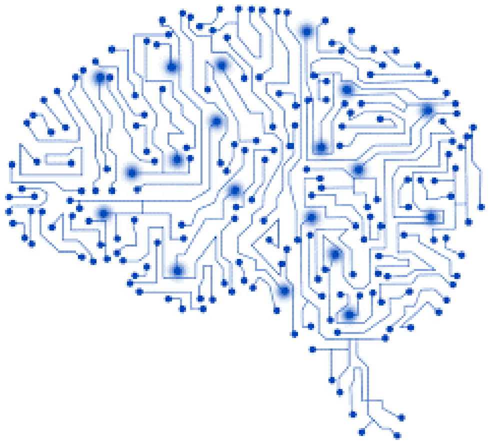
University of Minho
School of Engineering



Dados e Aprendizagem Automática

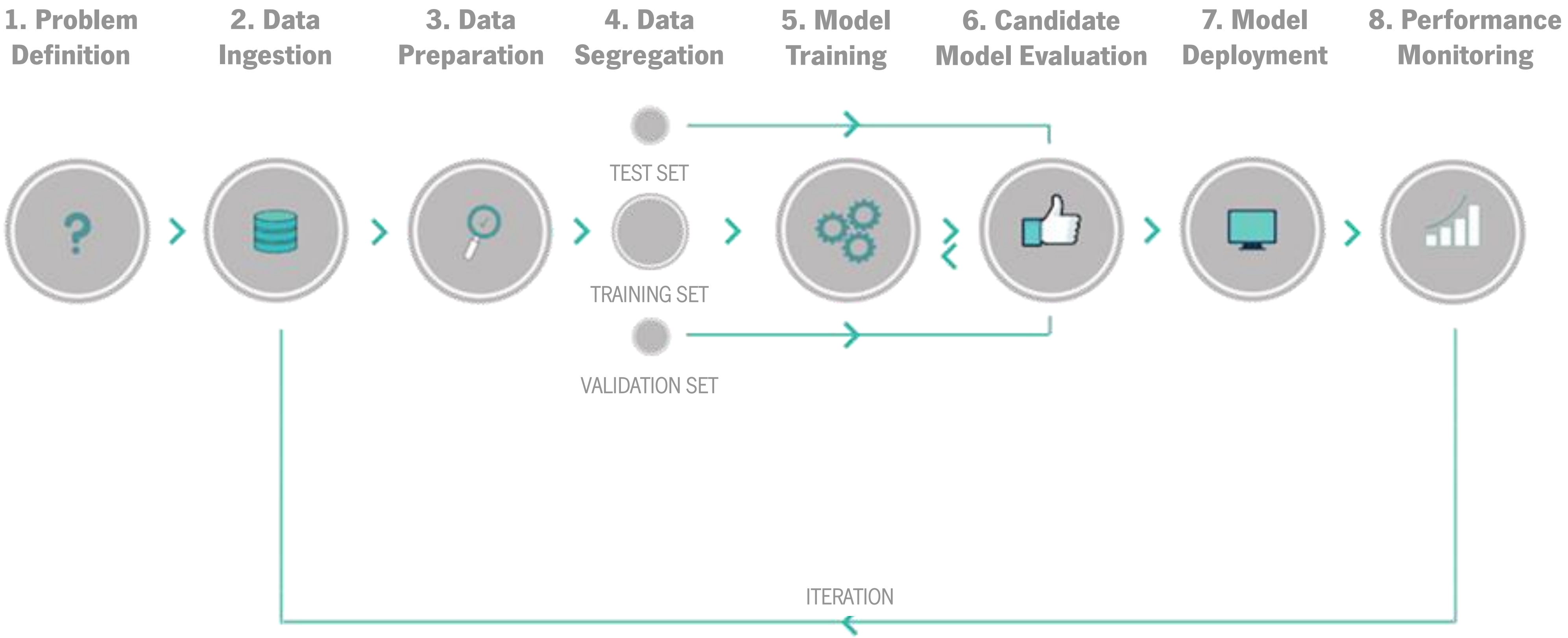
Artificial Neural Networks:
Multilayer Perceptron

DAA @ MEI-1º/MiEI-4º – 1º Semestre

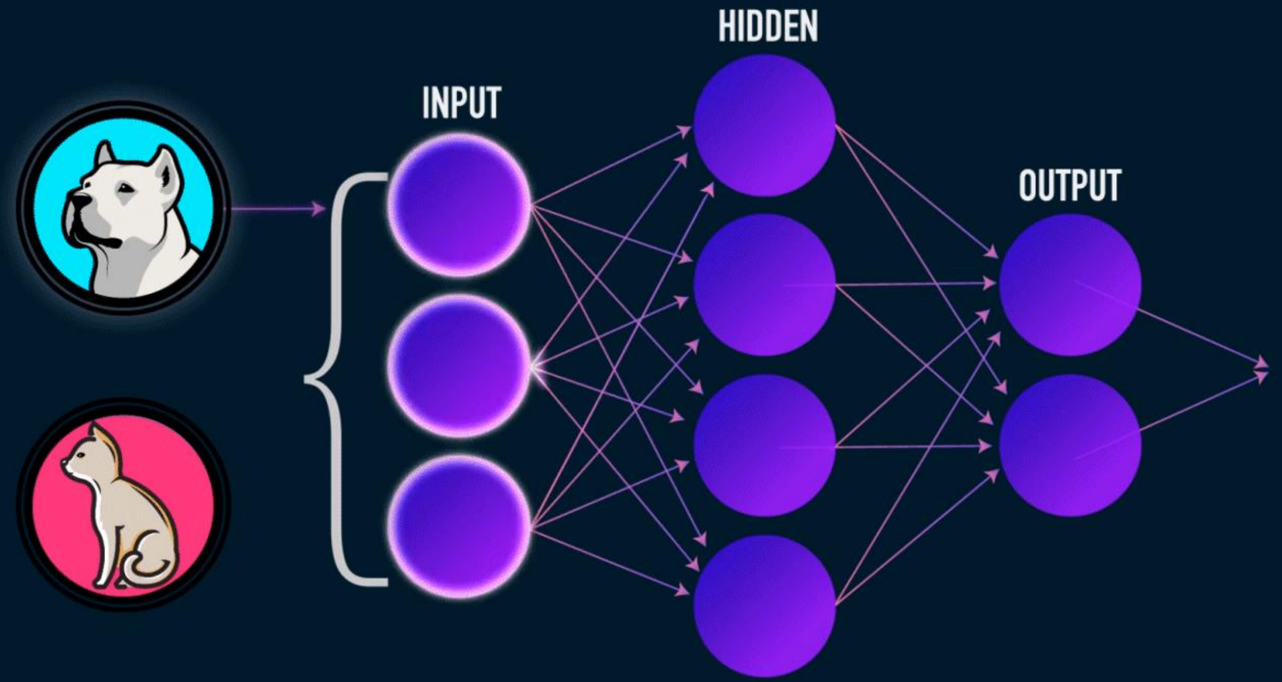
Bruno Fernandes, Dalila Alves, Filipa Ferraz, Victor Alves

Part VIII

Contents

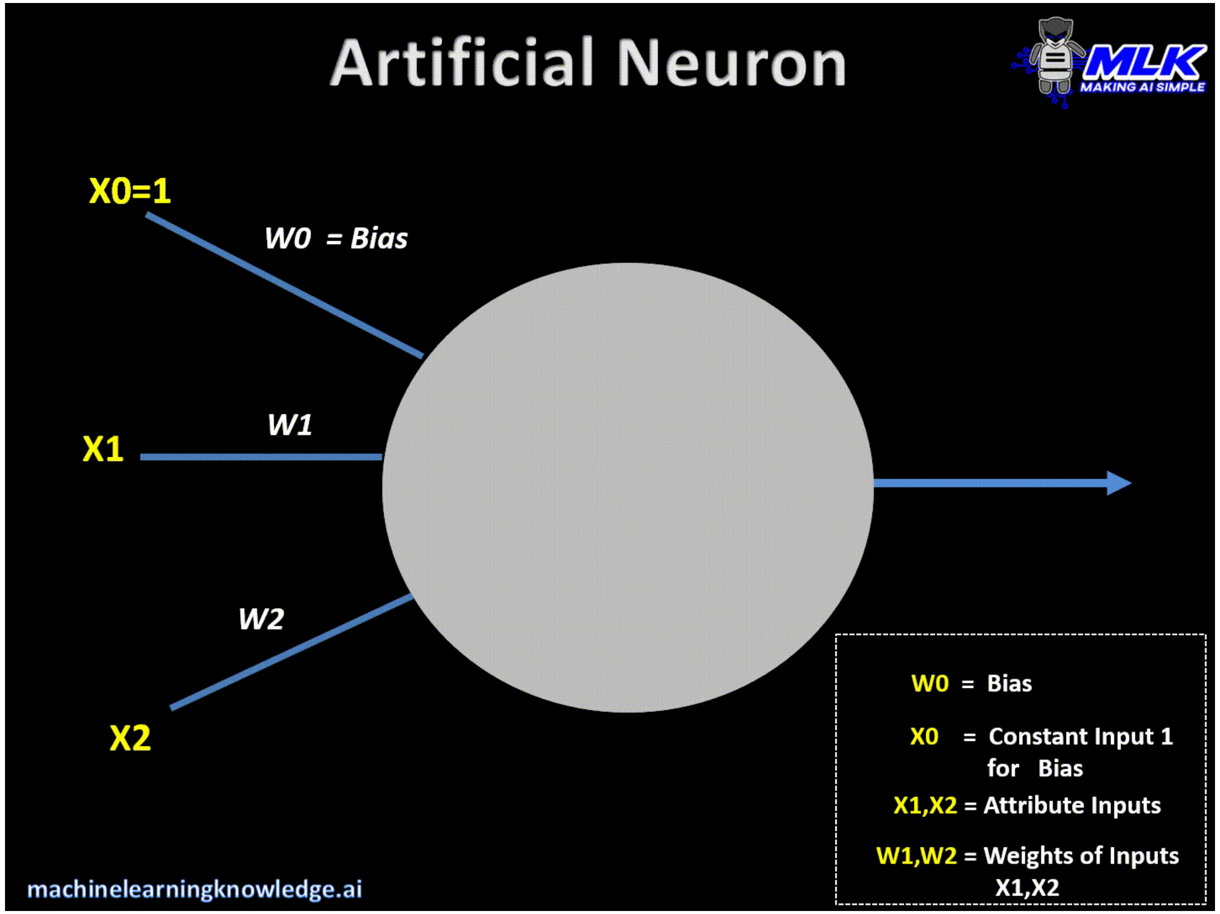

1. Problem Definition
2. Data Ingestion
3. Data Preparation
4. Data Segregation
5. Model Training
6. Candidate Model Evaluation
7. Model Deployment
8. Performance Monitoring
TEST SET
TRAINING SET
VALIDATION SET
ITERATION

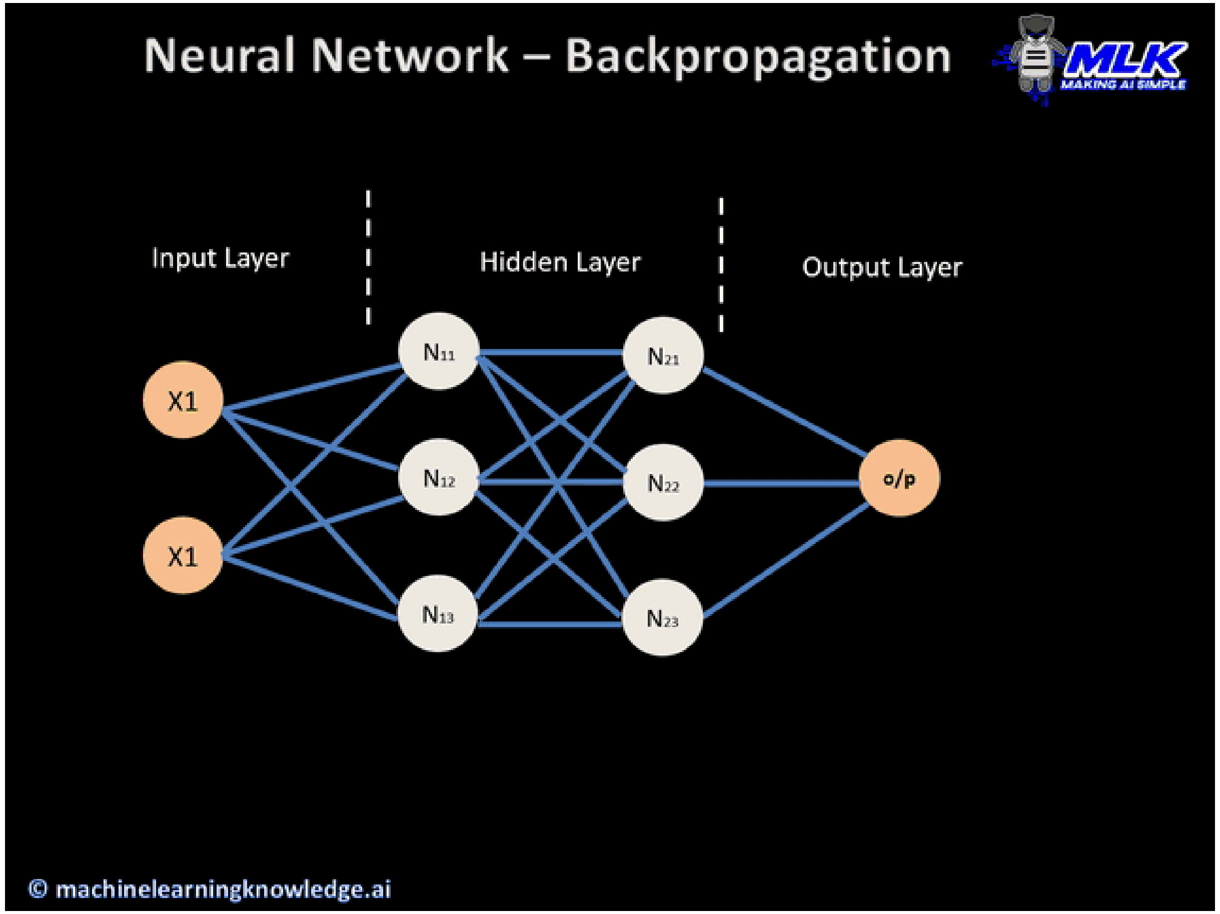
Artificial Neural Networks



(https://miro.medium.com/max/2000/1*bhFifratH9DjKqMBTeQG5A.gif)

Artificial Neural Networks

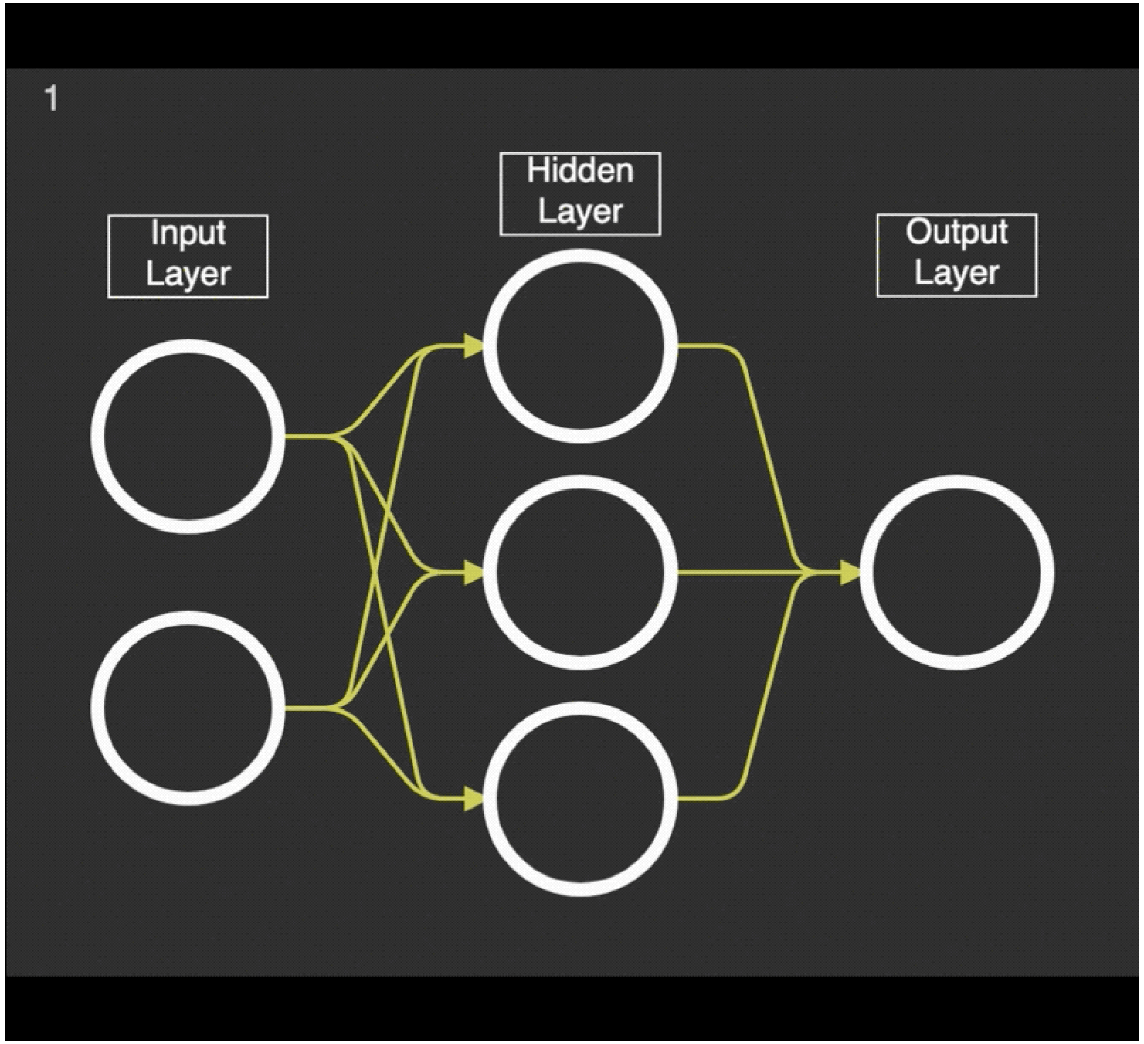




(https://machinelearningknowledge.ai/wp-content/uploads/2019/06/Artificial-Neuron-Working.gif)

(https://machinelearningknowledge.ai/wp-content/uploads/2019/10/Backpropagation.gif)

Artificial Neural Networks



(https://mlfromscratch.com/content/images/size/w2000/2019/12/nn2.png)

Artificial Neural Networks

For this example, we will use the already known the **Titanic dataset**. To our goal, we will try to predict the class of the passenger.
We have used several classifiers already; now, let's try using **Multilayer Perceptrons (MLPs)**, a class of Artificial Neural Networks.

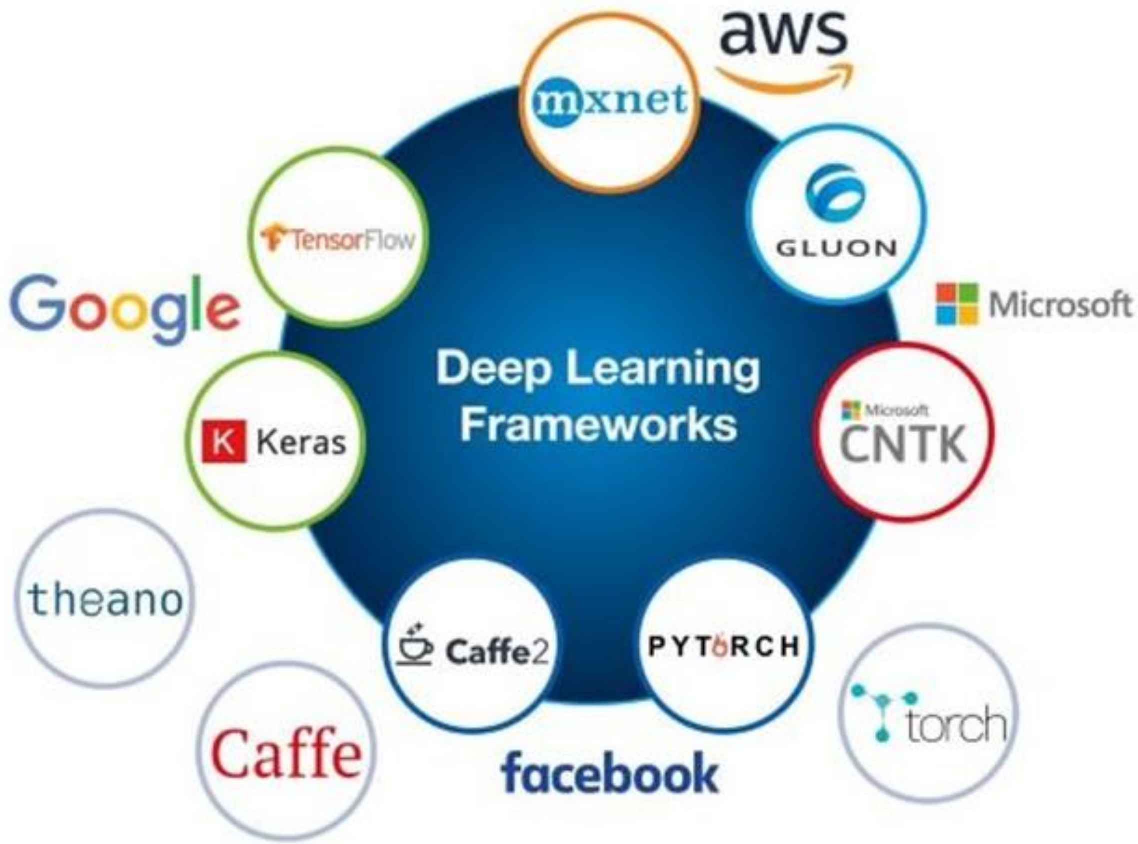
To implement our first Artificial Neural Network, we will use:

PyTorch

ML learning library based on the Torch library, used for applications such as computer vision and natural language processing, originally developed by Meta AI and now part of the Linux Foundation.

Common Frameworks

Theano	Python library that allows you to define, optimize and calculate mathematical expressions with multidimensional arrays efficiently. It works with GPUs and efficiently performs differential calculations. University of Montreal's lab, MLA
Lasagne	Light library for building and training neural networks using Theano
Blocks	Theano-based framework for building and training neural networks
TensorFlow	Open-source library for numerical computation using graphs Google Brain team
Keras	Deep learning library for Python; runs on Theano or TensorFlow
MXNet	Deep learning framework designed for efficiency and flexibility Amazon
PyTorch	Optimized flexible tensors and neural networks library with strong GPU support. Facebook Artificial Intelligence Research team (FAIR)
Torch	Ronan Collobert
Caffe	Berkeley Vision and Learning Center
CNTK	Microsoft
Deeplearning4j	Skymind



PyTorch

PyTorch is an **optimized tensor library for deep learning** using GPUs and CPUs:

- Open-source software library for high-performance numerical computation;
- Strong support for ML and DL;
- Flexibility, ease of use, performance optimization capabilities, and thriving ML community
- Dynamic computational graphs, Pythonic nature, and ease of use for prototyping models have made it a top choice in the research community. Many large companies like Amazon, Tesla, Meta, and Open AI use PyTorch to power ML and AI research initiatives;
- Commonly used in applications like image recognition and language processing;
- PyTorch allows quicker prototyping than TensorFlow. TensorFlow treats the NN as a static object; so, if you want to change the behavior of your model, you have to start from scratch.

You may need to install `pytorch`. Select the configuration that suits your machine: https://pytorch.org/get-started/locally/

Multilayer Perceptron on Titanic Dataset

FE and EDA

```
df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Multilayer Perceptron on Titanic Dataset

FE and EDA

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Let's impute by mean the `NaN` values of *Age*

```
def med_impute_nan(df):
    med_impute = df.copy()
    med_impute["Age"] = med_impute["Age"].fillna(med_impute["Age"].median())
    return med_impute
```

```
med_impute = med_impute_nan(df)
```

```
df = med_impute
df.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

Multilayer Perceptron on Titanic Dataset

Drop `NaN` values and fill with mode for *Embarked* and *Cabin*

```
embark = df['Embarked'].dropna()
```

```
df['Embarked'] = df['Embarked'].fillna(df['Embarked'].mode()[0])
```

```
df['Cabin'] = df['Cabin'].fillna(df['Cabin'].mode()[0])
```

```
df.isnull().sum()
```

```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin          0
Embarked       0
dtype: int64
```

Multilayer Perceptron on Titanic Dataset

Transform *Sex* values to 0 and 1

```
df['Sex'] = df['Sex'].apply(lambda x: 1 if x == 'male' else 0)
```

Drop categoric columns

```
df.drop(columns=['Name', 'Ticket', 'Cabin'], inplace=True)
```

Label encode feature *Embarked*

```
cols = ['Embarked']
le = LabelEncoder()

for col in cols:
    df[col] = le.fit_transform(df[col])
df.head()
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	1	0	3	1	22.0	1	0	7.2500	2
1	2	1	1	0	38.0	1	0	71.2833	0
2	3	1	3	0	26.0	0	0	7.9250	2
3	4	1	1	0	35.0	1	0	53.1000	2
4	5	0	3	1	35.0	0	0	8.0500	2

Save the data to file

```
t = pd.DataFrame(df)
filename = "titanic_ds.csv"
t.to_csv(filename, index=False, encoding='utf-8')
```

Multilayer Perceptron on Titanic Dataset

The target is *Pclass*. Let's define X and y and save to file:

```
df_X = t.drop('Pclass', axis=1)
df_X.head()
```

	PassengerId	Survived	Sex	Age	SibSp	Parch	Fare	Embarked
0	1	0	1	22.0	1	0	7.2500	2
1	2	1	0	38.0	1	0	71.2833	0
2	3	1	0	26.0	0	0	7.9250	2
3	4	1	0	35.0	1	0	53.1000	2
4	5	0	1	35.0	0	0	8.0500	2

```
t_X = pd.DataFrame(df_X)
filename = "titanic_X.csv"
t_X.to_csv(filename, index=False, encoding='utf-8')
```

```
df_y = t['Pclass']
df_y.head()
```

```
0    3
1    1
2    3
3    1
4    3
Name: Pclass, dtype: int64
```

```
t_y = pd.DataFrame(df_y)
filename = "titanic_y.csv"
t_y.to_csv(filename, index=False, encoding='utf-8')
```

Multilayer Perceptron on Titanic Dataset

Data preparation: we will use `torch` library to handle *data loaders* and *tensors*

```
def prepare_data(df, n_test):
    #create an instance of the dataset
    dataset = CSVDataset(df)
    #calculate the split
    train, test = dataset.get_splits(n_test)
    #prepare the data loaders
    train_dl = DataLoader(train, batch_size=len(train), shuffle=True)
    test_dl = DataLoader(test, batch_size=len(train), shuffle=True)
    return train_dl, test_dl
```

```
train_dl, test_dl = prepare_data(df, 0.33)
```

```
(891, 8)
torch.Size([891])
2
1
float32
torch.int64
```

```
class CSVDataset(Dataset):
    def __init__(self, path):
        #define inputs and outputs
        df_X = pd.read_csv("titanic_X.csv", header=0)
        df_y = pd.read_csv("titanic_y.csv", header=0)
        #convert to numpy array
        self.X = df_X.values
        self.y = df_y.values[:, 0]-1
        #ensure X and y are float32 and long tensor
        self.X = self.X.astype('float32')
        self.y = torch.tensor(self.y, dtype=torch.long, device=device)
        print(self.X.shape)
        print(self.y.shape)
        print(self.X.ndim)
        print(self.y.ndim)
        print(self.X.dtype)
        print(self.y.dtype)

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        #get an instance
        return [self.X[idx], self.y[idx]]

    def get_splits(self, n_test):
        #define test and train size
        test_size = round(n_test * len(self.X))
        train_size = len(self.X) - test_size
        #return holdout split
        return random_split(self, [train_size, test_size])
```

Multilayer Perceptron on Titanic Dataset

Data balance

```
def visualize_dataset(train_dl, test_dl):
    print(f"Train size:{len(train_dl.dataset)}")
    print(f"Test size:{len(test_dl.dataset)}")
    x, y = next(iter(train_dl)) #iterate through the loaders to fetch a batch of cases
    print(f"Shape tensor train data batch - input: {x.shape}, output: {y.shape}")
    x, y = next(iter(test_dl))
    print(f"Shape tensor test data batch - input: {x.shape}, output: {y.shape}")
```

```
visualize_dataset(train_dl, test_dl)
```

```
Train size:597
Test size:294
Shape tensor train data batch - input: torch.Size([597, 8]), output: torch.Size([597])
Shape tensor test data batch - input: torch.Size([294, 8]), output: torch.Size([294])
```

Multilayer Perceptron on Titanic Dataset

Data balance

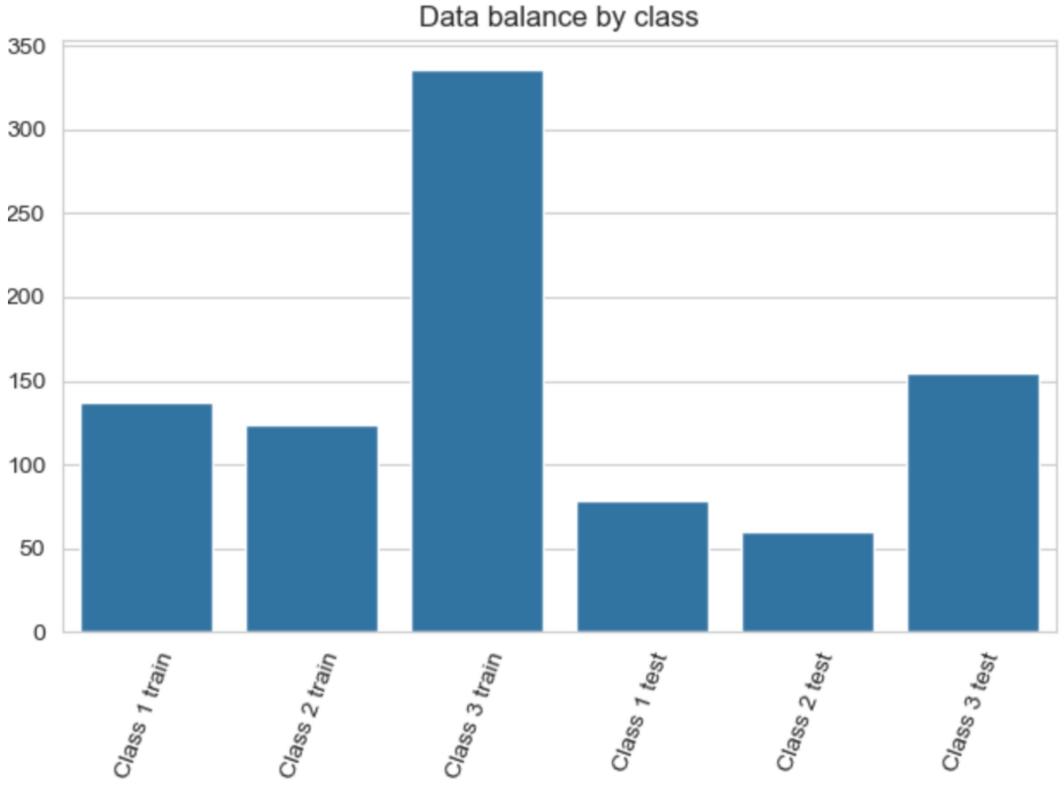
```python
def visualize_holdout_balance(train_dl, test_dl):
    _, y_train = next(iter(train_dl))
    _, y_test = next(iter(test_dl))

    sns.set_style('whitegrid')
    train_df = len(y_train)
    test_df = len(y_test)
    Class_1_train = np.count_nonzero(y_train == 0)
    Class_2_train = np.count_nonzero(y_train == 1)
    Class_3_train = np.count_nonzero(y_train == 2)
    print("train data: ", train_df)
    print("Class 1: ", Class_1_train)
    print("Class 2: ", Class_2_train)
    print("Class 3: ", Class_3_train)
    print("Values' mean (train): ", np.mean(y_train.numpy()))
```

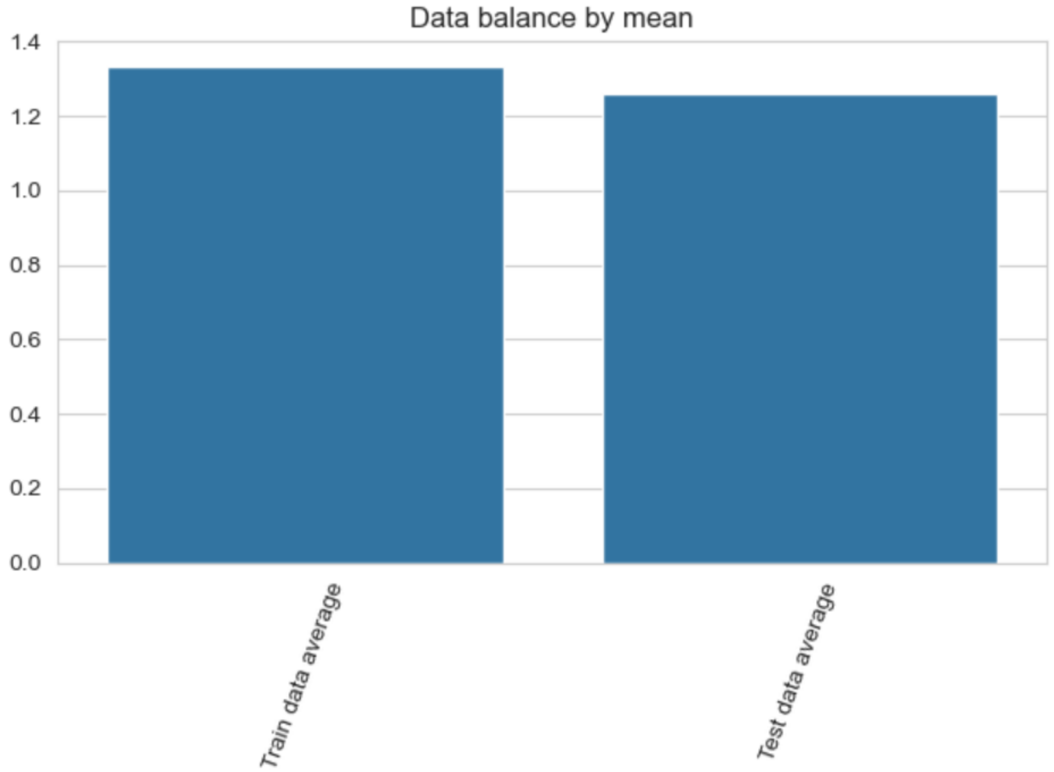
```python
Class_1_test = np.count_nonzero(y_test == 0)
Class_2_test = np.count_nonzero(y_test == 1)
Class_3_test = np.count_nonzero(y_test == 2)
print("test data: ", test_df)
print("Class 1: ", Class_1_test)
print("Class 2: ", Class_2_test)
print("Class 3: ", Class_3_test)
print("Values' mean (test): ", np.mean(y_test.numpy()))

graph = sns.barplot(x=['Class 1 train', 'Class 2 train', 'Class 3 train',
                       'Class 1 test', 'Class 2 test', 'Class 3 test'],
                    y=[Class_1_train, Class_2_train, Class_3_train,
                       Class_1_test, Class_2_test, Class_3_test])

graph.set_title('Data balance by class')
plt.xticks(rotation=70)
plt.tight_layout()
plt.savefig('data_balance_MLP.png')
plt.show()

graph = sns.barplot(x=['Train data average','Test data average'],
                    y=[np.mean(y_train.numpy()), np.mean(y_test.numpy())])
graph.set_title('Data balance by mean')
plt.xticks(rotation=70)
plt.tight_layout()
plt.show()
```

```
train data:  597
Class 1:  137
Class 2:  124
Class 3:  336
Values' mean (train):  1.3333333333333333
test data:  294
Class 1:  79
Class 2:  60
Class 3:  155
Values' mean (test):  1.2585034013605443
```

Multilayer Perceptron on Titanic Dataset

Data balance





Multilayer Perceptron on Titanic Dataset

Let's build our model using more `torch` libraries. We will use:

- `torch.nn.Module()` – *base class* to build neural networks
- `torch.nn.Linear(in_features, out_features, bias=True, device=None, dtype=None)` – *Linear* transformer
- `torch.nn.ReLU(inplace=False)` – *Rectified Linear Unit* function element-wise
- `torch.nn.Softmax(dim=None)` – *SoftMax* function to an n-dimensional input tensor. To calculate the probability of each class

Functions to **initialize** neural network parameters:

- `torch.nn.init.xavier_uniform_(tensor, gain=1.0, generator=None)` – fill the input tensor with values using a *Xavier Uniform Distribution*
- `torch.nn.init.kaiming_uniform_(tensor, a=0, mode='fan_in', nonlinearity='leaky_relu', generator=None)` – fill the input Tensor with values using a *Kaiming Uniform Distribution*

Multilayer Perceptron on Titanic Dataset

Building **Model 1**

- *Feedforward* network with fully interconnected layers
- Three *linear* layers
- Initialization using *Kaiming* and *Xavier*
- *ReLu* and *Softmax* as activation functions

```
EPOCHS = 200
LEARNING_RATE = 0.001
```

```python
class MLP_1(Module):
    def __init__(self, n_inputs):
        super(MLP_1, self).__init__()
        #1st layer input
        self.hidden1 = Linear(n_inputs, 24)
        kaiming_uniform_(self.hidden1.weight, nonlinearity='relu') # He initialization
        self.act1 = ReLU()
        #2nd layer
        self.hidden2 = Linear(24, 12)
        kaiming_uniform_(self.hidden2.weight, nonlinearity='relu')
        self.act2 = ReLU()
        #3rd layer and output
        self.hidden3 = Linear(12, 3) #one node for the predicted value output
        xavier_uniform_(self.hidden3.weight) #Glorot initialization
        #activation function
        self.act3 = Softmax(dim=1) # softmax since it is multiclass

    #input propagation sequence
    def forward(self, X):
        #input for the 1s layer
        X = self.hidden1(X)
        X = self.act1(X)
        #2nd layer
        X = self.hidden2(X)
        X = self.act2(X)
        #3rd layer and output
        X = self.hidden3(X)
        X = self.act3(X)
        return X
```

Multilayer Perceptron on Titanic Dataset

Building **Model 1**

- *Feedforward* network with fully interconnected layers
- Three *linear* layers
- Initialization using *Kaiming* and *Xavier*
- *ReLu* and *Softmax* as activation functions

```
EPOCHS = 200
LEARNING_RATE = 0.001
```

- Number of times each case is trained by the net

```python
class MLP_1(Module):
    def __init__(self, n_inputs):
        super(MLP_1, self).__init__()
        #1st layer input
        self.hidden1 = Linear(n_inputs, 24)
        kaiming_uniform_(self.hidden1.weight, nonlinearity='relu') # He initialization
        self.act1 = ReLU()
        #2nd layer
        self.hidden2 = Linear(24, 12)
        kaiming_uniform_(self.hidden2.weight, nonlinearity='relu')
        self.act2 = ReLU()
        #3rd layer and output
        self.hidden3 = Linear(12, 3) #one node for the predicted value output
        xavier_uniform_(self.hidden3.weight) #Glorot initialization
        #activation function
        self.act3 = Softmax(dim=1) # softmax since it is multiclass

    #input propagation sequence
    def forward(self, X):
        #input for the 1s layer
        X = self.hidden1(X)
        X = self.act1(X)
        #2nd layer
        X = self.hidden2(X)
        X = self.act2(X)
        #3rd layer and output
        X = self.hidden3(X)
        X = self.act3(X)
        return X
```

Multilayer Perceptron on Titanic Dataset

Building **Model 1**

- *Feedforward* network with fully interconnected layers
- Three *linear* layers
- Initialization using *Kaiming* and *Xavier*
- *ReLu* and *Softmax* as activation functions

```
EPOCHS = 200
LEARNING_RATE = 0.001
```

- Number of input features

```
class MLP_1(Module):
    def __init__(self, n_inputs):
        super(MLP_1, self).__init__()
        #1st layer input
        self.hidden1 = Linear(n_inputs, 24)
        kaiming_uniform_(self.hidden1.weight, nonlinearity='relu') # He initialization
        self.act1 = ReLU()
        #2nd layer
        self.hidden2 = Linear(24, 12)
        kaiming_uniform_(self.hidden2.weight, nonlinearity='relu')
        self.act2 = ReLU()
        #3rd layer and output
        self.hidden3 = Linear(12, 3) #one node for the predicted value output
        xavier_uniform_(self.hidden3.weight) #Glorot initialization
        #activation function
        self.act3 = Softmax(dim=1) # softmax since it is multiclass

    #input propagation sequence
    def forward(self, X):
        #input for the 1s layer
        X = self.hidden1(X)
        X = self.act1(X)
        #2nd layer
        X = self.hidden2(X)
        X = self.act2(X)
        #3rd layer and output
        X = self.hidden3(X)
        X = self.act3(X)
        return X
```

Multilayer Perceptron on Titanic Dataset

Building **Model 1**

- *Feedforward* network with fully interconnected layers
- Three *linear* layers
- Initialization using *Kaiming* and *Xavier*
- *ReLu* and *Softmax* as activation functions

```
EPOCHS = 200
LEARNING_RATE = 0.001
```

- Number of classes to predict

```
class MLP_1(Module):
    def __init__(self, n_inputs):
        super(MLP_1, self).__init__()
        #1st layer input
        self.hidden1 = Linear(n_inputs, 24)
        kaiming_uniform_(self.hidden1.weight, nonlinearity='relu') # He initialization
        self.act1 = ReLU()
        #2nd layer
        self.hidden2 = Linear(24, 12)
        kaiming_uniform_(self.hidden2.weight, nonlinearity='relu')
        self.act2 = ReLU()
        #3rd layer and output
        self.hidden3 = Linear(12, 3) #one node for the predicted value output
        xavier_uniform_(self.hidden3.weight) #Glorot initialization
        #activation function
        self.act3 = Softmax(dim=1) # softmax since it is multiclass

    #input propagation sequence
    def forward(self, X):
        #input for the 1s layer
        X = self.hidden1(X)
        X = self.act1(X)
        #2nd layer
        X = self.hidden2(X)
        X = self.act2(X)
        #3rd layer and output
        X = self.hidden3(X)
        X = self.act3(X)
        return X
```

Multilayer Perceptron on Titanic Dataset

Building **Model 1**

- *Feedforward* network with fully interconnected layers
- Three *linear* layers
- Initialization using *Kaiming* and *Xavier*
- *ReLu* and *Softmax* as activation functions

```
EPOCHS = 200
LEARNING_RATE = 0.001
```

```
model = MLP_1(8)
```

- The `dim` argument is required unless your input tensor is a vector;
- If we do not put an activation function at the end, it is considered a *linear activation*.

```
class MLP_1(Module):
    def __init__(self, n_inputs):
        super(MLP_1, self).__init__()
        #1st layer input
        self.hidden1 = Linear(n_inputs, 24)
        kaiming_uniform_(self.hidden1.weight, nonlinearity='relu') # He initialization
        self.act1 = ReLU()
        #2nd layer
        self.hidden2 = Linear(24, 12)
        kaiming_uniform_(self.hidden2.weight, nonlinearity='relu')
        self.act2 = ReLU()
        #3rd layer and output
        self.hidden3 = Linear(12, 3) #one node for the predicted value output
        xavier_uniform_(self.hidden3.weight) #Glorot initialization
        #activation function
        self.act3 = Softmax(dim=1) # softmax since it is multiclass

    #input propagation sequence
    def forward(self, X):
        #input for the 1s layer
        X = self.hidden1(X)
        X = self.act1(X)
        #2nd layer
        X = self.hidden2(X)
        X = self.act2(X)
        #3rd layer and output
        X = self.hidden3(X)
        X = self.act3(X)
        return X
```

Multilayer Perceptron on Titanic Dataset


```
print(summary(model, input_size=(len(train_dl.dataset), 8), verbose=0))
model.to(device)
```

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
MLP_1                                    [713, 3]                  --
├─Linear: 1-1                            [713, 24]                 216
├─ReLU: 1-2                              [713, 24]                 --
├─Linear: 1-3                            [713, 12]                 300
├─ReLU: 1-4                              [713, 12]                 --
├─Linear: 1-5                            [713, 3]                  39
├─Softmax: 1-6                           [713, 3]                  --
==========================================================================================
Total params: 555
Trainable params: 555
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 0.40
==========================================================================================
Input size (MB): 0.02
Forward/backward pass size (MB): 0.22
Params size (MB): 0.00
Estimated Total Size (MB): 0.25
==========================================================================================

MLP_1(
  (hidden1): Linear(in_features=8, out_features=24, bias=True)
  (act1): ReLU()
  (hidden2): Linear(in_features=24, out_features=12, bias=True)
  (act2): ReLU()
  (hidden3): Linear(in_features=12, out_features=3, bias=True)
  (act3): Softmax(dim=1)
)
```

Multilayer Perceptron on Titanic Dataset

To train the model we will use other `torch` libraries:

- `torch.nn.CrossEntropyLoss(weight=None, size_average=None, ignore_index=-100, reduce=None, reduction='mean', label_smoothing=0.0)` – computes *Cross Entropy Loss* function. It is useful when training a *classification problem* with C classes because it accepts *ground truth labels* directly as integers in `[0, no. of classes]`. There is *no need to one-hot encode* the labels. This is particularly useful when you have an *unbalanced training set*
- `torch.optim.SGD(params, lr=0.001, momentum=0, dampening=0, weight_decay=0, nesterov=False, *, maximize=False, foreach=None, differentiable=False, fused=None)` – implements *Stochastic Gradient Descent* as optimizer
- `torch.optim.Adam(params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0, amsgrad=False, *, foreach=None, maximize=False, capturable=False, differentiable=False, fused=None)` – implements *Adam* algorithm as optimizer

Multilayer Perceptron on Titanic Dataset

Train the model

```
def train_model(train_dl, val_dl, model):
    #to visualize the training process
    liveloss = PlotLosses()
    #define loss function and optimization
    criterion = CrossEntropyLoss() #sparse_categorical_crossentropy
    optimizer = SGD(model.parameters(), lr=LEARNING_RATE, momentum=0.9) #stochastic gradient descent
    #iterate the epochs
    for epoch in range(EPOCHS):
        logs = {} #
        #train phase
        model.train()
        running_loss  = 0.0
        running_corrects  = 0.0
        for inputs, labels in train_dl: #backpropagation
            inputs = inputs.to(device)
            labels = labels.to(device)
            #calculate model output
            outputs = model(inputs)
            #calculate the loss
            loss = criterion(outputs, labels)
```

Multilayer Perceptron on Titanic Dataset


```
#iterate the epochs
for epoch in range(EPOCHS):
    logs = {} #
    #train phase
    model.train()
    running_loss  = 0.0
    running_corrects  = 0.0
    for inputs, labels in train_dl: #backpropagation
        inputs = inputs.to(device)
        labels = labels.to(device)
        #calculate model output
        outputs = model(inputs)
        #calculate the loss
        loss = criterion(outputs, labels)

        optimizer.zero_grad() #sets the gradients of all parameters to zero
        loss.backward()
        #update model weights
        optimizer.step()
        running_loss += loss.detach() * inputs.size(0)
        _, preds = torch.max(outputs, 1) # Get predictions from the maximum value
        running_corrects += torch.sum(preds == labels.data)
    epoch_loss = running_loss / len(train_dl.dataset)
    epoch_acc = running_corrects.float() / len(train_dl.dataset)
    logs['loss'] = epoch_loss.item()
    logs['accuracy'] = epoch_acc.item()
```

Multilayer Perceptron on Titanic Dataset

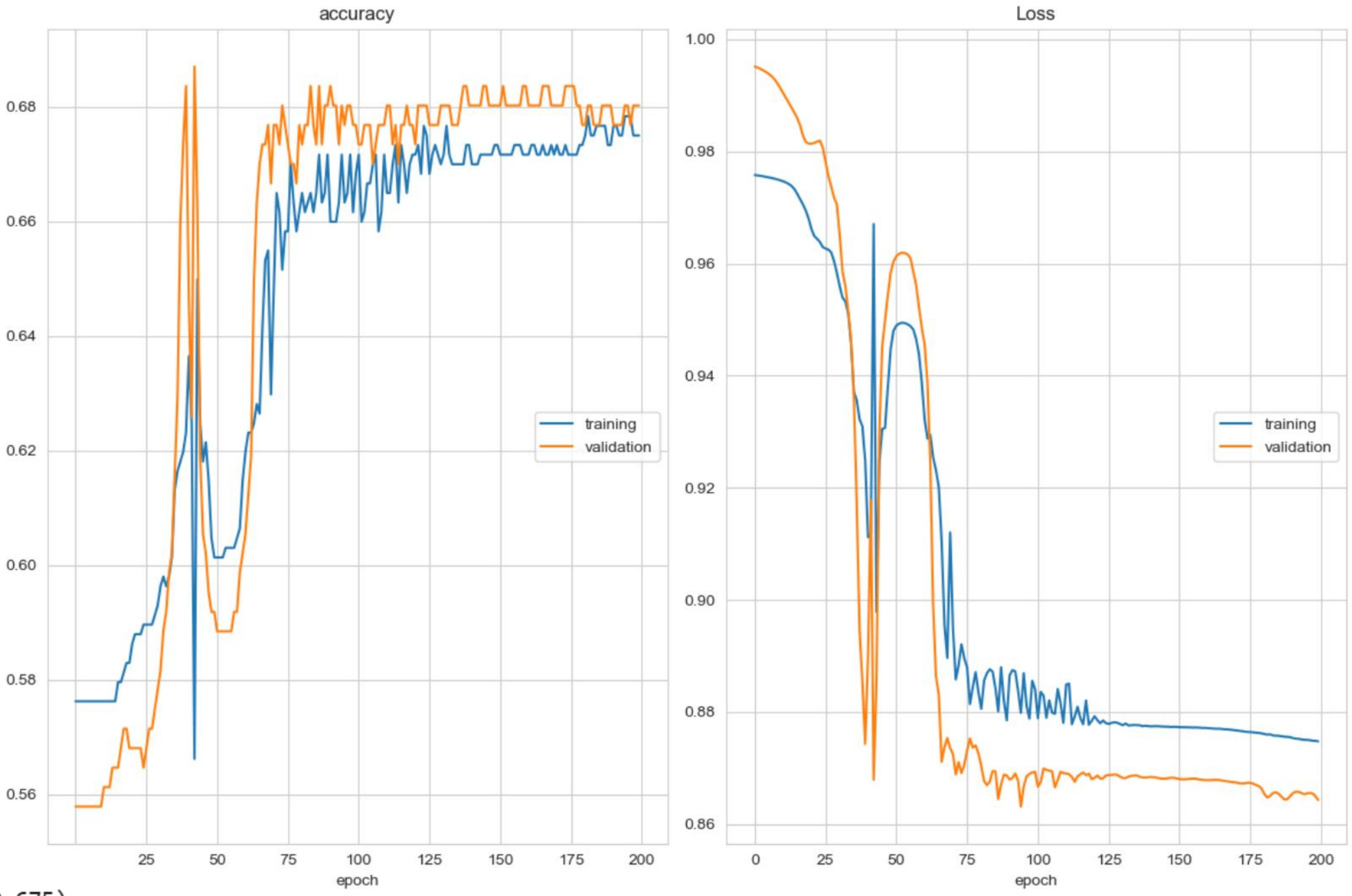

```
            loss.backward()
            #update model weights
            optimizer.step()
            running_loss += loss.detach() * inputs.size(0)
            _, preds = torch.max(outputs, 1) # Get predictions from the maximum value
            running_corrects += torch.sum(preds == labels.data)
        epoch_loss = running_loss / len(train_dl.dataset)
        epoch_acc = running_corrects.float() / len(train_dl.dataset)
        logs['loss'] = epoch_loss.item()
        logs['accuracy'] = epoch_acc.item()

        #Validation phase
        model.eval()
        running_loss  = 0.0
        running_corrects  = 0.0
        for inputs, labels in val_dl:
            inputs = inputs.to(device)
            labels = labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            running_loss += loss.detach() * inputs.size(0)
            _, preds = torch.max(outputs, 1) # Get predictions from the maximum value
            running_corrects += torch.sum(preds == labels.data)
        epoch_loss = running_loss / len(val_dl.dataset)
        epoch_acc = running_corrects.float() / len(val_dl.dataset)
        logs['val_loss'] = epoch_loss.item()
        logs['val_accuracy'] = epoch_acc.item()
        liveloss.update(logs)
        liveloss.send()
```

Fraction of the data to be used as validation data. The model will set apart this fraction of the data, will not train on it, and will evaluate the loss and the model metrics on this data at the end of each epoch.

```
train_model(train_dl, test_dl, model)
```

Multilayer Perceptron on Titanic Dataset

Live training results




```
accuracy
	training           (min:    0.566, max:    0.678, cur:    0.675)
	validation         (min:    0.558, max:    0.687, cur:    0.680)
Loss
	training           (min:    0.875, max:    0.976, cur:    0.875)
	validation         (min:    0.863, max:    0.995, cur:    0.864)
```

Multilayer Perceptron on Titanic Dataset

Evaluate the model

```python
def evaluate_model(test_dl, model):
    predictions = list()
    actual_values = list()
    for i, (inputs, labels) in enumerate(test_dl):
        #evaluate the model with test cases
        yprev = model(inputs)
        #remove numpy array
        yprev = yprev.detach().numpy()
        actual = labels.numpy()
        #convert to labels' class
        yprev = np.argmax(yprev, axis=1)
        #reshape for stacking
        actual = actual.reshape((len(actual), 1))
        yprev = yprev.reshape((len(yprev), 1))
        #save
        predictions.append(yprev)
        actual_values.append(actual)
        break
    predictions, actual_values = np.vstack(predictions), np.vstack(actual_values)
    return predictions, actual_values
```

```python
def display_confusion_matrix(cm):
    plt.figure(figsize = (16,8))
    sns.heatmap(cm,annot=True,xticklabels=['Class 1', 'Class 2', 'Class 3'],
                yticklabels=['Class 1', 'Class 2', 'Class 3'],
                annot_kws={"size": 12}, fmt='g', linewidths=.5)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
```

```python
predictions, actual_values = evaluate_model(test_dl, model)
```

Multilayer Perceptron on Titanic Dataset


```
success = 0
failure = 0
for r,p in zip(actual_values, predictions):
    print(f'real:{r+1} prediction:{p+1}')
    if r==p: success+=1
    else: failure+=1
```

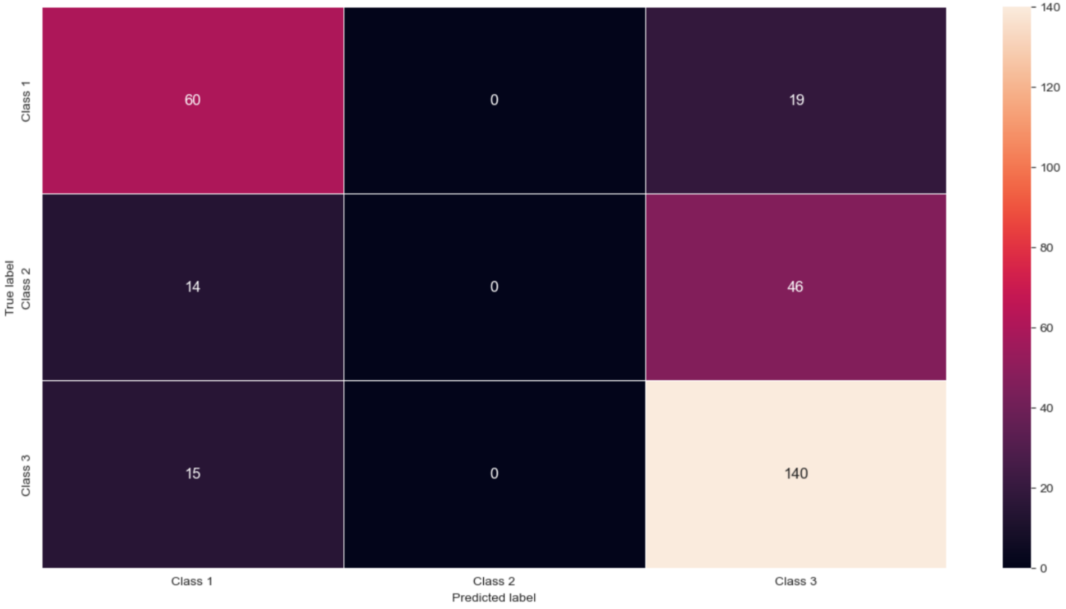
```
real:[3] prediction:[3]
real:[3] prediction:[3]
real:[3] prediction:[3]
real:[3] prediction:[3]
real:[1] prediction:[1]
real:[2] prediction:[3]
real:[1] prediction:[1]
```

```
acc = accuracy_score(actual_values, predictions)
print(f'Accuracy: {acc:0.3f}\n')
print(f'success:{success} failure:{failure}')
```

```
Accuracy: 0.680

success:200 failure:94
```

```
print(classification_report(actual_values, predictions)
cm = confusion_matrix(actual_values, predictions)
print(cm)
display_confusion_matrix(cm)
```

```
              precision    recall  f1-score   support

           0       0.67      0.76      0.71        79
           1       0.00      0.00      0.00        60
           2       0.68      0.90      0.78       155

    accuracy                           0.68       294
   macro avg       0.45      0.55      0.50       294
weighted avg       0.54      0.68      0.60       294

[[ 60   0  19]
 [ 14   0  46]
 [ 15   0 140]]
```



Multilayer Perceptron on Titanic Dataset

<u>Apply the model:</u> make a prediction for one case

```
def predict(row, model):
    #convert row to tensor
    row = Tensor([row])
    #make a prediction
    yprev = model(row)
    #remove the numpy array
    yprev = yprev.detach().numpy()
    return yprev
```

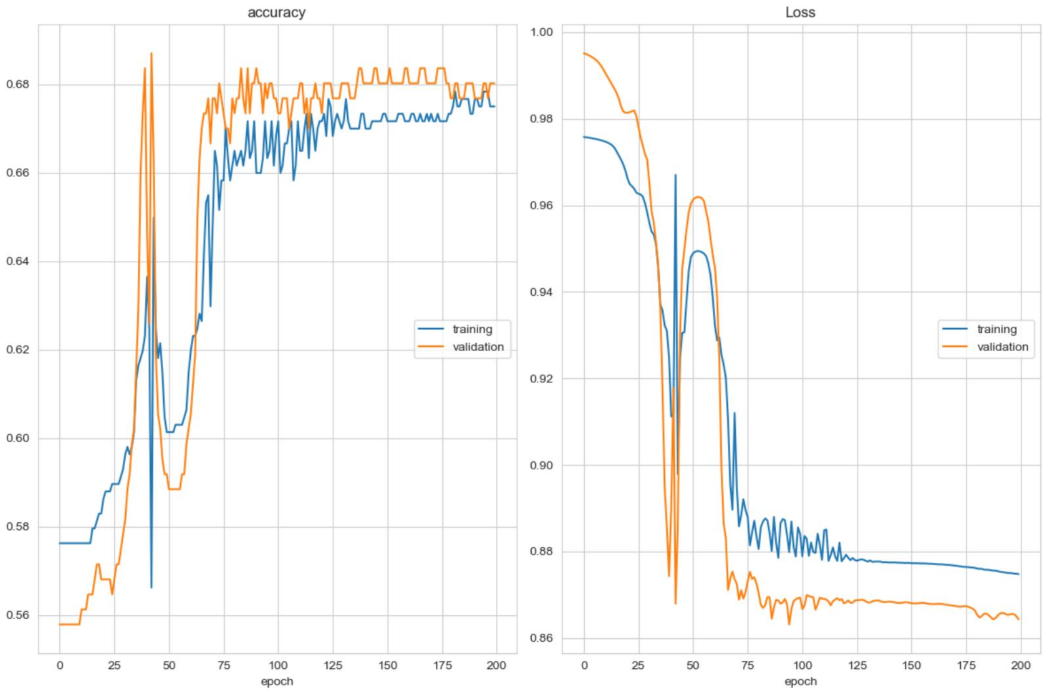
```
row = [5, 0, 1, 34, 0, 1, 8, 1]
yprev = predict(row, model)
print('Predicted: %s (class=%d)' % (yprev, np.argmax(yprev)+1))
```

```
Predicted: [[0.96104145 0.02388389 0.01507465]] (class=1)
```

Multilayer Perceptron on Titanic Dataset

Model 1: 200 epochs, LR: 0.001, 3 layers – accuracy: 0.680

What can be modified to improve this model?




```
accuracy
	training	(min:	0.566, max:	0.678, cur:	0.675)
	validation	(min:	0.558, max:	0.687, cur:	0.680)
Loss
	training	(min:	0.875, max:	0.976, cur:	0.875)
	validation	(min:	0.863, max:	0.995, cur:	0.864)
```

	precision	recall	f1-score	support
0	0.67	0.76	0.71	79
1	0.00	0.00	0.00	60
2	0.68	0.90	0.78	155
accuracy			0.68	294
macro avg	0.45	0.55	0.50	294
weighted avg	0.54	0.68	0.60	294

```
[[ 60   0  19]
 [ 14   0  46]
 [ 15   0 140]]
```

Multilayer Perceptron on Titanic Dataset

Let's create another model and use other hyperparameters.

Model 2: 200 epochs, ↑ LR: 0.01, ↑ 4 layers

But first, let's normalize the data. Data scaling or normalization is a process of making model data in a standard format so that the training is improved, accurate and faster. We will work on features *Age* and *Fare:*

```
df = pd.read_csv("titanic_ds.csv")
```

```
min_age = df['Age'].min()
max_age = df['Age'].max()

df['Age'] = (df['Age'] - min_age)/(max_age - min_age)
```

```
df['Age'].describe()
```

```
count    891.000000
mean       0.363679
std        0.163605
min        0.000000
25%        0.271174
50%        0.346569
75%        0.434531
max        1.000000
Name: Age, dtype: float64
```

```
min_fare = df['Fare'].min()
max_fare = df['Fare'].max()

df['Fare'] = (df['Fare'] - min_fare)/(max_fare - min_fare)
```

```
df['Fare'].describe()
```

```
count    891.000000
mean       0.062858
std        0.096995
min        0.000000
25%        0.015440
50%        0.028213
75%        0.060508
max        1.000000
Name: Fare, dtype: float64
```

Multilayer Perceptron on Titanic Dataset


```
df.head()
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	1	0	3	1	0.271174	1	0	0.014151	2
1	2	1	1	0	0.472229	1	0	0.139136	0
2	3	1	3	0	0.321438	0	0	0.015469	2
3	4	1	1	0	0.434531	1	0	0.103644	2
4	5	0	3	1	0.434531	0	0	0.015713	2

Let’s save the new data:

```
t = pd.DataFrame(df)
filename = "titanic_scaled.csv"
t.to_csv(filename, index=False, encoding='utf-8')
```

Multilayer Perceptron on Titanic Dataset

And redefine X and y:

```
df_X = t.drop('Pclass', axis=1)
df_X.head()
```

	PassengerId	Survived	Sex	Age	SibSp	Parch	Fare	Embarked
0	1	0	1	0.271174	1	0	0.014151	2
1	2	1	0	0.472229	1	0	0.139136	0
2	3	1	0	0.321438	0	0	0.015469	2
3	4	1	0	0.434531	1	0	0.103644	2
4	5	0	1	0.434531	0	0	0.015713	2

```
t_X = pd.DataFrame(df_X)
filename = "titanic_X_scaled.csv"
t_X.to_csv(filename, index=False, encoding='utf-8')
```

```
df_y = t['Pclass']
df_y.head()
```

```
0    3
1    1
2    3
3    1
4    3
Name: Pclass, dtype: int64
```

```
t_y = pd.DataFrame(df_y)
filename = "titanic_y_scaled.csv"
t_y.to_csv(filename, index=False, encoding='utf-8')
```

Multilayer Perceptron on Titanic Dataset

Data preparation

```
class CSVDataset(Dataset):
    def __init__(self, path):

        df_X = pd.read_csv("titanic_X_scaled.csv", header=0)
        df_y = pd.read_csv("titanic_y_scaled.csv", header=0)

        self.X = df_X.values
        self.y = df_y.values[:, 0]-1

        self.X = self.X.astype('float32')
        self.y = torch.tensor(self.y, dtype=torch.long, device=device)

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        return [self.X[idx], self.y[idx]]

    def get_splits(self, n_test):
        test_size = round(n_test * len(self.X))
        train_size = len(self.X) - test_size
        return random_split(self, [train_size, test_size])

def prepare_data(df, n_test):
    dataset = CSVDataset(df)
    train, test = dataset.get_splits(n_test)
    train_dl = DataLoader(train, batch_size=len(train), shuffle=True)
    test_dl = DataLoader(test, batch_size=len(train), shuffle=True)
    return train_dl, test_dl
```

```
train_dl, test_dl = prepare_data(df, 0.33)
```

Multilayer Perceptron on Titanic Dataset

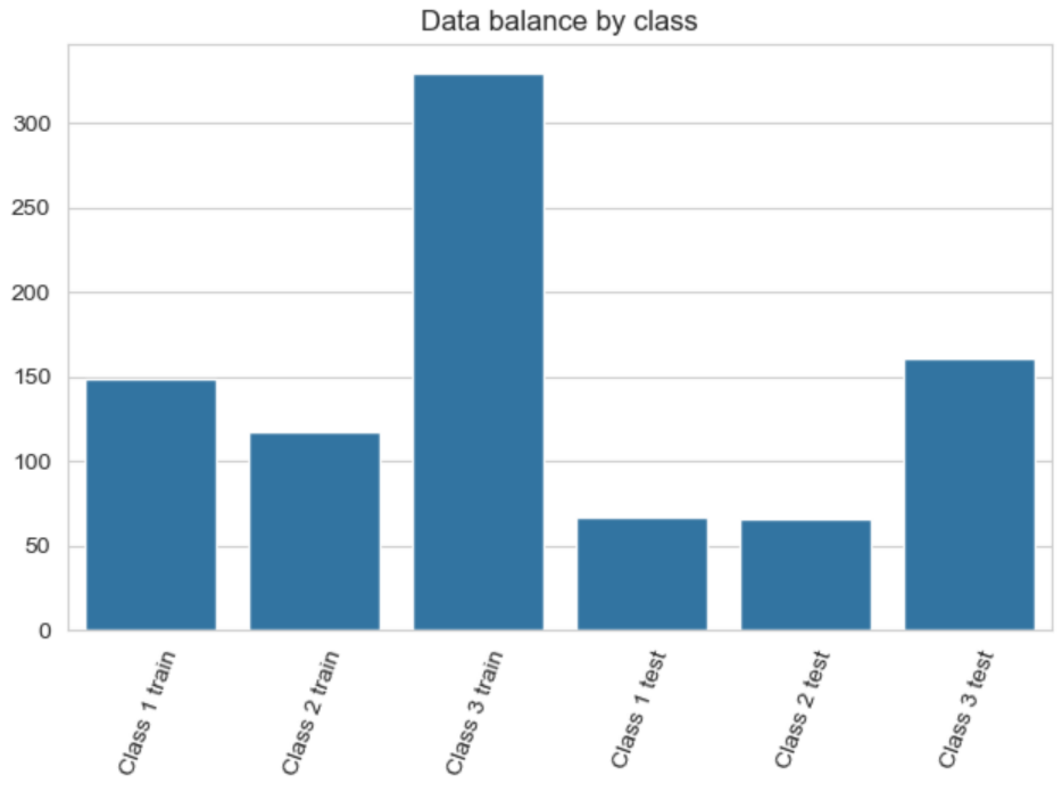
Data balance

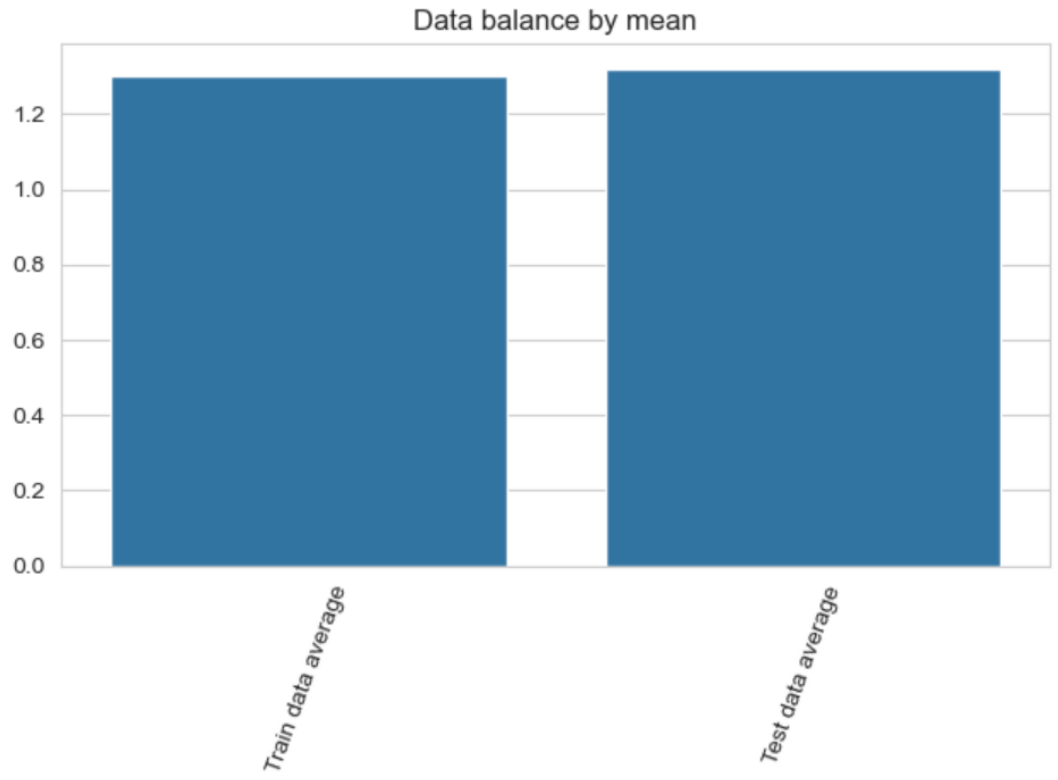
```
Train size:597
Test size:294
Shape tensor train data batch - input: torch.Size([597, 8]), output: torch.Size([597])
Shape tensor test data batch - input: torch.Size([294, 8]), output: torch.Size([294])
```

```
train data:  597
Class 1:  149
Class 2:  118
Class 3:  330
Values' mean (train):  1.3031825795644891
test data:  294
Class 1:  67
Class 2:  66
Class 3:  161
Values' mean (test):  1.3197278911564625
```

Multilayer Perceptron on Titanic Dataset

Data balance





Multilayer Perceptron on Titanic Dataset

Building **Model 2**

```
EPOCHS = 200
LEARNING_RATE = 0.01
```

```
model = MLP_2(8)
```

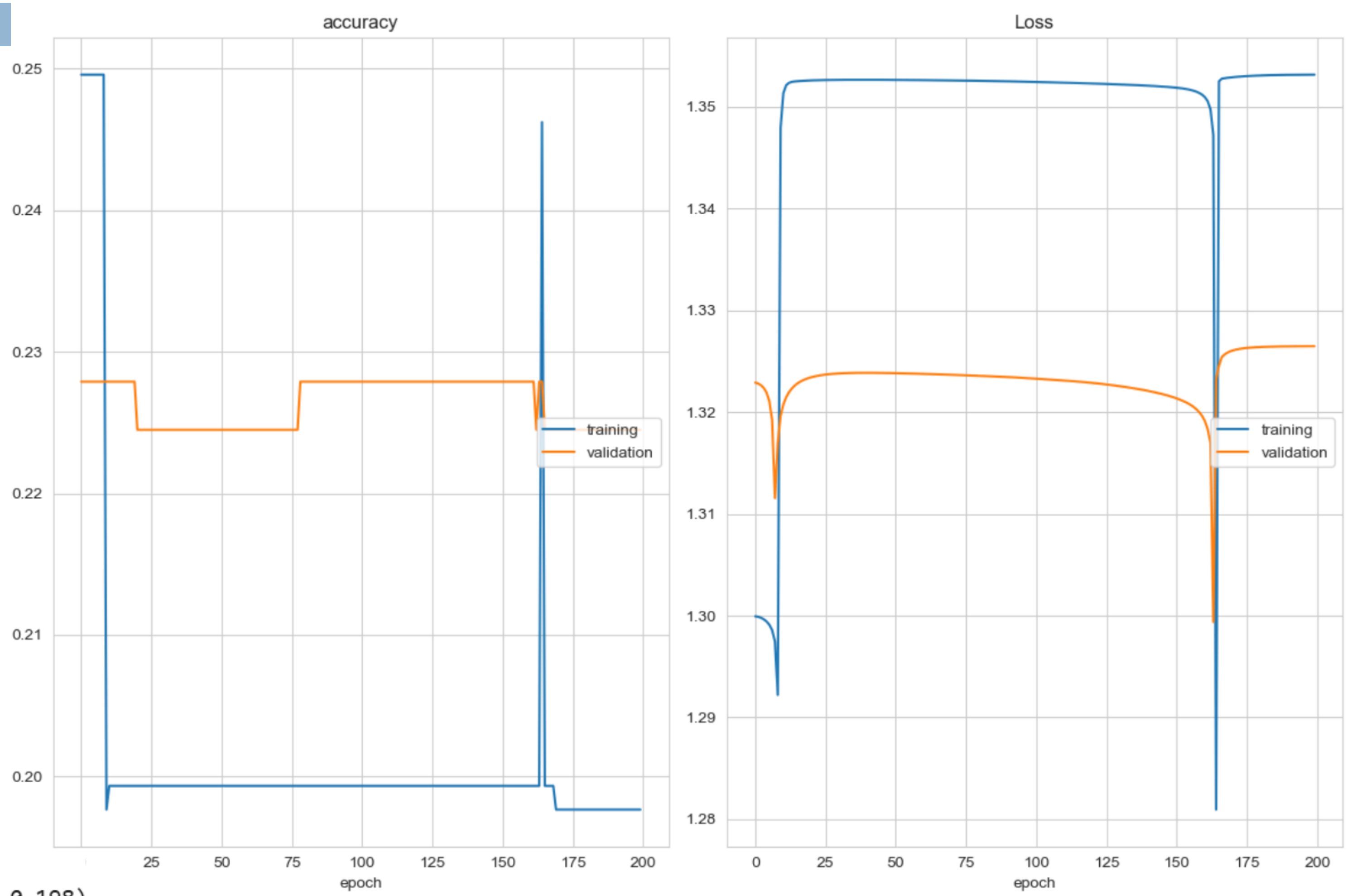
```
class MLP_2(Module):
    def __init__(self, n_inputs):
        super(MLP_2, self).__init__()
        #1st layer input
        self.hidden1 = Linear(n_inputs, 20)
        kaiming_uniform_(self.hidden1.weight, nonlinearity='relu')
        self.act1 = ReLU()
        #2nd layer
        self.hidden2 = Linear(20, 32)
        kaiming_uniform_(self.hidden2.weight, nonlinearity='relu')
        self.act2 = ReLU()
        #3rd layer
        self.hidden3 = Linear(32, 12)
        kaiming_uniform_(self.hidden3.weight, nonlinearity='relu')
        self.act3 = ReLU()
        #4th layer and output
        self.hidden4 = Linear(12, 3)
        xavier_uniform_(self.hidden4.weight)
        self.act4 = Softmax(dim=1)

    def forward(self, X):
        X = self.hidden1(X)
        X = self.act1(X)
        X = self.hidden2(X)
        X = self.act2(X)
        X = self.hidden3(X)
        X = self.act3(X)
        X = self.hidden4(X)
        X = self.act4(X)
        return X
```

Multilayer Perceptron on Titanic Dataset


```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
MLP_2                                    [597, 3]                  --
├─Linear: 1-1                            [597, 20]                 180
├─ReLU: 1-2                              [597, 20]                 --
├─Linear: 1-3                            [597, 32]                 672
├─ReLU: 1-4                              [597, 32]                 --
├─Linear: 1-5                            [597, 12]                 396
├─ReLU: 1-6                              [597, 12]                 --
├─Linear: 1-7                            [597, 3]                  39
├─Softmax: 1-8                           [597, 3]                  --
==========================================================================================
Total params: 1,287
Trainable params: 1,287
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 0.77
==========================================================================================
Input size (MB): 0.02
Forward/backward pass size (MB): 0.32
Params size (MB): 0.01
Estimated Total Size (MB): 0.34
==========================================================================================

MLP_2(
  (hidden1): Linear(in_features=8, out_features=20, bias=True)
  (act1): ReLU()
  (hidden2): Linear(in_features=20, out_features=32, bias=True)
  (act2): ReLU()
  (hidden3): Linear(in_features=32, out_features=12, bias=True)
  (act3): ReLU()
  (hidden4): Linear(in_features=12, out_features=3, bias=True)
  (act4): Softmax(dim=1)
)
```

Multilayer Perceptron on Titanic Dataset

Live training results




```
accuracy
	training		(min:	0.198, max:	0.250, cur:	0.198)
	validation		(min:	0.224, max:	0.228, cur:	0.224)
Loss
	training		(min:	1.281, max:	1.353, cur:	1.353)
	validation		(min:	1.299, max:	1.326, cur:	1.326)
```

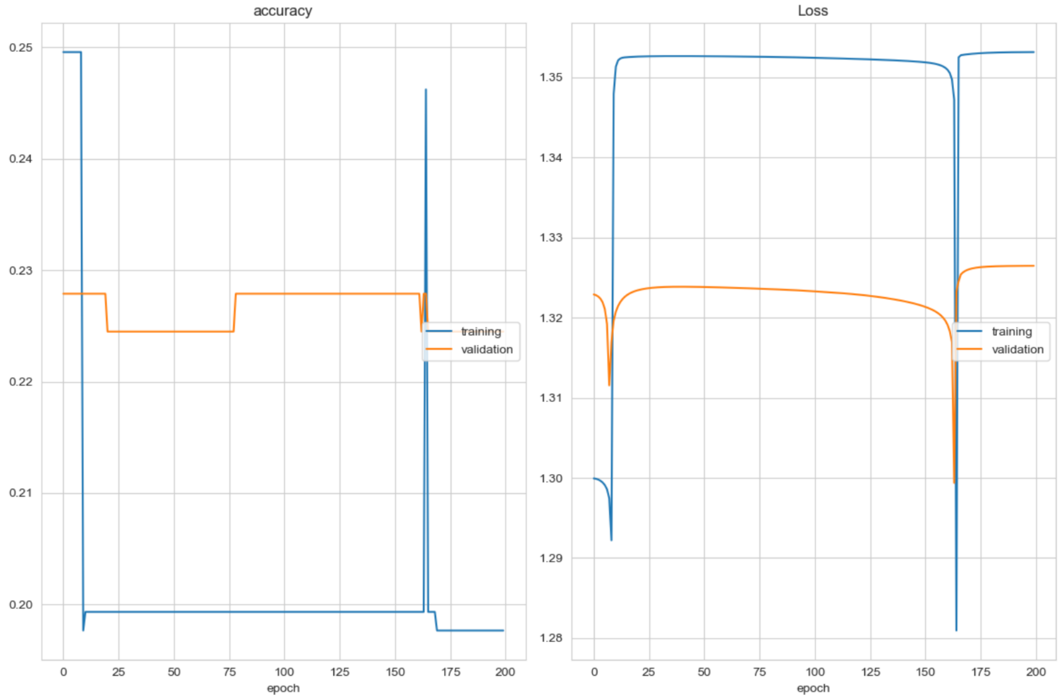
Multilayer Perceptron on Titanic Dataset

Evaluate the model

```
Accuracy: 0.224

success:66 failure:228


              precision    recall  f1-score   support

           0       0.00      0.00      0.00        67
           1       0.22      1.00      0.37        66
           2       0.00      0.00      0.00       161

    accuracy                           0.22       294
   macro avg       0.07      0.33      0.12       294
weighted avg       0.05      0.22      0.08       294
```

Multilayer Perceptron on Titanic Dataset

Model 2: 200 epochs, ↑ LR: 0.01, ↑ 4 layers – accuracy: 0.224 ↓




```
accuracy
	training           (min:    0.198, max:    0.250, cur:    0.198)
	validation         (min:    0.224, max:    0.228, cur:    0.224)
Loss
	training           (min:    1.281, max:    1.353, cur:    1.353)
	validation         (min:    1.299, max:    1.326, cur:    1.326)
```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	67
1	0.22	1.00	0.37	66
2	0.00	0.00	0.00	161
accuracy			0.22	294
macro avg	0.07	0.33	0.12	294
weighted avg	0.05	0.22	0.08	294

Multilayer Perceptron on Titanic Dataset

Let's create another model.

Model 3: ↑ 250 epochs, ↓ LR: 0.005, ↑ 6 layers

Data preparation →

```
class CSVDataset():
    def __init__(self, ):

        df_X = pd.read_csv("titanic_X_scaled.csv", header=0)
        df_y = pd.read_csv("titanic_y_scaled.csv", header=0)

        self.X = df_X.values
        self.y = df_y.values[:, 0]-1

        self.X = self.X.astype('float32')
        self.y = torch.tensor(self.y, dtype=torch.long, device=device)

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        return [self.X[idx], self.y[idx]]

    def get_splits(self, n_test):
        test_size = round(n_test * len(self.X))
        train_size = len(self.X) - test_size
        return random_split(self, [train_size, test_size])

def prepare_data(n_test):
    dataset = CSVDataset()
    train, test = dataset.get_splits(n_test)
    train_dl = DataLoader(train, batch_size=len(train), shuffle=True)
    test_dl = DataLoader(test, batch_size=len(train), shuffle=True)
    return train_dl, test_dl
```

```
train_dl, test_dl = prepare_data(0.33)
```

Multilayer Perceptron on Titanic Dataset

Building **Model**

```
EPOCHS = 250
LEARNING_RATE = 0.005
```

```
model = MLP_3(8)
```

```
class MLP_3(Module):
    def __init__(self, n_inputs):
        super(MLP_3, self).__init__()
        #1st layer input
        self.hidden1 = Linear(n_inputs, 24)
        kaiming_uniform_(self.hidden1.weight, nonlinearity='relu')
        self.act1 = ReLU()
        #2nd layer
        self.hidden2 = Linear(24, 40)
        kaiming_uniform_(self.hidden2.weight, nonlinearity='relu')
        self.act2 = ReLU()
        #3rd layer
        self.hidden3 = Linear(40, 32)
        kaiming_uniform_(self.hidden3.weight, nonlinearity='relu')
        self.act3 = ReLU()
        #4rd layer
        self.hidden4 = Linear(32, 16)
        kaiming_uniform_(self.hidden4.weight, nonlinearity='relu')
        self.act4 = ReLU()
        #5th layer
        self.hidden5 = Linear(16, 8)
        kaiming_uniform_(self.hidden5.weight, nonlinearity='relu')
        self.act5 = ReLU()
        #6th layer and output
        self.hidden6 = Linear(8, 3)
        xavier_uniform_(self.hidden6.weight)
        self.act6 = Softmax(dim=1)
```

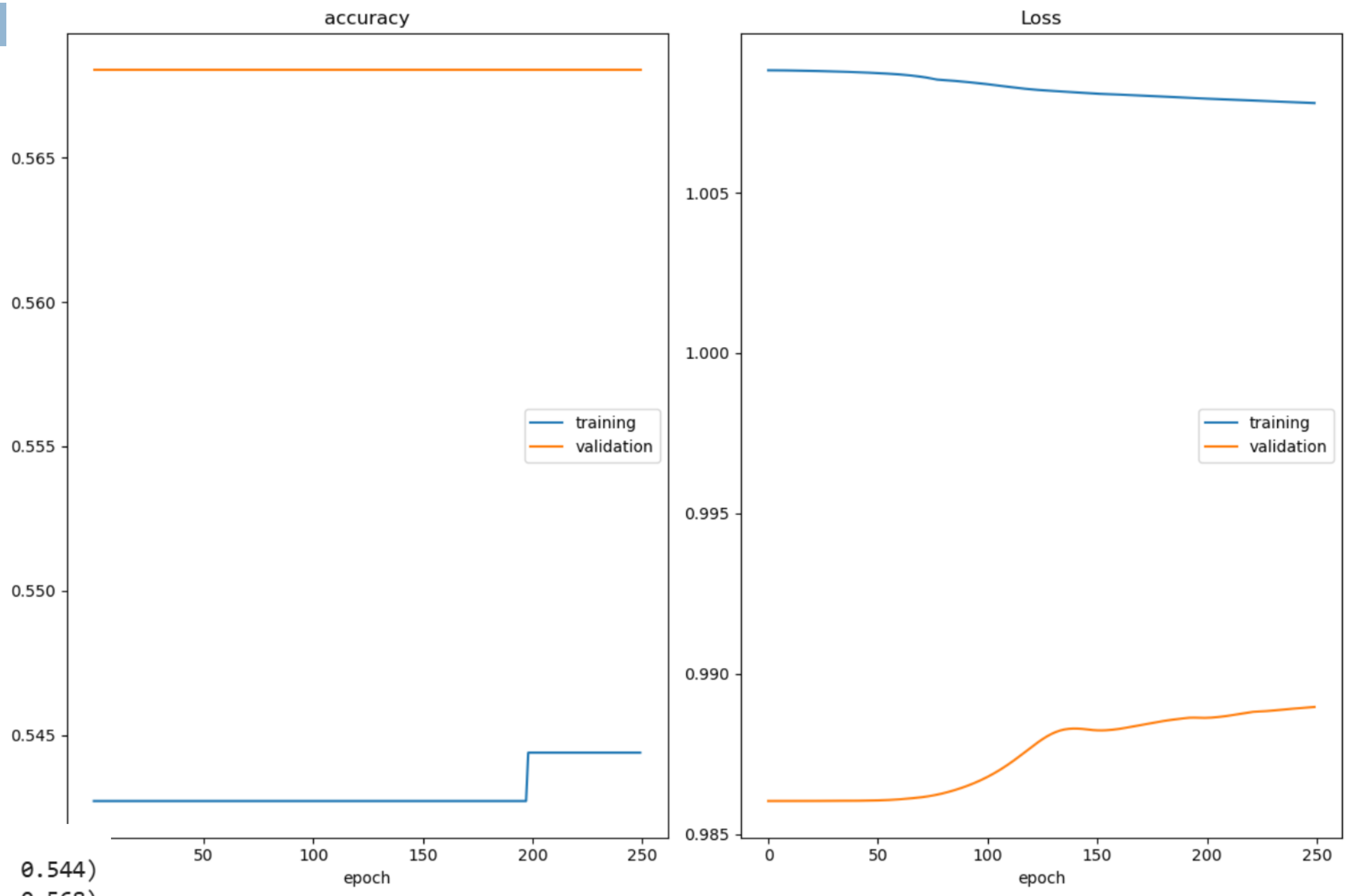
```
    def forward(self, X):
        X = self.hidden1(X)
        X = self.act1(X)
        X = self.hidden2(X)
        X = self.act2(X)
        X = self.hidden3(X)
        X = self.act3(X)
        X = self.hidden4(X)
        X = self.act4(X)
        X = self.hidden5(X)
        X = self.act5(X)
        X = self.hidden6(X)
        X = self.act6(X)
        return X
```

Multilayer Perceptron on Titanic Dataset


```
===============================================================================
Layer (type:depth-idx)                        Output Shape              Param #
===============================================================================
MLP_3                                         [597, 3]                  --
├─Linear: 1-1                                 [597, 24]                 216
├─ReLU: 1-2                                   [597, 24]                 --
├─Linear: 1-3                                 [597, 40]                 1,000
├─ReLU: 1-4                                   [597, 40]                 --
├─Linear: 1-5                                 [597, 32]                 1,312
├─ReLU: 1-6                                   [597, 32]                 --
├─Linear: 1-7                                 [597, 16]                 528
├─ReLU: 1-8                                   [597, 16]                 --
├─Linear: 1-9                                 [597, 8]                  136
├─ReLU: 1-10                                  [597, 8]                  --
├─Linear: 1-11                                [597, 3]                  27
├─Softmax: 1-12                               [597, 3]                  --
===============================================================================
Total params: 3,219
Trainable params: 3,219
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 1.92
===============================================================================
Input size (MB): 0.02
Forward/backward pass size (MB): 0.59
Params size (MB): 0.01
Estimated Total Size (MB): 0.62
===============================================================================
MLP_3(
  (hidden1): Linear(in_features=8, out_features=24, bias=True)
  (act1): ReLU()
  (hidden2): Linear(in_features=24, out_features=40, bias=True)
  (act2): ReLU()
  (hidden3): Linear(in_features=40, out_features=32, bias=True)
  (act3): ReLU()
  (hidden4): Linear(in_features=32, out_features=16, bias=True)
  (act4): ReLU()
  (hidden5): Linear(in_features=16, out_features=8, bias=True)
  (act5): ReLU()
  (hidden6): Linear(in_features=8, out_features=3, bias=True)
  (act6): Softmax(dim=1)
)
```

Multilayer Perceptron on Titanic Dataset

Live training results




```
accuracy
	training             (min:    0.543, max:    0.544, cur:    0.544)
	validation           (min:    0.568, max:    0.568, cur:    0.568)
Loss
	training             (min:    1.008, max:    1.009, cur:    1.008)
	validation           (min:    0.986, max:    0.989, cur:    0.989)
```

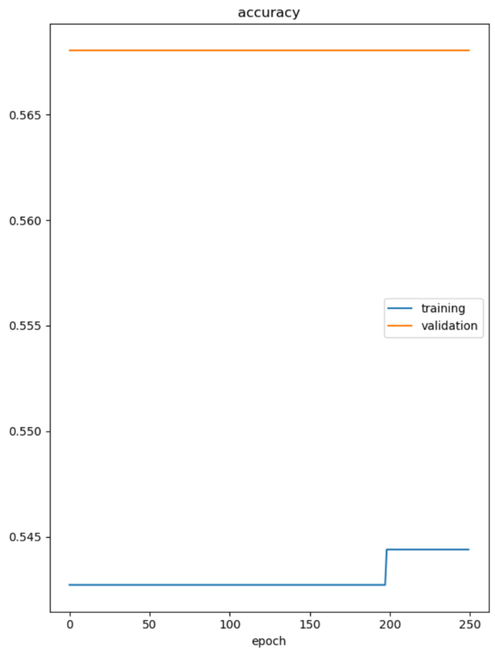
Multilayer Perceptron on Titanic Dataset

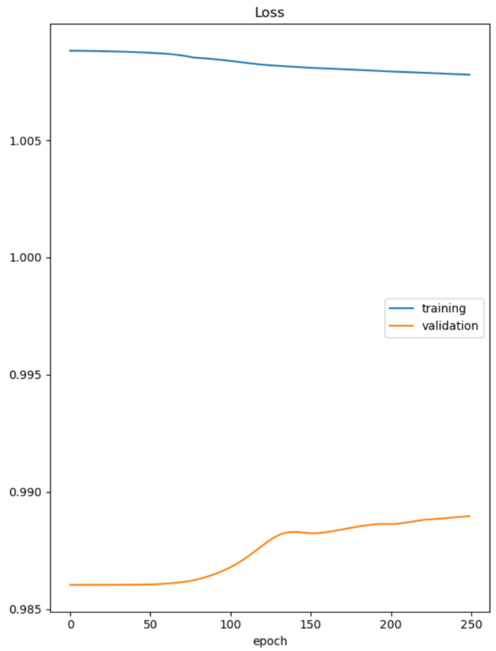
Evaluate the model

```
Accuracy: 0.568

success:167 failure:127

              precision    recall  f1-score   support

           0       0.00      0.00      0.00        63
           1       0.00      0.00      0.00        64
           2       0.57      1.00      0.72       167

    accuracy                           0.57       294
   macro avg       0.19      0.33      0.24       294
weighted avg       0.32      0.57      0.41       294
```

Multilayer Perceptron on Titanic Dataset

Model 3: ↑ 250 epochs, ↓ LR: 0.005, ↑ 6 layers – accuracy: 0.568 ↑






```
accuracy
	training           (min:    0.543, max:    0.544, cur:    0.544)
	validation         (min:    0.568, max:    0.568, cur:    0.568)
Loss
	training           (min:    1.008, max:    1.009, cur:    1.008)
	validation         (min:    0.986, max:    0.989, cur:    0.989)
```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	63
1	0.00	0.00	0.00	64
2	0.57	1.00	0.72	167
accuracy			0.57	294
macro avg	0.19	0.33	0.24	294
weighted avg	0.32	0.57	0.41	294

Multilayer Perceptron on Titanic Dataset

Let's create another model.

Model 4: 250 epochs, LR: 0.005,
6 layers – Adam as optimizer

```
EPOCHS = 250
LEARNING_RATE = 0.005
```

```
model = MLP_4(8)
```

```
class MLP_4(Module):
    def __init__(self, n_inputs):
        super(MLP_4, self).__init__()
        #1st Layer input
        self.hidden1 = Linear(n_inputs, 32)
        kaiming_uniform_(self.hidden1.weight, nonlinearity='relu')
        self.act1 = ReLU()
        #2nd Layer
        self.hidden2 = Linear(32, 40)
        kaiming_uniform_(self.hidden2.weight, nonlinearity='relu')
        self.act2 = ReLU()
        #3rd Layer
        self.hidden3 = Linear(40, 32)
        kaiming_uniform_(self.hidden3.weight, nonlinearity='relu')
        self.act3 = ReLU()
        #4rd Layer
        self.hidden4 = Linear(32, 16)
        kaiming_uniform_(self.hidden4.weight, nonlinearity='relu')
        self.act4 = ReLU()
        #5th Layer
        self.hidden5 = Linear(16, 8)
        kaiming_uniform_(self.hidden5.weight, nonlinearity='relu')
        self.act5 = ReLU()
        #6th Layer and output
        self.hidden6 = Linear(8, 3)
        xavier_uniform_(self.hidden6.weight)
        self.act6 = Softmax(dim=1)
```

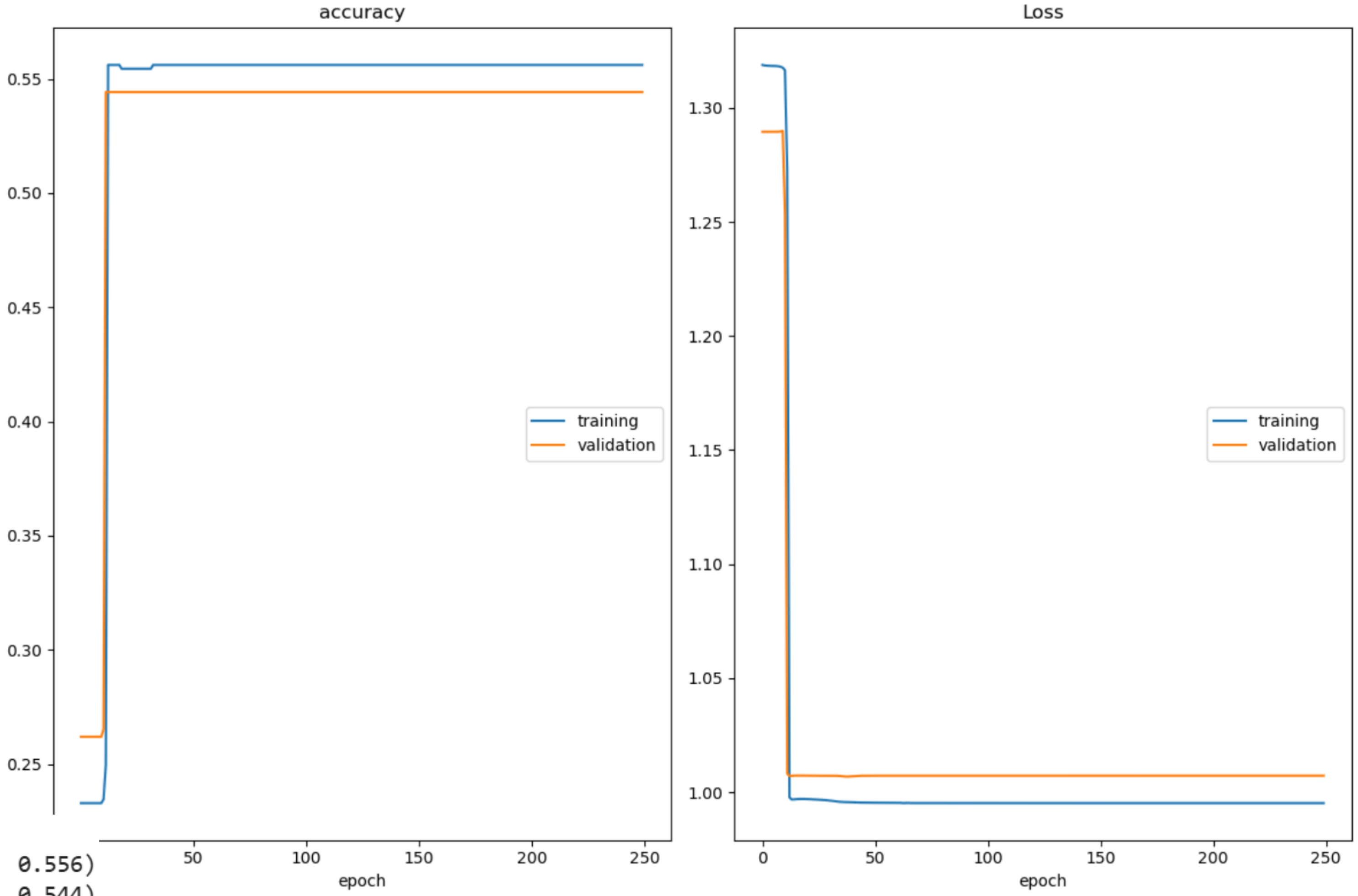
```
def forward(self, X):
    X = self.hidden1(X)
    X = self.act1(X)
    X = self.hidden2(X)
    X = self.act2(X)
    X = self.hidden3(X)
    X = self.act3(X)
    X = self.hidden4(X)
    X = self.act4(X)
    X = self.hidden5(X)
    X = self.act5(X)
    X = self.hidden6(X)
    X = self.act6(X)
    return X
```

Multilayer Perceptron on Titanic Dataset


```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
MLP_4                                    [597, 3]                  --
├─Linear: 1-1                            [597, 32]                 288
├─ReLU: 1-2                              [597, 32]                 --
├─Linear: 1-3                            [597, 40]                 1,320
├─ReLU: 1-4                              [597, 40]                 --
├─Linear: 1-5                            [597, 32]                 1,312
├─ReLU: 1-6                              [597, 32]                 --
├─Linear: 1-7                            [597, 16]                 528
├─ReLU: 1-8                              [597, 16]                 --
├─Linear: 1-9                            [597, 8]                  136
├─ReLU: 1-10                             [597, 8]                  --
├─Linear: 1-11                           [597, 3]                  27
├─Softmax: 1-12                          [597, 3]                  --
==========================================================================================
Total params: 3,611
Trainable params: 3,611
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 2.16
==========================================================================================
Input size (MB): 0.02
Forward/backward pass size (MB): 0.63
Params size (MB): 0.01
Estimated Total Size (MB): 0.66
==========================================================================================
MLP_4(
  (hidden1): Linear(in_features=8, out_features=32, bias=True)
  (act1): ReLU()
  (hidden2): Linear(in_features=32, out_features=40, bias=True)
  (act2): ReLU()
  (hidden3): Linear(in_features=40, out_features=32, bias=True)
  (act3): ReLU()
  (hidden4): Linear(in_features=32, out_features=16, bias=True)
  (act4): ReLU()
  (hidden5): Linear(in_features=16, out_features=8, bias=True)
  (act5): ReLU()
  (hidden6): Linear(in_features=8, out_features=3, bias=True)
  (act6): Softmax(dim=1)
)
```

Multilayer Perceptron on Titanic Dataset

Live training results




```
accuracy
        training                 (min:    0.233, max:    0.556, cur:    0.556)
        validation               (min:    0.262, max:    0.544, cur:    0.544)
Loss
        training                 (min:    0.995, max:    1.319, cur:    0.995)
        validation               (min:    1.007, max:    1.290, cur:    1.007)
```

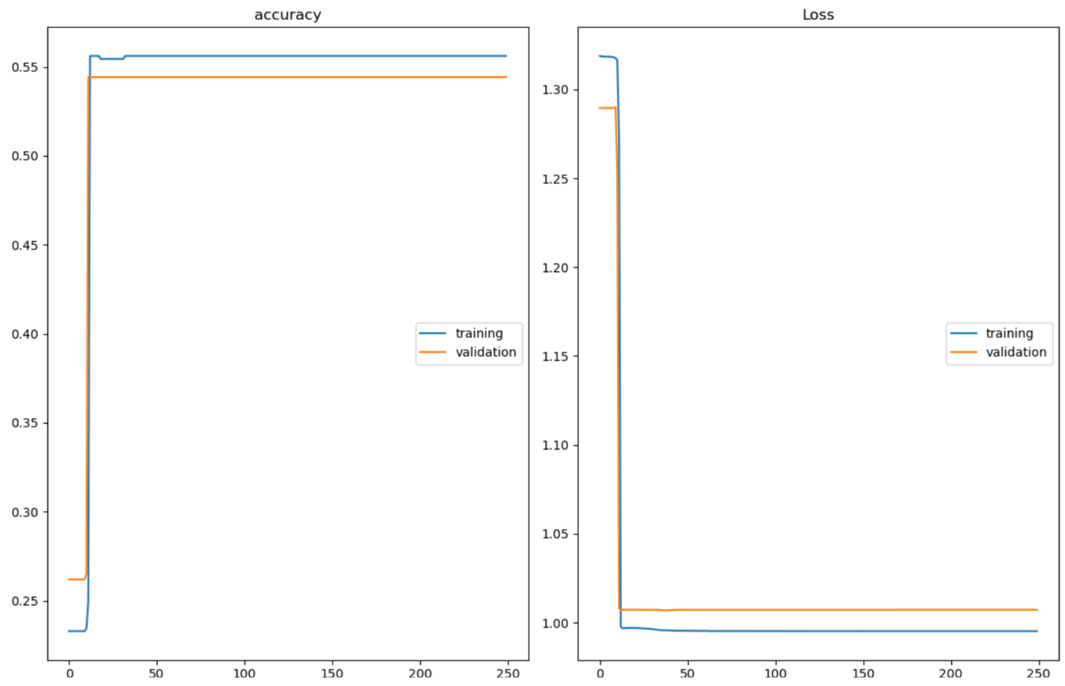
Multilayer Perceptron on Titanic Dataset

Evaluate the model

```
Accuracy: 0.544

success:160 failure:134
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        77
           1       0.00      0.00      0.00        57
           2       0.54      1.00      0.70       160

    accuracy                           0.54       294
   macro avg       0.18      0.33      0.23       294
weighted avg       0.30      0.54      0.38       294
```

Multilayer Perceptron on Titanic Dataset

Model 4: 250 epochs, LR: 0.005, 6 layers, Adam – accuracy: 0.544 ↓




```
accuracy
	training           (min:    0.233, max:    0.556, cur:    0.556)
	validation         (min:    0.262, max:    0.544, cur:    0.544)
Loss
	training           (min:    0.995, max:    1.319, cur:    0.995)
	validation         (min:    1.007, max:    1.290, cur:    1.007)
```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	77
1	0.00	0.00	0.00	57
2	0.54	1.00	0.70	160
accuracy			0.54	294
macro avg	0.18	0.33	0.23	294
weighted avg	0.30	0.54	0.38	294

Multilayer Perceptron on Titanic Dataset

- **Model 1:** 200 epochs, LR: 0.001, 3 layers – accuracy: 0.680
- **Model 2:** 200 epochs, LR: 0.01, 4 layers – accuracy: 0.224
- **Model 3:** 250 epochs, LR: 0.005, 6 layers – accuracy: 0.568
- **Model 4:** 250 epochs, LR: 0.005, 6 layers, Adam – accuracy: 0.544

Which model performed better?

What settings can be modified to improve the results?

Hands On