PICT, PUNE

Assignment 1.

Title: A comparative study of deep learning packages: Tensorflow, Keras, Theano and Pytorch.

Aim: To document the distinct features and functionality of popular deep learning packages namely tensorflow, Keras, Theano and pytorch and to classify them based on their design philosophy and use case.

Objective: It is to provide a clear overview of each package, highlighting their core characteristics, strengths and ideal applications. This study aims to help users from begining to experienced researchers, make an informed decision when choosing a deep learning framework for their specific project needs.

Theory:

Deep learning packages are open source libraries that provide a robust ecosystem of tools and API's for building and training neural networks. They simplify complex mathematical operations, such as matrix multiplication and automatic differentiation, which are fundamental to deep learning.

- Abstraction level: This refers to how much a package hides underlying complexity from the user. High level APIs, like Keras, prioritize simplicity and fast prototyping, while low-level API like Tensorflow and Pytorch, offer more granual control for advanced users.

PICT, PUNE

- Computational Graph: This is the internal representation of the models operations. Static graphs (Tensorflow's default) are defined once and then run, which is great for performance and deployment. Dynamic graphs (PyTorch) are defined 'on the fly' during execution, making them more flexible.

Tensorflow.

→ Powerful, end to end and open source platform developed by Google.

- Features:

① Versatility: Supports multiple platform (CPU, GPU, Web) and languages (Python, C++, Java etc).

② Static computational graph (by default): Allows for high performance optimization.

③ Robust Ecosystem: tools like Tensorflow Serving for deployment, Tensorflow Lite for mobile and IOT device, TensorBoard for visualization.

→ Best for large scale, production ready applications.

Keras

Keras is a high level API designed for simplicity and rapid prototyping. It was originally a standalone library but is now the official high level API for tensorflow.

- Features:

① User friendly: Focuses on user experience with a clean, concise and Pythonic API.

② Modularity: Models are built from simple, composable layers, making it easy to experiment with different architecture.

PICT, PUNE

- Best for: Beginners, educational purpose; and fast prototyping where ease of use is prioritized.

Theano

Theano is an older, low-level library that was a pioneer in the deep learning space. It's primarily known for its efficiency in symbolic graph based computations. It is no longer actively developed.

- Feature:
  1. Symbolic Differentiation: Theano was a frontrunner in providing automatic differentiation.
  2. Highly optimized: It compiles mathematical expressions into efficient code for both CPU and GPU.
  3. Rigid structure: Its low-level nature and static graph made it more cumbersome to use.
- Best for: It is now considered a historical framework, but its contribution laid the groundwork for modern lib.

Pytorch

Deep learning framework developed by META.

- Features:
  1. Dynamic computational Graphs: The 'define by run' approach allows for realtime changes to the model architecture and makes debugging much easier.
  2. Pythonic: Integrated with python language.
  3. Research friendly: It is flexible and easy to use.
- Best for: Academic research, fast prototyping, and projects that require highly customized or dynamic model architecture.

PICT. PUNE

Conclusion

The deep learning landscape offers diverse tools tailored for different needs. Keras is the go to for simplicity and rapid prototyping. Tensorflow is robust choice for large scale production and deployment while Pytorch excells in research and development.

PICT, PUNE

Assignment 2.

Title : Implementing a feedforward neural network for image classification on the MNIST dataset.

Aim : Implement a feedforward neural network using Keras and Tensorflow to classify MNIST dataset.

Objective : • load and preprocess the MNIST dataset.
• Define a sequential neural network architecture.
• Train the model using stochastic Gradient Descent (SGD).
• Evaluate model performance.
• Visualize train/test loss by plotting.

Theory :
A feedforward neural network is a type of artificial neural network where connections between the nodes do not form a cycle. Data flows in one direction, from input layer, through one or more hidden layer. and finally to the output layer.

Keras is a high level API for building and training deep learning models. It's user friendly and runs on top of Tensorflow, a powerful open source machine learning platform. Together they simplify the process of creating complex neural network. The MNIST dataset consists of 70,000 images of handwritten digits (0-9). with 60,000 for training and 10,000 for testing. It is a standard benchmark for image classification tasks.

PICT, PUNE

Algorithm

1. Import libraries: Import necessary library like tensorflow.keras, matplotlib and numpy.
2. load and preprocess data: load the MNIST dataset. we will reshape and normalize the data by scalling the pixel values from the range [0,255] to [0,1].
3. Define Network Architecture: Use keras.models.Sequential to create a linear stack of layers.
   - A Flatten layer to convert the 2D image to 1D vector
   - Dense hidden layer with a ReLU activation
   - Softmax activation function for classification.
4. Compile the model: Stochastic Gradient Descent optimizer used.
5. Visualise the results: Plot loss curves.

conclusion:

A ~~feedfoor~~ feedforward neural network for classifying handwritten digits from MNIST dataset. The network architecture, built using Keras, demonstrate the effectiveness of deep learning for image classification.

PICT, PUNE

Assignment 3.

Title: Image classification model

Aim: Develop and evaluate an image classification model capable of accurately categorizing image into predefined classes. This involves understanding and implementing the fundamental stages of machine learning pipeline for computer vision task.

Objective:
- To load and preprocess image dataset.
- Design a suitable CNN architecture for image classification
- To train the designed model using appropriate optimization techniques.
- To estimate and evaluate the models performance using standard metrics.

Theory:
Image classification is a core task in computer vision that involves assigning a label or class to an input image. This process typically relies on deep learning techniques, particularly Convolutional Neural Network (CNN). CNN are well suited for image data due to their ability to automatically learn hierarchical feature from raw pixel values.
- Convolutional layers: These layers apply filters to input images to detect patterns and features (eg, edges, textures).
- Pooling layers: These layers reduce the spatial dimensions of the feature maps, helping to make the

model more robust to variations and computational load.
- Fully Connected layer: These layers interpret the high level features learned by the convolutional and pooling layers and perform the final classification.

The model learns by minimizing a loss function during training, which quantifies the difference between the model's internal parameter (weights and biases) to reduce this loss.

Algorithm:

(a) Loading and preprocessing the Image Data:
Data loading, resizing, normalization, splitting.

(b) Defining the model architecture:
Input layer, convolutional block, flatten layer, dense layer and output layer.

(c) Training the model:
Compilation, fitting and callbacks.

(d) Estimation:
Evaluation on Test Set / Train Test.

Conclusion

Outlines the systematic process of building an image classification model. The use of CNNs, coupled with proper data handling and evaluation techniques, ensures the model's effectiveness in real world image categorization tasks.

PICT, PUNE

Assignment 4

Title: Anomaly detection using Autoencoder.

Aim: Implement an Autoencoder neural network for the unsupervised detection of anomalies within a given dataset.

Objective: Use Autoencoder to implement anomaly detection. Build the model using:

a. Import required libraries
b. Upload / access the dataset.
c. Encoder converts it into latent representation
d. Decoder networks. convert it back to the original input.
e. Compile the models with optimizer, loss and Evaluation metrics.

Theory

An autoencoder is a type of artificial neural network used for learning efficient data codings in an unsupervised manner. It consists of two main part:

1. Encoder: This part compresses the input data into a lower-dimensional representation, often called the latent space or bottleneck layer.
2. Decoder - This part reconstructs the input data from the latent representation. The goal is for the reconstructed output to be as close as possible to the original input.

For anomaly detection, autoencoders are particularly useful because they are trained to minimize the

PICT, PUNE

reconstruction error (the difference between the input and its reconstruction) for normal data when an anomalous data point is fed into the trained autoencoder, it will likely result in a significantly higher reconstruction error compared to normal data, as the model has not learned to efficiently encode and decode such patterns This high reconstruction error serves as the indicator for an anomaly.

Algorithm.

1. Data Preparation:
   - Crucially, separate the normal data from any known anomalies, or assume the majority of training data is normal.
2. Model Architecture: Combine the encoder and decoder to form the complete autoencoder model.
3. Monitor the training loss to ensure convergence.
4. Anomaly detection:
   Calculate the reconstruction error for each point. Set a threshold for the reconstruction error. Data points over threshold are classified as anomalies.

Conclusion:

Autoencoders provide an effective and unsupervised approach to anomaly detection. Particular model in scenarios where labeled anomaly data is scare or non-existent are valuable.

PICT, PUNE

Assignment 5

Title: Implement the continous bag of words (CBOW) method.

Aim: Continous bag of words (CBOW) model, a neural network architecture used in natural language processing (NLP) to learn word embeddings. This model predicts a target word based on its surrounding context words.

Objective: Implement the CBOW model from scratch, covering all key strategies: data preparation, training data generation, model training, and outputting the learned word vectors.

Theory:
The CBOW model is a shallow, two layer neural network designed to learn efficient representations of words, known as word embeddings. Unlike the skip-gram model, which predicts context words from a target word, CBOW work in the opposite direction. It takes a 'bag' of context words (the words immediately before and after the target words) and predicts the target word itself.
The core idea is that a words meaning can be inferred from its context. The model learns to map a one hot encoded representation of the context words to a probability distribution over the entire vocabulary, with the highest probability corresponding to the

actual target word. The hidden layer of the network, after training, represents the learned word embeddings. These embeddings are dense vectors where semantically similar words are located close to each other in vector space

Algorithm

(a) Data Preparation :
① Corpus selection: Choose a text corpus
② Tokenization and preprocessing : lowercasing, creating vocabulary.

(b) Generate training data :
① Define a context window.
② Slide windows and create pair.

(c) Train Model :
① Build simple neural network with forward propagation with softmax activation function.
② Backpropagation with optimizer like Stochastic Gradient Descent (SGD) to minimize loss.

(d) Output :
① Extract Embedding.
② Evaluate.

Conclusion :
The implementation of the CBOW model successfully demonstrates a foundational approach to learning distributed representation of words. We obtain dense, low-dimensional vectors that capture semantic and syntactic relationships.

PICT, PUNE

Assignment 6.

Title: Object detection using transfer learning of CNN architectures.

Aim: Develop an object detection model using transfer learning from a pretrained convolutional Neural network (CNN). This approach leverages the powerful feature extraction capabilities.

Objective: Use transfer learning of CNN architecture
(a) load a pretrained model on a large dataset
(b) freeze parameters in models lower convolutional layers
(c) Train classifier layer on training data for task.
(d) Fine tune hyper parameter and unfreeze more layers as needed.

Theory:
Transfer learning is a machine learning technique where a model trained on one task is reused as the starting point for a model on a second, related task. In the context of CNNs for image tasks, a model pretrained on a massive dataset like ImageNet has already learned to identify a wide range of fundamental features, such as edges textures and shapes, in its initial convolutional layers. The subsequent layers learn more complex patterns.

By using a pretrained model, we can bypass the need to train a CNN from scratch, which require a vast amount of data and computational resource

PICT, PUNE

Instead, we can 'transfer' this knowledge. The initial convolutional layers act as a feature extractor, providing a rich representation of the input image. We then add a new, trainable classifier on top of this frozen base. This new classifier learns to map the extracted features to the specific classes in our target dataset. This is highly effective and efficient way to build high performance models, actually when the target dataset is small.

Algorithm:

1. Choose a suitable pretrained CNN model.
2. Remove the original fully connected layers, set the trainable attribute of the layers in the convolutional base to False.
3. Construct and add a new custom classifier on top of the frozen base.
4. Compile the modified model with a suitable optimizer, a loss function and metrics.
5. After the intial training, the model can be further optimized and gently adjust the weight.

Conclusion:

Transfer learning provides an efficient and powerful method for object detection, particularly when working with limited datasets.