Questionnaire (4th chapter)

Total de pontos 0/0

How is a grayscale image represented on a computer? How about a color image? *

A grayscale image is represented as a matrix of brightness intensity values for each pixel, while a color image is represented using the RGB color model with three intensity values for each pixel.

How are the files and folders in the MNIST_SAMPLE dataset structured? Why? *

The MNIST_SAMPLE dataset has two folders, "train" and "valid", which have subfolders for each digit from 0 to 9. Each subfolder has images of that digit for either the training or validation set. The purpose of this structure is to label and organize the data based on the digits in the images, making it easier to use and process for machine learning models.

Explain how the "pixel similarity" approach to classifying digits works. *

The pixel similarity approach involves comparing the pixel intensity values of an input image with those of known images in a reference dataset;

What is a list comprehension? Create one now that selects odd numbers from a list and doubles them. *

It consists of square brackets enclosing an expression followed by a for loop and optional conditional statements.

What is a "rank-3 tensor"? *

A "rank-3 tensor" is a mathematical object that can be represented as a three-dimensional array of numbers. In machine learning, rank-3 tensors are commonly used to represent images, where each element of the array corresponds to a pixel in the image and each dimension of the array corresponds to the width, height, and color channels of the image.

What is the difference between tensor rank and shape? How do you get the rank from the shape? *

tensor rank is the number of dimensions or axes in the tensor, while its shape refers to the size of each dimension, a rank-2 tensor could have a shape of (3, 4), which means it has 2 axes or dimensions and the first axis has size 3 while the second axis has size 4

What are RMSE and L1 norm? *

RMSE and L1 norm are two things we use in data science to see how good our model is. They help us understand how much our predictions are different from the true values. RMSE is like a way of measuring how wrong we are on average, while L1 norm is like the sum of all the mistakes we made. Both of them are really useful for us to see if our model is good enough, so we can make better decisions with the data.

How can you apply a calculation on thousands of numbers at once, many thousands of times faster than a Python loop? *

The calculation on thousands of numbers can be applied many thousands of times faster than a Python loop by using vectorized operations.

Create a 3×3 tensor or array containing the numbers from 1 to 9. Double it. Select the bottom-right four numbers. *

```
import torch
x = torch.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
y = x * 2
bottom_right = y[1:, 1:]
```

What is broadcasting? *

Broadcasting is a feature in NumPy that allows arrays with different shapes to be operated on with arithmetic operations.

Are metrics generally calculated using the training set, or the validation set? Why? *

Metrics can be calculated using either the training set or the validation set, but they are typically calculated using the validation set. This is because the purpose of metrics is to evaluate the performance of the model on data that it has not seen before, which is the role of the validation set. Calculating metrics on the training set can lead to overfitting, as the model may have memorized the training data and perform well on it, but not generalize well to new data.

What is SGD? *

Stochastic Gradient Descent (SGD) is an optimization algorithm commonly used in deep learning to minimize the loss function during training.

Why does SGD use mini-batches? *

To balance the trade-off between computing efficiency (using small batches) and convergence stability (using larger batches).

What are the seven steps in SGD for machine learning? *

The seven steps in SGD for machine learning are: (1) Inicialize random weights, (2) Calculate predictions for a random mini-batch, (3) Calculate loss function, (4) Calculate gradients of loss function with respect to weights, (5) Update weights based on gradients and learning rate, (6) Repeat steps 2-5 until convergence, (7) Evaluate final model performance on validation set.

How do we initialize the weights in a model? *

To prevent the model from getting stuck in a local minimum, weights in a model are initialized randomly with small values drawn from a normal or uniform distribution, thereby breaking the symmetry. Other methods like weight decay or layer-specific initialization can also be implemented.

What is "loss"? *

is a function that measures how poorly a model's predictions match the actual target values.

Why can't we always use a high learning rate? *

May result in unstable and divergent training, causing the model to overshoot the optimal weights and fail to converge.

What is a "gradient"? *

It's is like a road map of how much a function changes in different directions. It helps us to find the direction of maximum increase or decrease. In machine learning, gradients are used to change the weights of a model to minimize the loss function during training using optimization algorithms like gradient descent.

Do you need to know how to calculate gradients yourself? *

While it can be beneficial for a machine learning practitioner to have a fundamental grasp of gradient calculation, manual gradient calculation is not always required as automatic differentiation is offered by most deep learning frameworks.

Why can't we use accuracy as a loss function? *

While accuracy can be a useful metric for evaluating the performance of a model, it cannot be used as a loss function because it is not a continuous or differentiable function. This means that it cannot be optimized using gradient-based methods, which are commonly used in machine learning for model training. Instead, we typically use differentiable loss functions such as cross-entropy or mean squared error, which can be optimized using gradient descent.

Draw the sigmoid function. *

Sigmoid - BRENO...

What is special about the shape of the sigmoid function? *

The sigmoid function has a distinctive S-shaped curve, which is bounded between 0 and 1, making it useful for modeling probabilities in binary classification problems. Additionally, the function is smooth and differentiable, which allows for easy calculation of gradients during backpropagation in neural networks.

What is the difference between a loss function and a metric? *

A loss function is used to optimize the parameters of a model during training, while a metric is used to evaluate the performance of the model on the validation or test set.

What is the function to calculate new weights using a learning rate? *

The function to calculate new weights using a learning rate is called the weight update rule, with the most common being: new_weight = old_weight - learning_rate * gradient.

What does the `DataLoader` class do? *

it loads the dataset and prepares it for batch processing in PyTorch.

Write pseudocode showing the basic steps taken in each epoch for SGD. *

```
using numpy:
import numpy as np

np.random.seed(42)
X = np.random.rand(100, 1)
y = 2 + 3 * X + np.random.randn(100, 1)

theta = np.random.randn(2, 1)

lr = 0.01
n_epochs = 1000

for epoch in range(n_epochs):
  for i in range(len(X)):
    xi = X[i:i+1]
    yi = y[i:i+1]
    gradients = 2 * xi.T.dot(xi.dot(theta) - yi)
    theta = theta - lr * gradients
```

Create a function that, if passed two arguments `[1,2,3,4]` and `'abcd'`, returns `[(1, 'a'), (2, 'b'), (3, 'c'), (4, 'd')]`. What is special about that output data structure? *

```
def combinar_as_listas_using_zip(list1, list2):
  return list(zip(list1, list2))
```

"zip" é uma função que combina elementos de iteráveis em tuplas, recebendo recebe dois ou mais iteráveis como argumentos e retorna um iterador que produz um conjunto de tuplas;

What does `view` do in PyTorch? *

iew in PyTorch is used to reshape a tensor to a new shape while preserving its elements.

What are the "bias" parameters in a neural network? Why do we need them? *

The bias parameters in a neural network are additional learnable parameters that are used to shift the activation function to the left or right, allowing the model to better fit the data. They are needed because in many real-world problems, the relationship between the inputs and outputs is not linear, and therefore, the bias terms help to capture the non-linear aspects of the problem.

What does the `@` operator do in Python? *

it is used for matrix multiplication when it is used between two arrays or matrices

What does the `backward` method do? *

performs automatic differentiation to compute and store gradients for tensors involved in a computation graph

Why do we have to zero the gradients? *

Zeroing the gradients before each batch update is necessary because PyTorch accumulates gradients by default, so if we don't clear them, the gradients from the previous batch will be added to the gradients of the current batch, leading to incorrect weight updates and poor model performance.

What information do we have to pass to `Learner`? *

the information we have to pass to Learner in PyTorch is the data, model architecture, optimizer, loss function, and any optional metrics.

Show Python or pseudocode for the basic steps of a training loop. *

```
import torch.nn as nn
import torch.optim as optim

if __name__ == '__main__':

    # Define o modelo
    model = nn.Sequential(
        nn.Linear(784, 256),
        nn.ReLU(),
        nn.Linear(256, 10),
        nn.LogSoftmax(dim=1)
    )

    # Define a função de perda
    loss_func = nn.CrossEntropyLoss()

    # Define o otimizador
    opt = optim.Adam(model.parameters(), lr=0.001)

    # Define o número de épocas
    num_epochs = 10

    # Loop de treinamento
    for epoch in range(num_epochs):
        # Etapa de treinamento
        model.train()
        train_loss = 0
        for xb, yb in train_dl:
            preds = model(xb)
            loss = loss_func(preds, yb)
            loss.backward()
            opt.step()
            opt.zero_grad()
            train_loss += loss.item() * xb.size(0)
        train_loss /= len(train_dl.dataset)

        # Etapa de validação
        model.eval()
        with torch.no_grad():
            valid_loss = 0
            for xb, yb in valid_dl:
                preds = model(xb)
                loss = loss_func(preds, yb)
                valid_loss += loss.item() * xb.size(0)
            valid_loss /= len(valid_dl.dataset)

        # Imprime as perdas do treinamento e da validação
```

    print(f'Epoch [{epoch + 1}/{num_epochs}], Train Loss: {train_loss:.4f}, Valid Loss: {valid_loss:.4f}')

What is "ReLU"? *

it's stands for Rectified Linear Unit and is an activation function commonly used in neural networks.

Draw a plot of a ReLU for values from `-2` to `+2`. *

Relu - BRENO SIL...

What is an "activation function"? *

An activation function is a mathematical function applied to the output of a neural network layer, used to introduce non-linearity into the network and help it learn complex patterns in the data.

What's the difference between `F.relu` and `nn.ReLU`? *

F.relu is a function from the PyTorch functional API for computing the ReLU activation function.
nn.ReLU is a class from the PyTorch module API for creating a ReLU activation function layer.
The difference is that nn.ReLU is a layer that can be added to a neural network model, while F.relu is a function that can be used within a layer or model definition.

The universal approximation theorem shows that any function can be approximated as closely as needed using just one nonlinearity. So why do we normally use more? *

The universal approximation theorem says that we can approximate any function with just one nonlinearity, but in practice we use more than one to get better results, using more non-linearities (activation functions) allows for more complex representations and better performance in practice.