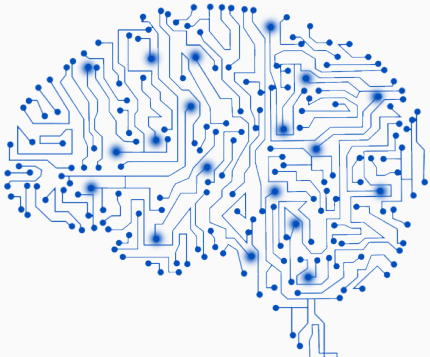
Master 2 Statistique & Data Science, Ingénierie Mathématique

Réseaux de neurones profonds pour l'apprentissage
Deep neural networks for machine learning

Course VI – Recurrent Neural Networks and Transformers

Bruno Galerne

2025-2026



Credits

Most of the slides from **Marc Lelarge's** course "Recurrent Neural Networks theory"

https://dataflowr.github.io/website/

Sequence prediction and classification

Text sequences

- Text auto-completion
- Sentiment analysis

Audio sequences

- Speech to text
- Music generation

Time-series forecasting

- Market price prediction
- Weather forecast

Standard approaches

Data: Sequences of the form $(x_1, \ldots, x_t)$. **Objective:** Guess next iterate x_{t+1}.

Standard approaches

Data: Sequences of the form $(x_1, \ldots, x_t)$. **Objective:** Guess next iterate x_{t+1}.

Classical ML models

- **Hidden Markov Models:** Probabilistic model where current value is drawn according to a distribution dependent on a hidden state.
- **Auto-regressive models:** Linear relationship between current and previous iterates.

Standard approaches

Data: Sequences of the form $(x_1, \ldots, x_t)$. **Objective:** Guess next iterate x_{t+1}.

Classical ML models

- **Hidden Markov Models:** Probabilistic model where current value is drawn according to a distribution dependent on a hidden state.
- **Auto-regressive models:** Linear relationship between current and previous iterates.

Convolutional Neural Networks

- We can integrate the temporal dimension with a **1d convolution**.
- Standard architecture: **WaveNet** (Van den Oord et al., 2016)

Standard approaches

Data: Sequences of the form $(x_1, \ldots, x_t)$. **Objective:** Guess next iterate x_{t+1}.

Classical ML models

- **Hidden Markov Models:** Probabilistic model where current value is drawn according to a distribution dependent on a hidden state.
- **Auto-regressive models:** Linear relationship between current and previous iterates.

Convolutional Neural Networks

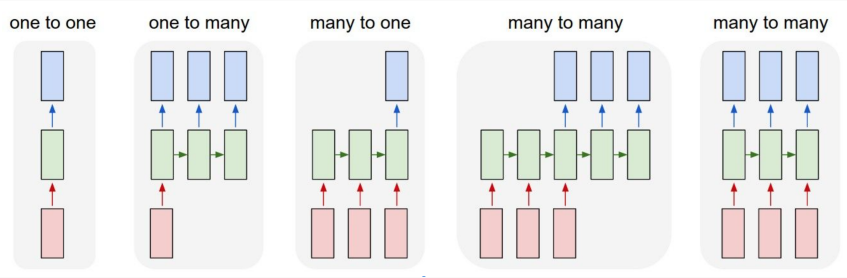
- We can integrate the temporal dimension with a **1d convolution**.
- Standard architecture: **WaveNet** (Van den Oord et al., 2016)

Transformers

- Based on a selection procedure using **attention** modules
- Current **state-of-the-art** for natural language processing

Recurrent Neural Networks

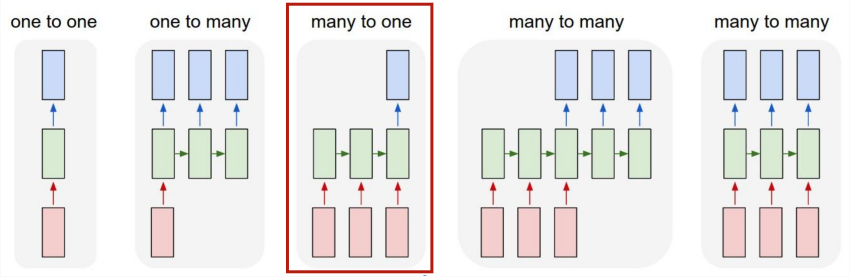
- Several variants



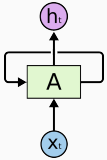
source: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Recurrent Neural Networks

- Focus first on many to one (e.g. sequence classification)



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



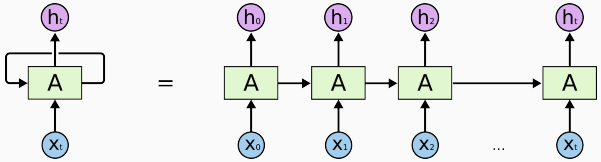
Causality & short-term dependency

We process a sequence of vectors x_t by applying a **recurrence formula** at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

- h_{t-1} = previous state, h_t = current state
- f_W = some function with parameters W
- x_t = input column vector at time step t

source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/



Usual implementation

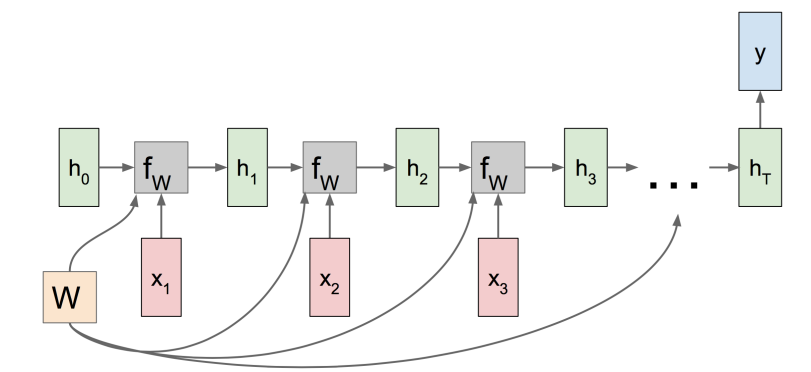
- Typically (note the use of the $\tanh$ non-linearity):

$$h_t = \tanh(W_{hh}\, h_{t-1} + W_{xh}\, x_t)$$

- Output: $y_t = W_{hy}\, h_t$ or $y_t = \mathrm{softmax}(W_{hy}\, h_t)$

source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN computational graphs



Backpropagation through time

Let us detail the gradient computation! (blackboard)

source: J. Johnson

A simple binary sequence classification problem

- Can you guess the task?

Sequence	Class
[1, 1, -1, -1, 1, -1]	1
[1, -1, 1, -1]	1
[1, -1, 1, 1, -1, 1, -1, -1]	1
[1, 1, -1, -1, -1, 1, -1, 1]	0
[1, -1, -1, 1, 1, -1]	0
[1, -1, -1, 1]	0

A simple binary sequence classification problem

- Can you guess the task?

Sequence	Class
[1, 1, -1, -1, 1, -1] = (())()	1
[1, -1, 1, -1] = ()()	1
[1, -1, 1, 1, -1, 1, -1, -1] = ()(()())	1
[1, 1, -1, -1, -1, 1, -1, 1] = (()))()(	0
[1, -1, -1, 1, 1, -1] = ())(()	0
[1, -1, -1, 1] = ())(	0

A simple binary sequence classification problem

- Can you guess the task?

Sequence	Class
[1, 1, -1, -1, 1, -1] = (())()	1
[1, -1, 1, -1] = ()()	1
[1, -1, 1, 1, -1, 1, -1, -1] = ()(()())	1
[1, 1, -1, -1, -1, 1, -1, 1] = (()))()(	0
[1, -1, -1, 1, 1, -1] = ())(()	0
[1, -1, -1, 1] = ())(	0

- How would you solve this task?

A (less) simple binary sequence classification problem

- We will make it a bit more complicated with **colored parenthesis**, example with 10 colors.
- **Rule:** Opening parenthesis $i \in [0, 4]$ with corresponding closing parenthesis $j \in [5, 9]$ such that $i + j = 9$.

Sequence	Class
[2, 0, 9, 7, 0, 9] = (()()	1
[1, 8, 3, 6] = ()()	1
[0, 9, 2, 4, 5, 2, 7, 7] = ()(()())	1
[0, 2, 7, 9, 7, 2, 7, 3] = (()))()(	0
[1, 8, 9, 0, 1, 9] = ())(()	0
[1, 8, 7, 1] = ())(	0

A (less) simple binary sequence classification problem

- We will make it a bit more complicated with **colored parenthesis**, example with 10 colors.
- **Rule:** Opening parenthesis $i \in [0, 4]$ with corresponding closing parenthesis $j \in [5, 9]$ such that $i + j = 9$.

Sequence	Class
[2, 0, 9, 7, 0, 9] = (()()	1
[1, 8, 3, 6] = ()()	1
[0, 9, 2, 4, 5, 2, 7, 7] = ()(()())	1
[0, 2, 7, 9, 7, 2, 7, 3] = (()))()(	0
[1, 8, 9, 0, 1, 9] = ())(()	0
[1, 8, 7, 1] = ())(	0

- How would you solve this task?

Elman network (1990)

First implementation of RNNs, simple ReLU activation and linear output.

- **Initial hidden state:** $h_0 = 0$
- **Update:** $h_t = \mathrm{ReLU}(W_{xh}\, x_t + W_{hh}\, h_{t-1} + b_h)$
- **Final prediction:** $y_T = W_{hy}\, h_T + b_y$

Elman network (1990)

First implementation of RNNs, simple ReLU activation and linear output.

- **Initial hidden state:** $h_0 = 0$
- **Update:** $h_t = \mathrm{ReLU}(W_{xh}\, x_t + W_{hh}\, h_{t-1} + b_h)$
- **Final prediction:** $y_T = W_{hy}\, h_T + b_y$

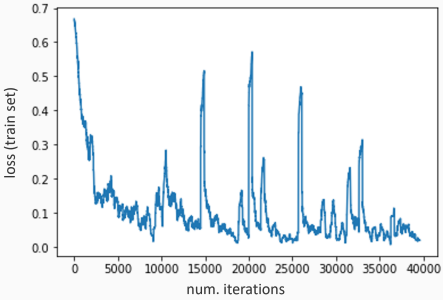
```
class RecNet(nn.Module):
    def __init__(self, dim_input, dim_recurrent, dim_output):
        super(RecNet, self).__init__()
        self.fc_x2h = nn.Linear(dim_input, dim_recurrent)
        self.fc_h2h = nn.Linear(dim_recurrent, dim_recurrent, bias = False)
        self.fc_h2y = nn.Linear(dim_recurrent, dim_output)

    def forward(self, x):
        h = x.new_zeros(1, self.fc_h2y.weight.size(1))
        for t in range(x.size(0)):
            h = torch.relu(self.fc_x2h(x[t,:]) + self.fc_h2h(h))
        return self.fc_h2y(h)
```

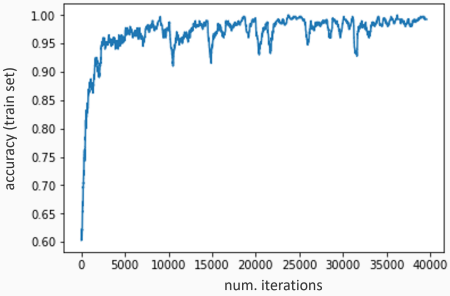
Training

- We encode the symbol at time t as a one-hot vector x_t
- To simplify the processing of variable-length sequences, we are processing samples (i.e. sequences) one at a time. **We do not consider batches**.

```
RNN = RecNet(dim_input = nb_symbol, dim_recurrent=50, dim_output=2)

cross_entropy = nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(RNN.parameters(), lr=learning_rate)

for k in range(nb_train):
    x,l = generator.generate_input()
    y = RNN(x)
    loss = cross_entropy(y,l)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

Results





- Loss decreases and fraction of correct classification increases but did our network learn?

Gating

Main idea

- Gates are a way to optionally let information through.
- The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through," while a value of one means "let everything through!".

Gating

Main idea

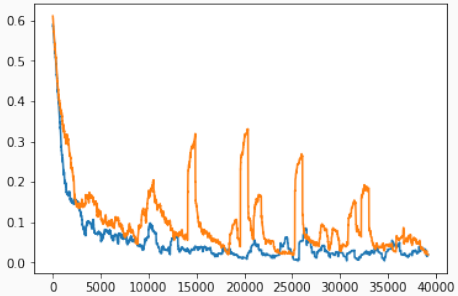
- Gates are a way to optionally let information through.
- The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through," while a value of one means "let everything through!".

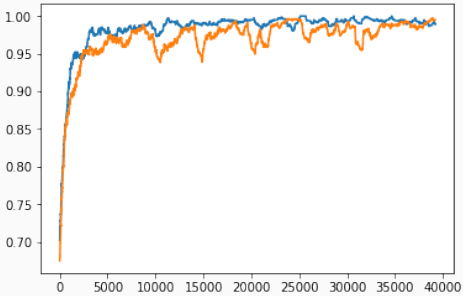
- **Recurrence relation:** $\overline{h}_t = \mathrm{ReLU}(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$
- **Forget gate:** $z_t = \mathrm{sigm}(W_{xz}x_t + W_{hz}h_{t-1} + b_z)$
- **Hidden state:** $h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \overline{h}_t$

Gated RNN

```
class RecNetGating(nn.Module):
    def __init__(self, dim_input=10, dim_recurrent=50, dim_output=2):
        super(RecNetGating, self).__init__()
        self.fc_x2h = nn.Linear(dim_input, dim_recurrent)
        self.fc_h2h = nn.Linear(dim_recurrent, dim_recurrent, bias = False)
        self.fc_x2z = nn.Linear(dim_input, dim_recurrent)
        self.fc_h2z = nn.Linear(dim_recurrent,dim_recurrent, bias = False)
        self.fc_h2y = nn.Linear(dim_recurrent, dim_output)

    def forward(self, x):
        h = x.new_zeros(1, self.fc_h2y.weight.size(1))
        for t in range(x.size(0)):
            z = torch.sigmoid(self.fc_x2z(x[t,:])+self.fc_h2z(h))
            hb = torch.relu(self.fc_x2h(x[t,:]) + self.fc_h2h(h))
            h = z * h + (1-z) * hb
        return self.fc_h2y(h)
```

Results

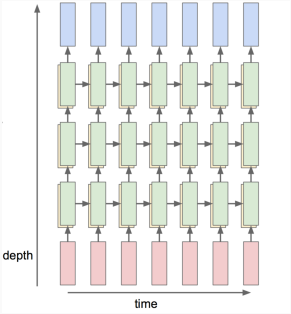




- Orange = previous RNN.
- Blue = Gated RNN.
- Is there a benefit with gating?

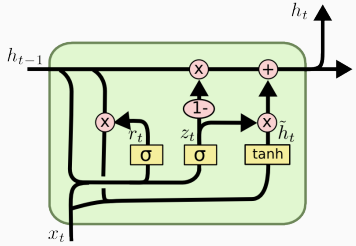
LSTM, GRU and multi-layer RNNs

- More parameters than RNN.
- Mitigates **vanishing gradient** problem through **gating**.
- Widely used and SOTA in many sequence learning problems.



GRU: Gated Recurrent Unit (Cho et al., 2014)

- **Recurrence relation:** $\overline{h}_t = \tanh(W_{xh}\, x_t + W_{hh}\, (r_t \odot h_{t-1}) + b_h)$
- **Forget gate:** $z_t = \mathrm{sigm}(W_{xz} x_t + W_{hz} h_{t-1} + b_z)$
- **Reset gate:** $r_t = \mathrm{sigm}(W_{xr}\, x_t + W_{hr}\, h_{t-1} + b_r)$
- **Hidden state:** $h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \overline{h}_t$



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

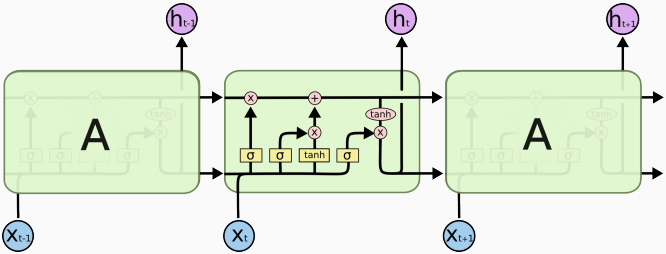
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

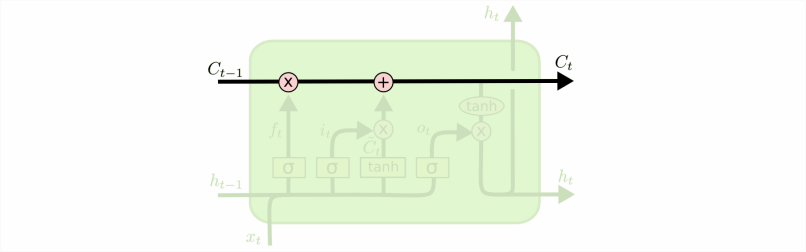
LSTM: Long Short-Term Memory (Hochreiter and Schmidhuber, 1997)



source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Inside LSTMs

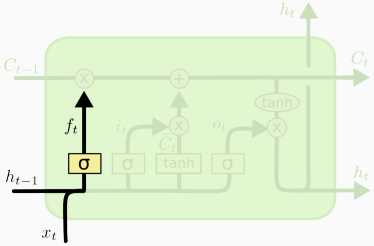
- Cell state



source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Inside LSTMs

- Forget gate layer

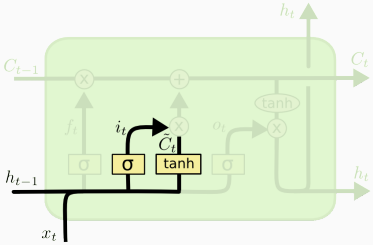


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Inside LSTMs

- Input gate layer



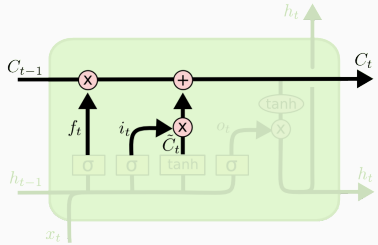
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Inside LSTMs

- Update cell state

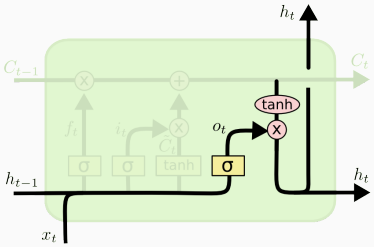


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Inside LSTMs

- Output gate



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

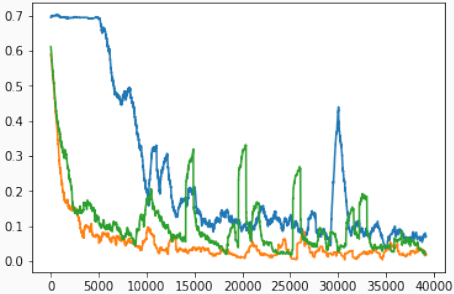
source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

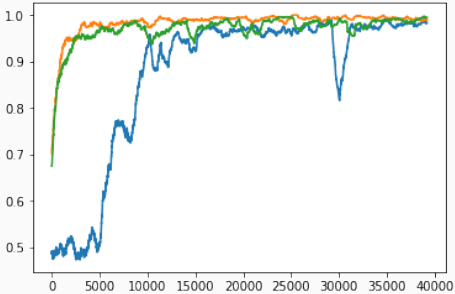
LSTMs in PyTorch

```
class LSTMNet(nn.Module):
    def __init__(self, dim_input, dim_recurrent, num_layers, dim_output):
        super(LSTMNet, self).__init__()
        self.lstm = nn.LSTM(input_size = dim_input,
                            hidden_size = dim_recurrent,
                            num_layers = num_layers)
        self.fc_o2y = nn.Linear(dim_recurrent,dim_output)

    def forward(self, x):
        x = x.unsqueeze(1)
        output, _ = self.lstm(x)
        # only last layer, shape (seq. len., bs, dim_recurrent)
        # drop the batch index
        output = output.squeeze(1)
        # keep only the last hidden variable
        output = output.narrow(0, output.size(0)-1,1)
        # shape (1, dim_recurrent)
        return self.fc_o2y(F.relu(output))
```

Note: the prediction is done from the hidden state, hence also called the output state.

Results





- Green = Elman RNN.
- Orange = Gated RNN.
- Blue = LSTM.
- Is there a benefit with LSTM?

Common wisdom in 2015

- Josefowicz et al. (2015) conducted an extensive exploration of different recurrent architectures, they wrote:

  "We have evaluated a variety of recurrent neural network architectures in order to find an architecture that reliably outperforms the LSTM. Though there were architectures that outperformed the LSTM on some problems, **we were unable to find an architecture that consistently beat the LSTM and the GRU in all experimental conditions**.*"*

Common wisdom in 2015

- Josefowicz et al. (2015) conducted an extensive exploration of different recurrent architectures, they wrote:

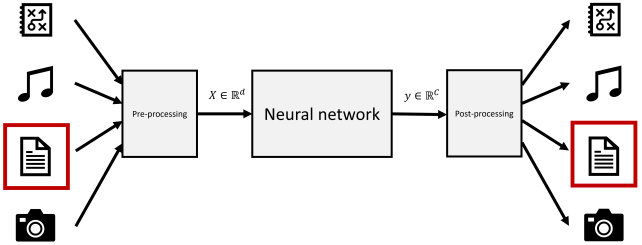
  "We have evaluated a variety of recurrent neural network architectures in order to find an architecture that reliably outperforms the LSTM. Though there were architectures that outperformed the LSTM on some problems, **we were unable to find an architecture that consistently beat the LSTM and the GRU in all experimental conditions**.*"*

- Now let see if the LSTM is performing better on our task of checking for **balanced parentheses**!

Introduction to Natural Language Processing (NLP)

Typical language tasks

- **Text to label:** Sentiment analysis, text categorization, true/false question answering
- **Text to text:** Translation, summarization, correction (grammar), question answering, chatbots, content creation, auto-completion
- **Others:** speech to text, text to speech / image / video.



Auto-completion is all you need...!

Next word prediction

- **Objective:** Guess the next character/word/token (this is a **classification** task!).
- **Definition:** Let $\mathcal{D}$ be a finite dictionary (i.e. set) of characters/words/tokens, and $(u_1^i, \dots, u_t^i)_{i \in [\![1,n]\!]} \in \mathcal{D}^{t \times n}$ a training dataset of sequences (e.g. text doc.) of length t.
- **Task:** Let $(u_1, \dots, u_t) \in$ a sequence. We want to predict u_t given $(u_1, \dots, u_{t-1})$.

"Hello, my name is Sam. How are"	$\rightarrow$	"you"
"To be or not to"	$\rightarrow$	
"I am playing in the"	$\rightarrow$	

Auto-completion is all you need...!

Next word prediction

- **Objective:** Guess the next character/word/token (this is a **classification** task!).
- **Definition:** Let $\mathcal{D}$ be a finite dictionary (i.e. set) of characters/words/tokens, and $(u_1^i, \dots, u_t^i)_{i \in [\![1,n]\!]} \in \mathcal{D}^{t \times n}$ a training dataset of sequences (e.g. text doc.) of length t.
- **Task:** Let $(u_1, \dots, u_t) \in$ a sequence. We want to predict u_t given $(u_1, \dots, u_{t-1})$.

"Hello, my name is Sam. How are"	$\rightarrow$	"you"
"To be or not to"	$\rightarrow$	"be"
"I am playing in the"	$\rightarrow$	

Auto-completion is all you need...!

Next word prediction

- **Objective:** Guess the next character/word/token (this is a **classification** task!).
- **Definition:** Let $\mathcal{D}$ be a finite dictionary (i.e. set) of characters/words/tokens, and $(u_1^i, \dots, u_t^i)_{i \in [\![1,n]\!]} \in \mathcal{D}^{t \times n}$ a training dataset of sequences (e.g. text doc.) of length t.
- **Task:** Let $(u_1, \dots, u_t) \in$ a sequence. We want to predict u_t given $(u_1, \dots, u_{t-1})$.

"Hello, my name is Sam. How are"	$\rightarrow$	"you"
"To be or not to"	$\rightarrow$	"be"
"I am playing in the"	$\rightarrow$	"garden"

Early generative models: statistical models

A simple Markov model (Shannon, 1948)

- **Idea:** Learn the transition probabilities from one word to another.
- **Method:** Learn the probability $p_v(u)$ of a token $u \in \mathcal{D}$ appearing after the token $u \in \mathcal{D}$.

We can think of a discrete source as generating the message, symbol by symbol. It will choose successive symbols according to certain probabilities depending, in general, on preceding choices as well as the particular symbols in question. A physical system, or a mathematical model of a system which produces such a sequence of symbols governed by a set of probabilities, is known as a stochastic process.[3] We may consider a discrete source, therefore, to be represented by a stochastic process. Conversely, any stochastic process which produces a discrete sequence of symbols chosen from a finite set may be considered a discrete source. This will include such cases as:

1. Natural written languages such as English, German, Chinese.

source: A Mathematical Theory of Communication (Shannon, 1948)

Early generative models: statistical models

A simple Markov model (Shannon, 1948)

- **Idea:** Learn the transition probabilities from one word to another.
- **Method:** Learn the probability $p_v(u)$ of a token $u \in \mathcal{D}$ appearing after the token $u \in \mathcal{D}$.

5. First-order word approximation. Rather than continue with tetragram, . . . , *n*-gram structure it is easier and better to jump at this point to word units. Here words are chosen independently but with their appropriate frequencies.

   REPRESENTING AND SPEEDILY IS AN GOOD APT OR COME CAN DIFFERENT NATURAL HERE HE THE A IN CAME THE TO OF TO EXPERT GRAY COME TO FURNISHES THE LINE MESSAGE HAD BE THESE.

6. Second-order word approximation. The word transition probabilities are correct but no further structure is included.

   THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH WRITER THAT THE CHARACTER OF THIS POINT IS THEREFORE ANOTHER METHOD FOR THE LETTERS THAT THE TIME OF WHO EVER TOLD THE PROBLEM FOR AN UNEXPECTED.

source: A Mathematical Theory of Communication (Shannon, 1948)

From text to tokens

Neural networks cannot process text directly, we need to convert it into squence of vectors.

Tokenization

- **Definition:** A token is a sequence of characters that are grouped together as a useful semantic unit for processing. Examples: words, sub-words, characters.
- **Tokenization:** The process of splitting text into tokens.

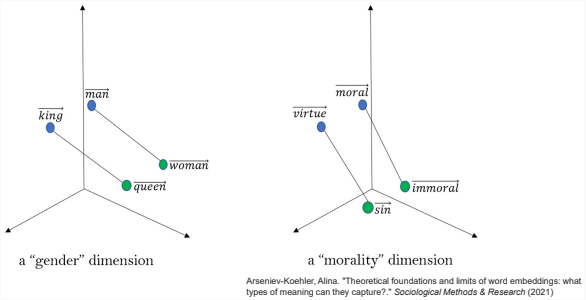
Many tokenization methods exist, for example:

- Word2vec: tokenize by words, and learn a vector representation for each word.
- WordPiece (BERT): tokenize by sub-words, and learn a vector representation for each sub-word.
- BPE (Byte Pair Encoding) (GPT): tokenize by sub-words by grouping frequent character pairs.

From text to tokens

Desirable features of tokenization:

- Convey semantic information (e.g. "dog" and "dogs" should be close).
- Convey grammatical information (noun, verb, plural, ...).



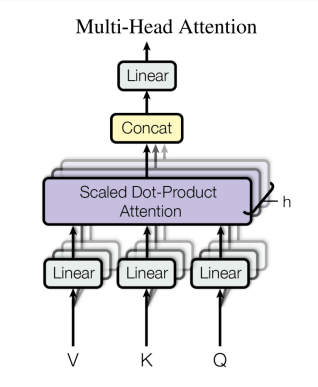
Arseniev-Koehler, Alina. "Theoretical foundations and limits of word embeddings: what types of meaning can they capture?." *Sociological Methods & Research* (2021)

Limitations

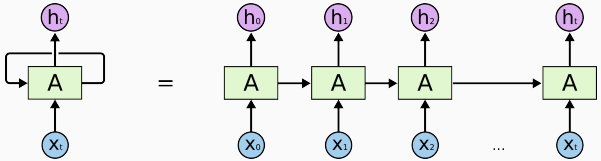
- The meaning of a word depends on context, e.g. "I can buy a can of fish."
- Word tokens should depend on context, not just word itself.
- Transformers / RNN let each word "absorb" influence from other words to be contextualized.

source: Binxu Wang's Transformers tutorial

Transformers



Early generative models: RNNs



Recurrent Neural Networks

- **Hidden variable dynamics:** $h_t = f_W(x_t, h_{t-1})$
- **Example:** $h_t = \mathrm{ReLU}(W_{hh}\, h_{t-1} + W_{xh}\, x_t)$
- **Prediction:** next token x_{t+1} is randomly sampled according to the probability

$$p_{h_t}(u) = \mathsf{Softmax}_u(W_{hp}\, h_t + W_{xp}\, x_t)$$

source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Generative models: general form

Next token predictors

- In general, we have a model that returns a probability distribution on the dictionary given the K last tokens of an input sequence $(u_1, \dots, u_{t-1}) \in \mathcal{D}^{(t-1)\times n}$:

$$\forall u \in \mathcal{D}, \quad p_{(u_{t-K},\dots,u_{t-1})}(u) = g_\theta((u_{t-K}, \dots, u_{t-1}))_u$$

Text generation

- We sample each token in the sequence iteratively given the K previous tokens.

Limitations

- If K is small, difficult to deal with **long-term dependencies**.
- **Sequential** by construction. Hard to parallelize.

Attention layers: the softmax family

Softmax probabilities

- **Idea:** Create differentiable **selection mechanisms** to identify valuable sequence elements.
- **Definition:** Let $x = x_1, \ldots, x_n$ be scores associated to n items, then

$$\text{Softmax}_i(x) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

- **Properties:** Discrete distribution $\text{Softmax}_i(x) \in (0, 1)$ and $\sum_{i=1}^{n} \text{Softmax}_i(x) = 1$.

Attention layers: the softmax family

Softmax probabilities

- **Idea:** Create differentiable **selection mechanisms** to identify valuable sequence elements.
- **Definition:** Let $x = x_1, \ldots, x_n$ be scores associated to n items, then

$$\text{Softmax}_i(x) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

- **Properties:** Discrete distribution $\text{Softmax}_i(x) \in (0, 1)$ and $\sum_{i=1}^{n} \text{Softmax}_i(x) = 1$.

Applications of Softmax

- **Cross-entropy:** We have $\ell_{CE}(y, y') = -\log(\text{Softmax}_{y'}(y))$.

Attention layers: the softmax family

Softmax probabilities

- **Idea:** Create differentiable **selection mechanisms** to identify valuable sequence elements.
- **Definition:** Let $x = x_1, \ldots, x_n$ be scores associated to n items, then

$$\text{Softmax}_i(x) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

- **Properties:** Discrete distribution $\text{Softmax}_i(x) \in (0, 1)$ and $\sum_{i=1}^{n} \text{Softmax}_i(x) = 1$.

Applications of Softmax

- **Cross-entropy:** We have $\ell_{CE}(y, y') = -\log(\text{Softmax}_{y'}(y))$.
- **Max:** We can return a "soft max" with $\sum_i \text{Softmax}_i(x)\, x_i$ or $\log(\sum_{j=1}^{n} e^{x_j})$.

Attention layers: the softmax family

Softmax probabilities

- **Idea:** Create differentiable **selection mechanisms** to identify valuable sequence elements.
- **Definition:** Let $x = x_1, \ldots, x_n$ be scores associated to n items, then

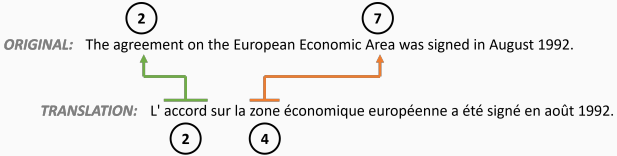
$$\text{Softmax}_i(x) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

- **Properties:** Discrete distribution $\text{Softmax}_i(x) \in (0, 1)$ and $\sum_{i=1}^{n} \text{Softmax}_i(x) = 1$.

Applications of Softmax

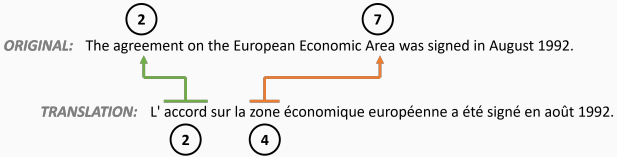
- **Cross-entropy:** We have $\ell_{CE}(y, y') = -\log(\text{Softmax}_{y'}(y))$.
- **Max:** We can return a "soft max" with $\sum_i \text{Softmax}_i(x)\, x_i$ or $\log(\sum_{j=1}^{n} e^{x_j})$.
- **(Differentiable) Argmax:** For items x_i having score s_i, we can approximate the argmax (item x_i with highest score s_i) by $\sum_i \text{Softmax}_i(s)\, x_i$.

Attention layers: first use in translation (Bahdanau et.al., 2015)



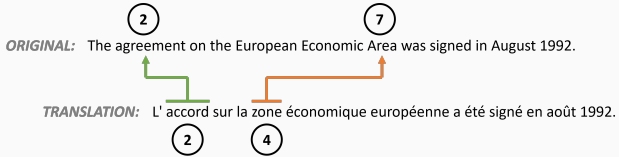
- **Idea:** Align the words between two different languages using attention.

Attention layers: first use in translation (Bahdanau et.al., 2015)



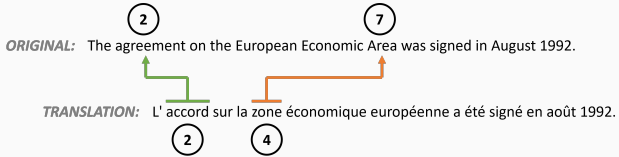
- **Idea:** Align the words between two different languages using attention.
- **Encoding:** For an input sentence (e.g. in English) $(u_1, \ldots, u_T)$, we use an LSTM to compute hidden vectors representing each word/token $(h_1, \ldots, h_T)$.

Attention layers: first use in translation (Bahdanau et.al., 2015)



- **Idea:** Align the words between two different languages using attention.
- **Encoding:** For an input sentence (e.g. in English) $(u_1, \ldots, u_T)$, we use an LSTM to compute hidden vectors representing each word/token $(h_1, \ldots, h_T)$.
- **Decoding:** We then compute recursively the hidden state of the translated sentence $s_t = f(s_{t-1}, y_{t-1}, c_t)$ where y_{t-1} is the previous token, and c_t is a context vector.

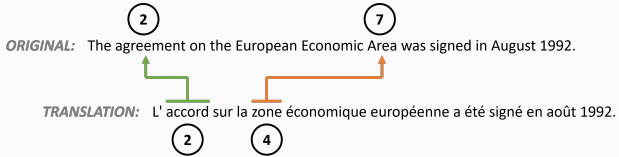
Attention layers: first use in translation (Bahdanau et.al., 2015)



- **Idea:** Align the words between two different languages using attention.
- **Encoding:** For an input sentence (e.g. in English) $(u_1, \ldots, u_T)$, we use an LSTM to compute hidden vectors representing each word/token $(h_1, \ldots, h_T)$.
- **Decoding:** We then compute recursively the hidden state of the translated sentence $s_t = f(s_{t-1}, y_{t-1}, c_t)$ where y_{t-1} is the previous token, and c_t is a context vector.
- **Context:** Using **attention**, we select the token element of the original sentence

$$c_t = \sum_{i=1}^{T} \text{Softmax}_i(a(s_{t-1}, y_{t-1}))h_i$$

Attention layers: first use in translation (Bahdanau et.al., 2015)

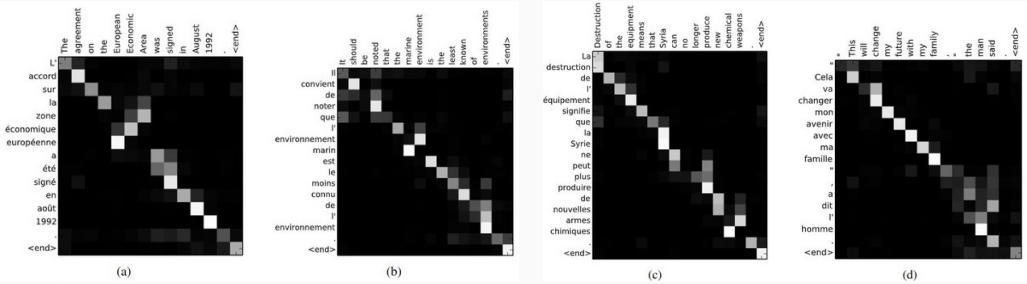


- **Idea:** Align the words between two different languages using attention.
- **Encoding:** For an input sentence (e.g. in English) $(u_1, \ldots, u_T)$, we use an LSTM to compute hidden vectors representing each word/token $(h_1, \ldots, h_T)$.
- **Decoding:** We then compute recursively the hidden state of the translated sentence $s_t = f(s_{t-1}, y_{t-1}, c_t)$ where y_{t-1} is the previous token, and c_t is a context vector.
- **Context:** Using **attention**, we select the token element of the original sentence

$$c_t = \sum_{i=1}^{T} \text{Softmax}_i(a(s_{t-1}, y_{t-1}))h_i$$

- **Prediction:** We sample according to $y_i \sim g(y_{i-1}, s_i, c_i)$.

Attention layers: first use in translation (Bahdanau et.al., 2015)



source: Neural Machine Translation by Jointly Learning to Align and Translate (Bahdanau et.al., 2015)

Attention layers in Transformers

Attention is all you need (Vaswani et.al., 2017)

- **Idea:** Select **values** by matching **keys** and **queries**.

Attention layers in Transformers

Attention is all you need (Vaswani et.al., 2017)

- **Idea:** Select **values** by matching **keys** and **queries**.
- **Example:** Return a video (value) for a search (query) matching the video's title (key).

Attention is all you need (Vaswani et.al., 2017)

- **Idea:** Select **values** by matching **keys** and **queries**.
- **Example:** Return a video (value) for a search (query) matching the video's title (key).
- **Definition:** For queries $Q \in \mathbb{R}^{k \times S}$, keys $K \in \mathbb{R}^{k \times T}$ and values $V \in \mathbb{R}^{d_{out} \times T}$, return, $\forall s \in [\![1, S]\!]$,

$$Y_s = \sum_{t=1}^{T} \text{Softmax}_t(\text{score}(Q_s, K))V_t$$

Attention is all you need (Vaswani et.al., 2017)

- **Idea:** Select **values** by matching **keys** and **queries**.
- **Example:** Return a video (value) for a search (query) matching the video's title (key).
- **Definition:** For queries $Q \in \mathbb{R}^{k \times S}$, keys $K \in \mathbb{R}^{k \times T}$ and values $V \in \mathbb{R}^{d_{out} \times T}$, return, $\forall s \in [\![1, S]\!]$,

$$Y_s = \sum_{t=1}^{T} \mathsf{Softmax}_t(\mathsf{score}(Q_s, K))V_t$$

- **Usual score:** Dot-product $\mathsf{score}(Q_s, K) = \frac{Q_s^\top K}{\sqrt{k}}$.

(Self-)attention layers in Transformers

Self-attention

- **Idea:** We keep keys, values and pairs in a single input tensor X.

Self-attention

- **Idea:** We keep keys, values and pairs in a single input tensor X.
- **Definition:** Let $Q = W_Q X$, $K = W_K X$ and $V = W_V X$, then

$$Y_s = \sum_{t=1}^{T} \mathsf{Softmax}_t \left(\frac{Q_s^\top K}{\sqrt{k}} \right) V_t$$

(Self-)attention layers in Transformers

Self-attention

- **Idea:** We keep keys, values and pairs in a single input tensor X.
- **Definition:** Let $Q = W_Q X$, $K = W_K X$ and $V = W_V X$, then

$$Y_s = \sum_{t=1}^{T} \text{Softmax}_t \left(\frac{Q_s^\top K}{\sqrt{k}} \right) V_t$$

- We start with an input tensor $X \in \mathbb{R}^{d_{in} \times T}$, and return an output tensor $Y \in \mathbb{R}^{d_{out} \times T}$ with a choice of weight parameters $W_Q \in \mathbb{R}^{k \times d_{in}}, W_K \in \mathbb{R}^{k \times d_{in}}, W_V \in \mathbb{R}^{d_{out} \times d_{in}}$.

(Self-)attention layers in Transformers

Self-attention

- **Idea:** We keep keys, values and pairs in a single input tensor X.
- **Definition:** Let $Q = W_Q X$, $K = W_K X$ and $V = W_V X$, then

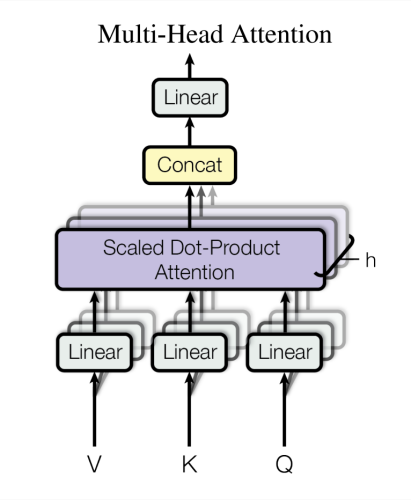
$$Y_s = \sum_{t=1}^{T} \text{Softmax}_t \left(\frac{Q_s^\top K}{\sqrt{k}} \right) V_t$$

- We start with an input tensor $X \in \mathbb{R}^{d_{in} \times T}$, and return an output tensor $Y \in \mathbb{R}^{d_{out} \times T}$ with a choice of weight parameters $W_Q \in \mathbb{R}^{k \times d_{in}}, W_K \in \mathbb{R}^{k \times d_{in}}, W_V \in \mathbb{R}^{d_{out} \times d_{in}}$.

Multi-head attention

- As for channels in convolution layers, we perform H **parallel** self-attention layers, and combine them with a linear layer. **Usual choice:** take $d_{in} = d_{out} \cdot H$.

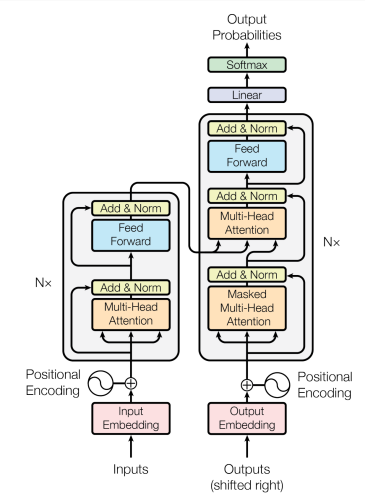
Multi-head attention



source: Attention is all you need (Vaswani et.al., 2017)

Dicussed in detail in these slides:
`lucasb.eyer.be/transformer`

The Transformer architecture (Vaswani et.al., 2017)



source: Attention is all you need (Vaswani et.al., 2017)

Dicussed in detail in these slides:
`lucasb.eyer.be/transformer`

Positional encoding

Limitations of the multi-head attention block

- **Time complexity:** Quadratic w.r.t. sequence length, $O(mT^2k)$ to generate m tokens.
- **Permutation-invariance:** All sequence elements are treated equally... **order is lost**!

Positional encoding

Limitations of the multi-head attention block

- **Time complexity:** Quadratic w.r.t. sequence length, $O(mT^2k)$ to generate m tokens.
- **Permutation-invariance:** All sequence elements are treated equally... **order is lost**!

Positional encoding

- **Idea:** We add the **position** t to each input token u_t... but in a more fancy way.
- **Implementation:** Let $v_t = u_t + p_t$, where

$$p_t = \left(\cos\left(\frac{t}{10000^{2i/k}}\right), \sin\left(\frac{t}{10000^{2i/k}}\right)\right)_{i \in [\![1, k/2]\!]}$$

- **Properties:** p_t uniquely defines t, but is better to encode **translations** and **periodicity**.

Layer normalization

Idea

- Same as batch normalization, but normalized **per layer** instead of per batch: Each token embedding is normalized across its feature components.
- Ensures that all elements in the sequence have approximately the same amplitude.

Layer normalization

Idea

- Same as batch normalization, but normalized **per layer** instead of per batch: Each token embedding is normalized across its feature components.
- Ensures that all elements in the sequence have approximately the same amplitude.

Definition

- If $(x_{i,j,k}) \in \mathbb{R}^{b \times L \times d}$ is a batch of b sequences of length L and feature dimension d, then the output is:

$$y_{i,j,k} = \frac{x_{i,j,k} - E_{i,j}}{\sqrt{V_{i,j} + \varepsilon}} \cdot \gamma_k + \beta_k$$

- where $E_{i,j} = \frac{1}{d} \sum_{k=1}^{d} x_{i,j,k}$ and $V_{i,j} = \frac{1}{d} \sum_{k=1}^{d} (x_{i,j,k} - E_{i,j})^2$
- γ and $\beta \in \mathbb{R}^d$ are learnable vectors.

The GPT architecture (Radford et.al., 2018)

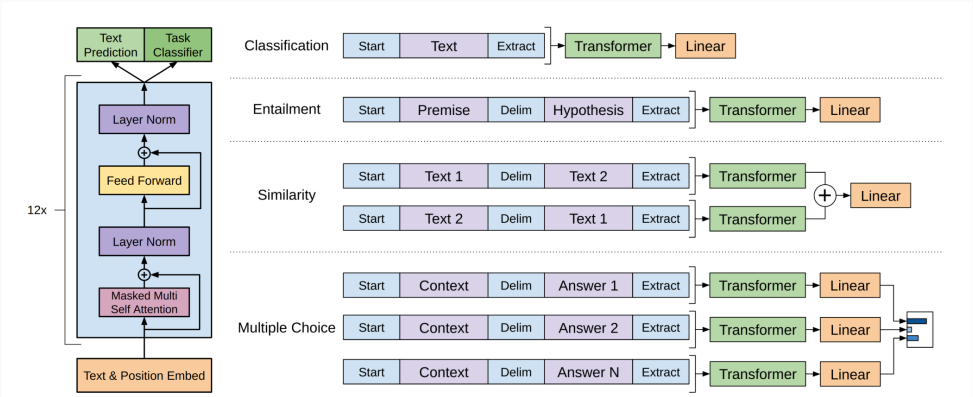
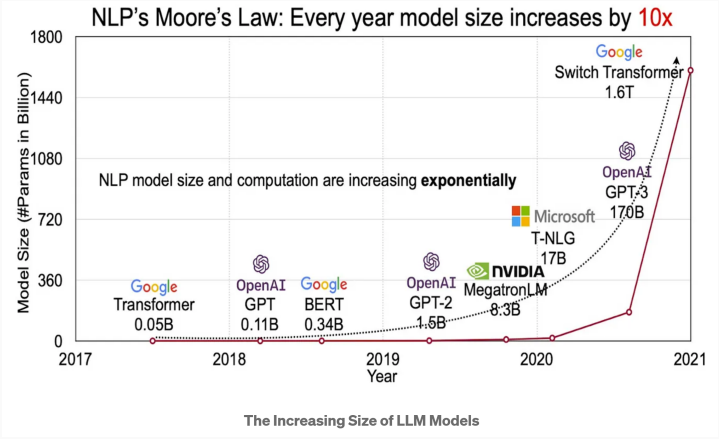


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

source: Improving Language Understanding by Generative Pre-Training (Radford et.al., 2018)

The LLM family: model size

Human brain: est. an average of **86B neurons** and **100T synapses**.



source: https://medium.com/@harishdatalab/unveiling-the-power-of-large-language-models-llms-e235c4eba8a9

The LLM family: recent models

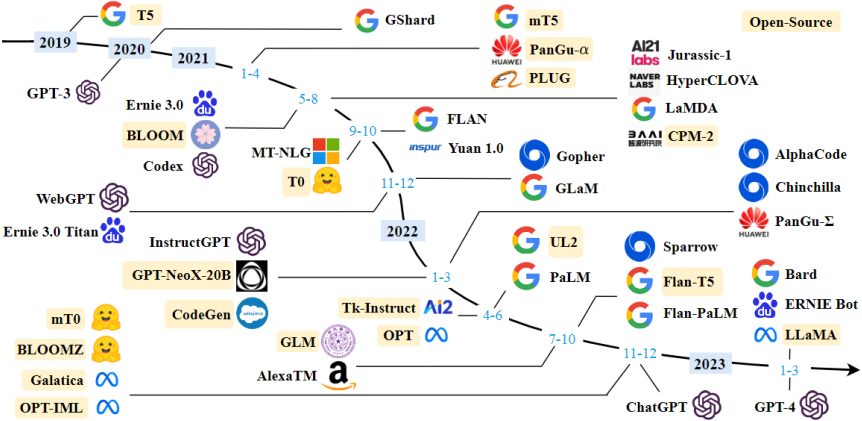


Fig. 1. A timeline of existing large language models (having a size larger than 10B) in recent years. We mark the open-source LLMs in yellow color.

source: https://wandb.ai/vincenttu/blog_posts/reports/A-Survey-of-Large-Language-Models--VmlldzoxOTY2MDM1

The LLM family: architecture details

Model	Category	Size	Normalization	PE	Activation	Bias	#L	#H	d_{model}	MCL
GPT3 [55]	Casual decoder	175B	Pre Layer Norm	Learned	GeLU	✓	96	96	12288	2048
PanGU- α [73]	Casual decoder	207B	Pre Layer Norm	Learned	GeLU	✓	64	128	16384	1024
OPT [79]	Casual decoder	175B	Pre Layer Norm	Learned	ReLU	✓	96	96	12288	2048
PaLM [56]	Casual decoder	540B	Pre Layer Norm	RoPE	SwiGLU	×	118	48	18432	2048
BLOOM [66]	Casual decoder	176B	Pre Layer Norm	ALiBi	GeLU	✓	70	112	14336	2048
MT-NLG [90]	Casual decoder	530B	-	-	-	-	105	128	20480	2048
Gopher [59]	Casual decoder	280B	Pre RMS Norm	Relative	-	-	80	128	16384	-
Chinchilla [34]	Casual decoder	70B	Pre RMS Norm	Relative	-	-	80	64	8192	-
Galactica [33]	Casual decoder	120B	Pre Layer Norm	Learned	GeLU	×	96	80	10240	2048
LaMDA [85]	Casual decoder	137B	-	Relative	GeGLU	-	64	128	8192	-
Jurassic-1 [89]	Casual decoder	178B	Pre Layer Norm	Learned	GeLU	✓	76	96	13824	2048
LLaMA [57]	Casual decoder	65B	Pre RMS Norm	RoPE	SwiGLU	✓	80	64	8192	2048
GLM-130B [80]	Prefix decoder	130B	Post Deep Norm	RoPE	GeGLU	✓	70	96	12288	2048
T5 [71]	Encoder-decoder	11B	Pre RMS Norm	Relative	ReLU	×	24	128	1024	-

source: https://wandb.ai/vincenttu/blog_posts/reports/A-Survey-of-Large-Language-Models--VmlldzozOTY2MDM1

To go further

What we didn't discuss

- **Masking:** to preserve causality.

To go further

What we didn't discuss

- **Masking:** to preserve causality.
- **Alignment Tuning:** Reinforcement Learning with Human Feedback (RHLF).

To go further

What we didn't discuss

- **Masking:** to preserve causality.
- **Alignment Tuning:** Reinforcement Learning with Human Feedback (RHLF).
- **Fast fine-tuning:** efficient fine-tuning with low rank approximations (LoRA and QLoRA).

To go further

What we didn't discuss

- **Masking:** to preserve causality.
- **Alignment Tuning:** Reinforcement Learning with Human Feedback (RHLF).
- **Fast fine-tuning:** efficient fine-tuning with low rank approximations (LoRA and QLoRA).
- **Improving model scalability:** Mixture of Experts (MoE) and Mamba.

To go further

What we didn't discuss

- **Masking:** to preserve causality.
- **Alignment Tuning:** Reinforcement Learning with Human Feedback (RHLF).
- **Fast fine-tuning:** efficient fine-tuning with low rank approximations (LoRA and QLoRA).
- **Improving model scalability:** Mixture of Experts (MoE) and Mamba.
- **Training details of LLMs:** a nice survey on traning hyper-param. and technical details:
  `https://wandb.ai/vincenttu/blog_posts/reports/A-Survey-of-Large-Language-Models--VmlldzozOTY2MDM1`

To go further

What we didn't discuss

- **Masking:** to preserve causality.
- **Alignment Tuning:** Reinforcement Learning with Human Feedback (RHLF).
- **Fast fine-tuning:** efficient fine-tuning with low rank approximations (LoRA and QLoRA).
- **Improving model scalability:** Mixture of Experts (MoE) and Mamba.
- **Training details of LLMs:** a nice survey on traning hyper-param. and technical details:
  `https://wandb.ai/vincenttu/blog_posts/reports/A-Survey-of-Large-Language-Models--VmlldzozOTY2MDM1`
- **Model performance and evaluation:** many benchmark tasks, but can overfit. Generation quality:
  `https://huggingface.co/spaces/lmsys/chatbot-arena-leaderboard`

Recap

- **Transformers** = **Attention** (+ LayerNorm + Residuals + MLPs + Positional encoding).
- Text generation using a simple **next token prediction** approach.
- Encoder-Decoder architecture for translation, **only Decoder** for generation.
- Attention is a **differentiable selection mechanism**.
- Large number of recent models, ranging between **1B and 1T parameters**.

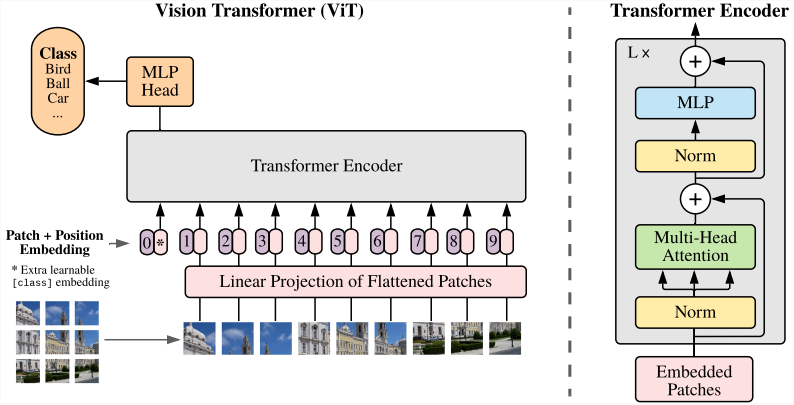
Transformers are the state-of-the-art architecture for NLP tasks...

Recap

- **Transformers** = **Attention** (+ LayerNorm + Residuals + MLPs + Positional encoding).
- Text generation using a simple **next token prediction** approach.
- Encoder-Decoder architecture for translation, **only Decoder** for generation.
- Attention is a **differentiable selection mechanism**.
- Large number of recent models, ranging between **1B and 1T parameters**.

Transformers are the state-of-the-art architecture for NLP tasks...
but also for vision, audio, and multimodal tasks!

Vision Transformers



- Images are split into patches (e.g. 14×14 patches of size 16×16 pixels) that are linearly embedded, combined with (learnable) positional embeddings, and fed to a standard Transformer encoder.

source: An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale (Dosovitskiy et.al., 2021)