You're your own best teacher: A Self-Supervised Learning Approach For Expressive Representations

Johan Vik Mathisen

May 20, 2024

TODO: Introduce the section, what we think and the philosophy of presenting material in such a way.

0.1 Notation

- Encoder E
- Decoder D
- Estimated values are presented with a hat, $\widehat{x}$ for a reconstructed value, $\widehat{f}$ for a trained model etc.
- Parameters θ
- Dataset $X = \{x_i\}_{i=1}^N$

0.2 Information theoretic/basic stats used in evaluation

- maximum entropy distribution

- mutual information

Definition 1 (Differential entropy) *The differential entropy of a random variable X with pdf or pmf p defined on a sample space $\mathcal{X}$ is*

$$h(X) = \mathbb{E}[-\log(f(X))] - \sum_{x \in \mathcal{X}} p(x) \log(p(x)), \text{ if } X \text{ is discrete,} \tag{1}$$

$$h(X) = \mathbb{E}[-\log(f(X))] - \int_{\mathcal{X}} p(x) \log(p(x)), \text{ if } X \text{ is continuous} \tag{2}$$

- perplexity

Definition 2 (KL-Divergence) *For probability distribution P and Q defined on the same sample space $\mathcal{X}$, if for all $x \in \mathcal{X}$ $Q(x) = 0$ implies $P(x) = 0$, the Kullback-Leibler divergence is defined as*

$$\mathrm{KL}(P||Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right), \text{ if } P \text{ and } Q \text{ are discrete,} \tag{3}$$

$$\mathrm{KL}(P||Q) = \int_x p(x) \log\left(\frac{p(x)}{q(x)}\right) dx, \text{ if } P \text{ and } Q \text{ are continuous.} \tag{4}$$

`https://stats.stackexchange.com/questions/188903/intuition-on-the-kullback-leibler-kl-` `189758#189758`

- Cross entropy

- Graphical probabilistic models - Ancestral sampling

- Generative models

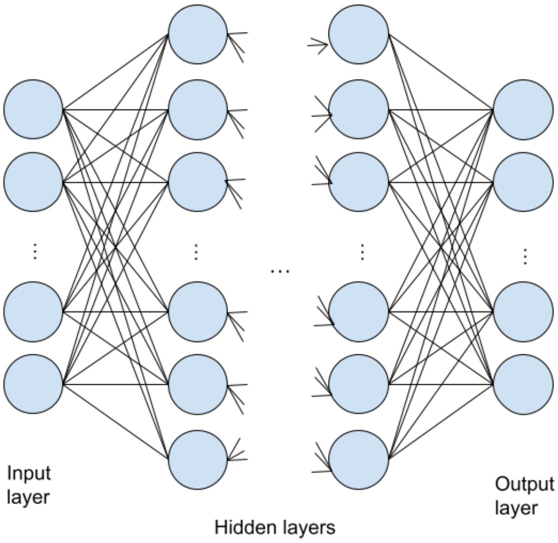

Figure 1: Illustration of a Neural Network model.

0.3 Time Series Inference

- short-time-fourier transform etc.

0.4 Neural Network

An *artificial neural network* or simply *neural network* is a fundamental model in machine learning, and more specifically in *deep learning*. Neural networks are loosely inspired by the way neurons are assembled in the brain. The model can be traced back the year of 1943 when Warren McCulloch and Walter Pitts developed the first artificial neuron [**MCCULLOCH199099**], which is considered to be the first neural model invented. It was first set out in the real world by Frank Rosenblatt in 1957 [**rosenblatt1957perceptron**]. But not until the development of the backpropagation algorithm in its modern form in the 1980's did the model really gain traction. Neural networks have since then been the highly influential in the development of the machine learning field, with an impressive resume of applications. Some of which, if we stretch the definition a bit, include face recognition, beating humans in chess, Go and Starcraft, self-driving cars and predicting the structure of proteins. The unreasonable effectiveness of neural networks on a broad range of tasks can in part be explained by the *universal approximation theorem*, proven by Kurt Hornlik in 1989 [**HORNIK1989359**], which roughly states that a neural network can approximate any (Borel measurable) function to any desired degree of accuracy.

A neural network takes in a vector $x \in \mathbb{R}^n$ and builds a nonlinear function $f(x)$ to predict the response $y \in \mathbb{R}^m$. More specifically a neural network maps an input vector x to an output vector y through a series of non-linear functions of linear combinations of the input. This particular structure, presented in figure **??** is what distinguishes neural networks from other nonlinear prediction models. The variables $x = [x_1, ..., x_n]$ constitutes the units of the *input layer*. The intermediate layers are called the *hidden layers*, and the final mapping to y is called the *output layer*. A neural network is parameterized by a set of *weight* matrices W_i and *bias* vectors b_i, together with a specified non-linear *activation function* σ. We denote the collection of parameters $\{W_1, ..., W_{K-1}, b_1, ..., b_{K-1}\}$ by θ. Written out a K layered neural network is given by

$$f_\theta(x) = f_K \circ f_{K-1} \circ \ldots \circ f_2 \circ f_1(x),$$

where

$$f_i(x) = \sigma(W_i x + b_i), \quad i \in \{1, ..., K-1\},$$

and f_K is the output layer, with application dependent structure.
The introduction of nonlinearity by the activation function is what enables the model to approximate nonlinear signals and differentiate itself from a linear regression model. Two of the most commonly used activation functions are $\text{Sigmoid}(x) = \frac{1}{1+\exp(-x)}$ and $\text{ReLU}(x) = \max(0, x)$, but countless options exists.
The architecture of neural networks, and most specializations thereof, is sequential in nature. They can effectively be described as compositions of some combination of a flavour of matrix multiplication, non-linear transformation and down or upsampling.

0.4.1 Training Neural Networks

The training of a neural network is the process of finding values for the weight and bias parameters. The general idea governing neural network training is to optimize the parameters based on some distance metric between the predicted values and the target values. This distance metric is referred to as the *loss function*, and a common choice of loss function is the mean squared error (MSE).

For a multi-variable function F which is differentiable around a point a, the direction which it decreases fastest when starting in the point a is given by the negative gradient at a, $-\nabla F(a)$. The gradient descent algorithm utilizes this property to find local minima of F by initializing the function with a value x_0 and iteratively update

$$x_{n+1} = x_n - \gamma \nabla F(x_n).$$

If the *learning rate* γ is small enough we are guaranteed that $F(x_{n+1}) \geq F(x_n)$, and the sequence $x_0, x_1, \ldots$ converge to a local minima of F.

A neural network $f_\theta(x)$ is itself a multi-variable function, and as long as the loss and activation function are differentiable, the network is as well, both in terms of it argument and parameters. If one differentiates the network with respect to its parameters across its domain, the negative gradient indicates the fastest direction in which to update the parameters to minimize the loss. The optimization problem at hand, for a dataset $X = \{x_i\}_{i=1}^N$ with corresponding labels $Y = \{y_i\}_{i=1}^N$, is

$$\widehat{\theta} = \min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f_\theta(x_i), y_i),$$

and one would iteratively update the parameters by gradient descent

$$\theta_{n+1} = \theta_n - \frac{\gamma}{N} \sum_{i=1}^{N} \nabla_\theta \mathcal{L}(f_\theta(x_i), y_i).$$

If the dataset X is large, the gradient calculations are expensive. In these cases, which in modern machine learning is the standard scenario, an effect estimator for the true gradient is used instead. Stochastic gradient descent (SDG) is a very popular approach. Instead of computing the gradient at each datapoint every iteration, SDG updates the parameters by iterating through the dataset and using the gradient at each single datapoint. The dataset is then permuted and iterated through again until an approximate minimum is reached. Pseudocode of SDG is presented in algorithm **??**.

Algorithm 1 Stochastic Gradient Descent (SDG)

```
Initialize parameters $\theta$ and learning rate $\gamma$
while Not converged do
    Permute training set $(X, Y)$
    for $i$ in $1, \dots, N$ do
        $\theta \leftarrow \theta - \gamma \nabla_\theta \mathcal{L}(f_\theta(x_i), y_i)$
    end for
end while
```

TODO: Mini batch SDG

For the actual gradient computations, the backpropagation algorithm is used. It provides an efficient way of computing gradients in neural networks by leveraging the compositional structure. In essence backpropagation is an efficient application of the Leibniz chain rule for differentiation.

For a through introduction to the subject of neural networks and the training thereof we refer to chapter 6 and 8 of [**deeplearningbook**].

0.5 Convolutional Neural Network

This section draw inspiration on the presentation of convolutional networks in chapter 9 of [**deeplearningbook**].

A convolutional neural network (CNN) is a particular type of neural network that is developed to learn local features in the data. This local feature learning is enabled by the mathematical operation of convolution. In essence a CNN is a neural network where matrix multiplication is switched for convolution at least one of the layers [**deeplearningbook**].
Fully connected neural networks have a fundamental drawback in that their computational complexity grows intractably large when the input dimensionality is high. This makes them unsuited for high dimensional data, such as images. Convolutional neural networks directly address this issue via various downsampling techniques. Convolutional neural networks has a rich history and a long track record of success stories. They are inspired by the architecture of visual cortex cells in mammals. Inspired by the discovery of Hubel and Wiesel [**https://doi.org/10.1113/jphysiol.1968.sp00**
the neocognitron was proposed in 1980[**6313076**]. The neocognitron is widely considered the predecessor of convolutional neural networks. In 1989 Yann LeCun et al. introduced the modern framework for CNNs [**LeCun1989ConvNet**] and demonstrated its effectiveness on the task of hand written digit recognition. Since then CNNs have been an indispensable part of machine learning research, especially in the computer vision domain. For an exposition on the advances on convolutional neural networks and its applications we refer to [**gu2017recent**].

0.5.1 The convolution operation

The convolution operation is an integral transform with extensive applications. It generalizes the notion of a moving weighted average. In mathematics it is ubiquitous because of its relationship with the Fourier transform.

Let f and g be real valued functions, then their convolution is defined as

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{5}$$

The mathematical nuances of the exact criteria for the above integral to exist is outside the scope of this thesis, and not particularly relevant. But if f and g are integrable (in the Riemann or Lebesgue sense) then the convolution exists. As a rule of thumb, the convolution of f and g is as "smooth" as the smoothest of f and g. It is worth mentioning that convolution is commutative, i.e that $f * g = g * f$, which can be seen by a simple change of variables.

As is typical for integral transforms, the function g is referred to as the *kernel*. In the context of convolutional networks the kernel consists of learnable parameters and the function f is the *input*. The output is referred to as the *feature map* [**deeplearningbook**]. In machine learning we handle discrete signals, represented as multidimensional arrays. As a result we must employ a discrete variation of the convolution operation. Let I be the input and K be the kernel, both discrete, then their convolution is defined as

$$(I * K)[n] = \sum_{m=-\infty}^{\infty} I[m]K[n-m]. \tag{6}$$

In practice I and K typically has finite support, i.e they are zero for large positive and negative arguments, which circumvents any convergence problem.
Convolutions are naturally defined for higher dimensional functions by component wise extension. For a two dimensional image I and a kernel K we calculate their convolution as

$$(I * K)[i,j] = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} I[n,m]K[i-n,j-m]. \tag{7}$$

The operation is illustrated in figure **??**.

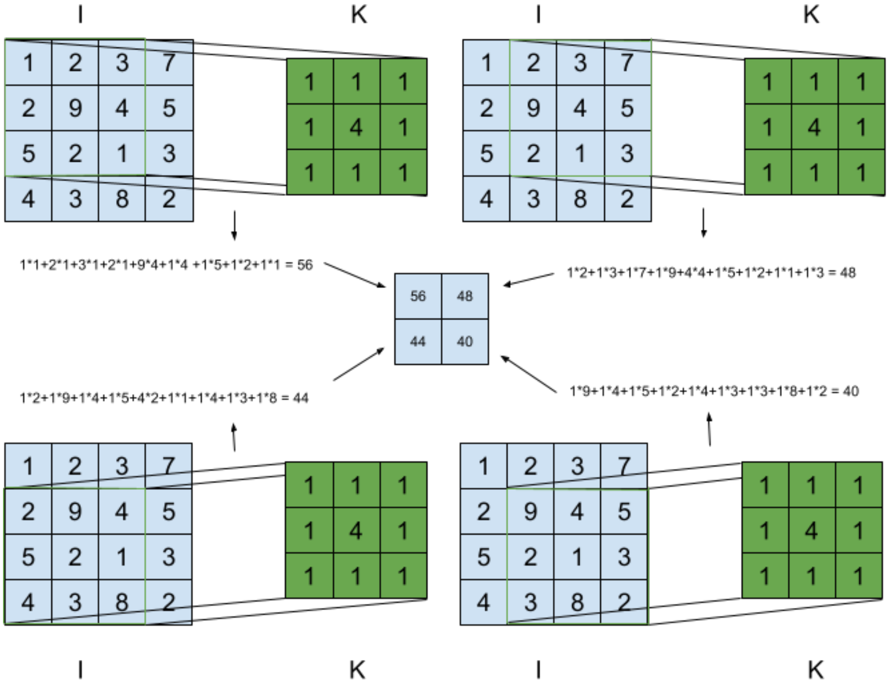

Figure 2: Illustration of discrete two dimensional convolution

Convolution in machine learning does not always correspond exactly to the mathematical definition of the operation, but rather to cross-correlation. The difference is just a sign flip in the kernel arguments. Operation is no longer commutative, but in practice this does not affect anything as the learned kernel parameters will be equivalent [**deeplearningbook**].

In the field of digital signal processing discrete convolutions are used extensively. Traditionally predefined kernels are used to alter a signal in predictable ways. Two well known kernels are Gaussian and edge detection kernels. Gaussian kernels are two dimensional gaussian distributions and are used for blurring. Edge detection kernels are designed in such a way that areas of similar intensity are mapped to 0 and areas of variable intensity are mapped to high values. In figure **??** we demonstrate the effect of two such kernels. Edge detection in particular provide a hint to how CNNs learn interesting features in the data. In a convolutional layer the kernels are learned such that the feature map is helpful for the training objective.

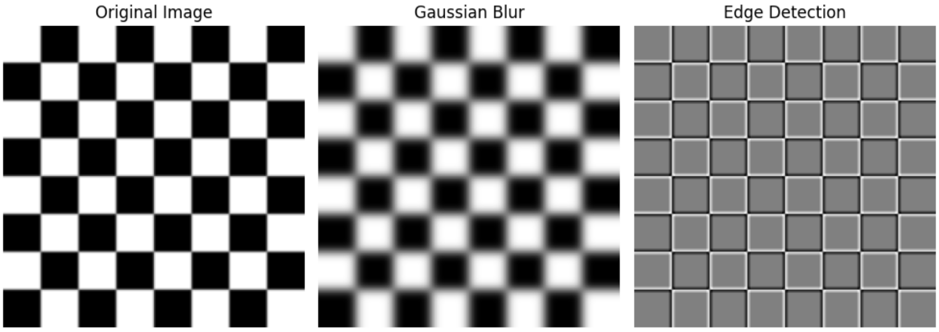

Figure 3: Illustration of discrete convolution applied to images. Image part of Scikit-image data package.

0.5.2 Pooling

As mentioned earlier CNNs are mainly used when the data has high dimensionality. In order to reduce the dimension to a manageable level *pooling* is used. A pooling operation is applied as a down sample technique on feature maps, replacing regions of the output with summary statistics. Two of the most common are max and average pooling, which replaces the region by its maximal or average value respectively. There are two hyperparameters for any pooling operation, the filter size, which determines the region of values to calculate the summary statistic, and stride length, which determines how the filter moves across the feature map. In addition to dimension reduction, pooling assist in making the representations approximately invariant to small distortions of input. Illustrations of max pooling with different stride is presented in figure **??** and **??**, while the effect of max and average pooling on image data is illustrated in **??**.

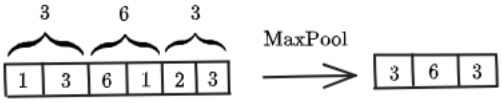

Figure 4: Max pooling of one dimensional array. Filter size: 2, stride: 2.

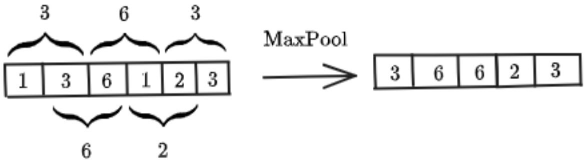

Figure 5: Max pooling of one dimensional array. Filter size: 2, stride: 1.

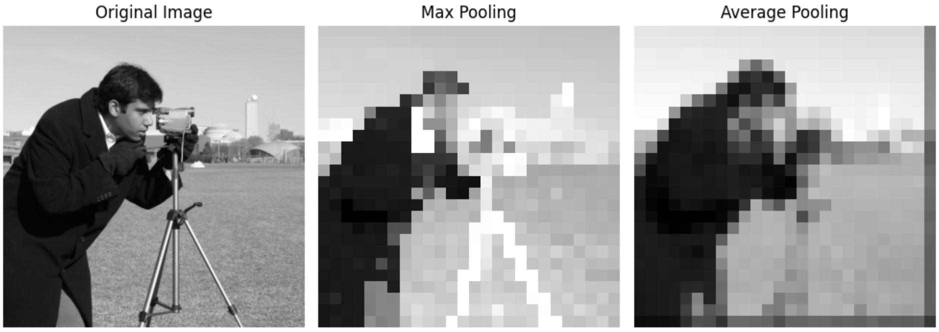

Figure 6: Illustration of mean and average pooling applied to images. Filter size and stride is 20×20. Original image size is 512×512, pooled images are 26×26. Image part of Scikit-image data package.

0.5.3 Architecture

There are a large variety of specific architectures which fall in the category of a CNN, but their basic components are largely the same. They consist of convolutional layers, pooling layers and fully connected layers. In figure **??** an illustation of the original LeNet-5 [**LeCun1989ConvNet**] is presented as an illustration of a general CNN.

A convolutional layer consists of several kernels used to compute different *feature maps*. Each kernel convolved with entire input, and the different feature maps are produced by changing kernel. A nonlinear activation function is then applied pointwise to the feature maps. A pooling layer is typically placed between convolutional layers and work as a stronger downsampler which aims to enforce approximate translation invariance. A pooling operation is applied across each feature map.

The fully connected layer is just your typical hidden layer in a neural network, i.e connect every input to every node in the output. Because of computational issues mentioned earlier fully connected layers are first introduced when the input data has been sufficiently downsampled.

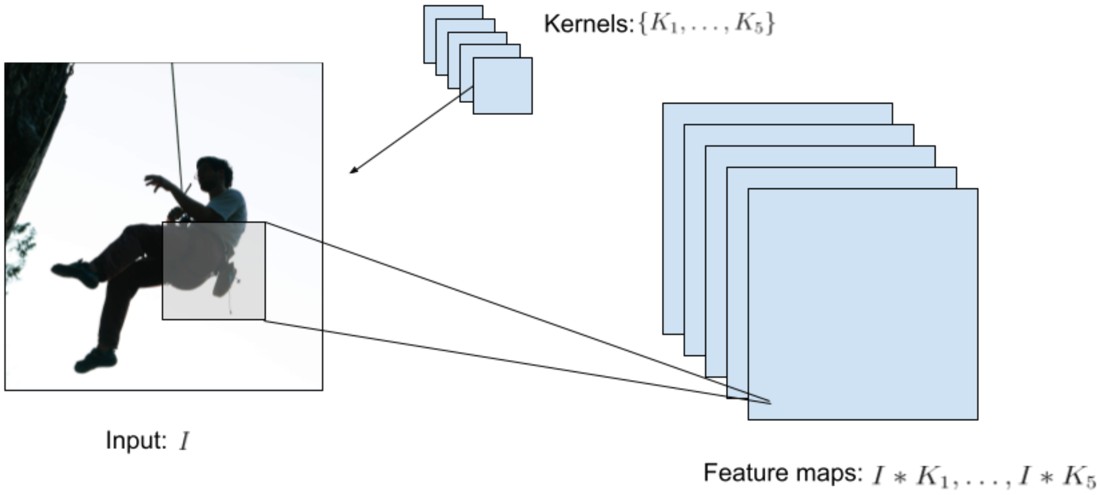
Kernels: $\{K_1, \ldots, K_5\}$
Input: I
Feature maps: $I * K_1, \ldots, I * K_5$

Figure 7: Convolutional layer.

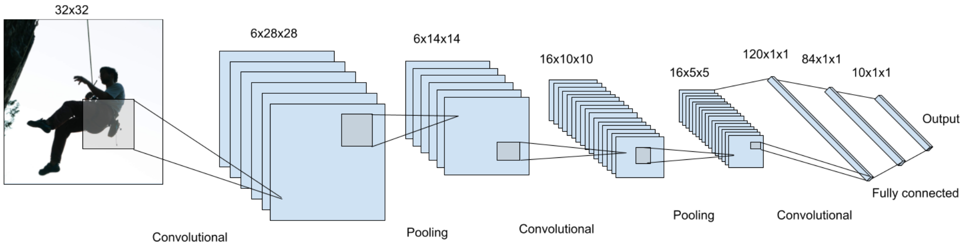

Figure 8: LeNet-5 network [**LeCun1989ConvNet**]

0.5.4 Transposed Convolutional Networks

Transposed convolution or deconvolution, also known as fractionally-strided convolution is a technique used to reverse the downsampling from convolutions. In essence it is an inpainting or upsampling technique known from digital signal processing. The flexibility of learning data dependent transposed convolutional kernels enables one to more effectively reverse the downsampling from convolutional layers. They are extensively used in combination with convolutional downsampling in *encoder-decoder architectures* presented in section **??**.

0.6 Representation Learning

The focus of this thesis is representation learning for improved time series generation.

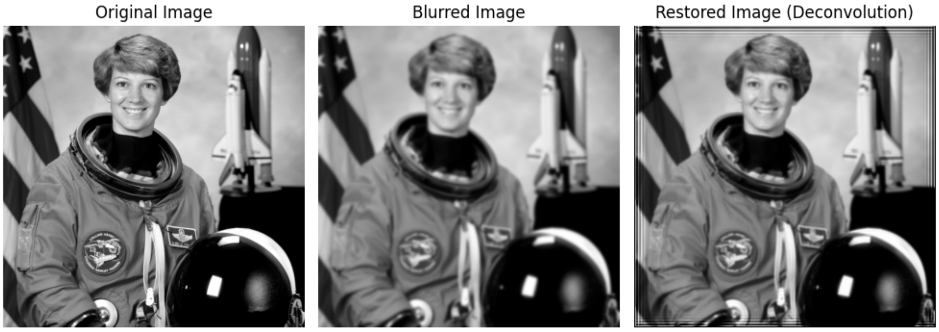

Figure 9: Illustration of deconvolution applied to images. Here using Gaussian blur and restoring using the Richardson Lucy algorithm with a Gaussian deconvolution kernel. Image part of Scikit-image data package.

0.6.1 Why do we care about representation learning?

Those who has worked with data science or machine learning and has come across feature engineering are familiar the effect good feature engineering has on a models performance. The same people too knows the level of domain expertise, creativity and time is needed to feature engineer well. In reality much of the actual time spent in the process of deploying machine learning methods revolves around constructing good data pipelines and applying transformations that produce representations beneficial for the algorithm at hand [**Rep-rev-persp**]. Thus the ability to automate such tasks would be incredibly beneficial, and ease the use of ML algorithms significantly. It is here one of the intriguing and promising features of neural networks, with its many specializations and architectures, comes into play. They have shown the ability to learn useful abstract representations of the data and provide automatic feature engineering [**Rep-rev-persp**].

0.6.2 What a representation?

Representation learning is a term not too easily defined, one reason being the abstraction level. It is helpful to first consider what is meant by *representation* of information. Lets begin by walking through a familiar and illustrative example. Consider the base ten integer $(4)_{10} = 4$. The number can equivalently (in terms of information content) be expressed, that is represented, in any other base. The particular base we choose depends on our intention with the number. If we want to work with digital electronics, a binary representation ($(4)_2 = 10$) is very useful, as transistors has two states. When humans do arithmetic, base ten representations of the integers are very natural, as we have ten fingers. A particular representation of information can make a task easier or harder. The information content is unchanged by a change of representation. What is changed is the easiness or difficulty of certain information processing tasks. Representation learning is then the process of learning a certain representation of information.

Representations are too highly dependent on who, or what, that will process it. An example is time. Humans have developed a standardized system for writing timestamps which works fairly well for us. But, if we want to model time dependent phenomena, say using tabular data, the DateTime representation is of very little help to a tree based model for instance. The reason being that the numerical representation of timestamps close in time is not necessarily close in numerical value. Think of 23:59 to 00:00. A possible solution is to change the representation such that the numerical values actually respect the periodic nature by mapping to the circle. The new representation is then useless for humans, but quite a lot more useful to a computer. Representation learning in machine learning can thus be thought of as algorithms which are designed to learn representations which is useful for a ML objective.

0.6.3 What is a good representation?

The goodness of a representation is then determined by how easy it makes a subsequent task. The concept of universally good representations is ill-defined, for any representation extracted of a non-invertible function, a *downstream tasks* can be designed (in principle) to based on the lost information, hence achieve arbitrarily bad performance. There is no free lunch in representation learning either. One must specify a set of predefined downstream tasks, and evaluate according to those. Intuitively the quality of the representations are also considered higher if the the representations is able to perform well on several downstream tasks i.e when they are more general.

0.6.4 How does one evaluate representations?

There are several different evaluation protocols in representation learning. They involve training a model on a *pretext task*, which as defined in [**jing2019selfsupervised**] is a task for a network to solve, where the goal is to learn representation. Then the learned representations are evaluated on a downstream task. In general the downstream task is solved in a supervised manner, using human annotated data.

In a N-layered network $f = f_N \circ ... \circ f_1$, the intermediate value of the data x in some layer n is what is meant by the networks learned feature representations. When we are interested in the representations learned it thus is helpful to dissect a model f, notation wise, into a *feature extractor* h and an *output function* g such that it can be factored as $f = g \circ h$. Representation learning algorithms typically follow the pattern

- Train $f = g \circ h$ on a pretext task.
- Discard g
- Use the learned feature extractor $\widehat{h}$ as part of a new model.
- Evaluate the new model on the downstream task.

The standard evaluation protocol is to train a linear head g_D on top of the *frozen* representations in a supervised manner and evaluate this models performance. This is to say that we train $f_D = g_D \circ \widehat{h}$ by only updating the parameters of the linear model g_D, and evaluate $\widehat{f_D}$ on some test set. This protocol is sometimes referred to as *linear probing*. A common downstream task is classification, where the idea is that good and informative representations should differentiate data in such a way that it is easy to separate them.

An common alternative protocol is similar to the one above, but rather than freezing the feature extractor one let all parameters of f_D to be learnable on the downstream task. This protocol is referred to as a pretraining-finetuning.

It is of interest how f_D performs in terms of accuracy on the downstream task and training time, and too how these sensitive metrics are to training data size. The baseline comparison would then be an identical but randomly initialized model. It is considered highly advantageous if one is able to pretrain a feature extractor using cheap and abundant data in a way which ensures faster convergence on a downstream task where data is expensive or scarce.

It is with mentioning that in cases where the sole goal is to create the best performing model, a more complex task specific head g_D is often used. For a comprehensive survey on representation learning we refer the reader to [**nozawa2022empirical**].

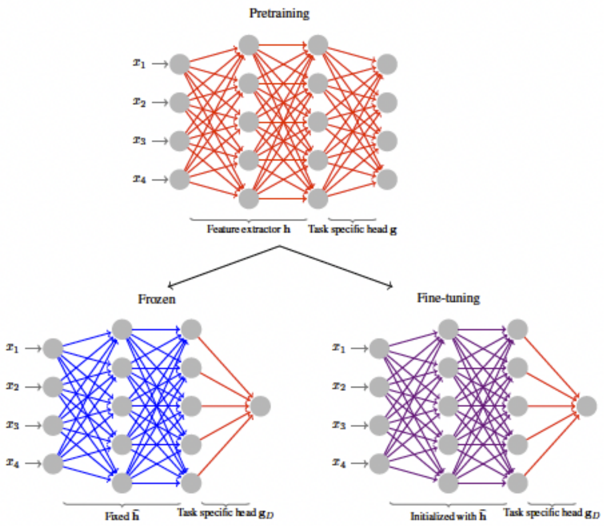

Figure 10: Taken from [**nozawa2022empirical**]

0.7 Transformers

Since their introduction in the seminal paper titled "Attention is all you need"[**vaswani2023attention**] transformers have taken the field of machine learning and artificial intelligence by storm. It is the enabling architecture behind large language models (LLMs) such as Googles BERT, Metas Llama and OpenAIs ChatGPT. Drawing inspiration from the initial success in natural language processing vision transformers [**dosovitskiy2021image**] were developed for computer vision applications, and similar trends are shown for other modalities such as audio [**latif2023transformers**] and time series [**wen2023transformers**].

Transformers were developed for sequential modelling and has capabilities of learning far ranging dependencies in data through the *multi-headed attention mechanism*. In the realm of sequential modelling recurrent neural networks and long short-term memory neural networks were the previous state of the art, but modeling long term dependencies has proven difficult [**279181**] as one encounters the *vanishing or exploding gradient problem*. One of the main novelties of the transformer architecture is not relying on recurrence, and instead solely using the attention mechanism to capture dependencies between input and output.

cite

cite

The transformer takes as input a sequence of symbols , referred to as the *input embedding* in **??**. This means in particular that for a specified modality, e.g text, speech, image and time series, they must first be translated into a sequence of vectors. This is accomplished by a process called *tokenization*. The conceptually easiest word-level tokenization in NLP, where one creates a dictionary of all words in the dataset, and assigns to them an integer value. A piece of text can then with relative ease be mapped to a sequence of integers. Other modalities as speech, image and time series are usually represented as matrices and can therefore naively be modeled as a sequence numbers. Because of the high dimensionality, this would require a significant amount of computational resources. In addition the transformer would need to model incredibly far ranging dependencies and handle redundant information. Hence various tokenization methods are used to represent the data as coarser sequences.

Transformers can be adopted for generative tasks, and has done so with great success. Autoregressive transformers such as the GPT series are trained to predict the next token in a sequence given the preceding tokens. By treating images as sequences of patches vision transformers can generate new images or inpaint missing patches conditional on the input. With the introduction if vision transformers [**dosovitskiy2021image**], input images were split into patches and linearly projected before being processed by the transformer. Since then *Vector-Quantized*-based tokenization introduced by [**VQVAE**] has emerged as a popular approach.

0.7.1 The attention mechanism

"Attention can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values and outputs are all vectors".

TODO: Wait for 3b1b video for context

Scaled dot-product attention,

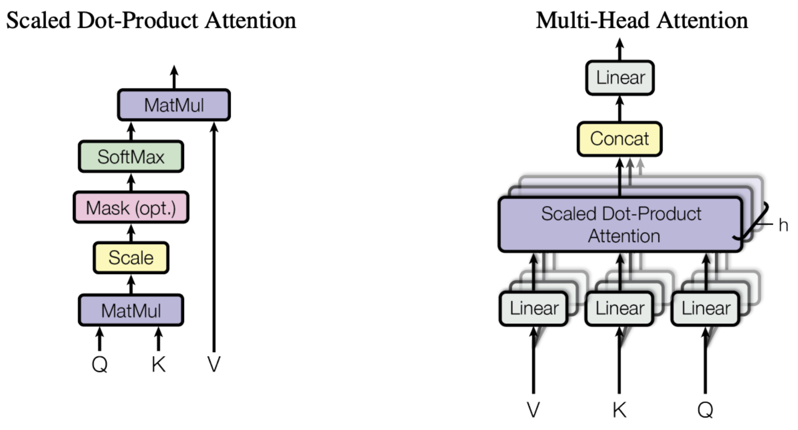

Figure 11: (left) Scaled dot-product attention. (right) Multi-headed attention. Taken with permission from [**vaswani2023attention**]

$$\text{Attention}(Q,K,V) = \text{SoftMax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{8}$$

Multi-headed attention

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,\dots,\text{head}_h)W^O \tag{9}$$

where

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{10}$$

0.7.2 Architecture

Encoder - decoder architecture.

Tokenizer

Positional encoding

Transformer layer (encoder/decoder)

0.7.3 Bi-directional transformer (BERT)

Encoder only transformer. Unidirectional

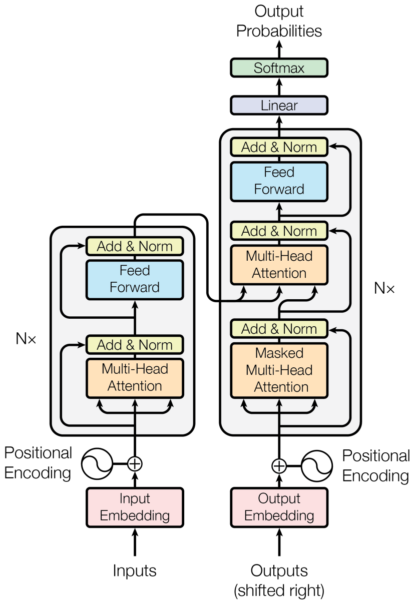

Figure 12: The Transformer - model architecture. Encoder (left), decoder (right). Taken with permission from [**vaswani2023attention**]

0.8 Self-Supervised learning

Self-supervised learning (SSL) has had great success in natural language processing and computer vision in recent years.

Machine learning can be coarsely divided into two classes, supervised and unsupervised learning.

0.8.1 Supervised

Supervised learning refers to models who learn using labeled data. That is to say for a given input x we already know what the desired output y is during training, and can therefore supervise (update) our models parameters by directly comparing model output and the true value. A bit more formally, for a dataset $X = \{x_i\}_{i=1}^N$ with corresponding human annotated labels $Y = \{y_i\}_{i=1}^N$, the objective of a supervised learning algorithm is to fit f_θ in such a way that the loss across the data is minimized

$$\widehat{f_\theta} = \min_\theta \frac{1}{N} \sum_{i=1}^{N} L(f_\theta(x_i), y_i). \tag{11}$$

Common approaches to supervised learning for neural networks is to calculate some distance metric between the predicted value $\widehat{y}$ and the true value y and update parameters by backpropagation.

The models falling under the supervised learning category are widely deployed and has seen tremendous success. Classical statistical models, as well as support vector machines and decision tree based models are all examples of models in this learning paradigm. The main issue with supervised learning is the need for labeled data, and labeled data is in many ways scarce.

0.8.2 Unsupervised

Unsupervised learning on the other hand refers to models or algorithms who learn exclusively from unlabeled data. That is to say that the models learn intrinsic patterns in the data. Examples of unsupervised learning models are clustering methods as K-means, K Nearest Neighbor and Gaussian mixture models, dimension reduction techniques as PCA/SVD and neural network architectures such as Autoencoders.

TODO: Where is it used an why?

Exploratory data analysis, data visualization, clustering

TODO: Pros and cons of unsupervised learning

Unlabeled data is cheap,

0.8.3 Self-supervised

Self-supervised learning is subcategory of unsupervised learning and refers to model who use the data itself to generate a supervisory signal, rather than external labels as in supervised learning. Even as SSL is considered unsupervised learning, the learning formulation is quite similar to that of supervised learning.

For a dataset $X = \{x_i\}_{i=1}^N$ with *pseudo labels* $P = \{p_i\}_{i=1}^N$ the objective of a self-supervised learning algorithm is to fit f_θ in such a way that the loss across the data is minimized. In other words find

$$\widehat{f_\theta} = \min_\theta \frac{1}{N} \sum_{i=1}^{N} L(f_\theta(x_i), p_i). \tag{12}$$

A pseudo label is an automatically generated label from the data attributes in the pretext task. In joint embedding architectures the pseudo labels are typically some *augmentation* of the original data.

SSL has in recent times (since 2018??) been a fruitful approach to unsupervised representation learning and has played an integral part in the boom of large language models together with the invention of the transformer architecture. With models growing ever larger and more data hungry SSL is essential in pretraining. Randomly initialized networks are difficult to train and requires a lot of time and computational resources. SSL has shown remarkable results when used for pretraining. That is as models for learning network parameters who capture semantics of data, without the need for labels. Pretraining networks enables foundation models, which can be trained for many different tasks in a supervised fashion requiring a lot less resources.

In recent years, especially in computer vision and natural language applications, SSL methods have shown incredible promise in representation learning. SSL methods have proven close to, and sometimes surpassing, supervised methods on downstream tasks. This has inspired researchers to apply SSL methods across other modalities.

TODO: Find sources backing this up.

TODO: What are the different flavours of SSL?

Generative: Masked modelling Discriminative: Contrastive/non-contrastive Autoassociative: Autoencoder type - Essentially try to make composition the identity despite a compression. Usually not considered SSL, but it technically is.

TODO: What are some of the issues?

Siamese Architecture-based SSL

[**lee2024computer**] A siamese network architecture [**siamese**] consists of two networks, called branches, with a shared encoder on each branch. Such networks are trained to produce similar representations for different views of the same data.

In models with such architecture, the existence of trivial solutions, such as both networks ignoring input and produce identical constant embeddings, is a major issue. This issue is referred to as *collapse* of the model.

Augmentations for creating different views. The role of augmentations, types etc. Augmentations across modalities. [**morningstar2024augmentations**]

In the computer vision domain several representation learning algorithms utilizing a siamese architecture have been proposed. They mainly fall under two categories: *contrastive* and *non-contrastive* methods.

In contrastive SSL, uses positive and negative samples and learns representations by pulling positive pairs closer together and negative pairs further apart. Examples include MoCo [**he2020momentum**] and SimCLR [**chen2020simple**]. SimCLR constructs two correlated samples by applying augmentations to an input sample, and considers these as positive pairs. Requires large batch sizes in order to work well.

Need large number of negative pairs compared to positive in order to learn representations effectively [**lee2024computer**].

The loss function of contrastive SSL attempts to minimize the distance of positive sample pairs (+,+) and (-,-), and maximize the distance between negative sample pairs (+,-) and (-,+).

Non-Contrastive: Collapse, where encodes produce uninformative or constant vectors. The different ways of handling collapse. How Variance-Covariance regularization (VCReg) [**mialon2024variance**] has emerged as a minimal(??) solution.

TODO: Talk about projectors in ncSSL. How and why they enforce pairwise independence

BYOL, Barlow Twins, VCReg, VICReg, VIbCReg

[**mialon2024variance**] VCReg for pairwise independence in SSL representations.

0.8.4 Masked modelling

Masked modelling is a conceptually simple self-supervised learning technique for generative models. The idea is to mask or cover a portion of the data and predict the masked portions. By comparing the prediction against unmasked data the model can learn useful representations without supervision. Since the introduction of masked modelling in natural language processing by [**devlin2019bert**] it has been the de-facto standard for self-supervised pretraining of language models. BERT introduced masked modeling for language representation learning as means to use bi-directional transformers without enabling each word to "see itself".

TODO: When did GPT [**Radford2018ImprovingLU**] use Masked modelling?

Masked image modelling [**he2021masked**]: Mask image patched directly

When vision transformers [**dosovitskiy2021image**] surpassed CNN in 2021[**he2015deep**], CV research began to draw inspiration more heavily from [**devlin2019bert**] and began tokenizing images and pretraining transformers.

Masked visual token modelling [**chang2022maskgit**]: Tokenize the image and mask tokens.

TODO: Masking in original domain or tokens. Masking strategies.

For a thorough survey on masked modelling for SSL in the vision domain we refer to [**li2024masked**].

0.9 Vector Quantized Variational Autoencoder (VQVAE)

Our model is based on the Vector Quantized Variational Autoencoder (VQVAE) introduced in [**VQVAE**], and includes an Auto-encoder (AE) branch. Therefore it is natural to dive into the models. We first start with introducing auto-encoders, then present the variational variation VAE before presenting VQVAE.

0.9.1 Autoencoder (AE)

Consists of two neural networks, an encoder E and decoder D. They can be seen as maps between spaces X and Z, where we refer to X as the data space and Z as the latent space.

$$X \xrightarrow{E} Z \xrightarrow{D} X \tag{13}$$

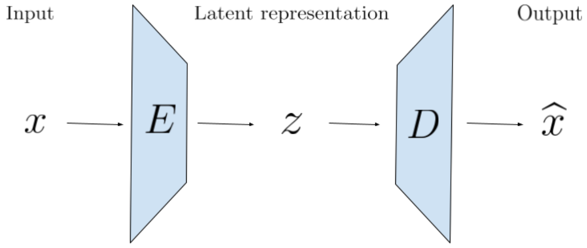

Figure 13: Schematics of the autoencoder architecture.

The autoencoder is trained in such a way that the composition of encoder and decoder is approximate to the identity. Typically the dimension of the latent space is much lower that that of the data space. Encoder and decoder then compresses and decompresses the data and learns efficient *latent representations*.

TODO: Information bottleneck

TODO: Issues with latent representations and the need for regularization

0.9.2 Variational Autoencoder (VAE)

Variational Autoencoders (VAE) were introduced in [**kingma2022autoencoding**] and is a variational Bayes approach to approximate inference and learning with directed probabilistic models. The architecture of VAE is similar to AE, but the mathematical formulation is quite different. For context variational inference is a technique in statistics used to approximate complex distributions by looking for the closest approximation within a simple, but flexible, parametric family.

In the VAE framework we assume that the dataset $X = \{x_i\}_{i=1}^N$ consists of iid samples from a random variable $\mathbf{x}$. We further assume that the data is generated by some unobservable random process. This is to say that we assume there is a random variable $\mathbf{z}$ such that $x_i \sim p_{\theta^*}(x|z_i)$, where $z_i \sim p_{\theta^*}(z)$. The distribution $p_{\theta^*}(z)$ is referred to at the true prior and $p_{\theta^*}(x|z_i)$ as the true likelihood. As $\mathbf{z}$ is unobservable and the true distributions are unknown, one has to assume their form. In general the prior and likelihood are assumed to be from parametric families $p_\theta(z)$ and $p_\theta(x|z)$. As with any model where one wishes to employ gradient based learning, the distributions are assumed to be differentiable almost everywhere, both with respect to their parameters and argument.

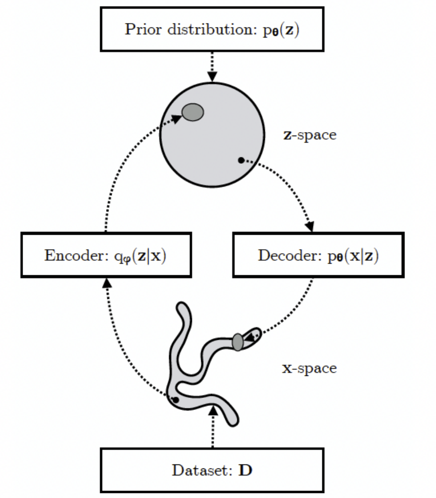
Prior distribution: $p_\theta(\mathbf{z})$
z-space
Encoder: $q_\varphi(\mathbf{z}|\mathbf{x})$
Decoder: $p_\theta(\mathbf{x}|\mathbf{z})$
x-space
Dataset: **D**

Figure 14: [**VAE**], need to ask for permission or make own

VAEs have two components to their architecture. The first is a probabilistic encoder, often called the inference model, $q_\phi(z|x)$ which approximates the true posterior. Secondly a probabilistic decoder, often called the generative model $p_\theta(x|z)$, which approximates the likelihood. These models are typically parameterized by some type of neural network, and in that case ϕ and θ are the weights and biases of the two networks. Given a datapoint x_i the probabilistic encoder provides a distribution over the possible values of the latent variable **z**. Similarly, given a latent representation z_i the probabilistic decoder produces a distribution over the possible corresponding values of **x**.

The probabilistic part of the encoder and decoder is the sampling. The output for a given x or z is a distribution, and for the same input the same output distribution is calculated (as long as the network parameters are frozen). Actually x and z are mapped to the parameters of a distribution, which uniquely determines the distribution in a particular family.

The most common situation is to assume

- Prior $p_\theta(z) \sim N(0, I)$
- Likelihood $p_\theta(x|z) \sim N(D(z), I)$
- Variational posterior $q_\phi(z|x) \sim N(E(x)) = N(\mu_x, \Sigma_x)$

The encoder maps datapoints to parameters of the variational distribution, i.e the approximate posterior q_ϕ, $x \mapsto E(x) = (\mu_x, \Sigma_x)$. A latent representation z is then sampled from q_ϕ, which constitutes the random part of the algorithm. The decoder maps z to the expected value of the likelihood $p(x|z)$, $z_i \mapsto D(z_i)$.

There are several reasons for choosing Gaussian distributions, one being that the Gaussian distribution a scale-location family. This enables us to circumvent the problematic random component in the VAE when using gradient based learning. As described above we sample $z \sim N(\mu_x, \Sigma_x)$ where $E(x) = (\mu_x, \Sigma_x)$. We can equivalently sample from $N(\mu_x, \Sigma_x)$ by reparameterization using the location-scale property. This means to introduce an auxillary variable $\epsilon \sim N(0, I)$ and rewriting $z = \mu_x + L_x\epsilon$, where L_x is the Cholesky decomposition of Σ_x. This factors the random part out of path of gradient flow.

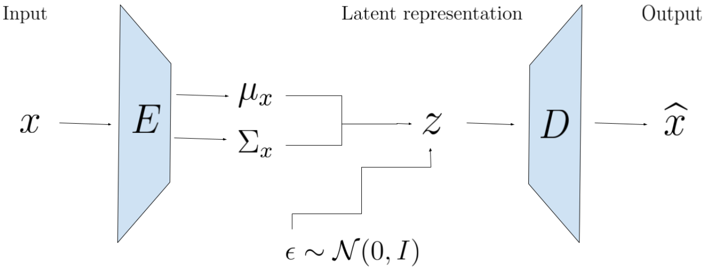

Figure 15: Schematics of the VAE architecture with Gaussian reparameterization.

Training objective

As with other variational methods VAEs are optimized with respect to the *evidence lower bound* or ELBO for short. Let **x** and **z** be two jointly distributed variables, with distribution p_θ. Then for any distribution q_ϕ the ELBO is defined as

$$\mathcal{L}_{\theta,\phi}(x) = \mathbb{E}_{q_\phi(z|x)} \log\left(\frac{p_\theta(x,z)}{q_\phi(z|x)}\right) \tag{14}$$

The ELBO can be reformulated in terms of the marginal likelihood and KL divergence of the variational to the true posterior.

$$\begin{aligned}\mathcal{L}_{\theta,\phi}(x) &= \mathbb{E}_{q_\phi(z|x)} \log(p_\theta(x)) + \mathbb{E}_{q_\phi(z|x)} \log\left(\frac{p_\theta(z|x)}{q_\phi(z|x)}\right) \\ &= \log(p_\theta(x)) - \mathrm{KL}(q_\phi(z|x)||p_\theta(z|x)).\end{aligned} \tag{15}$$

Due to the non-negativity of the KL-divergence, we see that the ELBO bounds the marginal log likelihood of the data from below. By maximizing the ELBO with respect to the model parameters ϕ and θ one simultaneously maximizes the marginal likelihood, which improves the generative model, as well as reducing the KL-divergence of the approximate to the true posterior, which improves the inference model [**VAE**].

An alternative reformulation gives a more evident connection to Autoencoders, with a the prior acting as a regularizer on the posterior

$$\begin{aligned}\mathcal{L}_{\theta,\phi}(x) &= \mathbb{E}_{q_\phi(z|x)}\log\left(p_\theta(x|z)\right) - \mathbb{E}_{q_\phi(z|x)}\log\left(\frac{q_\theta(z|x)}{p_\phi(z)}\right)\\ &= \underbrace{\mathbb{E}_{q_\phi(z|x)}\log\left(p_\theta(x|z)\right)}_{\text{Expected reconstruction log likelihood}} \quad \underbrace{-\mathrm{KL}(q_\phi(z|x)||p_\theta(z))}_{\text{Regularizer}}.\end{aligned} \tag{16}$$

As the $p_\theta(x|z)$ is typically assumed to be Gaussian we have

$$\log p_\theta(x|z) = -\frac{1}{2}\left[k\log(2\pi) + (x - D(z))^T(x - D(z))\right], \tag{17}$$

which is equivalent, as an optimization objective of $D(z)$, to $||x - D(z)||_2^2$. Consequently the likelihood in the loss is implemented as the MSE of the input x and the output $D(z)$.

Generative model

The role of the prior in a VAE is two fold. Firstly as a regularizing constraint for the posterior during training, and secondly as a way of obtaining new samples x via ancestral sampling. Ancestral sampling refers to $\mathbf{z}$ being the parent node of $\mathbf{x}$ in the VAE, and that we can generate samples x by drawing a sample $z \sim p(z)$ and decode $D(z)$.

TODO: Speak on limitations of VAE before introducing VQVAE

Variational autoencoders has the issue of collapsing. Variance issues.

0.9.3 Vector Quantization (VQ)

Dictionary learning model [**Gray1984VQ**]

0.9.4 VQVAE

The Vector Quantized Variational AutoEncoder (VQVAE) was first introduced in [**VQVAE**] and presented a new way of training VAEs with discrete latent variables. It is the first discrete VAE model which has similar performance to the continuous variant. VQVAE was developed to learn useful discrete representations for generation. This is enforced by more flexible distributional assumptions, as they say "Our model, the Vector Quantised- Variational AutoEncoder (VQ-VAE), differs from VAEs in two key ways: the encoder network outputs discrete, rather than continuous, codes; and the prior is learnt rather than static."-[**VQVAE**].
The overall architecture of VQVAE consists of an encoder and decoder as before, together with a *codebook* which is used in the quantization process. The codebook, also referred to as the latent embedding space, $\mathcal{Z} = \{z_k\}_{i=1}^{K}$ is consists of K latent vectors with dimensionality D. It can be considered a lookup table of vectors that are used for vector quantization (VQ). The output of the encoder $E(x)$

is quantized by the means of nearest neighbor lookup

$$z_q = \arg\min_{z_k \in \mathcal{Z}} ||E(x) - z_k||. \tag{18}$$

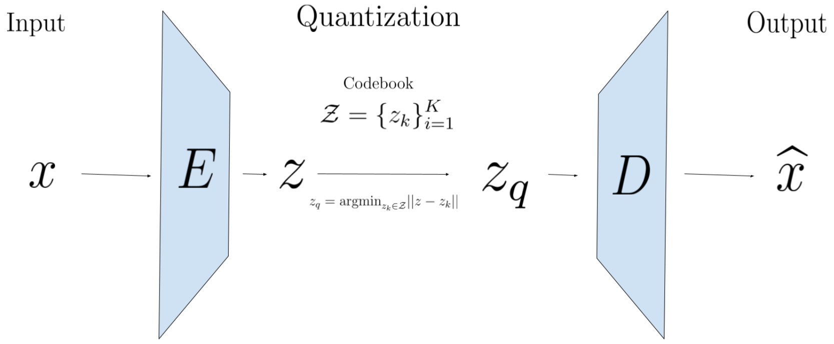
Input
Quantization
Output
Codebook
$\mathcal{Z} = \{z_k\}_{i=1}^{K}$
x
E
z
$z_q = \text{argmin}_{z_k \in \mathcal{Z}} ||z - z_k||$
z_q
D
$\widehat{x}$

Figure 16: Schematics of the VQVAE architecture.

From **??** we can see that the VQVAE can be seen as an autoencoder with a nonlinearity introduced by the vector quantization.

In the original article [**VQVAE**] they propose to separately learn the prior distribution, after the encoder, decoder and codebook is learned. During the initial training the prior is assumed to be uniform, which poses no constraint on the posterior learning.

In contrast to VAEs the posterior is assumed to be categorical, as opposed to normal and the probabilities are defined as

$$p(z = k|x) = \begin{cases} 1 & \text{for } k = \arg\min_j ||E(x) - z_j||_2 \\ 0 & \text{otherwise} \end{cases}, \tag{19}$$

Sampling from the posterior amounts to quantizing the output of the encoder, as it is deterministic.

VQVAE can be viewed as a VAE, hence we can bound the marginal likelihood with the ELBO **??**. As the variational posterior is deterministic and the prior (during training) is uniform over $\{1, ..., K\}$ we get that the regularizing term

$$\begin{aligned} \text{KL}(q(z|x)||p(z)) &= \sum_z q(z|x) \log\left(\frac{q(z|x)}{p(z)}\right) \\ &= q(z = k|x) \log\left(\frac{q(z = k|x)}{p(z = k)}\right), \ q \text{ is deterministic} \\ &= \log\left(\frac{1}{1/K}\right), \text{ uniform prior} \\ &= \log(K), \end{aligned} \tag{20}$$

is constant. Consequently the ELBO reduces to the reconstruction term only.

TODO: Can think of the codes as the modes of a mixture of Gaussians, and instead of sampling from the mixtrure you just choose the closest mode.

Prior Learning

During the initial training the prior is as mentioned constant and uniform such that the posterior has more flexibility. After training an autoregressive prior is fit on z so we can sample x via ancestral sampling. The prior is learned with a generative objective, which increases the quality of generated samples. The choice of prior model depends on the particular application and in VQVAE PixelCNN [**oord2016pixel**] was used for image, while WaveNet [**oord2016wavenet**] was used for audio. More recently in TimeVQVAE [**TimeVQVAE**] utilized a bidirectional transformer [**chang2022maskgit**] for prior learning in the time series domain.

Loss funciton

As we want to do gradient based learning, and the quantization process is quite far off from being continuous, measures has to be taken. The output of the encoder and the input of the decoder has the same dimension, which opens the possibility of simply copying the gradients from the decoder input to the decoder output. This is referred to as the *straight through estimator*. The gradients from the decoder contains relevant information to how the encoder should change its output, and in the subsequent forward pass the encoder output can be quantized to different discrete latent codes.

The loss function consists of three parts, the reconstruction loss, VQ loss and commitment loss. We already mentioned that the ELBO objective reduced to the reconstruction term only. In order to learn the codebook one uses the VQ loss, which is the euclidean distance between the encoder output and the quantized output. The final part of the loss is the commitment loss, which is introduced to keep the distance between embedding vectors (or codes) from growing arbitrarily large. To summarize the total loss is given by

$$\begin{aligned} \mathcal{L}_{\text{Recon}} &= ||x - \hat{x}||_2^2 \\ \mathcal{L}_{\text{VQ}} &= ||sg(z) - z_q||_2^2 + \beta||z - sg(z_q)||_2^2 \\ \mathcal{L}_{\text{VQ-VAE}} &= \mathcal{L}_{\text{VQ}} + \mathcal{L}_{\text{Recon}} \end{aligned} \tag{21}$$

where β is a tuning parameter typically set to be 0.25 and $sg()$ is the stop-gradient operation, defined to be the identity with zero partial derivatives.

To illustrate the need for commitment loss and what it does consider a model where the data is either 0 or 1 and we initialize the codebook with $\mathcal{Z} = \{-1, 1\}$.

The range of the encoder is $\mathbb{R}$ and $E(x)$ is quantized to -1 if its negative and 1 if its positive. Assume that the encoder and decoder will try to differentiate between the two classes by pushing $E(0)$ and $E(1)$ away from each other. As the reconstruction loss only affects the encoder and decoder, and the VQ loss only affect the codebook, we get that when the encoder and decoder parameter trains faster than the codebook, then the distance between the encodings and the codewords will steadily increase and the VQ loss diverge. This behavior is observed experimentally.

0.10 Evaluation metrics

0.10.1 Tokenization model

The tokenization model, as we are interested in representation learning, is evaluated on two metrics. Firstly, and most importantly its ability to reconstruct the input data once compressed into latent space. In essence the latent representation encodes "everything" (important information is preserved) about the original data if the model is able to reconstruct well. Secondly we evaluate linear classifiers on the latent representations, which provides good results if the model learns discriminative features of the different classes and produces an approximately linear separable space. Finally, as the tokenization model is a part of the generative model, the ultimate evaluation metric is the corresponding evaluation of the generative model.

Reconstruction

Downstream Classification

SVM, KNN. The difference in inductive bias for the two classifiers.

0.10.2 Generative model

Good evaluation protocols for generative models are hard to come by, and the hunt for such is an active area of research. There are data modalities for which such evaluation metrics have more established standards than others. Most notably in the computer vision domain there is a benchmark dataset (ImageNet) and classifier (Inception v3).

The typical sanity check for generative models is human visual inspection. In development of evaluation metrics for generative models, how well a proposed metric correlates with quality determined by visual inspection is standard. Most people know what cats and dogs look like and can recognize features even in crudely generated images. In contrast, many time series dataset measure quantities and signals for which the untrained eye has no idea of determining the quality.

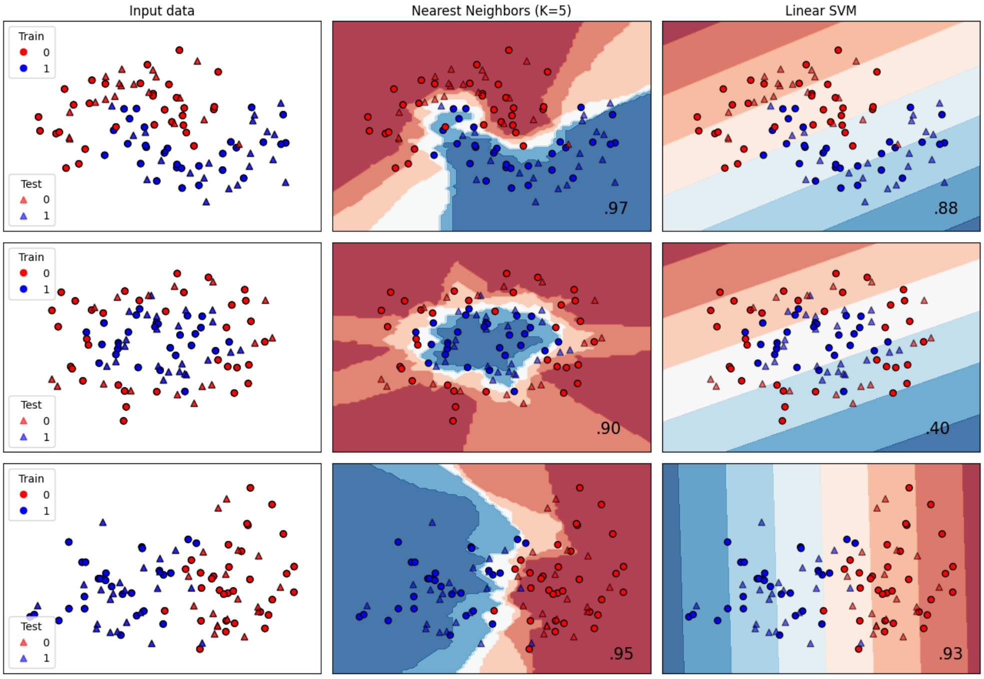

Figure 17: Modified example taken from `https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html`.

This is one of the obstacles in TSG. Despite this, according to [**TimeVQVAE**] the most common evaluation protocols in the TSG literature is PCA and t-SNE analyses on time series to visually see similarities of two distributions. The major limitations of this is that the visual inspections cannot be reduced to a single score, which makes objective comparison difficult.

Following [**TimeVQVAE**] we mainly consider three evaluation metrics in this thesis. *Inception Score* (IS), *Fréchet Inception Distance* (FID) and *Classification Accuracy Score* (CAS). IS measures quality of synthetic samples using the entropy of the label distribution and the evenness in prediction across labels. The *Fréchet Inception Distance* measures a distance between the generative and ground truth distribution. Finally the *Classification Accuracy Score* measures the quality of the class-conditionally generated samples by training a classifier on synthetic data and testing on real.

Common for all the above evaluation metrics is the reliance on a pre-trained classification model. In contrast to computer vision there is no widely adopted classification model for time series, but in [**wang2016time**] a fully convolutional network (FCN) was presented as a proposed baseline classification model for time series. As their model is not available we use the open source FCN trained on the UCR Archive presented in [**TimeVQVAE**] for our evaluations.

Inception Score (IS)

Inception Score (IS) was first introduced in [**salimans2016improved**] as an automatic evaluation method of synthetic samples.
The Inception Score works by applying a classifier to every generated sample to get the conditional label distribution $p(y|x)$. Samples containing clear characteristics should have low entropy, i.e one should be certain of the label. In addition we expect a good model to generate varied samples, so the marginal $\int p(y|x = D(z))dz$ should have high entropy. On this basis, the Inception Score is defined as

$$\mathrm{IS}(\theta) = \exp\left(\mathbb{E}(D_{\mathrm{KL}}(p_\theta(y|\mathbf{x})||p_\theta(y)))\right). \tag{22}$$

In contrast to computer vision there is no widely adopted classification model for Time Series. In [**wang2016time**] a fully convolutional network (FCN) was presented as a proposed baseline classification model, and we use the open source FCN trained on the UCR Archive presented in [**TimeVQVAE**] for our evaluations.

Let $X_{\text{gen}} = \{x_{i,\text{gen}}\}_{i=1}^N$ be a set of generated samples. We use the apply a SoftMax layer to pretrained FCN representations to obtain an estimate of the conditional label distribution as follows

$$x_{i,\text{gen}} \xrightarrow{\text{FCN+SoftMax}} p(y|x_{i,\text{gen}}).$$

The marginal label distribution is obtained by averaging across all the synthetic data as follows

$$p(y) = \frac{1}{N}\sum_{i=1}^{N} p(y|x_{i,\text{gen}}).$$

Issues with IS [**barratt2018note**] (Use different network, for image classification). Important to report different metrics that indicate that the model has not overfitted. Also issues with IS [**borji2021pros**].

Fréchet Inception Distance (FID)

As an attempt to improve on IS [**heusel2018gans**] introduced the Fréchet Inception Distance (FID). Since then FID has been the standard for assessing generative models [**borji2021pros**].
The primary concern with IS is, as mentioned earlier, that it does not use any statistics of real world samples to compare with the statistics of the generated samples. In contrast FID relates the synthetic sample to the real world samle via the *Fréchet distance*. For any two probability distributions, f, g over $\mathbb{R}^n$, with finite

mean and variances, their distance is defined as

$$\begin{aligned} d_F(f,g) &= \left(\inf_{\gamma \in \Gamma(f,g)} \int_{\mathbb{R}^n \times \mathbb{R}^n} ||x-y||_2^2 d\gamma(x,y) \right)^{\frac{1}{2}} \\ &= \left(\inf_{\gamma \in \Gamma(f,g)} \mathbb{E}_{(x,y)\sim\gamma} ||x-y||_2^2 \right)^{\frac{1}{2}}, \end{aligned} \tag{23}$$

where $\Gamma(f,g)$ is the set of all *couplings* of f and g. The Fréchet distance is a special case of the Wasserstein metric and will in literature be referred to as such.

In [**DOWSON1982450**] it was shown that for two Gaussian distributions the Fréchet distance is explicitly solvable as

$$d(\mathcal{N}(\mu,\Sigma),\mathcal{N}(\mu',\Sigma'))^2 = ||\mu-\mu'||_2^2 + \mathrm{Tr}\left(\Sigma+\Sigma'-2(\Sigma\Sigma')^{\frac{1}{2}}\right) \tag{24}$$

The *Fréchet Inception Distance* is an application of the Fréchet distance on the representation distributions from a specified model. The models are Inception v3 in the image domain and SupervisedFCN in the time series domain.

In [**heusel2018gans**] they argue that since the Gaussian distribution is the maximum entropy distribution over $\mathbb{R}^n$ for a given mean and covariance it is a reasonable distribution to assume for the representations. The mean and covariance is estimated from the samples and the explicit formula **??** is used to calculate the FID.

FID is not without fault, and the Gaussian assumption has been shown not to hold [**jayasumana2024rethinking**]. Further, as the FID relies on estimating large covariance matrices, a large number of samples is needed to obtain a reliable estimate.

Rethinking FID: [**jayasumana2024rethinking**]

[**chong2020effectively**] FID and IS are biased.

FID score measures difference in the distribution of representations of time series. The representations capture high level semantics of the time series and analyses of these representations can provide greater insights to the realism of generated samples.

Classification Accuracy Score (CAS)

[**smith2020conditional**] report the CAS. A method for evaluating the models ability to learn class conditional distributions is to train a separate classifier on the on synthetic data and test on real data (TSTR). That is we generate N synthetic samples $\{x_i, y_i\}_{i=1}^N$ using class conditional sampling. We then train a classifier on this synthetic dataset, and evaluate the CAS of this classifier on a real test dataset. High CAS values indicate that the generative model produces samples similar to the real data and captures relevant class specific features.

We evaluate the CAS for TSTR on all our models considered and compare against the baseline model to investigate the relative performance.