Self Supervised Learning

Redha Moulla

Automn 2024

Syllabus

- Introduction to Self Supervised Learning
- Vision Transformer
- Multimodel Learning
- Contrastive Learning
- Explainability for Deep Learning

Introduction to Self Supervised Learning

Introduction Self-Supervised Learning Concepts

Supervised Learning :

- Requires labeled data (input-output pairs).
- Objective: Learn a function $f(x) = y$ to predict output y from input x.
- Example: Image classification (cat vs dog).

Unsupervised Learning :

- Does not require labeled data.
- Objective: Discover hidden structures in data (clustering, dimensionality reduction).
- Example: Customer clustering for recommendation systems.

Self-Supervised Learning (SSL) :

- Learning based on pseudo-labels generated from the data itself.
- Objective: Learn useful representations without human-annotated labels.
- Example: Predicting next word, masked pixels, image rotation, etc.

Approaches in Self-Supervised Learning

Pretext Tasks:

- The model solves simple auxiliary tasks to learn useful representations.
- Examples:
  - Predicting image rotations (0°, 90°, 180°, 270°).
  - Colorization: Predicting the original color of a grayscale image.
  - Jigsaw puzzles: Predicting the correct arrangement of shuffled image patches.

Contrastive Learning:

- The model learns to differentiate between similar and dissimilar samples.
- Pairs of positive (similar) and negative (dissimilar) samples are generated.
- Examples:
  - **SimCLR** (Simple Framework for Contrastive Learning of Representations).
  - **MoCo** (Momentum Contrast).
  - **BYOL** (Bootstrap Your Own Latent).

Masked Prediction:

- The model learns by predicting missing parts of the input data.
- Commonly used in vision (masked patches) and NLP (masked tokens).
- Example: **MAE** (Masked Autoencoders for Vision).

Importance of Learning Dense Representations

What are Dense Representations?

- Dense representations encode information in a compact, continuous vector space.
- Each dimension in the vector contributes meaningful information about the data.
- Examples: Word embeddings (e.g., Word2Vec), image embeddings (from deep networks).

Why Dense Representations Matter?

- **Generalization**: Dense embeddings can capture complex patterns in the data, enabling models to generalize to unseen examples.
- **Efficiency**: They reduce the dimensionality of the data, making computations faster and more scalable.
- **Transferability**: Dense representations learned from one task (e.g., SSL) can be fine-tuned for different downstream tasks (e.g., classification, object detection).

Learning Représentations from Text

Introduction to Embeddings

- **What is an Embedding?**
  - An embedding is a vector representation of a word that captures its context in a document, semantic, and syntactic relationships with other words.
  - Embeddings transform words into vectors of numbers so that machine learning algorithms can efficiently process them.
- **Why are they important?**
  - Embeddings capture not only the identity of a word but also its semantic and contextual aspects.
  - They facilitate tasks such as text classification, machine translation, and sentiment analysis.
- **Evolution of embeddings**
  - Historically, words were represented as indices or one-hot vectors, where each word is independent of the others.
  - Modern embeddings, such as Word2Vec and GloVe, represent words in continuous vector spaces where similar words are close to each other.

Difference Between One-hot Encoding and Embeddings

- **One-hot Encoding**
  - Each word is represented by a vector with a '1' at its specific position and '0's everywhere else. This representation is simple but very inefficient in terms of space and does not capture relationships between words.
  - Example: for a vocabulary size of 100,000:

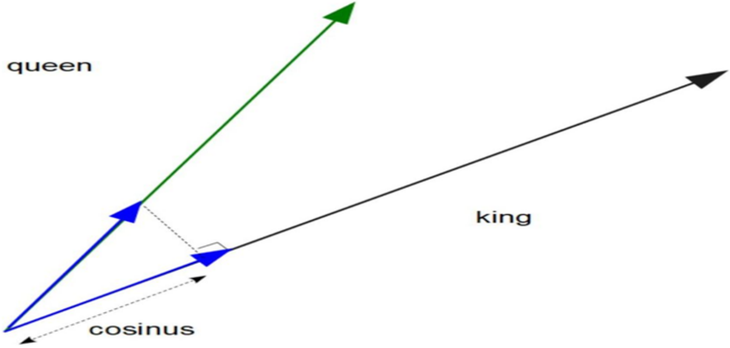
    Vehicle = [0, 0, 1, 0, 0, 0, 0, 0, 0,..., 0, 0]
    Car = [0, 0, 0, 0, 0, 1, 0, 0, 0,..., 0, 0]

    This representation does not capture semantic dimensions.

- **Embeddings**
  - Embeddings represent words as dense vectors of floating-point numbers (usually between 50 and 300 dimensions).
  - This representation is much richer and can capture complex relationships between words, such as semantic similarity.

Semantic Similarity

We are interested in word representations that capture semantic distance, using methods like dot product or cosine distance.



Distributed Representations: Principles

"You shall know a word by the company it keeps"

— J.R. Firth (1957)

Words are similar if they frequently appear in the same context.

- He drives his *vehicle* home.
- He drives his *car* home.

Constructing a Distributed Representation

Let's consider the following corpus as an example:

cnn in crop analysis
cnn and svm are widely used.
linear_regression performed along with svm
linear_regression for crop and farm
svm being used for farm monitoring
Do cnn, svm and linear_regression appear in the same context?

The vocabulary would be:

`[cnn, in, crop, analysis, and, svm, are, widely, used, linear_regression, performed, along, with, for, farm, being, monitoring, do, appear, the, same, context]`

$$\text{dim(vocabulary)} = 22$$

Semantic Similarity

The co-occurrence matrix represents how often words appear together in pairs.

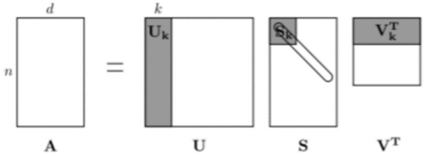
```
         cnn in crop ...
cnn   [[0, 2, 1, 1, 1, 2, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1],
in     [2, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1],
crop   [1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0],
       [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
...    [1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0],
       [2, 1, 0, 0, 1, 0, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 1, 2, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [1, 1, 1, 0, 1, 2, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1],
       [0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 2, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 2, 0, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0],
       [1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1],
       [1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1],
       [1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1],
       [1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1],
       [1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0]])
```

Dimensionality Reduction

Singular value decomposition (SVD) is a technique used to reduce the dimensionality of data (similar to PCA).

Given a matrix $A \in \mathbb{R}^{n \times d}$

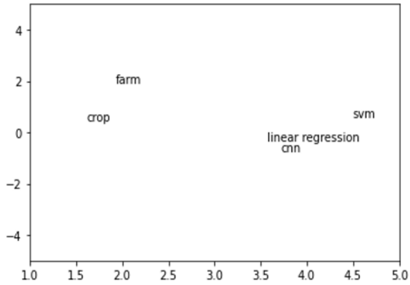
$$A = UDV^T \quad \text{where} \quad U \in \mathbb{R}^{n \times r}, D \in \mathbb{R}^{r \times r}, V \in \mathbb{R}^{d \times r}.$$



U is the matrix containing the representations (word vectors).

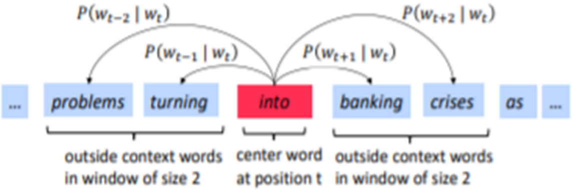
Visualization of Distributed Representations

```
cnn  [[ 3.72113518, -0.73233585],
In    [ 2.7940387 , -1.40864784],
crop  [ 1.61178082,  0.44786021],
...   [ 0.77784994, -0.31230389],
      [ 2.12245048,  1.29571556],
      [ 4.49293777,  0.58090417],
      [ 1.33312748,  0.66758743],
      [ 1.33312748,  0.66758743],
      [ 2.25882809,  1.80738544],
      [ 3.56593605, -0.31006976],
      [ 0.95394179,  0.079159  ],
      [ 0.95394179,  0.079159  ],
      [ 0.95394179,  0.079159  ],
      [ 1.92921553,  1.94783497],
      [ 1.92921553,  1.94783497],
      [ 1.12301312,  1.42126096],
      [ 1.12301312,  1.42126096],
      [ 2.26025282, -1.31571175],
      [ 2.26025282, -1.31571175],
      [ 2.26025282, -1.31571175],
      [ 2.26025282, -1.31571175],
      [ 2.26025282, -1.31571175]]
```

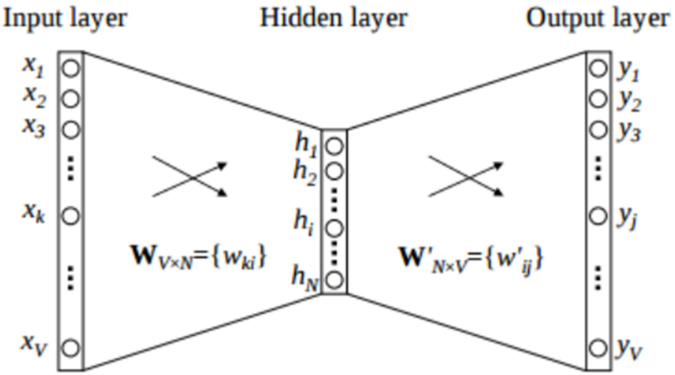


Word2Vec: Principles

- **Basic Principles:**
  - Word2Vec is based on the distributional hypothesis: words that appear in similar contexts have similar meanings.
  - Uses a shallow neural network to learn word embeddings from large corpora.
- **Two main architectures:**
  1. *Continuous Bag of Words (CBOW)*: Predict the target word from its context.
  2. *Skip-Gram*: Predict the context from the target word.



CBOW Model Architecture



CBOW Model: Mathematical Formulation

- **Objective:** Predict the target word w_t based on its context C.
- **Prediction function:**

$$P(w_t|C) = \frac{e^{\mathbf{v}_{w_t}^T \cdot \mathbf{h}}}{\sum_{w \in W} e^{\mathbf{v}_w^T \cdot \mathbf{h}}} \tag{1}$$

- **Where:**
  - $\mathbf{v}_w$ is the embedding vector for word w.
  - $\mathbf{h}$ is the context vector, the average of the embeddings of the context words.
  - W is the set of all words in the vocabulary.
- **Optimization:** Minimizing the cost function, often a form of cross-entropy or negative log-likelihood.

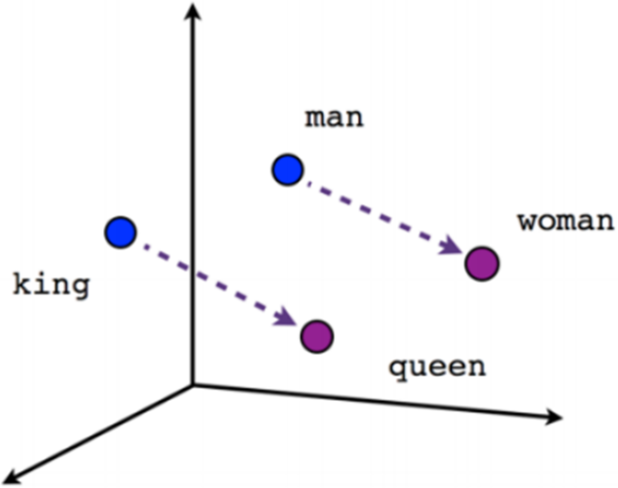
Skip-Gram Model: Mathematical Formulation

- **Objective:** Predict the context words from the target word w_t.
- **Prediction function:**

$$P(C|w_t) = \prod_{w_c \in C} \frac{e^{\mathbf{v}_{w_c}^T \cdot \mathbf{h}}}{\sum_{w \in W} e^{\mathbf{v}_w^T \cdot \mathbf{h}}} \tag{2}$$

- **Where:**
  - $\mathbf{v}_{w_c}$ is the embedding vector for the context word w_c.
  - $\mathbf{h}$ is the embedding vector for the target word w_t.
- **Optimization:** Minimizing the cost function, often a form of cross-entropy or negative log-likelihood.

Word2Vec Representations



Practical Applications of Word2Vec

- **Word Analogies:**
  - Word2Vec is famous for capturing complex relationships, like "man is to woman as king is to queen."
  - It solves analogies using simple arithmetic operations on word vectors.
- **Semantic Clustering:**
  - Word embeddings can be used to group semantically similar words, facilitating the analysis of large text corpora.
- **Enhancing Recommender Systems:**
  - Word2Vec embeddings can improve recommendation accuracy by better understanding user preferences.

Learning Representations from Images

Learning Representations from Images

How do models learn from images?

- Image representation learning involves transforming raw pixel values into meaningful features that capture important visual patterns.
- The model learns to extract **hierarchical features** at different levels, from simple edges to complex shapes.
- Deep learning models, such as **Convolutional Neural Networks (CNNs)**, are commonly used for this task.

Goal:

- Learn **dense representations** (feature vectors) that preserve key information about the image.
- Use these representations for tasks like classification, object detection, and image generation.

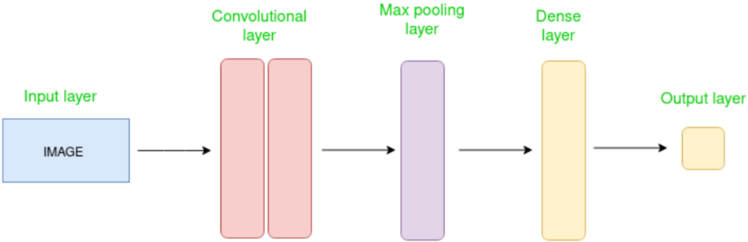
Convolutional Neural Networks (CNNs)

How CNNs work:

- CNNs apply filters (kernels) to the input image to detect **local patterns**, such as edges, textures, and corners.
- As the network goes deeper, it learns more **abstract features** by combining low-level patterns into higher-level concepts.

Layers in CNNs:

- **Convolutional Layers:** Extract features from local regions of the image.
- **Pooling Layers:** Reduce the spatial dimensions while preserving important features.
- **Fully Connected Layers:** Combine features to make final predictions.



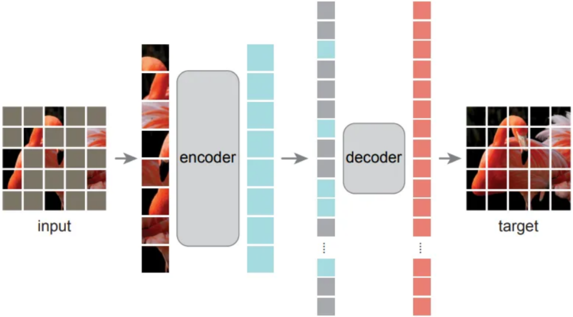
Masked Image Modeling

Masked Autoencoders (MAE):

- Inspired by masked language models (e.g., BERT), the model learns to reconstruct the missing parts of an image.
- A large portion of the image is masked (e.g., 75

Why it's useful:

- The model learns **global features** to complete the image.
- It captures both low-level and high-level semantic information from the visible parts.



Autoencoders

Introduction to Autoencoders

Definition and Purpose:

- Autoencoders are neural networks used to learn efficient representations of data, typically for the purpose of data compression and reconstruction.
- They consist of an encoder and a decoder:
  - The **encoder** compresses the input into a latent-space representation.
  - The **decoder** reconstructs the input data from this compressed representation.

Applications in Image Processing:

- Dimensionality reduction
- Noise reduction
- Anomaly detection in images

Goal of Autoencoders:

- Minimize the reconstruction error, i.e., the difference between the original input and the output after encoding and decoding.

Basic Architecture of an Autoencoder 1/2

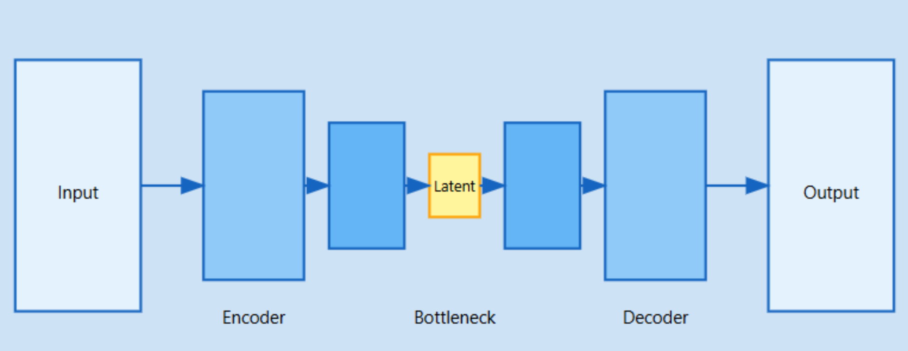
Overview of the Architecture:

- Autoencoders consist of three main components:
  - **Encoder**: Maps the input to a lower-dimensional latent space representation.
  - **Latent Space (Bottleneck)**: Holds a compressed version of the input data.
  - **Decoder**: Reconstructs the input data from the latent representation.

Details of Each Component:

- **Encoder**:
  - Series of layers that reduce the input's dimensionality.
  - Typically includes convolutional layers for image data to capture local features.
- **Bottleneck**:
  - Smallest dimension in the network, representing the most essential features of the input.
  - Acts as a constraint, forcing the model to learn compact representations.
- **Decoder**:
  - Series of layers that progressively upsample to reconstruct the input.
  - Uses transposed convolution layers to recover spatial dimensions in image data.

Basic Architecture of an Autoencoder 2/2



Encoding with Convolutional Layers

Purpose of Convolutional Layers in the Encoder:

- Convolutional layers help capture spatial hierarchies by learning local patterns in the input image.
- They are particularly effective for images, allowing the model to detect edges, textures, and shapes in a hierarchical manner.

How Convolutional Layers Work:

- **Filters**: Small matrices (e.g., 3x3 or 5x5) that slide across the image, learning features from different regions.
- **Strides and Padding**:
  - **Stride**: Controls the step size for the filter movement, affecting the output size.
  - **Padding**: Maintains the spatial dimensions by adding a border of zeros around the image.
- **Activation Functions**: Commonly, ReLU is used to introduce non-linearity and help the model learn complex patterns.

Convolutional Layers

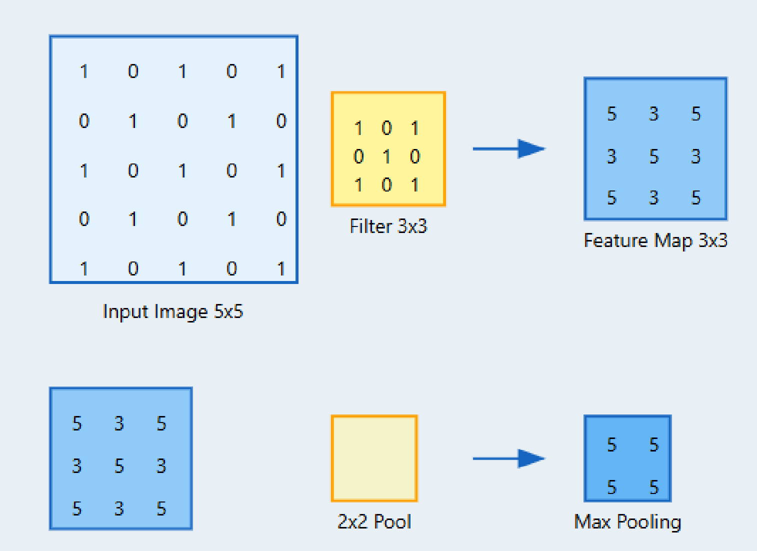
Filter Operation in Convolutional Layers:

- **Filters (Kernels)**: Small matrices (e.g., 3x3, 5x5) used to detect specific patterns in local regions of the input.
- **Convolution Process**:
  - The filter slides across the input image, performing an element-wise multiplication followed by summation to create a feature map.
  - Multiple filters are used to extract different types of features (edges, textures, etc.).

Pooling for Downsampling:

- **Max Pooling**: Reduces the spatial dimensions of the feature map, retaining the most salient features and decreasing computational load.
- Pooling layers are usually added after convolutional layers to progressively reduce the image size and focus on the most important features.

Convolutional Layer: example



Transposed Convolution (Deconvolution) Layers

Purpose of Transposed Convolution:

- Transposed convolutions (also called deconvolutions) are used to upsample feature maps, restoring spatial dimensions in the decoder.
- They are essential in reconstructing images from the compressed representation in the latent space.

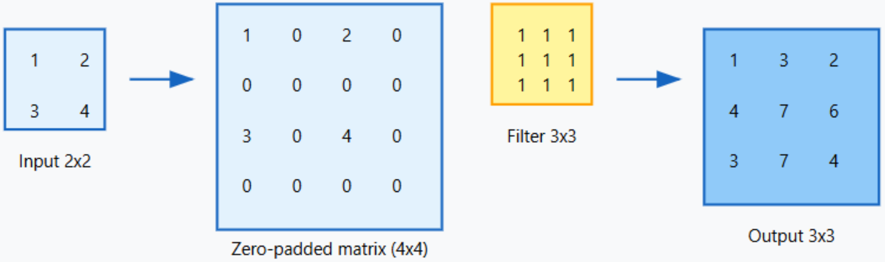
How Transposed Convolution Works:

- Unlike standard convolution, transposed convolution increases spatial dimensions.
- **Reversing the Convolution Process**:
  - In a transposed convolution, zeros are typically inserted between pixels in the feature map before applying the convolution.
  - This technique effectively "expands" the input dimensions, allowing the decoder to generate a higher-resolution output.

Stride and Padding in Transposed Convolution:

- **Stride**: Controls the upsampling factor by defining the spacing between each output pixel.
- **Padding**: Similar to standard convolution, padding can be applied to control the final output size.

Deconvolution: example



Transposed Convolution Process:

1. Input 2x2 is expanded with zeros (4x4)
2. Apply 3x3 convolution filter
3. Result is a 3x3 feature map

Latent Space (Bottleneck) Layer

Purpose of the Latent Space:

- The latent space, also known as the bottleneck, holds a compressed representation of the input data.
- It forces the autoencoder to learn the most essential features, discarding less important details.

Characteristics of the Latent Space Representation:

- The latent space has a lower dimensionality than the input, which encourages the model to capture high-level features.
- Often, the features in the latent space are not directly interpretable but represent important patterns or structures in the data.

Introduction to Variational Autoencoders (VAE)

What are Variational Autoencoders (VAE)?

- VAE is a generative model that learns to encode data into a structured latent space and decode it back to approximate the original input.
- Unlike traditional autoencoders, VAEs treat the latent space as a probabilistic distribution, allowing for more flexible and meaningful representations.

Key Differences from Standard Autoencoders:

- VAEs introduce a probabilistic element by encoding the input as a distribution rather than a single point.
- This allows VAEs to generate new data points by sampling from the latent distribution, making them useful for data generation tasks.

Applications of VAEs:

- Image generation and synthesis (e.g., creating new, realistic images).
- Interpolation in latent space, which enables smooth transitions between data points.
- Useful in applications like anomaly detection, where abnormal inputs deviate from the learned distribution.

Architecture of a Variational Autoencoder (VAE)

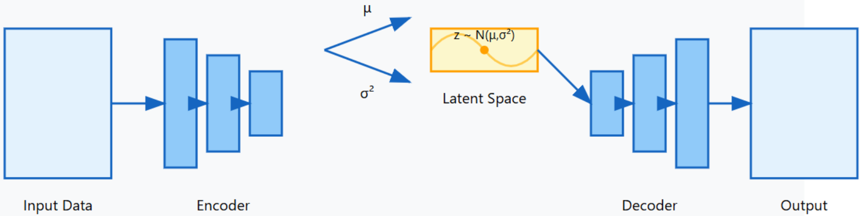
Overview of VAE Structure:

- Like a standard autoencoder, a VAE consists of an encoder, a latent space, and a decoder.
- However, in VAEs, the encoder maps the input into a distribution in the latent space rather than a fixed point.

Components of a VAE:

- **Encoder**:
  - Maps the input data to a latent distribution, producing a mean (μ) and variance (σ^2) for each dimension in the latent space.
- **Latent Space (Probabilistic)**:
  - Represents the data as a distribution, typically a Gaussian, from which new data points can be sampled.
  - Allows for structured data generation and smooth transitions in the latent space.
- **Decoder**:
  - Reconstructs the input from a sampled point in the latent distribution, generating new data based on the learned structure.

VAE Architecture



Key Components:

- Encoder progressively reduces dimensions to latent space
- Latent space captures probabilistic distribution
- Decoder progressively expands dimensions to reconstruction

Loss Function in Variational Autoencoders (VAE)

VAE Loss Function Overview:

- The VAE loss combines two terms: **reconstruction loss** and **KL-divergence loss**.
- This combination ensures that the model learns both accurate reconstruction and a structured latent space.

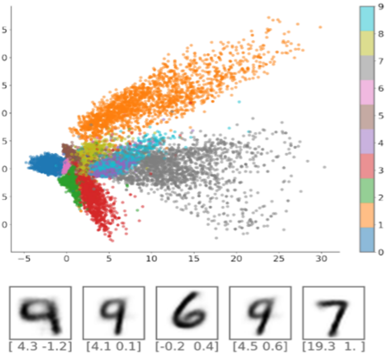
1. Reconstruction Loss:

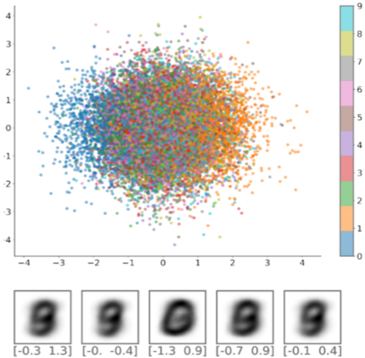
- Measures the difference between the input and its reconstruction, typically using Mean Squared Error (MSE) or binary cross-entropy.
- Encourages the decoder to generate outputs that closely resemble the original input.

2. KL-Divergence Loss:

- Measures the difference between the learned latent distribution and a standard normal distribution ($N(0, 1)$).
- Acts as a regularizer, encouraging the latent space to be smooth and structured, with samples close to a normal distribution.
- Formula: $\text{KL}(q(z|x)||p(z)) = \int q(z|x) \log \frac{q(z|x)}{p(z)} dz$

Decomposition of Loss Function





Training and Optimization of VAE 1/2

Training Objective:

- The goal is to minimize the combined VAE loss, balancing reconstruction accuracy with a structured latent space.
- The training process adjusts the encoder and decoder weights to optimize both reconstruction and regularization terms.

Reparameterization Trick:

- To backpropagate through the sampling operation in the latent space, the reparameterization trick is used.
- Instead of directly sampling $z \sim q(z|x)$, we sample $\epsilon \sim N(0, 1)$ and compute $z = \mu + \sigma \cdot \epsilon$.
- This allows gradients to flow through μ and σ, making the training process differentiable.

Training and Optimization of VAE 2/2

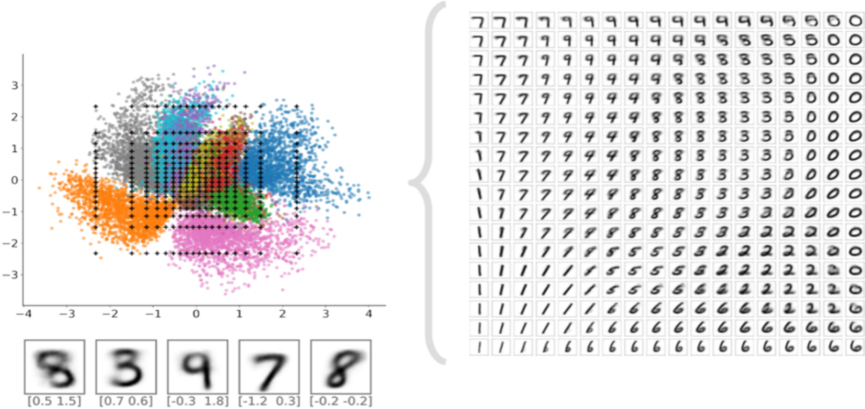
Optimization Process:

- Stochastic Gradient Descent (SGD) or Adam optimizer is typically used to minimize the VAE loss.
- Training involves iteratively adjusting the encoder and decoder weights to refine the learned latent distribution and reconstruction quality.

Trade-off between Reconstruction and Regularization:

- Adjusting the weight β on the KL-divergence term can control the balance between reconstruction fidelity and latent space regularization.
- A higher β promotes smoother latent spaces but may reduce reconstruction accuracy.

Regularization of VAEs



Thank you for your attention!

redha_moulla@yahoo.fr