Project 2

A comparison between Bayesian Neural Network and Classical Neural Network

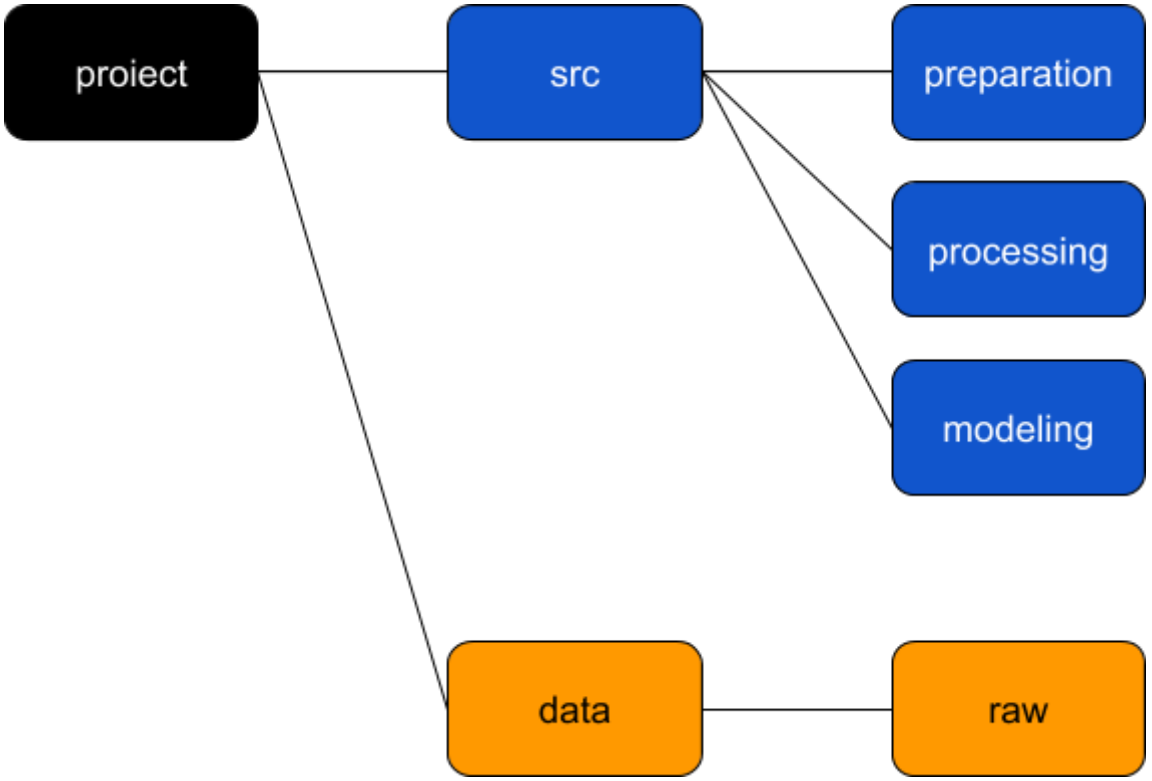
Adrian Ispas
Grupa 507 - Master IA

Informaţii generale

Prima variantă a proiectului (cea care nu conţine extra-urile asociate) este descrisă de-a lungul acestui document până la capitolul **Extra.**

Implementarea până în acest punct se poate găsi pe github cu tagul asociat **v1.0.0** la următorul link: https://github.com/adilspas/Bayesian-Neural-Network/tree/v1.0.0

Descriere structură proiect



1.1 src

Înglobează fişierele python în care se realizează extragerea datelor, procesarea, descrierea modelului şi antrenarea lui.

1.1.1 preparation

Citirea datelor stocate în folderul **/data/raw** pentru reţeaua de clasificare multi class şi generarea de date pentru reţeaua de clasificare binară.

1.1.2 processing

Nu este cazul.

1.1.3 modeling

Definirea şi antrenarea modelului pentru multi class (**BNNm**) şi pentru reţeaua binară (**BNNb**).

1.2 data

Stocarea datelor pentru clasificarea multi class.

1.2.1 raw

Stocarea offline a dataset-ului **MNIST** (http://yann.lecun.com/exdb/mnist/).

Descrierea implementării

2.1 Implementarea modelului

În ceea ce priveşte implementarea modelelor putem evidenţia pentru fiecare în parte următoarele definiţii de variabile.

2.1.1 Definire multi class Bayesian Neural Network

- **X**

```
self.x = tf.placeholder(tf.float32, [self.N, self.D])
```

Definită ca un placeholder pentru a stoca batch-uri de dimensiune **N** din data set. **D**-ul reprezintă numărul de feature-uri pentru fiecare imagine din dataset.

- **W şi B**

```
self.w = Normal(loc=tf.zeros([self.D, self.K]), scale=tf.ones([self.D, self.K]))

self.b = Normal(loc=tf.zeros(self.K), scale=tf.ones(self.K))
```

În estimarea funcţiei densitate (inferenţa variaţională) avem nevoie de parametrii **w** şi **b** pe care îi modelăm cu o distribuţie cunoscută, în cazul nostru alegem o Gaussiană.

Odată definite placeholderele **w** şi **b** putem defini şi **qw** şi **qb** ca fiind distribuţii normale cu media dată de variabile normale şi deviaţia dată de funcţia de activare **softplus**.

```
self.qw = Normal(loc=tf.Variable(tf.random_normal([self.D, self.K])),
                 scale=tf.nn.softplus(tf.Variable(tf.random_normal([self.D,
self.K]))))

self.qb = Normal(loc=tf.Variable(tf.random_normal([self.K])),
                 scale=tf.nn.softplus(tf.Variable(tf.random_normal([self.K]))))
```

- **Y**

```
self.y = Categorical(tf.matmul(self.x, self.w) + self.b)
```

Definit ca fiind o probabilitate categorică determinată de formula clasică **X * W + B**. Definim de asemenea și un placeholder pentru **y**.

```
self.y_ph = tf.placeholder(tf.int32, [self.N])
```

- **Inference**

```
self.inference = ed.KLqp({self.w: self.qw, self.b: self.qb}, data={self.y:
self.y_ph})
```

Inferența variațională este definită prin **KL divergence** care are scopul de a optimiza parametrii **qw** și **qb**.

2.1.2 Definire binary Bayesian Neural Network

- **X**

```
self.x = tf.placeholder(tf.float32, [self.N, self.D])
```

Definită ca un placeholder pentru a stoca batch-uri de dimensiune **N** din data set. **D**-ul reprezintă numărul de feature-uri pentru fiecare imagine din dataset.

- **W și B**

```
self.w = Normal(loc=tf.zeros([self.D]), scale=tf.ones([self.D]))

self.b = Normal(loc=tf.zeros(), scale=tf.ones())
```

În estimarea funcției densitate (inferența variațională) avem nevoie de parametrii **w** și **b** pe care îi modelăm cu o distribuție cunoscută, în cazul nostru alegem o Gaussiană.

Odată definite placeholderele **w** și **b** putem defini și **qw** și **qb** ca fiind distribuții normale cu media dată de variabile normale și deviația dată de funcția de activare **softplus**.

```
self.qw = Normal(loc=tf.Variable(tf.random_normal([self.D])),
                 scale=tf.nn.softplus(tf.Variable(tf.random_normal([self.D]))))

self.qb = Normal(loc=tf.Variable(tf.random_normal([])),
                 scale=tf.nn.softplus(tf.Variable(tf.random_normal([]))))
```

Spre deosebire de definirea variabilelor în clasificarea multiclasă, aici putem renunţa la definirea dimensiunilor variabilelor **w, b, qw, qb** şi în funcţie de **K**, numărul de clase, astfel definim doar în funcţie de numărul de feture-uri, în cazul nostru două (coordonatele x şi y).

- **Y**

```
self.y = Bernoulli(tf.matmul(self.x, self.w) + self.b)
```

Definit ca fiind o probabilitate Bernoulli determinată de formula clasică **X * W + B**. Definim **y**-ul ca fiind o distribuţie Bernoulli în comparaţie cu una Categorical folosită pentru clasificarea multiclasă, deoarece **y**-ul va lua valori doar din mulţimea **{0, 1}**.

Definim de asemenea şi un placeholder pentru **y**.

```
self.y_ph = tf.placeholder(tf.int32, [self.N])
```

- **Inference**

```
self.inference = ed.KLqp({self.w: self.qw, self.b: self.qb}, data={self.y:
self.y_ph})
```

Inferenţa variaţională este definită prin **KL divergence** care are scopul de a optimiza parametrii **qw** şi **qb**.

2.1.3 Antrenare

Pentru antrenare putem folosi funcţia

train(iterations=5000)

implementată în model. În cazul clasificării multiclasă **train** este definită după cum urmează:

```
def train(self, iterations=5000):
   self.__initialize__(self.data.train.num_examples, iterations)

   for _ in range(self.inference.n_iter):
       x_batch, y_batch = self.data.train.next_batch(self.N)

       y_batch = np.argmax(y_batch, axis=1)
       info_dict = self.inference.update(feed_dict={self.x: x_batch, self.y_ph:
y_batch})

       self.inference.print_progress(info_dict)
```

Pentru fiecare iteraţie din procesul de antrenare luăm un nou batch de exemple de dimensiune **N** cu care actualizăm inferenţa, iar la final afişăm progresul.

Similar şi pentru antrenarea modelului de clasificare binară, cu excepţia faptului ca nu mai avem nevoie de o procesare a datelor pentru **y_batch**:

```
def train(self, iterations=5000):
   self.__initialize__(iterations * self.N, iterations)

   for index in range(self.inference.n_iter):
       x_batch, y_batch = Reader.next_make_moons_batch(self.noise, self.N)
       info_dict = self.inference.update(feed_dict={self.x: x_batch, self.y_ph:
y_batch})

       self.inference.print_progress(info_dict)
```

Funcţia

__initialize__(number_of_examples, iterations)

are rolul de a seta numărul de exemple şi de iteraţii pentru inferenţă.

Funcţia

next_make_moons_batch(noise, batch_size)

generează pentru clasificarea binară un nou batch de exemple de dimensiune **batch_size** cu o perturbare de **noise**.

2.1.4 Evaluarea

În evaluarea modelelor putem folosi funcţia

evaluating(number_of_samples)

definită pentru modelul multiclasă după cum urmează

```
def evaluating(self, number_of_samples):
   self.__load_test_data__()
   self.w_values = []

   for _ in range(number_of_samples):
       w_samp = self.qw.sample()
```

```
        b_samp = self.qb.sample()

        self.w_values.append(w_samp.eval())

        prob = tf.nn.softmax(tf.matmul(self.x_test, w_samp) + b_samp)
        self.probabilities.append(prob.eval())

    self.w_values = np.reshape(self.w_values, [-1])

    y_pred = np.argmax(np.mean(self.probabilities, axis=0), axis=1)
    print("Accuracy in predicting the test data = ", (y_pred ==
self.y_test).mean() * 100)
```

Pentru fiecare iteraţie generăm valori pentru **w** şi **b** conform distribuţiilor acestora, iar cu valorile din **x_test** compunem probabilitatea pe ultimul layer cu ajutorul funcţiei **softmax.**

La final afișăm acuratețea modelului ca fiind media acurateții pentru fiecare batch de sample-uri.

Definiţie similară şi pentru clasificarea binară

```
def evaluating(self, number_of_samples):
    self.w_values = []
    self.accuracy = []

    for _ in range(number_of_samples):
        x_test, y_test = Reader.next_make_moons_batch(self.noise, self.N)
        x_test = tf.cast(x_test, tf.float32)

        w_samp = self.qw.sample()
        b_samp = self.qb.sample()

        self.w_values.append(w_samp.eval())

        prob = tf.nn.sigmoid(ed.dot(x_test, w_samp) + b_samp)

        y_pred = prob.eval()
        y_pred[y_pred > 0.5] = 1
        y_pred[y_pred <= 0.5] = 0

        self.accuracy.append(accuracy_score(y_test, y_pred))

    self.w_values = np.reshape(self.w_values, [-1])

    print("Accuracy in predicting the test data = ", np.mean(self.accuracy) *
100)
```

cu excepţia faptului ca probabilitatea este compusă pe ultimul layer cu ajutorul funcţiei **sigmoid**.

2.1.5 Vizualizarea rezultatelor

Vizualizare distribuţie acurateţe

Pentru vizualizarea distribuției acurateței putem folosi funcţia

plot_accuracy()

definită după cum urmează

```
def plot_accuracy(self):
    #  pentru clasificarea multiclasă compunem direct în funcție acuratețea
    #  accuracy_test = []
    # for prob in self.probabilities:
    #     y_pred = np.argmax(prob, axis=1).astype(np.float32)
    #     accuracy = (y_pred == self.y_test).mean() * 100
    #     accuracy_test.append(accuracy)
    #
    # plt.hist(accuracy_test)

    plt.hist(self.accuracy)

    plt.title("Histogram of prediction accuracies in the MNIST test data")
    plt.xlabel("Accuracy")
    plt.ylabel("Frequency")

    plt.show()
```

Vizualizare distribuţie W-uri

Pentru vizualizarea distribuției de W-uri putem folosi funcţia

plot_w()

definită după cum urmează

```
def plot_w(self):
   plt.hist(self.w_values)

   plt.title("Histogram of W in the MNIST test data")
   plt.xlabel("W samples")
   plt.ylabel("Frequency")

   plt.show()
```

Flow-ul

În continuare vom descrie flow-ul aplicației și cum ajungem la rezultatele dorite.

Pasul 1: Citim dataset-ul în prealabil, dacă este cazul - doar pentru clasificarea multiclasă, cu funcția

read(classification_type)

din clasa **Reader**, iar **classification_type** poate avea valorea **MULTI_CLASS** pentru care este implementată funcția de citire.

Pentru clasificarea binară citirea datelor se face în timpul antrenării sau testării cu funcția

next_make_moons_batch(noise, batch_size)

descrisă anterior.

Pasul 2: Definim parametrii modelului

```
batch_size = 100
number_of_features = 784 # sau 2 pentru clasificarea binară
number_of_classes = 10 # sau 2 pentru clasificarea binară

iterations = 10000
samples_1 = 100 # pe câte exemple vrem să rulăm comparația
samples_2 = 1
```

Pasul 3: instanțiem un model

```
# pentru clasificarea multiclasă
model = BNNm(batch_size, number_of_features, number_of_classes, dataset)
```

```
# pentru clasificarea binară
model = BNNb(batch_size, number_of_features, number_of_classes)
```

Pasul 4: antrenăm

```
model.train(iterations)
```

Pasul 5: evaluăm pentru un număr de sample-uri

```
model.evaluating(samples_1)

model.evaluating(samples_2)
```

Pasul 6: plotăm rezultatele

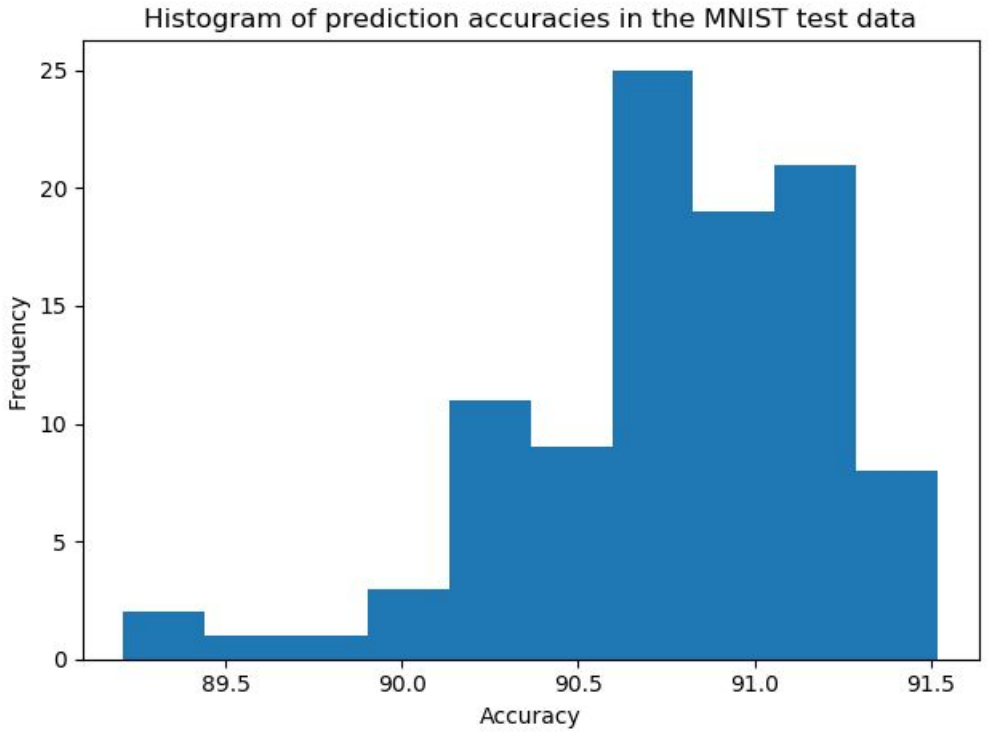
```
model.plot_accuracy()
model.plot_w()
```

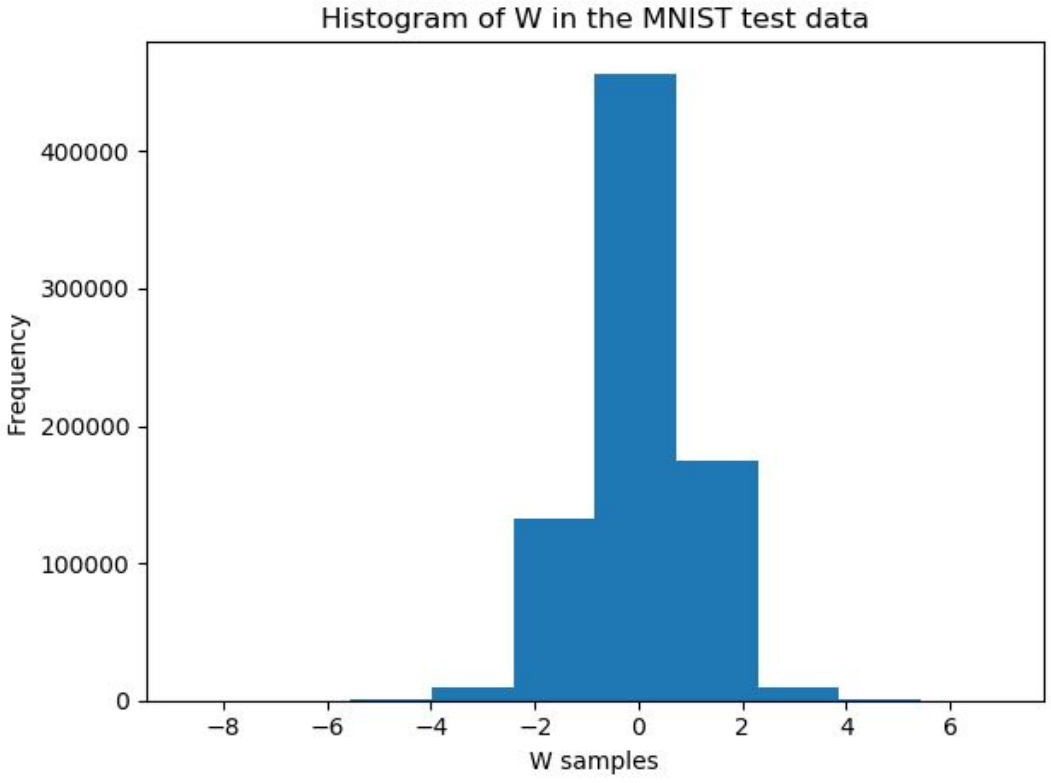
Rezultate obținute

4.1 Clasificarea multiclasă

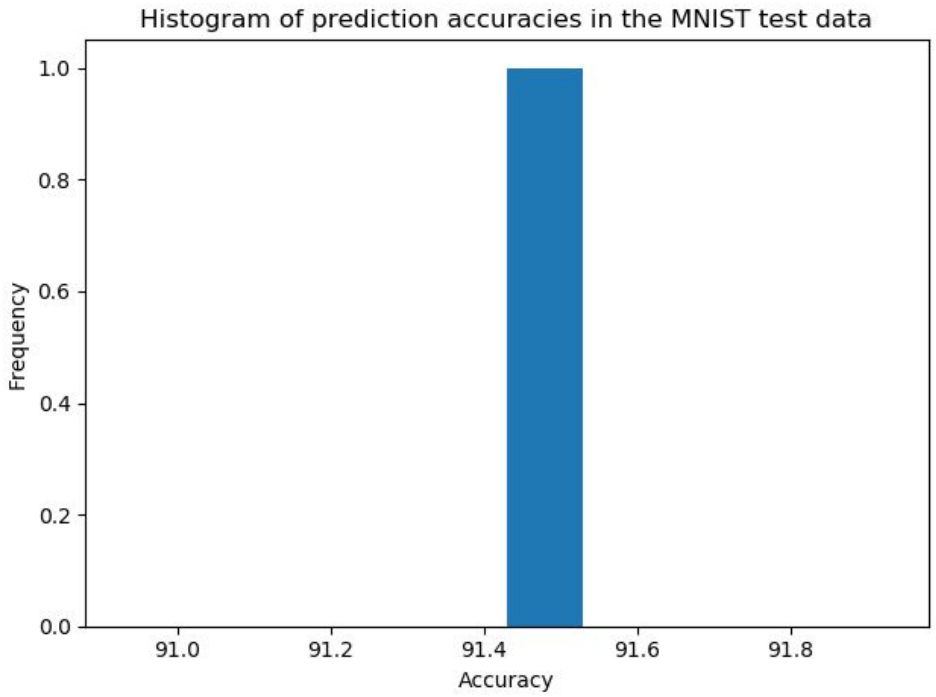
```
=== Multi class classification ===

Extracting ../data/raw/train-images-idx3-ubyte.gz
Extracting ../data/raw/train-labels-idx1-ubyte.gz
Extracting ../data/raw/t10k-images-idx3-ubyte.gz
Extracting ../data/raw/t10k-labels-idx1-ubyte.gz


Training ...

Device mapping:
/job:localhost/replica:0/task:0/gpu:0 -> device: 0, name: GeForce GTX 950M, pci
bus id: 0000:01:00.0
10000/10000 [100%] ████████████████████████████████ Elapsed: 41s |
Loss: 26230.666


Evaluation 100 sample(s)

Accuracy in predicting the test data =  92.54


Evaluation 1 sample(s)

Accuracy in predicting the test data =  91.43
```

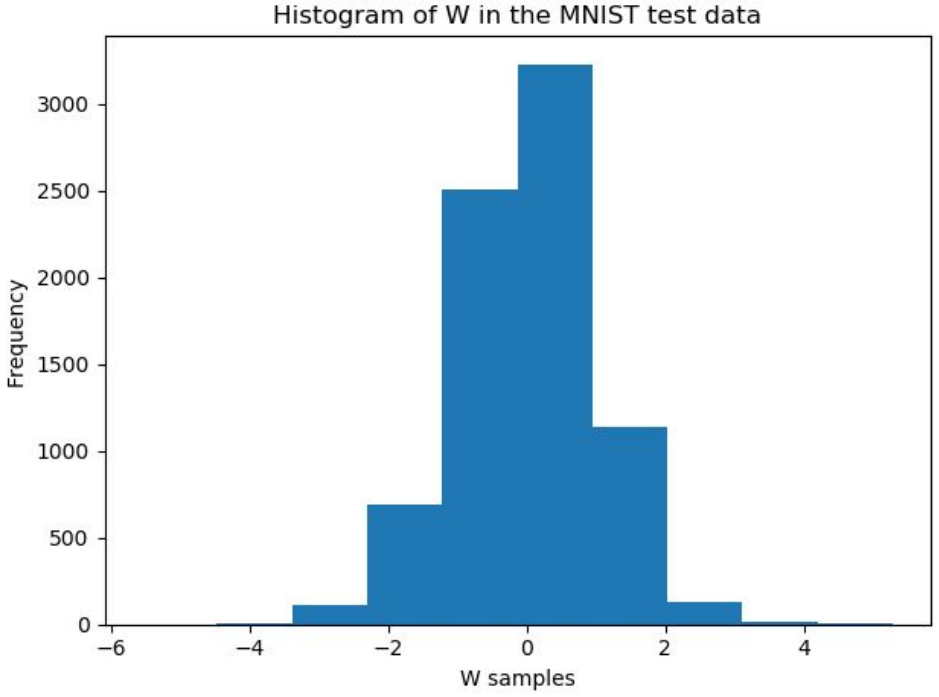
100 samples


Histogram of prediction accuracies in the MNIST test data
Frequency
Accuracy
0
5
10
15
20
25
89.5
90.0
90.5
91.0
91.5


Histogram of W in the MNIST test data
Frequency
W samples
0
100000
200000
300000
400000
−8
−6
−4
−2
0
2
4
6

1 sample

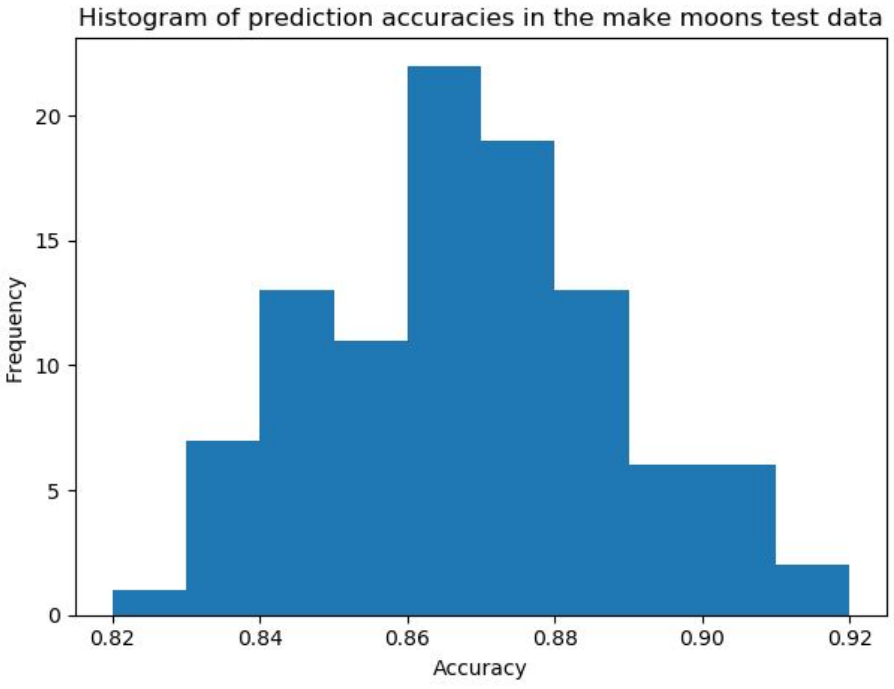

Histogram of prediction accuracies in the MNIST test data
Frequency
Accuracy
0.0
0.2
0.4
0.6
0.8
1.0
91.0
91.2
91.4
91.6
91.8

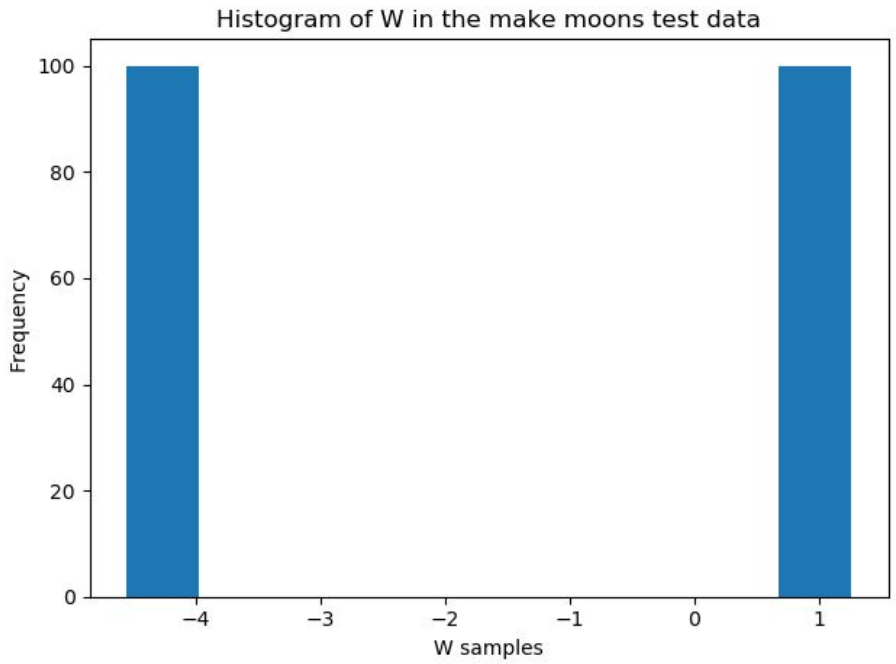

Histogram of W in the MNIST test data
Frequency
W samples
0
500
1000
1500
2000
2500
3000
−6
−4
−2
0
2
4

4.2 Clasificarea binară

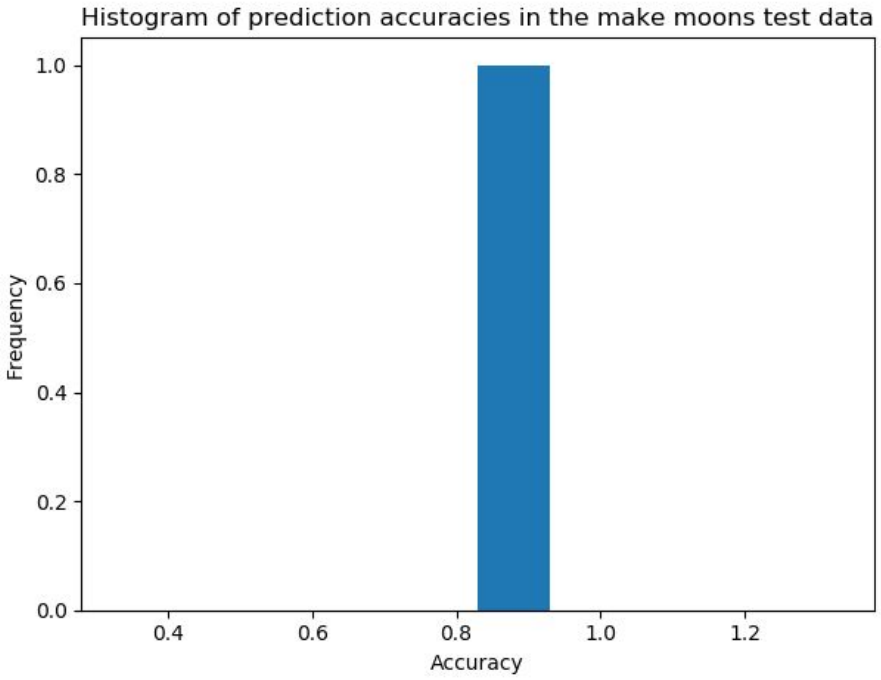
```
=== Binary class classification ===


Training ...

Device mapping:
/job:localhost/replica:0/task:0/gpu:0 -> device: 0, name: GeForce GTX 950M, pci
bus id: 0000:01:00.0
10000/10000 [100%] ████████████████████████████████ Elapsed: 47s |
Loss: 296806.875


Evaluation 100 sample(s)

Accuracy in predicting the test data =  86.36


Evaluation 1 sample(s)

Accuracy in predicting the test data =  83.0
```

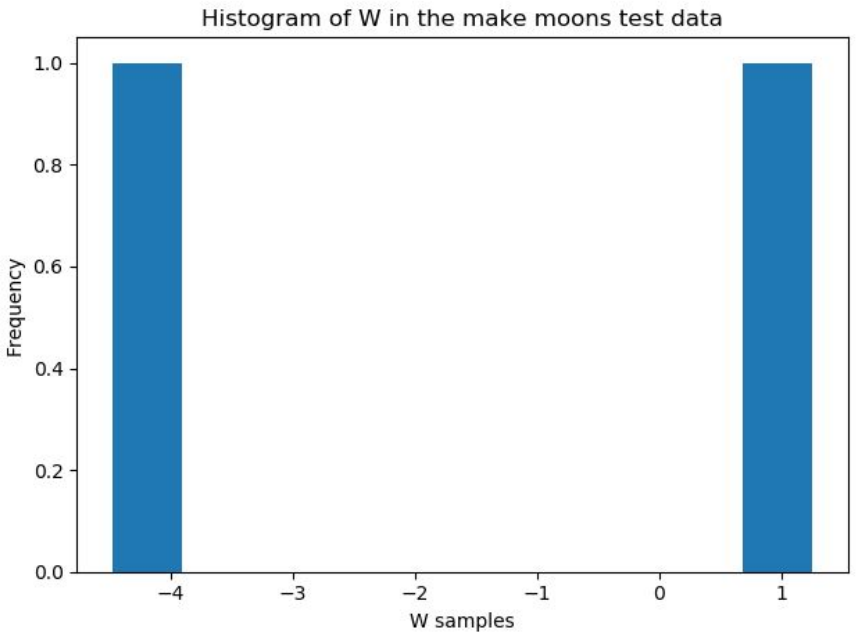
100 samples




Histogram of W in the make moons test data
Frequency
W samples
100
80
60
40
20
0
−4
−3
−2
−1
0
1

1 sample


Histogram of prediction accuracies in the make moons test data
Frequency
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
0.4
0.6
0.8
1.0
1.2



După cum putem observa în ambele situații, clasificarea multiclasă și binară, predicțiile pentru un sample, atât pe acurateţe cât și pe valorile lui **W** sunt conținute de intervalul descris de predicțiile pe un batch mai mare, de 100 de sample-uri.

Extra