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

Johan Vik Mathisen

May 7, 2024

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

0.1 Information theoretic/basic stats used in evaluation

- mutual information

- entropy

- perplexity

- KL-divergence `https://stats.stackexchange.com/questions/188903/intuition-on-the-kullbac`
`189758#189758`

- Graphical probabilistic models - Ancestral sampling

0.2 Time Series Inference

- short-time-fourier transform etc.

0.3 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 backbone in the development of machine learning, 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 [fig of nn here] 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

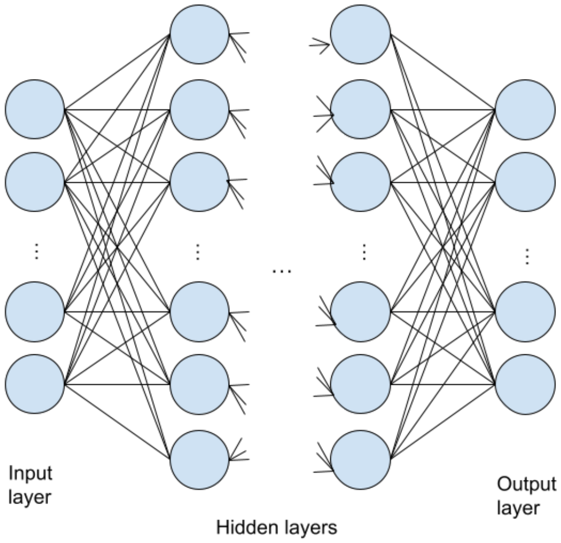


Figure 1: Illustration of a Neural Network model.

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. Written out a K layered neural network is given by

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

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.

0.3.1 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 common examples of loss functions include mean squared error (MSE) and mean absolute error (MAE). Gradient based methods, such as backpropagation, are usually used to estimate the parameters based on the loss.

The backpropagation algorithm 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**].

0.4 Convolutional Neural Network

This section draws heavily 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**].

TODO: history

1959 visual cortex cells. The Neocognitron 1980. CNN for MNIST Yann LeCun 1989- 1998 [**LeCun1989ConvNet**]. AlexNet: GPU training and popularization of deep neural nets.

TODO: Why was it introduced?

Regular neural networks have a fundamental drawback in that their computational complexity

Fully connected neural networks suffer badly from overfitting. Vanishing gradient / exploding gradient.

What problems did it solve?

0.4.1 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

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{1}$$

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 sometimes 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

$$(I * K)[n] = \sum_{m=-\infty}^{\infty} I[m]K[n-m]. \tag{2}$$

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

$$(I * K)[i,j] = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} I[n,m]K[i-n,j-m]. \tag{3}$$

TODO: Figure of convolution

- In applications the kernel dimensions are much smaller than the input dimensions.

- Stride

- Kernel dim and stride length determines downsample rate (reduction in size).

- In a convolutional layer the kernel is learned such that the feature map is helpful for the training objective.

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**]. Machine learning applications are focused on what works, rather than writing proofs.

"Convolution leverages three important ideas that can help improve ML systems: sparse interactions, parameter sharing and equivariant representations" [**deeplearningbook**].

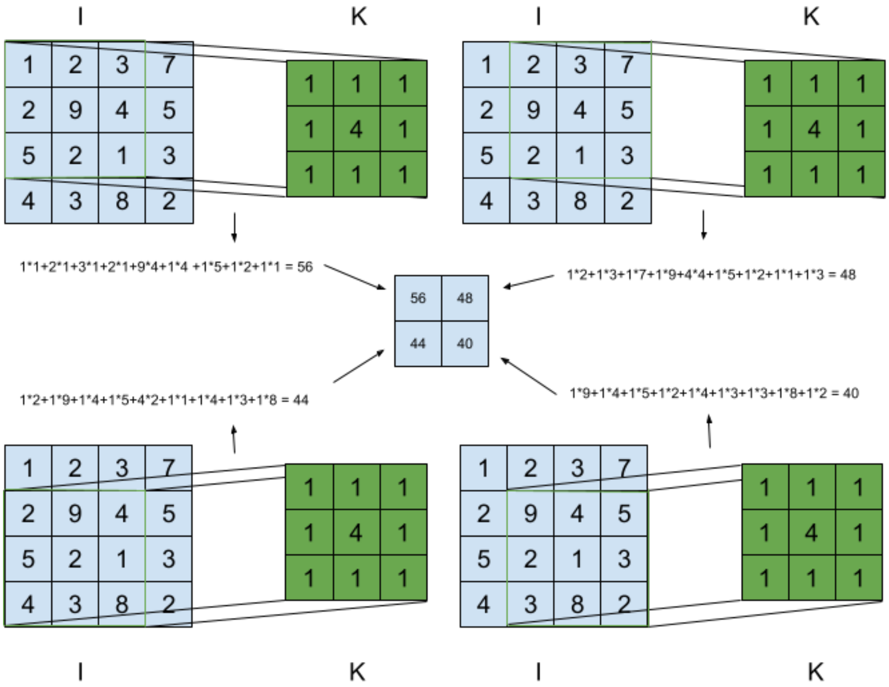


Figure 2: Illustration of discrete two dimensional convolution

0.4.2 Pooling

A pooling operation is applied as a down sample technique on feature maps in CNNs, 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.

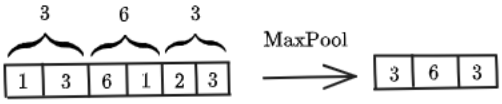


Figure 3: Max pooling of one dimensional array. Filter size: 2, stride: 2.

- Pooling assist in making the representations approximately invariant to small distortions of input.

- Pooling is often essential in handling variable input size.

-

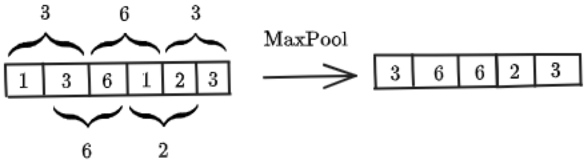


Figure 4: Max pooling of one dimensional array. Filter size: 2, stride: 1.

0.4.3 Architecture

A typical layer in a convolutional network consists of three stages:

Convolution: Hidden layer with regular matrix multiplication with weight matrix W is switched out with convolution with kernel matrix K.

non-linear activation: Point wise nonlinearity, ReLU, sigmoid etc.

pooling layer: Max, average etc. pooling of feature map.

0.4.4 Transposed Convolutional Networks

Transposed convolution, also known as fractionally-strided convolution is a technique used to reverse the downsampling from convolutions.

0.5 Representation Learning

TODO: Speak on the evaluation metrics for rep learning and the like from [**nozawa2022empirical**]

[**Rep-rev-persp**]

TODO: Sneak in something about Gödel and Turing in terms of representation. There are many philosophical aspects of this

0.5.1 What is representation learning?

Representation learning is a term not too easily defined, one reason being the abstraction level. It is helpfull 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.

Find som references on higher order features learned in CNNs

better name??

Anyone who has worked with data science or machine learning has come across feature engineering, and 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. One of the intriguing and promising features of neural networks, with its many specializations and architectures, is the ability to learn abstract representations of the data. This is sometimes referred to as automatic feature engineering. In a N-layered network $f = f_N \circ ... \circ f_1$, the intermediate value of the data x in some layer n, f_n, is what is meant by the networks learned feature representations. When we are interested in the representations learned it 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 some task, be it supervised, unsupervised etc.
- Discard g
- Use the learned feature extractor $\widehat{h}$ as part of a new model.

The different use cases of $\widehat{h}$ is as stand alone one/few-shot learners (downstream task / frozen protocol) or as initialization of other models (pre-trained / fine-tuning protocol).

When training a neural network of any variation in a supervised fashion, one always get a feature extractor, at no additional cost. Unsupervised representation learning loss: reconstruction in AE type, similarity losses in joint embedding networks

TODO: When mentioning autoencoders, it is a natural spot to talk about compression. "If you can still reconstruct the signal, then you know everything about it in a sense"

Typically in representation learning algorithms, the output dimension of $\widehat{h}$ is smaller than the input dimension. The idea of compression in representation learning has gotten theoretical and philosophical attention .

Information bottleneck, compression

0.5.2 Why do we care about representation learning?

TODO: Rewrite, i have copied to much

Representation learning is particularly interesting because it provides one way to perform unsupervised and semi-supervised learning. It promises to unlock deep learning for unlabeled datasets. Furthermore it is known that the performance of machine learning methods is heavily dependent on the choice of data representations. Therefore much of actual efforts in deploying machine learning algorithms revolves around constructing good data pipelines and data transformations that results in representations suited for the ML algorithm. Being able to automate such processes, i.e automatic feature engineering, would solve massive problems and ease the use of ML considerably.

0.5.3 What is a good representation?

TODO: Talk about the trend of pre-trained models, language representations, tokenization, GPT etc

TODO: Rewrite, i have copied to much

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

0.5.4 How does one evaluate representations?

As defined in [**jing2019selfsupervised**] a *pretext task* is a pre-designed task for a network to solve, where the goal is to learn representation. A downstream task is a task used to evaluate the quality of learned representations. In general the downstream task is solved in a supervised manner, using human annotated data. We let $f = g \circ h$ be a model and train it on a pretext task in order to obtain $\widehat{f} = \widehat{g} \circ \widehat{h}$. As mentioned previously, the quality of learned representations is determined by the performance of the feature extractor $\widehat{h}$ 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. 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.

TODO: The role of Visual inspection

0.6 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.

0.6.1 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

$$\widehat{f_\theta} = \min_\theta \frac{1}{N}\sum_{i=1}^{N} L(f_\theta(x_i), y_i). \tag{4}$$

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.

0.6.2 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,

0.6.3 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

$$\widehat{f_\theta} = \min_\theta \frac{1}{N} \sum_{i=1}^{N} L(f_\theta(x_i), p_i). \tag{5}$$

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.

TODO: Explain why we care:

A fruitful approach to unsupervised representation learning.

Randomly initialized networks are difficult to train and requires a lot of time and computational resources. SSL has shown remarkable results when used for pre-training. That is as models for learning network parameters who capture semantics of data, without the need for labels. Pre-training networks enables foundation models, which can be trained for many different tasks in a supervised fashion, called fine-tuning, requiring a lot less resources.

TODO: Explain where we are now:

In recent years, especially in computer vision applications, SSL methods have shown incredible promise in representation learning. SSL methods have proven close to, and sometimes surpassing, supervised methods on downstream tasks.

TODO: Find sources backing this up.

TODO: What are the different flavours of SSL?

Autoassociative SSL - Autoencoder type - Essentially try to make composition the identity despite a compression. Usually not considered SSL, but it technically is. Contrastive vs non-contrastive

TODO: What are some of the issues?

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.

Contrastive

Contrastive SSL includes both positive and negative samples. The loss function of contrasitive SSL attempts to minimize the distance of positive sample pairs (+,+) and (-,-), and maximize the distance between negative sample pairs (+,-) and (-,+).

Non contrastive

Augmentations for creating different views. The role of augmentations, types etc. Augmentations across modalities.

0.6.4 Joint embedding architecture

Joint embedding architecture: An architecture where two networks are trained to produce similar embeddings for different views of the same data. A popular joint embedding architecture is the siamese network architecture [**siamese**], in which the two networks share the same weights. 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.

The role of projectors

0.7 Transformers

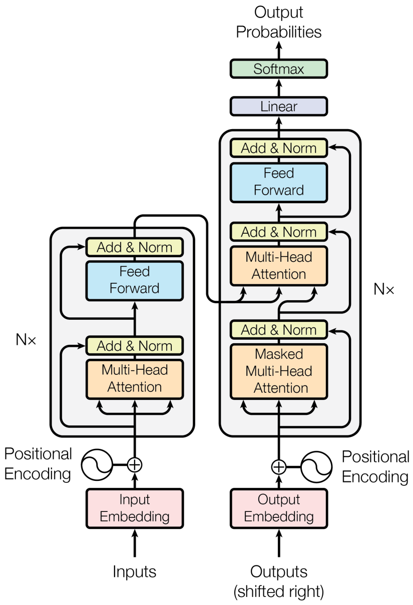
When was it introduced, and by whom? What lead up to its creation? Why and where is it now widely used?

Wiki: A deep learning architecture based on the multi-headed attention mechanism proposed in the 2017 paper "Attention is all you need".

One fo the main novelties of the architecture is not relying on recurrence, and instead solely using the attention mechanism to capture dependencies between input and output. As recurrent models are, in computational aspects, inherently sequential, there are major challenges when training as context windows, and sequence lengths become longer.

0.7.1 The attention mechanism

TODO: Wait for 3b1b video for context



0.7.2 Architecture

Tokenizer

Positional encoding

Transformer layer (encoder/decoder)

0.7.3 Bi-directional transformer (BERT)

Unidirectional /

0.8 Vector Quantized Variantional Autoencoder (VQVAE)

Our model is based on the Vector Quantized Variantional 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.

0.8.1 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.

TODO: Architecture figure

$$X \xrightarrow{E} Z \xrightarrow{D} X \tag{6}$$

The autoencoder is trained in such a way that the composition of encoder and decoder is approximate to the identity. Typically, if the model is to learn interesting, the dimension of the latent space is 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

0.8.2 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, parameter family.

TODO: Intractabilities, why an how?

We assume that the dataset $X = \{x_i\}_{i=1}^N$ consists of iid samples from a random variable $\mathbf{x}$. In the VAE framework 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.

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 $\mathbf{z}$. Similarly, given a latent representation z_i the probabilistic decoder produces a distribution over the possible corresponding values of $\mathbf{x}$.

The probabilistic part of the encoder and decoder is the sampling part. 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 determ-

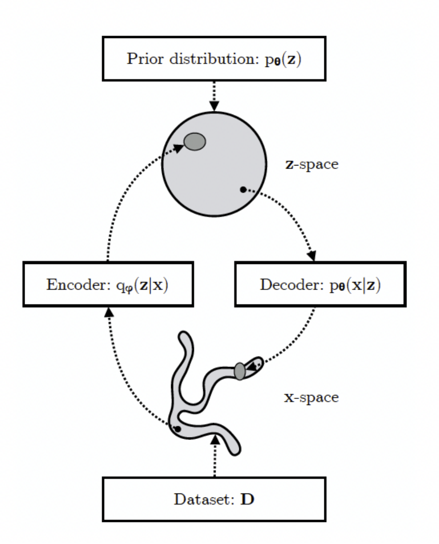

Prior distribution: $p_\theta(\mathbf{z})$
z-space
Encoder: $q_\varphi(\mathbf{z}|\mathbf{x})$
Decoder: $p_\theta(\mathbf{x}|\mathbf{z})$
x-space
Dataset: **D**

Figure 5: [**VAE**], need to ask for permission or make own

ines the distribution in a particular family.

The most common situation is to assume

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

In a VAE the encoder maps datapoints to parameters of the variational distribution, i.e the approximate posterior q. $x \mapsto E(x) = (\mu_x, \Sigma_x)$. A latent representation z is then sampled from q, 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 this family, 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.

TODO: Illustration of computational flow/gradient flow.

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

$$\mathcal{L}_{\theta,\phi}(x) = \mathbb{E}_{q_\phi(z|x)} \log\left(\frac{p_\theta(x,z)}{q_\phi(z|x)}\right) \tag{7}$$

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

$$\begin{aligned}\mathcal{L}_{\theta,\phi}(x) &= \mathbb{E}_{q_\phi(z|x)} \log(p_\theta(x)) + \mathbb{E}_{q_\phi(z|x)} \log\left(\frac{p_\theta(z|x)}{q_\phi(z|x)}\right) \\ &= \log(p_\theta(x)) - \mathrm{KL}(q_\phi(z|x)||p_\theta(z|x)).\end{aligned} \tag{8}$$

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 Auto-encoders, with a the prior acting as a regularizer.

$$\begin{aligned}\mathcal{L}_{\theta,\phi}(x) &= \mathbb{E}_{q_\phi(z|x)} \log(p_\theta(x|z)) - \mathbb{E}_{q_\phi(z|x)} \log\left(\frac{q_\theta(z|x)}{p_\phi(z)}\right) \\ &= \underbrace{\mathbb{E}_{q_\phi(z|x)} \log(p_\theta(x|z))}_{\text{Expected reconstruction log likelihood}} \underbrace{-\mathrm{KL}(q_\phi(z|x)||p_\theta(z))}_{\text{Regularizer}}.\end{aligned} \tag{9}$$

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

$$\log p_\theta(x|z) = -\frac{1}{2}\left[k\log(2\pi) + (x - D(z))^T(x - D(z))\right], \tag{10}$$

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

As we have explicit distributions in a trained VAE, we can use this to generate synthetic samples x.

We can either use *prior predictive sampling* by sampling from the prior distribution and decode the latent representation using the decoder to obtain a sample x. The other option is to use *posterior predictive sampling* by using a datapoint x_i to obtain a posterior distribution $q(z|x_i)$, sample a latent representation z_i and then obtain $p(x|z_i)$. From this likelihood we can sample implicitly conditional on x_i.

TODO: Speak on limitations of VAE before introducing VQVAE

Variational autoencoders has the issue of collapsing. Variance issues.

0.8.3 Vector Quantization (VQ)

Dictionary learning model [**Gray1984VQ**]

0.8.4 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 latent VAE model which has similar performance to the continuous variant. VQVAE was developed to learn useful discrete representations. Difference between VAE and VQ-VAE: 1. Maps input to discrete latent space instead of continuous. 2. The prior, $p(z)$, is learned rather than static.

In contrast to VAEs, the prior and posterior is assumed to be categorical, as opposed to normal.

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 posterior categorical distribution probabilities are defined as

$$p(z=k|x) = \begin{cases} 1 & \text{for } k = \arg\min_j ||z_e(x) - e_j||_2 \\ 0 & \text{otherwise} \end{cases}, \tag{11}$$

Sampling from the posterior amounts to quantizing the output of the encoder.

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

$$\begin{aligned} \mathrm{KL}(q(z|x)||p(z)) &= \sum_z q(z|x)\log\left(\frac{q(z|x)}{p(z)}\right) \\ &= q(z=k|x)\log\left(\frac{q(z=k|x)}{p(z=k)}\right), \ q \text{ is deterministic} \\ &= \log\left(\frac{1}{1/K}\right), \text{ uniform prior} \\ &= \log(K), \end{aligned} \tag{12}$$

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

A forward pass of a datapoint x to a discrete latent is given

Prior

During training the prior is constant and uniform. The uniform distribution on a set is the *maximum entropy distribution* among all discrete distribution supported on that set. This means in particular it is the least informative, making it a natural selection for an intermediate step. After training an autoregressive prior is fit on z so we can sample x via ancestral sampling.

Loss funciton

$$\mathcal{L}_{\text{VQ-VAE}} = \mathcal{L}_{\text{VQ}} + \mathcal{L}_{\text{Recon}} \tag{13}$$

VQ loss:

$$\mathcal{L}_{\text{VQ}} = ||sg(z) - z_q||_2^2 + \beta||z - sg(z_q)||_2^2 \tag{14}$$

Reconstruction loss:

$$\mathcal{L}_{\text{Recon}} = ||x - \widehat{x}||_2^2 \tag{15}$$

How is the codebook updated? Using VQ. Ether with l_2 error or moving average.

Terms with stop gradient are "frozen", so the l_2 codebook update loss, $||sg(z) - z_q||_2^2$ pushes the embedding vectors towards the encoder output.

TODO: Dont really onderstand why this is so. Cant seem to figure it out in the code.

In the code quantize is x (all data) + sg(quantize - x).detatch()

Codebook loss: Distance between z and z_q

Codebook

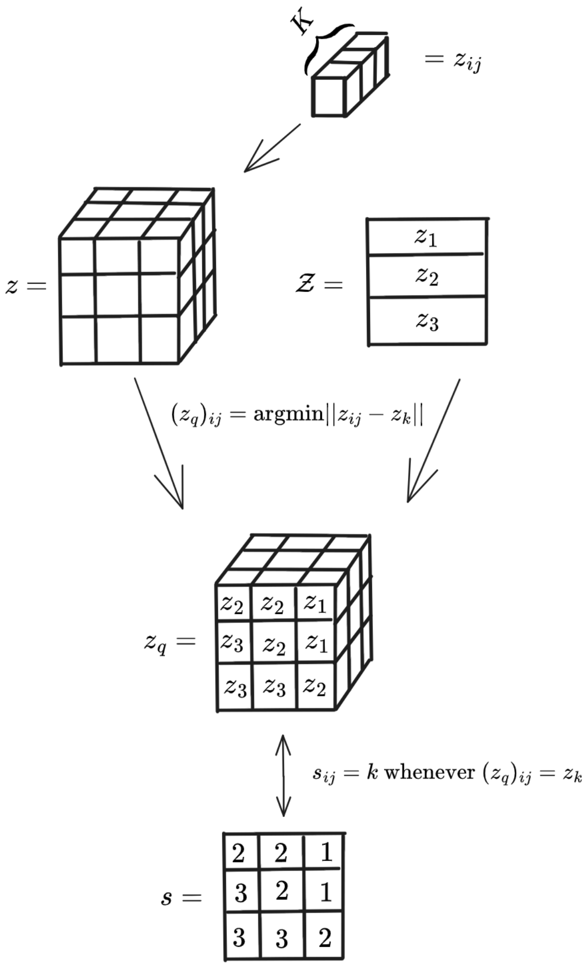
Vector Quantization (VQ) is a dictionary learning algorithm

Orthogonal regularization loss

There has been several attempts to improve vector quantization. In [**shin2023exploration**] they advocate for learning orthogonal/decorated codewords.

0.9 Masked modelling

First introduced in [**devlin2019bert**].



0.10 Evaluation metrics

0.10.1 Tokenization 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.

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

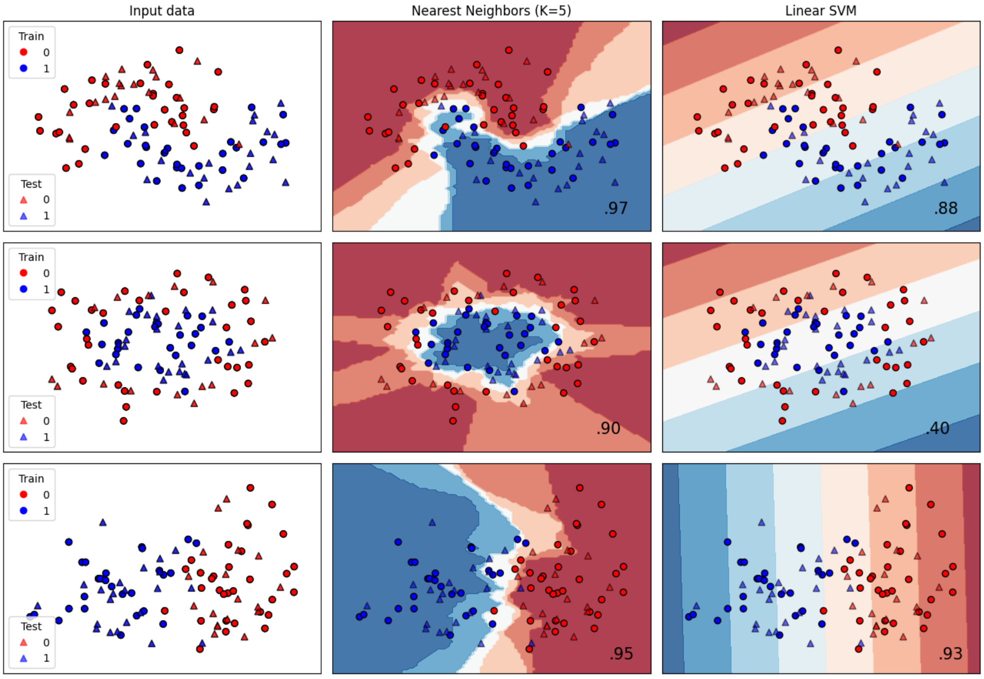


Figure 6: Modified example taken from `https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html`.

two distributions. The major limitations of this is that the visual inspections cannot be reduced to a single score, which makes objective comparison difficult.

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.

Visual Inspection

Inception Score (IS)

First introduced in [**salimans2016improved**]

$$d_{\mathrm{IS}}(\theta) = \exp\left(\mathbb{E}(D_{\mathrm{KL}}(p_\theta(y|\mathbf{x})||p_\theta(y)))\right), \tag{16}$$

The

Evaluation of IS using FCN [**wang2016time**]. Open source FCN used by us presented in [**TimeVQVAE**].

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)

Introduced in [**heusel2018gans**].

For any two probability distributions, f, g over $\mathbb{R}^n$, with finite mean and variances, their *Fréchet distance* is

$$d_F(f,g) = \left(\inf_{\gamma \in \Gamma(f,g)} \int_{\mathbb{R}^n \times \mathbb{R}^n} ||x-y||_2^2 d\gamma(x,y) \right)^{\frac{1}{2}}, \tag{17}$$

where $\Gamma(f,g)$ is the set of all *couplings* of f and g. In [**DOWSON1982450**] it was shown that for two Gaussian distributions the Fréchet distance is explicitly solvable as

$$d(\mathcal{N}(\mu,\Sigma),\mathcal{N}(\mu',\Sigma'))^2 = ||\mu-\mu'||_2^2 + \mathrm{Tr}\left(\Sigma+\Sigma'-2(\Sigma\Sigma')^{\frac{1}{2}}\right) \tag{18}$$

The *Fréchet Inception Distance* is an application of the Fréchet distance used to evaluate the generative distribution of a model against the ground truth. The name comes from its original application in image generation model.

The *Fréchet Inception Distance* calculates the Fréchet distance between two distributions by embedding a sample of each in the representation space of a classifier (Inveptio v3 / SupervisedFCN). The distribution of these representations are assumed to be Gaussian. The mean and covariance is estimated from the sample and the explicit formula is used to calculate the FID.

Rethinking FID: [**jayasumana2024rethinking**]
[**chong2020effectively**] FID and IS are biased.