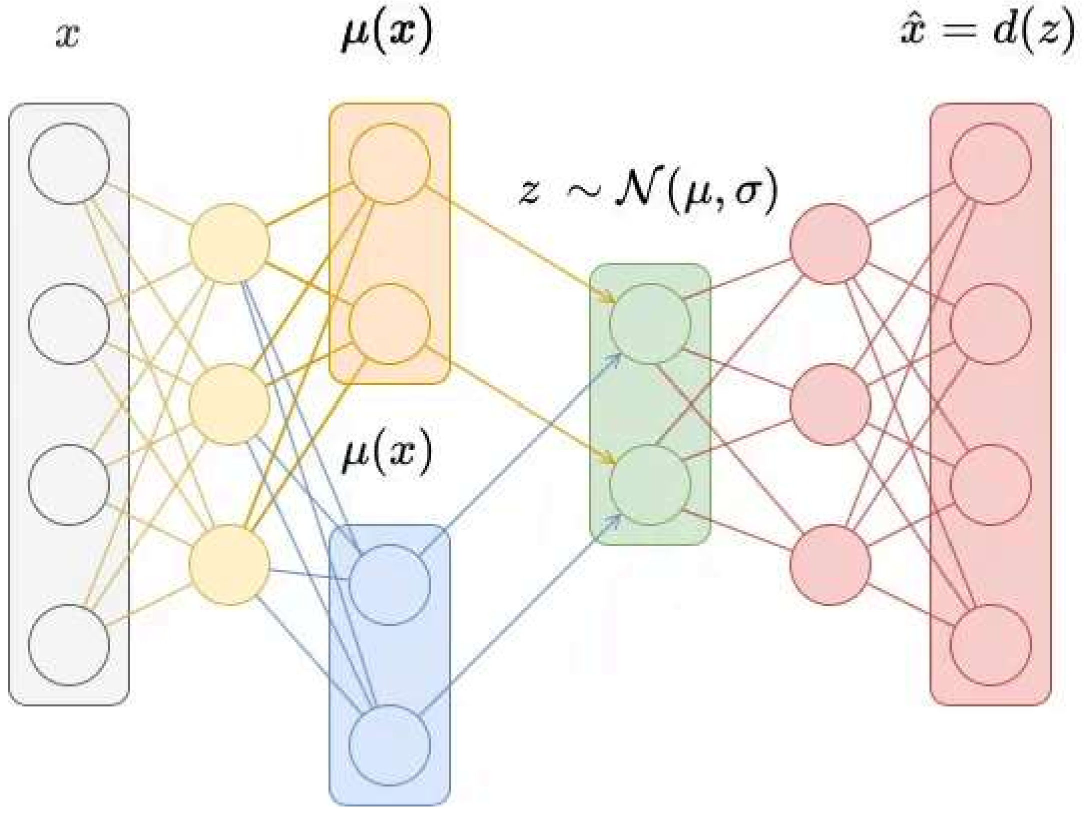
Demystifying Neural Networks: Variational AutoEncoders

Dagang Wei · Follow
6 min read · Mar 27, 2024

97 1



Source

This article is part of the series Demystifying Neural Networks.

Introduction

In the realm of deep learning, autoencoders are well known for their ability to compress and reconstruct data. Their goal is to learn efficient representations of input data, usually serving purposes like dimensionality reduction or feature extraction. Enter the Variational Autoencoder (VAE), a close relative of the traditional autoencoder, but one infused with a touch of probabilistic magic. This special twist makes VAEs incredibly powerful generative models. Let's dive in!

What is a Variational Autoencoder (VAE)?

A variational autoencoder is a type of generative neural network architecture. At its heart, a VAE still has the same structural components as a traditional autoencoder: an encoder and a decoder.

- **Encoder:** The encoder's job is to take an input (like an image) and compress it into a compact, latent representation. Traditional autoencoders output a single point in the latent space. A VAE, however, outputs a *distribution* (usually a Gaussian) over the latent space.
- **Decoder:** The decoder samples a point from this distribution in the latent space and attempts to reconstruct the original input as accurately as possible.

capabilities. Because a VAE learns to model the underlying probability distribution of the input data, it can **generate new samples** that are similar to the original data. Think generating realistic faces, composing new pieces of music, or crafting creative text.

- **Continuous Latent Space:** Unlike traditional autoencoders, VAEs promote a continuous and smooth latent space. This means that we can interpolate between points in this space to generate smoothly transitioning variations of our data. Imagine smoothly morphing one person's face into another's!

VAE vs. Autoencoder: Differences

Variational Autoencoder (VAE)

- **Latent Space:** Continuous and smooth, enabling interpolation and meaningful transitions of representations.
- **Generative Power:** Powerful generative capabilities to produce new, similar data samples.
- **Loss:** Reconstruction loss combined with KL divergence to regularize the latent space, forcing it to follow a prior distribution (often Gaussian).

Autoencoder (AE)

- **Latent Space:** Potentially discrete, which can lead to gaps or abrupt changes in representations.
- **Generative Power:** More limited in generating entirely new data samples.
- **Loss:** Primarily focused on reconstruction loss, ensuring accurate reproduction of the input.

How Does a VAE Work?

1. **The Distribution Trick:** Instead of a point in the latent space, the encoder of a VAE outputs the parameters that define a probability distribution

(usually mean and variance). During training, we sample a point from this distribution to feed into the decoder.

2. **The Reparameterization Trick:** This is the clever part. Directly backpropagating gradients through random sampling is tricky. The reparameterization trick lets us express the sampled point in the latent space as a deterministic function of the distribution parameters and an external random variable. This allows for proper training.
3. **Loss Function: Beyond Reconstruction:** The VAE's loss function has two parts:

- **Reconstruction Loss:** Just like in an autoencoder, this part ensures that the decoder accurately reconstructs the input.
- **KL Divergence:** This is where the probabilistic twist comes in. The Kullback-Leibler (KL) divergence measures how much the encoder's learned distribution diverges from a standard prior distribution (often a standard Gaussian). This encourages a well-structured, regular latent space.

The Reparameterization Trick

The core difficulty lies in the concept of backpropagation, a training technique in neural networks. Backpropagation allows us to adjust the network's internal parameters based on the difference between the predicted output and the actual output. However, it struggles when dealing with random sampling, which is inherent in VAEs.

The Traditional (Broken) Approach

Imagine the encoder outputs the mean (μ) and standard deviation (σ) of a Gaussian distribution. We then sample a random variable from the distribution as:

z = random_sample(μ, σ)

This seems straightforward, but there's a catch. Backpropagation requires calculating the gradients with respect to the model's parameters (μ and σ in this case). However, calculating the gradient with respect to z (a randomly sampled variable) is mathematically undefined. This throws a wrench in the training process.

The reparameterization trick offers an elegant solution. Instead of directly sampling z from its distribution N(μ,σ^2), we **decompose *z* into a deterministic component and a stochastic component** that is independent of the parameters we want to optimize.

Here's the magic:

1. **Introduce a new random rariable:** We introduce another independent random variable, typically another standard normal variable (ε). This new variable (ε) serves as a source of randomness entirely separate from the sampling process within the VAE.
2. **Reparameterize the latent variable:** Instead of directly adding noise, we create a scaled version of the new random variable (ε) using the standard deviation (σ) predicted by the encoder. We achieve this with element-wise multiplication:

z = μ + σ * ε

This equation might seem very similar to the original equation, but there's a crucial difference. Now, both μ and σ are deterministic outputs from the encoder, and ε is a separate source of randomness we can control.

Why Does This Work?

Even though new equation introduces a new variable (ε), it still captures the essence of the original Gaussian distribution. The scaled ε effectively injects randomness while maintaining the relationship between the mean (μ) and standard deviation (σ) defined by the encoder.

Benefits:

- **Backpropagation is Happy:** Because the new equation only involves deterministic operations, we can now calculate gradients with respect to μ and σ during backpropagation. This allows the network to learn and adjust its parameters effectively.
- **Preserves Stochasticity:** Even though the reparameterization trick uses a deterministic transformation, the final sampled latent variable (z) retains the desired statistical properties of the original Gaussian distribution. This ensures the VAE maintains its ability to explore the latent space effectively.

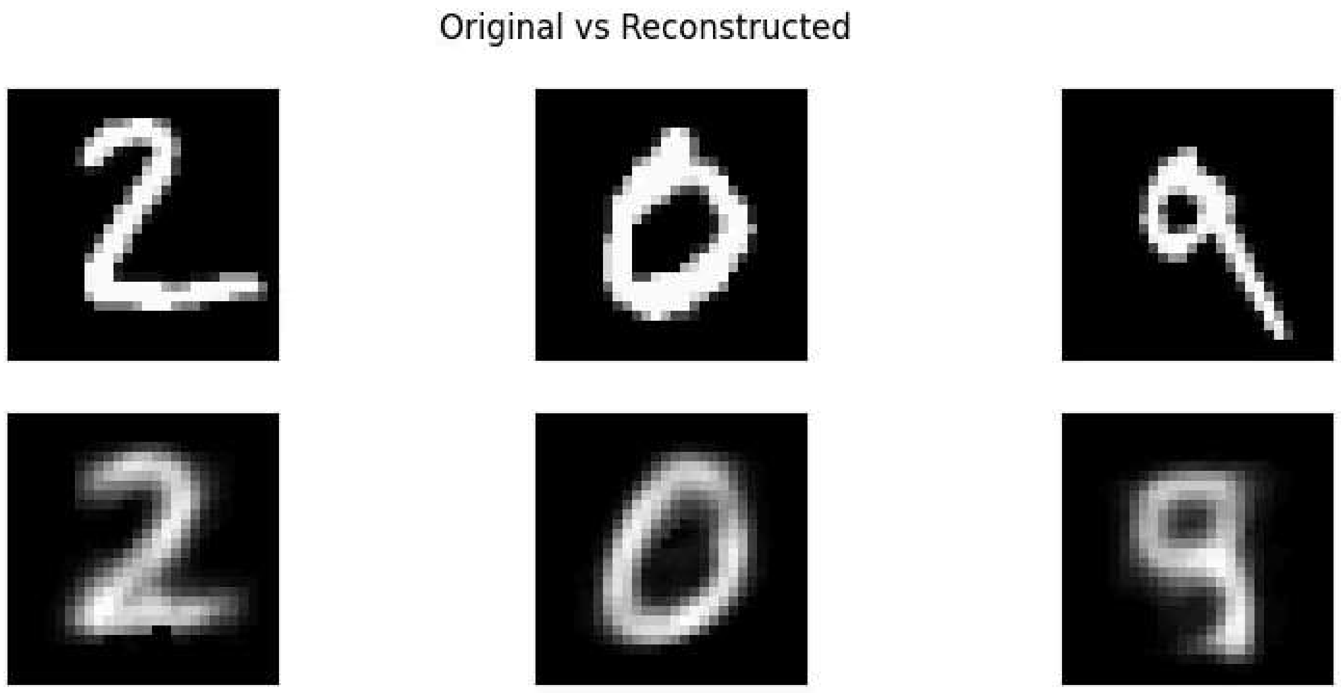
The reparameterization trick is a brilliant illustration of how a seemingly simple mathematical transformation can have a profound impact on the training process of complex neural network architectures like VAEs.

Example

The following is an example implementation of VAE on the MNIST dataset. The code is available in this colab notebook.

```python
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from keras import layers, models, losses
from keras.datasets import mnist

# Load and prepare the MNIST dataset
(x_train, _), (x_test, _) = mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

# VAE Architecture
latent_dim = 2  # Dimension of the latent space

# Encoder
encoder_inputs = layers.Input(shape=(784,))
x = layers.Dense(256, activation='relu')(encoder_inputs)
z_mean = layers.Dense(latent_dim)(x)
z_log_var = layers.Dense(latent_dim)(x)

# Reparameterization Trick
def sampling(args):
    z_mean, z_log_var = args
    epsilon = tf.random.normal(shape=tf.shape(z_mean), mean=0., stddev=1.)
    return z_mean + tf.exp(z_log_var / 2) * epsilon

z = layers.Lambda(sampling)([z_mean, z_log_var])
encoder = models.Model(encoder_inputs, [z_mean, z_log_var, z], name='encoder')

# Decoder
latent_inputs = layers.Input(shape=(latent_dim,))
x = layers.Dense(256, activation='relu')(latent_inputs)
decoder_outputs = layers.Dense(784, activation='sigmoid')(x)
decoder = models.Model(latent_inputs, decoder_outputs, name='decoder')

# VAE model (combining encoder and decoder)
vae_outputs = decoder(encoder(encoder_inputs)[2])
vae = models.Model(encoder_inputs, vae_outputs, name='vae')

# Loss function (reconstruction loss + KL divergence)
```

```
reconstruction_loss = losses.mse(encoder_inputs, vae_outputs)
reconstruction_loss *= 784  # Rescale due to input shape
kl_loss = 1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var)
kl_loss = tf.reduce_mean(kl_loss)
kl_loss *= -0.5
vae_loss = tf.reduce_mean(reconstruction_loss + kl_loss)
vae.add_loss(vae_loss)

# Training
vae.compile(optimizer='adam')
vae.fit(x_train, x_train, epochs=30, batch_size=128)

# Test and Visualization
n_to_visualize = 3
digit_size = 28
figure = np.zeros((digit_size * 2, digit_size * n_to_visualize))  # Two rows for

# Choose specific test images
test_image_indices = [1, 3, 7]  # Feel free to change these indices
images = x_test[test_image_indices]

# Generate reconstructions
_, _, encoded_images = encoder.predict(images)
decoded_images = decoder.predict(encoded_images)

for i, idx in enumerate(test_image_indices):
    # Original image
    figure[0:digit_size, i * digit_size:(i + 1) * digit_size] = images[i].reshap

    # Reconstructed image
    figure[digit_size:, i * digit_size:(i + 1) * digit_size] = decoded_images[i]

plt.figure(figsize=(10, 4))  # Adjust figure size
plt.suptitle('Original vs Reconstructed')  # Add a title
for i in range(n_to_visualize):
    plt.subplot(2, n_to_visualize, i + 1)
    plt.imshow(figure[0:digit_size, i*digit_size :(i+1)*digit_size], cmap='gray'
    plt.xticks([])  # Remove ticks
    plt.yticks([])

    plt.subplot(2, n_to_visualize, n_to_visualize + i + 1)  # Subplot for recons
    plt.imshow(figure[digit_size:, i*digit_size :(i+1)*digit_size], cmap='gray')
    plt.xticks([])
    plt.yticks([])

plt.show()
```



Conclusion

Variational autoencoders provide a fascinating blend of autoencoders and probabilistic modeling. Their ability to learn organized latent spaces and their generative prowess makes them a remarkably versatile tool in the vast landscape of deep learning.

References

- https://avandekleut.github.io/vae/

Machine Learning

Written by Dagang Wei

724 Followers · 671 Following

Follow

https://www.linkedin.com/in/dagang-wei-595a68247/

Responses (1)

Write a response

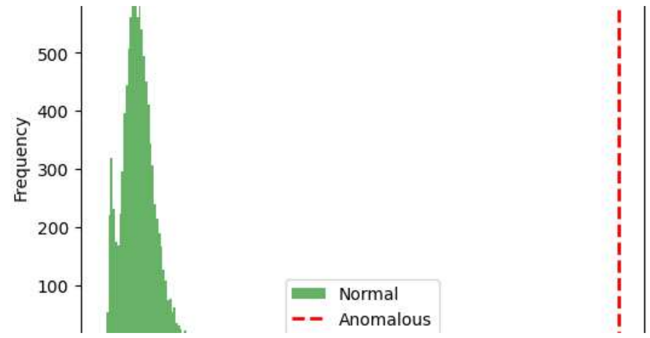
What are your thoughts?

Konrad Ciecierski
Dec 7, 2024

in the picture you have twice \mu(x) while one of them should be a \sigma(x)

Reply

More from Dagang Wei



Dagang Wei

Demystifying Neural Networks: Anomaly Detection with...

This article is part of the series Demystifying Neural Networks.

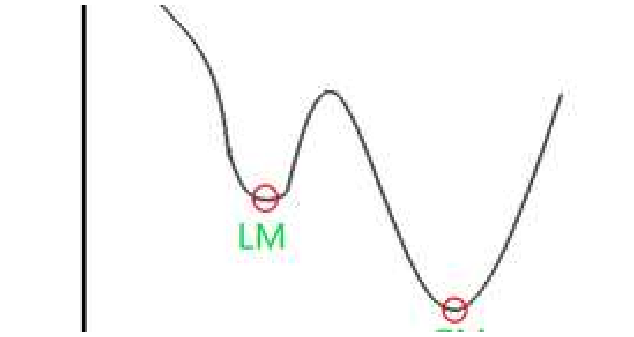
Jan 29, 2024 85 1

Dagang Wei

Having Fun with TypeScript: Zod

Schema Validation Made Easy

Mar 12, 2024 123



Dagang Wei

Demystifying the Adam Optimizer in Machine Learning

Introduction

Jan 31, 2024 53 1

Dagang Wei

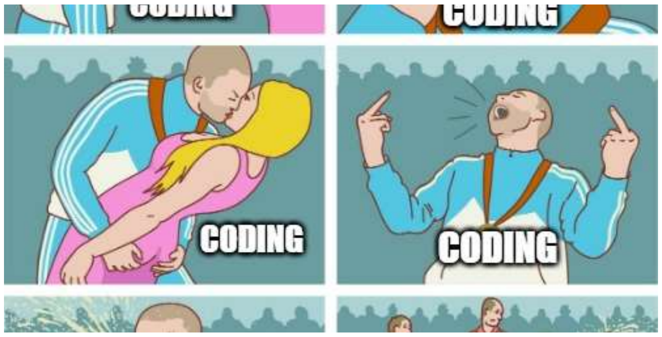
Modern C++: std::variant and std::visit

The Duo to Boost Type Safety

Aug 27, 2024 27 2

See all from Dagang Wei

Recommended from Medium

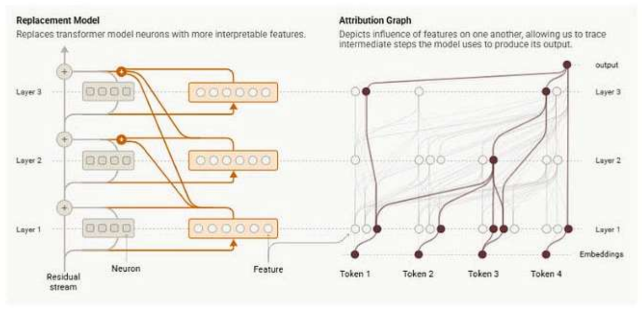


why amit

Latent Space Representations in Variational Autoencoders (VAEs)

If you think you need to spend $2,000 on a 120-day program to become a data scientist,...

Dec 9, 2024

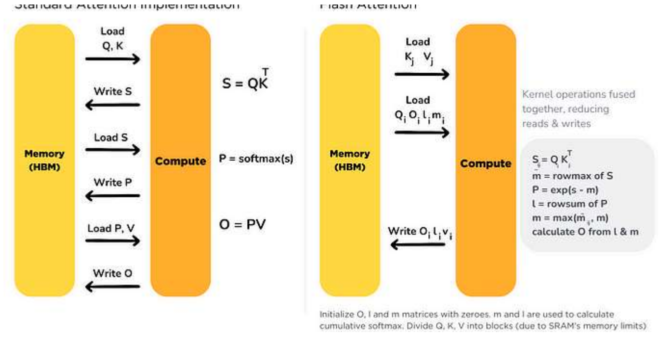

Replacement Model
Replaces transformer model neurons with more interpretable features.
Attribution Graph
Depicts influence of features on one another, allowing us to trace intermediate steps the model uses to produce its output.
Layer 3
Layer 2
Layer 1
Residual stream
Neuron
Feature
output
Embeddings
Token 1
Token 2
Token 3
Token 4

Lee Fischman

Anthropic drops an amazing report on LLM interpretability

Circuit Tracing: Revealing Computational Graphs in Language Models:

Mar 30 6

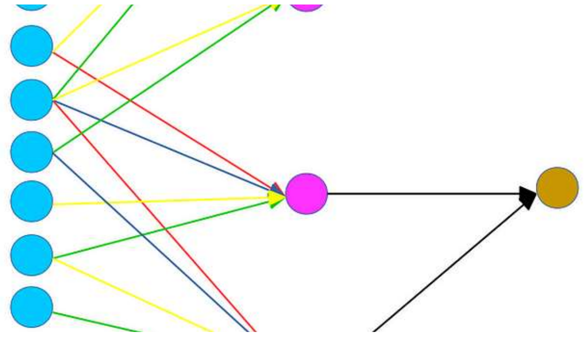


In Nailing the AI ML Interview by Dr. R. Li

AI Coding Interview: Attention Mechanism

The Attention Mechanism is the core idea behind modern large language models...

Mar 15 7

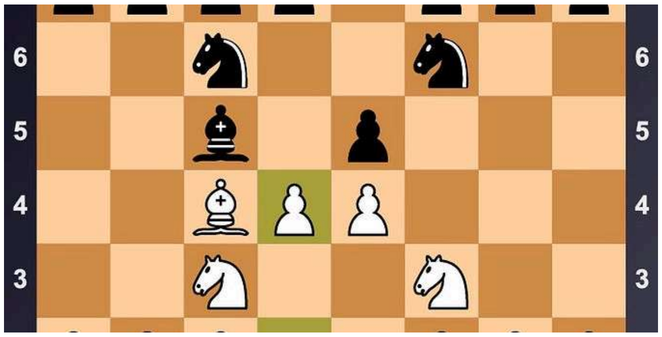


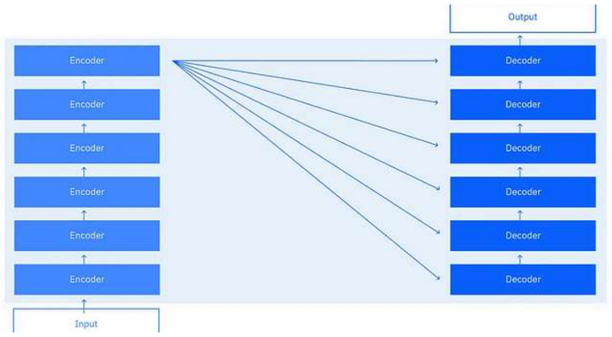
Jo Wang

CNN Basics: Convolutional Layers and Pooling Layer | How to...

Key Ingredient 1: Convolutional Layers

Oct 29, 2024 31





In AI Advances by Harys Dalvi

LLMs Do Not Predict the Next Word

RLHF forces us to view LLMs as agents in an environment, not just statistical models.

Apr 3 1.93K 41

Kaouthar EL BAKOURI

Transformers, GANs

Transformers are designed to understand context.

Feb 13 7

See more recommendations