STU FIIT
Umela Inteligencia

“MNIST klasifikator Backpropagation Algoritmus”

Zadanie č. 3a

Made by: Meredov Nazar
Cvičiaci: Bc. Jakub František Setnický
AIS ID: 122092

Meredov Nazar
122092

Anotácia:

Podúloha: MNIST klasifikátor

Cieľom tohto projektu je vyvinúť model neurónovej siete schopný klasifikovať ručne písané číslice pomocou súboru údajov MNIST (Modifikovaný národný inštitút pre štandardy a technológie).
Cieľom je presne identifikovať jednociferné celé čísla od 0 do 9 na základe obrázkov poskytnutých v súbore údajov.
Táto úloha slúži ako referenčný problém v oblasti strojového učenia a počítačového videnia a umožňuje posúdiť rôzne architektúry modelov a techniky trénovania

Podúloha: Backpropagation algoritmus

Táto úloha sa zameriava na implementáciu plne funkčného algoritmu spätného šírenia (**Backpropagation**), ktorý slúži ako základný komponent pri trénovaní neurónových sietí.
Algoritmus spätného šírenia umožňuje neurónovým sieťam učiť sa optimalizáciou nákladovej funkcie spojenej s ich predikciami.
Implementácia musí zahŕňať priamy aj spätný prechod pre rôzne operácie a funkcie spolu s aktualizáciami parametrov potrebnými pre proces trénovania

Meredov Nazar
122092

Opis úlohy:

MNIST klasifikátor - Vytvoriť doprednú neurónovú sieť (viacvrstvový perceptrón) na klasifikáciu ručne písaných číslic z dátového súboru MNIST.

- Dátový súbor obsahuje 60 000 obrázkov na trénovanie a 10 000 obrázkov na testovanie.
- Každý obrázok je v odtieňoch sivej, veľkosti 28 x 28 pixelov a reprezentuje jednu číslicu od 0 do 9.

Backpropagation algoritmus - Implementovať plne funkčný algoritmus backpropagation. Učiť sa pomocou minimalizácie zadanej chybovej funkcie.

- Implementovať doprednú aj spätnú časť pre jednotlivé operátory a funkcie, ako aj aktualizácie parametrov siete.
- Algoritmus overiť natrénovaním jednoduchej neurónovej siete

Meredov Nazar
122092

Obsah:

Meredov Nazar
122092

Ako môžeme štartovať program? MNIST

- **Nainstalovať všetky knižnici zo suboru requirements.txt**

“pip install -r requirements.txt”

```
# For start,you need to install libraries in the requirements.txt
# pip install -r requirements.txt


numpy
pandas
torch
matplotlib
sklearn
```

- **Spustiť klasifikátor MNIST s optimalizátormi Adam, SGD a SGD s momentum**

1) Pred spustenim možete upraviť hyperparametre v subore init_config.py

```
import torch

# Hyperparameters
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
INPUT_SIZE = 784  # 28x28
HIDDEN_SIZES = [256, 128]
NUM_CLASSES = 10
BATCH_SIZE = 64

LEARNING_RATE = 0.001
DROPOUT_RATE = 0.2
NUM_LAYERS = 3

EPOCHS = 50
```

2) Spustime subor main.py v prečinku MNIST

“python MNIST/main.py”

Mame logovane vystupy a konečne vysledky v subore (mnist.log) tak-to:

```
[2024-11-29 14:21:28,511] SGD - Epoch 1: Train Loss = 2.1726, Test Loss = 1.9683, Accuracy = 62.35%
[2024-11-29 14:21:31,735] SGD - Epoch 2: Train Loss = 1.6810, Test Loss = 1.2482, Accuracy = 78.11%
[2024-11-29 14:21:35,280] SGD - Epoch 3: Train Loss = 1.0624, Test Loss = 0.7594, Accuracy = 83.56%
[2024-11-29 14:21:38,948] SGD - Epoch 4: Train Loss = 0.7651, Test Loss = 0.5748, Accuracy = 86.12%
[2024-11-29 14:21:42,928] SGD - Epoch 5: Train Loss = 0.6347, Test Loss = 0.4856, Accuracy = 87.87%
[2024-11-29 14:21:46,957] SGD - Epoch 6: Train Loss = 0.5589, Test Loss = 0.4337, Accuracy = 88.73%
[2024-11-29 14:21:50,835] SGD - Epoch 7: Train Loss = 0.5149, Test Loss = 0.3989, Accuracy = 89.34%
```

Meredov Nazar
122092

```
[2024-11-29 14:24:49,981] SGD_Momentum - Epoch 1: Train Loss = 0.9425, Test Loss = 0.3442, Accuracy = 90.32%
[2024-11-29 14:24:54,436] SGD_Momentum - Epoch 2: Train Loss = 0.3740, Test Loss = 0.2585, Accuracy = 92.21%
[2024-11-29 14:24:58,638] SGD_Momentum - Epoch 3: Train Loss = 0.2979, Test Loss = 0.2131, Accuracy = 93.73%
[2024-11-29 14:25:02,797] SGD_Momentum - Epoch 4: Train Loss = 0.2509, Test Loss = 0.1861, Accuracy = 94.36%
[2024-11-29 14:25:06,889] SGD_Momentum - Epoch 5: Train Loss = 0.2192, Test Loss = 0.1610, Accuracy = 95.23%
[2024-11-29 14:25:10,942] SGD_Momentum - Epoch 6: Train Loss = 0.1933, Test Loss = 0.1442, Accuracy = 95.70%
[2024-11-29 14:25:14,962] SGD_Momentum - Epoch 7: Train Loss = 0.1762, Test Loss = 0.1322, Accuracy = 95.97%
```

```
[2024-11-29 14:28:19,022] Adam - Epoch 1: Train Loss = 0.2751, Test Loss = 0.1173, Accuracy = 96.24%
[2024-11-29 14:28:23,239] Adam - Epoch 2: Train Loss = 0.1302, Test Loss = 0.0883, Accuracy = 97.20%
[2024-11-29 14:28:27,428] Adam - Epoch 3: Train Loss = 0.1000, Test Loss = 0.0833, Accuracy = 97.40%
[2024-11-29 14:28:31,656] Adam - Epoch 4: Train Loss = 0.0817, Test Loss = 0.0736, Accuracy = 97.71%
[2024-11-29 14:28:35,810] Adam - Epoch 5: Train Loss = 0.0734, Test Loss = 0.0669, Accuracy = 97.95%
[2024-11-29 14:28:39,955] Adam - Epoch 6: Train Loss = 0.0640, Test Loss = 0.0796, Accuracy = 97.52%
[2024-11-29 14:28:44,096] Adam - Epoch 7: Train Loss = 0.0588, Test Loss = 0.0689, Accuracy = 98.01%
```

```
[2024-11-29 14:31:46,436]
SGD Results:
[2024-11-29 14:31:46,436] Final Test Accuracy: 95.03%
[2024-11-29 14:31:46,436]
SGD_Momentum Results:
[2024-11-29 14:31:46,436] Final Test Accuracy: 98.25%
[2024-11-29 14:31:46,436]
Adam Results:
[2024-11-29 14:31:46,436] Final Test Accuracy: 98.42%
```

- **Spustiť vyhľadávanie najlepších hyperparametrov pre model zo suboru model_search.py**

“python hypersearch/model_search.py”

Mame logovane vystupy a konečne vysledky tak-to:

```
[2024-11-29 17:38:28,025] Trial 0 finished with value: 0.17405724644295606
[2024-11-29 17:42:07,578] Trial 1 finished with value: 0.07104295122012874
[2024-11-29 17:45:46,357] Trial 2 finished with value: 2.3052947521209717
[2024-11-29 17:49:05,208] Trial 3 finished with value: 2.3122380372065647
[2024-11-29 17:51:52,649] Trial 4 finished with value: 0.08082737735935552
[2024-11-29 17:54:25,887] Trial 5 finished with value: 2.304564023017883
[2024-11-29 17:57:42,022] Trial 6 finished with value: 0.08849324168727579
[2024-11-29 18:01:20,990] Trial 7 finished with value: 0.06811147175743283
[2024-11-29 18:04:18,312] Trial 8 finished with value: 0.15752188919577748
[2024-11-29 18:07:21,454] Trial 9 finished with value: 0.09021864101105165
```

Meredov Nazar
122092

Ako funguje program? MNIST

- **Zloženie neurónovej sieti**

  Zloženie neurónovej siete je **dynamicke**, čo znamená, že jej štruktúra (počet vrstiev, veľkosti vrstiev a pod.) závisí od parametrov, ktoré zadáme pri vytváraní objektu triedy

```python
class MNISTClassifier(nn.Module):  7 usages  new *
    def __init__(self, input_size, hidden_sizes, num_classes, dropout_rate):
        super(MNISTClassifier, self).__init__()
        layers = []
        prev_size = input_size

        # Dynamic create hidden layerss
        for size in hidden_sizes:
            layers.append(nn.Linear(prev_size, size))
            layers.append(nn.ReLU())
            layers.append(nn.Dropout(dropout_rate))
            prev_size = size

        layers.append(nn.Linear(prev_size, num_classes))

        self.network = nn.Sequential(*layers)

    def forward(self, x):  new *
        x = x.view(x.size(0), -1)  # Flatten the image
        return self.network(x)
```

1. Konštruktor (__init__):

- input_size: Veľkosť vstupných dát (784)
- hidden_sizes: Zoznam veľkostí skrytých vrstiev (128 a 64 neuroni a td)
- num_classes: Počet výstupných tried (napr. 10 pre MNIST, kde sú triedy 0-9)
- dropout_rate: Miera dropout pre zníženie preťaženia siete

2. Tvorba vrstiev:

Sieť je vytvorená dynamicky v cykle:

- nn.Linear(prev_size, size): Lineárna vrstva. prev_size je veľkosť predchádzajúcej vrstvy a size je aktuálna veľkosť vrstvy
- nn.ReLU(): aktivačna funkcia ReLU
- nn.Dropout(dropout_rate): Aplikujem dropout

3. Výstupná vrstva:

- Posledná vrstva je lineárna s počtom neurónov zodpovedajúcim počtu tried (num_classes)

4. Funkcia forward:

- Transformuje vstupné dáta z 2D na 1D (flatten) a následne ich prechádza sieťou pomocou self.network

Meredov Nazar
122092

- **Ako stiahneme data?**

Na stiahnutie a prípravu údajov použijeme funkciu **prepare_data**.

Táto funkcia vykonáva nasledujúce kroky:
Transformuje a normalizuje údaje: Definujeme transformačný kanál pomocou funkcie transforms.Compose na prevod obrázkov na tenzory a ich normalizáciu

Nastavenie ciest k údajom: Zadáme cesty pre trénujúce a testovacie údaje

Kontrola existujúcich údajov: Skontrolujeme, či údaje už existujú na zadaných cestách. Ak nie, nastavíme príznak preberania na hodnotu True

Načítať údaje: Na stiahnutie a načítanie súboru údajov MNIST použijeme triedu datasets.MNIST
Údaje sú potom zabalené do objektov DataLoader na jednoduché dávkové spracovanie

```python
def prepare_data(batch_size):  6 usages  new *
    # Transform and normalizate data
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize( mean: (0.1307,), std: (0.3081,))
    ])

    data_path = "/Users/faustyn/Developer/STU_2324/UI/Zadanie3a/data"
    train_path = os.path.join(data_path, 'raw/train-images-idx3-ubyte')
    test_path = os.path.join(data_path, 'raw/t10k-images-idx3-ubyte')

    download = not (os.path.exists(train_path) and os.path.exists(test_path))

    # Load train and test data
    train_dataset = datasets.MNIST(root=data_path, train=True, download=download, transform=transform)
    test_dataset = datasets.MNIST(root=data_path, train=False, download=download, transform=transform)

    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
    test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

    return train_loader, test_loader
```

- **Ako trénujeme model?**

  Na trénovanie modelu použijeme funkciu **train_model**

  Táto funkcia vykonáva nasledujúce kroky:
  Nastaví model do režimu trénovania: model.train()

  Inicializácia straty: Inicializujeme premennú na sledovanie celkovej straty

  Iterovať nad dávkami: Iterujeme po batches údajov v train_loader

  Presunúť údaje do zariadenia: Presunieme údaje (CPU alebo GPU)

  Nastavime tieto parametre

Zero Gradients -> Forward Pass -> Compute Loss -> Spätný priechod -> Aktualizácia váh a strat

```
def train_model(model, train_loader, optimizer, criterion):  4 us
    model.train()
    total_loss = 0

    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(DEVICE), target.to(DEVICE)

        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    return total_loss / len(train_loader)
```

Meredov Nazar
122092

- **Evaluating methoda**

Na vyhodnotenie modelu použijeme funkciu **evaluate_model**

Táto funkcia vykonáva nasledujúce kroky:
Nastavíme model do režimu vyhodnocovania: model.eval()

Inicializujeme metriky: celkovej straty, počtu správnych predpovedí a celkového počtu vzoriek

Zakážeme výpočet gradientu: Vypneme výpočet gradientu pomocou torch.no_grad(), aby sme urýchlili vyhodnotenie
Iterácia nad dávkami: Iterujem nad dávkami údajov v test_loader
Presun údajov do zariadenia: Presunieme údaje a cieľové štítky do určeného zariadenia
Forward Pass: Údaje prechádzame cez model, aby sme získali výstup

Vypočítame stratu
Kumulujem stratu
Vypočítame predpovede
Spočítajte správne predpovede
Vypočítame presnosť

```
def evaluate_model(model, test_loader, criterion):  4 usages  new *
    model.eval()
    total_loss = 0
    correct = 0
    total = 0

    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(DEVICE), target.to(DEVICE)
            output = model(data)
            loss = criterion(output, target)

            total_loss += loss.item()
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()
            total += target.size(0)

    return total_loss / len(test_loader), 100. * correct / total
```

Meredov Nazar
122092

Confusion matrix pre najlepši model MNIST:

- Loudneme najlepši model

```python
def load_best_model():  1 usage  new *
    model = MNISTClassifier(INPUT_SIZE, HIDDEN_SIZES, NUM_CLASSES, dropout_rate=DROPOUT_RATE).to(DEVICE)
    model.load_state_dict(torch.load('mnist_logs/best_model.pth'))
    return model
```

- Cez knižnicu confusion_matrix na testovacom súbore údajov vypočitame confusion matrix

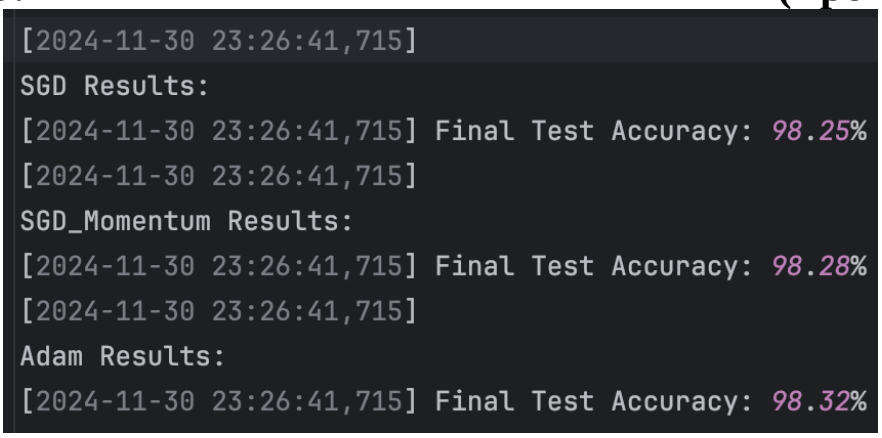
```python
def evaluate_with_confusion_matrix(model, test_loader):  1 usage
    #model eval mode
    model.eval()

    all_preds = []
    all_targets = []

    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(DEVICE), target.to(DEVICE)

            # get model predictions
            output = model(data)
            preds = output.argmax(dim=1, keepdim=True)

            # get predictions and targets
            all_preds.extend(preds.cpu().numpy())
            all_targets.extend(target.cpu().numpy())

    # Compute the cm
    cm = confusion_matrix(all_targets, all_preds)
    return cm
```

- Vizualizujeme confusion matrix pomocou heatmap

```python
def plot_confusion_matrix(cm):  1 usage  new *
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=range(10), yticklabels=range(10))
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title('Confusion Matrix')
    plt.show()
```

Meredov Nazar
122092

RESULTS: (Epohs 500)



Parametre pre optimizatory:

SGD:
'sgd_lr': *0.011667030131725665*
'input_size': *784*
'num_layers': *3*
'hidden_size_0': *256*
'hidden_size_1': *256*
'hidden_size_2': *256*
'num_classes': *10*
'batch_size': *64*
'dropout_rate': *0.1*

SGD_Momentum:
'sgd_momentum_lr': *0.008005276745626849*
'momentum': *0.8987060343120898*
'input_size': *784*
'num_layers': *2*
'hidden_size_0': *256*
'hidden_size_1': *256*
'num_classes': *10*
'batch_size': *256*
‘dropout_rate': *0.2*

Adam:
'adam_lr': *7.85101358317508e-05*
'input_size': *784*
'num_layers': *3*
'hidden_size_0': *256*
'hidden_size_1': *128*
'hidden_size_2': *128*
'num_classes': *10*
'batch_size': *64*
'dropout_rate': *0.2*

Meredov Nazar
122092

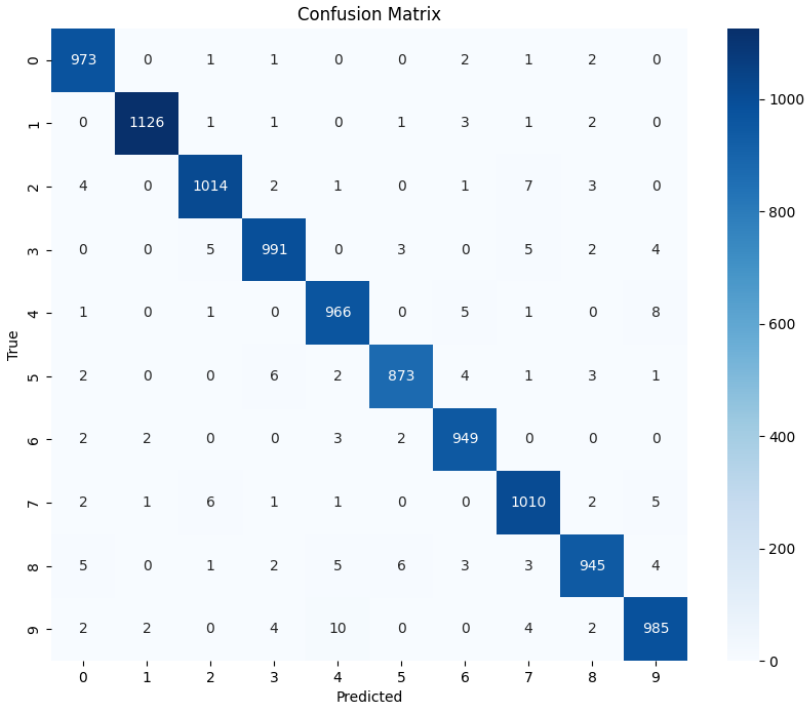
Stručne zhodnotenie MNIST

V priebehu svojej analýzy a práce som skúmal 3 algoritmy na optimalizáciu neurónových sietí a výber hyperparametrov.

Pri hľadaní najlepších hyperparametrov pre každý z algoritmov som použil bibliotéku **optuna** a získal som perfekte hyperparametre (strana hore hneť)

Aj z 3 optimiyatorov bol najlepší – Adam (98.32%)

Taktiež som pre prehľadnosť odhadol maticu zámeny pre každé číslo a algoritmus s týmito výsledkami:



Celkovo som s výsledkami úplne spokojný a mame presnosťou väčšou ako 97%

Meredov Nazar
122092

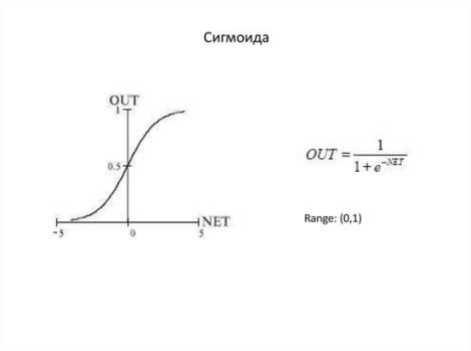
Backpropagation algoritmus forward/backward

(source https://www.geeksforgeeks.org/gradient-descent-algorithm-and-its-variants/)

SigmoidMethod

Metoda implementuje sigmoidnú aktivačnú funkciu a jej deriváciu na spätné šírenie chyby
Sigmoid sa používa na zavedenie **nelinearity** do modelu a transformáciu vstupnej hodnoty do rozsahu od 0 do 1

Teoria:



$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\frac{d(\sigma(x))}{dx} = \frac{0 * (1+e^{-x}) - (1) * (e^{-x} * (-1))}{(1+e^{-x})^2}$$

$$\frac{d(\sigma(x))}{dx} = \frac{(e^{-x})}{(1+e^{-x})^2} = \frac{1\text{-}1+(e^{-x})}{(1+e^{-x})^2} = \frac{1+e^{-x}}{(1+e^{-x})^2} - \frac{1}{(1+e^{-x})^2}$$

$$\frac{d(\sigma(x))}{dx} = \frac{1}{1+e^{-x}} * \left(1 - \frac{1}{1+e^{-x}}\right) = \sigma(x)(1-\sigma(x))$$

Implementacia:

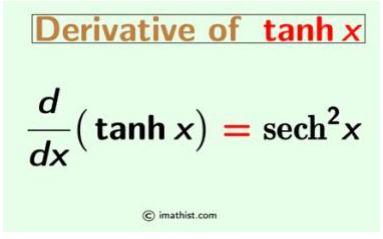
```
def forward(self, x):  4 usages (2 dynamic)  Nazar Faustyn *
    self.x = x
    return 1/(1+np.exp(-x))

def backward(self, gradient):  3 usages (3 dynamic)  Nazar Faustyn *
    sigmoid_deriv = self.forward(self.x)*(1-self.forward(self.x))
    return gradient * sigmoid_deriv
```

TanhMethod

Trieda TanhMethod implementuje hyperbolickú aktivačnú funkciu tanh a jej deriváciu na spätné šírenie chyby.
Funkcia sa používa aj na pridanie **nelinearity** a **normalizáciu** výstupných hodnôt do rozsahu -1 až 1.

Teoria:

$$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$



Meredov Nazar
122092

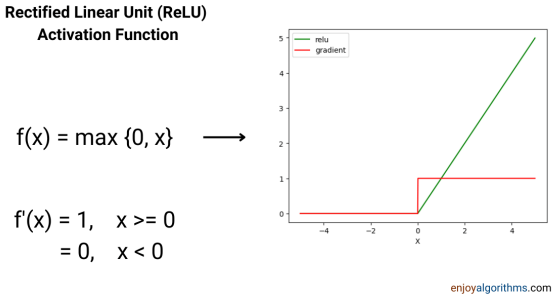
Implementacia:

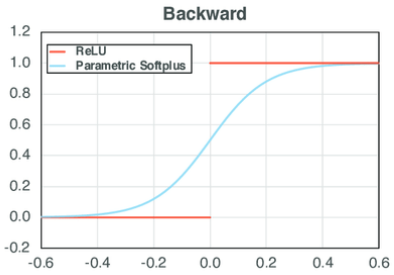
```
def forward(self, x):  2 usages (2 dynamic)  new *
    self.output = np.tanh(x)
    return self.output

def backward(self, d_output):  3 usages (3 dynamic)  new *
    return d_output * (1 - self.output ** 2)
```

ReLUMethod

Metóda ReLUMethod implementuje aktivačnú funkciu „Rektifikovaná lineárna jednotka“ a jej deriváciu na spätné šírenie chyby

Teoria:





Implementacia:

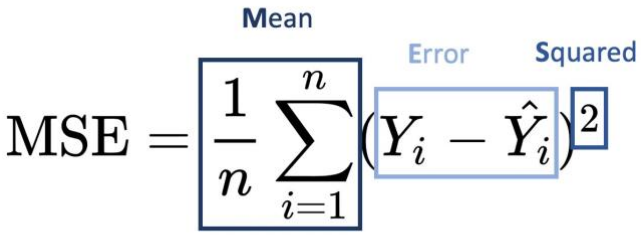
```
def forward(self, x):  2 usages (2 dynamic)  Nazar Faustyn *
    self.x = x
    return np.maximum(0, x) # return 0 or x

def backward(self, gradient):  3 usages (3 dynamic)  Nazar Faustyn
    return gradient * (self.x > 0)
```

Meredov Nazar
122092

LossMethod

LossMethod implementuje výpočet funkcie MSE (Mean Squared Error), ktorú budeme v našej úlohe používať na meranie rozdielu medzi predpovedanými hodnotami (predicted) a skutočnými hodnotami (target)

Forward:
vypočíta strednú kvadratickú chybu (MSE) medzi predpovedanými hodnotami a skutočnými cieľovými hodnotami



Backward:
Backward vráti gradient stratovej funkcie MSE nad predpovedanými hodnotami. Tento gradient použijeme na aktualizáciu váh pre backpropagation

Implementacia:

```
def forward(self, predicted, target):  3 usages (2 dynamic)  new *
    self.predicted = predicted
    self.target = target
    return np.mean((predicted - target) ** 2)

def backward(self):  4 usages (3 dynamic)  new *
    return (2 * (self.predicted - self.target))/self.predicted.size
```

Meredov Nazar
122092

LayerContainer

LayerContainer predstavuje kontajner na správu postupnosti vrstiev neurónovej siete a vykonávanie operácií prechodu dopredu a dozadu cez nu

layers - zoznam vrstiev neurónovej siete (každá vrstva má dopredné a spätné metódy)
Vrstvy sa odovzdávajú v poradí, v akom sa vykonávajú

```
def __init__(self, layers):  Nazar Faustyn
    self.layers = layers
```

Metóda „forward“ postupuje vstupné údaje postupne cez všetky vrstvy
Metóda „backward“ vykonáva spätné šírenie gradientu cez všetky vrstvy v **opačnom poradí**

```
def forward(self, x):  2 usages (2 dynamic)  Nazar Faustyn
    for layer in self.layers:
        x = layer.forward(x)
    return x

def backward(self, gradient):  3 usages (3 dynamic)  Nazar Faustyn
    for layer in reversed(self.layers):
        gradient = layer.backward(gradient)
```

LinearLayer

LinearLayer je lineárna vrstva neurónovej siete, ktorá vykonáva lineárnu transformáciu vstupných údajov pomocou váh a skreslení (bias) a podporuje aj mechanizmus momentového učenia na urýchlenie konvergencie gradientného zostupu.

```
def __init__(self, input_size, output_size, learning_rate=0.1, momentum=0.0):  Nazar Faustyn *
    self.weights = np.random.randn(input_size, output_size) * 0.01
    self.biases = np.zeros((1, output_size))
    self.learning_rate = learning_rate
    self.momentum = momentum

    self.weight_momentum = np.zeros_like(self.weights)
    self.bias_momentum = np.zeros_like(self.biases)

    self.last_input = None
    self.last_output = None
```

Meredov Nazar
122092

Metóda forward vykonáva lineárnu transformáciu vstupných údajov pomocou vzorca y=Wx+b

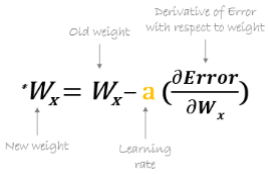
```
def forward(self, x):  2 usages (2 dynamic)  Nazar Faustyn
    self.last_input = x
    self.last_output = np.dot(x, self.weights) + self.biases
    return self.last_output
```

Metóda backward počíta gradienty a aktualizuje váhy a posuny

- **input_gradient:** $y_p = xW_T + b$

Chybový gradient odovzdaný predchádzajúcej vrstve

- **weights_gradient:**




```
def backward(self, gradient):  3 usages (3 dynamic)  Nazar Faustyn *
    input_gradient = np.dot(gradient, self.weights.T)
    weights_gradient = np.dot(self.last_input.T, gradient)
    biases_gradient = np.sum(gradient, axis=0, keepdims=True)

    if self.momentum > 0:
        self.weight_momentum = self.momentum * self.weight_momentum - self.learning_rate * weights_gradient
        self.bias_momentum = self.momentum * self.bias_momentum - self.learning_rate * biases_gradient

        self.weights += self.weight_momentum
        self.biases += self.bias_momentum
    else:
        self.weights -= self.learning_rate * weights_gradient
        self.biases -= self.learning_rate * biases_gradient

    return input_gradient
```

Gradient chyby podľa váh

- **biases_gradient:**

$$\sum_{i=m}^{n} a_i$$

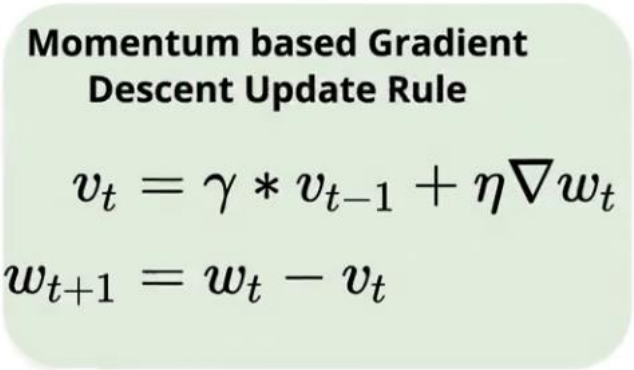
m – počet gradientov
a - gradient

Súčet gradientov za všetky príklady v dávke Súčet gradientov za všetky príklady v dávke

- **Aktualizácia váh a posunov:**
- IF **momentum** > 0:

μ - koeficient hybnosti
η - miera učenia

- IF **momentum** = 0, váhy a posuny sa aktualizujú pomocou obvyklého gradientného zostupu



Meredov Nazar
122092

Metoda trenerovania modelu

Model trénujeme pomocou metódy spätného šírenia s gradientným zostupom

loss_fn = LossMethod() - vytvorí sa objekt stratovej funkcie (MSE)

Vykoná trénovanie modelu pre daný počet epoch (epoch)

prediction = model.forward(X[i].reshape(1, -1))
Každá vstupná vzorka X[i] sa preženie cez model a vypočíta sa predikcia

loss = loss_fn.forward(prediction, y[i].reshape(1, -1))
total_loss += loss
Vypočíta sa chyba medzi predikciou modelu predikcie a skutočnou hodnotou y[i] a pripočíta sa k celkovej chybe

gradient = loss_fn.backward()
model.backward(gradient)
Gradient stratovej funkcie sa odovzdá modelu, ktorý aktualizuje svoje váhy a posuny

avg_loss = total_loss / len(X)
losses.append(avg_loss)
Priemerná chyba sa uloží pre ďalšii analýz

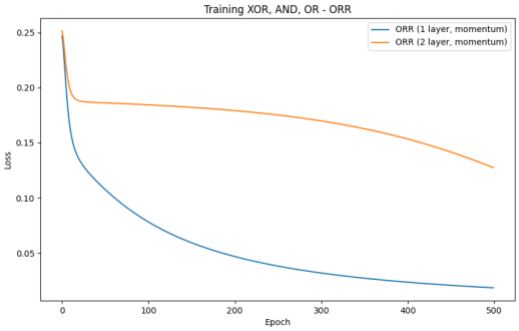
Po ukončení trénovania metóda vráti zoznam priemerných chýb v každej epoche

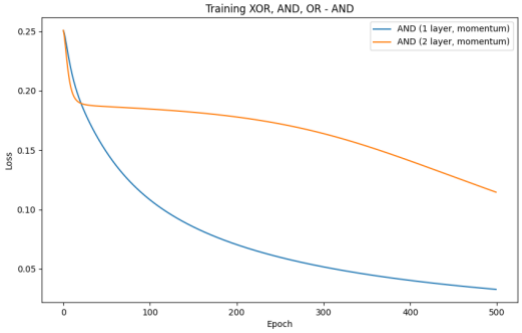
```python
def train_model(X, y, model, epochs, verbose=True):  1 usage  Nazar F
    loss_fn = LossMethod()
    losses = []

    for epoch in range(epochs):
        total_loss = 0

        for i in range(len(X)):
            # forward pass
            prediction = model.forward(X[i].reshape(1, -1))

            # loss calculation
            loss = loss_fn.forward(prediction, y[i].reshape(1, -1))
            total_loss += loss

            # backpropagation
            gradient = loss_fn.backward()
            model.backward(gradient)

        avg_loss = total_loss / len(X)
        losses.append(avg_loss)

        if verbose and epoch % 50 == 0:
            print(f"Epoch {epoch}, Loss: {avg_loss}")

    return losses
```

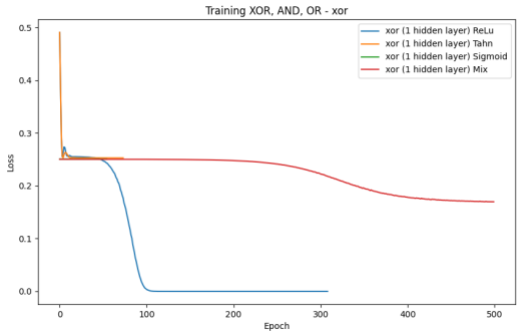
Meredov Nazar
122092

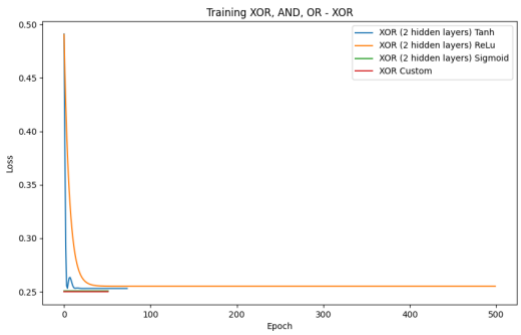
Vypysy a testovanie

Počas analýzu najlepšieho modelu z viróznymi parametrami dostal som také výsledky:










```
[2024-12-02 01:19:16,362] Train: xor (1 hidden layer) Tahn
[2024-12-02 01:19:16,363] Epoch 0, Loss: 0.4907352326437151
[2024-12-02 01:19:16,386] Epoch 50, Loss: 0.2529748968373269
[2024-12-02 01:19:16,393] Early stopping at epoch 73
```

```
[2024-12-02 01:19:16,394] Train: XOR (2 hidden layers) Tanh
[2024-12-02 01:19:16,395] Epoch 0, Loss: 0.4909280462796399
[2024-12-02 01:19:16,410] Epoch 50, Loss: 0.25296944374428426
[2024-12-02 01:19:16,417] Early stopping at epoch 73
```

```
[2024-12-02 01:19:16,265] Train: xor (1 hidden layer) ReLu
[2024-12-02 01:19:16,266] Epoch 0, Loss: 0.4908523352629769
[2024-12-02 01:19:16,278] Epoch 50, Loss: 0.23708566996228608
[2024-12-02 01:19:16,295] Epoch 100, Loss: 0.16978600747359995
[2024-12-02 01:19:16,307] Epoch 150, Loss: 0.03661203509697997
[2024-12-02 01:19:16,319] Epoch 200, Loss: 1.926538679311998e-11
[2024-12-02 01:19:16,324] Epoch 250, Loss: 1.531181667015596e-20
[2024-12-02 01:19:16,335] Epoch 300, Loss: 2.1477970739806454e-30
[2024-12-02 01:19:16,357] Epoch 350, Loss: 0.0
[2024-12-02 01:19:16,362] Early stopping at epoch 362
```

```
[2024-12-02 01:19:16,417] Train: XOR (2 hidden layers) ReLu
[2024-12-02 01:19:16,418] Epoch 0, Loss: 0.5000996266547234
[2024-12-02 01:19:16,434] Epoch 50, Loss: 0.2551289119425127
[2024-12-02 01:19:16,441] Epoch 100, Loss: 0.2550499716149226
[2024-12-02 01:19:16,458] Epoch 150, Loss: 0.2550499109682498
[2024-12-02 01:19:16,465] Epoch 200, Loss: 0.25504987438803645
[2024-12-02 01:19:16,486] Epoch 250, Loss: 0.25504983750868715
[2024-12-02 01:19:16,495] Epoch 300, Loss: 0.2550498002771683
[2024-12-02 01:19:16,503] Epoch 350, Loss: 0.25504976265688495
[2024-12-02 01:19:16,510] Epoch 400, Loss: 0.2550497245778086
[2024-12-02 01:19:16,519] Epoch 450, Loss: 0.25504968599988836
```

```
[2024-12-02 01:19:16,527] Train: xor (1 hidden layer) Sigmoid
[2024-12-02 01:19:16,527] Epoch 0, Loss: 0.25065440299996045
[2024-12-02 01:19:16,532] Epoch 50, Loss: 0.2506572865404335
[2024-12-02 01:19:16,532] Early stopping at epoch 51
```

```
[2024-12-02 01:19:16,532] Train: XOR (2 hidden layers) Sigmoid
[2024-12-02 01:19:16,533] Epoch 0, Loss: 0.25065791925111425
[2024-12-02 01:19:16,550] Epoch 50, Loss: 0.2506595232407921
[2024-12-02 01:19:16,551] Early stopping at epoch 51
```

Meredov Nazar
122092

Vysledky:

1. XOR (1 skrytá vrstva):

- **Aktivácia ReLU:**
  - Výrazné zlepšenie straty (Loss) počas tréningu, až po epochu 350, kde sa strata stabilizuje na 0, čo naznačuje perfektné naučenie modelu.
  - Early stopping sa aktivovalo pri 362. epoche, čo naznačuje dostatočnú kapacitu modelu pre riešenie problému XOR.
- **Aktivácia Tanh:**
  - Strata sa zlepšila už v 73. epoche, kedy došlo k early stoppingu.
  - Model sa naučil rýchlo, čo ukazuje efektívnosť funkcie Tanh pre tento problém.
- **Aktivácia Sigmoid:**
  - Strata zostáva takmer konštantná, čo naznačuje, že Sigmoid je menej efektívny pre riešenie XOR v tejto konfigurácii. Early stopping nastalo už pri 51. epoche.
- **Mix aktivácií:**
  - Pozvoľné zlepšovanie straty, ale zlepšenie je viditeľné až po dlhšom tréningu (350+ epoch). Výsledná strata ukazuje, že model sa síce zlepšuje, ale nie tak efektívne ako pri ReLU alebo Tanh.

Meredov Nazar
122092

---

2. XOR (2 skryté vrstvy):

- **Aktivácia Tanh:**
  - Podobne ako pri 1 vrstve, early stopping nastalo pri 73. epoche s porovnateľným výkonom.
- **Aktivácia ReLU:**
  - Model dosiahol minimálne zlepšenie straty a pretrváva stabilná strata okolo 0,255, čo naznačuje nedostatočné zlepšenie s touto architektúrou.

---

3. AND a OR:

- **AND (1 vrstva, momentum):**
  - Strata postupne klesá počas celého tréningu, pričom model dosiahol významné zlepšenie, a to z 0,25 na približne 0,036.
  - Tento model sa dobre prispôsobil problému.
- **AND (2 vrstvy, momentum):**
  - Model vykazuje pomalšie zlepšovanie, no postupne sa adaptuje na problém, hoci o niečo menej efektívne než model s 1 vrstvou.
- **OR (1 vrstva, momentum):**
  - Výrazné zlepšenie straty z 0,252 na približne 0,020 počas 450 epoch.
  - Model rýchlo konverguje a ukazuje dobré prispôsobenie.
- **OR (2 vrstvy, momentum):**
  - Zlepšenie je podobné ako pri 1 vrstve, ale konvergencia je pomalšia.

Stručne zhodnotenie Backpropagation

- **ReLU** je mimoriadne efektívny pre problémy ako XOR s 1 skrytou vrstvou, čo vedie k perfektnej presnosti

- **Tanh** je univerzálna a rýchlo konvergujúca aktivácia, ktorá funguje dobre aj pre XOR

- **Sigmoid** je menej efektívny pre riešenie XOR a výsledky ukazujú, že model sa ťažko optimalizuje

- **Problémy AND a OR** boli úspešne vyriešené pomocou jednej alebo dvoch vrstiev s využitím momentu. Jedna vrstva sa však ukázala ako efektívnejšia
- Modely s 2 vrstvami nepreukázali výrazné zlepšenie oproti modelom s 1 vrstvou, čo naznačuje, že pre tieto jednoduché problémy nemusí byť potrebná zvýšená komplexita

Meredov Nazar
122092

Zhodnotenie

1. **Pre úlohu klasifikácie MNIST datasetu** je optimálnym riešením použitie algoritmu ADAM, ktorý dosiahol vysokú presnosť nad 97% a rýchlu konvergenciu v porovnaní s SGD a SGD s momentum

2. **Backpropagation algoritmus** bol uspešne implementovany na klasických logických problémoch XOR, AND a OR

3. Použitie optimalizačných techník ako **moment** a nastavenie **vhodnej rýchlosti učenia** má významný vplyv na rýchlosť a kvalitu učenia neurónových sietí

Obe podúlohy boli úspešne splnené, pričom implementované modely a algoritmy spĺňajú požiadavky zadania a dosahujú očakávané výsledky

Meredov Nazar
122092