Deep Learning

Generative models in vision and text (Transformers, GANs)

Lessons: **Kevin Scaman**
TPs: Paul Lerner

Class overview

Lessons

1. Introduction, simple architectures (MLPs) and autodiff 09/02
2. Training pipeline, optimization and image analysis (CNNs) 16/02
3. Sequence regression (RNNs), stability and robustness 08/03
4. **Generative models in vision and text (Transformers, GANs)** 15/03

Generative models

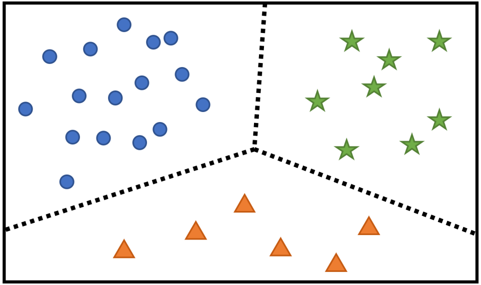
Beyond classification tasks

What is a generative model?

Generative vs. discriminative

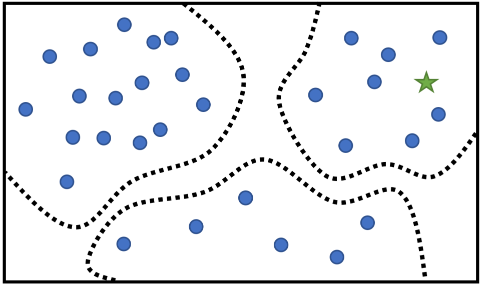
- Discriminative tasks such as classification aim at **separating** data.
- Generative tasks aim at **creating** new data.

Discriminative tasks



Classification (access to (X,y) pairs)

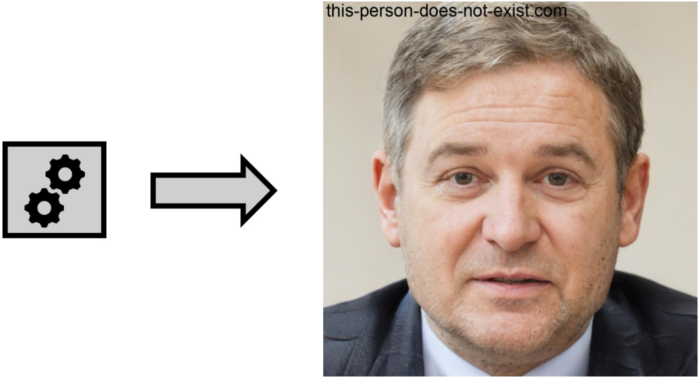
Generative tasks



Sampling (access to X only)

Examples of generative models

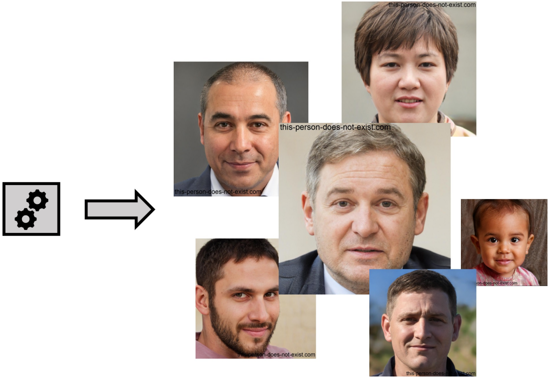
- Image generation (face generation, deepfakes, ...).



source: `https://this-person-does-not-exist.com/en`

Examples of generative models

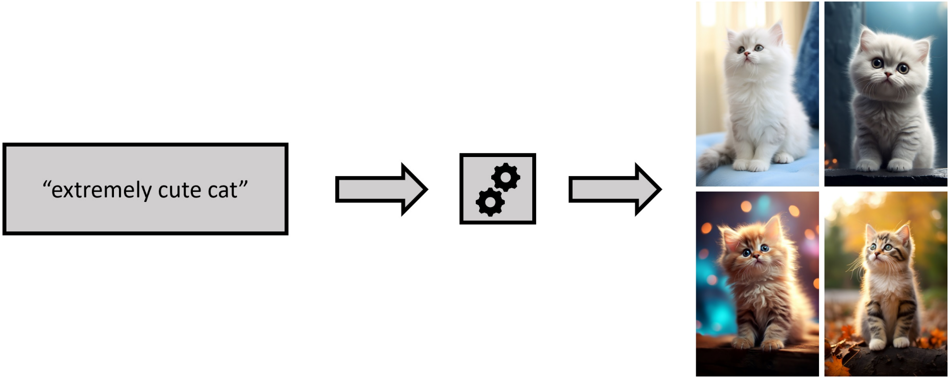
- Image generation (face generation, deepfakes, ...).



source: `https://this-person-does-not-exist.com/en`

Examples of generative models

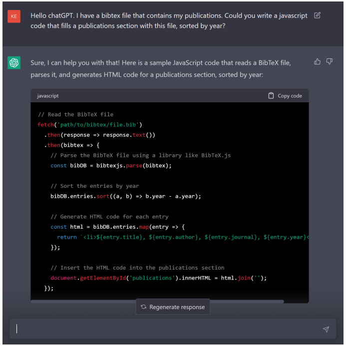
- Image generation (face generation, deepfakes, ...).
- Prompt-based image generation (Dalle2, Imagen, MidjourneyAI, ...).



source: MidjourneyAI. `https://midjourney.com/`

Examples of generative models

- Image generation (face generation, deepfakes, ...).
- Prompt-based image generation (Dalle2, Imagen, MidjourneyAI, ...).
- Text generation (Bert, GPT2, GPT3, ChatGPT, Bard, Sparrow, ...).



source: ChatGPT. `https://chat.openai.com/`

Neural architectures for generative tasks

Key aspects of a generative model

- We want to **output complex data** (e.g. images, text, ...).
- We want to **sample random outputs** from a learnt distribution.
- Usually involves more **difficult optimization problems** than standard ERM.
- How do we measure **performance**?

Neural architectures for generative tasks

Key aspects of a generative model

- We want to **output complex data** (e.g. images, text, ...).
- We want to **sample random outputs** from a learnt distribution.
- Usually involves more **difficult optimization problems** than standard ERM.
- How do we measure **performance**?

Three main approaches

1. Variational auto-encoders (VAEs)
2. Generative Adversarial Networks (GANs)
3. Score-based generative models / diffusion models

Generating random variables

Classical approaches to sampling probability distributions

Generative models

Approximating distributions with NNs

- **Intuition:** How do we create models whose outputs are **random variables**?

Generative models

Approximating distributions with NNs

- **Intuition:** How do we create models whose outputs are **random variables**?
- **Data:** $\hat{\mathcal{D}}_n = (X_1, \dots, X_n)$ i.i.d. according to some target distribution $\mathcal{D}$.

Generative models

Approximating distributions with NNs

- **Intuition:** How do we create models whose outputs are **random variables**?
- **Data:** $\hat{\mathcal{D}}_n = (X_1, \ldots, X_n)$ i.i.d. according to some target distribution $\mathcal{D}$.
- **Objective:** sample new elements $\tilde{X} \sim \mathcal{D}$ from the target distribution.

Generative models

Approximating distributions with NNs

- **Intuition:** How do we create models whose outputs are **random variables**?
- **Data:** $\hat{\mathcal{D}}_n = (X_1, \ldots, X_n)$ i.i.d. according to some target distribution $\mathcal{D}$.
- **Objective:** sample new elements $\tilde{X} \sim \mathcal{D}$ from the target distribution.

Extensions

- **Prompt-based models:** one data distribution per input query. Equivalent to supervised learning with a random output.

Generative models

Approximating distributions with NNs

- **Intuition:** How do we create models whose outputs are **random variables**?
- **Data:** $\hat{\mathcal{D}}_n = (X_1, \ldots, X_n)$ i.i.d. according to some target distribution $\mathcal{D}$.
- **Objective:** sample new elements $\tilde{X} \sim \mathcal{D}$ from the target distribution.

Extensions

- **Prompt-based models:** one data distribution per input query. Equivalent to supervised learning with a random output.
- **Learn a density function:** some models also provide a density function.

No clear cut: classification tasks also generate probability distributions...

How to sample from a known distribution $\mathcal{D}$?

How to sample from a known distribution $\mathcal{D}$?

Standard approaches

- **Parametric families of distributions:** sampled by a simple function of a base distribution. E.g. Gaussian $X = \mu + \sigma Y$ where $Y \sim \mathcal{N}(0, 1)$.

How to sample from a known distribution $\mathcal{D}$?

Standard approaches

- **Parametric families of distributions:** sampled by a simple function of a base distribution. E.g. Gaussian $X = \mu + \sigma Y$ where $Y \sim \mathcal{N}(0, 1)$.
- **Inversion sampling:** For 1D r.v., we have $X = F^{-1}(Y)$ where $Y \sim \mathcal{U}([0, 1])$ is uniform in $[0, 1]$ and F is the cumulative distribution function of $\mathcal{D}$.

How to sample from a known distribution $\mathcal{D}$?

Standard approaches

- **Parametric families of distributions:** sampled by a simple function of a base distribution. E.g. Gaussian $X = \mu + \sigma Y$ where $Y \sim \mathcal{N}(0, 1)$.
- **Inversion sampling:** For 1D r.v., we have $X = F^{-1}(Y)$ where $Y \sim \mathcal{U}([0, 1])$ is uniform in $[0, 1]$ and F is the cumulative distribution function of $\mathcal{D}$.
- **Monte-Carlo Markov Chains:** Start with a base distribution (e.g. Gaussian), and iteratively refine it to get closer and closer to the target distribution $\mathcal{D}$.

How to sample from a known distribution $\mathcal{D}$?

Standard approaches

- **Parametric families of distributions:** sampled by a simple function of a base distribution. E.g. Gaussian $X = \mu + \sigma Y$ where $Y \sim \mathcal{N}(0, 1)$.
- **Inversion sampling:** For 1D r.v., we have $X = F^{-1}(Y)$ where $Y \sim \mathcal{U}([0, 1])$ is uniform in $[0, 1]$ and F is the cumulative distribution function of $\mathcal{D}$.
- **Monte-Carlo Markov Chains:** Start with a base distribution (e.g. Gaussian), and iteratively refine it to get closer and closer to the target distribution $\mathcal{D}$.

How to use it for generative models?

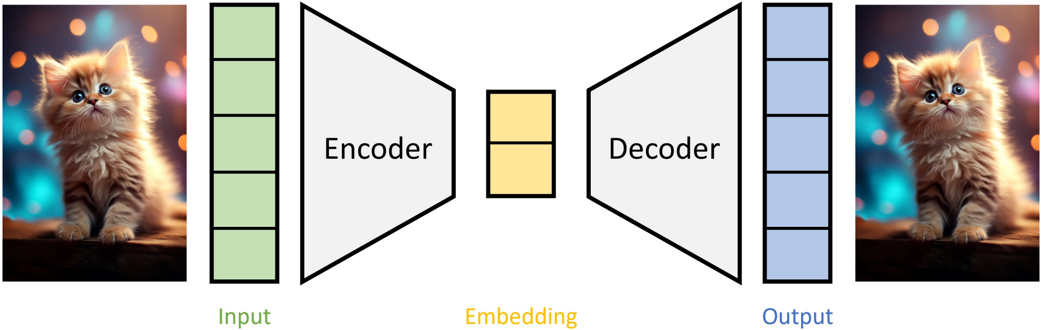
- **Parameter modelling:** Learn the parameters $(\mu, \sigma) = g_\theta(x)$ to generate $\mathcal{N}(\mu, \sigma)$.
- **Transformation:** generate with $g_\theta(Y) \sim \mathcal{D}$ where $Y \sim \mathcal{N}(0, I)$ (**VAEs**, **GANs**).
- **Dynamics:** Learn iterative refinements that transform $\mathcal{N}(0, I)$ into $\mathcal{D}$ (**diffusion**).

Variational Autoencoders (VAEs)

From compression to generation

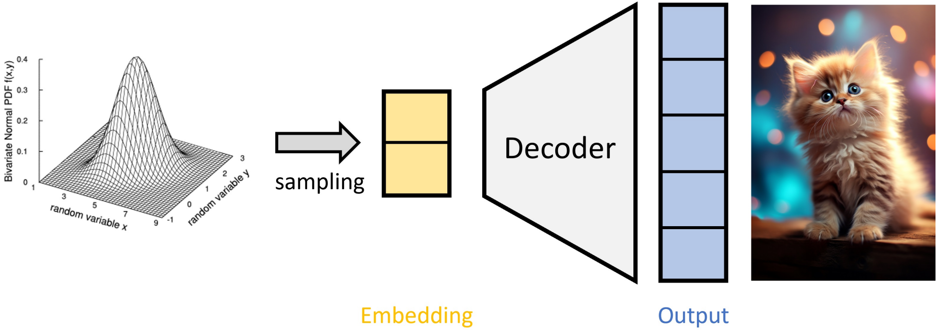
But first... what is an autoencoder?

- **Objective:** Learn a **compressed data representation** in an unsupervised manner.
- **Idea:** Map **data points to themselves** $g_\theta(x) = x$ with **small inner representation**.
- **Loss:** Let $e_\theta, d_{\theta'}$ be two NNs, we want to minimize $\mathbb{E}(\|X - d_{\theta'}(e_\theta(X))\|^2)$.



But first... what is an autoencoder?

- **Compression:** If latent space is smaller than input space, information is **compressed**.
- **Generation:** We can sample from the **latent space**.



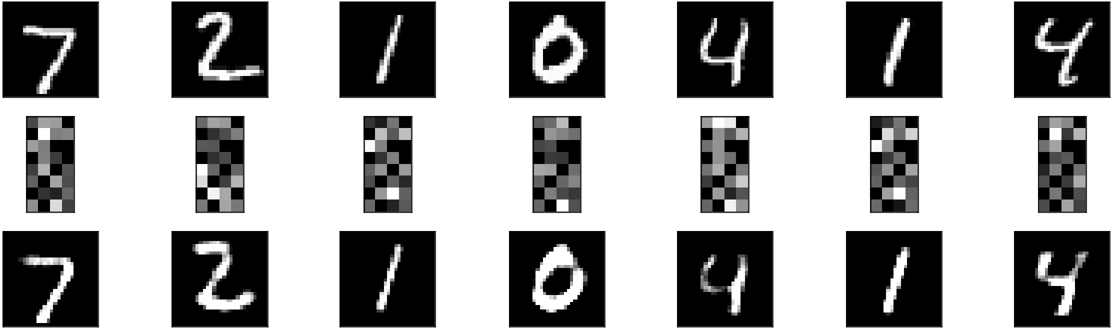
Autoencoders in PyTorch

The simplest possible autoencoder with a **single affine layer** as encoder and as decoder:

```
class AutoEncoder(nn.Module):
    def __init__(self, input_dim, encoding_dim):
        super(AutoEncoder, self).__init__()
        self.encoder = nn.Linear(input_dim, encoding_dim)
        self.decoder = nn.Linear(encoding_dim, input_dim)
    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded
```

Autoencoders in PyTorch

After training, we obtain:

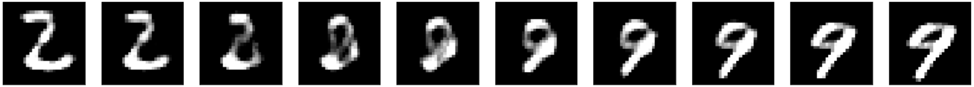


Representation learning with autoencoders

- **Interpolation in latent space:** We can interpolate between two images x and y with

$$x_\alpha \quad = \quad d_{\theta'}\Big(\alpha\, e_\theta(x) + (1-\alpha)\, e_\theta(y)\Big)$$

  for $\alpha \in [0, 1]$.
- **Results:** Interpolation between digits 2 and 9.



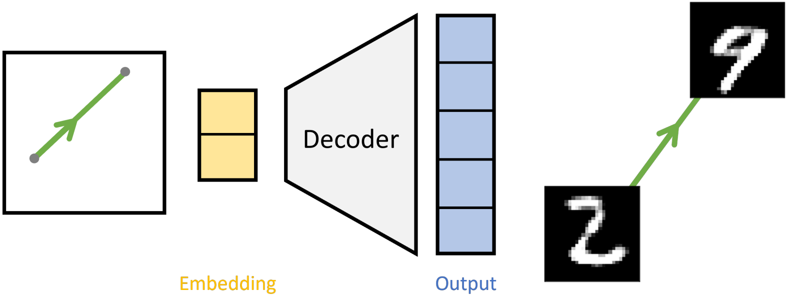
- Better than in the **pixel space**, but not perfect still...

Representation learning with autoencoders

- **Interpolation in latent space:** We can interpolate between two images x and y with

$$x_\alpha \quad = \quad d_{\theta'}\Big(\alpha\, e_\theta(x) + (1-\alpha)\, e_\theta(y)\Big)$$

for $\alpha \in [0, 1]$.



Is this a good generative model?

- **Limitations:** There is **no constraint** on the **regularity** of the latent space embedding.

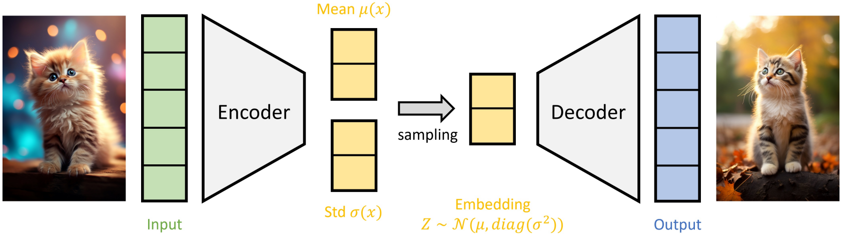
source: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

Variational Autoencoders (VAEs)

- **Objective:** Regularize by forcing the embedding to be **robust to noise**.
- **Idea:** The encoder returns the parameters $(\mu_x, \sigma_x) = e_\theta(x)$ of a Gaussian distribution. We sample $Z_x \sim \mathcal{N}(\mu_x, \sigma_x)$ and minimize

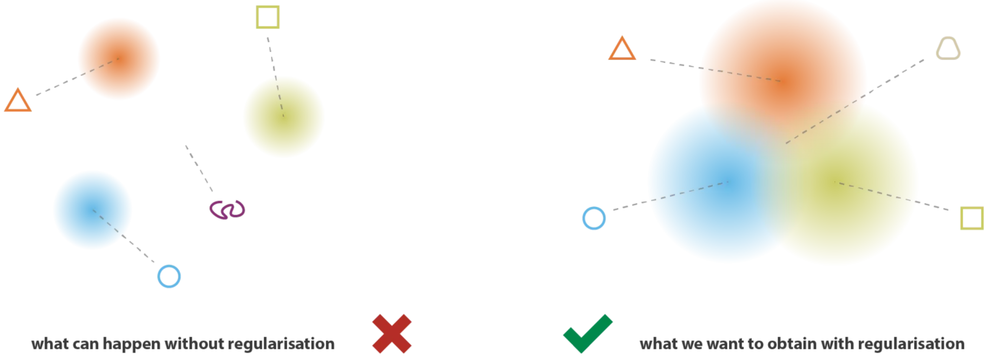
$$\min_{\theta,\theta'} \frac{1}{n} \sum_{i=1}^{n} \|x_i - d_{\theta'}(Z_{x_i})\|^2 + d_{\mathsf{KL}}\Big(\mathcal{N}(\mu_{x_i}, \sigma_{x_i}), \mathcal{N}(0, I)\Big)$$

where $d_{\mathsf{KL}}(p, q) = \mathbb{E}_{X\sim p}(\log(p(X)/q(X)))$ measures the "distance" between p and q.



Regularization with KL divergence

- **Benefits:** Each image is pushed to be mapped to a normal distribution.
- **Sampling:** We can sample new images with $d_{\theta'}(Z)$ where $Z \sim \mathcal{N}(0, I)$.

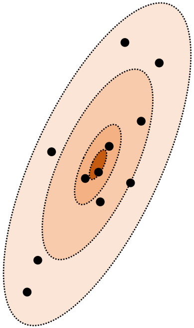


Performance measures

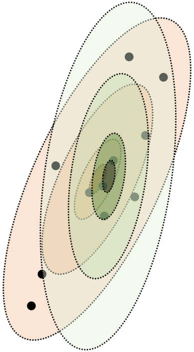
When is our model good enough?

Comparing data distribution and generated distribution

- **Question:** How should we measure distances between real and generated distributions?



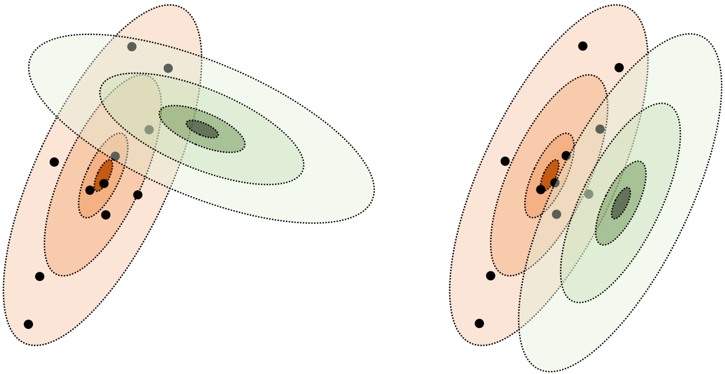
Training dataset and
underlying distribution



Good fit!

Comparing data distribution and generated distribution

- **Question:** How should we measure distances between real and generated distributions?



Bad fit? Better fit!?

Likelihood of the data distribution

- We already saw that we can train a probabilistic model by **maximizing the likelihood** of the training data points (approach used by most of statistics!).

Likelihood of the data distribution

- We already saw that we can train a probabilistic model by **maximizing the likelihood** of the training data points (approach used by most of statistics!).
- Equivalent to minimizing the **negative log-likelihood**:

$$\min_\theta - \sum_{i=1}^n \log p_\theta(x_i)$$

where $(x_1, \dots, x_n)$ are the training data points and p_θ is the density of the distribution.

Likelihood of the data distribution

- We already saw that we can train a probabilistic model by **maximizing the likelihood** of the training data points (approach used by most of statistics!).
- Equivalent to minimizing the **negative log-likelihood**:

$$\min_{\theta} - \sum_{i=1}^{n} \log p_\theta(x_i)$$

  where $(x_1, \ldots, x_n)$ are the training data points and p_θ is the density of the distribution.
- **Cons:** Requires to have access to the density p_θ. Can overfit training data.

Likelihood of the data distribution

- We already saw that we can train a probabilistic model by **maximizing the likelihood** of the training data points (approach used by most of statistics!).
- Equivalent to minimizing the **negative log-likelihood**:

$$\min_{\theta} - \sum_{i=1}^{n} \log p_\theta(x_i)$$

  where $(x_1, \ldots, x_n)$ are the training data points and p_θ is the density of the distribution.
- **Cons:** Requires to have access to the density p_θ. Can overfit training data.
- This is equivalent to minimizing the **Kullback-Leibler divergence** $d_{KL}(\hat{p}_n, p_\theta)$, where:

$$d_{KL}(p, q) = \mathbb{E}\left(\ln\left(\frac{p(X)}{q(X)}\right)\right)$$

  where $\hat{p}_n = \frac{1}{n}\sum_i \delta_{x_i}$ and $X \sim p$.

Other performance metrics

Wasserstein distance

- Measures how similar are the two measures via **evaluation functions**:

$$d_W(\mu,\nu) = \sup_{f \in \mathsf{Lip}_1} |\mathbb{E}(f(X)) - \mathbb{E}(f(Y))|$$

where $X \sim \mu$, $Y \sim \nu$ and Lip_1 is the space of 1-Lipschitz functions.

Other performance metrics

Wasserstein distance

- Measures how similar are the two measures via **evaluation functions**:

$$d_W(\mu,\nu) = \sup_{f \in \mathsf{Lip}_1} |\mathbb{E}(f(X)) - \mathbb{E}(f(Y))|$$

where $X \sim \mu$, $Y \sim \nu$ and Lip_1 is the space of 1-Lipschitz functions.
- Measures are similar if there is **no way to distinguish** them with (Lipschitz) statistics.
- Another (equivalent) definition via **optimal transport**.

Other performance metrics

Wasserstein distance

- Measures how similar are the two measures via **evaluation functions**:

$$d_W(\mu, \nu) = \sup_{f \in \mathsf{Lip}_1} |\mathbb{E}(f(X)) - \mathbb{E}(f(Y))|$$

  where $X \sim \mu$, $Y \sim \nu$ and Lip_1 is the space of 1-Lipschitz functions.
- Measures are similar if there is **no way to distinguish** them with (Lipschitz) statistics.
- Another (equivalent) definition via **optimal transport**.

Human evaluation

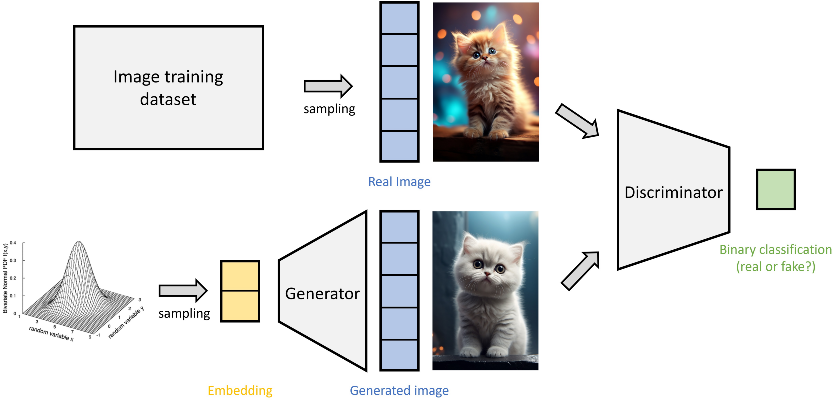
- Compare the outputs and decide which generative model you prefer...
- **Limitations:** subjective, and difficult to assess **diversity**.

Generative Adversarial Networks (GANs)

Asking another NN if your NN is good enough

Generative Adversarial Networks (Goodfellow et.al., 2014)

- **Idea:** Use another NN (discriminator) to **compare true and generated images**.
- Discriminator finds **mistakes** in the generation, and generator learns to **fool** the critic.



Traning GANs: a min-max optimization problem

- **Generator:** g_θ generates a fake sample $g_\theta(Z)$ with a Gaussian r.v. $Z \sim \mathcal{N}(0, I)$.

Traning GANs: a min-max optimization problem

- **Generator:** g_θ generates a fake sample $g_\theta(Z)$ with a Gaussian r.v. $Z \sim \mathcal{N}(0, I)$.
- **Discriminator:** $d_{\theta'}$ is a **classifier** and $d_{\theta'}(x)$ is the probability for x to be a real sample.

Traning GANs: a min-max optimization problem

- **Generator:** g_θ generates a fake sample $g_\theta(Z)$ with a Gaussian r.v. $Z \sim \mathcal{N}(0, I)$.
- **Discriminator:** $d_{\theta'}$ is a **classifier** and $d_{\theta'}(x)$ is the probability for x to be a real sample.
- **Learning:** g_θ and $d_{\theta'}$ are learnt **alternatively**, i.e. one is fixed when the other is learnt.
- **Loss:** For real images $(x_1, \ldots, x_n)$ and generated images $(g_\theta(Z_1), \ldots, g_\theta(Z_n))$, we want

$$\max_{\theta} \min_{\theta'} \mathcal{L}(\theta, \theta') = -\frac{1}{n} \sum_{i=1}^{n} \log \Big(d_{\theta'}(x_i) \Big) + \log \Big(1 - d_{\theta'}(g_\theta(z_i)) \Big)$$

Traning GANs: a min-max optimization problem

- **Generator:** g_θ generates a fake sample $g_\theta(Z)$ with a Gaussian r.v. $Z \sim \mathcal{N}(0, I)$.
- **Discriminator:** $d_{\theta'}$ is a **classifier** and $d_{\theta'}(x)$ is the probability for x to be a real sample.
- **Learning:** g_θ and $d_{\theta'}$ are learnt **alternatively**, i.e. one is fixed when the other is learnt.
- **Loss:** For real images $(x_1, \ldots, x_n)$ and generated images $(g_\theta(Z_1), \ldots, g_\theta(Z_n))$, we want

$$\max_{\theta} \min_{\theta'} \mathcal{L}(\theta, \theta') = -\frac{1}{n} \sum_{i=1}^{n} \log \Big(d_{\theta'}(x_i) \Big) + \log \Big(1 - d_{\theta'}(g_\theta(z_i)) \Big)$$

- **Interpretation:** Discriminator minimizes its **BCE loss**, generator tries to **maximize** it.

Recap

- Generative models rely on **learning to sample** probability distributions.
- VAEs use an **Encoder-Decoder** architecture to learn a **low-dimensional latent representation** of the data distribution.
- GANs use two adversarial networks trained alternatively (**Generator** and **Discriminator**).
- To create images from low-dimensional vectors, we need to use **transposed convolutions**.
- Training is very **unstable**, and requires lots of tricks in practice.

Understanding the Transformer architecture*

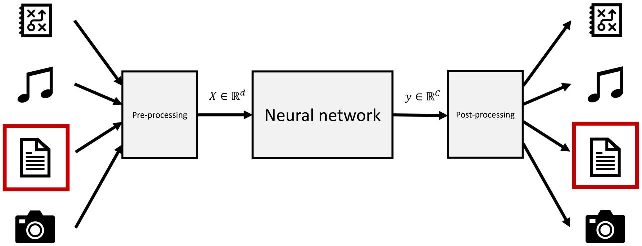
State-of-the-art natural language processing models

*...with the help of a Transformer architecture. ☺

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 $(u_1^i, \ldots, u_t^i)_{i \in [\![1,n]\!]} \in \mathcal{D}^{t \times n}$ a training dataset of sequences (e.g. text doc.) of length t.
- **Task:** Let $(u_1, \ldots, u_t) \in$ a sequence. We want to predict u_t given $(u_1, \ldots, u_{t-1})$.

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

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 $(u_1^i, \ldots, u_t^i)_{i \in [\![1,n]\!]} \in \mathcal{D}^{t \times n}$ a training dataset of sequences (e.g. text doc.) of length t.
- **Task:** Let $(u_1, \ldots, u_t) \in$ a sequence. We want to predict u_t given $(u_1, \ldots, u_{t-1})$.

"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

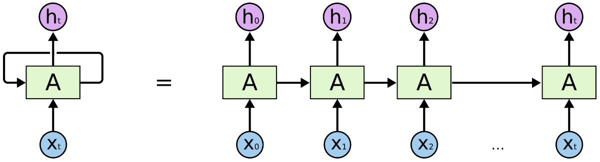
A simple Markov model (Shannon, 1948)

- **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 $u \in \mathcal{D}$.

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)

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 $(u_1, \ldots, u_{t-1}) \in \mathcal{D}^{(t-1)\times n}$:

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

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:** We have $\text{Softmax}_i(x) \in [0, 1]$ and $\sum_i \text{Softmax}_i(x) = 1$.

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:** We have $\text{Softmax}_i(x) \in [0, 1]$ and $\sum_i \text{Softmax}_i(x) = 1$.

Applications

- **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 $x = x_1, \dots, x_n$ be scores associated to n items, then

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

- **Properties:** We have $\text{Softmax}_i(x) \in [0, 1]$ and $\sum_i \text{Softmax}_i(x) = 1$.

Applications

- **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

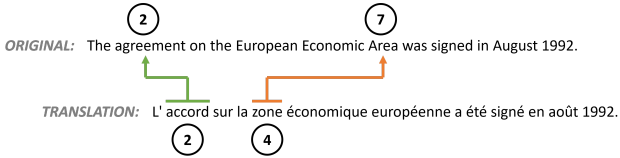
$$\text{Softmax}_i(x) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

- **Properties:** We have $\text{Softmax}_i(x) \in [0, 1]$ and $\sum_i \text{Softmax}_i(x) = 1$.

Applications

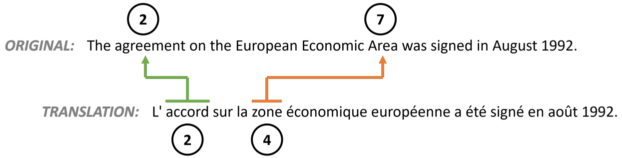
- **Cross-entropy:** We have $\ell_{CE}(y, y') = -\log(\text{Softmax}_{y'}(y))$.
- **Max:** We can return a "soft max" with $\sum_i \text{Softmax}_i(x)\, x_i$ or $\log(\sum_{j=1}^{n} e^{x_j})$.
- **Argmax:** We can select the item with highest score s_i with $\sum_i \text{Softmax}_i(s)\, x_i$.

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



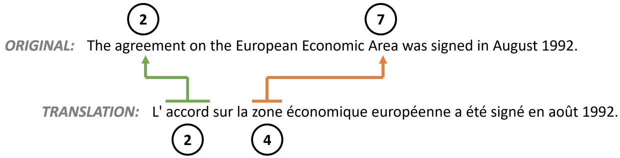
- **Idea:** Align the words between two different languages using attention.

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



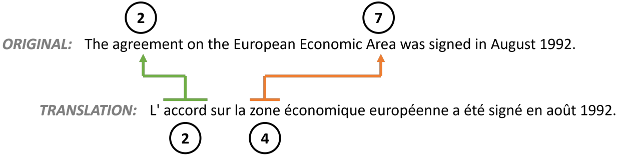
- **Idea:** Align the words between two different languages using attention.
- **Encoding:** For an input sentence (e.g. in English) $(u_1, \ldots, u_T)$, we use an LSTM to compute hidden vectors representing each word/token $(h_1, \ldots, h_T)$.

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) $(u_1, \dots, u_T)$, we use an LSTM to compute hidden vectors representing each word/token $(h_1, \dots, h_T)$.
- **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.

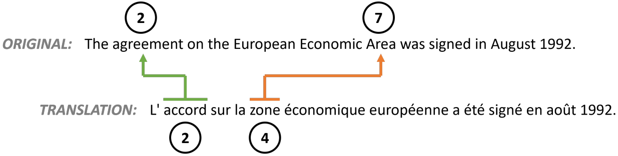
Attention layers: first use in translation (Bahdanau et.al., 2015)



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

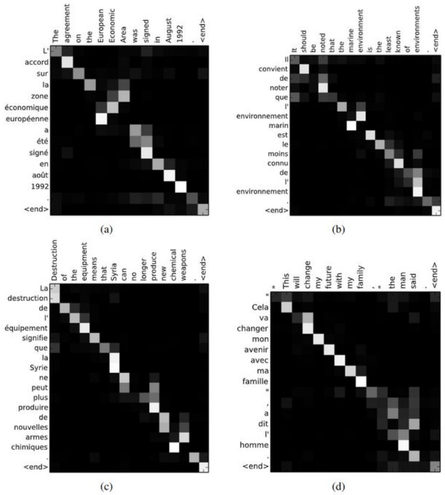


- **Idea:** Align the words between two different languages using attention.
- **Encoding:** For an input sentence (e.g. in English) $(u_1, \ldots, u_T)$, we use an LSTM to compute hidden vectors representing each word/token $(h_1, \ldots, h_T)$.
- **Decoding:** We then compute recursively the hidden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

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

- **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 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).
- **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, $\forall s \in [\![1, S]\!]$,

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

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).
- **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, $\forall s \in [\![1, S]\!]$,

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

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

(Self-)attention layers in Transformers

Self-attention

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

(Self-)attention 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

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

(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

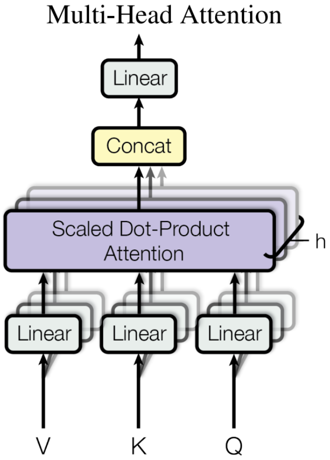
$$Y_s = \sum_{t=1}^{T} \text{Softmax}_t \left(\frac{Q_s^\top K}{\sqrt{k}} \right) V_t$$

- 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

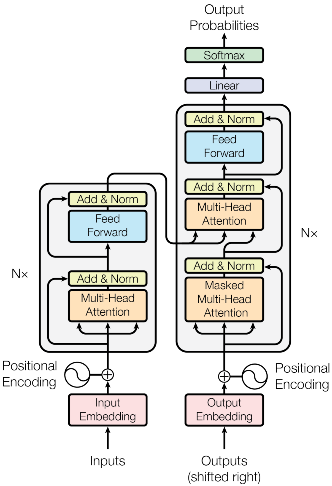
- As for channels in convolution layers, we perform H **parallel** self-attention layers, and combine them with a linear layer. **Usual choice:** take $d_{in} = d_{out} \cdot H$.

Multi-head attention



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

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



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

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**!

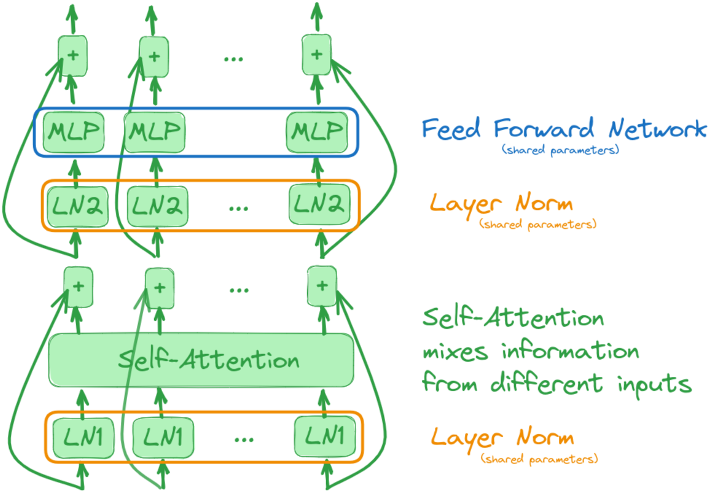
Positional encoding

- **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**.

Positional encoding: parallel computations



source: https://dataflowr.github.io/website/modules/12-attention/

Layer normalization

Idea

- Same as batch normalization, but normalized **per layer** instead of per batch.
- 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.
- Ensures that all elements in the sequence have approximately the same amplitude.

Definition

- If $(x_i)_i$ is a batch of b inputs (to the layer), then the output is:

$$y_i = \frac{x_i - E}{\sqrt{V + \varepsilon}} \cdot \gamma + \beta$$

where $E = \frac{1}{d} \sum_i x_i$ and $V = \frac{1}{d} \sum_i (x_i - E)^2$ (coord.-wise), γ and β are learnable vectors.

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

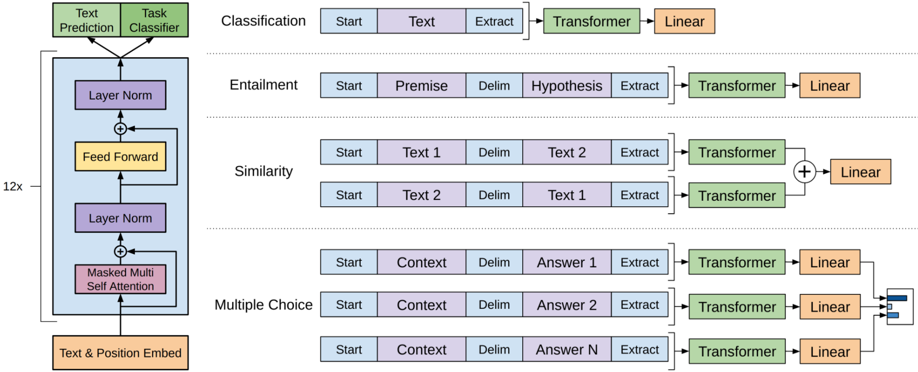
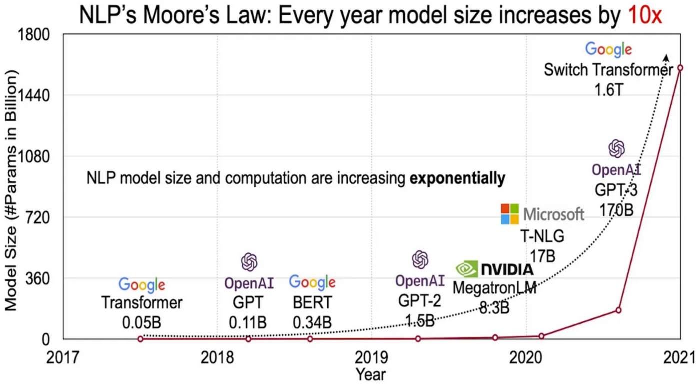


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

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**.



The Increasing Size of LLM Models

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

The LLM family: recent models

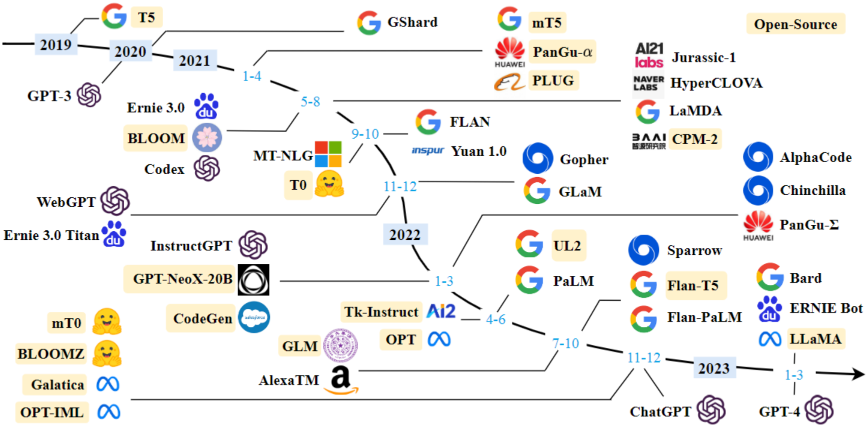


Fig. 1. A timeline of existing large language models (having a size larger than 10B) in recent years. We mark the open-source LLMs in yellow color.

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

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 [35]	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**.