DEEP LEARNING

**DINH HOANG VIET PHUONG – 301123263**

LAB 03 REPORT

In this report, I discuss the process of loading the Fashion MNIST dataset using TensorFlow. The Fashion MNIST dataset is a collection of grayscale images of fashion items belonging to 10 different classes, each represented by 60,000 training images and 10,000 test images.

A screen shot of a computer program

Description automatically generated

Loading the Dataset

The Fashion MNIST dataset is loaded using TensorFlow's built-in function fashion\_mnist.load\_data(). This function returns two tuples, representing training and testing sets respectively, each containing images and corresponding labels.

A computer screen shot of a program

Description automatically generated

Storing Data: The loaded dataset is split into two dictionaries, train\_phuong and test\_phuong, where each dictionary contains 'images' and 'labels' keys holding the respective data arrays.

Verifying Data Structure: The lengths of the arrays within the dictionaries are printed to verify the structure of the stored data.

Upon execution, the code verifies that the loading and storage of the Fashion MNIST dataset were successful. The printed lengths confirm that the training set consists of 60,000 images and labels while the test set comprises 10,000 images and labels, consistent with the dataset's specifications.

A computer screen shot of a program code

Description automatically generated

Main Components of the Data Pre-processing

* Normalization: The pixel values of both training and testing images are converted from integers ranging between 0 and 255 to floating-point values ranging between 0 and 1. This normalization is achieved by first converting the image data type to float32 and then dividing by 255.
* Reshaping: The images are expanded along the last axis to add an additional dimension, changing the shape from (28, 28) to (28, 28, 1). This is required for TensorFlow models that expect images to have three dimensions: height, width, and channel, even if the images are grayscale.
* Structuring Data for Supervised and Unsupervised Learning: Two dictionaries, supervised\_phuong and unsupervised\_phuong, are created to hold the normalized and reshaped images along with their corresponding labels. Despite the names suggesting a difference in learning paradigms (supervised vs. unsupervised), both dictionaries are structured similarly for this demonstration. The naming could reflect the intended use in different model training contexts.

The output displays the shapes of the images stored in supervised\_phuong and unsupervised\_phuong dictionaries:

* Supervised\_phuong['images']: The shape (60000, 28, 28, 1) indicates that the training dataset consists of 60,000 images, each of 28x28 pixels in size, with a single channel (grayscale). This confirms that the reshaping and normalization were successfully applied.
* Unsupervised\_phuong['images']: Similarly, the shape (10000, 28, 28, 1) for the testing dataset indicates 10,000 images, each also of 28x28 pixels with a single channel, prepared identically to the training images.

A computer screen shot of a program

Description automatically generated

Components of the SampleLayer:

* Inputs: The layer takes two inputs, mean and log\_var, which represent the mean and log variance of the latent space distribution for each data point in the batch. These parameters are learned by the encoder part of the VAE.
* Batch and Dimensionality: It dynamically retrieves the size of the batch (batch) and the dimensionality of the latent space (dim) from the shape of the mean input.
* Epsilon: Generates a tensor of random normal values (epsilon) with the same shape as the latent space distribution. This random tensor allows for the sampling process, introducing the necessary randomness that characterizes generative models.
* Sampling: Computes the sampled latent vector (z\_phuong) by applying the reparameterization trick: mean + exp(0.5 \* log\_var) \* epsilon. This operation effectively samples from a distribution with the given mean and variance, ensuring that the sampling process is differentiable and suitable for backpropagation.

Importance of the Sampling Layer

* The SampleLayer is critical for the functionality of the VAE. By incorporating randomness directly into the model architecture while maintaining differentiability, it allows the VAE to not only learn an efficient encoding of the data but also to explore variations around the encoded points. This ability is what makes VAEs powerful for tasks such as data generation, anomaly detection, and more.
* The latent dimension size of 2, as specified for the VAE, means that every image in the Fashion MNIST dataset is being encoded into a two-dimensional latent space. This choice allows for easy visualization of the latent space but can constrain the amount of information that can be encoded compared to higher-dimensional latent spaces. The balance between the latent space's dimensionality and the model's capability to reconstruct or generate data is a crucial aspect of VAE design.

A screenshot of a computer program

Description automatically generated

Encoder Architecture

* Input Layer: The encoder begins with an Input layer that specifies the shape of the input images, which, for the Fashion MNIST dataset, are 28x28 pixels with a single channel (grayscale).
* Convolutional Layers: A series of Convolutional layers (Conv2D) follow the input layer. These layers are designed to extract hierarchical features from the input images. Each convolutional layer uses the ReLU activation function and 'same' padding to preserve the spatial dimensions of the input, except for one layer that applies a stride of 2 to reduce the spatial dimensions by half, effectively performing downsampling.
* Flattening and Dense Layers: The output from the convolutional layers is flattened into a vector to serve as input to the dense layers. A dense layer with ReLU activation is used for further feature extraction and dimensionality reduction before reaching the latent space representation.
* Latent Space Representation: Two separate dense layers without activation functions output the mean (z\_mu\_phuong) and log variance (z\_log\_sigma\_phuong) of the latent space distribution. These layers define the parameters of the Gaussian distribution from which latent space vectors are sampled.
* Sampling Layer: The SampleLayer custom layer takes the mean and log variance as inputs and outputs a sampled latent space vector (z\_phuong). This layer implements the reparameterization trick, enabling the backpropagation of gradients through the random sampling process.

Role in the VAE Framework

The encoder plays a pivotal role in the VAE architecture by mapping input data to a probabilistic latent space. This mapping is characterized by the mean and variance parameters, which are learned during training. The latent space representation is then used by the decoder component of the VAE (not detailed in this snippet) to reconstruct the input data. The encoder, therefore, is key to understanding and learning the underlying data distribution, facilitating not only the reconstruction of input images but also the generation of new data points that resemble the original dataset.

A screenshot of a computer

Description automatically generated

A diagram of a computer

Description automatically generated

Visual Representation Analysis

The visual graph shows the flow of the data through the encoder model:

* InputLayer: It takes an input of shape (28, 28, 1), suitable for grayscale images of 28x28 pixels.
* Conv2D layers: A series of 2D convolutional layers follow, with the first maintaining the image size due to padding='same' and the second halving the dimensions to (14, 14) due to the stride of (2, 2). The next two convolutional layers keep the dimensions constant.
* Flatten layer: The output of the final convolutional layer is flattened, resulting in a single long vector of features.
* Dense layers: The first dense layer reduces the dimensionality further to 32. After that, two separate dense layers produce the mean and log variance for the latent space, each of size 2.
* SampleLayer: The custom SampleLayer takes the outputs of the mean and log variance layers to produce the final latent space output of size 2.

Textual Summary Analysis

The textual summary confirms the structure shown in the visual graph and provides additional details, such as the number of parameters in each layer:

* The total number of trainable parameters is 494,244, which is relatively large considering the small size of the latent space. This indicates a complex model that can capture detailed features from the input images.
* The parameter counts in the convolutional layers are as expected, increasing with the number of filters and size of the filters.
* The dense layer before the latent space has a significantly higher number of parameters, which is typical as it is responsible for learning a dense representation before reducing the dimensionality to the latent space.
* There are no parameters associated with the SampleLayer since it is a procedural layer that applies a mathematical operation rather than learning weights.

A screen shot of a computer code

Description automatically generated

Decoder Architecture

* Input Layer: The decoder starts with an input layer that takes a 2-dimensional vector, which represents a point in the latent space.
* Dense Layer: This layer (dense\_3) upscales the 2D input to a vector of 12544 units, using ReLU activation. This transformation prepares the latent vector for reshaping into a format suitable for deconvolution.
* Reshape Layer: The subsequent reshape layer converts the flat vector into a 3D tensor with dimensions (14, 14, 64), matching one of the encoder's intermediate representations.
* Conv2DTranspose Layer: Also known as a deconvolutional layer, it upscales the spatial dimensions from (14, 14) to (28, 28) while reducing the depth from 64 to 32, using a stride of 2, ReLU activation, and padding='same' to maintain the output size.
* Conv2D Layer: The final convolutional layer produces the reconstructed output with a single channel (depth of 1) using a sigmoid activation function, resulting in an output image with pixel values between 0 and 1.

A screenshot of a computer

Description automatically generated

Decoder Model Summary

* The summary and the provided code confirm the structure depicted in the visual representation:
* The Model: "decoder" line indicates the name of the model.
* The layers and their outputs are listed in sequence, matching the flow of data through the model.
* The Total params: 56385 line reports the number of trainable parameters in the decoder, which is substantially less than in the encoder. This is common in VAEs, as the decoder often has fewer parameters due to the simpler task of mapping the lower-dimensional latent representations back to the data space.
* The size of the model in kilobytes (220.25 KB) is also given, which is useful information for considering the computational load and memory requirements for running the model.

A diagram of a computer

Description automatically generated

A screen shot of a computer program

Description automatically generated

VAE Architecture Overview

The VAE model vae\_phuong is composed of two main components:

* Encoder: The encoder takes the input image and outputs three things: the mean and log variance of the latent space distribution, and a sampled latent vector. These components allow the VAE to encode the input data into a distribution within the latent space and sample from this distribution to generate a new output.
* Decoder: The decoder then takes the sampled latent vector and reconstructs the input image from it. This reconstruction is not typically an exact replica of the input, but it aims to be as close as possible, thereby learning the distribution of the input data.

A screenshot of a computer

Description automatically generated

Connecting Encoder and Decoder

In the code provided, the encoder and decoder\_phuong are linked as follows:

* The encoder model is applied to the input\_img to produce the latent variables z\_mu\_phuong, z\_log\_sigma\_phuong, and z\_phuong.
* Only the sampled latent vector z\_phuong is then passed to the decoder\_phuong.
* The decoder\_phuong generates the output image from the latent vector.

VAE Model Summary

The model summary reflects the complete architecture:

* The InputLayer shows that the model takes 28x28 pixel images with a single channel (grayscale).
* The encoder listed as a Functional layer (indicating that it is a model within a model) outputs three vectors, each of dimension 2, corresponding to the encoded latent space.
* The decoder, also a Functional layer, takes the 2D latent vector and outputs a 28x28x1 reconstructed image.

Parameters of the VAE

The summary indicates that the total number of trainable parameters in the VAE is 550,629. This includes:

* 494,244 parameters from the encoder.
* 56,385 parameters from the decoder.

A screenshot of a computer program

Description automatically generated

Defining the KL Divergence Loss

* KL Divergence Loss (kl\_loss): This is an essential part of the loss function for VAEs. It measures how much information is lost when using the approximate latent distribution (defined by z\_mu\_phuong and z\_log\_sigma\_phuong) to represent the true latent distribution, assumed to be a standard normal distribution. The KL divergence contributes to the regularization of the encoder by forcing the learned distribution to approximate a standard normal distribution. The formula given is a simplification that applies when the target distribution is a standard normal distribution.

Adding the KL Divergence Loss to the Model

* vae\_phuong.add\_loss(kl\_loss): This line adds the calculated KL divergence loss to the VAE model as an additional loss to be minimized. In TensorFlow, this means that the KL divergence loss will be considered along with the reconstruction loss (defined in the next step) during training.

Compiling the VAE Model

* vae\_phuong.compile(...): The model compilation step involves specifying the optimizer and the loss function. The adam optimizer is a popular choice due to its effectiveness in a wide range of problems and its adaptive learning rate capabilities.
* Loss: The main loss function used here is the 'mean\_squared\_error', which is typical for regression problems. In the context of a VAE, this loss function quantifies the difference between the input images and their reconstructed versions outputted by the decoder. It encourages the VAE to produce outputs that are as close as possible to the original inputs.

A computer screen shot of a program

Description automatically generated

Normalization and Reshaping

* Normalization: The training images from the train\_phuong dictionary are converted to float32 type and normalized to have values between 0 and 1 by dividing by 255. This step is crucial for models to converge faster and perform better because neural networks work well with small input values.
* Reshaping: The code checks if the training images are 3-dimensional (which they would be if they're only height, width, and channel) and, if so, adds an additional dimension at the end. This is necessary because TensorFlow expects images to have four dimensions: batch size, height, width, and channels.

Model Training

* The vae\_phuong.fit() method trains the model using the normalized and potentially reshaped training images as both the inputs and the targets. This is typical for autoencoders since the goal is to reconstruct the input image as output.
* The training is set to run for 10 epochs, where an epoch is one full pass through the entire training dataset.
* The batch size is set to 256, meaning that the model will update weights after processing 256 images.

Training Output

The loss values are negative, which is atypical for most neural network training scenarios. However, in the context of this VAE:

* The loss function includes a KL divergence term, which can be negative since it measures the difference between two probability distributions. It's not the raw KL divergence values that matter, but the change in those values over time.
* The goal is to minimize the negative KL divergence (maximize the KL divergence term), along with the reconstruction error, to bring the encoder's output distribution closer to a standard normal distribution.
* The increasing negative value of the loss indicates that the model is optimizing both the KL divergence and the reconstruction error effectively across epochs.

Conclusion of Training

Over the 10 epochs, the VAE seems to have improved its loss steadily, which suggests that the model's parameters are being refined and it's learning a more effective representation of the data in its latent space. This should lead to better reconstructions of the input images, as well as more meaningful generation of new images similar to the original dataset.

A screenshot of a computer program

Description automatically generated

A black and white image of a grid

Description automatically generated

Visualization of Latent Space

The code you've shared outlines the process to generate images from a VAE:

* Grid Creation: A grid size of 10x10 is specified, and percentile values are generated using TensorFlow Probability to sample points evenly spaced in the quantile function of the normal distribution. This approach ensures that the latent space is sampled uniformly, covering the variations learned by the VAE.
* Image Generation: For each pair of coordinates (xi, yi) in the latent space, the decoder (decoder\_phuong) is used to predict and generate images.
* Assembly of the Grid: These images are then placed in a grid in the order they were generated to produce a large image made up of smaller images.

Interpretation

In the visualization, the rows and columns can be thought of as variations along the two dimensions of the latent space. The patterns observed across the grid offer insight into how the VAE interpolates between points in the latent space:

* Images in the same row or column typically show a smooth transition, indicating how one latent variable changes while the other remains constant.
* Such a grid can also reveal if the VAE has learned meaningful representations. For instance, certain rows or columns might correspond to changes in a particular style or feature of the Fashion MNIST dataset, like the type of clothing.

The Generated Images

Since the VAE has been trained on the Fashion MNIST dataset, the images in the grid are expected to be different types of clothing or accessories, such as shirts, shoes, or bags, each varying slightly from its neighbors in the grid.

Conclusion

The process and the resulting grid of images are an excellent way to visually assess the performance of a VAE and its ability to generate new, varied data points. It provides a window into the "imagination" of the model, showing what it has learned about the data structure of the Fashion MNIST dataset through its latent space. If the generated images represent distinct, recognizable items of clothing, it suggests that the VAE is effectively capturing and generating the diversity of the dataset.

A collection of images of shoes

Description automatically generated

A computer screen shot of a program code

Description automatically generated

Analyzing the Latent Space Visualization

The latent space is a lower-dimensional representation learned by the encoder part of the VAE, where similar data points are expected to be closer together. The scatter plot uses z\_mu, which is the mean of the latent variable distribution assumed by the encoder. Here's what the visualization tells us:

* Axes: The x-axis (z\_mu[0]) and y-axis (z\_mu[1]) represent the two dimensions of the latent space.
* Data Points: Each point in the scatter plot corresponds to an image from the test dataset, positioned according to its z\_mu values.
* Color Coding: The color represents the true labels of the test images, which typically correspond to different classes or types of items within the dataset. The color scale is shown on the right, where each color corresponds to a label (e.g., a type of clothing item in the Fashion MNIST dataset).
* Density and Overlap: Areas of higher density in the scatter plot suggest that many images share similar z\_mu values, indicating clusters of similar items. Overlap between different colors in these dense areas could suggest similarities between different classes or potential ambiguities in how the VAE has encoded the data.

A colorful dots on a white background

Description automatically generated

Insights

* Clustering: If the VAE has learned effective and distinct representations for different classes, we would expect to see clear clusters of the same color, with minimal overlap between different colors.
* Continuity: The plot should ideally exhibit continuity, meaning that gradual transitions in color should occur along smooth gradients, suggesting that the VAE understands the underlying structure of the data.
* Outliers: Any widely scattered points could be considered as anomalies or outliers, which might indicate unusual data points that do not fit well with the rest of the data distribution.