Building blocks of modern machine learning: (self-)attention and diffusion models

Joscha Diehl

July 31, 2023

Contents

For coding tutorials we will work with `tensorflow`, just for the sake of concreteness (and since several groups in Greifswald are using it). But the concepts taught are of course independent of the particular framework. Moreover, currently available frameworks (`pytorch` and `JAX`/`Flax` are other examples) are so similar, that switching between them is very easy.

Initial remarks:

- We *will* get our hands dirty, and implement (again and again) the mechanisms we talk about. We will not be too interested in the (very successful) large pipelines; instead we will prefer to probe and take apart small models.
- The machine learning content will be interspersed with various kinds of rigorous mathematics. It is good to want to understand all of this in detail, and I am happy to answer all questions. Just note that sometimes it pays off to take quoted results, momentarily, for granted. (This will probably happen mostly in the section on diffusion models.)
- Text in blue is generated with the help of GPT4 (with possible minor edits by me). If meaningful, a link to the prompt is given.

Notation:

- $\mathbb{R}$ for the set of real numbers, elements are usually written in standard text (e.g. $x, y, ..$)
- $\mathbb{R}_{>0}$ for the set of positive real numbers
- $\mathbb{N} = \{0, 1, \dots\}$ for the set of natural numbers
- $\mathbb{N}_{\geq 1} = \{1, \dots\}$ for the set of positive natural numbers
- $\mathbb{Z}$ for the set of integers
- $\mathbb{R}^d$ for the set of d-dimensional vectors, elements are usually written in bold (e.g. $\mathbf{x}, \mathbf{y}, ..$)
- $|x|$ for the absolute value of $x \in \mathbb{R}$
- $||\mathbf{x}|| := \sqrt{\sum_{i=1} x_i^2}$ for the Euclidean norm of $\mathbf{x} \in \mathbb{R}^d$.

1 Neural networks

A **neural network** is nothing but a parametrized function

$$f_\theta : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}},$$

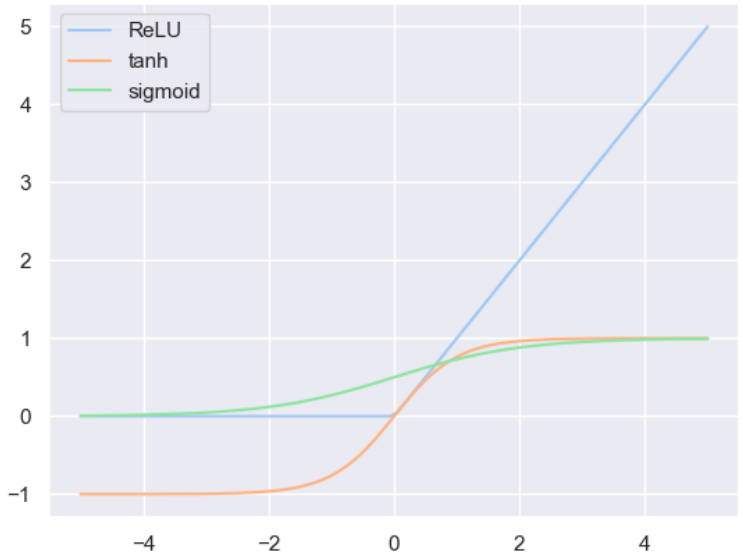
where $\theta \in \mathbb{R}^p$ are the **parameters** or **weights** of the network.

To really warrant the name "neural network" it is usually built by **composing** very simple functions. Important ingredients are

- **affine maps** [1] for $A \in \mathbb{R}^{n \times m}$ (also called a **kernel**), $\mathbf{b} \in \mathbb{R}^n$ (also called a **bias term**), this is a map

$$\begin{aligned} \mathbb{R}^m &\to \mathbb{R}^n \\ \mathbf{x} &\mapsto A\mathbf{x} + \mathbf{b}. \end{aligned}$$

- **activation functions**; this is any kind of nonlinear, one-dimensional, function, e.g.



  - (rectifier linear unit)

$$\begin{aligned} \text{ReLU} : \mathbb{R} &\to \mathbb{R} \\ x &\mapsto 1_{[0,\infty)}(x) \cdot x = \max\{0, x\}. \end{aligned}$$

[1] Often misnamed **linear maps**; they are linear only if the bias term is zero.

–

$$\tanh : \mathbb{R} \to \mathbb{R}.$$

–

$$\begin{aligned} \text{sigmoid} : \mathbb{R} &\to \mathbb{R} \\ x &\mapsto \frac{1}{1+e^{-x}}. \end{aligned}$$

–

$$\begin{aligned} \mathsf{GeLU} : \mathbb{R} &\to \mathbb{R} \\ x &\mapsto x\mathbb{P}_{\mathcal{N}(0,1)}[X \leq x] = x\frac{1}{2}\left(1 + \operatorname{erf}(\frac{x}{\sqrt{2}})\right). \end{aligned}$$

–

$$\begin{aligned} \mathsf{ELU} : \mathbb{R} &\to \mathbb{R} \\ x &\mapsto \begin{cases} x & x > 0 \\ \exp(x) - 1 & x \leq 0. \end{cases} \end{aligned}$$

Remark 1.1. *We will mostly use the ReLU function.*[2]

Pros:

- *Empirical evidence. ("Deep CNNs with ReLUs train several times faster than their equivalents with tanh units." [KSH17])*
- *"creating sparse representations with true zeros" [GBB11]*
- *"folding" / changing the topology of input data [NZL20]*
- *(restricted) homogeneity: for all $\alpha > 0, x \in \mathbb{R}$,*

$$\operatorname{ReLU}(\alpha x) = \alpha \operatorname{ReLU}(x)$$

Cons:

- *If all units are non-active (i.e. the input is smaller than 0), gradient descent will not work. ("dead neurons")*
- *Discontinuous derivative at 0. This is usually* not *a problem (since one never "hits" zero).*
- ReLU *is unbounded which can lead to instabilities and/or overconfidence. [HAB19]*

[2] Other activation functions are in use, in particular in the last layer; more on this later.

We now build an example of a neural network. First, for $A \in \mathbb{R}^{n\times m}, \mathbf{b} \in \mathbb{R}^n$, (so here, $\theta = (A, \mathbf{b})$) define

$$\begin{aligned} \mathrm{Dense}'_{A,\mathbf{b}} : \mathbb{R}^m &\to \mathbb{R}^n \\ \mathbf{x} &\mapsto A\mathbf{x} + \mathbf{b}. \end{aligned}$$

Problem: if we stack this layer, for example with $A_1 \in \mathbb{R}^{n_1\times m}, \mathbf{b}_1 \in \mathbb{R}^{n_1}, A_2 \in \mathbb{R}^{n\times n_1}, \mathbf{b}_2 \in \mathbb{R}^n$, we get

$$\begin{aligned} \mathrm{Dense}'_{A_2,\mathbf{b}_2} \circ \mathrm{Dense}'_{A_1,\mathbf{b}_1} : \mathbb{R}^m &\to \mathbb{R}^n \\ \mathbf{x} &\mapsto A_2A_1\mathbf{x} + A_2\mathbf{b}_1 + \mathbf{b}_2, \end{aligned}$$

which is again an affine map! **We need to introduce some non-linearity to make compositions of layers interesting.** Define

$$\begin{aligned} \mathrm{Dense}_{A,\mathbf{b}} := \mathrm{Dense}_{A,\mathbf{b};\mathrm{ReLU}} : \mathbb{R}^m &\to \mathbb{R}^n \\ \mathbf{x} &\mapsto \mathrm{ReLU}(A\mathbf{x} + \mathbf{b}). \end{aligned} \tag{1}$$

Here, ReLU is applied *entrywise*. Spelled out in coordinates this is,

$$\mathrm{Dense}_{A,\mathbf{b}}(\mathbf{x}) = \begin{pmatrix} \mathrm{ReLU}\left(\sum_{j=1}^m A_{1j}x_j + b_1\right) \\ \mathrm{ReLU}\left(\sum_{j=1}^m A_{2j}x_j + b_2\right) \\ \cdots \\ \mathrm{ReLU}\left(\sum_{j=1}^m A_{nj}x_j + b_n\right) \end{pmatrix}.$$

Now we can stack, to get interesting functions.

Example 1.2. $d_{\mathrm{in}} = 2, d_1 = 5, d_{\mathrm{out}} = 1$, *Note the annoying fact that in figures, the input is usually on the left, and the output on the right, whereas in mathematical formulas we usually write the input on the right.*

https://github.com/diehlj/2023-building-blocks-lecture/blob/master/examples/nn251.py

Of course we can use more layers,

$$\mathrm{Dense}_{A_n,\mathbf{b}_n} \circ \ldots \circ \mathrm{Dense}_{A_1,\mathbf{b}_1}.$$

The number n is also called the **depth** of such a network. The last (outermost) layer, $\mathrm{Dense}_{A_n,\mathbf{b}_n}$, is also called **output layer**. It often uses a different activation function, e.g. the sigmoid function or the identity function. The other layers, $\mathrm{Dense}_{A_i,\mathbf{b}_i}$, $i = 1, \ldots, n-1$, are **hidden layers**. The **depth** of such a network is $n - 1$.

The **width** of a layer $\mathrm{Dense}_{A,b}$ is the dimension of the codomain. The **width** of a network is the maximal width of its layers.

Since the matrices A are arbitrary, in a graphic representation like in Figure 1 all nodes in neighboring layers are connected. Such neural networks are said to be **fully connected** and are also called **multilayer perceptrons**.

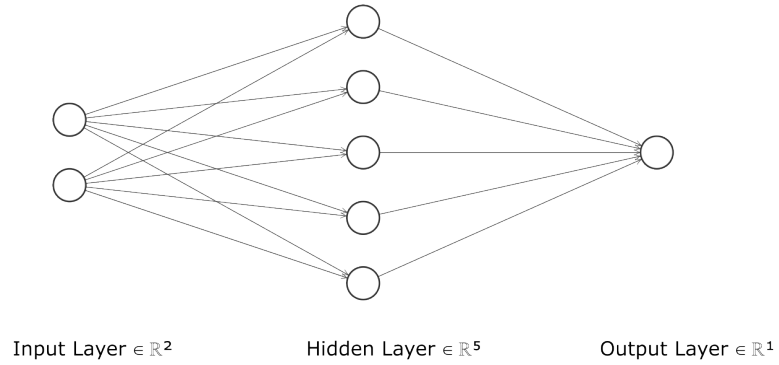


Figure 1: *A neural network with* $d_{\text{in}} = 2, d_{inner} = 5, d_{\text{out}} = 1$.

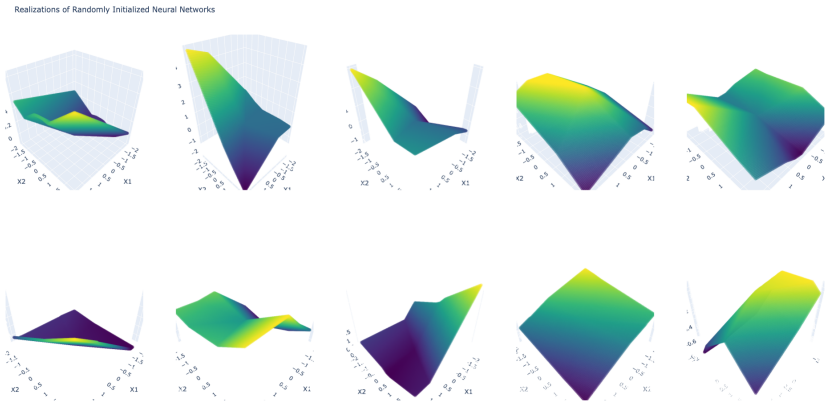


Figure 2: *Example realizations of a neural network with* $d_{\text{in}} = 2, d_{inner} = 5, d_{\text{out}} = 1$.

1.1 Expressiveness

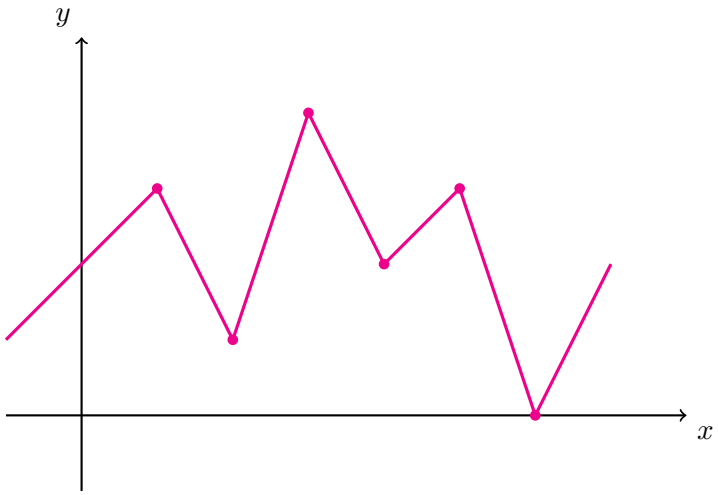
Let us consider the following network,

$$\text{Dense}_{A_2,\mathbf{b}_2;\text{id}} \circ \text{Dense}_{A_1,\mathbf{b}_1;\text{ReLU}} : \mathbb{R} \to \mathbb{R}, \tag{2}$$

with one hidden layer (with ReLU activation) and one output layer (with identity activation).

What functions can this network compute?

Lemma 1.3. *A continuous function* $f : \mathbb{R} \to \mathbb{R}$ *is* ***piecewise linear*** *if there exist*



$t_0 < t_1 < \cdots < t_p$ *and linear maps* $L_0, \ldots, L_p, L_{p+1}$ *such that*

$$f(x) = \begin{cases} L_0(x), & x \in (-\infty, t_0] \\ L_\ell(x), & x \in [t_{\ell-1}, t_\ell] \\ L_{p+1}(x), & x \in [t_p, +\infty). \end{cases}$$

For every such f *there exists a neural network* (2) *(with one hidden layer with ReLU activation and one linear output layer) that is equal to* f*. In other words, there exist* n*,* $a^{(1)}, a^{(2)}, b_i^{(1)}, b^{(2)} \in \mathbb{R}$ *such that*

$$f(x) = \sum_{i=1}^{n} a_i^{(2)} \operatorname{ReLU}(a_i^{(1)} x + b_i^{(1)}) + b^{(2)}.$$

Remark 1.4. *This is* not *true for neural networks with input dimension* 2*, i.e. there are piecewise linear functions* $\mathbb{R}^2 \to \mathbb{R}$ *that cannot be represented by a neural network with one hidden layer (ReLU activation) and one output layer (identity activation). See [Fou22, Theorem 24.1].*

Proof. Write

$$L_\ell(x) = \lambda_\ell x + c_\ell.$$

Then

$$f(x) = \lambda_0 \, (x \wedge t_0) + c_0 + \sum_{\ell=1}^{p-1} \lambda_\ell \, \big((0 \vee (x - t_\ell)) \wedge (t_{\ell+1} - t_\ell)\big) + \lambda_p \, \big(0 \vee (x - t_p)\big) \, .$$

Note that this uses the fact that $L_{\ell-1}(t_{\ell-1}) = L_\ell(t_{\ell-1}), \ell = 1, \ldots, p+1$.

Then, using

$$\lambda_\ell \left((0 \vee (x - t_\ell)) \wedge (t_{\ell+1} - t_\ell)\right) = \lambda_\ell \operatorname{ReLU}(x - t_\ell) - \lambda_\ell \operatorname{ReLU}(x - t_{\ell+1}),$$

we get that this is equal to

$$\begin{aligned}
&\lambda_0 x + c_0 + \sum_{\ell=1}^{p+1} (\lambda_\ell - \lambda_{\ell-1}) \operatorname{ReLU}(x - t_\ell) \\
&\quad = \operatorname{ReLU}(\lambda_0 x) - \operatorname{ReLU}(-\lambda_0 x) + c_0 + \sum_{\ell=1}^{i-1} (\lambda_\ell - \lambda_{\ell-1}) \operatorname{ReLU}(x - t_\ell),
\end{aligned}$$

as desired. □

As a corollary we obtain a special case of the following theorem.

Theorem 1.5 (Universal approximation theorem; arbitrary width; ReLU activation)**.** [3] *Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a continuous function. Then the functions*

$$\operatorname{Dense}_{A_2, \mathbf{b}_2; \mathrm{id}} \circ \operatorname{Dense}_{A_1, \mathbf{b}_1; \sigma},$$

are dense in the space of continuous functions on compact sets (under the supremum norm) if and only if σ is not *a polynomial function.*

This means that for all continuous functions $g : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$, every $\epsilon > 0$, and every compact set $K \subset \mathbb{R}^{d_{\text{in}}}$, there exist $d_{\text{hidden}} \in \mathbb{N}_{\geq 1}$ and weights $A_1 \in \mathbb{R}^{d_{\text{hidden}} \times d_{\text{in}}}, A_2 \in \mathbb{R}^{d_{\text{out}} \times d_{\text{hidden}}}, \mathbf{b}_1 \in \mathbb{R}^{d_{\text{hidden}}}, \mathbf{b}_2 \in \mathbb{R}^{d_{\text{out}}}$ such that

$$\sup_{x \in K} ||g(x) - \operatorname{Dense}_{A_2, \mathbf{b}_2; \mathrm{id}} \circ \operatorname{Dense}_{A_1, \mathbf{b}_1; \sigma}(x)|| < \epsilon.$$

Bis hier: Lecture 3. 20.4.23

Proof. $\Rightarrow$: If σ is a polynomial of order N, then the output of the neural network is also a polynomial of order at most N. Polynomials of bounded degree are not dense in continuous functions.

$\Leftarrow$: We only show the case $\sigma = \operatorname{ReLU}$; the case of arbitrary σ is shown in [Fou22, Theorem 25.1].

First $d_{\text{in}} = d_{\text{out}} = 1$. By Lemma 1.3 it remains to show that piecewise linear functions are dense in the space of continuous functions on K for every compact set $K \subset \mathbb{R}$. Without loss of generality we can take $K = [0, 1]$.

Let g be a continuous function on $[0, 1]$. By the Heinor-Cantor theorem [Lan12, Proposition 3.11], g is *uniformly continuous*, that is

$$\forall \epsilon > 0 : \exists \delta > 0 : \forall x, y \in [0, 1] : |x - y| < \delta \Rightarrow |g(x) - g(y)| < \epsilon.$$

[3] References: 1D: [Fou22, Theorem 24.1], general: [Fou22, Theorem 25.1]

Now, for a given $\epsilon > 0$ pick such a $\delta > 0$. Possibly making it smaller, let $\delta = T/N$ for some $N \in \mathbb{N}$. Let L be the continuous function that is linear on the intervals $[t_{\ell-1}, t_\ell]$ for $\ell = 1, \dots, N$ and satisfies

$$L(t_\ell) = g(t_\ell), \quad \ell = 0, \dots, N.$$

Let $x \in [0,1]$ be given. Then we have $x \in [t_{\ell-1}, t_\ell]$ for some $\ell \in \{0, \dots, N\}$ and

$$\begin{aligned} |g(x) - L(x)| &= |g(x) - g(t_\ell)| + |g(t_\ell) - L(t_\ell)| + |L(t_\ell) - L(x)| \\ &\leq |g(x) - g(t_\ell)| + |g(t_\ell) - L(t_\ell)| + |L(t_\ell) - L(x)| \\ &\leq \epsilon + 0 + \frac{|g(t_\ell) - g(t_{\ell-1})|}{\delta} |x - t_{\ell-1}| \\ &\leq \epsilon + |g(t_\ell) - g(t_{\ell-1})| \leq 2\epsilon. \end{aligned}$$

Hence

$$\sup_{x \in [0,1]} |g(x) - L(x)| \leq 2\epsilon.$$

Now, for arbitrary d_{in} (and still $d_{\text{out}} = 1$, $\sigma = \text{ReLU}$). Consider the subspace of $C(K, \mathbb{R})$ given by

$$\mathcal{A} := \text{span}_{\mathbb{R}} \{ \exp(\langle \nu, \cdot \rangle) : \nu \in \mathbb{R}^{d_{\text{in}}} \}.$$

It is a subalgebra, since

$$\exp(\langle \nu, \cdot \rangle) \exp(\langle \mu, \cdot \rangle) = \exp(\langle \nu + \mu, \cdot \rangle).$$

It contains the constant function $1 = \exp(\langle 0, \cdot \rangle)$. It separates points, since for all $x, y \in K$, there exists $\nu \in \mathbb{R}^{d_{\text{in}}}$ such that

$$\langle \nu, x \rangle \neq \langle \nu, y \rangle.$$

By Theorem 1.6, for $g \in C(K, \mathbb{R})$ and there exist an integer k and $\nu_i \in \mathbb{R}^{d_{\text{in}}}, \gamma_i \in \mathbb{R}$, $i = 1, \dots, k$, such that

$$|g(x) - \sum_{i=1}^{k} \gamma_i \exp(\langle \nu_i, x \rangle)| < \epsilon, \quad \forall x \in K.$$

Now, for every i,

$$t \mapsto \gamma_i \exp(t),$$

is a continuous function and we only evaluate it in the compactum

$$K_i := \{ \langle \nu_i, x \rangle : x \in K \} \subset \mathbb{R}.$$

By above result in the one-dimensional case, there exist integers $r_i, i = 1, \ldots, k$ and $a_\ell^{(i)}, b_\ell^{(i)}, c_\ell^{(i)} \in \mathbb{R}$, $\ell = 1, \ldots, r_i$, $i = 1, \ldots, k$, such that

$$\sup_{t \in K_i} |\gamma_i \exp(t) - \left(\sum_{\ell=1}^{r_i} c_\ell^{(i)} \operatorname{ReLU}(a_\ell^{(i)} t + b_\ell^{(i)}) + d^{(i)} \right)| < \frac{\epsilon}{k}, \quad i = 1, \ldots, k.$$

Then for $x \in K$,

$$\begin{aligned}
&|g(x) - \left(\sum_{i=1}^{k} \sum_{\ell=1}^{r_i} c_\ell^{(i)} \operatorname{ReLU}(a_\ell^{(i)} t + b_\ell^{(i)}) + d^{(i)} \right)| \\
&\leq |g(x) - \sum_{i=1}^{k} \gamma_i \exp(\langle \nu_i, x \rangle)| \\
&\quad + \sum_{i=1}^{k} |\gamma_i \exp(\langle \nu_i, x \rangle) - \left(\sum_{\ell=1}^{r_i} c_\ell^{(i)} \operatorname{ReLU}(a_\ell^{(i)} \langle \nu_i, x \rangle + b_\ell^{(i)}) + d^{(i)} \right)| \\
&\leq 2\epsilon.
\end{aligned}$$

□

Theorem 1.6 (Stone-Weierstrass theorem)**.** *Let $K \subset \mathbb{R}^d$ be a compact set and $C(K, \mathbb{R})$ the $\mathbb{R}$-algebra of continuous, real-valued functions on K. Let $\mathcal{A}$ be a subalgebra of $C(K, \mathbb{R})$, satisfying*

1. *$\mathcal{A}$ is **nowhere vanishing**, that is for all $x \in K$ there is $\phi \in \mathcal{A}$ such that*

$$\phi(x) \neq 0.$$

2. *$\mathcal{A}$ is **separating points**, that is for all $x, y \in K$ there is $\phi \in \mathcal{A}$ such that*

$$\phi(x) \neq \phi(y).$$

Then: $\mathcal{A}$ is dense in $C(K, \mathbb{R})$ with norm given by the supremum norm.

Proof. See https://diehlj.github.io/stone-weierstrass/stone-weierstrass.html for a proof following [BD81].

Other proofs: [Fou22, Theorem E.3], [Loo13, p.9]. □

So, why not just work with single layer neural networks then?

> "In summary, a feedforward network with a single layer is sufficient to represent any function, but the layer may be infeasibly large and may fail to learn and generalize correctly. In many circumstances, using deeper models can reduce the number of units required to represent the desired function and can reduce the amount of generalization error." [GBC16].

Some results in this direction:

Theorem 1.7.

- *(Universal approximation theorem; arbitrary depth) [HS17] Width $d_{\text{in}} + 1$ and arbitrary depth, neural networks with ReLU activation are dense in the space of continuous functions on a compactum $K \subset \mathbb{R}^{d_{\text{in}}}$.*
- *A FFN with ReLU activation has number of affine regions of the order ([MPCB14])*

$$\mathcal{O}(\binom{\mathsf{width}}{d_{\text{in}}}^{d_{\text{in}}(\mathsf{depth}-1)} \mathsf{width}^{d_{\text{in}}})$$

- *[Tel16] For all $L \geq 1$ there is a depth $\mathcal{O}(L^2)$ FFN f with $\mathcal{O}(L^2)$ nodes such that for all depth $\mathcal{O}(L)$ FFNs g with $\mathcal{O}(2L^2)$ nodes we have*

$$\int_0^1 |f(x) - g(x)| \, \mathrm{d}x > \frac{1}{32}.$$

So, deep feed forward networks are what we should work with? **Usually not.** Often it pays to posit some **inductive bias**, usually affecting the network architecture. Starting from FFNs this usually means putting restrictions on the weight matrices.

Bis hier: Lecture 4. 24.4.23

- Convolutional Neural Networks (CNNs): Local and translation-invariant feature learning due to convolutional layers with smaller shared weight matrices (kernels), reducing the number of parameters and focusing on spatial hierarchies.[4]

  Consider a 1-dimensional convolutional layer, with *stride* 1, kernel $a \in \mathbb{R}^3$ of size 3, no bias, no *padding* and ReLU activation. The input is any $x \in \mathbb{R}^{d_{\text{in}}}$ and the output is

$$\begin{pmatrix} \mathrm{ReLU}(a_1x_1 + a_2x_2 + a_3x_3) \\ \mathrm{ReLU}(a_1x_2 + a_2x_3 + a_3x_4) \\ \vdots \\ \mathrm{ReLU}(a_1x_{d_{\text{in}}-2} + a_2x_{d_{\text{in}}-1} + a_3x_{d_{\text{in}}}) \end{pmatrix} \in \mathbb{R}^{d_{\text{in}}-2}.$$

  We can realize this as a dense layer with a very structured weight matrix, namely

$$A = \begin{pmatrix} a_1 & a_2 & a_3 & 0 & \cdots & 0 \\ 0 & a_1 & a_2 & a_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & a_1 & a_2 & a_3 \end{pmatrix} \in \mathbb{R}^{d_{\text{in}}-2 \times d_{\text{in}}}.$$

  The fact that the (small) kernel is shared across the input (i.e. is re-used again and again) is called **weight sharing**.

[4] GPT4-prompt: https://sharegpt.com/c/4htnBbg

See https://adamharley.com/nn_vis/cnn/2d.html for a visualization of a 2-dimensional convolutional layer (and a $2d$ FFN for comparison: https://adamharley.com/nn_vis/mlp/2d.html).

- Recurrent Neural Networks (RNNs): Sequence processing and long-range dependency learning through looping connections, enabling internal state maintenance for tasks like natural language processing and time-series analysis.
- Graph Neural Networks (GNNs): Efficient graph-structured data handling by structuring weight matrices to aggregate information from neighboring nodes, learning meaningful node representations.

Weight sharing, prominently featured in CNNs, offers multiple benefits, such as reduced memory requirements, faster training, and increased generalization due to fewer parameters.

One can also more fundamentally modify the architecture, using layers that are not of the form form (1), i.e. of the form "affine map, then entrywise nonlinearity". We will see that attention layers are such an example.

1.2 Tensors

Data is usually stored in **tensors**: scalar values (0-dimensional tensors), vectors (1-dimensional tensors), matrices (2-dimensional tensors), ...

In this course we consider tensors just as *multi-dimensional arrays* of numbers. To be precise, a tensor of **order** n is an element $T \in \mathbb{R}^{d_1 \times \ldots \times d_n}$. For $i_1 \in \{1, \ldots, d_1\}$, $\ldots$, $i_n \in \{1, \ldots, d_n\}$, we write

$$T_{i_1,\ldots,i_n} \in \mathbb{R},$$

for the entry of T at position $(i_1, \ldots, i_n)$. i_k is said to index the k-th **axis** of T. The **shape** of T is the tuple $(d_1, \ldots, d_n)$.

Einstein summation is a convenient notation to express tensor operations. For example the matrix-vector product, $A\mathbf{v}$ results in a vector (tensor of order 1)

$$\begin{aligned}(A\mathbf{v})_i &= \sum_{j=1}^{d} A_{i,j} v_j \\ &= A_{i,j} v_j.\end{aligned}$$

In Einstein's notation, a summation is implicitly applied to every index (here, just the index j) that appears at least twice in the expression.

Other example:

- Matrix-matrix product:

$$A_{i,k} B_{k,j} = (AB)_{i,j}.$$

- Matrix trace:

$$A_{i,i} = \mathrm{Tr}(A).$$

- Matrix-matrix product, with a different order:

$$A_{i,k}B_{j,k} = (AB^\top)_{i,j}.$$

- A bigger example:

$$A_{i,k,\ell}B_{k,\ell,j} = \sum_{k,\ell} A_{i,k,\ell}B_{k,\ell,j}.$$

All deep learning frameworks provide a method to compute the Einstein summation of a tensor expression. In tensorflow this is the function `tf.einsum`.

The **tensor product** of tensors $S \in \mathbb{R}^{d_1 \times \ldots \times d_n}$ and $T \in \mathbb{R}^{d_{n+1} \times \ldots \times d_{n+m}}$ is defined as

$$S \otimes T \in \mathbb{R}^{d_1 \times \ldots \times d_n \times d_{n+1} \times \ldots \times d_{n+m}}.$$

where

$$(S \otimes T)_{i_1,\ldots,i_n,j_1,\ldots,j_m} = S_{i_1,\ldots,i_n} T_{j_1,\ldots,j_m}.$$

The tensor product is associative, but not commutative.

A **pure** tensor is a tensor T that can be represented as the tensor product of a set of vectors,

$$\mathcal{T} = \mathbf{v}_1 \otimes \mathbf{v}_2 \otimes \cdots \otimes \mathbf{v}_n,$$

with $\mathbf{v}_i \in \mathbb{R}^{d_i}$.

The **rank**[5] of a tensor T is the minimal r such that T is the sum of r pure tensors, i.e. the minimal r such that there exixts $\mathbf{u}_i^{(j)}$

$$T = \sum_{i=1}^{r} \mathbf{u}_i^{(1)} \otimes \mathbf{u}_i^{(2)} \otimes \cdots \otimes \mathbf{u}_i^{(n)}.$$

Note that pure tensors have rank one.

Bis hier: Lecture 5. 27.4.23

1.3 Learning

The usual task in (supervised) machine learning is the prediction of a value given some input.

- **Classification**: we want to predict a discrete label of the input. (e.g. `cat` or `dog` for an image)

[5] Sometimes people denote the *order* of a tensor with rank. This is *not* good use of nomenclature.

- **Regression**: we want to approximate a, usually real-valued and continuous, function. Classical example: linear regression.

In **classification**, a collection $(\mathbf{x}_i^{\mathsf{train}}, y_i^{\mathsf{train}}) \in \mathbb{R}^{d_{\text{in}}} \times [n]$, $1 \leq i \leq N_{\mathsf{train}}$, of examples (training data) is given, where $\mathbf{x}_i^{\mathsf{train}}$ is the input, say an image, and y_i is the label, say 1 (`cat`) or 2 (`dog`). We want to construct a function

$$\hat{f} : \mathbb{R}^{d_{\text{in}}} \to [n],$$

that does a 'good job' at predicting the labels.

First, we aim for a function that does well on the training data. To measure 'goodness' we use a **loss function**

$$\mathsf{loss}' : [n] \times [n] \to \mathbb{R}$$

that measures the distance between the predicted label and the true label. For reasons that will become clear, we usually let the network predict a *probability distribution over the labels*, i.e.

$$\hat{f}(\mathbf{x}) \in \mathbb{R}^n,$$

with $\sum_{i=1}^n \hat{f}(\mathbf{x})_i = 1$ and $\hat{f}(\mathbf{x})_i \geq 0$. Then

$$\mathsf{loss} : \mathbb{R}^n \times [n] \to \mathbb{R}.$$

Define the **training loss** as

$$\mathsf{loss}_{\text{train}}(f) := \frac{1}{N_{\mathsf{train}}} \sum_{i=1}^{N_{\mathsf{train}}} \mathsf{loss}(f(\mathbf{x}_i^{\mathsf{train}}), y_i^{\mathsf{train}}).$$

We want to find a function $\hat{f}$ that minimizes the training loss. This is called **training** the network. If f_θ is our parametrized function, this amounts to

$$\theta^* \in \operatorname{argmin}_\theta \mathsf{loss}_{\text{train}}(f_\theta).$$

We usually do this via **gradient descent** (later). This will find (if we are lucky) a parameter $\hat{\theta}$, such that $\hat{f} := f_{\hat{\theta}}$ that does well on the training data. However, this is not enough. We want a function $\hat{f}$ that does well on **unseen** data. This is called **generalization**. We can measure generalization by evaluating the function $\hat{f}$ on a **test set**, a collection $(\mathbf{x}_i^{\mathsf{test}}, y_i^{\mathsf{test}}) \in \mathbb{R}^{d_{\text{in}}} \times [n]$, $1 \leq i \leq N_{\mathsf{test}}$, of examples. The **test loss** is defined as

$$\mathsf{loss}_{\mathsf{test}}(f) := \frac{1}{N_{\mathsf{test}}} \sum_{i=1}^{N_{\mathsf{test}}} \mathsf{loss}(f(\mathbf{x}_i^{\mathsf{test}}), y_i^{\mathsf{test}}).$$

Remark 1.8. *If the function $\hat{f}$ does well on the training data, but not on the test data, we say that the network* ***overfits*** *the training data. This is a common problem and many ways to combat it exist.*

In **regression** we aim to approximate an (unknown) function

$$\phi : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}.$$

Again, we are given training data $(\mathbf{x}_i^{\mathsf{train}}, \mathbf{y}_i^{\mathsf{train}}) \in \mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}}$, where we think of $\mathbf{y}_i \approx \phi(\mathbf{x}_i)$. We want to construct a function

$$\hat{f} : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}},$$

that minimizes some loss on the data (first, on the training data, but really we are interested in doing well on test data).

$$\mathsf{loss}_{\mathsf{train}}(f) := \frac{1}{N_{\mathsf{train}}} \sum_{i=1}^{N_{\mathsf{train}}} \mathsf{loss}(f(\mathbf{x}_i^{\mathsf{train}}), \mathbf{y}_i),$$

where

$$\mathsf{loss} : \mathbb{R}^{d_{\text{out}}} \times \mathbb{R}^{d_{\text{out}}} \to \mathbb{R}.$$

The usual example is

$$\mathsf{loss}(\mathbf{v}, \mathbf{w}) := \frac{1}{d_{\text{out}}} \sum_{j=1}^{d_{\text{out}}} |v_j - w_j|^p,$$

for $p = 1$ (**mean absolute error**) or $p = 2$ (**mean squared error**).

1.4 Objective functions / Loss functions for classification

As we have seen, in the context of classification we shall need a loss function $\mathsf{loss} : \mathbb{R}^n \times [n] \to \mathbb{R}$. We can think of this as a function $\mathsf{loss} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ where we interpret the second argument as a one-hot vector

$$\mathsf{onehot}(i) = \mathbf{e}_i \in \mathbb{R}^n.$$

Interpreted as a probability measure, $\mathsf{onehot}(i)$ is the probability distribution with probability 1 on the i-th element and 0 elsewhere.

So, we are looking for a loss function

$$\mathsf{loss} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R},$$

or more specificially

$$\mathsf{loss} : \mathcal{P}_n \times \mathcal{P}_n \to \mathbb{R},$$

where $\mathcal{P}$ is the set of probability distributions on $[n]$.

Bis hier: Lecture 6. 4.5.23

Possible distances for discrete probability measures:

- L^p **distances**; for $p \geq 1$,

$$d_{L^p}(P,Q) := \left(\sum_{i=1}^{n} |P(i) - Q(i)|^p \right)^{1/p}.$$

  Note: this is an actual metric.

- **total-variation distance**;

$$d_{\mathsf{total}}(P,Q) := \sup_{A \subset [n]} |P(A) - Q(A)| = \frac{1}{2} d_{L^1}(P,Q).$$

  Hence: this is an actual metric.

  Proof of the equality. First, for any A we have

$$P(A) - Q(A) = 1 - P(A^c) - (1 - Q(A^c)), = Q(A^c) - P(A^c).$$

  Hence

$$|P(A) - Q(A)| = |Q(A^c) - P(A^c)|,$$

  and hence

$$\begin{aligned} |P(A) - Q(A)| &= \frac{1}{2} \left(|P(A) - Q(A)| + |Q(A^c) - P(A^c)| \right) \\ &\leq \frac{1}{2} \sum_{i=1}^{n} |P(i) - Q(i)|. \end{aligned}$$

  This shows

$$\sup_{A \subset [n]} |P(A) - Q(A)| \leq \frac{1}{2} \sum_{i=1}^{n} |P(i) - Q(i)|.$$

  To show equality, define

$$A := \{ i \mid P(i) \geq Q(i) \}.$$

  Then

$$\begin{aligned} |P(A) - Q(A)| &= P(A) - Q(A) = \sum_{i \in A} P(i) - Q(i) \\ |P(A^c) - Q(A^c)| &= Q(A^c) - P(A^c) = \sum_{i \in A^c} Q(i) - P(i). \end{aligned}$$

  Hence

$$|P(A) - Q(A)| = \frac{1}{2} \left(|P(A) - Q(A)| + |P(A^c) - Q(A^c)| \right) = \frac{1}{2} \sum_{i} |P(i) - Q(i)|.$$

  □

■ **Kullback-Leibler divergence**[6]

$$D_{\mathrm{KL}}(\mathbf{p},\mathbf{q}) := D_{\mathrm{KL}}(\mathbf{p}||\mathbf{q}) := \sum_{i=1}^{n} \mathbf{p}_i \log\left(\frac{\mathbf{p}_i}{\mathbf{q}_i}\right),$$

where $0\log(0/0) := 0$. Or, as probability measures,

$$D_{\mathrm{KL}}(P,Q) = D_{\mathrm{KL}}(P||Q) = \mathbb{E}_{x\sim P}\left[\log\frac{P(x)}{Q(x)}\right].$$

D_{KL} is *not* a metric.

Interpretation: the Kullback-Leibler divergence is the expected number of extra bits needed to encode a sample from $\mathbf{p}$ when using a code optimized for $\mathbf{q}$ instead of a code optimized for $\mathbf{p}$.

Properties:

–

$$D_{\mathrm{KL}}(P||Q) = +\infty \Leftrightarrow \exists i : P(i) > 0, Q(i) = 0.$$

–

$$D_{\mathrm{KL}}(P||Q) \geq 0, \quad D_{\mathrm{KL}}(P||Q) = 0 \iff P = Q.$$

– (Convexity)

$$D_{\mathrm{KL}}\left(\lambda P + (1-\lambda)P'||\lambda Q + (1-\lambda)Q'\right) \leq \lambda D_{\mathrm{KL}}\left(P||Q\right) + (1-\lambda)D_{\mathrm{KL}}\left(P||Q\right)$$

Proof.

1. Immediate.
2. First, the function

$$\phi(x) := x\log(x),$$

is convex on $(0,\infty)$. Indeed,

$$\phi'(x) = \log(x) + 1, \qquad \phi''(x) = \frac{1}{x} + 1 > 0.$$

Assume that $D_{\mathrm{KL}}(P||Q) \neq +\infty$, i.e. for all i we have $P(i) > 0 \Rightarrow Q(i) > 0$. Then, using Jensen's inequality

$$\sum_i p_i \log\left(\frac{p_i}{q_i}\right) = \sum_i q_i \frac{p_i}{q_i}\log\left(\frac{p_i}{q_i}\right) \quad \geq \phi\left(\sum_i q_i \frac{p_i}{q_i}\right) = \phi\left(\sum_i p_i\right) = \phi(1) = 0.$$

[6] On nomenclature: a **divergence** in statistics is a binary map on a (statistical) manifold, satisfying certain properties.

□

We used the following lemma:

Lemma 1.9 (Jensen's inequality)**.** *Let $\phi : \mathbb{R}^p \to \mathbb{R}$ be **convex**, that is, for every $\mathbf{x}, \mathbf{y} \in \mathbb{R}^p, \lambda \in [0,1]$,*

$$\phi(\lambda \mathbf{x} + (1-\lambda)\mathbf{y}) \leq \lambda\phi(\mathbf{x}) + (1-\lambda)\phi(\mathbf{y}).$$

Let $p_i \in [0,1], i = 1, \ldots, n, \mathbf{x}_i \in \mathbb{R}^p$ with $\sum_i p_i = 1$. Then

$$\phi\left(\sum_i p_i x_i\right) \leq \sum_i p_i \phi(x_i).$$

Proof. Induction on n. For $n = 1$ it is trivially true.

Let it be true for n. If $p_{n+1} = 1$ there is nothing to show. Assume $p_{n+1} < 1$. Then

$$\begin{aligned}
\phi\left(\sum_{i=1}^{n+1} p_i x_i\right) &= \phi\left((1-p_{n+1})\sum_{i=1}^{n} \frac{p_i}{1-p_{n+1}} x_i + p_{n+1}x_{n+1}\right) \\
&\leq (1-p_n)\phi\left(\sum_{i=1}^{n} \frac{p_i}{1-p_{n+1}} x_i\right) + p_{n+1}\phi\left(x_{n+1}\right) \\
&\leq (1-p_n)\sum_{i=1}^{n} \frac{p_i}{1-p_{n+1}} \phi\left(x_i\right) + p_{n+1}\phi\left(x_{n+1}\right) \\
&= \sum_{i=1}^{n+1} p_i \phi\left(x_i\right).
\end{aligned}$$

□

- **cross-entropy**;

$$\mathrm{CE}(\mathbf{p}, \mathbf{q}) := -\sum_{i=1}^{n} \mathbf{p}_i \log(\mathbf{q}_i) = D_{\mathrm{KL}}(\mathbf{p}||\mathbf{q}) - \sum_{i=1}^{n} \mathbf{p}_i \log(\mathbf{p}_i).$$

This is *not* a metric.

Properties: From the properties of the KL divergence we get

1. $\mathrm{CE}(\mathbf{p}, \mathbf{q}) \geq -\sum_i p_i \log(p_i) \geq 0$.
2. $\mathrm{CE}(\mathbf{p}, \mathbf{q}) = -\sum_i p_i \log(p_i) \iff \mathbf{p} = \mathbf{q}$.

Interpretation: the cross-entropy is the expected number of bits needed to encode a sample from P using a code Q.

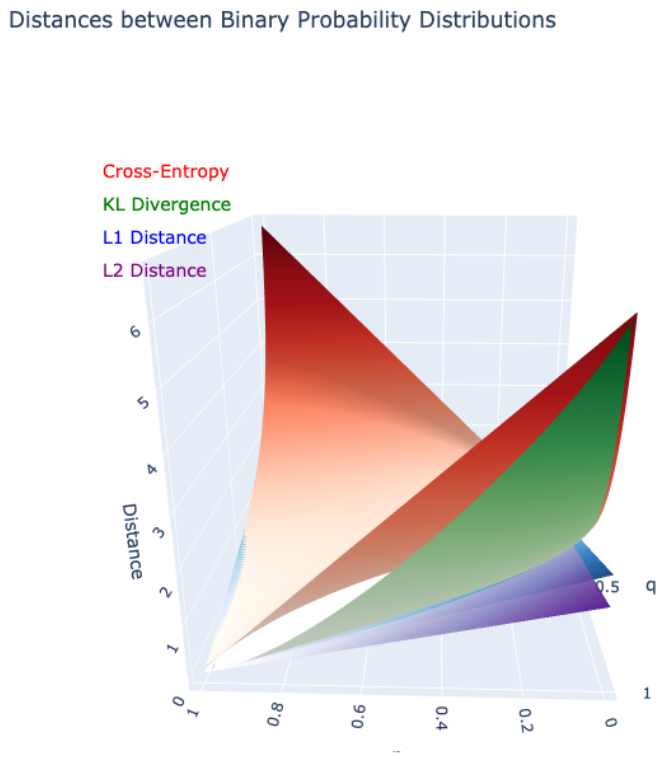


Figure 3: *Comparing several distances on binary probability distributions.*

Remark 1.10. *By the first point, in a classification task, it 'only makes sense' to plug the one-hot vector, which is the target, in the P-slot. (If Q is one-hot, then the only case for which $D_{\mathrm{KL}}(P,Q)$ is finite, is $P = Q$.)*

Then

$$D_{\mathrm{KL}}(P||Q) = \sum_i p_i \log(p_i) + \mathrm{CE}(P,Q).$$

Hence, for a fixed P, optimizing the cross-entropy $\mathrm{CE}(P,Q)$ is equivalent to optimizing the KL-divergence $D_{\mathrm{KL}}(P||Q)$ (over Q). If $\mathbf{p}$ is hot on position $\hat{i}$ then

$$\mathrm{CE}(\mathbf{p},\mathbf{q}) = -\log(\mathbf{q}_{\hat{i}}).$$

1.5 (Stochastic) gradient descent

An abstract optimization problem is of the form[7]

$$\operatorname{argmin}_\theta C(\theta) := \{\theta \in \mathbb{R}^p : C(\theta) \leq C(\theta') \text{ for all } \theta' \in \mathbb{R}^p\},$$

for some **objective** or **cost** function $C : \mathbb{R}^p \to \mathbb{R}$. A simple approach to solving it is: start with some initial parameters θ_0, determine in which direction in parameter space we should move to decrease the cost, and then move in that direction.

We know that, infinitesimally, the negative of the **gradient**

$$\nabla_\theta C(\theta) = \begin{bmatrix} \partial\theta_1 C \\ \vdots \\ \partial\theta_p C \end{bmatrix},$$

points in the direction of steepest descent. Indeed, for a vector $\mathbf{x} \in \mathbb{R}^p, \mathbf{x} \neq 0$, and for every $\mathbf{y} \in \mathbb{R}^p$ with $||\mathbf{y}|| = 1$, by the Cauchy-Schwarz inequality we have

$$\langle \mathbf{y}, \mathbf{x} \rangle \geq -||\mathbf{y}||||\mathbf{x}|| = -||\mathbf{x}|| = \langle \mathbf{x}, -\frac{\mathbf{x}}{||\mathbf{x}||} \rangle.$$

This yields the method of **gradient descent**:

$$\theta_{t+1} := \theta_t - \eta \nabla_\theta C(\theta_t).$$

Here η is the **learning rate**, which is a hyperparameter usually of the order of 10^{-3} or 10^{-4}.

We will apply gradient descent in settings where there are absolutely no theoretical guarantees (to date) of convergence. In more restricted settings, however, we can prove that gradient descent converges.

Theorem 1.11. *Assume that $C : \mathbb{R}^p \to \mathbb{R}$ is **convex**, Assume that $\min C$ is attained, i.e. $\operatorname{argmin} C \neq \emptyset$. Assume also that C is continuously differentiable and that ∇C is Lipschitz continuous with Lipschitz constant L. If the learning rate η is chosen smaller than $\frac{1}{L}$, then*[8]

$$C(\theta_t) - \min C \leq \frac{||\theta_0 - \theta^*||^2}{2\eta t}.$$

In particular, from any starting point*, gradient descent converges to a global minimum.*

For the proof, see the appendix.

There are a plethora of variants of vanilla gradient descent. A classical modification is the introduction of **momentum**. The updates then look as follows. For fixed $m \in (0, 1]$

[9]

[7] Note that argmin is a *set*. We will abuse notation and write argmin also for an *element* of this set.

[8] References: [GG23, Theorem 3.4]

and $\eta \in (0, \infty)$,

$$v_{t+1} = m \cdot v_t - \gamma \cdot \nabla_\theta C(\theta_t)$$
$$\theta_{t+1} = \theta_t + v_t,$$

or with a different parametrization, $\beta \in (0, 1)$,

$$v_{t+1} = \beta \cdot v_t + (1 - \beta) \cdot \nabla_\theta C(\theta_t)$$
$$\theta_{t+1} = \theta_t - \eta \cdot v_t.$$

A more recent modification is **Adam**. Let [10]

$$g_t := \nabla C(\theta_t),$$

fix some initial $x_0 \in \mathbb{R}^p$. Fix $\beta_1, \beta_2 \in [0, 1), \eta, \epsilon > 0$. Consider the "moment updates"

$$v_t := \beta_1 v_{t-1} + (1 - \beta_1) g_t, \qquad s_t := \beta_2 s_{t-1} + (1 - \beta_2) g_t^2,$$

Define the "bias-corrected moments" as

$$\hat{v}_t := \frac{v_t}{1 - (\beta_1)^t}, \qquad \hat{s}_t := \frac{s_t}{1 - (\beta_2)^t}.$$

Then, the final update is

$$\theta_{t+1} := \theta_t - \frac{\eta}{\sqrt{\hat{s}_t} + \epsilon} \hat{v}_t.$$

1.6 Various other ingredients (Regularization, standardization, normalization, skip connections, ..)

Dropout

Dropout is a regularization technique used in neural networks to prevent overfitting. The technique involves randomly "dropping out" (i.e., temporarily removing) a number of output features of the layer *during training*. For each training sample, each neuron has a probability of being temporarily dropped out, meaning it will not contribute to the forward pass nor participate in backpropagation. This means that the model is forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons. Dropout has the effect of making the training process noisy, forcing nodes within a layer to probabilistically take on more or less responsibility for the inputs.

Here's how you might implement a dropout layer in a neural network using TensorFlow:

[9] References: https://distill.pub/2017/momentum/, https://www.cs.mcgill.ca/~siamak/COMP551/slides/7-gradient-descent_short.pdf

[10] References: [KB], [Cal20, Section 4.7]

```
import tensorflow as tf

model = tf.keras.models.Sequential([
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

In this example, the `Dropout` layer will randomly set 20% of the input units to 0 at each update during training time, which helps prevent overfitting.

Skip connections

Skip connections, also known as **shortcut connections** or **residual connections**, are a type of architecture used in neural networks that allow for the direct propagation of information from earlier layers to later layers without going through all the intermediate layers.

In a traditional feedforward neural network, information from one layer is passed to the next layer, and so on. This means that information from the first layer must pass through all intermediate layers before it reaches the final layer.

However, in very deep networks, this approach can cause two significant problems: *vanishing gradients* and *representational bottlenecks*. The vanishing gradient problem occurs when gradients are backpropagated through the network and become increasingly small, making it difficult for the network to learn. Representational bottlenecks occur when the dimensionality of the layers is not enough to encode the necessary information about the input.

Skip connections address these issues by allowing information to bypass some layers. This is typically achieved by adding the output of an earlier layer to the output of a later layer, a concept known as identity shortcut connection introduced in the ResNet model (Residual Network). This direct path has the effect of enabling the backpropagation of gradients through the network without the risk of them vanishing, as they can be directly propagated back through the shortcut connections.

Skip connections have another beneficial effect: they encourage the network to learn residual mappings. A residual mapping is the difference (or residual) between the input and the desired output. The idea is that it should be easier to optimize the network to produce a residual of zero (i.e., to produce an output identical to the input) than to optimize the network to produce any particular non-zero residual. This can significantly improve the network's learning capability, especially in the case of very deep networks.

An example from https://github.com/keras-team/keras/blob/v2.12.0/keras/applications/resnet.py:

```
# Shortcut is either identity or 1x1 conv
if conv_shortcut:
  shortcut = layers.Conv2D(
      4 * filters, 1, strides=stride, name=name + "_0_conv"
  )(x)
  shortcut = layers.BatchNormalization(
      axis=bn_axis, epsilon=1.001e-5, name=name + "_0_bn"
  )(shortcut)
else:
  shortcut = x

# The main path
x = layers.Conv2D(filters, 1, strides=stride, name=name + "
   _1_conv")(x)
...
x = layers.BatchNormalization(
  axis=bn_axis, epsilon=1.001e-5, name=name + "_3_bn"
)(x)

# Add shortcut value to main path
x = layers.Add(name=name + "_add")([shortcut, x])
x = layers.Activation("relu", name=name + "_out")(x)
```

Batch normalization

Each dimension of the non-batch dimensions is normalized (over the batch dimension). [11]
For a order-1 tensor $\mathbf{x}$,

$$\mathsf{mean}(\mathbf{x}) := \frac{1}{|\mathbf{x}|}\sum_{i=1}^{|\mathbf{x}|}\mathbf{x}_i$$

$$\mathsf{var}(\mathbf{x}) := \frac{1}{|\mathbf{x}|}\sum_{i=1}^{|\mathbf{x}|}(\mathbf{x}_i - \mathsf{mean}(\mathbf{x}))^2.$$

In the example of data format $\mathbb{R}^{|n|} \otimes \mathbb{R}^{|c|}$,

$$\mathsf{BatchNormalization} : \mathbb{R}^{|b|} \otimes \mathbb{R}^{|n|} \otimes \mathbb{R}^{|c|} \to \mathbb{R}^{|b|} \otimes \mathbb{R}^{|n|} \otimes \mathbb{R}^{|c|}$$

$$\mathsf{BatchNormalization}(T)_{b,n,c} := \frac{T_{b,n,c} - \mathsf{mean}(T_{\bullet,n,c})}{\sqrt{\mathsf{var}(T_{\bullet,n,c}) + \varepsilon}}$$

Here, $\varepsilon > 0$ is a small (fixed) number, added for numerical stability.

[11] References: [IS15], [GBC16, Section 8.7.1]

This is at training time. At test time the mean and variance are replaced by running averages collected at training.

Layer normalization

Similar to batch normalization, but now normalizing a fixed batch (or fixed batch and temporal dimension). For example, the way it is usually done in transformers [12]

$$\mathsf{LayerNormalization}(T)_{b,n,c} := \gamma_c \frac{T_{b,n,c} - \mathsf{mean}(T_{b,n,\bullet})}{\sqrt{\mathsf{var}(T_{b,n,\bullet}) + \varepsilon}}. + \beta_c$$

Here, γ, β are learned parameters. Unlike batch normalization, layer normalization performs exactly the same computation at training and test times.

[12] References: [BKH16]

2 Attention

2.1 Softmax

[13]

Given a vector $\mathbf{x} \in \mathbb{R}^n$, the softmax function is defined as

$$\operatorname{softmax} \mathbf{x} := \frac{1}{\sum_{j=1}^n \exp(x_j)} \begin{pmatrix} \exp(x_1) \\ \exp(x_2) \\ \dots \\ \exp(x_n) \end{pmatrix}$$

Lemma 2.1. *Let* $\mathbf{x} \in \mathbb{R}^n$.

1. $$(\operatorname{softmax} \mathbf{x})_i \in (0, 1] \qquad \forall i \in [n].$$

2. $$\sum_{i=1}^n (\operatorname{softmax} \mathbf{x})_i = 1.$$

3. *For all* $c \in \mathbb{R}$
$$\operatorname{softmax}(\mathbf{x} + c) = \operatorname{softmax}(\mathbf{x}).$$

4. *Assume that* $\mathbf{x}$ *has a unique maximal value at position* $\hat{i}$. *Then*
$$\operatorname{softmax}(\lambda \mathbf{x}) \to_{\lambda \to \infty} \mathbf{e}_{\hat{i}}.$$

Remark 2.2. *Point 1.,2. imply that the softmax yields a* probability measure *on* $[n]$. *Point 4. shows that this probability measure is concentrated at the maximum position of* $\mathbf{x}$. *Hence, the name* softargmax *would be more fitting.*

Point 3. shows that softmax *only sees the* relative *size of the inputs.*

Proof.

1. This follows from the fact that $\exp : \mathbb{R} \to (0, \infty)$.
2. $$\sum_{i=1}^n (\operatorname{softmax} \mathbf{x})_i = \sum_{i=1}^n \frac{\exp(x_i)}{\sum_{j=1}^n \exp(x_j)} = \frac{\sum_{i=1}^n \exp(x_i)}{\sum_{j=1}^n \exp(x_j)} = 1.$$
3. Direct calculation.

[13] References: [VSP+17]

4. For $i \neq \hat{\imath}$ we have

$$x_i - x_{\hat{\imath}} < 0.$$

Hence

$$\frac{\exp(\lambda x_i)}{\sum_{j=1}^n \exp(\lambda x_j)} \leq \frac{\exp(\lambda x_i)}{\exp(\lambda x_{\hat{\imath}})} = \exp(\lambda(x_i - x_{\hat{\imath}})) \xrightarrow[\lambda\to\infty]{} 0.$$

Hence

$$\mathrm{softmax}(\lambda \mathbf{x})_i \xrightarrow[\lambda\to\infty]{} 0.$$

By ii. we then get

$$\mathrm{softmax}(\lambda \mathbf{x})_{\hat{\imath}} \xrightarrow[\lambda\to\infty]{} 1.$$

□

Bis hier: Lecture 7. 8.5.23

We shall need the softmax function applied to only certain axis of larger tensors, mostly the last axis. For example, for $T \in \mathbb{R}^{b\times n\times c}$,

$$\mathrm{softmax}_{\mathsf{axis}=-1}(T)_{b,n,c} := \frac{\exp(T_{b,n,c})}{\sum_{c'=1}^c \exp(T_{b,n,c'})}$$

2.2 Attention

Example 2.3.

How to model a database/lookup table as a differential function?

Let there be n many "values" $\mathbf{v}_i \in \mathbb{R}^{|k|}, i = 1, \ldots, n$, for example stored in a python array. Given an integer $i = 1, \ldots, n$ we can of course pick out $\mathbf{v}_i$, by indexing the array. How can we turn this (discrete) operation into a differentiable one?

We assign to each value $\mathbf{v}_i$ a "key", which we - in this example - choose to be $\mathbf{e}_i \in \mathbb{R}^n$ the i-th standard basis vector.

PICTURE IN LECTURE

We can now "query" these keys with a vector $q \in \mathbb{R}^n$,

$$\langle q, \mathbf{e}_i \rangle$$

For example, with $q = \mathbf{e}_3$,

$$\langle q, \mathbf{e}_i \rangle = \delta_{3,i}.$$

It then makes sense to form a linear combination of the values, depending on how well the corresponding key has matched:

$$\mathsf{lookup}(q) := \sum_{i=1}^{n} \langle q, \mathbf{e}_i \rangle \ \mathbf{v}_i.$$

For example,

$$\mathsf{lookup}(\mathbf{e}_3) = \mathbf{v}_3, \quad \mathsf{lookup}(\mathbf{e}_3 + 2\mathbf{e}_5) = \mathbf{v}_3 + 2\mathbf{v}_5.$$

Note that lookup *is a differentiable function.*

Several observations on

- lookup *is linear in each of its argument,*
- *The coefficients* $\langle q, \mathbf{e}_i \rangle$ *can take any value in* $\mathbb{R}$.

It is often beneficial to restrict the coefficients to the interval $[0, 1]$ *and moreover to make them sum to* 1*, i.e. have them be a probability distribution.*

As we have seen, this can be achieved by the softmax function. This leads to

$$\mathsf{lookup}(q) := \sum_{i=1}^{n} \mathrm{softmax}\left((\langle q, \mathbf{e}_j \rangle)_{j=1,\dots,n}\right)_i \ \mathbf{v}_i.$$

We now define the general attention layer, with learnable weights.

Given

$$Q \in \mathbb{R}^{n_q \times d_q}, K \in \mathbb{R}^{n_k \times d_k}, V \in \mathbb{R}^{n_k \times d_v}$$

and

$$W^Q \in \mathbb{R}^{d_q \times k}, W^K \in \mathbb{R}^{d_k \times k}, W^V \in \mathbb{R}^{d_v \times v},$$

define

$$\mathsf{Attention}_{W^Q, W^K, W^V}(Q, K, V) := \mathrm{softmax}_{\mathsf{axis}=-1}\left((QW^Q)(KW^K)^\top\right) VW^V.$$

Note that attention is *not* of the form (1), since the nonlinear function (the softmax) does *not* act entrywise.

If $Q = V = K$, then this layer is also called a **self-attention** layer.

The canonical example comes from sequential data $X \in \mathbb{R}^{n \times d}$, where the first dimension is thought of as time, and $W^Q \in \mathbb{R}^{d \times k}, W^K \in \mathbb{R}^{d \times k}, W^V \in \mathbb{R}^{d \times v}$. Then

$$\mathsf{Attention}_{W^Q, W^K, W^V}(X, X, X) \in \mathbb{R}^{n \times v},$$

which can again be fed into another self-attention layer (now with v taking the role of d ..).

Remark 2.4. *Note that the calculation of*

$$\mathsf{Attention}_{W^Q,W^K,W^V}(X,X,X),$$

has complexity $\mathcal{O}(n^2)$*. This is the* ***quadratic-cost problem of self-attention****. It prohibits the use of (vanilla) self-attention for long sequences. We will see solutions to this problem in Section 4.*

In practice, a slight modification called Multi Head Attention is used, for some $|h|\geq 1$, and $A_{\mathsf{projection}}, \mathbf{b}_{\mathsf{projection}}$ of appropriate dimensions:

$$\mathsf{MultiHeadAttention}(Q,K,V) := A_{\mathsf{projection}} \cdot \begin{pmatrix} \mathsf{Attention}_{W_1^Q,W_1^K,W_1^V}(Q,K,V) \\ \cdots \\ \mathsf{Attention}_{W_{|h|}^Q,W_{|h|}^K,W_{|h|}^V}(Q,K,V) \end{pmatrix} + \mathbf{b}_{\mathsf{projection}}$$

3 Self-attention: positional information

Vanilla attention *ignores* the order of the key/value pairs. In other words it is **invariant** to the action of the symmetric group:

$$\mathsf{Attention}_{W^Q,W^K,W^V}(Q,K_{\tau(\bullet),\bullet},V_{\tau(\bullet),\bullet}) = \mathsf{Attention}_{W^Q,W^K,W^V}(Q,K_{\bullet,\bullet},V_{\bullet,\bullet}) \qquad \forall \tau \in S_n.$$

Vanilla self-attention is **equivariant** with respect to the symmetric group acting on the queries

$$\mathsf{Attention}_{W^Q,W^K,W^V}(Q_{\tau(\bullet),\bullet},K,V)_{t,i} = \mathsf{Attention}_{W^Q,W^K,W^V}(Q_{\bullet,\bullet},K,V)_{\tau(t),i} \qquad \forall \tau \in S_n.$$

In many cases one *is* interested in the order, absolute and/or relative, of the time-points.

Positional embedding

We have seen

$$\begin{aligned} q_m &= f_q(x_m) \\ k_m &= f_k(x_m) \\ v_m &= f_k(v_m), \end{aligned}$$

where, usually, f_q, f_k, f_v are linear functions.

A simple way of encoding chronological information is by adding a time-parameter to the transformation

$$\begin{aligned} q_m &= f_q(x_m, m) \\ k_m &= f_k(x_m, m) \\ v_m &= f_k(v_m, m). \end{aligned}$$

Most are of the form

$$f_\eta(x, m) = W^\eta(x + \mathsf{pe}_m),$$

for some $\mathsf{pe}_m \in \mathbb{R}^{d_\eta}$.

sin/cos positional encoding

Already in [VSP+17] the following positional encoding is suggested.

$$\mathsf{pe}_{2i}^{\mathsf{sin/cos}} = \left(\sin(\frac{\mathsf{pos}}{10000^{2i/d_{model}}})\right)$$
$$\mathsf{pe}_{2i+1} = \left(\cos(\frac{\mathsf{pos}}{10000^{2i/d_{model}}})\right), \qquad i = 0, \ldots, d_{model}/2 - 1, \mathsf{pos} = 1, \ldots, d_{model}.$$

Rotary position embedding

[14]

$$f_\eta(x, m) = R_m W_\eta x_m$$

where

$$R_m = \begin{pmatrix} \cos(m\theta_1) & \sin(m\theta_1) & 0 & \ldots & 0 & 0 \\ -\sin(m\theta_1) & \cos(m\theta_1) & 0 & \ldots & 0 & 0 \\ 0 & 0 & \cos(m\theta_2) & \ldots & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots & \cos(m\theta_{d/2}) & \sin(m\theta_{d/2}) \\ \ldots & \ldots & \ldots & \ldots & -\sin(m\theta_{d/2}) & \cos(m\theta_{d/2}) \end{pmatrix},$$

$\theta_i := 10000^{-2(i-1)/d}, i = 1, \ldots, d/2.$

4 Self-attention: alleviating quadratic cost

We use **Big-O notation** to describe the complexity of an algorithm.

Consider an algorithm whose input size is n. (For example: the length of a time series.) We say an algorithm is $\mathcal{O}(g(n))$ if the algorithm's running time satisfies that there exist a constant $C > 0$ and integer n_0 such that

$$\text{algorithm's running time, depending on } n \ \leq Cg(n)$$

for all $n \geq n_0$.

Similarily, consider an algorithm whose input size is parametrized by several variables $n_1, \ldots, n_d$. We say an algorithm is $\mathcal{O}(g(n_1, \ldots, n_d))$ if the algorithm's running time satisfies that there exist a constant $C > 0$ and integer n_0 such that

$$\text{algorithm's running time, depending on } n_1, \ldots, n_d \ \leq Cg(n_1, \ldots, n_d),$$

for $n_i \geq n_0$ for all $i \in \{1, \ldots, d\}$.

[14] References: [SLP+21]

Example 4.1.

- $\mathcal{O}(1)$*: constant time*

  Example: array access
- $\mathcal{O}(\log n)$*: logarithmic time*

  Example: binary search
- $\mathcal{O}(n)$*: linear time*

  Example: linked list access
- $\mathcal{O}(n \log n)$*: linearithmic time*

  Example: merge sort
- $\mathcal{O}(n^2)$*: quadratic time*

  Example: bubble sort (worst case)

4.1 "Linearizing"/ low-rank methods

The general idea is as follows ("separating queries and keys"). We assume there are function $\phi_m, m = 1, \dots, M$ such that

$$\begin{aligned}\sum_j \exp(\langle q_i, k_j \rangle) &\approx \sum_j \sum_m \phi_m(q_i)^\top \phi_m(k_j) \\ &= \sum_m \phi_m(q_i)^\top \sum_j \phi_m(k_j) =: \sum_m \phi_m(q_i)^\top E_m,\end{aligned}$$

where E_m can be computed (and stored) in $\mathcal{O}(n)$ time (for a fixed M). And then the computation for n queries can be done in $\mathcal{O}(n+n) = \mathcal{O}(n)$ time.

Transformers are RNNs '20

[15]

$$\frac{\phi(Q_i)^\perp \sum_{i=1}^N \phi(K_j) V_j^\perp}{\phi(Q_i)^\perp \sum_{i=1}^N \phi(K_j)}$$

They use $\phi(x) = \text{elu}(x) + 1$ (entrywise). No theoretical justification.

Performer '20

[16]

This seems to be the most mathematically well-founded method around.

First, we note

[15] References: https://paperswithcode.com/paper/transformers-are-rnns-fast-autoregressive
[16] References: https://paperswithcode.com/paper/rethinking-attention-with-performers

Lemma 4.2. *If $Z \sim \mathcal{N}(0, I_d)$, then*

$$\mathbb{E}_{Z\sim\mathcal{N}(0,I_d)}\left[\exp(Z^\top x)\right] = \exp(||x||^2/2).$$

Proof. We do $d = 1$.

$$\begin{aligned}
\mathbb{E}_{Z\sim\mathcal{N}(0,1)}\left[\exp(Zx)\right] &= \int \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}z^2 + zx)\, dz \\
&= \int \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}(z-x)^2 + \frac{1}{2}x^2)\, dz \\
&= \exp(\frac{1}{2}x^2) \int \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}(z-x)^2)\, dz \\
&= \exp(\frac{1}{2}x^2).
\end{aligned}$$

□

Then

$$\exp\left(\frac{1}{2}||x+y||^2\right) = \mathbb{E}_{Z\sim\mathcal{N}(0,I_d)}\left[\exp\left(Z^\top(x+y)\right)\right],$$

and

$$\exp\left(\frac{1}{2}||x+y||^2\right) = \exp\left(\frac{1}{2}||x||^2 + \frac{1}{2}||y||^2 + \langle x, y\rangle\right)$$

Hence

$$\begin{aligned}
\exp\left(\langle x, y\rangle\right) &= \exp\left(-\frac{1}{2}||x||^2\right) \mathbb{E}_{Z\sim\mathcal{N}(0,I_d)}\left[\exp(Z^\top(x+y))\right] \exp\left(-\frac{1}{2}||y||^2\right) \qquad (3) \\
&\approx \psi(x)^\top \psi(y).
\end{aligned}$$

where

$$\psi(x) := \frac{\exp\left(-\frac{1}{2}||x||^2\right)}{\sqrt{k}} \begin{pmatrix} \exp(\omega_1^\top x) \\ \exp(\omega_2^\top x) \\ \cdots \\ \exp(\omega_k^\top x) \end{pmatrix},$$

and $\omega_1, \ldots, \omega_k$ are iid $\mathcal{N}(0, I_d)$ samples.

Remark 4.3. *Since*

$$\mathbb{E}_{Z\sim\mathcal{N}(0,I_d)}\left[\exp(Z^\top(x+y))\right] = \mathbb{E}_{Z\sim\mathcal{N}(0,I_d)}\left[\exp(-Z^\top(x+y))\right],$$

we get a different identity,

$$\begin{aligned}\exp\left(\langle x, y\rangle\right) &= \exp\left(-\frac{1}{2}||x||^2\right)\mathbb{E}_{Z\sim\mathcal{N}(0,I_d)}\left[\frac{1}{2}\exp(Z^\top(x+y)) + \frac{1}{2}\exp(-Z^\top(x+y))\right]\exp\left(-\frac{1}{2}||y||^2\right)\\ &= \exp\left(-\frac{1}{2}||x||^2\right)\mathbb{E}_{Z\sim\mathcal{N}(0,I_d)}\left[\cosh(Z^\top(x+y))\right]\exp\left(-\frac{1}{2}||y||^2\right),\end{aligned}$$

which leads to an estimator with lower variance.

Now this helps us as follows:

$$\begin{aligned}r_i &= \frac{1}{Z}\sum_j \exp(\langle q_i, k_j\rangle)v_j\\ &\approx \frac{1}{Z}\sum_j\sum_m \phi_m(q_i)^\top\phi_m(k_j)v_j\\ &= \frac{1}{Z}\sum_m \phi_m(q_i)^\top S_m,\end{aligned}$$

where

$$S_m := \sum_j \phi_m(k_j)v_j.$$

This can be calculated once, *and then be used for every query* q_i.

$$\rightsquigarrow \mathcal{O}(\mathsf{TimeseriesLength}\cdot\#\mathsf{MonteCarloSamples})$$

Remark 4.4. *In [RR07] the same reasoning is used to calculate the kernel matrix (or Gram matrix) of a kernel function, for example*

$$k(x, y) = \exp(-\gamma||x - y||^2).$$

If we want to calculate the kernel matrix $K \in \mathbb{R}^{n\times n}$*, then we can use the same trick:*

$$\begin{aligned}K_{ij} &= \exp(-\gamma||x_i - x_j||^2)\\ &= \exp(-\gamma||x_i||^2)\exp(-\gamma||x_j||^2)\exp(\gamma x_i^\top x_j)\\ &\approx \psi(x_i)^\top\psi(x_j),\end{aligned}$$

Learning is then done on $\psi(x_i)$ *(instead of the "true" feature space for the kernel, which is infinite-dimensional).*

5 Background: probability theory

A **probability** space consists $(\Omega, \mathcal{F}, \mathbb{P})$ consists of a set Ω, a σ-algebra $\mathcal{F}$ on it and a probability measure assigning probabilities to the elements of $\mathcal{F}$.

A **random variable** is a measurable map

$$X : (\Omega, \mathcal{F}) \to (V, \mathcal{G}),$$

for some space V. In most examples $V = \mathbb{R}^m$ is either Euclidean space and $\mathcal{G} = \mathcal{B}(\mathbb{R}^m)$ is the σ-algebra of Borel sets on it or V is countable and $\mathcal{G}$ is the power set of V (in this case, we say that X is **discrete**).

The measure space Ω is usually irrelevant and not specified. One is usually only interested in the **pushforward** measure $\mathbb{P}_X := X_*\mathbb{P}$ (or the **law** of X). This is the measure on $(V, \mathcal{G})$ defined by

$$(X_*\mathbb{P})[A] := \mathbb{P}[X^{-1}(A)] = \mathbb{P}[X \in A] = \mathbb{P}[\{\omega \in \Omega : X(\omega) \in A\}].$$

Then, for every bounded, measurable function $f : \mathbb{R}^{d_X} \to \mathbb{R}$

$$\mathbb{E}[f(X)] := \int f(X(\omega))\mathbb{P}[d\omega] = \int f(x)\mathbb{P}_X[dx].$$

Bis hier: Lecture 19.6.23

Two random variables $X : (\Omega, \mathcal{F}) \to (V, \mathcal{G}), X' : (\Omega, \mathcal{F}) \to (V', \mathcal{G}')$ are **independent** if for all $A \in \mathcal{G}, A' \in \mathcal{G}'$

$$\mathbb{P}[X \in A, X' \in A'] = \mathbb{P}[X \in A]\mathbb{P}[X' \in A'].$$

Equivalently, if for all $f : V \to \mathbb{R}, g : V' \to \mathbb{R}$ bounded, measurable

$$\mathbb{E}[f(X)g(X')] = \mathbb{E}[f(X)]\mathbb{E}[g(X')].$$

$X : \Omega \to \mathbb{R}^{d_X}$ has a **density** $p_X \in L^1(\mathbb{R}^{d_X}, \mathbb{R})$ (with respect to Lebesgue measure) if

$$\mathbb{P}_X[B] = \int 1_B(x)p_X(x)dx \quad \forall B \in \mathcal{B}(\mathbb{R}^{d_X}).$$

In this case, we also write $X \sim p_X$. Then, for every bounded, measurable function $f : \mathbb{R}^{d_X} \to \mathbb{R}$

$$\mathbb{E}[f(X)] = \int_{\mathbb{R}^{d_X}} f(x)p_X(x)dx.$$

If $Z = (X, Y)$ has a density p_Z, then X has density p_X and Y has density p_Y, where

$$p_X(x) = \int p_Z(x, y)dy,$$
$$p_Y(y) = \int p_Z(x, y)dx.$$

In that case, X and Y are independent if and only if

$$p_Z(x, y) = p_X(x)p_Y(y).$$

Remark 5.1. *Two random variables X, Y can both have a density on their own, without $Z = (X, Y)$ having a density. For example, take $X \sim \mathcal{N}(0, 1)$ and $Y = X$.*

Let X, Y be two random variables. A **(regular) conditional distribution** of X given Y is a family of probability measures $\mathbb{P}_{X|Y=y}$ on $\mathbb{R}^{d_X}$, indexed by $y \in \mathbb{R}^{d_Y}$ with appropriate measurability properties[17] and satisfying

$$\mathbb{P}[X \in A] = \int_{\mathbb{R}^{d_Y}} \mathbb{P}_{X|Y=y}(A)\mathbb{P}_Y[dy].$$

Theorem 5.2.

1. *Assume that X takes values in a Euclidean space or in a countable set, then for any random variable Y, a conditional distribution $\mathbb{P}_{X|Y=y}$ exists. It is unique in the following sense: if $\mathbb{P}'$ is another conditional distribution, then for any $A \in \mathcal{B}(\mathbb{R}^{d_X})$*

   $$\mathbb{P}_{X|Y=y}(A) = \mathbb{P}'_{X|Y=y}(A) \quad \mathbb{P}_Y\text{-}a.e.$$

   If the conditional distribution exists, then X, Y are independent if and only if for all $A \in \mathcal{B}(\mathbb{R}^{d_X})$

   $$\mathbb{P}_{X|Y=y}(A) = \mathbb{P}_X(A) \quad \mathbb{P}_Y\text{-}a.e.$$

2. *If Y is discrete,*

   $$\mathbb{P}_{X|Y=y}(A) = \begin{cases} \frac{\mathbb{P}[X\in A, Y=y]}{\mathbb{P}[Y=y]} & \mathbb{P}[Y=y] \neq 0 \\ 0 & else. \end{cases}$$

3. *If $Z := (X, Y)$ takes values in some $\mathbb{R}^{d_X \times d_Y}$, and they have a joint joint density p_Z, then $\mathbb{P}_{X|Y=y}$ has a density*

   $$p_{X|Y=y}(x) = \begin{cases} \frac{p_{X,Y}(x,y)}{p_Y(y)} & p_Y(y) \neq 0 \\ 0 & else. \end{cases}$$

[17] In detail:

$$\mathbb{P} : \mathbb{R}^{d_Y} \times \mathcal{B}(\mathbb{R}^{d_X}) \to [0, 1], \quad (y, A) \mapsto \mathbb{P}_{X|Y=y}(A),$$

is a function which satisfies

- $$y \mapsto \mathbb{P}_{X|X=y}(A),$$

  is $\mathcal{B}(\mathbb{R}^{d_Y})$-measurable for every $A \in \mathcal{B}(\mathbb{R}^{d_X})$,
- $$A \mapsto \mathbb{P}_{X|X=y}(A),$$

  is a probability measure on $\mathbb{R}^{d_X}$ for every $y \in \mathbb{R}^{d_Y}$.

Proof.

1. For the Euclidean case, [Kle13, Theorem 8.29] (applying it coordinate-wise).
2. We just verify that the claimed family of measures does the job. Let $A \in \mathcal{B}(\mathbb{R}^{d_X})$. Then

$$\begin{aligned}\int \mathbb{P}_{X|Y=y}(A)\mathbb{P}_Y[dy] &= \sum_{y\in Y} \mathbb{P}_{X|Y=y}(A)\mathbb{P}_Y[Y=y] = \sum_{y\in Y,\mathbb{P}[Y=y]\neq 0} \mathbb{P}_{X|Y=y}(A)\mathbb{P}_Y[Y=y] \\ &= \sum_{y\in Y,\mathbb{P}[Y=y]\neq 0} \frac{\mathbb{P}[X\in A, Y=y]}{\mathbb{P}[Y=y]}\mathbb{P}_Y[Y=y] = \sum_{y\in Y,\mathbb{P}[Y=y]\neq 0} \mathbb{P}[X\in A, Y=y] \\ &= \sum_{y\in Y} \mathbb{P}[X\in A, Y=y] = \mathbb{P}[X\in A].\end{aligned}$$

3. [Kle13, Example 8.31]

□

Bis hier: Lecture 22.6.23

We define the multivariate **normal distribution** on $\mathbb{R}^d$ with mean $\mu \in \mathbb{R}^d$ and covariance matrix $\Sigma \in \mathbb{R}^{d\times d}$, where Σ is symmetric and positive definite. It has the following density

$$p_{\mathcal{N}(\mu,\Sigma)}(x) := \frac{1}{\sqrt{(2\pi)^d \det\Sigma}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right).$$

If $\mu = 0$ and $\Sigma = I_d$, it is called the **standard normal distribution**.

Theorem 5.3.

1. *(Moment generating function)*

$$\mathbb{E}[\exp(\langle a, X\rangle)] = \exp(\frac{1}{2}a^\top\Sigma a).$$

2. *Let $X \sim \mathcal{N}(\mu,\Sigma)$. Then*

$$\begin{aligned}\mathbb{E}[X] &= \mu, \\ \mathrm{Cov}[X] &= \Sigma.\end{aligned}$$

Here, $\mathrm{Cov}[X] \in \mathbb{R}^{d\times d}$ is the covariance matrix of X, defined by

$$\mathrm{Cov}[X]_{ij} := \mathbb{E}\left[(X_i - \mathbb{E}[X_i])(X_j - \mathbb{E}[X_j])\right].$$

3. *Let $X \sim \mathcal{N}(\mu,\Sigma)$. If $e \le d$, $A \in \mathbb{R}^{e\times d}$ with $\mathrm{rank}(A) = e$, $\mathbf{b} \in \mathbb{R}^e$, then*

$$AX + \mathbf{b} \sim \mathcal{N}(A\mu + \mathbf{b}, A\Sigma A^T).$$

4. *Let $X \sim \mathcal{N}(\mu, \Sigma)$, $Y \sim \mathcal{N}(\nu, \Lambda)$, and assume that X, Y are independent. Then*

$$X + Y \sim \mathcal{N}(\mu + \nu, \Sigma + \Lambda).$$

In particular, if $X, Y \sim \mathcal{N}(0, \Sigma)$ are independent and identically distributed, then

$$\frac{1}{\sqrt{2}}(X + Y) \sim \mathcal{N}(0, \Sigma).$$

5. *Let $X \sim \mathcal{N}(\mu, \Sigma)$, $Y \sim \mathcal{N}(\nu, \Lambda)$, and assume that X, Y are independent, then*

$$(X, Y) \sim \mathcal{N}\left(\begin{pmatrix}\mu\\ \nu\end{pmatrix}, \begin{pmatrix}\Sigma & 0\\ 0 & \Lambda\end{pmatrix}\right).$$

The other way around, if

$$(X, Y) \sim \mathcal{N}\left(\begin{pmatrix}\mu\\ \nu\end{pmatrix}, \begin{pmatrix}\Sigma & 0\\ 0 & \Lambda\end{pmatrix}\right),$$

then $X \sim \mathcal{N}(\mu, \Sigma)$, $Y \sim \mathcal{N}(\nu, \Lambda)$, and X, Y are independent. "Two jointly normal random variables are independent if and only if they are uncorrelated."

In particular: a standard normal distribution in $\mathbb{R}^d$ is the same as d independent, one-dimensional, standard normal distributions.

Proof. These follow from straightforward calculations. They can also be found, for example, in [Ton12]. □

Remark 5.4. *For the definition of the density, it was important that Σ is invertible (since it is positive definite).*

*More generally one can define a **singular normal distribution** with mean $\mu \in \mathbb{R}^d$ and covariance matrix $\Sigma \in \mathbb{R}^{d\times d}$, where Σ is symmetric and positive semi-definite. This is a random variable X with the following property: for every $a \in \mathbb{R}^d$,*

$$a^\top X \sim \mathcal{N}(a^\top \mu, a^\top \Sigma a).$$

Here, $\mathcal{N}(0, 0)$ is, by definition, the Dirac measure at 0.

With this definition, the above theorem holds for singular normal distributions as well; in iii. one can then pick ϵ arbitrary.

Theorem 5.5 (Isserlis' theorem / Wick's theorem)**.** *If $X = (X_1, \dots, X_n) \sim \mathcal{N}(0, \Sigma)$, is a zero-mean normal random vector with covariance matrix $\Sigma \in \mathbb{R}^{n\times n}$, then*

$$\mathbb{E}[X_1 \dots X_n] = \sum_{\pi \in P_n^2} \prod_{\{i,j\}\in\pi} \mathbb{E}[X_i, X_j],$$

where P_n^2 is the set of partitions of $\{1,\dots,n\}$ into pairs.

For example

$$\mathbb{E}[X_1X_2X_3X_4] = \mathbb{E}[X_1X_2]\mathbb{E}[X_3X_4] + \mathbb{E}[X_1X_3]\mathbb{E}[X_2X_4] + \mathbb{E}[X_1X_4]\mathbb{E}[X_2X_3].$$

Proof.

Proof A:

$$\mathbb{E}[\exp(\langle a, X\rangle)] = \exp(\frac{1}{2}a^\top \Sigma a),$$

and

$$\mathbb{E}[X_1 \dots X_n] = \frac{\partial^n}{\partial a_1 \dots \partial a_n}|_{a=0}\mathbb{E}[\exp(\langle a, X\rangle)].$$

So it remains to identify the terms in the Taylor expansion of $\exp(\frac{1}{2}a^\top \Sigma a)$.

Proof B:

Claim: the following "Gaussian integration-by-parts" formula holds

$$\mathbb{E}[X_1 f(X)] = \sum_{i=1}^{d} \mathbb{E}[X_1X_i]\mathbb{E}[\partial_i f(X)].$$

Indeed, for a one-dimensional Gaussian random variable $X_1 \sim \mathcal{N}(0,\sigma^2)$,

$$\begin{aligned}
\mathbb{E}[X_1X_1]\mathbb{E}[f'(X_1)] &= \sigma^2 \frac{1}{\sqrt{2\pi\sigma^2}} \int f'(x) \exp\left(-\frac{x^2}{2\sigma^2}\right) dx \\
&= 0 - \sigma^2 \frac{1}{\sqrt{2\pi\sigma^2}} \int f(x) \frac{d}{dx} \exp\left(-\frac{x^2}{2\sigma^2}\right) dx \\
&= -\sigma^2 \frac{1}{\sqrt{2\pi\sigma^2}} \int f(x) \exp\left(-\frac{x^2}{2\sigma^2}\right) (-x\sigma^2)\, dx \\
&= \mathbb{E}[X_1 f(X_1)].
\end{aligned}$$

Now, for general $X = (X_1,\dots,X_d)$, define

$$\hat{X} := X - \lambda_i X_1.$$

where $\lambda_1 = 1$ and $\lambda_i := \mathbb{E}[X_1X_i]/\mathbb{E}[X_1X_1]$, $i = 2,\dots,d$. Then

$$\mathbb{E}[\hat{X}_i X_1] = \mathbb{E}[(X_i - \lambda_i X_1)X_1] = \mathbb{E}[X_iX_1] - \lambda_i \mathbb{E}[X_1X_1] = 0.$$

Moreover, $X_1, \hat{X}$ is jointly (degenerate) Gaussian, since it is a linear transformation of X (Remark 5.4).

Hence, X_1 and $\hat{X}$ are independent and then

$$\begin{aligned}\mathbb{E}[X_1 f(X)] &= \mathbb{E}_{\hat{X}}\left[\mathbb{E}_{X_1|\hat{X}}\left[X_1 f(\hat{X}+\lambda X_1)\right]\right] \\ &= \mathbb{E}_{\hat{X}}\left[\mathbb{E}_{X_1}\left[X_1 f(\hat{X}+\lambda X_1)\right]\right].\end{aligned}$$

Now, for a fixed $\hat{X}$, by the one-dimensional case,

$$\mathbb{E}_{X_1}\left[X_1 f(\hat{X}+\lambda X_1)\right] = \sum_j \mathbb{E}\left[X_1 X_1\right]\mathbb{E}\left[\partial_j f(\hat{X}+\lambda X_1)\lambda_j\right]. = \sum_j \mathbb{E}\left[X_1 X_j\right]\mathbb{E}\left[\partial_j f(\hat{X}+\lambda X_1)\right].$$

Integrating over $\hat{X}$ yields the claim. □

Bis hier:
Lecture 26.6.23

Example 5.6 (Gaussian conditioning)**.** *Let*

$$Z = (X, Y) \sim \mathcal{N}(\mu, \Sigma),$$

where we write $\mu \in \mathbb{R}^{d_X+d_Y}$ *and* $\Sigma \in R^{(d_X+d_Y)\times(d_X+d_Y)}$ *(positive definite and symmetric) as*

$$\mu = \begin{pmatrix}\mu_X \\ \mu_Y\end{pmatrix}, \quad \Sigma = \begin{pmatrix}\Sigma_{XX} & \Sigma_{XY} \\ \Sigma_{YX} = \Sigma_{XY}^\top & \Sigma_{YY}\end{pmatrix}.$$

Then, the conditional distribution of X *given* $Y = y$ *is*

$$\mathbb{P}_{X|Y=y} = \mathcal{N}\left(\mu_{X|Y=y}, \Sigma_{X|Y=y}\right),$$

where

$$\begin{aligned}\mu_{X|Y=y} &= \mu_X + \Sigma_{XY}\Sigma_{YY}^{-1}(y-\mu_Y), \\ \Sigma_{X|Y=y} &= \Sigma_{XX} - \Sigma_{XY}\Sigma_{YY}^{-1}\Sigma_{YX}.\end{aligned}$$

Let us verify this for the case $d_X = d_Y = 1$*. Then*

$$\mu = \begin{pmatrix}\mu_X \\ \mu_Y\end{pmatrix}, \quad \Sigma = \begin{pmatrix}\Sigma_{XX} & \Sigma_{XY} \\ \Sigma_{XY} & \Sigma_{YY}\end{pmatrix}.$$

Then

$$\begin{aligned}\mu_{X|Y=y} &= \mu_X + \frac{\Sigma_{XY}}{\Sigma_{YY}}(y-\mu_Y), \\ \Sigma_{X|Y=y} &= \Sigma_{XX} - \Sigma_{XY}^2/\Sigma_{YY}.\end{aligned}$$

And indeed,

$$p_Y(y) = \frac{1}{\sqrt{2\pi\Sigma_{YY}}} \exp\left(-\frac{(y-\mu_Y)^2}{2\Sigma_{YY}}\right),$$

and

$$\Sigma^{-1} = \frac{1}{\Sigma_{XX}\Sigma_{YY} - \Sigma_{XY}^2}\begin{pmatrix}\Sigma_{YY} & -\Sigma_{XY}\\ -\Sigma_{XY} & \Sigma_{XX}\end{pmatrix} = \begin{pmatrix}\frac{1}{\Sigma_{XX}} & -\frac{\Sigma_{XY}}{\Sigma_{XX}\Sigma_{YY}}\\ -\frac{\Sigma_{XY}}{\Sigma_{XX}\Sigma_{YY}} & \frac{1}{\Sigma_{YY}}\end{pmatrix},$$

and

$$\begin{aligned}&p_{X,Y}(x,y)\\ &= \frac{1}{2\pi\sqrt{\Sigma_{XX}\Sigma_{YY} - \Sigma_{XY}^2}}\\ &\exp\left(-\frac{1}{2}\frac{1}{\Sigma_{XX}\Sigma_{YY} - \Sigma_{XY}^2}(\Sigma_{YY}(x-\mu_X)^2 + \Sigma_{XX}(y-\mu_Y)^2 - 2\Sigma_{XY}^2(x-\mu_X)(y-\mu_Y))\right),\end{aligned}$$

and hence

$$\begin{aligned}p_{X|Y=y}(x) &= p_{X,Y}(x,y)/p_Y(y)\\ &= \frac{\frac{1}{2\pi\sqrt{\Sigma_{XX}\Sigma_{YY}-\Sigma_{XY}^2}}\exp(...)}{\frac{1}{\sqrt{2\pi\Sigma_{YY}}}\exp\left(-\frac{(y-\mu_Y)^2}{2\Sigma_{YY}}\right)},\\ &= \frac{1}{\sqrt{2\pi}\sqrt{\Sigma_{XX} - \frac{\Sigma_{XY}^2}{\Sigma_{YY}}}}\exp\left(-\frac{1}{2}\frac{(x-\mu_X - \frac{\Sigma_{XY}}{\Sigma_{YY}}(y-\mu_Y))^2}{\Sigma_{XX} - \frac{\Sigma_{XY}^2}{\Sigma_{YY}}}\right).\end{aligned}$$

There are several concepts of convergence of random variables $X_1, X_2, ...$

1. **Convergence in distribution**: $X_n \to X$ in distribution if

   $$\lim_{n\to\infty} \mathbb{E}[f(X_n)] = \mathbb{E}[f(X)],$$

   for all bounded, continuous functions $f : \mathbb{R}^{d_X} \to \mathbb{R}$. This is also called **weak convergence**. Alternatively, for all $A \in \mathcal{B}(\mathbb{R}^{d_X})$, with $\mathbb{P}[X \in \partial A] = 0$, [18]

   $$\lim_{n\to\infty} \mathbb{P}[X_n \in A] = \mathbb{P}[X \in A].$$

2. **Convergence in probability**: $X_n \to X$ in probability if

   $$\limsup_{n\to\infty} \mathbb{P}\left[|X_n - X| > \epsilon\right] = 0$$

[18] This convergence is metrizable: https://en.wikipedia.org/wiki/Lévy-Prokhorovmetric

3. **Almost sure convergence**: $X_n \to X$ almost surely if

$$\mathbb{P}\left[\lim_{n\to\infty} X_n = X\right] = 1.$$

Equivalently, to see the difference to convergence in probability, for all $\epsilon > 0$,

$$\mathbb{P}\left[\limsup_{n\to\infty} |X_n - X| > \epsilon\right] = 0.$$

4. L^p **convergence** $X_n \to X$ in L^p if

$$\lim_{n\to\infty} \mathbb{E}\left[||X_n - X||^p\right] = 0.$$

Theorem 5.7.

almost sure convergence $\implies$ *convergence in probability* $\implies$ *convergence in distribution.*

For any $p \geq q \geq 1$,

convergence in L^p $\implies$ *convergence in* L^q $\implies$ *convergence in probability.*

Remark 5.8. *All implications are strict. For example,*

convergence in probability $\not\Rightarrow$ *almost sure convergence.*

Indeed, consider $\Omega = [0,1]$ *with uniform measure and*

$$\begin{aligned}
&X_1(\omega) = 1_{[0,1/2)}(\omega), \quad X_2(\omega) = 1_{[1/2,1)}(\omega),\\
&X_3(\omega) = 1_{[0,1/3)}(\omega), \quad X_4(\omega) = 1_{[1/3,2/3)}(\omega), \quad X_5(\omega) = 1_{[2/3,1)}(\omega),\\
&X_6(\omega) = 1_{[0,1/4)}(\omega), \quad X_7(\omega) = 1_{[1/4,2/4)}(\omega), \quad X_8(\omega) = 1_{[2/4,3/4)}(\omega), \quad X_9(\omega) = 1_{[3/4,1)}(\omega),\\
&\dots
\end{aligned}$$

We say a family $X_i, i \in I$, of random variables is independent, identically distributed (i.i.d.) if they are independent and have the same distribution.

Theorem 5.9. *Let* $X_1, X_2, ..$ *be i.i.d. random, real-valued variables.*

1. *(Strong law of large numbers) If* $\mathbb{E}[|X_1|] < +\infty$, *then*

$$\frac{1}{n}\sum_{i=1}^{n} X_i \to \mathbb{E}[X_1] \quad a.s.$$

2. *(Central limit theorem) If* $\mathbb{E}[X_1] = 0$ *and* $\mathbb{E}[X_1^2] < +\infty$, *then*

$$\frac{1}{\sqrt{n}}\sum_{i=1}^{n} X_i \to \mathcal{N}\left(0, \mathbb{E}[X_1^2]\right) \quad \textit{in distribution.}$$

Proof. Strong law of large numbers: [Kle13, Theorem 5.17]

Central limit theorem: [Kle13, Theorem 15.37]. □

5.1 Stochastic processes

A discrete-time stochastic process is a family of random variables $X_0, X_1, X_2, \ldots$. A continuous-time stochastic process is a family of random variables $X_t, t \in [0, T]$.[19]

PLOTS

A fundamental class of discrete-time stochastic processes are **Markov chains**. A discrete-time stochastic process $X_0, X_1, X_2, \ldots$ in some discrete state space X is a Markov chain if for all $n \in \mathbb{N}$ and $x_0, \ldots, x_n \in X$, with

$$\mathbb{P}[X_0 = x_0, \ldots, X_{n-1} = x_{n-1}] > 0,$$

we have

$$\mathbb{P}[X_n = x_n | X_0 = x_0, \ldots, X_{n-1} = x_{n-1}] = \mathbb{P}[X_n = x_n | X_{n-1} = x_{n-1}].$$

"The future is independent of the past given the present."

Example 5.10.

1. *(Random walk) Let $X_0, X_1, X_2, \ldots$ be i.i.d. random variables in $\{-1, 1\}$. Then*

$$Y_n := \sum_{i=0}^{n} X_i$$

is a Markov chain.

2. *Consider the set $X = \{1, \ldots, n\}$. Let A be a stochastic matrix (on X), i.e. $A \in [0,1]^{n \times n}$ with*

$$\sum_{j=1}^{n} A_{ij} = 1 \quad \forall i \in \{1, \ldots, n\}.$$

Let ν be a probability measure on X, that is $\mu \in [0,1]^n$ with

$$\sum_{i=1}^{n} \mu_i = 1.$$

Then there is a (unique, in law) Markov process $X_0, X_1, ..$ such that

$$\mathbb{P}\left[X_0 = i\right] = \nu_i, \quad \mathbb{P}\left[X_{n+1} = j | X_n = i\right] = A_{ij}.$$

The distribution of X_n is given by (considering ν as a row vector)

$$\mathbb{P}\left[X_n = j\right] = (\nu A^n)_j$$

[19] Usually: plus some conditions of measurability and adaptiveness.

We can also make this example ***time-inhomogeneous****. Let $A_0, A_1, \dots$ be a sequence of stochastic matrices. Then there is a (unique, in law) Markov process $X_0, X_1, ..$ such that*

$$\mathbb{P}\left[X_0 = i\right] = \nu_i, \quad \mathbb{P}\left[X_{n+1} = j | X_n = i\right] = A_{n,ij}.$$

We can also generalize from a discrete state space V to a general state space V. Let V be a general state space. Let A be a stochastic kernel on V, i.e. $A : V \times \mathcal{B}(V) \to [0,1]$ with

$$\int_X A(x, B)\, dx = 1 \quad \forall B \in \mathcal{B}(V).$$

Let ν be a probability measure on V. Then there is a (unique, in law) Markov process $X_0, X_1, ..$ such that

$$\mathbb{P}\left[X_0 \in B\right] = \nu(B), \quad \mathbb{P}\left[X_{n+1} \in B | X_n = x\right] = A(x, B).$$

Consider $V = \mathbb{R}^d$ and let

$$A(x, B) = \mathcal{N}(x, \Sigma)(B) = \int_B \frac{1}{(2\pi)^{d/2}\sqrt{\det \Sigma}} \exp\left(-\frac{1}{2}(y-x)^\top \Sigma^{-1}(y-x)\right) dy.$$

Then the corresponding Markov chain is a "random walk with Gaussian steps".

3. *(non-example / Hidden Markov model)*

   Let $X_0, X_1, X_2, \dots$ be a Markov chain in $\{1, .., n\}$. Let $f : \{1, .., n\} \to \mathbb{N}$ be a function. Let

   $$Y_n := f(X_n).$$

   Then, in general, Y_n is not a Markov chain. Indeed, consider $n = 2$,

   $$A = \begin{pmatrix} 0 & 1 & 0 & 1/5 & 0 \\ 0 & 0 & 1 & 1/5 & 0 \\ 1/2 & 0 & 0 & 1/5 & 0 \\ 0 & 0 & 0 & 1/5 & 1 \\ 1/2 & 0 & 0 & 1/5 & 0, \end{pmatrix},$$

   and

   $$f(1) = f(2) = f(3) = 1, \quad f(4) = f(5) = 2.$$

4. *("Everything is Markovian if we take the state space large enough")*

Let $X_0, X_1, X_2, \dots$ be a stochastic process in $\{1, .., n\}$. Consider the stochastic process $Y_0, Y_1, Y_2, \dots$ in $[n]^{\mathbb{N}}$.

$$Y_n := (X_0, \dots, X_n, 0, 0, 0, \dots).$$

Then: $Y_0, Y_1, Y_2, \dots$ is a Markov chain. Indeed, for all $n \in \mathbb{N}$ and $y_0, \dots, y_{n-1}, y_n \in [n]^{\mathbb{N}}$, if

$$\begin{aligned} \mathbb{P}\left[Y_0 = y_0, \dots, Y_{n-1} = y_{n-1}\right] > 0 \\ \mathbb{P}\left[Y_n = y_n, Y_{n-1} = y_{n-1}\right] > 0, \end{aligned}$$

then there is a unique $x_0, \dots, x_n \in \{1, .., n\}$ such that

$$y_i = (x_0, \dots, x_i, 0, 0, 0, \dots), \quad i = 0, \dots, n.$$

And then

$$\begin{aligned} \mathbb{P}\left[Y_n = y_n | Y_0 = y_0, \dots, Y_n = y_n\right] &= \mathbb{P}\left[X_n = x_n | X_0 = x_0, \dots, X_n = x_n\right] \\ &= \mathbb{P}\left[Y_n = y_n | Y_{n-1} = y_{n-1}\right]. \end{aligned}$$

5.2 Framing (deep) learning in probabilistic terms

Classification

Classification models aim to predict the class label of a given input. For example, the input could be an image represented as a matrix of pixel values in $\mathbb{R}^{28\times 28}$, where each entry corresponds to the grayscale intensity of a pixel, and the class labels could represent the digits 0-9 for a handwritten digit recognition task. In this case, the classification model would analyze the image data to determine the most likely digit represented in the image, based on the pixel values. The input space would be $\mathbb{R}^{28\times 28}$, with each element in the input matrix representing the grayscale intensity of a pixel in the image. The class labels could be denoted as $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$, each corresponding to a specific digit.

There is a "true" or hypothetical probability measure on $V = \{0, \dots, 9\} \times \mathbb{R}^{28\times 28}$ representing the underlying distribution of the data, which captures the relationship between the input images in $\mathbb{R}^{28\times 28}$ and the class labels $0, \dots, 9$. We can express this probability measure as a joint probability distribution $P[C, x]$, where $C \in 0, \dots, 9$ denotes the class labels (digits) and $x \in \mathbb{R}^{28\times 28}$ denotes the input images (pixel matrices).

The goal of a classification model is to approximate this true probability measure P as closely as possible, given a set of observed samples from P. In order to make predictions using the classification model, we are interested in estimating the conditional probability distribution $P(C|x) = \frac{P(C,x)}{P(x)}$, where $P(C|x)$ represents the probability of a class label C given input image x. This can be achieved using Bayes' theorem, which states that $P(C|x) = \frac{P(x|C)P(C)}{P(x)}$. Here, $P(x|C)$ is the class-conditional probability distribution, and $P(C)$ is the prior probability of the classes.

The learning process involves finding a suitable approximation $\hat{P}(C, x)$ to the true probability measure $P(C, x)$, based on a set of training samples $(C_i, x_i)_{i=1}^N$ drawn from P. This is typically achieved by minimizing a loss function or maximizing a likelihood function, which measures the discrepancy between the true class labels and the estimated probabilities produced by the classification model.

Once the model is trained, we can use it to classify new data points by computing the posterior probabilities $\hat{P}(C|x)$ and selecting the class with the highest probability. The choice of the specific classification model, such as a Convolutional Neural Network (CNN) or a Support Vector Machine (SVM), will determine how the approximation $\hat{P}(C, x)$ is modeled and how the learning process is conducted.

In conclusion, framing the classification problem in terms of a hypothetical probability measure P on the set $V = 0, \ldots, 9 \times \mathbb{R}^{28\times 28}$ provides a rigorous foundation for understanding the goals and principles of classification models applied to image recognition tasks. This probabilistic perspective allows us to quantify uncertainty, reason about the data-generating process, and make informed decisions based on the estimated probabilities.

Generative models

Consider the task of generating realistic images 100×100 of human faces. This can be framed as follows. Consider the space $V = \mathbb{R}^{3\times 100\times 100}$ of all RGB, 100×100 images. One posits that there is a hypothetical probability measure Q on V, such that if one samples from Q one obtains a realistic image of a human face.

The goal of a generative model in this context is to learn an approximation of the true probability measure Q on the space $V = \mathbb{R}^{3\times 100\times 100}$, given a set of observed samples of realistic human face images drawn from Q. By approximating the probability measure Q, the generative model aims to capture the underlying structure and patterns present in the space of realistic face images.

Once the generative model is trained, it can be used to generate new, realistic human face images by sampling from the approximated probability measure $\hat{Q}$. The quality of the generated images will depend on how well the model has approximated the true probability measure Q during the training process.

There are several types of generative models that can be used for this task, such as Variational Autoencoders (VAEs), Generative Adversarial Networks (GANs), and autoregressive models like PixelCNN. Each of these models has a different approach to learning the underlying probability distribution, but they all share the common goal of approximating the true measure Q.

For example, in a GAN, there are two neural networks, the generator and the discriminator, that compete with each other during the training process. The generator's goal is to produce realistic face images by sampling from a noise distribution and transforming it to match the distribution of the training data. The discriminator's goal is to distinguish between real face images from the training set and fake face images generated by the generator. The generator improves by learning to generate more realistic images that

can fool the discriminator, while the discriminator improves by learning to better differentiate between real and fake images. Through this adversarial process, the generator learns to approximate the true probability measure Q of realistic human face images.

In conclusion, generative models can be used to approximate the hypothetical probability measure Q on the space $V = \mathbb{R}^{3\times100\times100}$, representing the distribution of realistic human face images. By learning this approximation, generative models can generate new, realistic images by sampling from the approximated measure $\hat{Q}$. This framework allows us to create diverse and realistic images, which can be used for various applications such as data augmentation, computer graphics, and virtual avatars.

6 Background: variational autoencoders

For $P \ll Q$, define the **Kullback-Leibler divergence** of P from Q

$$D_{\mathrm{KL}}(P||Q) := \int \log\left(\frac{dP}{dQ}(x)\right) P(dx) = \mathbb{E}_{X\sim P}\left[\log\left(\frac{dP(X)}{dQ(X)}\right)\right]$$

Here $\frac{dP}{dQ}$ is the Radon-Nikodym derivative of P with respect to Q. It satisfies

$$P[A] = \int_A \frac{dP}{dQ}(x)Q(dx).$$

If P is not absolutely continuous with respect to Q, we set

$$D_{\mathrm{KL}}(P||Q) := +\infty.$$

Remark 6.1.

1. *Interpretation: in terms of "surprise" and coding.*

   To understand KL divergence in terms of "surprise," consider the concept of surprise as the negative log-probability of an event. If an event is very likely, it is less "surprising," and thus has a lower surprise score. Conversely, if an event is very unlikely, it is highly "surprising," and has a higher surprise score.

   In the context of KL divergence, we compare the "surprise" of an event according to two different probability distributions P and Q. KL divergence is then the expectation of the difference in surprise according to P and Q. If P and Q are similar, the expected difference in surprise is low, and so the KL divergence is low. If P and Q are dissimilar, the expected difference in surprise is high, so the KL divergence is high. In terms of coding, KL divergence can be thought of as the amount of "wasted" code length when using a code optimized for a distribution Q to encode samples from a distribution P.

   In information theory, we often want to encode information as efficiently as possible. The optimal code length for an event with probability p is $-\log(p)$. *If we*

have a distribution P and we create our encoding based on P, our expected code length is minimized. However, if we have to use an encoding based on a different distribution Q, our expected code length is greater.

The KL divergence is the expected increase in code length when we use a code optimized for Q to encode events from P. If P and Q are similar, the KL divergence (the "wasted" code length) is small. If P and Q are dissimilar, the KL divergence is large.

It is important to note that the KL divergence is not symmetric: $D_{\mathrm{KL}}(P||Q)$ is not generally equal to $D_{\mathrm{KL}}(Q||P)$. This asymmetry reflects the fact that the "waste" incurred when using a code optimized for Q to encode P is not the same as the waste incurred when using a code optimized for P to encode Q.

2. *D_{KL} is* not *a metric. Neither is it symmetric, for example*

$$P(a) = P(b) = \frac{1}{2}, P(c) = 0$$
$$Q(a) = Q(b) = \frac{1}{4}, Q(c) = \frac{1}{2}$$

Exercise: Verify that $D_{\mathrm{KL}}(P||Q) \neq D_{\mathrm{KL}}(Q||P)$**.**

Nor does it satisfy the triangle inequality.

In the special case where P, Q have densities p, q with respect to Lebesgue measure, the condition $P \ll Q$ is equivalent to

$$q(x) = 0 \Rightarrow p(x) = 0 \text{ for Lebesgue-almost all } x,$$

and in this case

$$D_{\mathrm{KL}}(P||Q) = \int p(x) \log\left(\frac{p(x)}{q(x)}\right) dx = \mathbb{E}_{X\sim P}\left[\log\left(\frac{p(X)}{q(X)}\right)\right].$$

In the special case where P, Q are discrete, the condition $P \ll Q$ is equivalent to $q(x) := Q[\{x\}], p(x) := P[\{x\}]$,

$$q(x) = 0 \Rightarrow p(x) = 0 \text{ for all } x,$$

and in this case

$$D_{\mathrm{KL}}(P||Q) = \sum_x p(x) \log\left(\frac{p(x)}{q(x)}\right).$$

Lemma 6.2.

1. *D_{KL} is well-defined (i.e. there does not appear a "$-\infty + \infty$" term).*

2.

$$D_{\mathrm{KL}}(P||Q) \geq 0$$

and

$$D_{\mathrm{KL}}(P||Q) = 0 \quad \Leftrightarrow \quad P = Q.$$

3. (Invariance to diffeomorphisms) Let $\Psi : \mathbb{R}^n \to \mathbb{R}^n$ be a diffeomorphism. Then

$$D_{\mathrm{KL}}(\Psi_* P||\Psi_* Q) = D_{\mathrm{KL}}(P||Q).$$

4. (Convexity)

$$D_{\mathrm{KL}}(\lambda P + (1-\lambda)P'||\lambda Q + (1-\lambda)Q') \leq \lambda D_{\mathrm{KL}}(P||Q) + (1-\lambda)D_{\mathrm{KL}}(P||Q)$$

Assume that we are interested in optimizing a family of probability densities p_θ over θ, so as to maximize

$$\log p_\theta(x).$$

This expression is called the **log-likelihood** of θ or the **evidence**.

We assume that the joint distribution with some latent variable Z is tractable, and given in the form

$$p^\theta_{X,Z}(X,Z) = p^\theta_{X|Z}(X|Z)p^\theta_Z(Z).$$

Tractable	**Untractable**
$p^\theta_Z(z),\ p^\theta_{X\|Z}(x\|z),\ p^\theta_{X,z}(x,z)$	$p^\theta_X(x) = \int p^\theta_{X,Z}(x,z)dz,\ p^\theta_{Z\|X}(z\|x) = \frac{p^\theta_{X,Z}(x,z)}{p^\theta_X(x)}$

Example 6.3. *(Mixture of Gaussians) This is a classical model for multi-modal distributions (i.e. distributions with several "peaks").* [20].

$$\mathbb{P}_Z[Z = i] = \phi_i, i = 1, ..., n$$
$$p^\theta_{X|Z}(x|z) = \mathcal{N}(x|\mu_z, \Sigma_z), z = 1, ..., n,$$

for some $\phi \in \mathbb{R}^n, \mu_z \in \mathbb{R}^d, \Sigma_z \in \mathbb{R}^{n\times d\times d}$, $z = 1, .., n$. Here $\theta = (\phi, \mu_1, \ldots, \mu_n, \Sigma_1, \ldots, \Sigma_n)$. In this case the marginal of X is in fact tractable:

$$p^\theta_X(x) = \sum_{i=1}^n \phi_i \mathcal{N}(x|\mu_i, \Sigma_i).$$

*But in general, the marginal $p^\theta_X(x)$ will not be tractable. Moreover, even here, determining the MLE is not easy (there is a sum in the likelihood), and methods like **Expectation Maximization (EM)** are used.*

[20] The distribution of Z is not a density, but a discrete measure. Mutatis mutandiso, the following discussion still applies.

Example 6.4. *Let $Z \in \mathbb{R}^{d_Z}$ be a random variable $Z \sim \mathcal{N}(0, I)$. Let $\mu_\theta : \mathbb{R}^{d_Z} \to \mathbb{R}^{d_X}$ $\Sigma_\theta : \mathbb{R}^{d_Z} \to \mathbb{R}^{d_X \times d_X}$ be neural networks with parameters θ, and let*

$$p_\theta(x|z) = \mathcal{N}(x|\mu_\theta(z), \Sigma_\theta(z)).$$

Then, in general, the marginal $p_\theta(x)$ is not tractable.

Let us now treat one observed data point x at a time. Goal: find θ such that $p^\theta(x)$ is large, i.e. determine the (arg)max of

$$\log p^\theta(x) = \log \int p^\theta(x|z)p^\theta(z)dz.$$

The usual approach of calculating this integral is via Monte Carlo sampling. Let $Z_1, \ldots, Z_M$ be i.i.d. samples from $p^\theta(z)$. Then

$$\log p^\theta(x) \approx \log \frac{1}{M} \sum_{i=1}^{M} p^\theta(x|Z_i).$$

Problem: usually, for many Z_i, $p^\theta_{X|Z}(x|Z_i)$ will be small.

Example 6.5. *In the case of a mixture of Gaussians, Example 6.3, for a sample X from $p^\theta(x)$, and Z from $p^\theta(z)$, $\mathcal{N}(X|\mu_Z, \Sigma_Z)$ will be small with high probability.*

PICTURE

This is a well-known problem in Monte Carlo sampling. One solution is the method of **importance sampling**. Let $q(z)$ be any probability density, satisfying

$$q(z) = 0 \Rightarrow p^\theta_Z(z) = 0, \qquad \text{for Lebesgue-almost all } z\ .$$

Then

$$p^\theta_X(x) = \int p^\theta_{X|Z}(x|z)p^\theta(z)dz = \int \frac{p^\theta_{X|Z}(x|z)p^\theta_Z(z)}{q_(z)} q(z)dz = \mathbb{E}_{Z\sim q(\bullet)}\left[\frac{p^\theta_{X|Z}(x|Z)p^\theta_Z(Z)}{q(Z)}\right].$$

With q we can "steer" the samples Z_i towards regions where $p^\theta(x|Z_i)$ is large. A naive choice would be $q(z) = p^\theta(z|x)$, but this is, in general, not tractable.

Instead, we choose a parametrized family $q(z) = q^\phi(z|x)$, where ϕ is a parameter vector. Then

$$\begin{aligned}
\log p^\theta_X(x) &= \log \mathbb{E}_{Z\sim q^\phi(\bullet|x)}\left[\frac{p^\theta_{X|Z}(x|Z)p^\theta(Z)}{q^\phi(Z|x)}\right] \\
&= \log \mathbb{E}_{Z\sim q^\phi(\bullet|x)}\left[\frac{p^\theta_{X,Z}(x,Z)}{q^\phi(Z|x)}\right] \\
&\geq \mathbb{E}_{Z\sim q^\phi(\bullet|x)}\left[\log \frac{p^\theta_{X,Z}(x,Z)}{q^\phi(Z|x)}\right] \\
&=: L_{\phi,\theta;x}.
\end{aligned}$$

Remark 6.6. *Why pull in the* log*?*

- *The quotient inside the expectation is unstable (when not pulling* log *in).*
- *Approach 2 (below) is not possible without pulling in the* log*.*

Since it is a *lower* bound on the evidence $\log p^\theta(x)$, it is called the **evidence lower bound** (ELBO),

$$\log p^\theta(x) \geq L_{\phi,\theta;x}.$$

How big is the gap between $\log p^\theta_X(x)$ and $L_{\phi,\theta;x}$?

$$\begin{aligned}
\log p^\theta_X(x) &= \int \log(p^\theta_X(x)) q_\phi(z|x) dz \\
&= \int \log\left(\frac{q^\phi(z|x)}{p^\theta_{Z|X}(z|x)} \frac{p^\theta_{X,Z}(x,z)}{q^\phi(z|x)}\right) q_\phi(z|x) dz \\
&= D_{\mathrm{KL}}\left(q^\phi(\cdot|x) || p^\theta_{Z|X}(\cdot|x)\right) - D_{\mathrm{KL}}\left(q^\phi(\cdot|x) || p^\theta_{X,Z}(x,\cdot)\right), \qquad (4)
\end{aligned}$$

where we abuse notation[21] and set

$$\begin{aligned}
L_{\phi,\theta;x} &:= -D_{\mathrm{KL}}\left(q^\phi(\cdot|x) || p^\theta_{X,Z}(x,\cdot)\right) \\
&:= -\int \log\left(\frac{q^\phi(z|x)}{p^\theta_{X,Z}(x,z)}\right) q_\phi(z|x) dz \\
&= \int \log\left(\frac{p^\theta_{X,Z}(x,z)}{q^\phi(z|x)}\right) q_\phi(z|x) dz \qquad \text{(ELBO)} \\
&= \mathbb{E}_{Z\sim q^\phi(\bullet|x)}\left[\log\left(\frac{p^\theta_{X,Z}(x,Z)}{q^\phi(Z|x)}\right)\right].
\end{aligned}$$

The first term in (4) is intractable, (because $p^\theta_{Z|X}(\cdot|x)$ is), but is non-negative (Lemma 6.2) and zero if and only if ϕ is chosen perfectly.

What is usually done is then to just maximize $L_{\phi,\theta;x}$ (over ϕ and θ).

Example 6.7. *For* $\theta, \phi \in \mathbb{R}_{>0}$*,*

$$\begin{aligned}
p^\theta_Z(z) &= \mathsf{Exp}(\theta)(z) = 1_{[0,\infty)}(z)\theta e^{-\theta z} \\
p^\theta_{X|Z}(x|z) &= \mathcal{N}(z,1) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(x-z)^2}{2}\right) \\
q^\phi(z|x) &= \mathsf{Exp}(\phi)(z).
\end{aligned}$$

[21] $p^\theta_{X,Z}(x,\cdot)$ is *not* a probability measure.

Then

$$\log p_X^\theta(x) = D_{\mathrm{KL}}\left(q^\phi(\bullet|x)||p_{Z|X}^\theta(\bullet|x)\right) + L(\theta,\phi;x),$$

where

$$\begin{aligned}
L(\theta,\phi;x) &:= -D_{\mathrm{KL}}\left(q^\phi(\bullet|x)||p_{X,Z}^\theta(x,\bullet)\right)\\
&= \int \log\left(\frac{p_{X,Z}^\theta(x,z)}{q^\phi(\bullet|x)}\right) q^\phi(z|x)dz\\
&= \int_0^\infty \log\left(\frac{\theta\exp(-\theta z)\frac{1}{\sqrt{2\pi}}\exp(-(x-z)^2/2)}{\phi\exp(-\phi z)}\right)\phi\exp(-\phi z)dz\\
&= \int_0^\infty \left(\log\theta - \theta z - \log(\sqrt{2\pi}) - \frac{(x-z)^2}{2} - \log\phi + \phi z\right)\phi\exp(-\phi z)dz\\
&= \log\theta - \frac{\theta}{\phi} - \log(\sqrt{2\pi}) - \frac{1}{2}(x^2 - 2x/\phi + 1/\phi^2) - \log\phi + 1.
\end{aligned}$$

MLE leads to

$$\begin{aligned}
0 &\overset{!}{=} \partial_\theta L = \frac{1}{\theta} - \frac{1}{\phi}\\
0 &\overset{!}{=} \partial_\phi L = \frac{\theta}{\phi^2} - \frac{x}{\phi^2} + \frac{1}{\phi^3} - \frac{1}{\phi}.
\end{aligned}$$

Hence $\theta = \phi$ and then

$$\begin{aligned}
0 &\overset{!}{=} \partial_\phi L = -\frac{x}{\phi^2} + \frac{1}{\phi^3}\\
0 &\overset{!}{=} \partial_\phi L = -\frac{x}{\phi} + \frac{1}{\phi^2},
\end{aligned}$$

hence

$$\phi = \frac{1}{x}.$$

For $x = 0$ there exists no maximum (the function grows as $\phi \to +\infty$).

For $x < 0$ the maximum is not at an allowed parameter (and the function grows as $\phi \to +\infty$).

We summarize the above discussion in the following algorithm of a **variational autoencoder** (VAE).

Given a family of probability densities $p_{X|Z}^\theta(x|z), p_Z^\theta(z)$ and a family of probability measures $q^\phi(z|x)$, we want to determine θ, ϕ maximizing (ELBO). (Or more to the point: a sums over realizations of x from the unknown data distribution.)

Approach 0:

$$\mathbb{E}_{Z\sim q^\phi(\bullet|x)}\left[\log\left(\frac{p^\theta(x,Z)}{q^\phi(Z|x)}\right)\right] \approx \frac{1}{N}\sum_{m=1}^{M}\log\left(\frac{p^\theta(x,Z_m)}{q^\phi(Z_m|x)}\right),$$

where $Z_1,\dots,Z_M \sim q^\phi(\bullet|x)$ iid. Problem: how to take automatic derivatives w.r.t. ϕ here (the distribution of the Z_m depends on ϕ!). Of course, we can take a numerical derivative, but this is very inefficient.

Approach 1: Assume that there is a distribution ν and a function g^ϕ such that for $\epsilon \sim \nu$ we have

$$g^\phi(x,\epsilon) \sim q^\phi(\bullet|x).$$

The canonical example is: $q^\phi(\bullet|x) \sim \mathcal{N}(\mu(x),\Sigma(x))$, $\epsilon \sim \mathcal{N}(0, I_{d_Z})$,

$$g^\phi(x,\epsilon) = \mu(x) + \Sigma(x)^{1/2}\epsilon.$$

Then we can approximate

$$ELBO \approx \frac{1}{N}\sum_{i=1}^{N}\log\left(\frac{p^\theta_{X,Z}(x, g^\phi(x,\epsilon_i))}{q^\phi(g^\phi(x,\epsilon_i)|x)}\right),$$

where $\epsilon_1,\dots,\epsilon_N \sim \nu$ i.i.d. This is called the **reparametrization trick**. Now we have an expressions that is automatically differentiable w.r.t. ϕ.

Approach 2: Write

$$\begin{aligned}
L_{\phi,\theta;x} &= \int \log\left(\frac{p^\theta(x,z)}{q^\phi(z|x)}\right) q_\phi(z|x)dz \\
&= \int \log\left(\frac{p^\theta(z)}{q^\phi(z|x)}p^\theta(x|z)\right) q_\phi(z|x)dz \\
&= D_{\mathrm{KL}}(p^\theta_Z||q^\phi(\bullet||x)) + \mathbb{E}_{Z\sim q^\phi(\bullet|x)}\left[\log\left(p^\theta(x|Z)\right)\right].
\end{aligned}$$

In many cases the first term is analytically known. Then we can approximate

$$L_{\phi,\theta;x} \approx D_{\mathrm{KL}}(p^\theta_Z||q^\phi(\bullet||x)) + \frac{1}{N}\sum_{i=1}^{N}\log\left(p^\theta_{X|Z}(x|g^\phi(x,\epsilon_i))\right),$$

where $\epsilon_1,\dots,\epsilon_N \sim \nu$ i.i.d.

Approach 3: see [RMW14].

Example 6.8. *Continuing Example 6.3, taking*

$$q^\phi(z|x) = \mathcal{N}(z|\mu_\phi(x), \sigma^2_\phi(x) I_{d_Z}),$$

we get, using the reparametrization trick,

$$\begin{aligned}
\mathsf{loss}(X_1, ..., X_N) &= \frac{1}{N}\sum_{n=1}^{N}\left(D_{\mathrm{KL}}(p^\theta_Z || q^\phi(\bullet||X_i)) + \frac{1}{M}\sum_{m=1}^{M} \log\left(p^\theta_{X|Z}(X_i|g^\phi(X_i,\epsilon_m))\right)\right) \\
&= \sum_{n=1}^{N}\left(D_{\mathrm{KL}}(p^\theta_Z || q^\phi(\bullet||X_i)) + \frac{1}{M} c \sum_{i=1}^{M} ||X_i - g^\phi(X_i,\epsilon_i)||^2\right) \\
&= \sum_{n=1}^{N}\left(D_{\mathrm{KL}}(p^\theta_Z || q^\phi(\bullet||X_i)) + \frac{1}{M} c \sum_{i=1}^{M} ||X_i - g^\phi(X_i,\epsilon_i)||^2 \sum_{i=1}^{N} ||X_i - (\mu_\phi(X_i) + \epsilon_i \sigma^2_\phi(X_i))||^2\right)
\end{aligned}$$

And

$$D_{\mathrm{KL}}(p^\theta_Z || q^\phi(\bullet||X_i)) = D_{\mathrm{KL}}(\mathcal{N}(0, I_{d_Z}) || \mathcal{N}(\mu_\phi(X_i), \sigma^2_\phi(X_i) I_{d_Z})) = \frac{1}{2}\left(\log\frac{\sigma^2_\phi(X_i)}{1} + \frac{\sigma^2_\phi(X_i)}{1} + \mu_\phi(X_i)^2 - 1\right)$$

Remark 6.9.

All implementations of vanilla VAE seem to assume that:

- *The "decoder" $p^\theta_{X|Z}(x|z)$ is Gaussian with fixed (untrainable) diagonal covariance matrix.*
- *The encoder $q^\phi(z|x)$ is Gaussian with trainable mean and diagonal covariance matrix.*

References

[BD81] Bruno Brosowski and Frank Deutsch. An elementary proof of the stone-weierstrass theorem. *Proceedings of the American Mathematical Society*, 81(1):89–92, 1981.

[BKH16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[Cal20] Ovidiu Calin. *Deep learning architectures*. Springer, 2020.

[Fou22] Simon Foucart. *Mathematical Pictures at a Data Science Exhibition*. Cambridge University Press, 2022.

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference*

on artificial intelligence and statistics, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[GG23] Guillaume Garrigos and Robert M Gower. Handbook of convergence theorems for (stochastic) gradient methods. *arXiv preprint arXiv:2301.11235*, 2023.

[HAB19] Matthias Hein, Maksym Andriushchenko, and Julian Bitterwolf. Why relu networks yield high-confidence predictions far away from the training data and how to mitigate the problem. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 41–50, 2019.

[HS17] Boris Hanin and Mark Sellke. Approximating continuous functions by relu nets of minimal width. *arXiv preprint arXiv:1710.11278*, 2017.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[KB] DP Kingma and JL Ba. Adam: a method for stochastic optimization. proceedings of the 3rd international conference on learning representations, iclr 2015—conference track proceedings. international conference on learning representations, iclr; 2015. *CA, USA*.

[Kle13] Achim Klenke. *Probability theory: a comprehensive course*. Springer Science & Business Media, 2013.

[KSH17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[Lan12] Serge Lang. *Real and functional analysis*, volume 142. Springer Science & Business Media, 2012.

[Loo13] Lynn H Loomis. *Introduction to abstract harmonic analysis*. Courier Corporation, 2013.

[MPCB14] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. *Advances in neural information processing systems*, 27, 2014.

[NZL20] Gregory Naitzat, Andrey Zhitnikov, and Lek-Heng Lim. Topology of deep neural networks. *The Journal of Machine Learning Research*, 21(1):7503–7542, 2020.

[RMW14] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pages 1278–1286. PMLR, 2014.

[RR07] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007.

[SLP+21] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.

[Tel16] Matus Telgarsky. Benefits of depth in neural networks. In *Conference on learning theory*, pages 1517–1539. PMLR, 2016.

[Ton12] Yung Liang Tong. *The multivariate normal distribution*. Springer Science & Business Media, 2012.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.