Day 14

Date 22 June 2024

Daily Report

Today's training session was based on another deep learning method - Convolutional Neural Network

---

Today's topic

Convolutional Neural Network

Convolutional Neural Networks (CNNs) are a type of deep learning model commonly used for image and object recognition tasks. They are composed of several layers, including convolutional layers, pooling layers, and fully connected layers.

Convolutional Layer The first layer of a Convolutional Neural Network is always a Convolutional Layer. Convolutional layers apply a convolution operation to the input, passing the result to the next layer.

Pooling Layer The goal of the pooling layer is to pull the most significant features from the convoluted matrix. Reduce the dimension of the feature map (convoluted matrix), hence reducing the memory used while training the network

Types of Pooling Layer:-

- Max Pooling Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.
- Average Pooling Average pooling computes the average of the elements present in the region of feature map covered by the filter. Thus, while max pooling gives the most prominent feature in a particular patch of the feature map, average pooling gives the average of features present in a patch.
- Sum pooling Average pooling computes the average of the elements present in the region of feature map covered by the filter. Thus, while max pooling gives the most prominent feature in a particular patch of the feature map, average pooling gives the average of features present in a patch.

Fully Connected Layer Fully Connected Layer is used to combine features learned by convolutional and pooling layers. Each neuron in a fully connected layer is connected to every neuron in the previous layer.

Image classification in CNN using Python

Libraries are used:-

- Numpy
- Tensor Flow
- Pytorch
- Keras
- Matplotlib

1. Import Libraries

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

# tensorflow and its keras module are used to build and train the neural network.
# matplotlib.pyplot is used for plotting images.
# numpy is used for numerical operations.
```

2. Loading MNIST dataset

```
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.
11490434/11490434 [==============================] - 0s 0us/step
```

3. Normalize the data :- The pixel values of the images are normalized to the range [0, 1] by dividing by 255. This helps in faster convergence during training. The images in the MNIST dataset are grayscale images with pixel values ranging from 0 to 255. Each pixel value represents the intensity of the pixel, where 0 is black, and 255 is white. Normalization is a process of scaling the pixel values to a specific range, usually [0, 1].

```
X_train, X_test = X_train / 255.0, X_test / 255.0
```

4. Reshaping the Data:- The images are reshaped to include a channel dimension, changing the shape from (28, 28) to (28, 28, 1) for compatibility with the CNN layers.

Original Shape of MNIST Data:

- The MNIST dataset consists of grayscale images of handwritten digits, each of size 28x28 pixels.

- When initially loaded, X_train and X_test are NumPy arrays with shapes (60000, 28, 28) and (10000, 28, 28) respectively, where:
- 60000 and 10000 are the number of images in the training and test sets. 28, 28 are the height and width of each image.

Grayscale images,

- A kind of black-and-white or gray monochrome, are composed exclusively of shades of gray.
- X_train.shape[0] extracts the first dimension (number of images) from the shape of X_train.
- X_train.shape[0] and X_test.shape[0] represent the number of images in the training and test datasets, respectively.
- The reshape method is used to add a channel dimension to the data.

After reshaping:

- X_train will have the shape (60000, 28, 28, 1).
- X_test will have the shape (10000, 28, 28, 1).

```
X_train = X_train.reshape((X_train.shape[0], 28, 28, 1))
X_test = X_test.reshape((X_test.shape[0], 28, 28, 1))
```

5. Defining a CNN Model:-

```
model = models.Sequential([
    #A 2D convolutional layer with 32 filters, each of size 3x3.
    #Specifies the input shape of the data. Here, each input image is 28x28 pixels with 1 ch
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),

    # A max pooling layer with a 2x2 pool size. It reduces the spatial dimensions (height ar
    layers.MaxPooling2D((2, 2)),

    #Uses the ReLU activation function to introduce non-linearity.
    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.MaxPooling2D((2, 2)),

    # A 2D convolutional layer with 64 filters, each of size 3x3.
    layers.Conv2D(64, (3, 3), activation='relu'),

    #Flattens the 3D output from the convolutional layers into a 1D vector
    layers.Flatten(),

    # A fully connected layer with 64 neurons and ReLU activation
    layers.Dense(64, activation='relu'),

    #The output layer with 10 neurons (one for each class of digits 0-9).
    layers.Dense(10, activation='softmax')
])
```

6. Compile the Model:-

```
#Optimizer: 'adam' is the optimizer that adjusts the learning rate dynamically based on grad
#Loss Function: 'sparse_categorical_crossentropy' is suitable for multi-class classificatior
# Metrics: ['accuracy'] specifies that accuracy should be monitored during training and eval
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

7. Training a model:-The model is trained for 5 epochs using the training data. Validation on the test data is performed after each epoch.

Training Process:

During each epoch, the model will:

- Iterate over batches of the training data (X_train, y_train).
- Compute the loss function sparse_categorical_crossentropy) and other metrics (like accuracy) on the training data.

- Use the optimizer (adam) to adjust the weights of the model based on the gradients computed during backpropagation.
- Evaluate the model on the validation data (X_test, y_test) after each epoch to monitor its performance on unseen data.

```
model.fit(X_train, y_train, epochs=5, validation_data=(X_test, y_test))
```

8. Evaluating a Model:-

After training the model (model.fit()), you use model.evaluate() to get the final performance metrics on unseen data (test set).

```
#test_loss gives you insight into how well the model generalizes to new, unseen data. Lower
#test_acc provides the accuracy of the model on the test set, which is the percentage of cor
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
print(f'Test accuracy: {test_acc}')
```

```
313/313 - 5s - loss: 2.3130 - accuracy: 0.0985 - 5s/epoch - 15ms/step
Test accuracy: 0.09849999845027924
```

9. Making a Predection:-

Predictions are made on the test set. The output is an array of probabilities for each class.

```
predictions = model.predict(X_test) #used to obtain predictions from a trained model (model)
```

```
313/313 [==============================] - 3s 10ms/step
```

10. Plotting a Image Predection:-

```
def plot_image_predictions(images, labels, predictions, num_images=5):
    plt.figure(figsize=(10, 10))
    for i in range(num_images):
        plt.subplot(1, num_images, i+1)
```

```
plot_image_predictions(X_test.reshape(-1, 28, 28), y_test, predictions, num_images=5)
#In the context of reshaping image data for machine learning models, -1 often represents the
```