Machine Learning

Volker Roth

Department of Mathematics & Computer Science
University of Basel

Chapter 11: Neural Encoder-Decoder Models

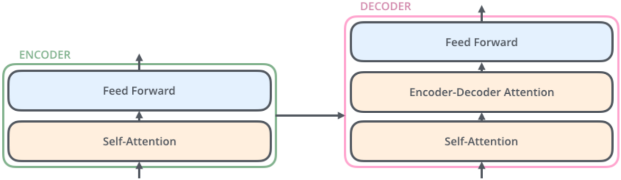
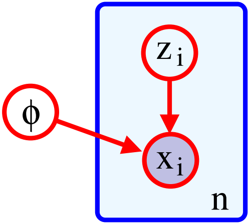


Figure 16.1 in the supplement of K.Murphy: Probabilistic Machine Learning, Advanced Topics. MIT Press, 2023. High level structure of the encoder-decoder transformer architecture. https://jalammar.github.io/illustrated-transformer/

Non-linear latent variable models

Latent variable $\boldsymbol{z} \rightsquigarrow$ Gaussian likelihood with **nonlinearly transformed mean** $\boldsymbol{\mu} = f(\boldsymbol{z}, \phi)$.

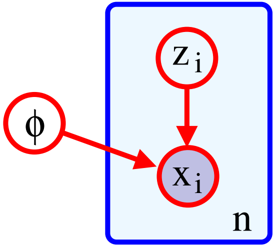


Prior and likelihood:

$$p(\boldsymbol{z}) = N(\boldsymbol{0}, I)$$
$$p(\boldsymbol{x}|\boldsymbol{z}, \phi) = N(\boldsymbol{f}(\boldsymbol{z}, \phi), \sigma^2 I).$$

- Given observed $\boldsymbol{x}$, we want to understand what possible values of the hidden variable $\boldsymbol{z}$ were responsible for it:
$$p(\boldsymbol{z}|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{p(\boldsymbol{x})}.$$
- No closed form expression available. Cannot evaluate denominator $p(\boldsymbol{x})$ and so we can't even compute the numerical value of the posterior for a given pair $\boldsymbol{z}$ and $\boldsymbol{x}$.

Sampling



- ...but it is easy to generate a new sample $\boldsymbol{x}^*$ using sampling:
  - Draw $\boldsymbol{z}^*$ from the prior $p(\boldsymbol{z})$, pass this through $f(\boldsymbol{z}^*, \phi)$ $\rightsquigarrow$ mean of likelihood $p(\boldsymbol{x}^*|\boldsymbol{z}^*)$,
  - then draw $\boldsymbol{x}^*$ from this distribution.
- Prior and likelihood are normal distributions $\rightsquigarrow$ sampling is easy.

Evaluating marginal likelihood (evidence)

$$\begin{aligned} p(\boldsymbol{x}|\phi) &= \int p(\boldsymbol{x}, \boldsymbol{z}|\phi) d\boldsymbol{z} \\ &= \int p(\boldsymbol{x}|\boldsymbol{z}, \phi) p(\boldsymbol{z}) d\boldsymbol{z} \\ &= \int N(\boldsymbol{f}[\boldsymbol{z}, \phi], \sigma^2 I) \cdot N(\mathbf{0}, I)\, d\boldsymbol{z}. \end{aligned}$$

No closed form for this integral $\rightsquigarrow$ lower bound (Jensen's inequality):

$$\begin{aligned} \log[p(\boldsymbol{x}|\phi)] &= \log\left[\int p(\boldsymbol{x}, \boldsymbol{z}|\phi) d\boldsymbol{z}\right] \\ &= \log\left[\int q(\boldsymbol{z}) \frac{p(\boldsymbol{x}, \boldsymbol{z}|\phi)}{q(\boldsymbol{z})} d\boldsymbol{z}\right] \\ &\geq \int q(\boldsymbol{z}) \log\left[\frac{p(\boldsymbol{x}, \boldsymbol{z}|\phi)}{q(\boldsymbol{z})}\right] d\boldsymbol{z}, \end{aligned}$$

Known as the evidence lower bound ELBO, because $p(\boldsymbol{x}|\phi)$ is the evidence (= marginal likelihood) in the context of Bayes' rule.

ELBO

- In practice, the distribution $q(\boldsymbol{z})$ will have some parameters $\boldsymbol{\theta}$:
$$\mathrm{ELBO}[\theta, \phi] = \int q(\boldsymbol{z}|\boldsymbol{\theta}) \log \left[\frac{p(\boldsymbol{x}, \boldsymbol{z}|\phi)}{q(\boldsymbol{z}|\boldsymbol{\theta})} \right] d\boldsymbol{z}.$$
- To learn the non-linear latent variable model, we'll maximize this quantity as a function of both ϕ and $\boldsymbol{\theta}$.
- We will see: the maximum is obtained (theoretically) if the variational distribution is the true posterior, $q(\boldsymbol{z}|\boldsymbol{\theta}) = p(\boldsymbol{z}|\boldsymbol{x}, \phi)$.
- In practice, we maximize ELBO over some tractable family of distributions $q(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{\theta})$ to obtain an approximation of the intractable posterior.
- The neural architecture that computes this is the **variational autoencoder.**

Tightness of ELBO

$$
\begin{aligned}
\mathrm{ELBO}[\boldsymbol{\theta}, \phi] &= \int q(\boldsymbol{z}|\boldsymbol{\theta}) \log \left[\frac{p(\boldsymbol{x}, \boldsymbol{z}|\phi)}{q(\boldsymbol{z}|\boldsymbol{\theta})} \right] d\boldsymbol{z} \\
&= \int q(\boldsymbol{z}|\boldsymbol{\theta}) \log \left[\frac{p(\boldsymbol{z}|\boldsymbol{x}, \phi) p(\boldsymbol{x}|\phi)}{q(\boldsymbol{z}|\boldsymbol{\theta})} \right] d\boldsymbol{z} \\
&= \int q(\boldsymbol{z}|\boldsymbol{\theta}) \log [p(\boldsymbol{x}|\phi)] \, d\boldsymbol{z} + \int q(\boldsymbol{z}|\boldsymbol{\theta}) \log \left[\frac{p(\boldsymbol{z}|\boldsymbol{x}, \phi)}{q(\boldsymbol{z}|\boldsymbol{\theta})} \right] d\boldsymbol{z} \\
&= \log[p(\boldsymbol{x}|\phi)] + \int q(\boldsymbol{z}|\boldsymbol{\theta}) \log \left[\frac{p(\boldsymbol{z}|\boldsymbol{x}, \phi)}{q(\boldsymbol{z}|\boldsymbol{\theta})} \right] d\boldsymbol{z} \\
&= \log[p(\boldsymbol{x}|\phi)] - \mathrm{D}_{KL} \left[q(\boldsymbol{z}|\boldsymbol{\theta}) \| p(\boldsymbol{z}|\boldsymbol{x}, \phi) \right].
\end{aligned}
$$

ELBO is the log marginal likelihood minus $\mathrm{D}_{KL} [q(\boldsymbol{z}|\boldsymbol{\theta}) \| p(\boldsymbol{z}|\boldsymbol{x}, \phi)]$.
D_{KL} zero when $q(\boldsymbol{z}|\boldsymbol{\theta}) = p(\boldsymbol{z}|\boldsymbol{x}, \phi) \rightsquigarrow \mathrm{ELBO} = \log[p(\boldsymbol{x}|\phi)]$.

ELBO as reconstruction loss minus KL to prior

$$
\begin{aligned}
\text{ELBO}[\boldsymbol{\theta}, \phi] &= \int q(\boldsymbol{z}|\boldsymbol{\theta}) \log \left[\frac{p(\boldsymbol{x}, \boldsymbol{z}|\phi)}{q(\boldsymbol{z}|\boldsymbol{\theta})} \right] d\boldsymbol{z} \\
&= \int q(\boldsymbol{z}|\boldsymbol{\theta}) \log \left[\frac{p(\boldsymbol{x}|\boldsymbol{z}, \phi) p(\boldsymbol{z})}{q(\boldsymbol{z}|\boldsymbol{\theta})} \right] d\boldsymbol{z} \\
&= \int q(\boldsymbol{z}|\boldsymbol{\theta}) \log \left[p(\boldsymbol{x}|\boldsymbol{z}, \phi) \right] d\boldsymbol{z} + \int q(\boldsymbol{z}|\boldsymbol{\theta}) \log \left[\frac{p(\boldsymbol{z})}{q(\boldsymbol{z}|\boldsymbol{\theta})} \right] d\boldsymbol{z} \\
&= \int q(\boldsymbol{z}|\boldsymbol{\theta}) \log \left[p(\boldsymbol{x}|\boldsymbol{z}, \phi) \right] d\boldsymbol{z} - \mathrm{D}_{KL}[q(\boldsymbol{z}|\boldsymbol{\theta}), p(\boldsymbol{z})]
\end{aligned}
$$

- First term measures the average agreement $p(\boldsymbol{x}|\boldsymbol{z}, \phi)$ of the hidden variable and the data (reconstruction loss)
- Second one measures the degree to which the auxiliary distribution $q(\boldsymbol{z}, \boldsymbol{\theta})$ matches the prior.

The variational approximation

- ELBO is tight when we choose $q(\boldsymbol{z}|\boldsymbol{\theta}) = p(\boldsymbol{z}|\boldsymbol{x}, \phi)$.
- Intractable $\rightsquigarrow$ variational approximation: choose simple parametric form for $q(\boldsymbol{z}|\boldsymbol{\theta})$, use it as an approximation to the true posterior.
- Choose a normal distribution with parameters $\boldsymbol{\mu}$ and $\Sigma = \sigma^2 I$.
- Optimization $\rightsquigarrow$ find normal distribution closest to true posterior $p(\boldsymbol{z}|\boldsymbol{x})$. Corresponds to minimizing the KL divergence.
- True posterior $p(\boldsymbol{z}|\boldsymbol{x})$ depends on $\boldsymbol{x}$
  $\rightsquigarrow$ variational approximation should also depend on $\boldsymbol{x}$:

$$q(\boldsymbol{z}|\boldsymbol{\theta}, \boldsymbol{x}) = N(g_\mu[\boldsymbol{x}|\boldsymbol{\theta}], g_{\sigma^2}[\boldsymbol{x}|\boldsymbol{\theta}]),$$

where $g[\boldsymbol{x}, \boldsymbol{\theta}]$ is a neural network with parameters $\boldsymbol{\theta}$.

The variational autoencoder

Recall

$$\text{ELBO}[\boldsymbol{\theta}, \phi] = \int q(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{\theta}) \log \left[p(\boldsymbol{x}|\boldsymbol{z}, \phi) \right] d\boldsymbol{z} - \mathrm{D}_{KL}[q(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{\theta}), p(\boldsymbol{z})]$$

Involves an intractable integral, but it is an expectation
$\rightsquigarrow$ approximate with samples:

$$E_{q(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{\theta})}[\log \left[p(\boldsymbol{x}|\boldsymbol{z}, \phi) \right]] \approx \frac{1}{N} \sum_{n=1}^{N} \log \left[p(\boldsymbol{x}|\boldsymbol{z}_n^*, \phi) \right]$$

where $\boldsymbol{z}_n^*$ is the n-th sample from $q(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{\theta})$. Limit: use a single sample:

$$\text{ELBO}[\boldsymbol{\theta}, \phi] \approx \quad \log \left[p(\boldsymbol{x}|\boldsymbol{z}^*, \phi) \right] - \mathrm{D}_{KL}[q(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{\theta}), p(\boldsymbol{z})]$$

The second term is just the KL divergence between two Gaussians and is available in closed form.

The reparameterization trick

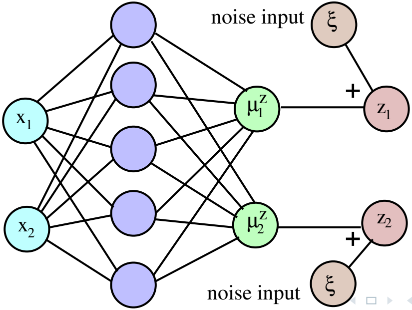
Recall: Want to sample from

$$q(\boldsymbol{z}|\boldsymbol{\theta}, \boldsymbol{x}) = N(g_{\boldsymbol{\mu}}[\boldsymbol{x}|\boldsymbol{\theta}], g_{\sigma^2}[\boldsymbol{x}|\boldsymbol{\theta}]),$$

To let PyTorch / Tensorflow perform automatic differentiation via backpropagation, we must avoid the sampling step.
Simple solution: draw a sample $\boldsymbol{\xi} \sim N(0, I)$ and use

$$\boldsymbol{z}^* = g_{\boldsymbol{\mu}} + g_{\sigma^2}^{1/2}\boldsymbol{\xi}.$$

Now "the gradient can flow through the network". **Encoder network:**



VAE

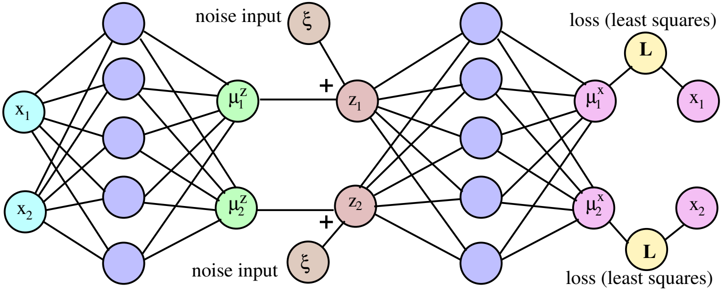
- Finally, minimize negative expectation of ELBO over $p(\boldsymbol{x})$:

$$\min_{\phi,\boldsymbol{\theta}} -E_{p(\boldsymbol{x})} E_{q(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{\theta})}[\log [p(\boldsymbol{x}|\boldsymbol{z},\phi)]] + E_{p(\boldsymbol{x})} \mathrm{D}_{KL}[q(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{\theta}), p(\boldsymbol{z})]$$

- The first term is approximated as

$$E_{p(\boldsymbol{x})} E_{q(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{\theta})}[\log [p(\boldsymbol{x}|\boldsymbol{z},\phi)]] \approx \frac{1}{n} \sum_{i=1}^{n} \log [p(\boldsymbol{x}_i|\boldsymbol{z}_i^*,\phi)] .$$

We assume $p(\boldsymbol{x}_i|\boldsymbol{z}_i^*,\phi) = \mathcal{N}(f_\phi(\boldsymbol{z}_i^*), \sigma^2)$,
where f is implemented via a neural net: $\rightsquigarrow$ **Decoder network**

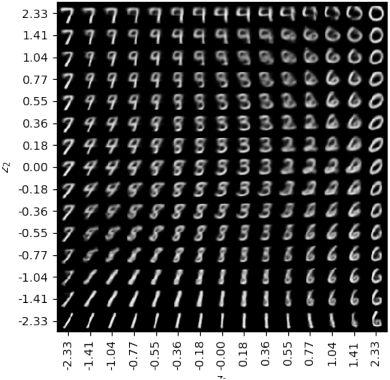


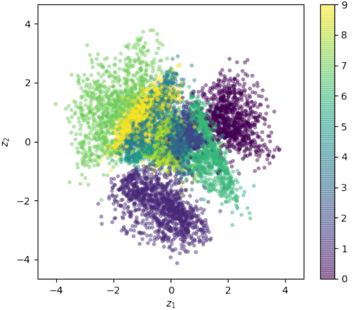
Further Variations

- For maximizing ELBO, we jointly optimize over the parameters of encoder and decoder network.
- When adjusting the decoder, we also change the "true" posterior that we are going to approximate!
- So approximation quality should not be our only goal...
  need "tuning knob" for steering the model into a desired direction.
- Solution: introduce parameter $\beta > 0$ that controls the relative importance of the two loss terms:

$$\min_{\boldsymbol{\theta},\phi} \; \frac{1}{n}\sum_{i=1}^{n} \mathrm{D}_{KL}[q(\boldsymbol{z}|\boldsymbol{x}_i,\boldsymbol{\theta}), p(\boldsymbol{z})] - \beta\frac{1}{n}\sum_{i=1}^{n} \log\left[p(\boldsymbol{x}_i|\boldsymbol{z}_i^*,\phi)\right]$$

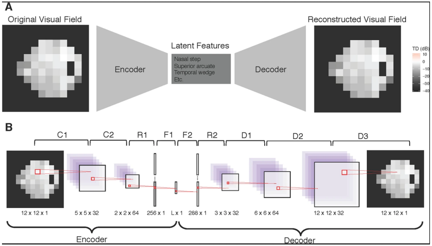
Applications: MNIST example

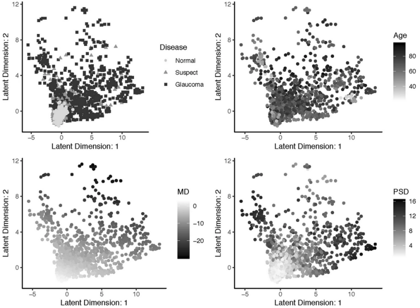




Taken from Louis Tiago: A Tutorial on Variational Autoencoders with a Concise Keras Implementation

Applications: Medical example

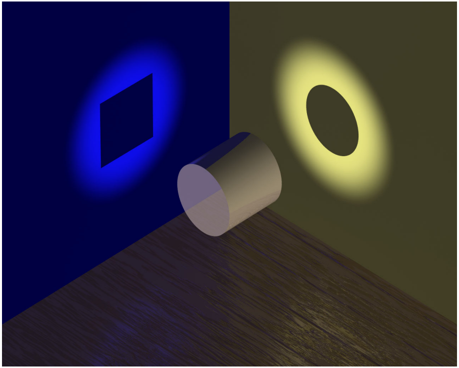




Berchuck, S.I., Mukherjee, S. & Medeiros, F.A. Estimating Rates of Progression and Predicting Future Visual Fields in Glaucoma Using a Deep Variational Autoencoder. Sci Rep 9, 18113 (2019). https://doi.org/10.1038/s41598-019-54653-6

Multiple Views: Deep Information Bottleneck

- Consider paired samples from different views.
- What is the dependency structure between the views ?
- Nonlinear model: dependency detected by deep IB.



Two-view version: The deep information bottleneck

- So far we argued that since the true posterior $p(\boldsymbol{z}|\boldsymbol{x})$ depends on $\boldsymbol{x}$, the variational approximation should also depend on $\boldsymbol{x}$.
- Restricted setting: explain posterior **only by external variable $\tilde{\boldsymbol{x}}$**:

$$q = q(\boldsymbol{z}|\boldsymbol{\theta}, \tilde{\mathbf{x}}).$$

$$\begin{aligned}\mathrm{ELBO}[\boldsymbol{\theta}, \phi] &= \int q(\boldsymbol{z}|\tilde{\mathbf{x}}, \boldsymbol{\theta}) \log\left[p(\boldsymbol{x}|\boldsymbol{z}, \phi)\right] d\boldsymbol{z} - \mathrm{D}_{KL}[q(\boldsymbol{z}|\tilde{\mathbf{x}}, \boldsymbol{\theta}), p(\boldsymbol{z})] \\ &= E_{q(\boldsymbol{z}|\tilde{\mathbf{x}},\boldsymbol{\theta})} \log\left[p(\boldsymbol{x}|\boldsymbol{z}, \phi)\right] - \mathrm{D}_{KL}[q(\boldsymbol{z}|\tilde{\mathbf{x}}, \boldsymbol{\theta}), p(\boldsymbol{z})]\end{aligned}$$

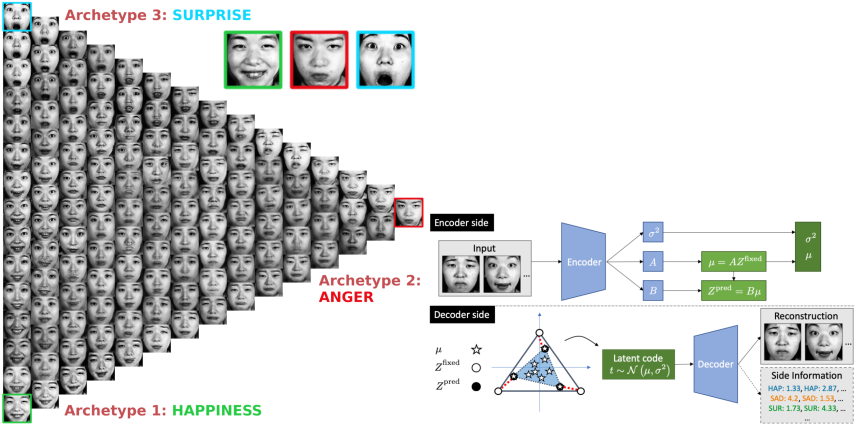
- Connection to IB:
  - Assume (or define) $q(\boldsymbol{z}|\tilde{\mathbf{x}}, \boldsymbol{\theta}) := p(\boldsymbol{z}|\tilde{\mathbf{x}}, \boldsymbol{\theta})$
  - Take expectation w.r.t. joint data distribution $p(\tilde{\mathbf{x}}, \boldsymbol{x})$:

$$E_{p(\tilde{\mathbf{x}},\boldsymbol{x})} E_{p(\boldsymbol{z}|\tilde{\mathbf{x}},\boldsymbol{\theta})} \log\left[p(\boldsymbol{x}|\boldsymbol{z}, \phi)\right] - E_{p(\tilde{\mathbf{x}})} \mathrm{D}_{KL}[p(\boldsymbol{z}|\tilde{\mathbf{x}}, \boldsymbol{\theta}), p(\boldsymbol{z})]$$

  - First term $\leq \mathcal{I}_{\boldsymbol{\theta},\phi}(\boldsymbol{z}; \boldsymbol{x}) + const$. Second term $= \mathcal{I}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}; \boldsymbol{z})$,
- This defines the deep information bottleneck (with weight β)

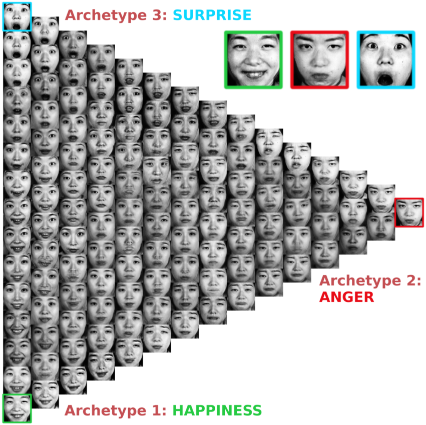
$$\min_{\phi,\boldsymbol{\theta}} \mathcal{I}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}; \boldsymbol{z}) - \beta \mathcal{I}^{\text{low}}_{\boldsymbol{\theta},\phi}(\boldsymbol{z}; \boldsymbol{x}), \quad \text{where } \mathcal{I}^{\text{low}} \text{ is a lower bound of } \mathcal{I}.$$

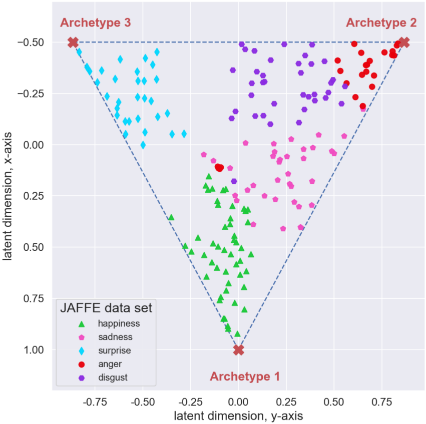
Applications: Face images



Keller et al. 2020: Learning Extremal Representations with Deep Archetypal Analysis

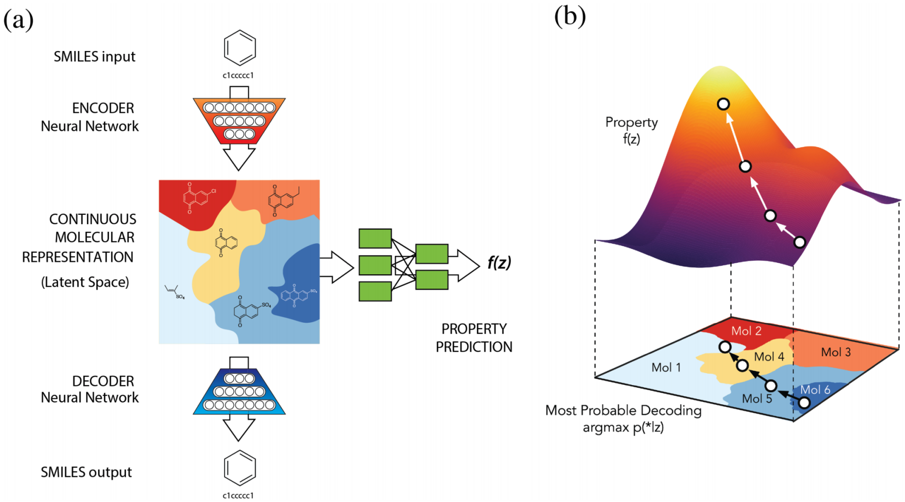
Applications: Face images





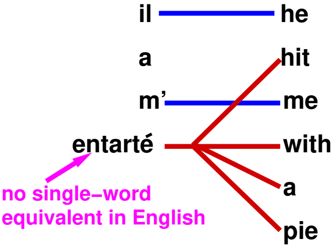
Keller et al. 2020: Learning Extremal Representations with Deep Archetypal Analysis

Applications: Deep Chemical Variational Autoencoders

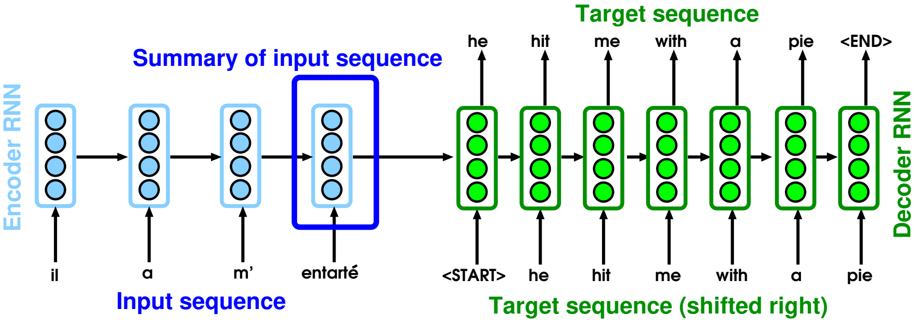


(Gomez-Bombarelli et al., ACS Cent Sci, 2018)

Neural Encoder/Decoder Models for Machine Translation



https://commons.wikimedia.org/w/index.php?curid=9105527



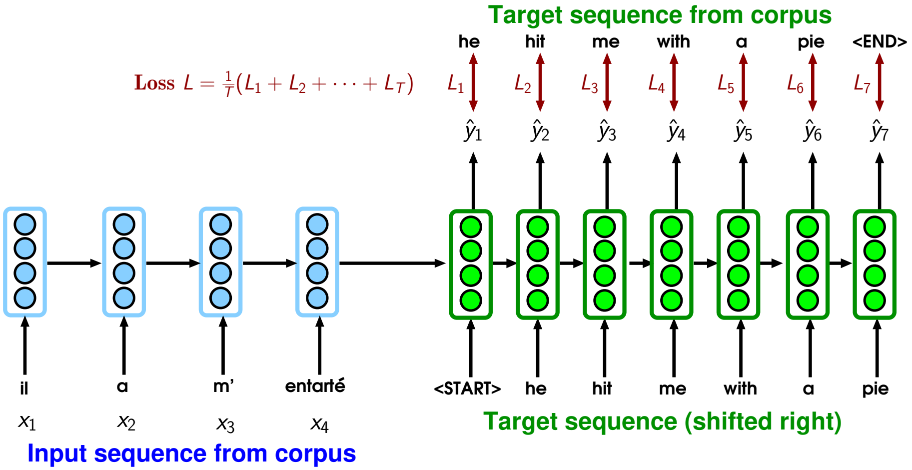
Neural Machine Translation (NMT)

- The sequence-to-sequence model is an example of a **Conditional Language Model:**
  - **Language Model** because the decoder is predicting the next word of the target sentence $\boldsymbol{y} = \{y_1, y_2, \ldots, y_T\}$
  - **Conditional** because its predictions are also conditioned on the source sentence $\boldsymbol{x}$.
- NMT directly calculates

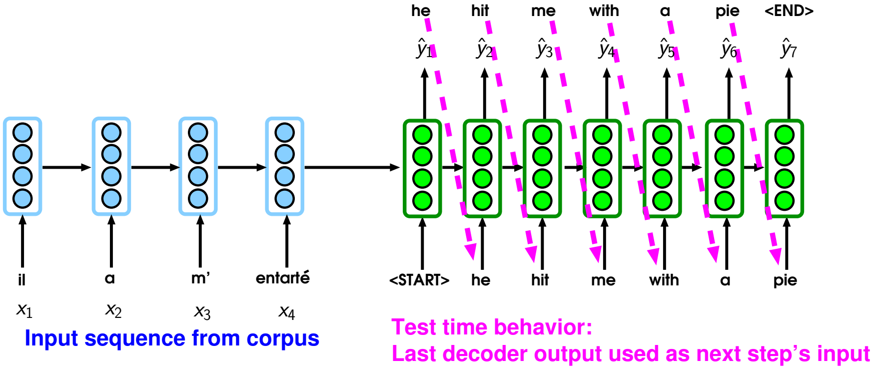
$$P(\boldsymbol{y}|\boldsymbol{x}) = P(y_1|\boldsymbol{x}) \cdot P(y_2|y_1, \boldsymbol{x}) \cdots P(y_T|y_{1:(T-1)}, \boldsymbol{x})$$

  One term is the probability of the next target word, given all target words so far and the input sequence.
- How to train a NMT system? Get a **big parallel corpus** containing input/target sequence pairs!
- Seq2seq is optimized as a **single system.**
  Backpropagation operates **end-to-end.**

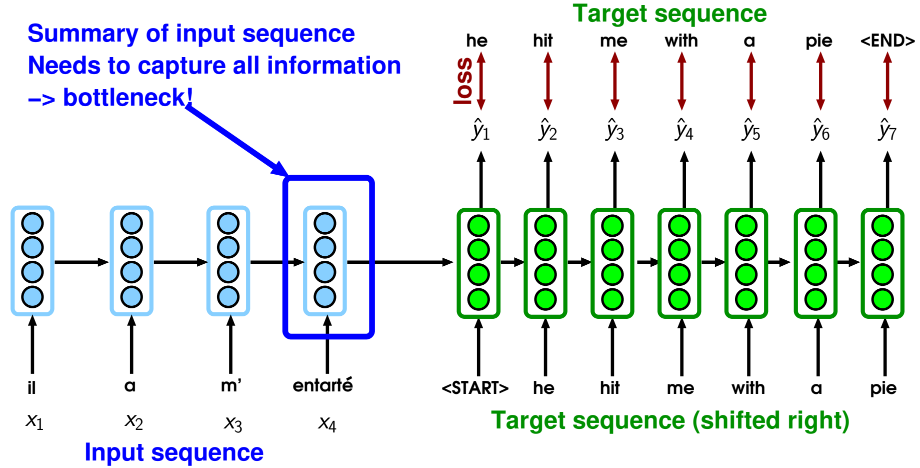
Sequence-to-sequence: Training



Sequence-to-sequence: Test time behavior

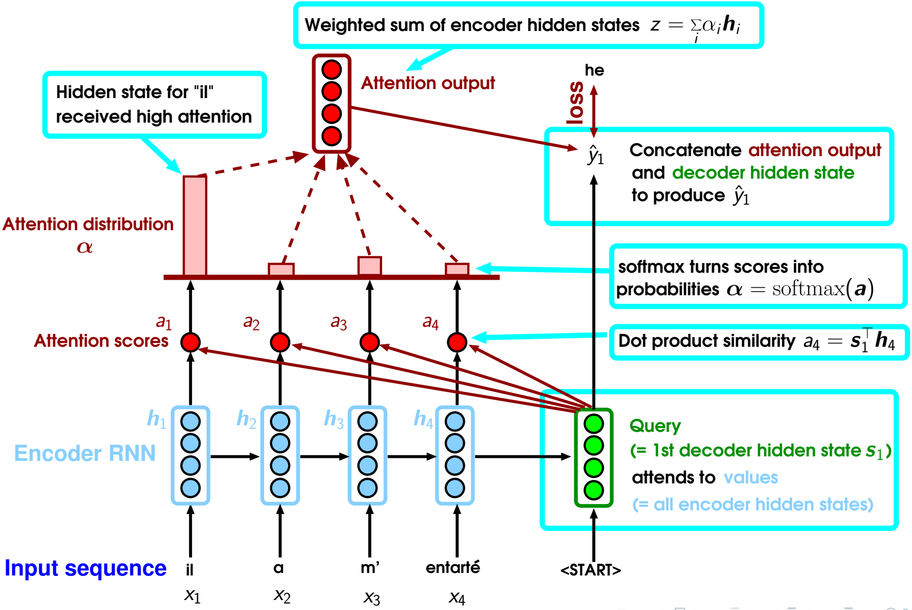


Sequence-to-sequence: bottleneck problem

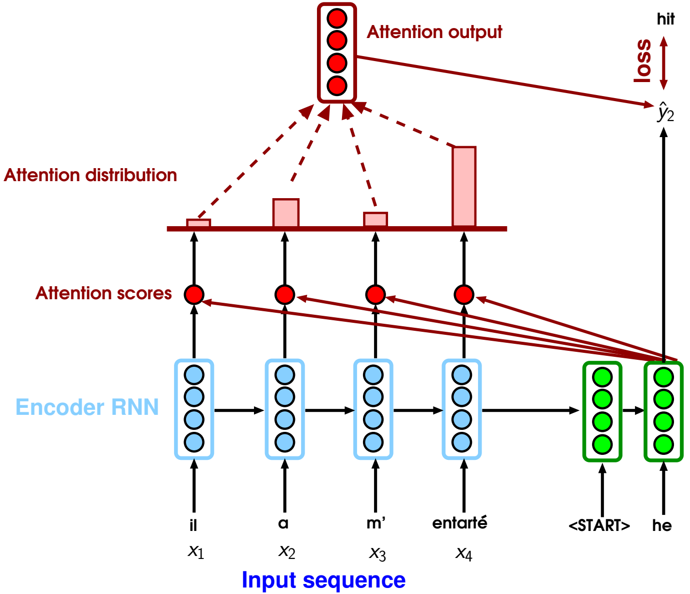


Idea: allow the decoder to look directly at input, bypass bottleneck.

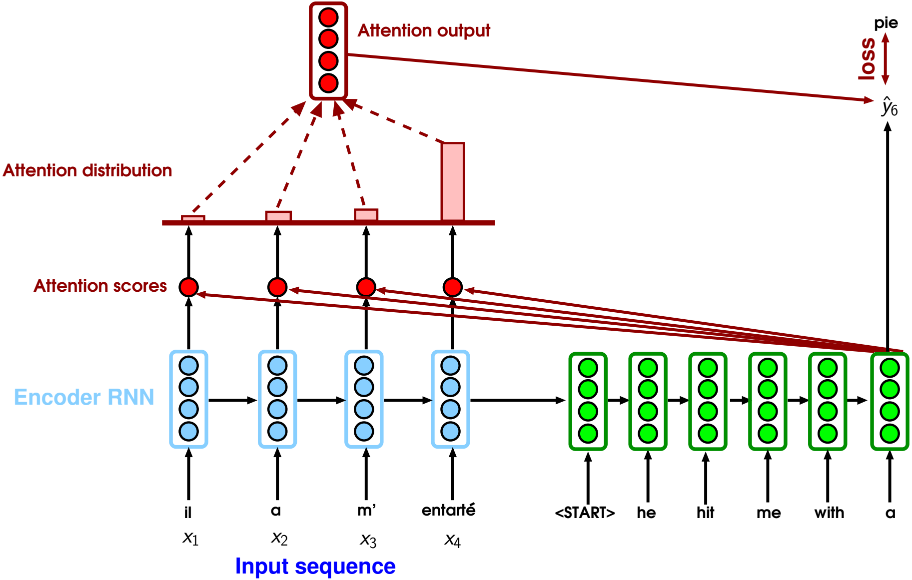
Sequence-to-sequence with attention



Sequence-to-sequence with attention



Sequence-to-sequence with attention

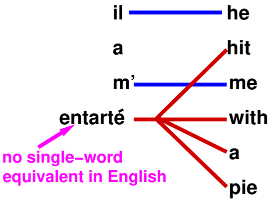


Sequence-to-sequence with attention

- We have encoder hidden states $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N \in \mathbb{R}^h$ ($\rightsquigarrow$ **values**)
- On timestep t, we have decoder hidden state $\boldsymbol{s}_t \in \mathbb{R}^h$ ($\rightsquigarrow$ **query**)
- We get the attention scores for this step (query/value similarities):
$$\boldsymbol{a}^t = (\boldsymbol{s}_t^\top \boldsymbol{h}_1, \ldots, \boldsymbol{s}_t^\top \boldsymbol{h}_N) \in \mathbb{R}^N$$
- We take softmax to get the attention distribution $\boldsymbol{\alpha}^t = \mathrm{softmax}(\boldsymbol{a}^t) \in \mathbb{R}^N$.
- We use $\boldsymbol{\alpha}^t$ to take a weighted sum of the encoder hidden states to get the attention output $\boldsymbol{z}_t = \sum_{i=1}^N \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$
- Finally we concatenate the attention output with the decoder hidden state $[\boldsymbol{z}_t; \boldsymbol{s}_t]$ and preceded as usual:
  - Use this to generate the probability of the next target word, given all target words so far and input sequence. Example: use a MLP with softmax output for generating $P(y_t|y_{1:(t-1)}, \boldsymbol{x})$.
  - predict next word as $\hat{y}_t = \arg\max P(y_t|y_{1:(t-1)}, \boldsymbol{x})$.

Attention is great

- Attention solves the bottleneck problem: It allows decoder to look directly at the input sequence, **bypass bottleneck**
- Attention helps with vanishing gradient problem: Provides **shortcut to faraway states**
- **Interpretability:** Attention distribution provides (probabilistic) word alignments for free!
- We never explicitly trained an alignment system, **the network learned it by itself!**



- **Attention is also the main building block of transformers:** We "transform" queries $\boldsymbol{s}_t$ to attention outputs $\boldsymbol{z}_t$, conditioned on inputs and previous queries.
- To understand this better, we first generalize the idea.

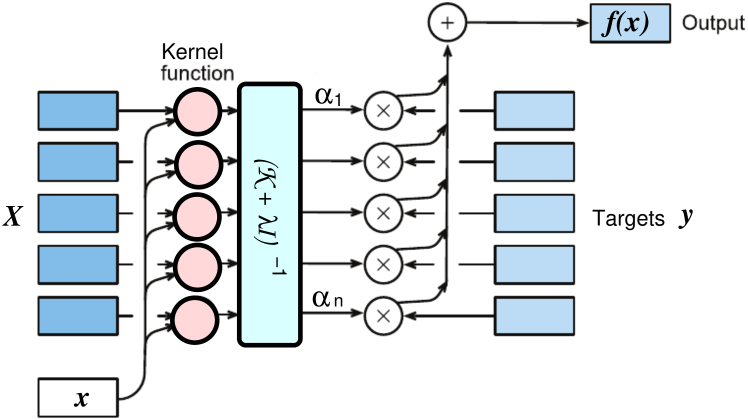
Attention: General setting

- We can better understand attention by comparing it to **kernel ridge regression (KRR).**
- In KRR, we compare input ("query") $\boldsymbol{x} \in \mathbb{R}^d$ to each of the training examples $X = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_n)$ using a **kernel function** $\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}')$ to get a vector of **similarity scores** $\boldsymbol{\alpha} = \boldsymbol{\alpha}(\boldsymbol{x}, X)$.
- We then use this to retrieve a weighted combination of the target values $\boldsymbol{y}_i \in \mathbb{R}^{d_v}$ to compute the **predicted output:** $\boldsymbol{z} = \sum_{i=1}^n \alpha_i \boldsymbol{y}_i$**.**
- KRR example: one-dimensional targets, i.e. $d_v = 1$. Given kernel function $\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}')$, the predicted regression output for query $\boldsymbol{x}$ is

$$z := f(\boldsymbol{x}) = \underbrace{\mathcal{K}_{\boldsymbol{x}}^t (\mathcal{K}(X, X) + \lambda \boldsymbol{I})^{-1}}_{\boldsymbol{\alpha}^t} \boldsymbol{y} = \sum_{i=1}^n \alpha_i(\boldsymbol{x}, X) y_i.$$

- Intuitively, $\boldsymbol{\alpha}(\boldsymbol{x}, X)$ measures how well the query $\boldsymbol{x}$ is aligned with the examples in the training set X.

Attention: General setting



KRR as input-output mapping. Adapted from Figure 16.6 in K.Murphy: Probabilistic Machine Learning, Advanced Topics. MIT Press, 2023.

A differentiable and parametric version

- Replace X with a **learned embedding**, to create a set of **keys,** $K = XW_k \in \mathbb{R}^{n \times d_k}$.
- Replace Y, to create a set of **values,** $V = YW_v \in \mathbb{R}^{n \times d_v}$.
- Embed input to create a **query,** $\boldsymbol{q} = W_q \boldsymbol{x} \in \mathbb{R}^{d_k}$.
- Parameters to be learned: the three embedding matrices.
- Replace similarity scoring function with a **soft attention layer:** Define the weighted output for query $\boldsymbol{q}$ to be

$$\boldsymbol{z} := \mathrm{Attn}(\boldsymbol{q}, (\boldsymbol{k}_1, \boldsymbol{v}_1), \ldots, (\boldsymbol{k}_n, \boldsymbol{v}_n)) = \sum_{i=1}^{n} \alpha_i(\boldsymbol{q}, K)\boldsymbol{v}_i$$

  $\alpha_i(\boldsymbol{q}, K)$ is i'th attention weight, satisfying $0 \le \alpha_i \le 1$ and $\sum_i \alpha_i = 1$.
- The $\alpha_i(\boldsymbol{q}, K)$ are computed from an **attention score** $a(\boldsymbol{q}, \boldsymbol{k}_i) \in \mathbb{R}$, that computes the similarity of query $\boldsymbol{q}$ to key $\boldsymbol{k}_i$.
- Example: (scaled) dot product attention $a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{q}^t \boldsymbol{k} / \sqrt{d_k}$.
- Given the scores, compute attention weights:

$$\alpha_i(\boldsymbol{q}, \boldsymbol{k}_{1:n}) = \mathrm{softmax}_i([a(\boldsymbol{q}, \boldsymbol{k}_1), \ldots, a(\boldsymbol{q}, \boldsymbol{k}_n)]).$$

Attention: General setting

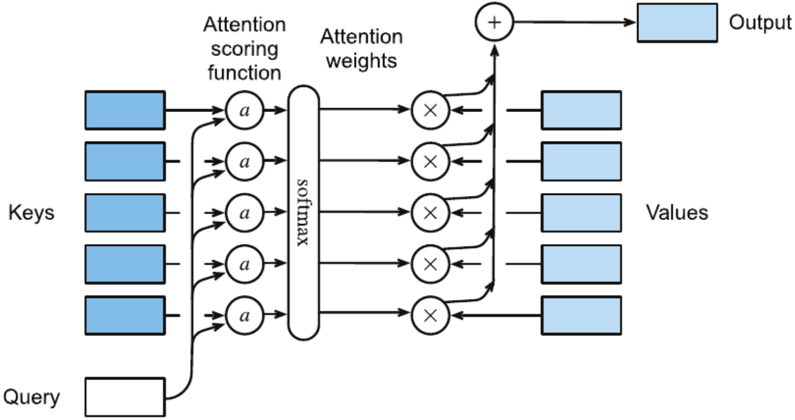
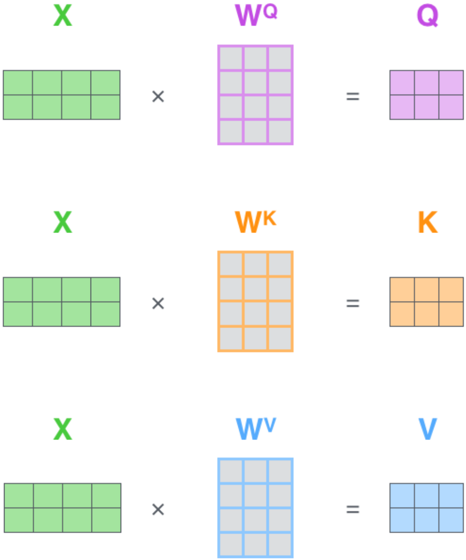
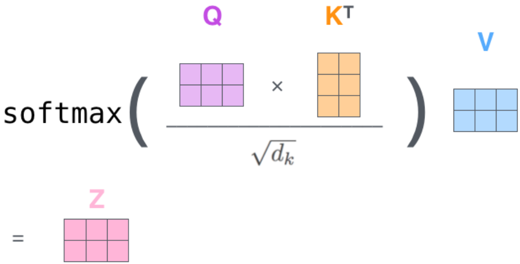


Figure 16.6 in K.Murphy: Probabilistic Machine Learning, Advanced Topics. MIT Press, 2023. Attention layer. (a) Mapping a single query q to a single output, given a set of keys and values.



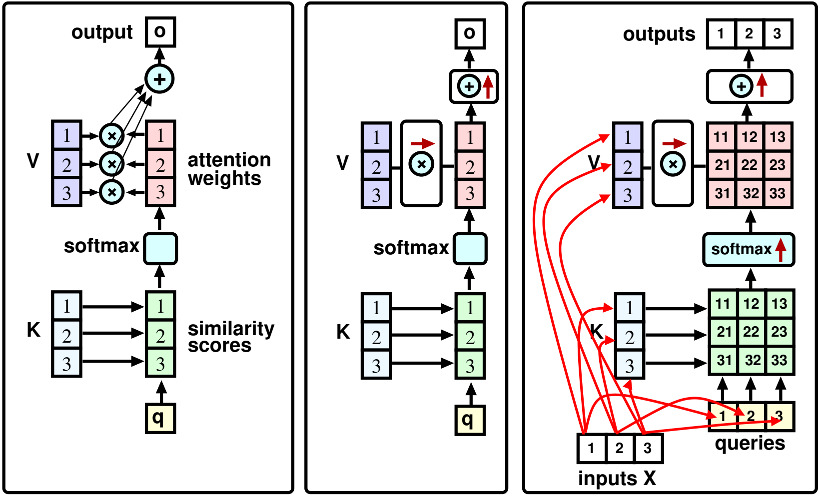
Matrix Calculation of Self-Attention. Taken from https://jalammar.github.io/illustrated-transformer/

Attention



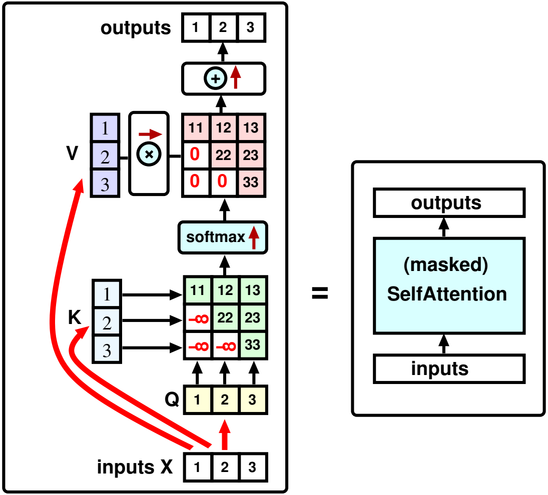
Matrix Calculation of Self-Attention. Taken from https://jalammar.github.io/illustrated-transformer/

Attention and Self-Attention



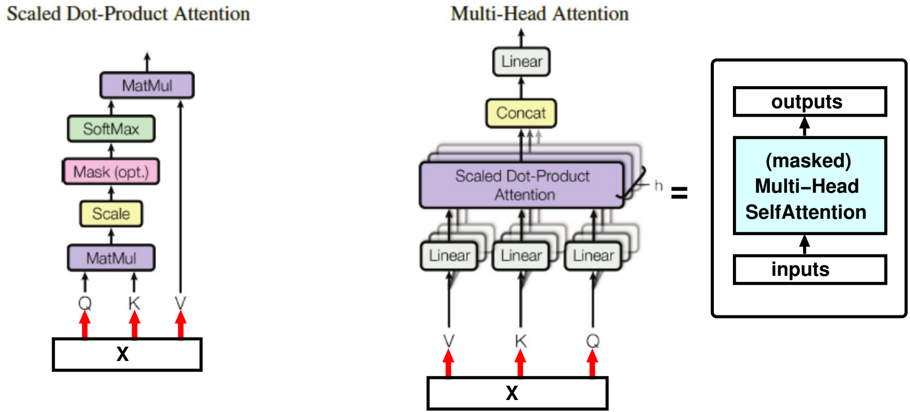
Left: Mapping a single query q to a single output o, given a set of keys and values. Middle: simplified notation. Right: Mapping multiple queries to multiple outputs, either for given values and keys (without the red arrows and without inputs X), or in the self-attention case, where queries, values and keys are functions of inputs X (red arrows) .

Masked Attention



Masked self-attention layer: Prevent vectors from looking at future vectors by setting similarity scores to $-\infty$.

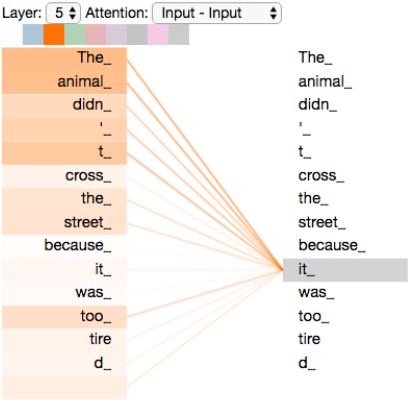
Multi-head Attention



Adapted from Figure 16.7: Left: (Masked) Scaled dot-product (self-)attention. Right: (Masked) Multi-head (self-)attention.

Self-Attention in Language Models

The animal didn't cross the street because it was too tired



https://jalammar.github.io/illustrated-transformer/

Warning: When using Multi-head self-attention, the results are often difficult to interpret...

"Transformer"-language models

- RNNs process one token at a time $\rightsquigarrow$ representation of a word at location t depends on hidden state s_t (a summary of previous words).
- Alternative approach: use attention to compute representation directly as a function of all other words.
- This is the idea of a **encoder-only transformer**, used by LMs such as BERT (Bidirectional Encoder Representations from Transformers).

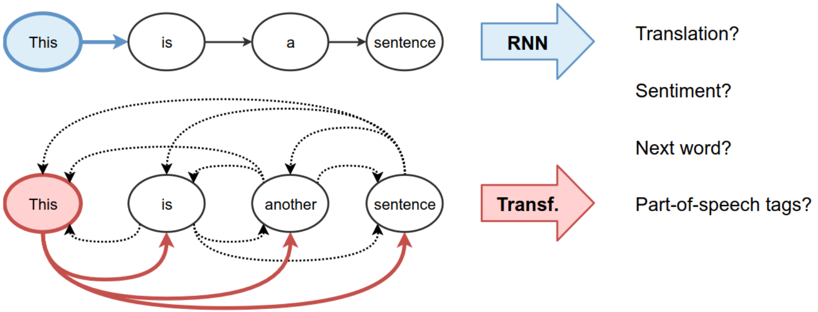


Fig. 16.16 in K.Murphy: Probabilistic Machine Learning, Advanced Topics. MIT Press, 2023.
Original Image published in C. Joshi. Transformers are Graph Neural Networks. Tech. rep. 2020.

"Transformer"-language models

- Alternative: **Decoder-only transformer:** each output y_t only attends to all previously generated outputs, $y_{1:(t-1)}$.
- This can be implemented using **masked self-attention**, and is useful for **generative language models**, such as GPT.
- Combination: **Sequence-to-sequence models**, $p(y_{1:T_y} | x_{1:T_x})$.

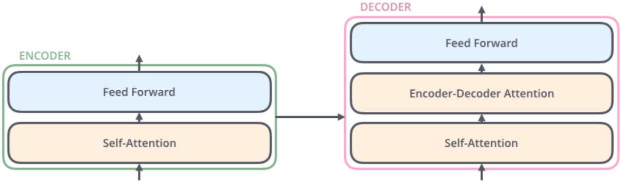
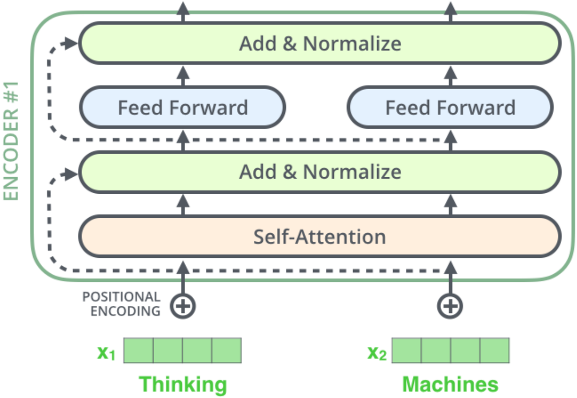


Figure 16.1 in the supplement of K.Murphy: Probabilistic Machine Learning, Advanced Topics. MIT Press, 2023. High level structure of the encoder-decoder transformer architecture. https://jalammar.github.io/illustrated-transformer/

Transformer: Encoder



https://jalammar.github.io/illustrated-transformer/

Scale issues, Normalization and Feed Forward Layer

- MHSA often produces features at **different scales or magnitudes:**
  - attention weights can be very different (i.e. sharp or distributed)
  - combining multiple attention heads makes this even more problematic.
- **Solution 1: add a normalization layer**
- **Solution 2: add a word-wise feed forward MLP** that updates the representation of the i-th word:

  $$h_i \leftarrow \text{Norm}(\text{FeedFwd}(\text{Norm}(h_i))).$$

  It seems that re-scaling the feature vectors independently of each other helps to overcome remaining scaling issues...
- Important role of the **residual connections:** allow the positional information to propagate to higher layers.

Transformer: Encoder

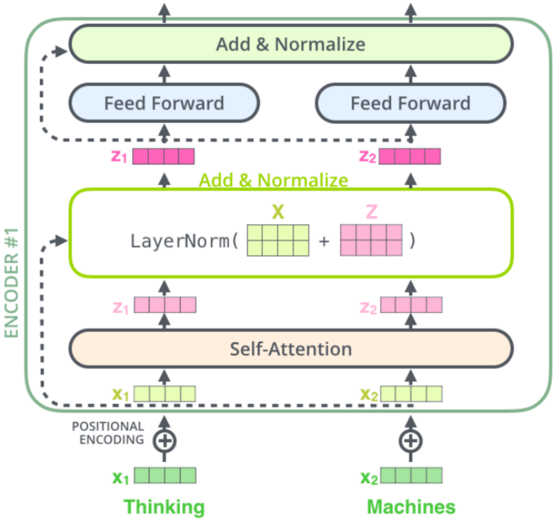
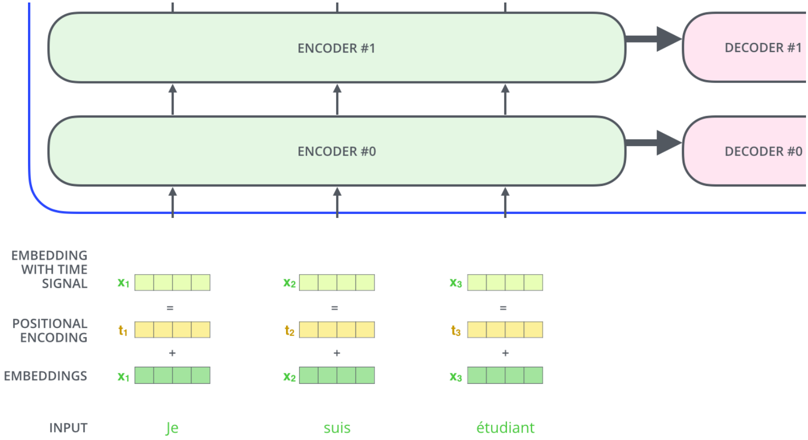


Figure 16.2 in the supplement of K.Murphy: Probabilistic Machine Learning, Advanced Topics. MIT Press, 2023. The encoder block of a transformer for two input tokens. https://jalammar.github.io/illustrated-transformer/

Transformer: Positional Encoding

- Processing of the feature vectors for computing query, key and value **occurs in parallel.**
- The order information between the words **is not known** anywhere inside the attention block.
- But without it, building **contextually rich embeddings** might be impossible! Example:
  1. The man drove the woman to the store.
  2. The woman drove the man to the store.
- The order information has to be modeled $\rightsquigarrow$ **positional encodings.**

Transformer: Positional Encoding



https://jalammar.github.io/illustrated-transformer/

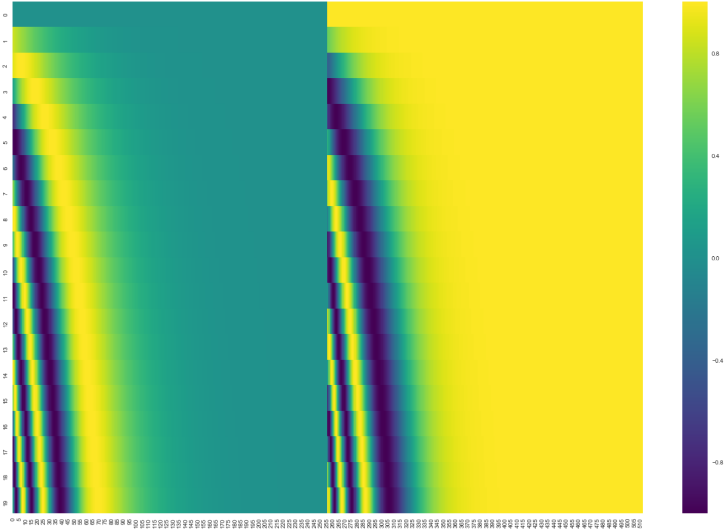
Transformer: Positional Encoding

$$\boldsymbol{p}_t = \begin{bmatrix} \sin(\omega_1.t) \\ \cos(\omega_1.t) \\ \\ \sin(\omega_2.t) \\ \cos(\omega_2.t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2}.t) \\ \cos(\omega_{d/2}.t) \end{bmatrix}_{d\times 1}$$

Intuition: binary representation:

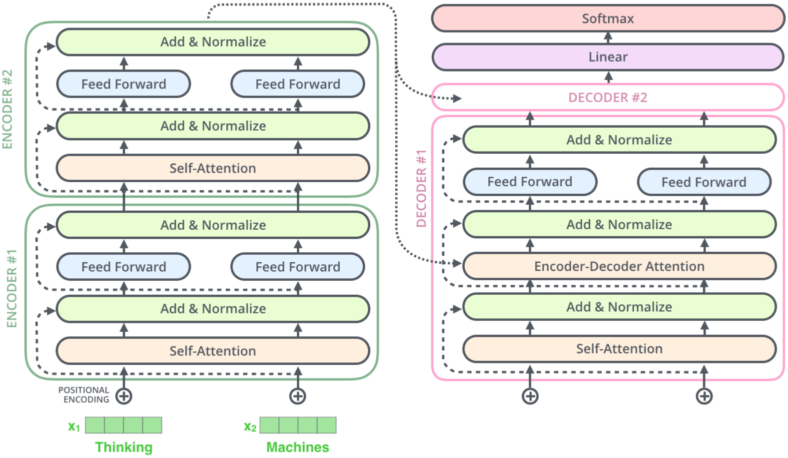
0 :	0	0	0	0	8 :	1	0	0	0
1 :	0	0	0	1	9 :	1	0	0	1
2 :	0	0	1	0	10 :	1	0	1	0
3 :	0	0	1	1	11 :	1	0	1	1
4 :	0	1	0	0	12 :	1	1	0	0
5 :	0	1	0	1	13 :	1	1	0	1
6 :	0	1	1	0	14 :	1	1	1	0
7 :	0	1	1	1	15 :	1	1	1	1

Transformer: Positional Encoding



https://jalammar.github.io/illustrated-transformer/

Seq2Seq with Transformers



https://jalammar.github.io/illustrated-transformer/