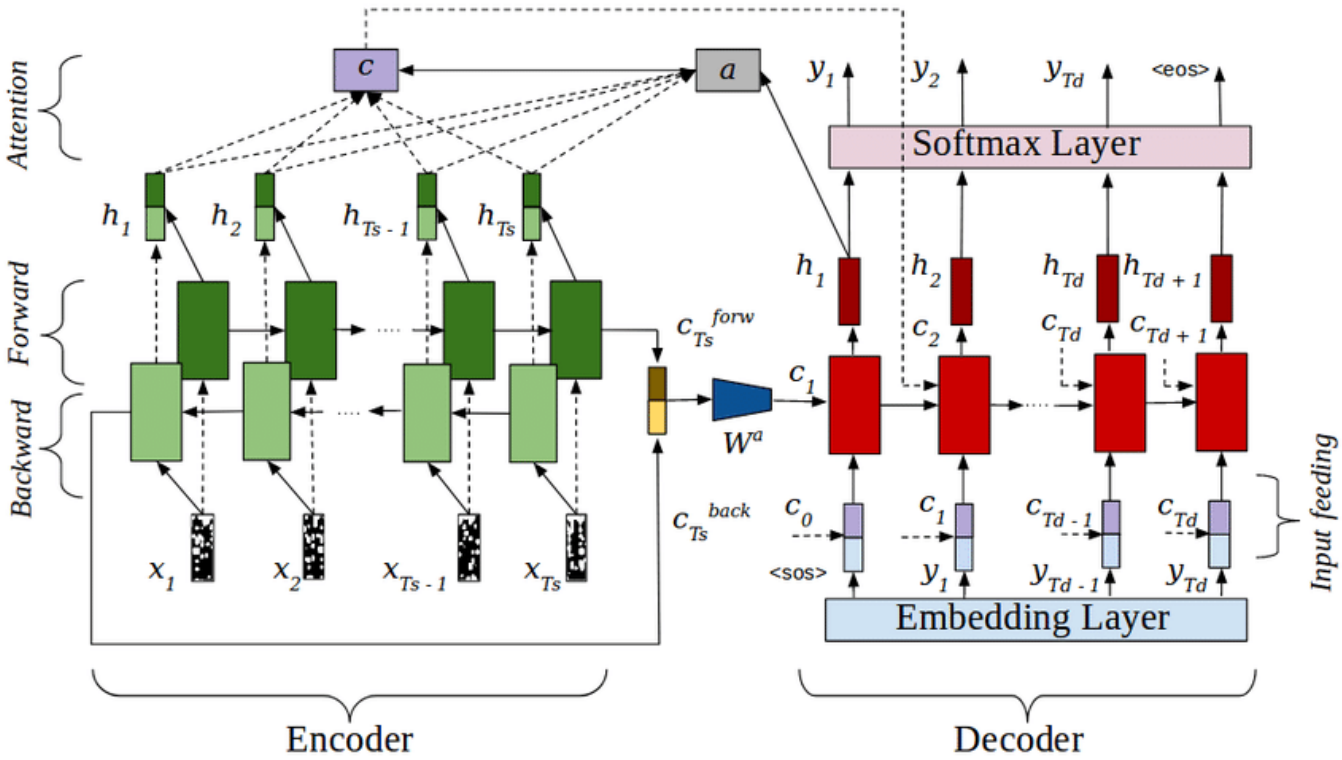
RNNs:

- **Turing-complete** (universal approximation over algorithms!)
- Even more difficult to train than feedforward neural nets
- Improvements: **LSTM** (Hochreiter and Schmidhuber 1997) and **GRU** (Cho *et al.* 2014) use forget-gates to "decide" when to "forget" some information of the past $\rightarrow$ easier to train



1.4/ Recurrent networks

RNNs with **attention** have direct access to all previous states, and learn to "decide" which ones are important for the current time step (in encoder-decoder models).



1.4/ Recurrent networks

RNNs for NLP?

- Google Translate's 2016 switch to neural translation: LSTMs
- Just as convnets, no longer BOW, and usable with pre-trained word embeddings
- Bidirectional RNN: process text start-to-end and end-to-start simultaneously
- Example: ELMo (Embeddings from Language Models, Peters *et al.* 2018) used bidirectional LSTMs to produce context-based word embeddings
- State of the art NLP before Transformers

1/ ANN - Concluding remarks

Deep learning, CNNs, RNNs :

- extremely potent (MLP $<$ CNN $<$ RNN $<$ RNN $+$ attention)
- hard to train! easy to get lost, hard to keep track
- much room for creativity
- user-friendly library (python and R): keras, with tensorflow or torch python backend
- in current NLP, mostly superseded by Transformer models

2/ Character-level language models

Language modeling:

- LM = predicting the probability of a sentence (*ie* is it likely to occur in the corpus the model was trained on?)
- mathematically equivalent to predicting the next word: $P(W_1 \bigcap W_2 \bigcap W_3) = P(W_1) \times P_{W_1}(W_2) \times P_{W_1 \bigcap W_2}(W_3)$
- central task in NLP: machine translation, summarization, speech recognition, grammar correction, dialogue, ...
- traditionally done with trigram models
- RNNs are a prime candidate for this task

2/ Character-level language models

Andrej Karpathy, 2015, "The Unreasonable Effectiveness of Recurrent Neural Networks"

`http://karpathy.github.io/2015/05/21/rnn-effectiveness/`

- LSTMs to predict the next character in a corpus
- Predict character: small dictionary, but hard task
- Generative model: once trained, it can produce new text

2/ Character-level language models

Example of generated text on complete Shakespeare corpus (4.4MB, 3-layer LSTM with 512 hidden nodes each):

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.
```

2/ Character-level language models

Continues with several examples: wikipedia (markdown), a book on algebraic geometry (producing LaTeX that almost compiles), Linux source code... all producing very random but somewhat credible text.

I tried it myself on the complete Johnny Hallyday corpus (around 1K songs, 1.1MB), on his death in 2017:

Je veux te voir aimer
Pardon
J'ai vu ce que je t'ai du les chansons
Que tout ce que je t'aime
Que tu sais que tu mens ce que je crois
J'ai la paix de mes mots de toi
Je suis né dans la nuit
Mais je n'ai pas défilé
Je n'ai pas de coeur dans le jour

Je suis mort de la vie
Mais j'ai tout comme un seul pour toi
Le coeur pour partir
Pour dire que je la prend pourtant la vie
Quand je vois que je vois
Et je sais au soleil
C'est plus fort
Pour moi ce **que je t'aime**

2/ Character-level language models

Also on a corpus of French *éditorialistes* (3.4 MB of Christophe Barbier, Bernard-Henri Lévy, Jacques Attali, Alain Duhamel...)

"Au contraire, le président de la démocratie s'applique de l'obsession de la guerre, c'est la construction de la sécurité tout pour autant répondre à la croissance de la réalité et de l'union et de la mondialisation et le commerce de la confiance de la désinvolture de sa terreur sur la mondialisation et au franc pour les solutions communes et la foi des finances de la compétitivité. Cette stratégie ne peut pas donner des riches et de relances d'autorité et de santé et de les autres. Le déficit de la crise est un aide. Le manque de confiance est en constat de la construction des réformes politiques qui ont considéré les déficits pour les confiances politiques et personnelles, des réformes de la construction de la relance des problèmes et des constructions des économistes et des plans de territoire, et celles des politiques politiques et de salariés et de l'avenir et le problème. Mais les confiances sont au pouvoir pour la promesse de la crise et de la recherche. Le changement de promesse sont devenues proposées de nouveaux libre-échanges et des banques sociales. Le régime de la société publique est profonde. Les communistes seront plus forts, à cette politique internationale. Le vide de la défiance est toujours plus autoritaire dans le terrorisme des confiances, et les trois natures de l'attention de l'Union européenne a

3/ Transformers

Transformers: one architecture to rule them all

1. Motivation
2. Transformer model, tokenizer
3. BERT: masked language model for NLP tasks
4. GPT: causal language model for text generation

3.1/ Transformers: motivation

Transformers are a neural network architecture that have been state of the art in all NLP tasks for the last 5 years.

- Original article (Google): Vaswani *et al*, 2017, "Attention is all you need", *Proceedings of the 31st Conference on Neural Information Processing Systems*
- Originally an encoder-decoder network (*eg* for machine translation)
- Encoder-only: BERT
- Decoder-only: GPT
- Also SOTA in image generation (Dall-E, Midjourney, Stable diffusion, ...), speech-to-text (Whisper), ...
- NLP benchmarks: GLUE and superGLUE `https://super.gluebenchmark.com/leaderboard`

3.1/ Transformers: motivation

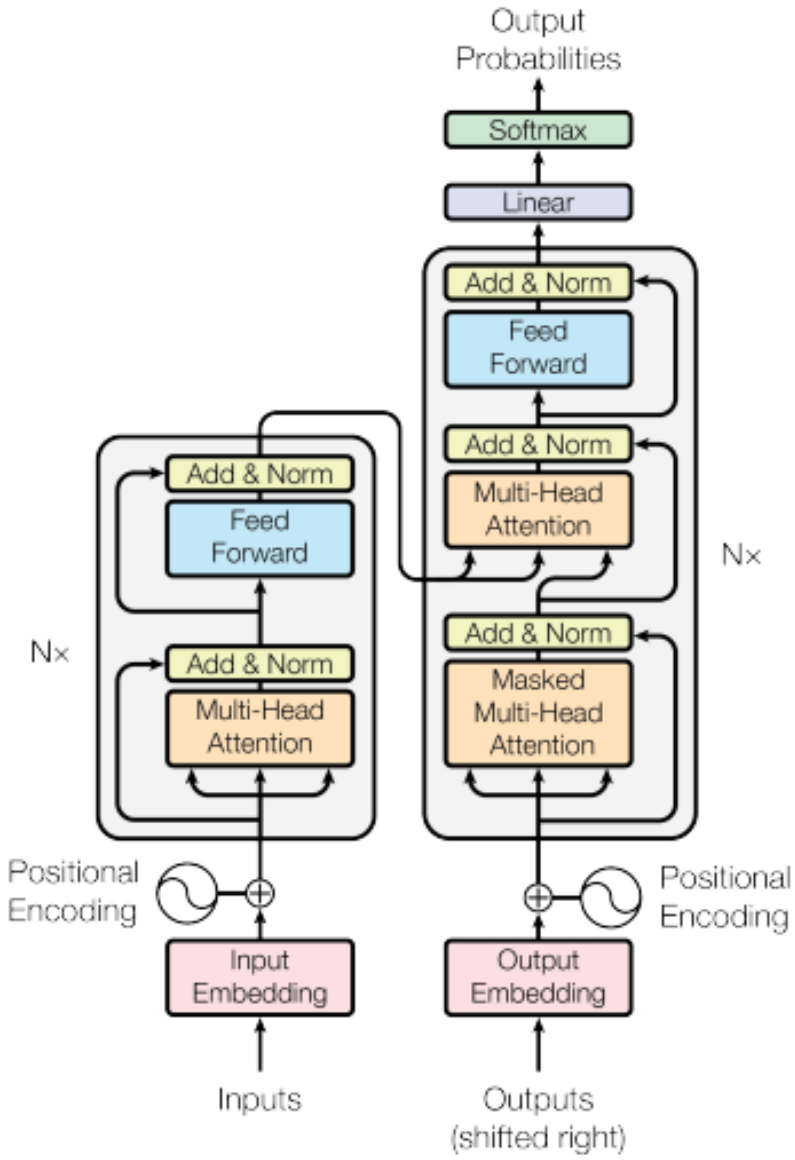
Transformers: how, why?

- Key concept: **self-attention** (*eg* attention without recurrence) $\rightarrow$ treat sentences/paragraphs as a block, no longer word by word
- Very parallelizable $\rightarrow$ very **scalable** (trained on billions or trillions of tokens)
- **New paradigm** in machine learning research:
  - transformers replaced nearly all other neural model
  - just transformers on massive datasets $\rightarrow$ no more creativity in architectures?
  - if you're not a GAFA or a billionnaire, you download a pre-trained transformer model and fine-tune it on your data, or do prompt-engineering.

3.2/ Transformers: model

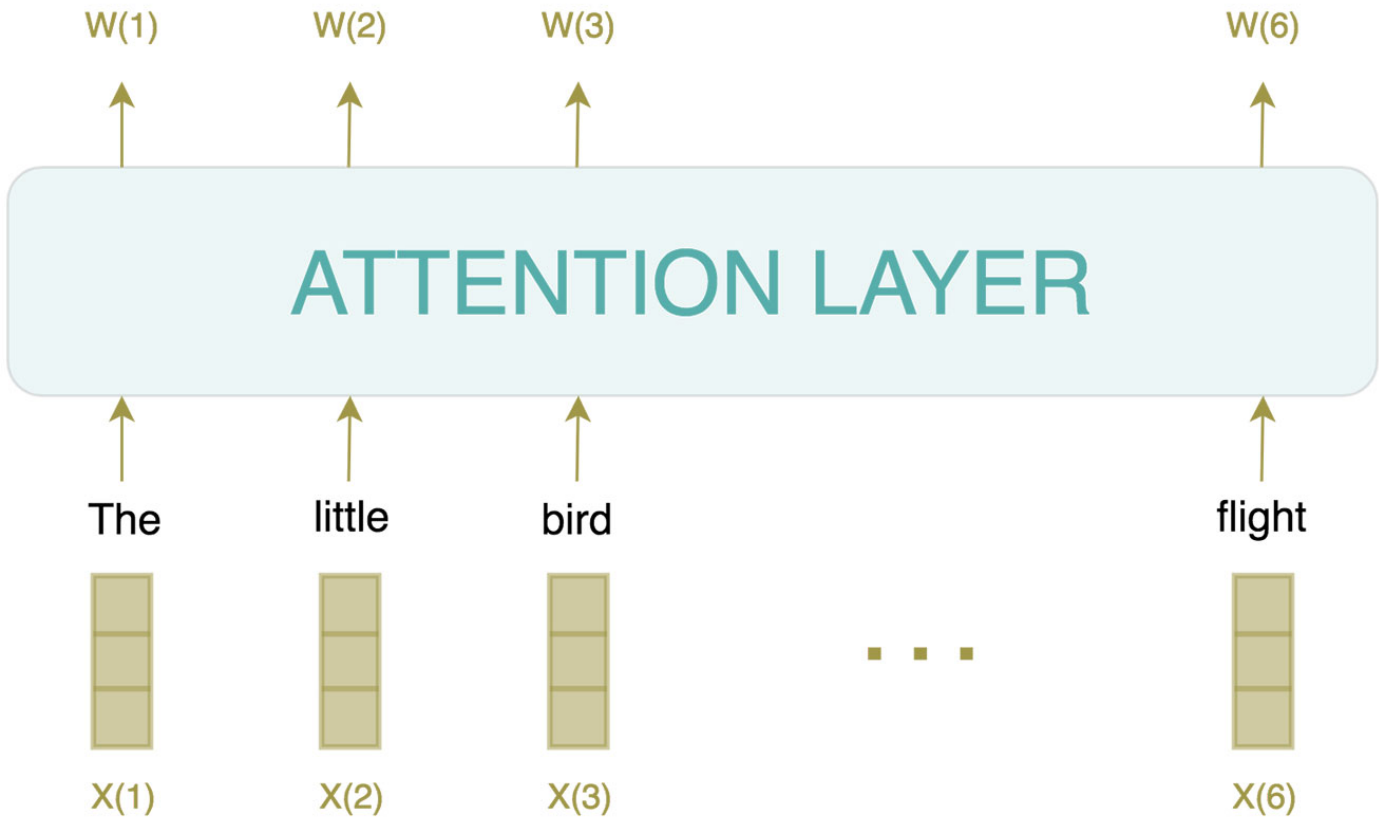
Building blocks of the transformer:

- token embeddings
- self-attention block
- positional encoding
- MLP, layer normalization, skip-layer connections



3.2/ Transformers: model

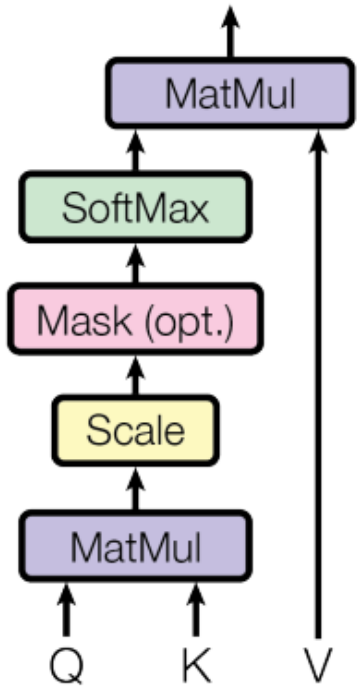
Each layer transforms a (fixed-size) sequence of embeddings to a sequence of embeddings.



3.2/ Transformers: model

Self-attention: formally

- the embeddings matrix (size = n_tokens × embedding_dim) is linearly projected to three separate matrices: **Query, Key, Value**
- Query: what to look for
  Key: information contained in token
  Value: information to pass to next layer
- trainable parameters: M_Q, M_K, M_V
  (matrices for linear production of Q, K, V)
- $Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$



3.2/ Transformers: model

Self-attention: prosaically

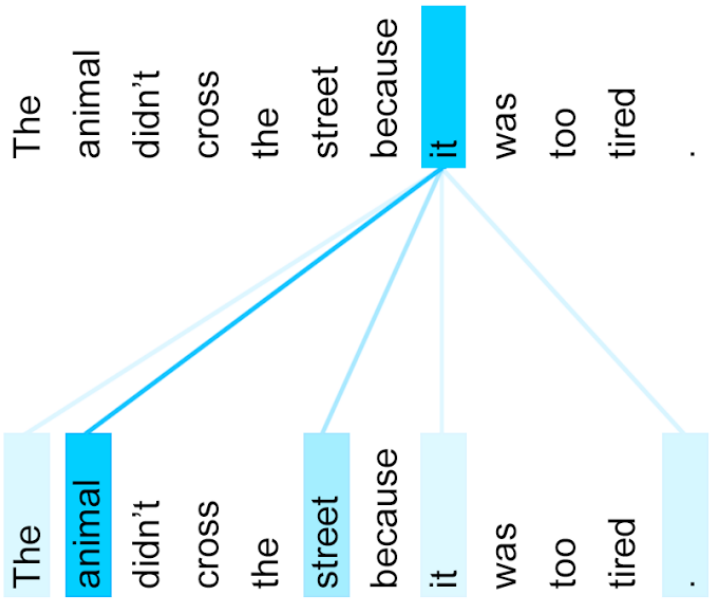
- Each token is directly connected to all other tokens
- QKV learns, for each token, which tokens to **attend** to
- Each output token is a weighted sum of all the Values
- Efficient way to combine the information of all tokens, and pass a transformed sequence to the next layer
- For convenience, embedding size of token embeddings Q, K, and V are usually all the same (*eg* 512 or 768)

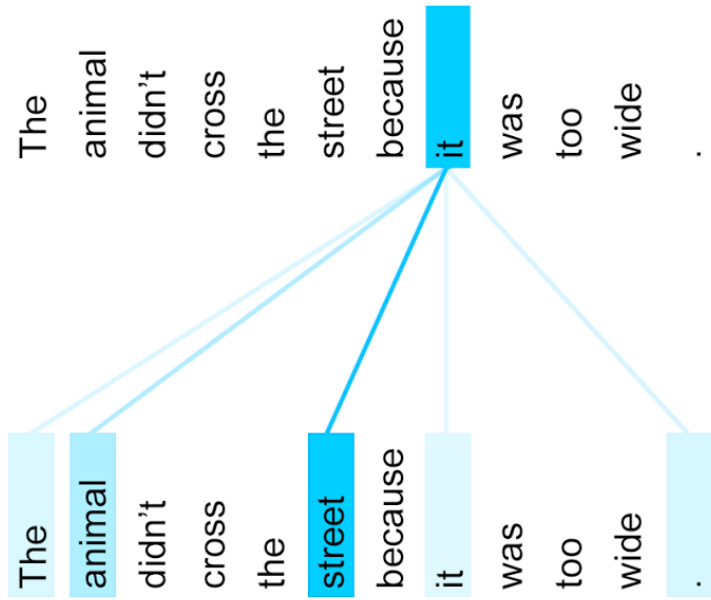
Multi-head attention: several (*eg* 12) mini-attentions, each responsible for a portion (*eg* 1/12) of the embedding size.

NB: Memory requirement $\sim$ (sequence length)^2 $\rightarrow$ **bottleneck**

3.2/ Transformers: model

Self-attention example: learning what "it" refers to



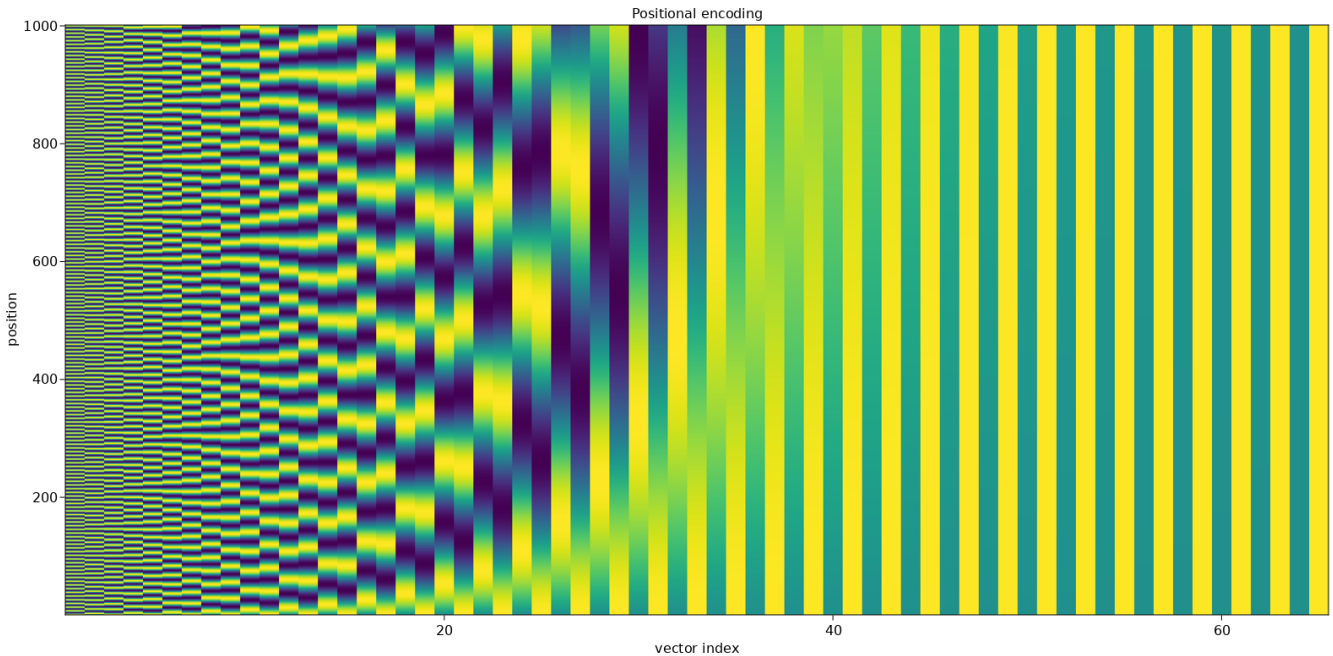


3.2/ Transformers: model

Self-attention treats sentence as a block, but has no positional information: order of tokens is not important.

To solve this, a **positional encoding** is added to the input embeddings: hard-code a position signature.
This can be learned, but is usually deterministic:

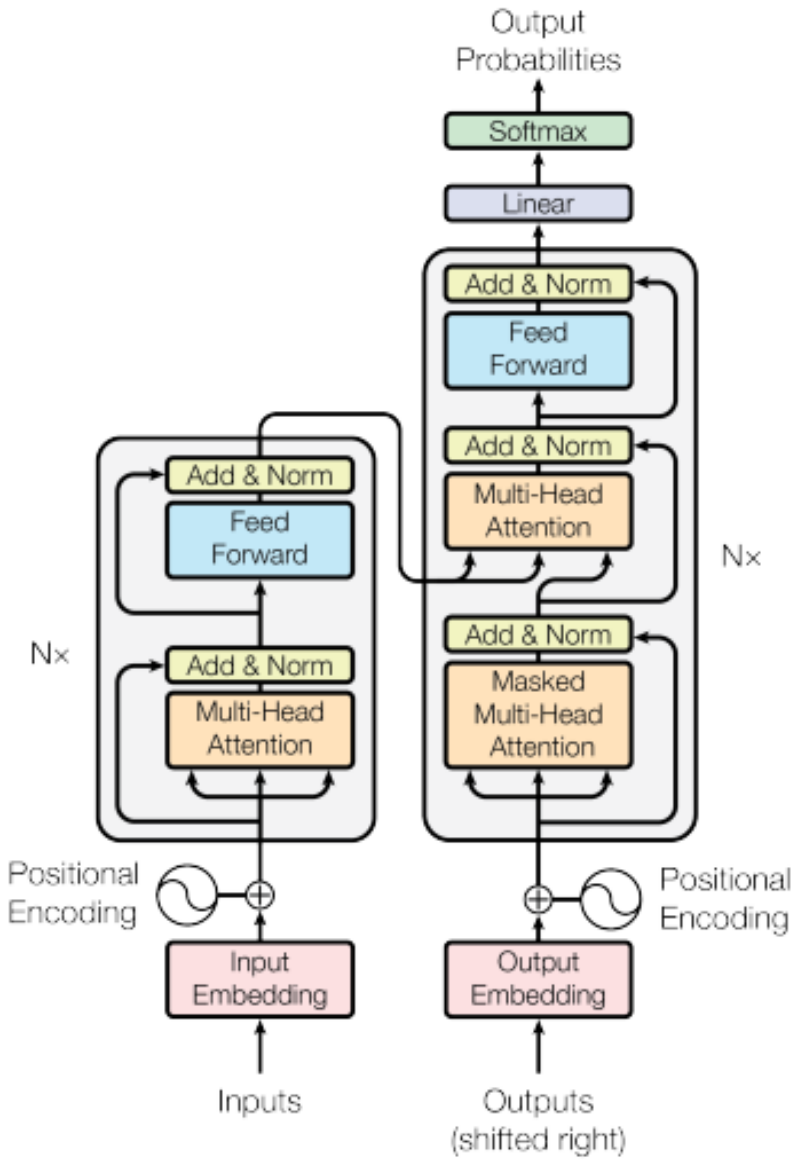
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$



3.2/ Transformers: model

Building blocks of the transformer:

- token embeddings
- self-attention block
- positional encoding
- **MLP, layer normalization, skip-layer connections**
- **masked attention** (in decoder): each token can only attend to previous tokens



3.2/ Transformers: tokenizer

But what are **tokens**? parts of words!

- Each transformer model comes with its specific **tokenizer** (typically $\sim$ 40k vocabulary), balance between two extremes:
  - characters (min vocabulary size, $\sim$ 100)
  - whole words (max vocab size, could be millions)
- Applies the same pre-treatment of raw text as for building the training set
  - cased / uncased
  - add special tokens: *eg* "[CLS]" at start, "[SEP]" at end, "[UNK]" when unknown, padding
  - turn into indices of model's vocabulary
- Ex bert-base-cased: "This is bedazzling!" ->
  ('[CLS]', 'This', 'is', 'bed', '##az', '##z', '##ling', '!', '[SEP]')
- No more elaborate pre-processing (no lemmatization, phrase consolidation, filtering on POS or frequency...): just feed tokenizer with natural text, using the same word tokenizer as was used for training (*eg* GPT: spacy).

3.2/ Transformers: tokenizer

Tokenizers are pre-trained on the training corpus:

- byte-pair encoding (BPE, originally used for compression):
  - start: vocabulary = all characters
  - iteratively add to vocabulary the most common pair of vocabulary items
  - repeat until pre-defined vocabulary size is attained
- modern variants, sentencepiece or wordpiece (with / without whitespace):
  - score for merging = $freq\_pair/(freq\_1st\_elt \times freq\_2d\_elt)$
  - tokenization: find longest subword in vocabulary
- Common words will be a single token
  Most OOV words will be several tokens:
  "hypatia" $\rightarrow$ "hyp", "##at", "##i", "##a"
  Extremely OOV words will be [UNK] (emojis, alphabets...)

3.2/ Transformers: remarks

Transformers in practice:

- Central hub for open source models:
  `https://huggingface.co`
- Python library: transformers
  Useful bindings, pipelines (*eg* sequence classification, token classification, text generation, translation, ...)
- Need GPU to run/train in reasonable time
- Most models come in different sizes: small, base, large... The bigger, the better (and slower).

3.3/ BERT: motivation

BERT: Bidirectional Encoder Representations from Transformers
Masked language model for NLP tasks

- only the encoder half of the Transformer
- pre-trained on massive corpora for **masked language model**
  Essentially, learn to reconstruct corrupted sentences:

  "The [MASK] is [MASK] the mat"

- fine-tunable for SOTA performance on NLP tasks: sequence classification, token classification, question-answering, language inference, ...
- intuition: the pre-training gives the model a fine "understanding" of semantics, that we can harness for many tasks. A few hundred examples are enough during fine-tuning.

3.3/ BERT: model

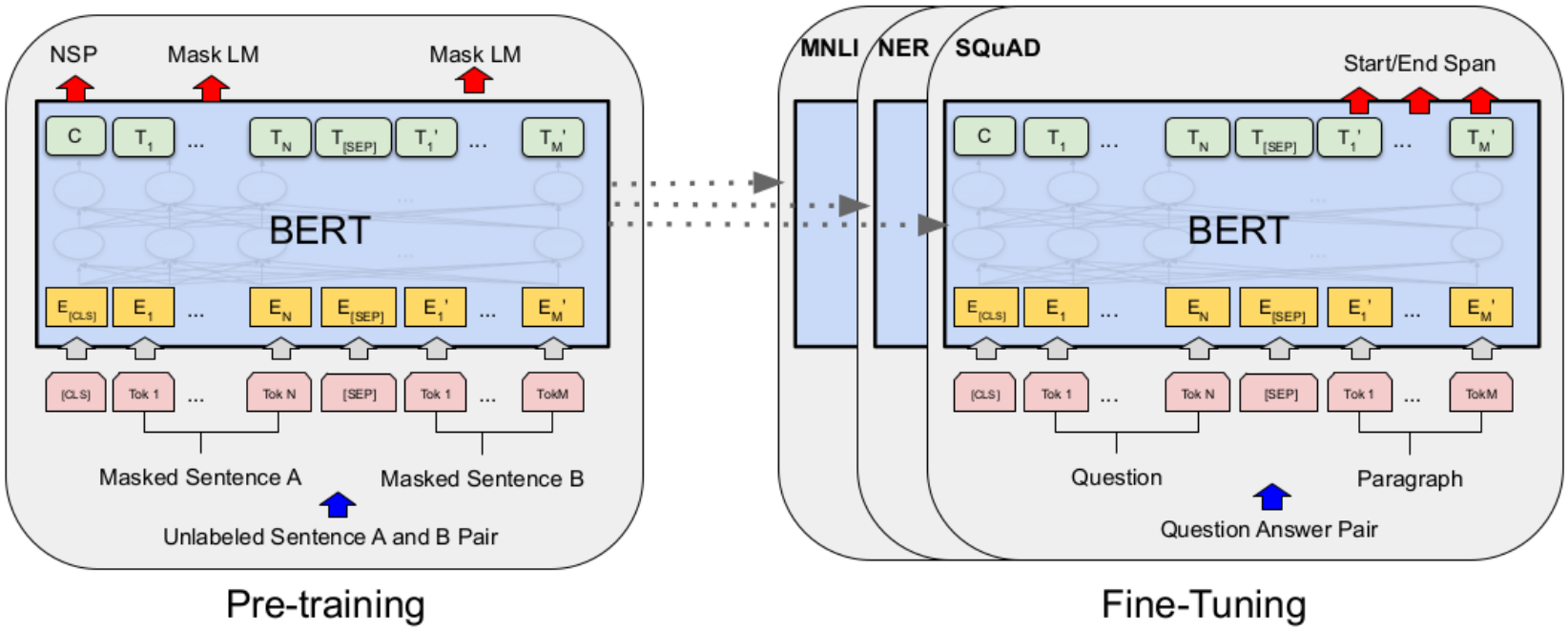
Original BERT: Devlin et al. (2018)

- Pre-trained on two simultaneous tasks:
  - MLM (randomly mask 15% of tokens)
  - next sentence prediction: sentences are fed 2 by 2 (with special separator and context embedding), learn to predict if they follow each other or not
- Fine-tuned by adding special trainable "head" for specific task:
  - sequence classification: MLP + softmax on the "[CLS]" output embedding
  - token classification: MLP + softmax on each token output embedding

J. Devlin, M. Chang, K. Lee, and K. Toutanova, 2019, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1*

3.3/ BERT: model

Original BERT: pretraining MLM + NSP, fine-tune custom head



3.3/ BERT: variants

Several variants:

- **RoBERTa** (Facebook, 2019): no NSP, fill with consecutive sentences, dynamic masking, larger batches, more data, trained longer
  First model to surpass Human baselines on GLUE!
- **DeBERTa** (Microsoft, 2020): better relative-position encoding, more adversarial MLM (Replaced Token Detection), larger models
  First model to surpass Human baselines on superGLUE!
- multilingual BERT, XLNet, language-specific (*eg* FlauBERT), distilBERT, ...

Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, 2019, "RoBERTa: A robustly optimized BERT pretraining approach", arXiv preprint arXiv:1907.11692

P. He, X. Liu, J. Gao, and W. Chen, 2020, "Deberta: Decoding-enhanced BERT with disentangled attention", *International Conference on Learning Representations*

3.3/ BERT: variants

Another useful variant: **SBERT**, for static sentence embeddings

- Authors note that raw BERT embeddings (w/o fine-tuning) perform worse than averaged word2vec embeddings for downstream tasks!
- Also, using BERT two sentences at a time to compute sentence similarity is very inefficient ($n(n-1)/2$ runs)
- Solution: fine-tune a BERT-like model using siamese networks on 1M natural language inference sentence-pairs (NLI = 3-way classification: entailment, contradiction, neutral)
- Result: high-quality, fast to compute sentence embeddings
- Also available in multilingual version

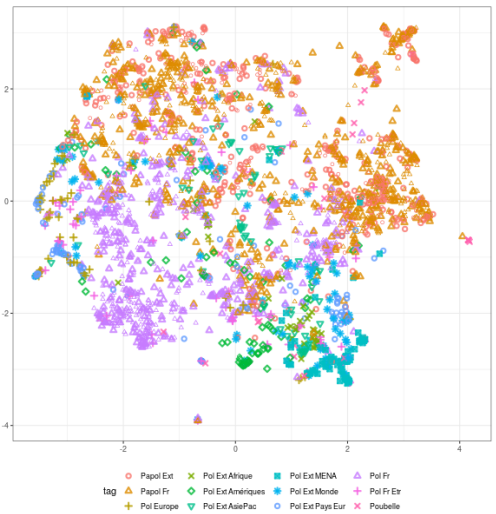
N. Reimers, I. Gurevych, 2019, "Sentence-BERT: Sentence embeddings using siamese bert-networks", *EMNLP*

N. Reimers, I. Gurevych, 2020, "Making monolingual sentence embeddings multilingual using knowledge distillation", *EMNLP*

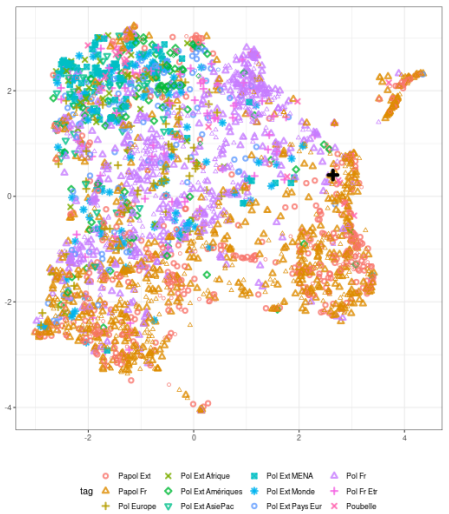
3.3/ BERT: variants

Example: UMAP on DTM, fastText and SBERT, on French press

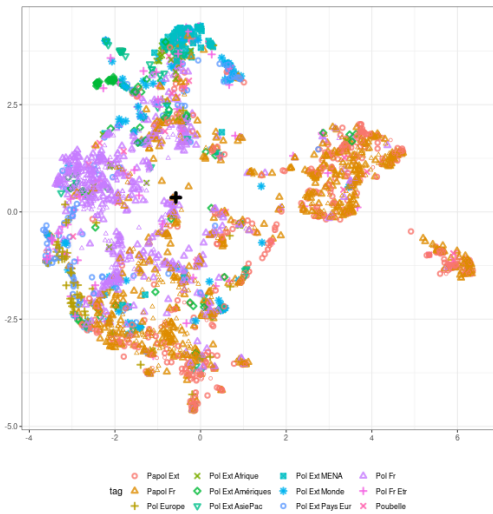
DTM (tf-idf)



fastText



SBERT



NB:

- BERTopic = SBERT + UMAP + DB-clustering
- Superiority of supervized learning

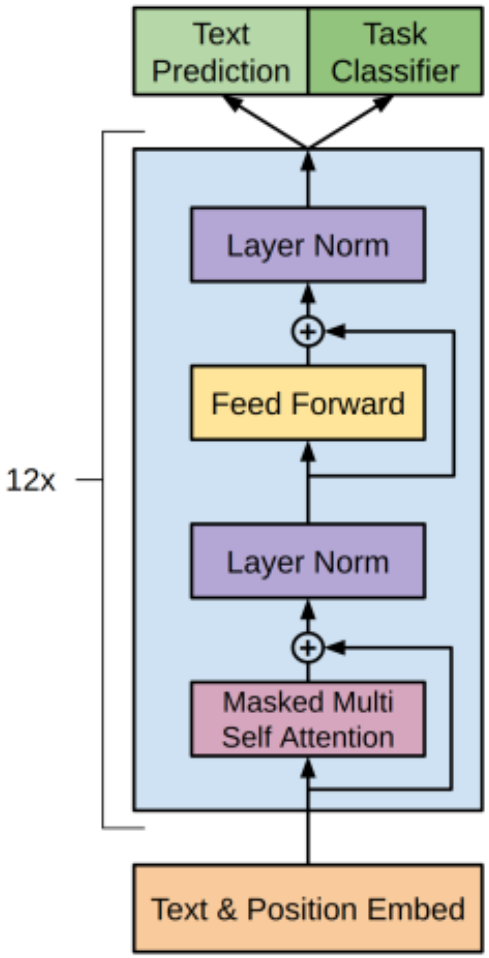
3.4/ GPT: motivation

GPT (OpenAI, 2018): Generative Pre-training on Transformers

- Causal language modeling: predict next token
- **Decoder-only** transformer (masked attention $\rightarrow$ parallel)
  Simple architecture: 12-layer transformer decoder, 12-head 64D attention, learned positional embeddings
- For **generation**, or **NLP tasks** (fine-tuning with custom head)
  (super)GLUE: BiLSTM++ $<$ GPT finetune $<$ BERT $<$ GPT-3 few-shot $<$ RoBERTa $<$ DeBERTa

A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, 2018, "Improving language understanding with unsupervised learning", Technical report, OpenAI.

3.4/ GPT: model



GPT-1:

- Simple architecture:
  - 12-layer transformer decoder
  - 12-head 64D attention
  - learned positional embeddings
- Trained on BookCorpus (4.5GB from 7K unpublished books)

3.4/ GPT: mutations

Mutations of GPT: bigger $\rightarrow$ emergent properties

- GPT-1 (2018): 0.12B params, 4.5GB text (BookCorpus)
- GPT-2 (2019): 1.5B params, 40GB text (WebText)
  - 10xGPT-1: 48 layers, 1600 dim
  - **zero-shot NLP**: SOTA (some tasks) **without fine-tuning**, just by asking the question (as generation "context")
  - Credible text generation
  - Open release postponed for "safety concerns"
- GPT-3 (2020): 175B params, 570GB text (CC, Wiki, ...)
  - 1000xGPT-1:
  - LLMs are **few-shot learners** for NLP tasks
    $\rightarrow$ equivalent to BERT-large with just 32 in-context examples
  - Still better generation, able to code in python, java, C++, ...
  - Closing of OpenAI: model not public, API only

3.4/ GPT: mutations

Mutations of GPT (ct'd)

- GPT-3.5 (2022): unknown details (smaller?, more data)
- GPT-4 (2023): unknown details (multimodal: text and image input), "sparks of AGI"

Outside of OpenAI:

- GPT-3 competitors: GPT-J, Gopher (280B), Megatron-Turing NLG (530B), PaLM (540B), BLOOM (176B), ...
- Chinchilla revolution (2022, DeepMind: 70B params, 1.4T tokens): smaller models, on more data, trained for more epochs
- **Llama** revolution (2023, Facebook, 7B-65B params, 1.4T tokens): smaller, competitive with largest SOTA models, and open-source (for research)
- StableLM, Falcon (40B, 1T tokens, current open SOTA), ...

3.4/ GPT: generation

Main difference with BERT: **generative** model

Several strategies for generation:

- **Greedy**: choose most probable token at each t
- **Temperature**: parameter to soften greediness (0: only most probable, >0: mix it up, get creative)
- **Beam search**: k parallel hypotheses (predict next token for each branch, then prune to keep k) until end, then choose
- **Typical**: force generated answers to have a minimum amount of information (measured by entropy), to make answers more interesting

3.4/ GPT: prompts to chat

Important difference: **foundational** models vs **chat** models:

- **foundational**: just a causal LLM (GPT-3.5, Llama, ...)
  - Excellent at **continuing text**
  - Lot of information stored, but hard to access
  - Will be racist if even slightly nudged
- **instruct**: fine-tuned on instruction-answer datasets
  - Slightly worse LLM, but more helpful, easier to access
- **chat**: fine-tuned by reinforcement learning with human feedback (RLHF) (ChatGPT, GPT4, Vicuna, OpenAsst...)
  - Fine-tuned and hidden-prompt-hacked to be a helpful chat assistant
  - RLHF: exploit RL advances (*eg* alphaGo, alphaStar, ...), with feedback from (underpaid) humans
  - Alignment: making models "safe for work" (not offensive, inclusive, not legally dangerous in any way...)

3.4/ GPT: current developments

Brave new **LLM** world:

- New job in town: **prompt hacker**!
  - learn to "condition" a LLM to extract the information you want from it (language models, but also image generation *eg* StableDiffusion, MidJourney...)
  - early challenges: get commercial LLM to reveal its system prompt, or say offensive/illegal things
- advanced auto-prompting for chat models
  - chain-of-thought, reflexion, tree-of-thought, ...
- Quantization (4bit): run LLMs on your laptop
  - *eg* `https://twitter.com/HubertMacron`

3.4/ GPT: current developments

Brave new **LLM** world (ct'd):

- Frontiers:
  - context size, positional embeddings,
  - LLM-agents (autoGPT, langchain), plugins (access to internet, mail, python, ...), ...

- Exciting (dubious?) promises in social sciences:
  - LLM-assisted data augmentation
    $\rightarrow$ *LLM learns researcher's judgment*
  - Automatized data augmentation?
    $\rightarrow$ *researcher trusts LLM's judgment*
  - Silicon samples?? (auto soc. survey answering)
    $\rightarrow$ *LLM replaces human response*

Further reading

- Dan Jurafsky and James H. Martin, 2023, *Speech and Language Processing* (3rd ed. draft), chapters I.7, I.9-12, `https://web.stanford.edu/~jurafsky/slp3`
- R. Rojas, 1996, *Neural Networks - A Systematic Introduction*, Springer-Verlag
- I. Goodfellow, Y. Bengio, A. Courville, 2016, *Deep Learning*, MIT Press

Tutorials:

- Tensorflow playground: `https://playground.tensorflow.org`
- Illustrated transformer: `https://jalammar.github.io/illustrated-transformer/`
- Learn prompting: `https://learnprompting.org/`
- Local Llama (quantized models, runnable on laptop): `https://www.reddit.com/r/LocalLLaMA/wiki/guide/`

Questions and comments: `julien.boelaert@univ-lille.fr`