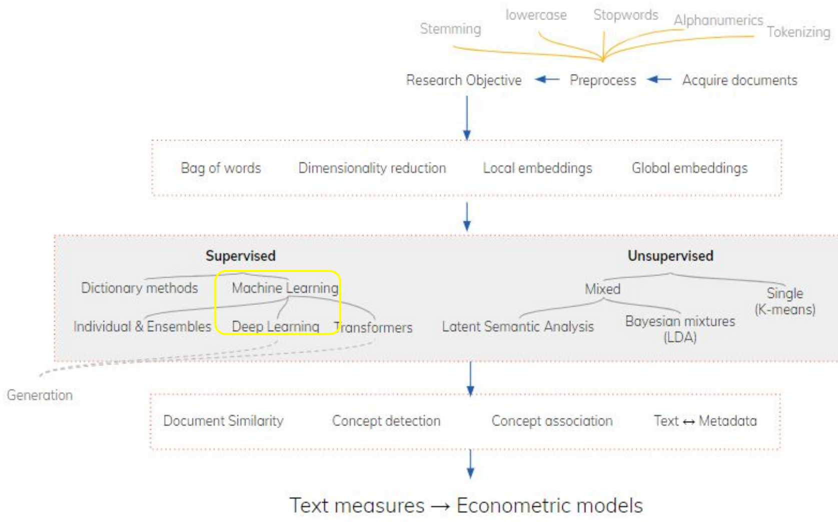
Innovation Growth Lab by nesta

Natural Language Processing in Economics

Advanced Supervised Learning



Neural Networks

Computation Graphs

- Recall the logistic regression model presented in the previous lecture

$$\hat{\mathbf{w}}_{ML} = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{N} \underbrace{-y_i \log \hat{y}_i - (1 - y_i) \log(1 - \hat{y}_i)}_{\text{Binary Cross Entropy Loss } \mathcal{L}(\hat{y}_i, y_i)}$$

$$\text{with} \quad \hat{y} = f_{\mathbf{w}}(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x}) \quad \text{and} \quad \sigma(x) = \frac{1}{1 + e^{-x}}$$

- Minimisation of a **non-linear objective** requires the calculation of gradients $\nabla_{\mathbf{w}} \mathcal{L}(\hat{y}_i, y_i) = (\hat{y}_i - y_i)\mathbf{x}_i$

- This is often a difficult problem to visualise and compute → **computation graphs**

Computation Graphs

Idea

- **Decompose** complex computations into a sequence of atomic assignments, called a **computation graph**
- The **forward pass** takes a training point (**x**, **y**) as input and computes a loss, ie.

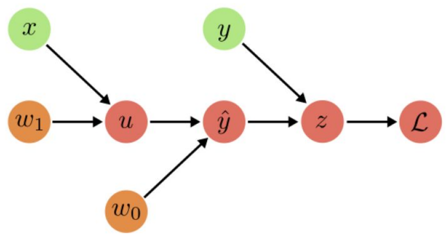
$$\mathcal{L} = -\log p_{model}(y|\mathbf{x}, \mathbf{w})$$

- The relevant gradients can be computed using a **backward pass**, which are efficient due to the use of dynamic programming, ie. storing and reusing intermediate results
- This decomposition and reuse of computation is key to the success of the backpropagation algorithm, the primary workhorse of deep learning

Three kind of nodes

- **Green**: Input nodes
- **Orange**: Parameter nodes
- **Red**: Compute nodes

$$
\begin{aligned}
(1) \quad & u = w_1 x \\
(2) \quad & \hat{y} = w_0 + u \\
(3) \quad & z = \hat{y} - y \\
(4) \quad & \mathcal{L} = z^2
\end{aligned}
$$



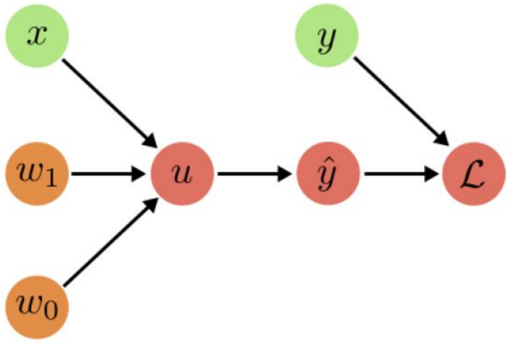
Computation Graphs

Example - Logistic Regression

$$(1) \quad u = w_0 + w_1 x$$

$$(2) \quad \hat{y} = \sigma(u)$$

$$(3) \quad \mathcal{L} = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$



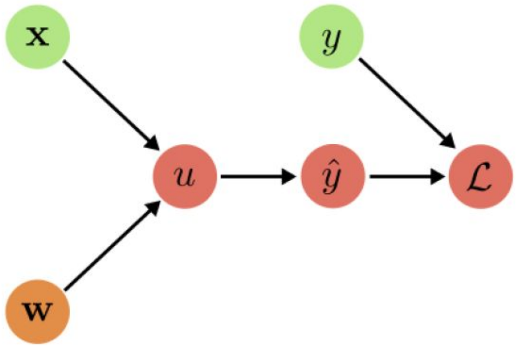
Computation Graphs

Example - Logistic Regression

$$(1) \quad u = \mathbf{w}^\top \mathbf{x}$$

$$(2) \quad \hat{y} = \sigma(u)$$

$$(3) \quad \mathcal{L} = -y \log \hat{y} - (1-y)\log(1-\hat{y})$$



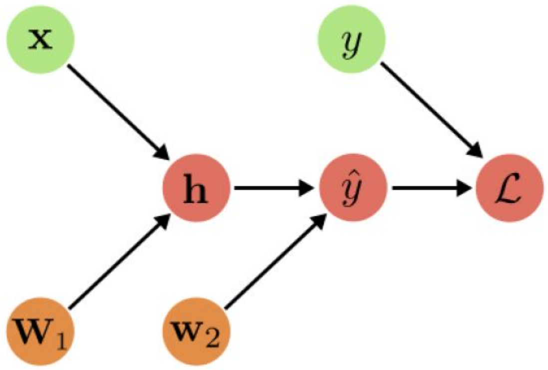
Computation Graphs

Example - Multilayer perceptron

(1) $\mathbf{h} = \sigma(\mathbf{W}_1^\top \mathbf{x})$

(2) $\hat{y} = \sigma(\mathbf{w}_2^\top \mathbf{h})$

(3) $\mathcal{L} = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$



Computation Graphs

$$\frac{\mathrm{d}}{\mathrm{d}x} f(g(x)) = \frac{\mathrm{d}f}{\mathrm{d}g}\frac{\mathrm{d}g}{\mathrm{d}x}$$

$$\frac{\mathrm{d}}{\mathrm{d}x} f(g_1(x), \ldots, g_M(x)) = \sum_{i=1}^{M} \frac{\partial f}{\partial g_i}\frac{\mathrm{d}g_i}{\mathrm{d}x}$$

Backpropagation

- We optimise model parameters **w** by using gradient descent with respect to the loss function

$$\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{y}, \mathbf{X}, \mathbf{w}) = \nabla_{\mathbf{w}} \sum_{i=1}^{N} \mathcal{L}(y_i, \mathbf{x}_i, \mathbf{w}) = \sum_{i=1}^{N} \nabla_{\mathbf{w}} \mathcal{L}(y_i, \mathbf{x}_i, \mathbf{w})$$

A simple example of **forward pass** followed by the **backward pass**

Forward Pass:

$$(1) \quad y = x^2$$

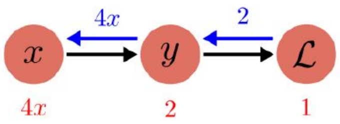
$$(2) \quad \mathcal{L} = 2y$$

Loss: $\mathcal{L} = 2x^2$

Backward Pass:

$$(2) \quad \frac{\partial \mathcal{L}}{\partial y} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}}\frac{\partial \mathcal{L}}{\partial y} = 2$$

$$(1) \quad \frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial x} = \frac{\partial \mathcal{L}}{\partial y} 2x$$



- **Red:** back-propagated gradients
- **Blue:** local gradients

Computation Graphs

$$\frac{\mathrm{d}}{\mathrm{d}x} f(g(x)) = \frac{\mathrm{d}f}{\mathrm{d}g}\frac{\mathrm{d}g}{\mathrm{d}x} \qquad \frac{\mathrm{d}}{\mathrm{d}x} f(g_1(x), \ldots, g_M(x)) = \sum_{i=1}^{M} \frac{\partial f}{\partial g_i}\frac{\mathrm{d}g_i}{\mathrm{d}x}$$

Backpropagation

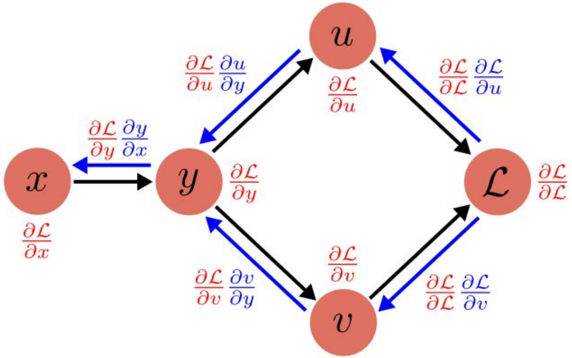
Forward Pass:

$$\begin{aligned}
&(1) \quad y = y(x) \\
&(2) \quad u = u(y) \\
&(2) \quad v = v(y) \\
&(3) \quad \mathcal{L} = \mathcal{L}(u, v)
\end{aligned}$$

Backward Pass:

$$\begin{aligned}
&(3) \quad \frac{\partial \mathcal{L}}{\partial u} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}}\frac{\partial \mathcal{L}}{\partial u} = \frac{\partial \mathcal{L}}{\partial u} \\
&(3) \quad \frac{\partial \mathcal{L}}{\partial v} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}}\frac{\partial \mathcal{L}}{\partial v} = \frac{\partial \mathcal{L}}{\partial v} \\
&(2) \quad \frac{\partial \mathcal{L}}{\partial y} = \frac{\partial \mathcal{L}}{\partial u}\frac{\partial u}{\partial y} + \frac{\partial \mathcal{L}}{\partial v}\frac{\partial v}{\partial y} \\
&(1) \quad \frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial x}
\end{aligned}$$

Loss: $\mathcal{L}(\, u(y(x)),\ v(y(x)) \,)$



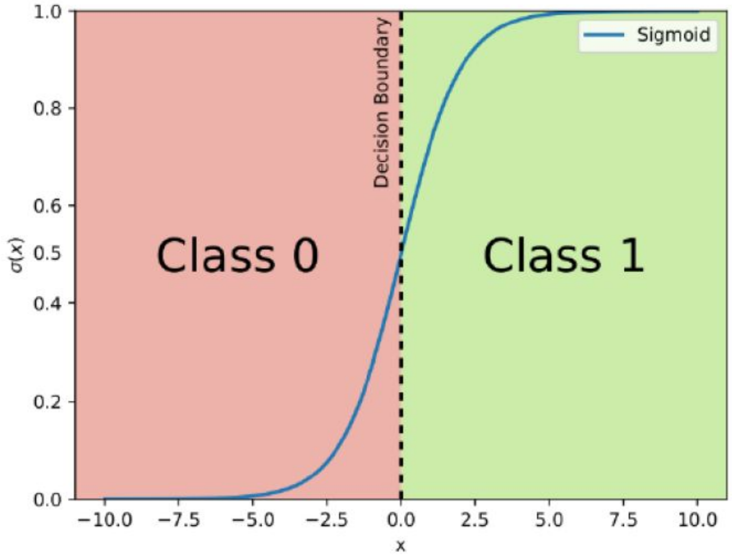
$$\frac{\mathrm{d}}{\mathrm{d}y}\mathcal{L}(u(y), v(y)) = \frac{\partial \mathcal{L}}{\partial u}\frac{\mathrm{d}u}{\mathrm{d}y} + \frac{\partial \mathcal{L}}{\partial v}\frac{\mathrm{d}v}{\mathrm{d}y}$$

All incoming gradients must be **summed** up!

A Logistic Regression is a Neural Network

$$\hat{y} = \sigma(\mathbf{w}^T\mathbf{x} + w_0) \quad \text{with } \sigma(x) = \frac{1}{1+\exp -x}$$

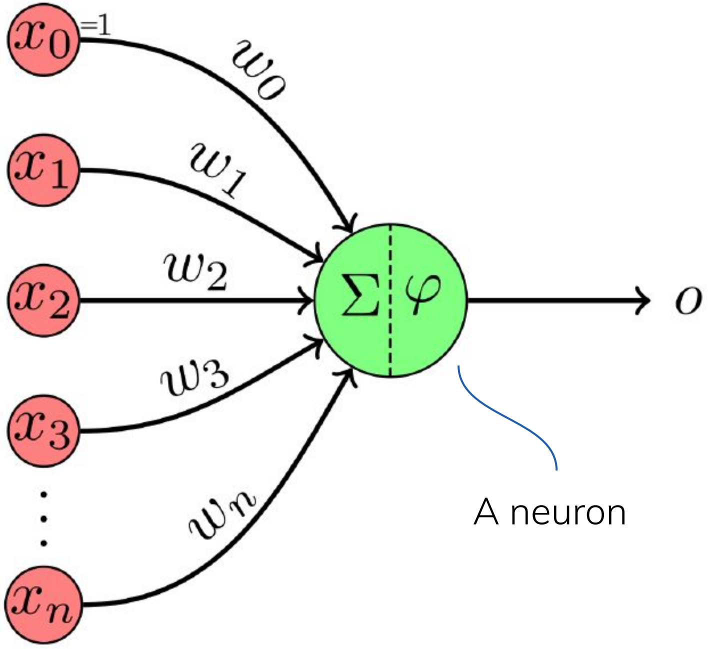
- Let $\mathbf{x} \in \mathbf{R}^2$
- Decision boundary: $\mathbf{w}^T\mathbf{x} + w_0 = 0$
- Decide for Class 0 $\leftrightarrow \mathbf{w}^T\mathbf{x} < -w_0$
- Decide for Class 1 $\leftrightarrow \mathbf{w}^T\mathbf{x} > -w_0$



The Perceptron

$$o = \sigma(w_0 x_0 + w_1 x_1 + w_2 x_2 \ldots) = \sigma(\sum w_i x_i) = \sigma(\mathbf{w}^T \mathbf{x})$$

- Let $\mathbf{x} \in \mathbf{R}^n$
- Decision boundary: $\sigma(\mathbf{w}^T \mathbf{x}) = 0.5$
- Binary Cross Entropy Loss
- Backpropagation
- Stochastic Gradient Descent



A neuron

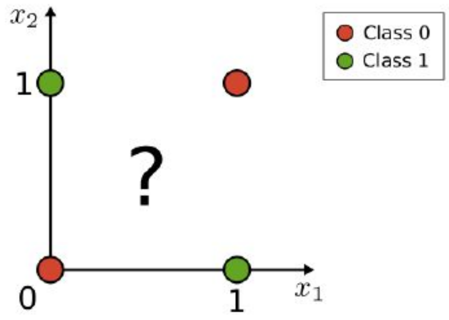
The XOR problem

Linear Classifier:

Class 1 $\Leftrightarrow \mathbf{w}^\top \mathbf{x} > -w_0$

$$\underbrace{\begin{pmatrix} ? & ? \end{pmatrix}}_{\mathbf{w}^\top} \underbrace{\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}}_{\mathbf{x}} > \underbrace{?}_{-w_0}$$

x_1	x_2	XOR(x_1,x_2)
0	0	0
0	1	1
1	0	1
1	1	0

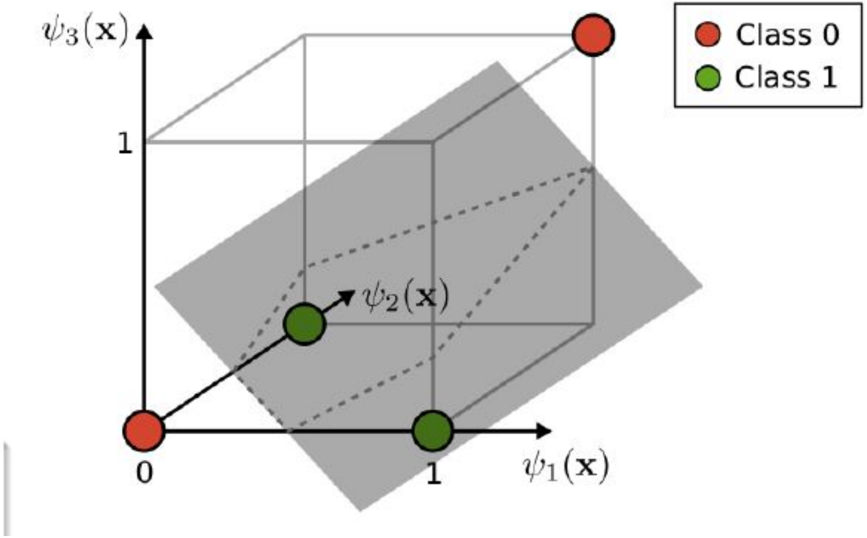


- Minsky & Papert’s 1969 book on the limitations of perceptrons
- The first AI winter

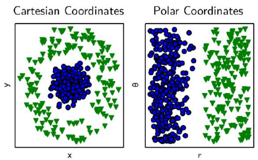
The XOR solution

Linear classifier with non-linear features ψ:

$$\mathbf{w}^\top \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{pmatrix}}_{\psi(\mathbf{x})} > -w_0$$



- Non-linear features allow linear classifier to solve non-linear classification problems
- Analogous to using kernel tricks in regressions or SVMs



Universal Approximation Theorem

Theorem (Cybenko, 1989; Hornik, 1991)

A feedforward neural network with a single hidden layer containing a finite number of neurons can approximate any continuous function on a compact subset of **R**n

- This result is agnostic about the choice of activation function
- The result does not imply the approximation (weights) can be learned

Multilayer Perceptrons

- MLPs are **feedforward** neural networks (no feedback connections)

- They compose several nonlinear functions $\mathbf{f}(\mathbf{x}) = \hat{\mathbf{y}}(\mathbf{h}_3(\mathbf{h}_2(\mathbf{h}_1(\mathbf{x}))))$ where **h** are hidden layers and **y** is the output layer

- The data specifies only the behavior of the output layer (thus the name **hidden**)

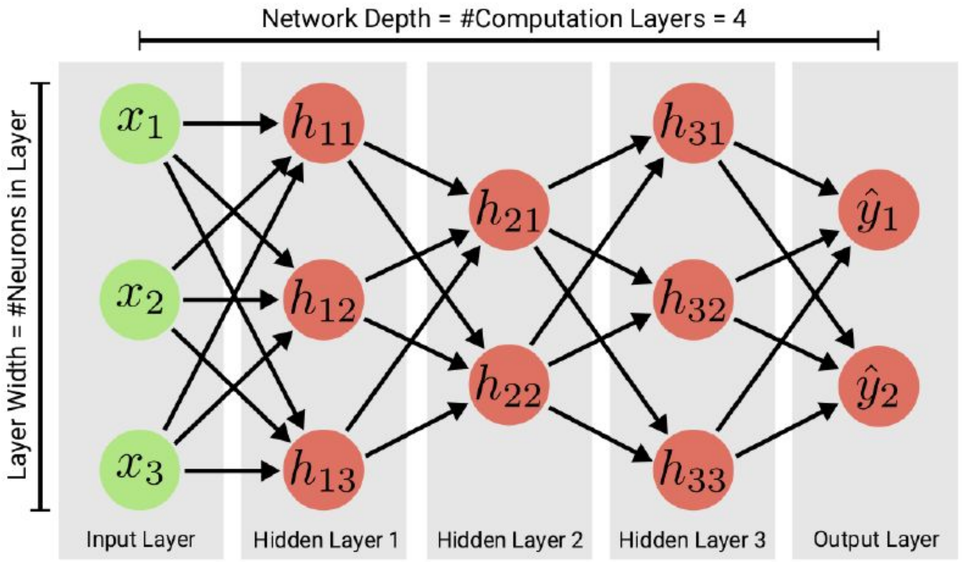
- Each layer **i** comprises multiple neurons **j** which are implemented as **affine transformations** $(\mathbf{a}^\top \mathbf{x} + \mathbf{b})$ followed by nonlinear activation functions **g**

$$h_{ij} = g(\mathbf{a}_{ij}^\top \mathbf{h}_{i-1} + \mathbf{b}_{ij})$$

- Each neuron in each layer is fully connected to all neurons of the previous layer

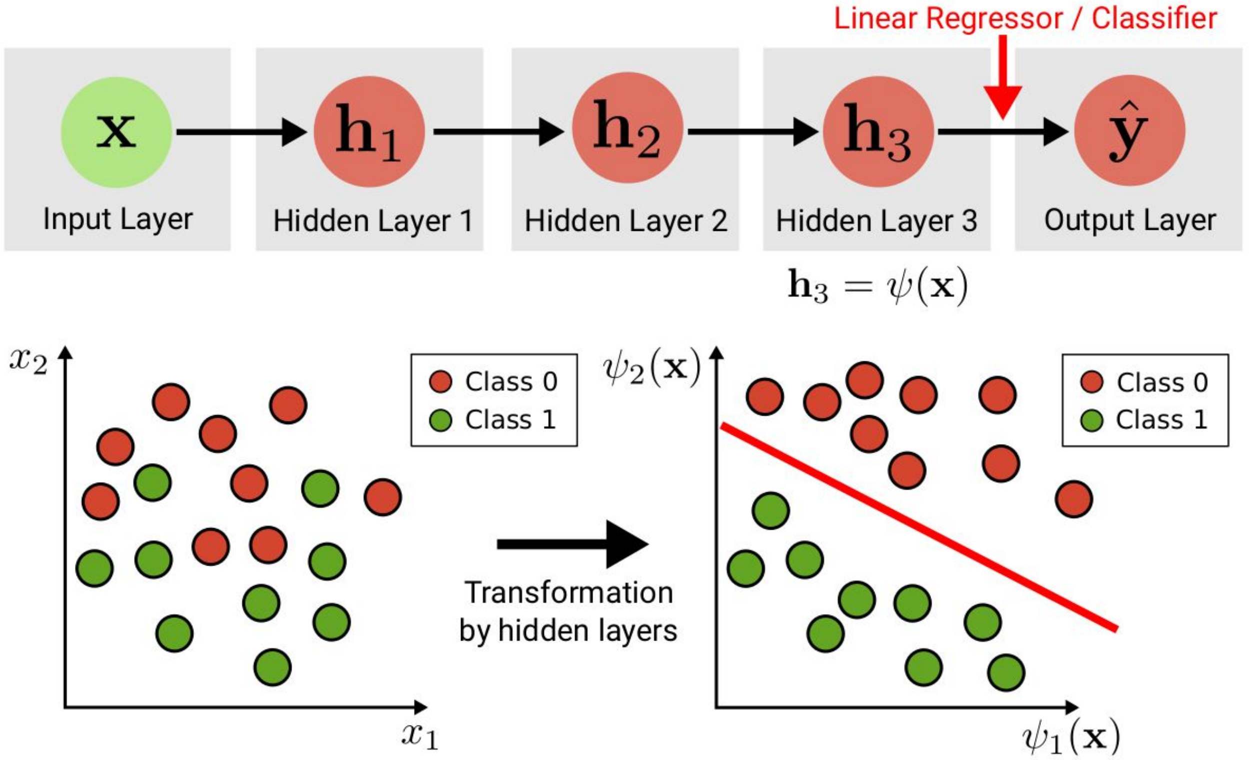
- The overall length of the chain is the **depth** of the model → Deep Learning

Multilayer Perceptrons

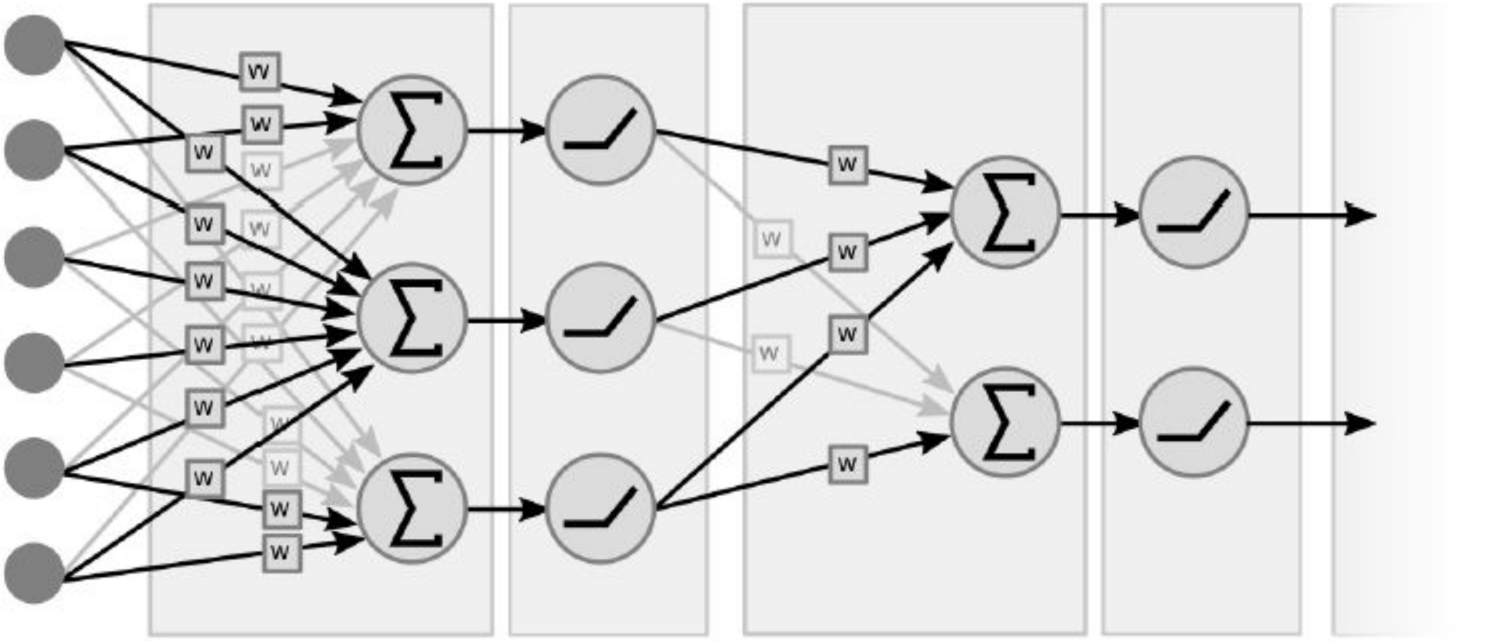


- Neurons are grouped into layers, each neuron is **fully connected** to all previous ones
- **Hidden layer h** has an activation function **g** and weights **A**, **b**

Multilayer Perceptrons - Feature learning



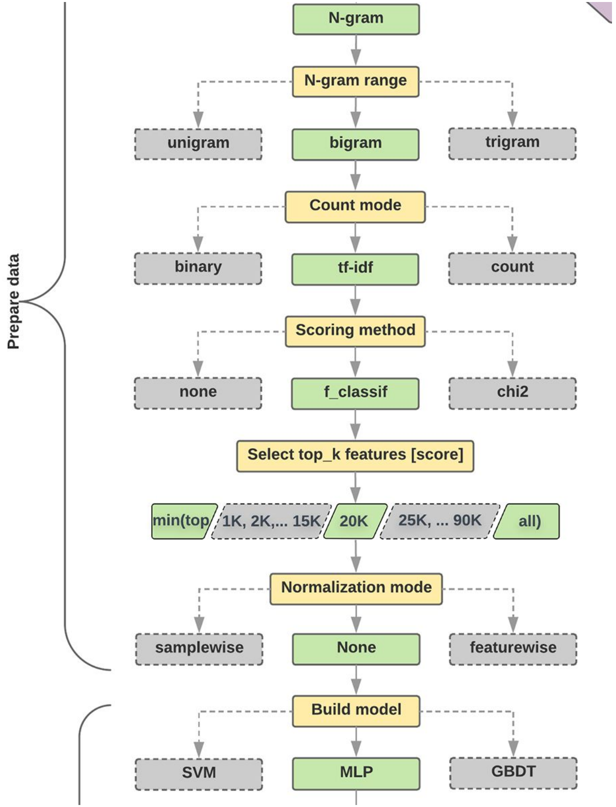
Multilayer Perceptrons

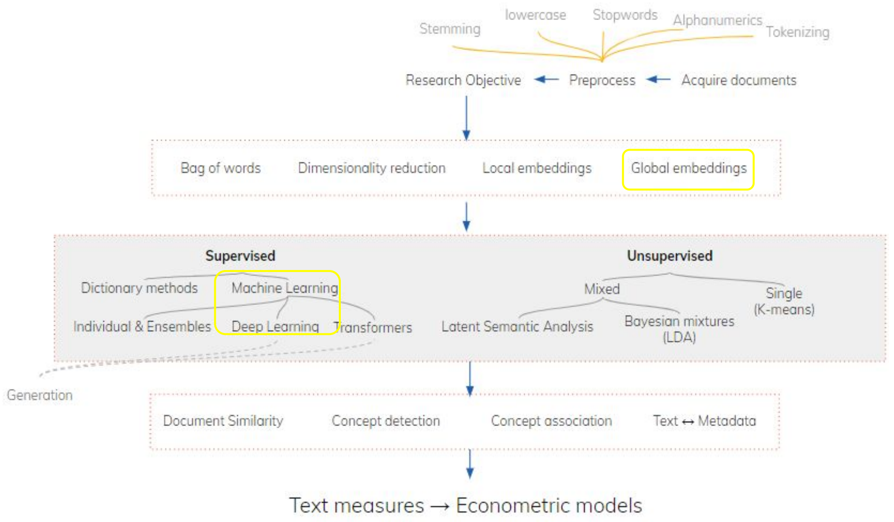


Google Developers Guide for Text Classification

- With relative few, longer documents, use an MLP
- The inputs **x** are **tf-idf weighted bigrams**
    - Select 20,000 features using SML
- Employ an MLP with two hidden layers

A MLP is one of the models tried by the Peterson and Spirling (2018) paper we saw on predicting polarisation in the UK parliament





Sequential Models

Classical IMDb classification problem

- Bag of words models can’t capture the importance of “don’t love” or “nothing don’t love”, even with many hidden layers

- N-grams have a large feature space and don’t share information across similar words and n-grams

Not addressed: Deep CBOW, Autoencoders, Convolutional networks

Sequence data

The deep learning breakthrough on NLP → move from bag-of-many representations to sequences

- Rather than inputting counts over n-grams, these models take as input a sequence of tokens

Note as well that feedforward MLP handle fix-length data, but many applications have variable lengths

$$\mathbf{x}_1 = (\mathbf{x}_{1,1}, \mathbf{x}_{1,2}, \mathbf{x}_{1,3}, \mathbf{x}_{1,4}, \mathbf{x}_{1,5})$$
$$\mathbf{x}_2 = (\mathbf{x}_{2,1}, \mathbf{x}_{2,2}, \mathbf{x}_{2,3}, \mathbf{x}_{2,4}, \mathbf{x}_{2,5})$$
$$\mathbf{x}_3 = (\mathbf{x}_{3,1}, \mathbf{x}_{3,2}, \mathbf{x}_{3,3}, \mathbf{x}_{3,4}, \mathbf{x}_{3,5})$$

VS

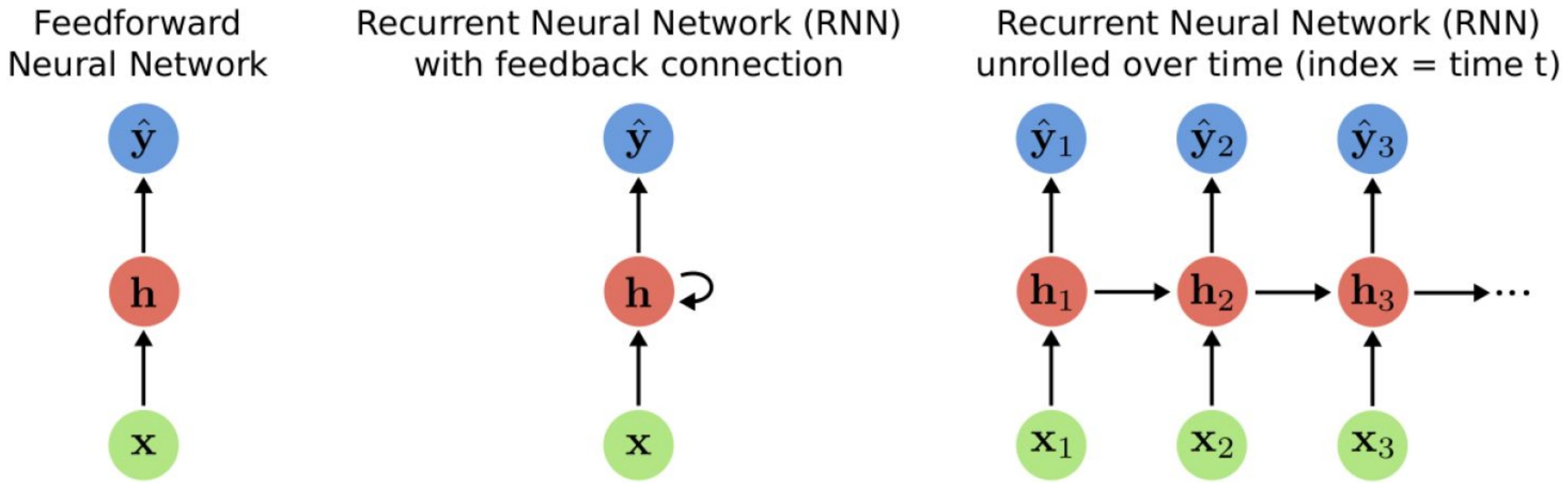
$$\mathbf{x}_1 = (\mathbf{x}_{1,1}, \mathbf{x}_{1,2}, \mathbf{x}_{1,3}, \mathbf{x}_{1,4}, \mathbf{x}_{1,5})$$
$$\mathbf{x}_2 = (\mathbf{x}_{2,1}, \mathbf{x}_{2,2}, \mathbf{x}_{2,3})$$
$$\mathbf{x}_3 = (\mathbf{x}_{3,1}, \mathbf{x}_{3,2}, \mathbf{x}_{3,3}, \mathbf{x}_{3,4})$$

Sentence sentiment classification **Sound phenomene Recognition** **Filmed activities**

Sub-par solutions

- Zero-pad up to length of longest sequence
- Iterate a single layer over each value

Recurrent Neural Network

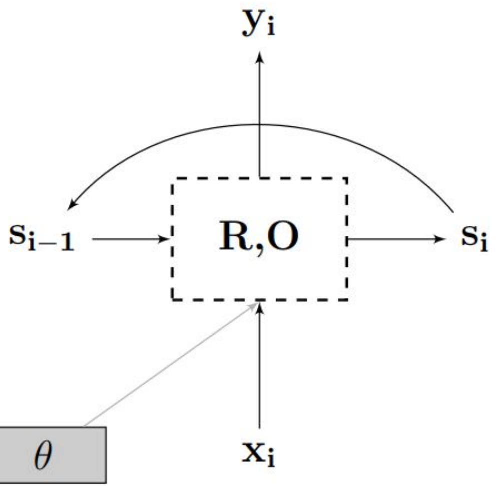


Core idea

- Update hidden state **h** based on a new input and the previous hidden state using the same parameters at each time step → **encodes** sequences into vectors, & **decodes** vectors into sequences
- Allows for processing **sequences of variable length**, and stores long-term dependencies

Recurrent Neural Network Architecture

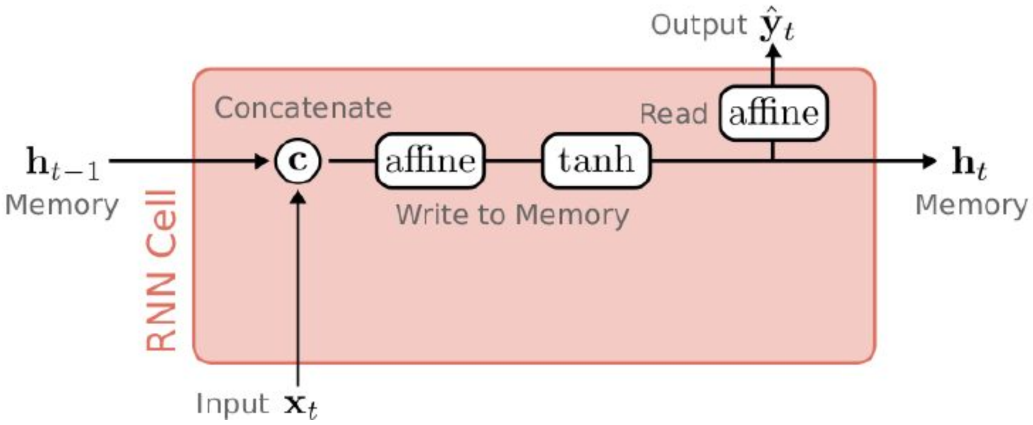
- At each step **t**:
  - A recursion function $R(s_{t-1}, x_t; \theta_R)$ computes the state vector $\boldsymbol{s}_t$ given current word $\boldsymbol{x}_t$ and previous state $\boldsymbol{s}_{t-1}$
  - An output function $O(s_t; \theta_O)$ computes the state vector **y** (to be compared to the outcome variable in the dataset.



$$\hat{\boldsymbol{y}}_t = O(\boldsymbol{s}_t, \theta_O)$$
$$\boldsymbol{s}_t = R(\boldsymbol{s}_{t-1}, \boldsymbol{x}_t, \theta_R)$$

The parameters of those functions $\theta = (\theta_R, \theta_O)$ are learned during training

Recurrent Neural Network Architecture

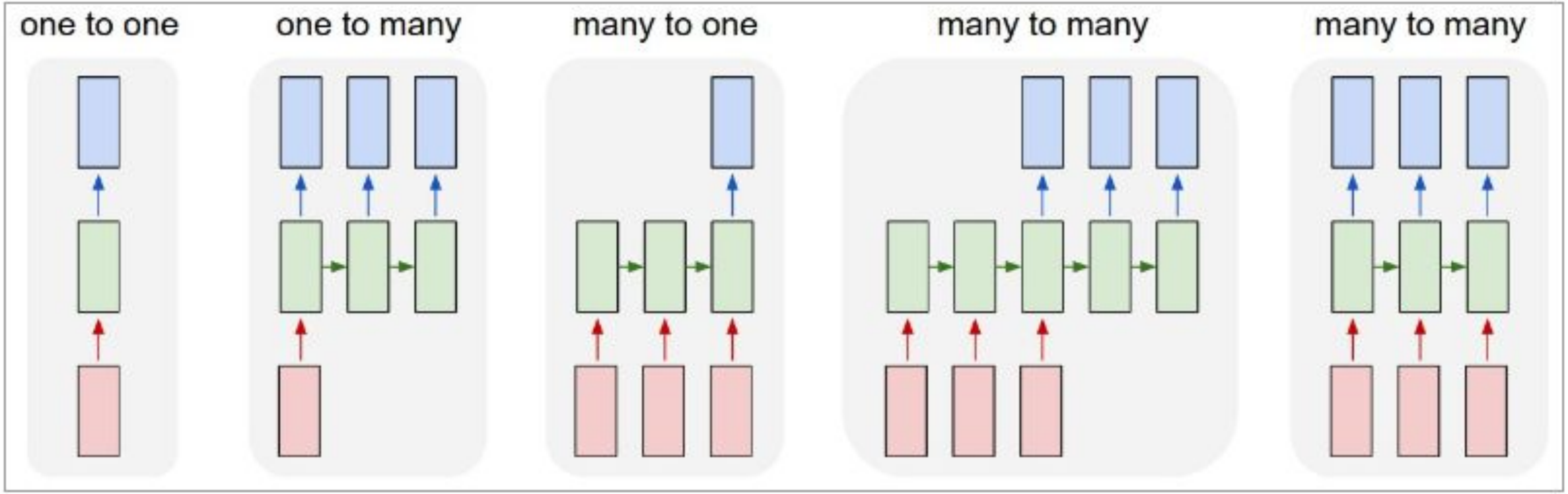


$$\mathbf{h}_t = f_h(\mathbf{h}_{t-1}, x_t) = \tanh\left(\mathbf{A}^{(h)}\mathbf{h}_{t-1} + \mathbf{A}^{(x)}x_t + b\right)$$

$$\hat{\mathbf{y}}_t = f_y(\mathbf{h}_t) = \mathbf{A}^{(y)}\mathbf{h}_t$$

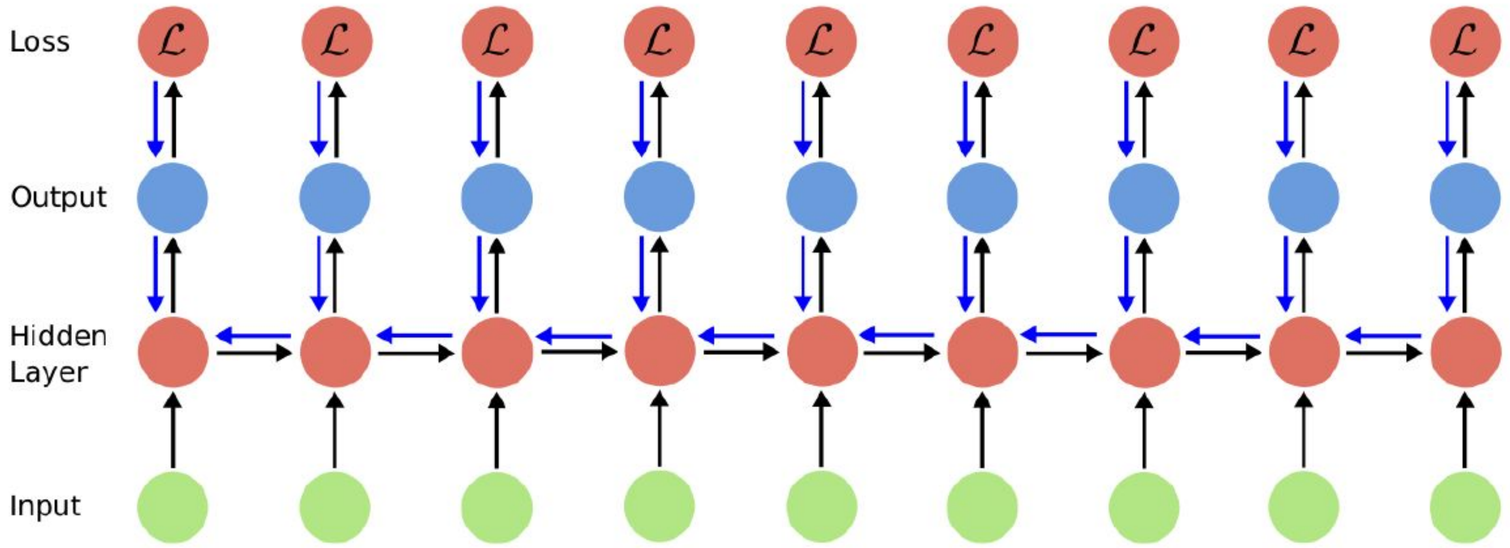
Parameters $\mathbf{A}^{(h)}$, $\mathbf{A}^{(x)}$, $\mathbf{A}^{(y)}$ and b are constant over time.

Recurrent Neural Network Architecture



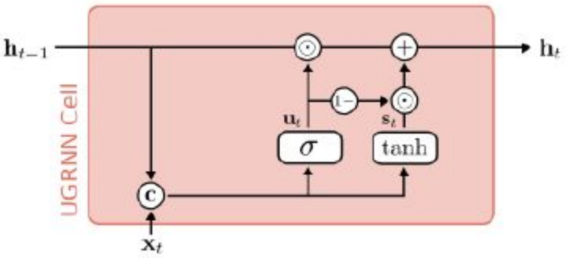
- **One to one**: image to label
- **One to many**: image to sentence
- **Many to one:** sentence to label
- **Many to many**: sentence to sentence

Training a Recurrent Neural Network



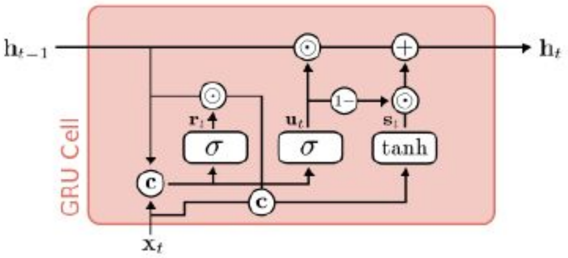
Gated Recurrent Neural Network

UGRNN



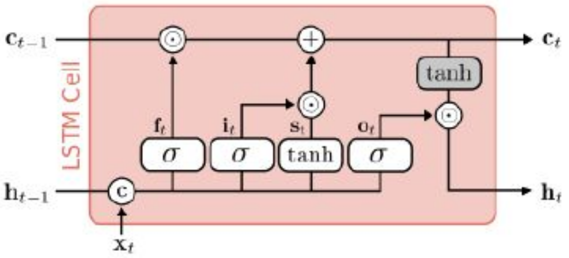
Collins, 2017

GRU



Cho, 2014

LSTM



Hochreiter, 1997

Shared attribute: **gates** for filtering information

- **UGRNN**: A sigmoid gate controls how much new information is sued
- **GRU**: A reset gate controls the relevance of the previous state on **f**
- **LSTM**: An additional cell **c** carries long term information

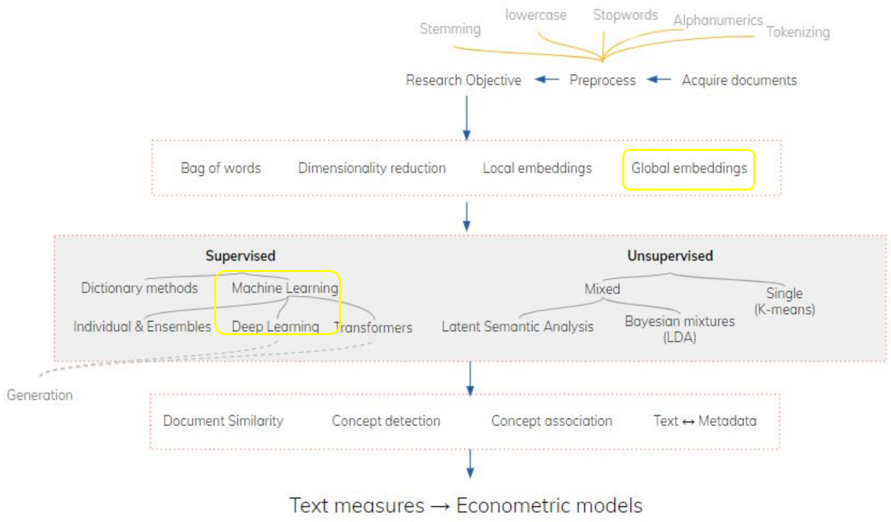
Gated Recurrent Neural Network

Caveats

- **Sequence-to-sequence mapping**: Traditional RNNs are designed for synchronized input-output mapping. In many applications, such as machine translation or text summarising, often we deal with variable-length outputs

- **Information bottleneck**: in traditional RNNs, the hidden state must encode all the relevant information from the input sequence into the fixed-length vector, leading to memory leakage and forgetting long-term dependencies

- **Bidirectional context**: RNNs process sequences in a forward-only manner, which means they have limited access to future context when making predictions. This can be problematic for tasks where future context matters

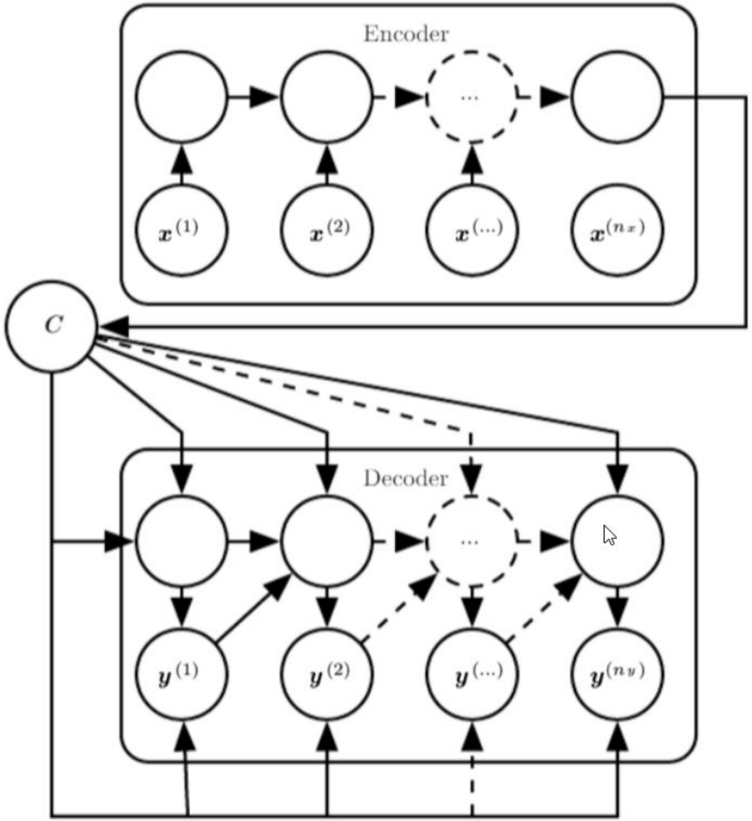


Encoder-Decoder models

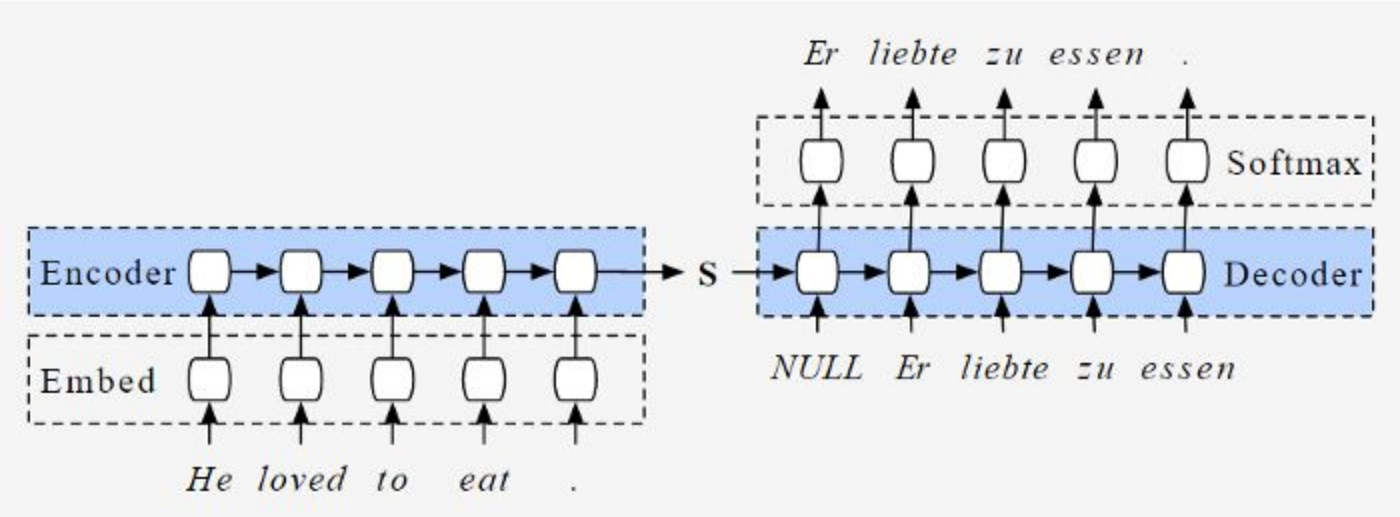
Encoder-Decoders: Machine Translation

- The **seq2seq** model:
  - $x^{(i)}$; i^{th} word input
  - $y^{(i)}$; i^{th} word output
  - c; context vector

- Applications
  - Question Answering
  - Machine translation
  - Machine conversation



Encoder-Decoders: Machine Translation



Problem: The **s** vector aggregates all contextual relations within a paragraph, and does not discriminate among within-string relations

Solution: The **Attention** mechanism

Attention mechanism

- Probabilistic retrieval of a **value** v_i for a **query** q_m based on a **key** k_i

$$\text{similarity}(q_m, k_i) = q_m^\top k_i$$

- Similarities are transformed into probabilities through a softmax

$$a_{m,i} = \frac{\exp\left(q_m^\top k_i\right)}{\sum_j \exp\left(q_m^\top k_j\right)}$$

- The output is a linear combination of values or vectors v_i

$$\text{attention}(q_m, \mathbf{k}, \mathbf{v}) = \sum_i a_{m,i} \cdot v_i$$

- The embeddings **q**, **k**, **v** are learned from the data

Attention mechanism in Machine Translation

- **values** v: The **encoder** hidden states h_i for $i \in \{1, 2, \ldots, H\}$
- **queries** q: The **decoder** hidden states s_i for $i \in \{1, 2, \ldots, S\}$
- **keys** k: The **encoder** hidden states h_i for $i \in \{1, 2, \ldots, H\}$

- The similarity measure for decoding step **i** and encoding step **j** is

$$\text{similarity}(s_i, h_j) = s_i^\top h_j$$

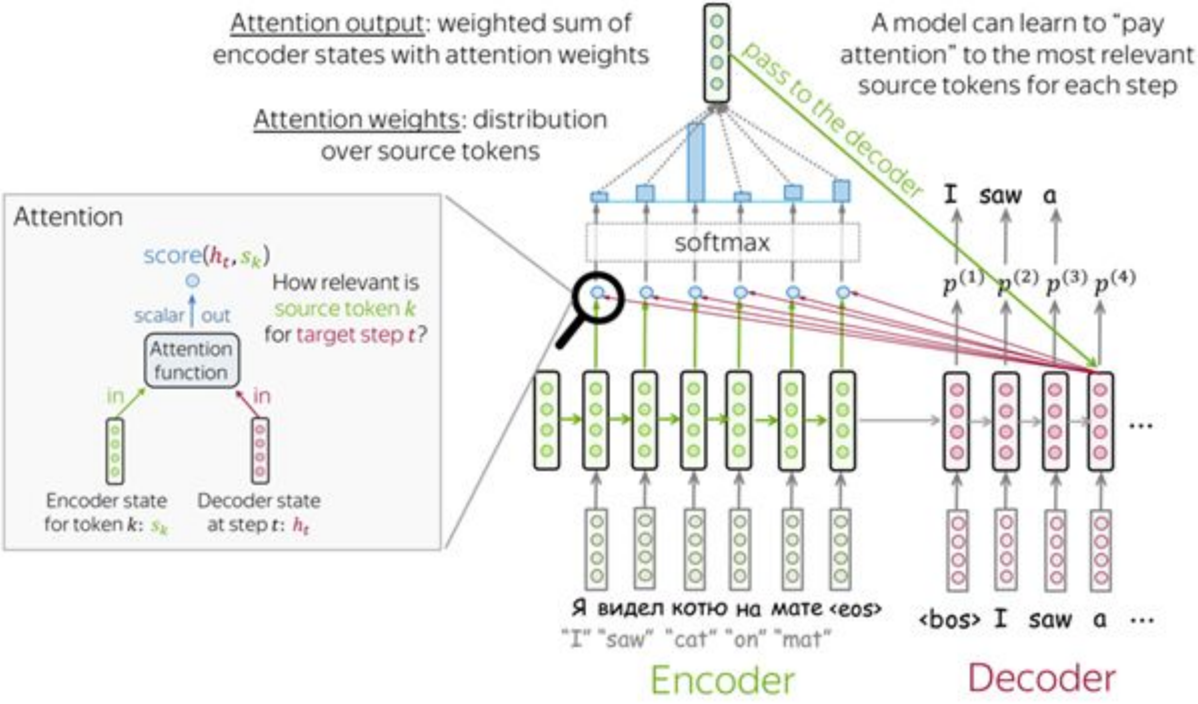
- The attention weight is the softmax probability

$$a_{i,j} = \frac{\exp\left(s_i^\top h_j\right)}{\sum_j \exp\left(s_i^\top h_j\right)}$$

- The new context vector for decoding step **i** is the **attention**

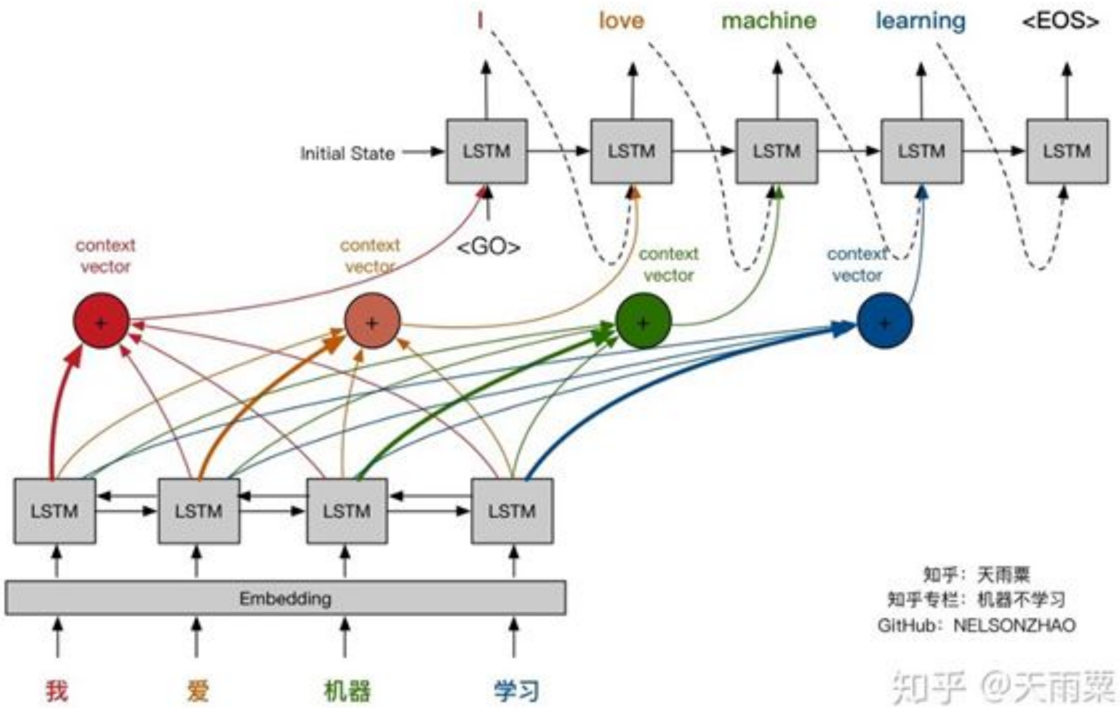
$$c_i = \sum_j a_{i,j} \cdot h_j$$

Attention mechanism in Machine Translation

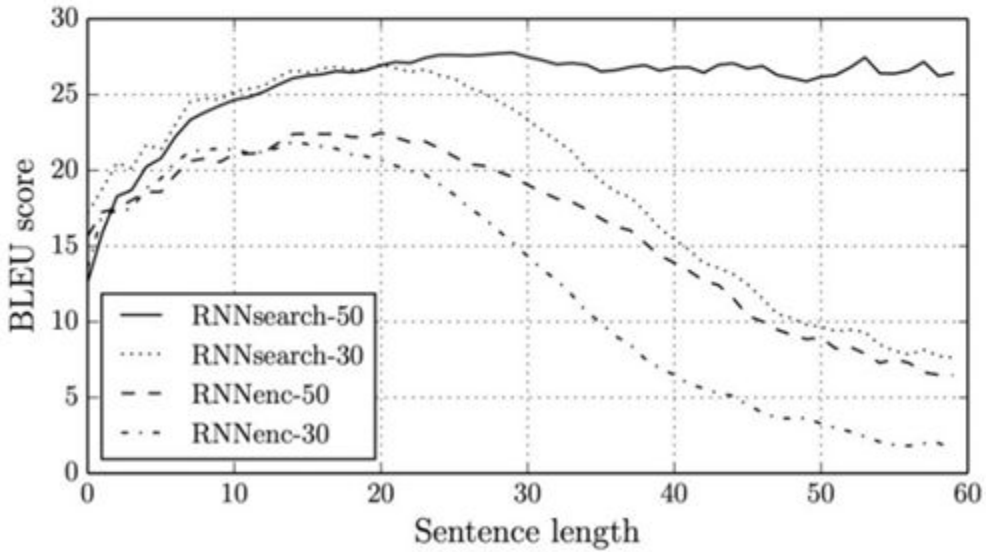


A layer, like a dense layer, an embedding layer

Attention mechanism in Machine Translation

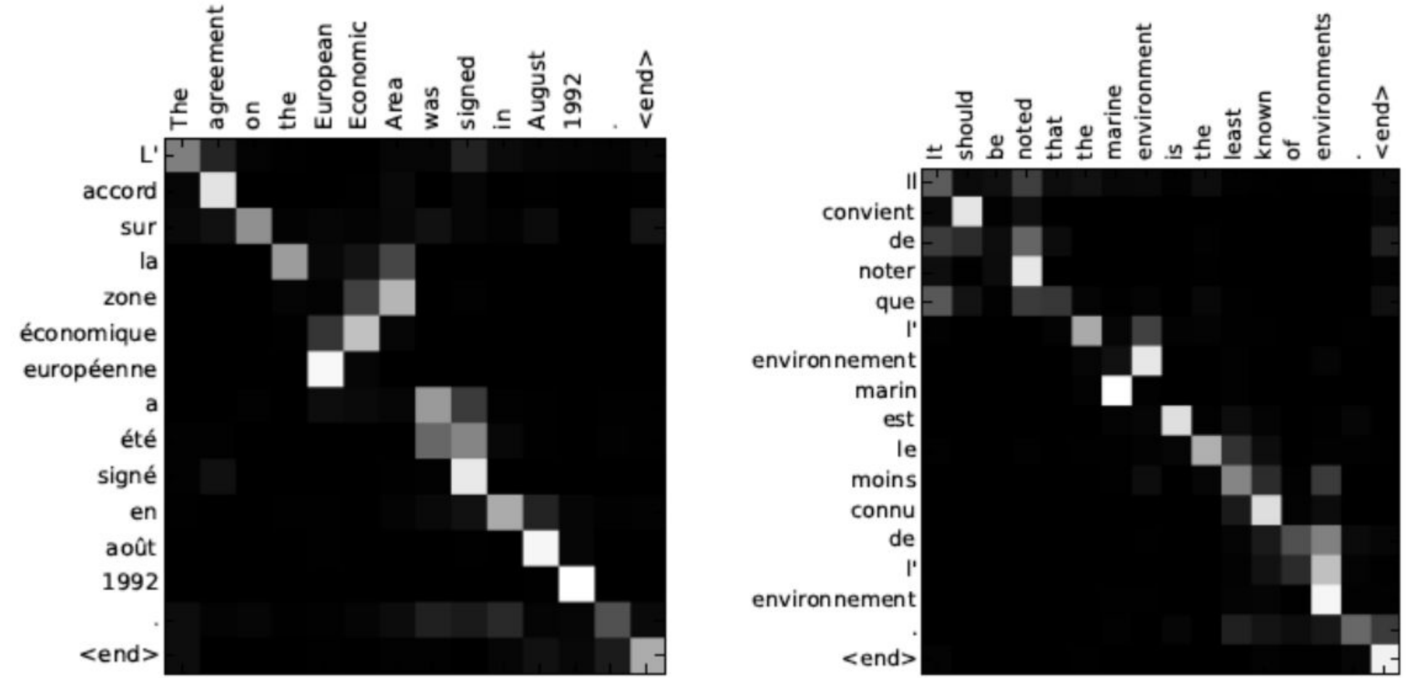


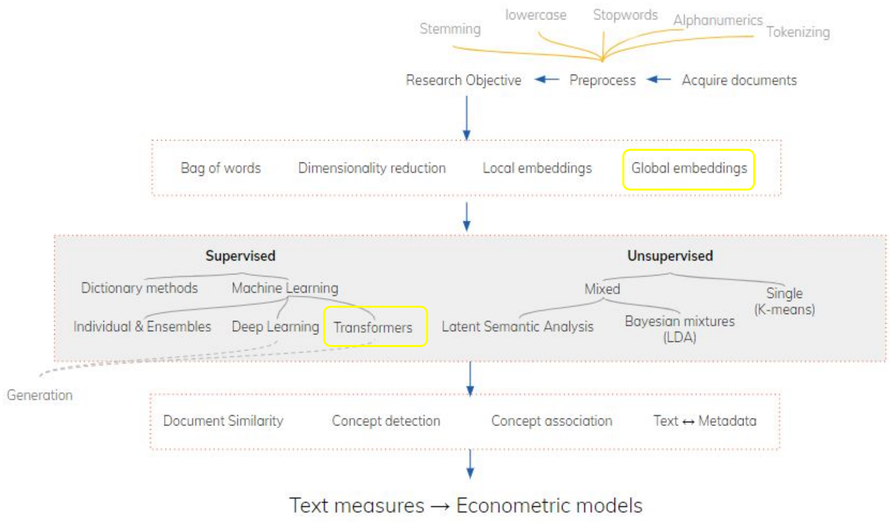
Attention mechanism in Machine Translation 🔗



The Bilingual Evaluation Understudy score is a metric used to evaluate the quality of machine-generated translations by comparing them to one or more human reference translations.

Attention mechanism in Machine Translation





Transformers

Transformers

- Most NLP deep learning use transformer-based architectures since Vaswani et al. (2017)

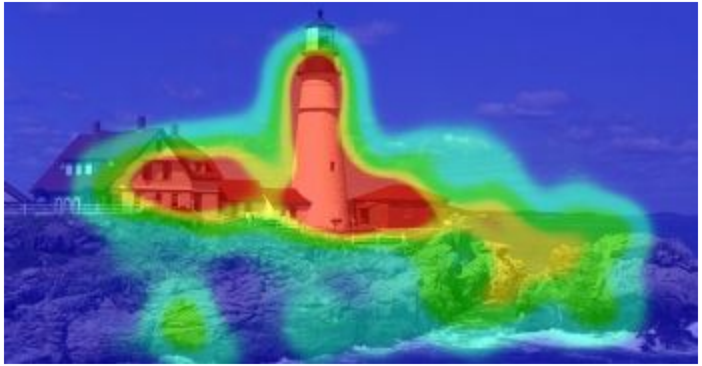
- Recurrent neural nets can process whole documents word-by-word, but they have to sweep through the whole document at each training epoch. This greatly reduces their scalability, as it creates I/O bottlenecks and prevents parallelization.

  Transformers overcome this limitation:

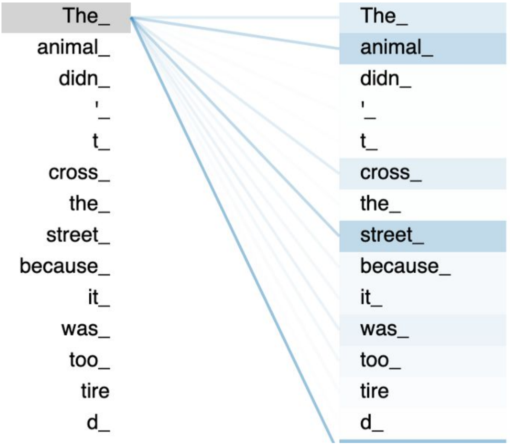
  - These methods process inputs in parallel, making them highly scalable and efficient for large-scale NLP tasks. They don't process the entire input sequentially (no recurrence).
  - They use self-attention mechanisms, enabling them to capture long-range dependencies in text effectively. This allows to retain relationships between words or tokens that are distant from each other.

Attention in AI Research

- **Attention in Computer Vision (2014)**
  - Used to highlight important parts of an image that assist downstream tasks



- **Attention in Natural Language Processing**
  - 2015: Aligned Machine Translation
  - 2017: Transformer Networks & Language Modeling



Transformers in NLP

- Transformer models consist of (many) stacked blocks of parallel **attention heads**
  - These are machine-reading filters, which allow each word to scan over every other word in the document and pick up the most predictive key-value interactions

- **Encoder models**
  - **BERT** - Trained to predict left-out words in the middle of a sequence
  - **AlphaFold** - Protein folding encoder
  - **CLIP** - Image & Object classification

- **Decoder models**
  - **GPT** - Train a transformer to predict the next word at the end of a sequence
  - **LLaMA** - Commonsense reasoning, reading comprehension
  - **Alpaca** - Text generation and classification task

Bottom line: many pre-trained models, can be quickly fine-tuned for downstream tasks (Transfer Learning)

huggingface 🤗

- A popular open-source library and platform for NLP tasks. It offers a comprehensive set of tools and deep learning resources, including Python's most popular **transformer** library.

- It contains hundreds of pre-trained models based on most popular transformer architectures, trained on massive amounts of text data and easily fine-tunable.

- Easy to use API interface, and a large repository of community-contributed pre-trained models, datasets, and evaluation metrics.

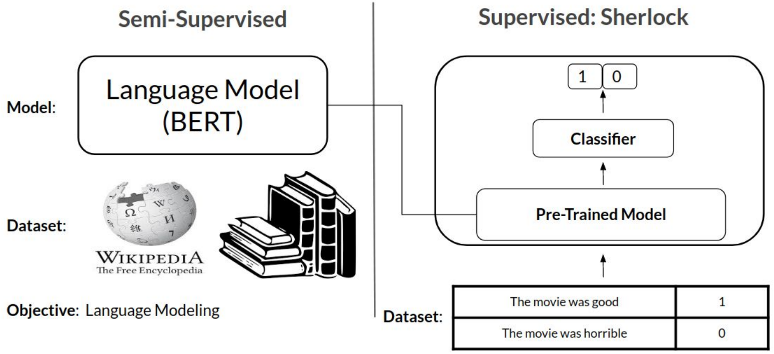
- A Hub where people upload and share their models, and explore others'.

```
from transformers import pipeline
sentiment_analysis = pipeline("sentiment-analysis")

pos_text = "I enjoy studying computational algorithms."
neg_text = "I dislike sleeping late everyday."

pos_sent = sentiment_analysis(pos_text)[0]
print(pos_sent['label'],0 pos_sent['score'])

neg_sent = sentiment_analysis(neg_text)[0]
print(neg_sent['label'], neg_sent['score'])
```



Self-attention mechanism

- A self-referencing attention mechanism
  - **values** v: Word embedding vectors **x** (either input or output)
  - **keys** k: Word embedding vectors **x** (either input or output)
  - **queries** q: Word embedding $\mathbf{x_i}$ of the previously generated output
- The similarity measure for **queried embedding** $\mathbf{x_i}$ and other embeddings is

$$h_i = \sum_{j=1}^{n_L} a(x_i, x_j) x_j$$

- The attention weight is the softmax probability (sum of weights = 1)

$$a(x_i, x_j) = \text{softmax}\left(\frac{x_i \cdot x_j}{\sqrt{n_E}}\right) = \frac{\exp\left(\frac{x_i \cdot x_j}{\sqrt{n_E}}\right)}{\sum_{k=1}^{n_L} \exp\left(\frac{x_i \cdot x_j}{\sqrt{n_E}}\right)}$$

Scaled by embedding length

- **Note**
  - Basic self-attention has no learnable parameters & ignores word order

Self-attention mechanism - example

- Consider a sentence

$$\text{the, cat, walks, on, the, street}$$

- with embeddings

$$\boldsymbol{x}_{\text{the}}, \boldsymbol{x}_{\text{cat}}, \boldsymbol{x}_{\text{walks}}, \boldsymbol{x}_{\text{on}}, \boldsymbol{x}_{\text{the}}, \boldsymbol{x}_{\text{street}}$$

- Feeding this sentence into **self-attention layer** produces

$$\boldsymbol{h}_{\text{the}}, \boldsymbol{h}_{\text{cat}}, \boldsymbol{h}_{\text{walks}}, \boldsymbol{h}_{\text{on}}, \boldsymbol{h}_{\text{the}}, \boldsymbol{h}_{\text{street}}$$

where

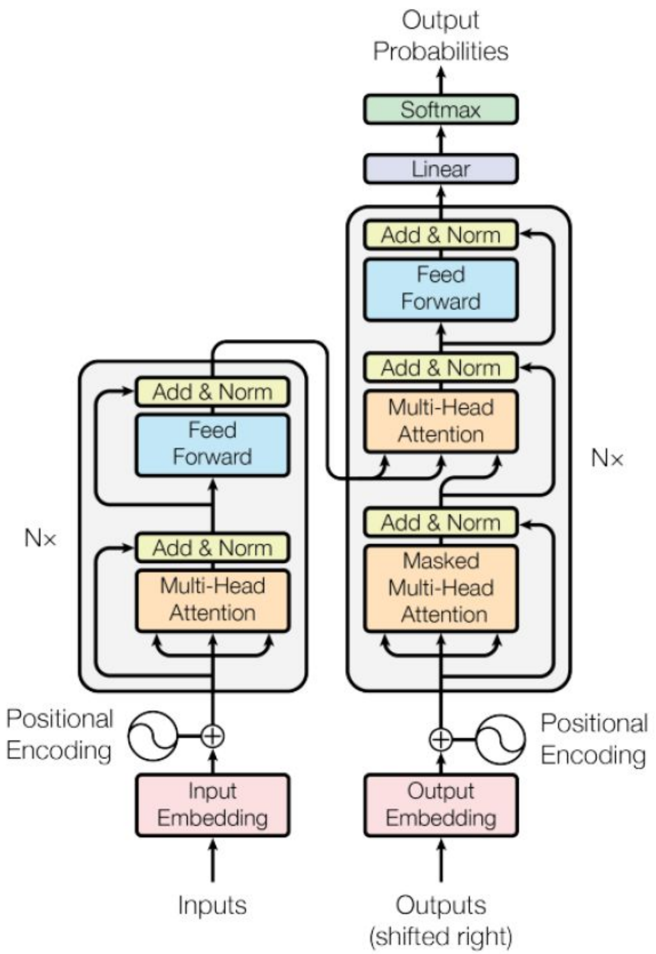
$$\boldsymbol{h}_i = a(x_i \cdot \boldsymbol{x}_{\text{the}})\boldsymbol{x}_{\text{the}} + a(x_i \cdot \boldsymbol{x}_{\text{cat}})\boldsymbol{x}_{\text{cat}} + \dots + a(x_i \cdot \boldsymbol{x}_{\text{street}})\boldsymbol{x}_{\text{street}}$$

Paper 📄

Vaswani et al. (2017) - Attention Is All You Need

- An **encoder-decoder** based on attention
- No recurrence
  - **Input**: entire sequence to translate
  - **Output**: entire translated sentence
  - **Positional encoding** necessary to distinguish embeddings from some BoW representation
- **Core idea**: use attention to learn abstract embeddings
  - **n=1**: pairs of embeddings
  - **n=2**: pairs of pairs of embeddings

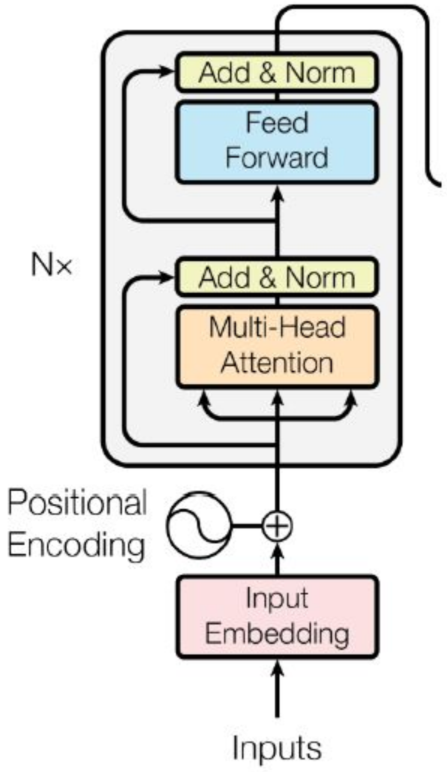
  ...



Paper 📄

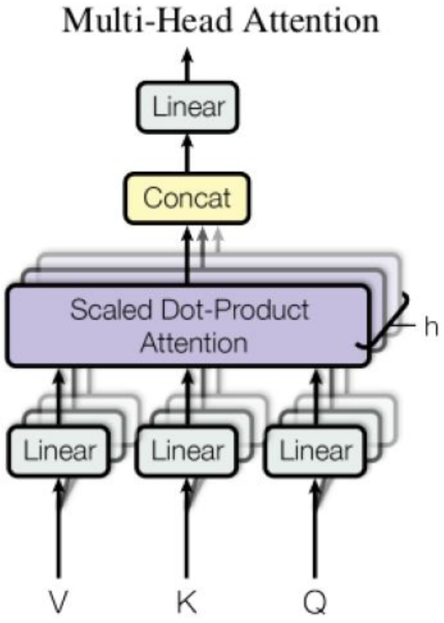
Vaswani et al. (2017) - Attention Is All You Need

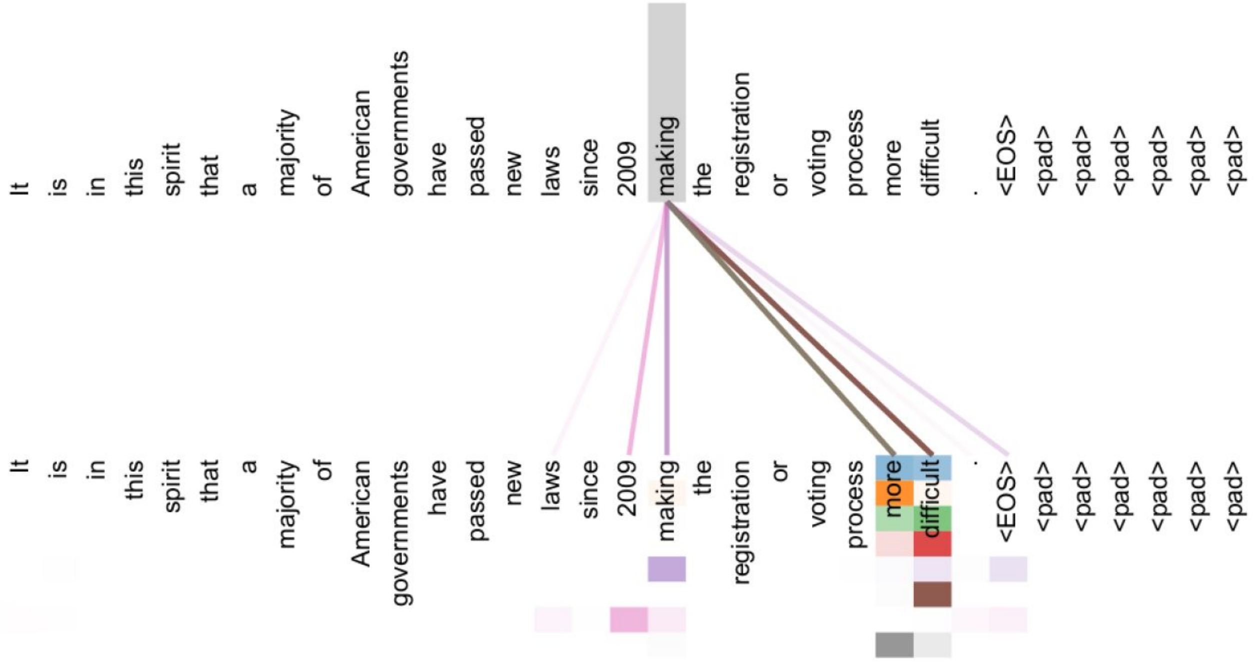
- Input **embedding**: vocabulary-sized embedding
- **Positional encodings**: joins chunk index position
- Repeat N times
  - **Multi-headed attention** layer
  - Bypass & add input embedding, normalise
  - Two linear transformations & ReLU activation
  - Bypass & add embeddings, normalise
- Feed forward to encoder
- Note: Model is bidirectional because next word predictions are done forwards and backwards



Paper 🔗

Vaswani et al. (2017) - Attention Is All You Need



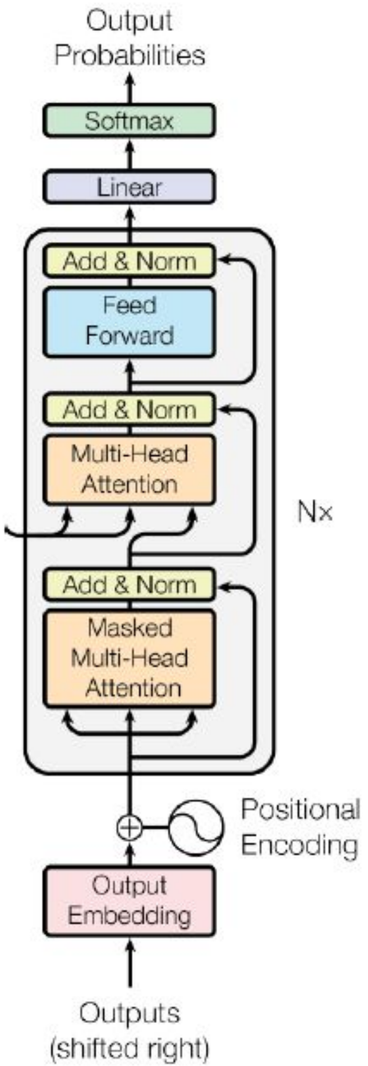


As nice as this is, this is also the models' main drawback: token limits

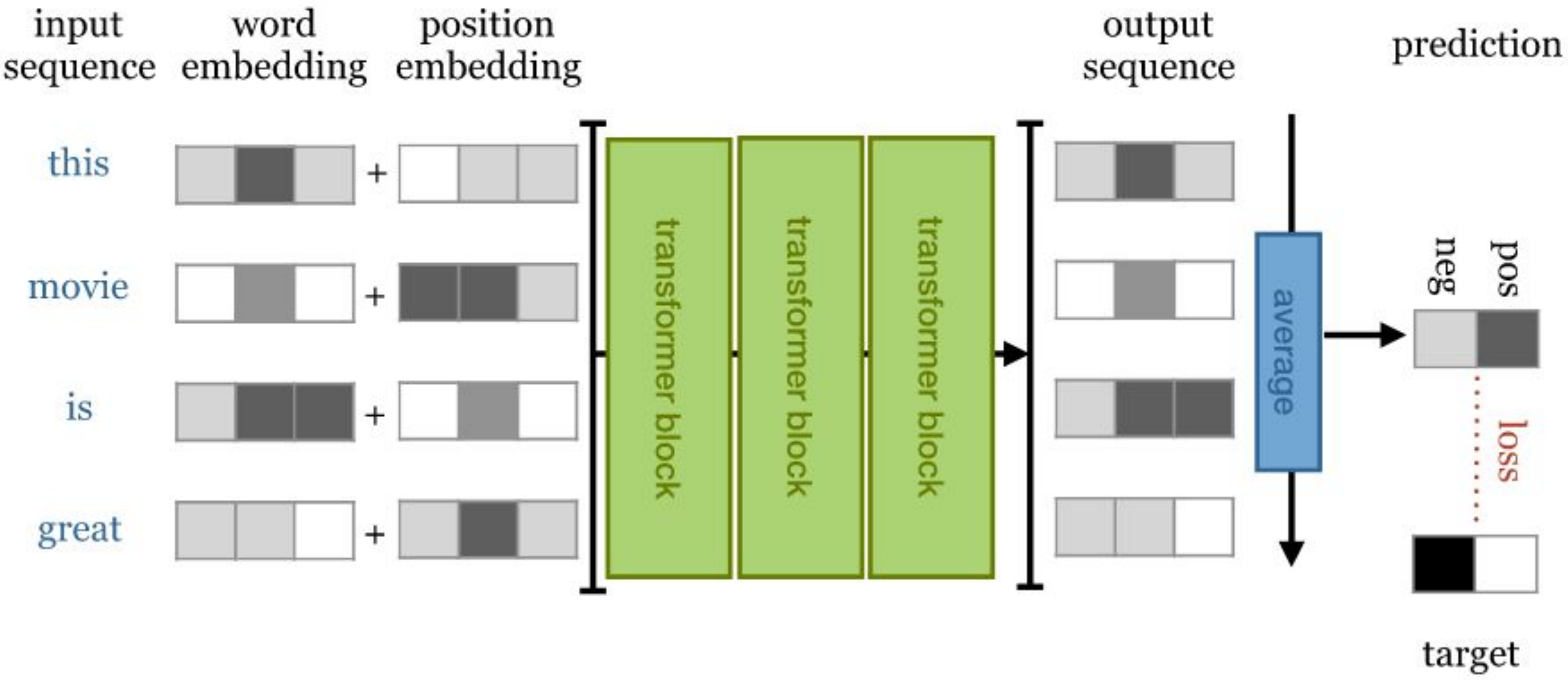
Paper

Vaswani et al. (2017) - Attention Is All You Need

- Output **embedding**: vocabulary-sized embedding
- **Positional encodings**: joins chunk index position
- Repeat N times
  - **Multi-headed attention** layer
  - Bypass & add input embedding, normalise
  - Multi-head attention for retrieving output embeddings using the encoder query
  - Bypass & add embeddings, normalise
  - Two linear transformations & ReLU activation
  - Bypass & add embeddings, normalise
- Linear transformation, softmax for probabilities
- Prediction



Transformers for practitioners: sentiment



Paper application

BIngler et al. (2023) - Cheap talk and cherry picking: ClimateBERT

Goal

Fine tune a BERT architecture to identify firms' voluntary disclosures related to climate (TCFD) & compare actions

Methodology

- **Create sample:** collect firms' annual reports and identify manually sentences related to climate disclosures align in one of several TCFD categories
- **Fine-tune model**: use a BERT decoder, and add Linear Layers on top to predict sentences categories (if apply)
- **Predict & Compare**: Compare talk with actual actions

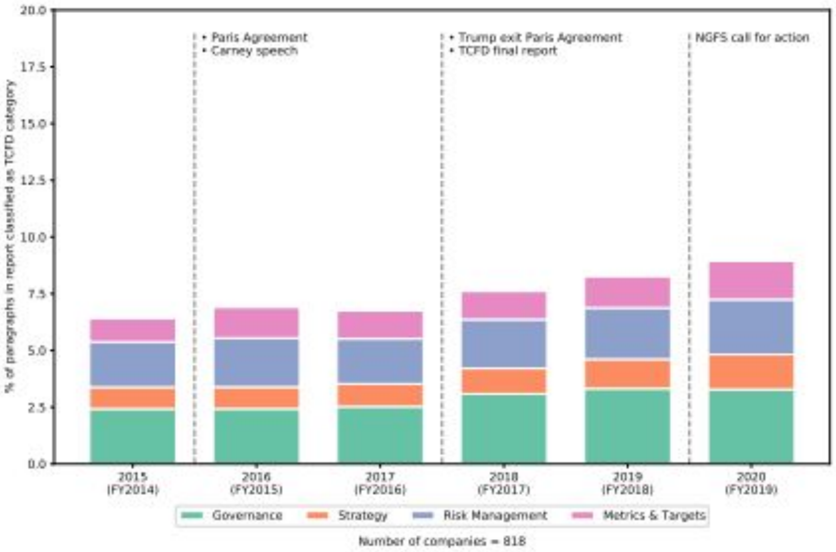
Results

- Voluntary disclosures suffer from **cheap talk**
- Announced TCFD support does not lead to an increase in disclosures, and cherry pick disclosures to overwhelmingly reflect non-material categories

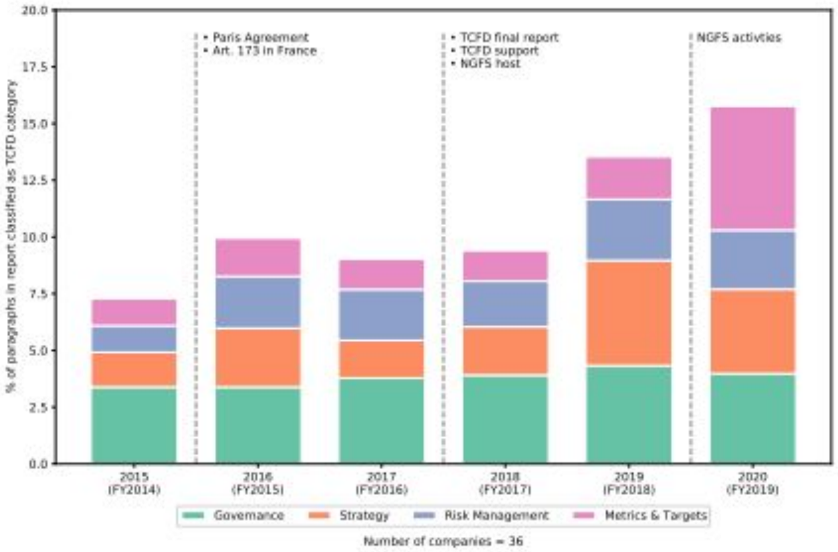
Governance	Strategy	Risk Management	Metrics and Targets
Disclose the organization's governance around climate-related risks and opportunities.	Disclose the actual and potential impacts of climate-related risks and opportunities on the organization's businesses, strategy, and financial planning where such information is material.	Disclose how the organization identifies, assesses, and manages climate-related risks.	Disclose the metrics and targets used to assess and manage relevant climate-related risks and opportunities where such information is material.
	Recommended Disclosures		
Describe the board's oversight of climate-related risks and opportunities	Describe the climate-related risks and opportunities the organization has identified over the short, medium, and long term.	Describe the organization's processes for identifying and assessing climate-related risks	Disclose the metrics used by the organization to assess climate-related risks and opportunities in line with its strategy and risk management process.
Describe management's role in assessing and managing climate-related risks and opportunities.	Describe the impact of climate-related risks and opportunities on the organization's businesses, strategy, and financial planning.	Describe the organization's processes for managing climate-related risks.	Disclose Scope 1, Scope 2, and, if appropriate, Scope 3 greenhouse gas (GHG) emissions, and the related risks.
	Describe the resilience of the organization's strategy, taking into consideration different climate-related scenarios, including a 2°C or lower scenario.	Describe how processes for identifying, assessing, and managing climate-related risks are integrated into the organization's overall risk management.	Describe the targets used by the organization to manage climate-related risks and opportunities and performance against targets.

Paper application

BIngler et al. (2023) - Cheap talk and cherry picking: ClimateBERT

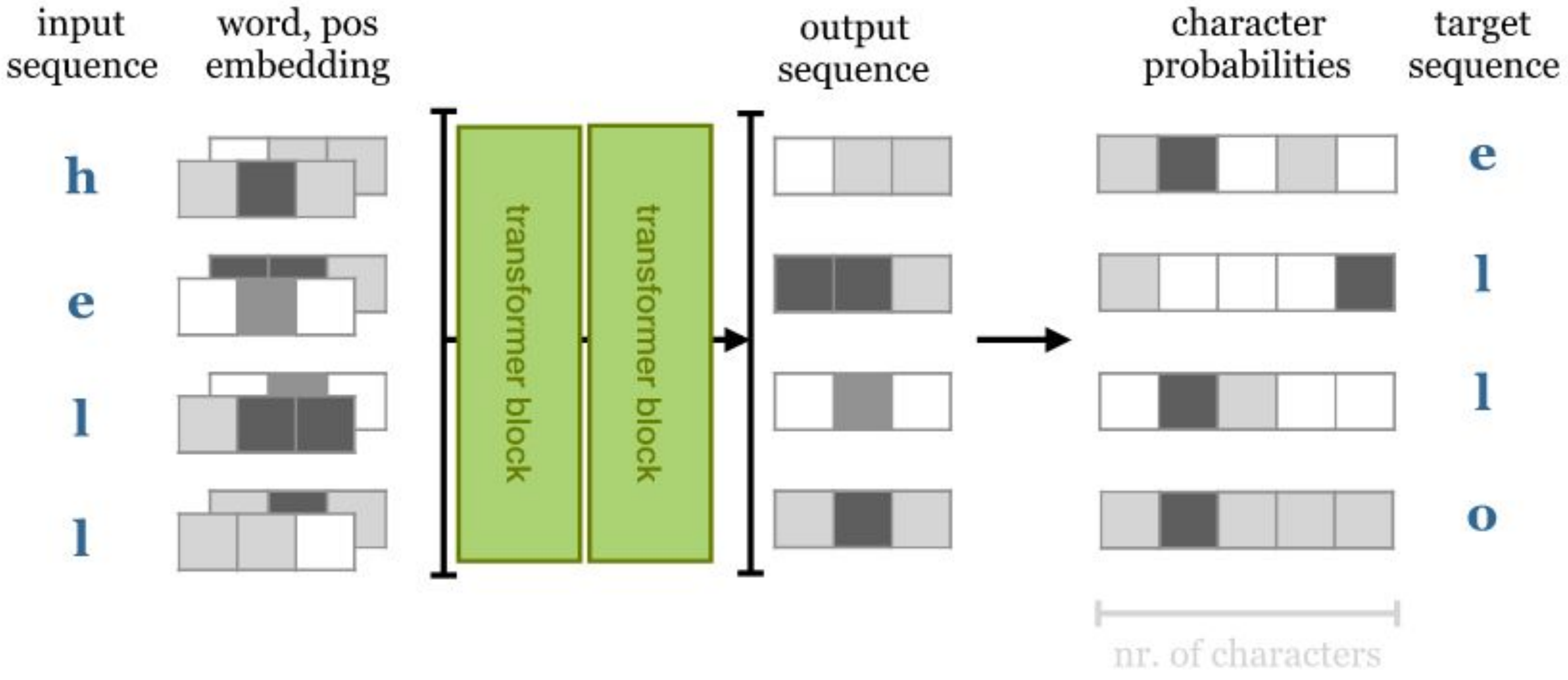


(a) Full sample



(b) France

Transformers for practitioners: autoregression



Evolution of Generative Pre-Trained Transformers

GPT-1 (2018) - 117M params

- Trained on the Books corpus
- Trained on a language modeling task, as well as a multi-task that adds a supervised learning task

GPT-2 (2019) - 1.5B params

- All articles linked from Reddit with at least 3 upvotes (8 million documents, 40GB of text)
- Dispense with supervised learning task, make adjustments, enlarge the model by many orders of magnitud

GPT-3 (2020) - 175B params

- Use an even larger corpus (Common Crawl, WebText2, Books1, Books2, Wikipedia)
- Model becomes much larger

InstructGPT ↔ GPT-3.5 ↔ GPT-4

- Add reinforcement learning with human feedback to improve responsiveness & user satisfaction
- Closed-source, controversial for OpenAI - an organisation built on the backs of the open-source crowd