Università degli
Studi di Milano-Bicocca

Classification with Pytorch

Prof. Flavio Piccoli - Dr. Mirko Paolo Barbato

R&D process

- Analysis of the state of art
- Data collection / analysis
- Architectural Design
- Parameter Search
- Benchmarking
- Deployment

Google Scholar

pandas

scikit learn

PyTorch

PyTorch Lightning

RAY

tune

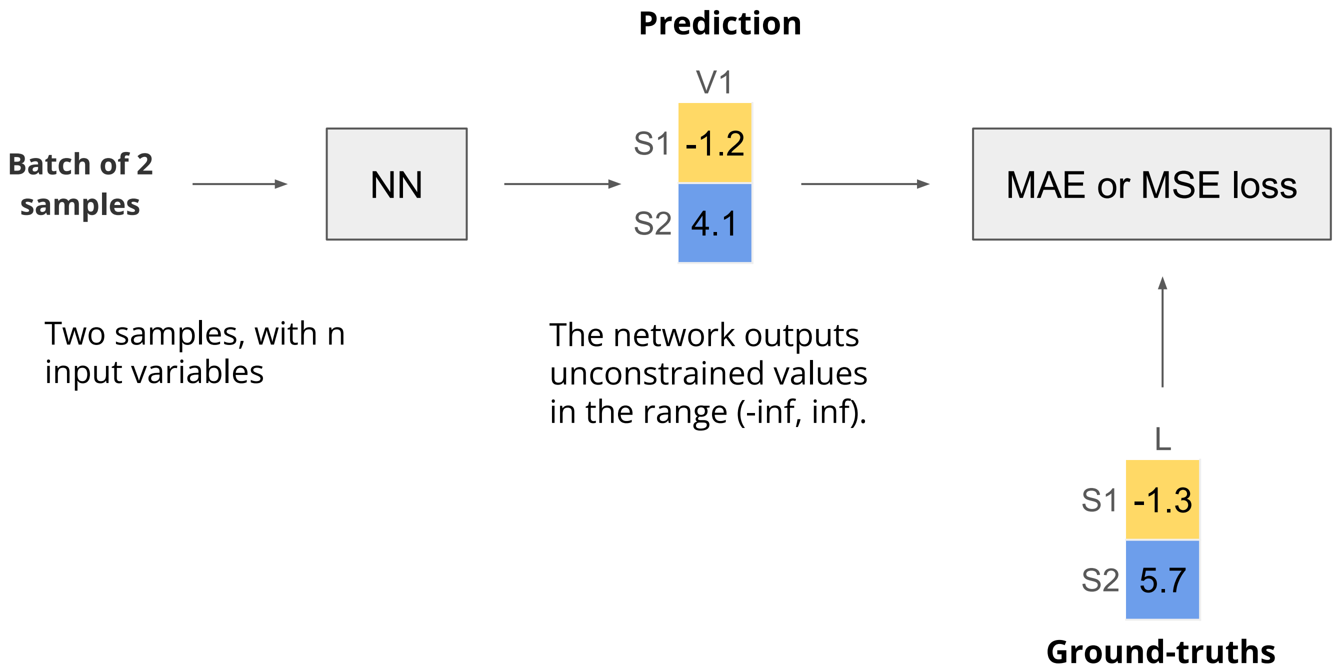
Streamlit

Flask

ONNX

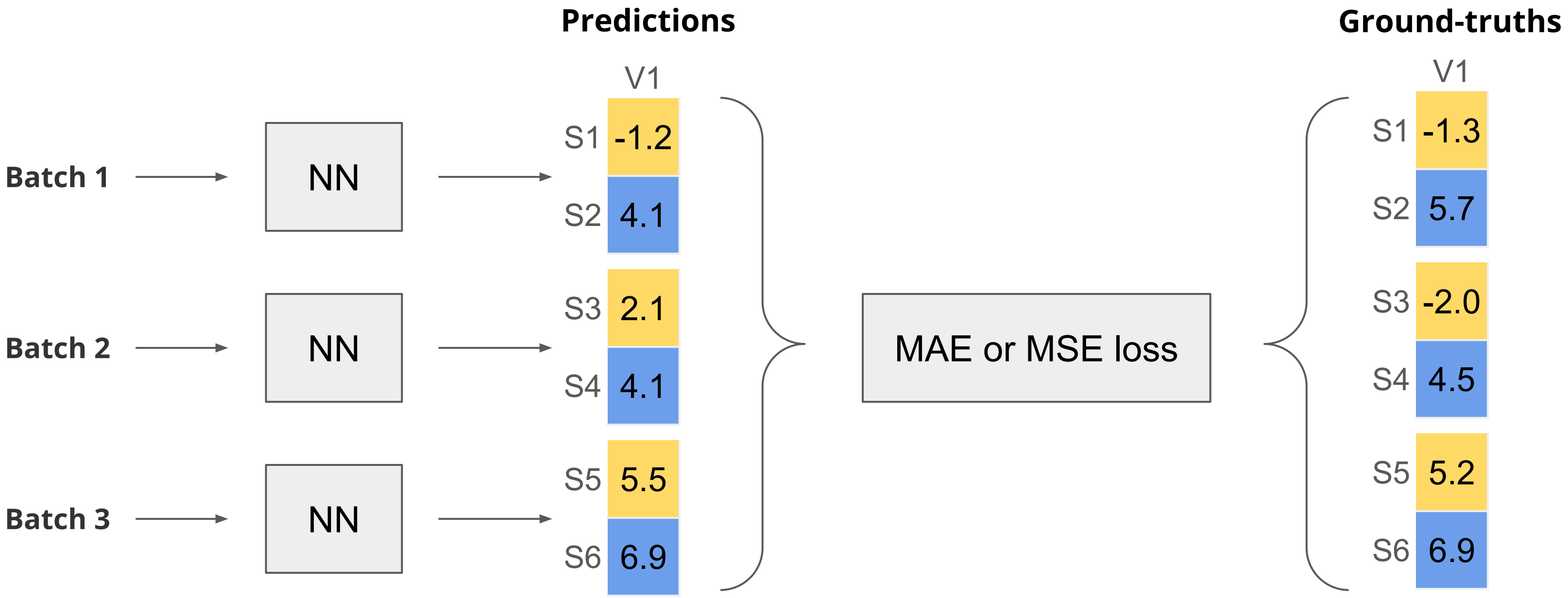
Regression setup

- network predicts directly the values of the continuous variable
- loss and performance score are: MAE or MSE



Validation / testing of the model

1. Set the network in eval mode (stops any randomness)
2. Predict the values of all the samples
3. Compute the MAE or MSE
4. Set network in training mode



Pipeline for classification

Suppose we are dealing with a classification task

- the target is a categorical variable
- there are three possible classes

Batch of 2 samples → NN → **Logits** → Softmax → **Probabilities** → Cross entropy loss ← **Ground-truths**

Logits

	C1	C2	C3
S1	-10	5	2
S2	4	-3	20

Probabilities

	C1	C2	C3
S1	0.1	0.6	0.3
S2	0.1	0.0	0.9

Ground-truths

	L
S1	0
S2	2

Two samples, with n input variables

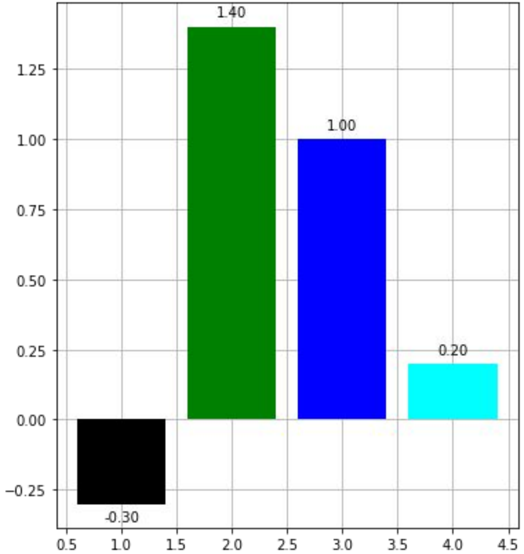
The network outputs unconstrained values in the range (-inf, inf).

Softmax converts the two unconstrained vectors in probability vectors

Softmax

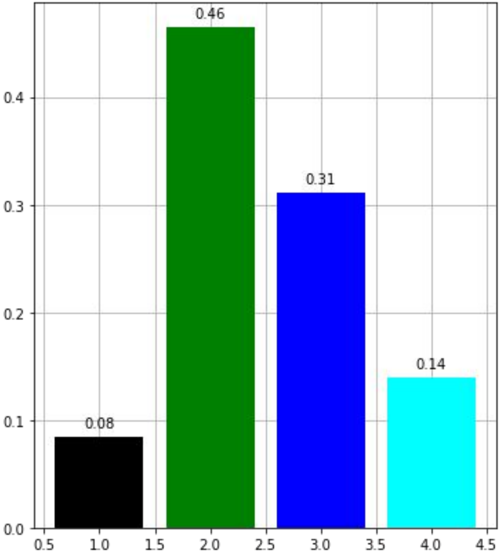
Converts an unconstrained vector in a probability vector

Logits



$$\sigma(y_i) = \frac{e^{y_i}}{\sum_{j=1}^{N} e^{y_j}}$$

Probabilities



$$y \rightarrow \begin{bmatrix} -.3 \\ 1.4 \\ 1.0 \\ 0.2 \end{bmatrix} \longrightarrow \begin{bmatrix} \dfrac{e^{y_i}}{\sum_j e^{y_j}} \end{bmatrix} \longrightarrow \begin{bmatrix} .08 \\ .46 \\ .31 \\ .14 \end{bmatrix} = 1$$

Cross-Entropy Loss

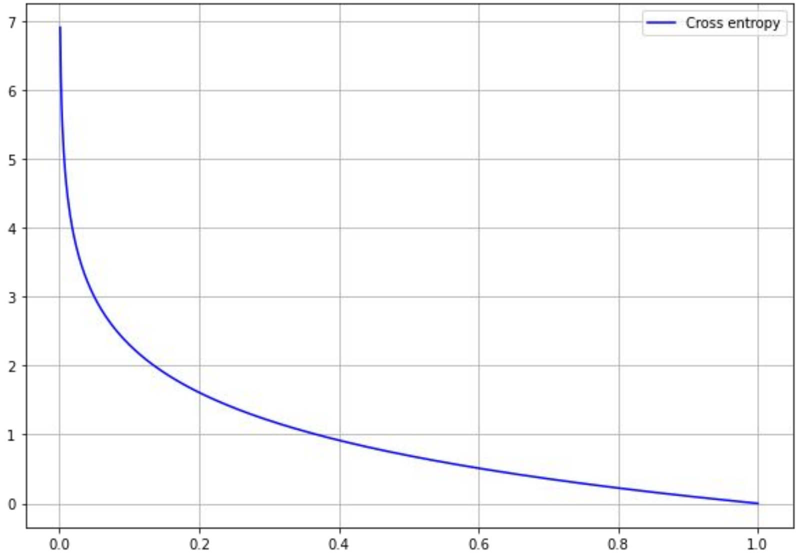
Used in classification tasks

$$L_{CE} = -\sum_{i=1}^{n} t_i log(p_i)$$

n = number of classes

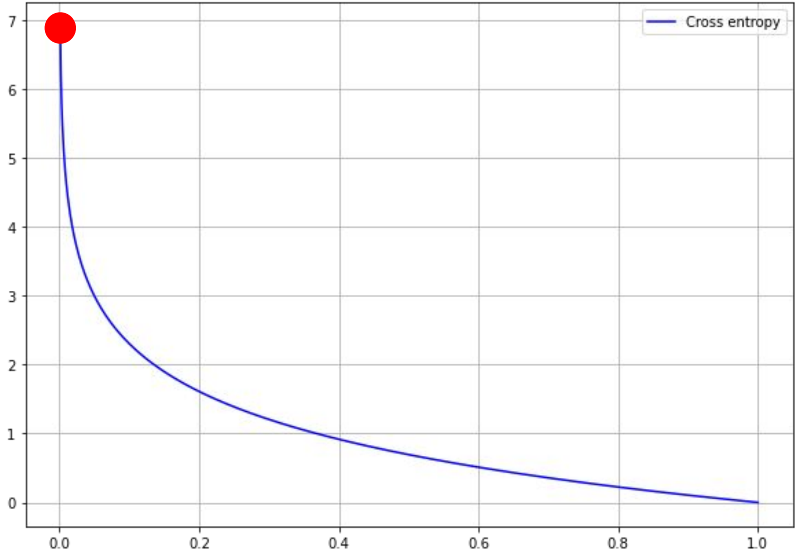
t_i = truth label [0,1]

p_i = softmax probability for i^{th} class



Cross-Entropy Loss

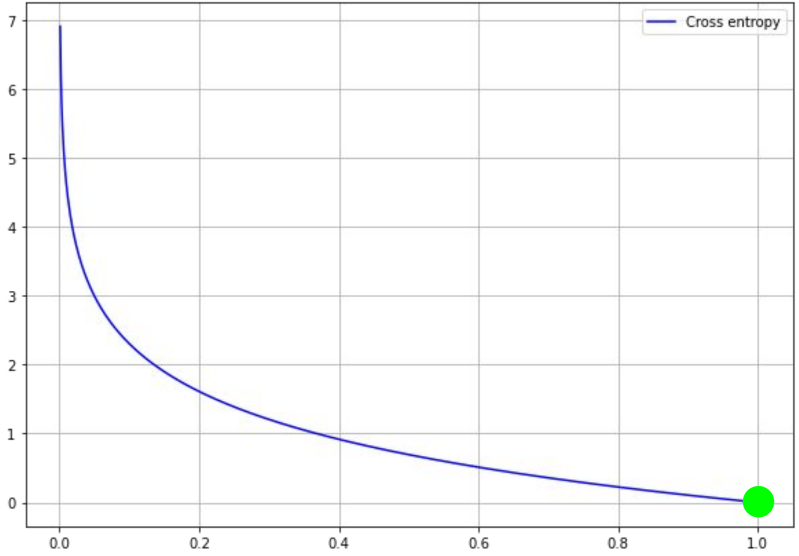
Graphical example



The predicted probability for the class is 0 but it should have been 1

Cross-Entropy Loss

Graphical example



The predicted probability for the class is 1 as it should have been. Loss = 0

Cross-Entropy Loss

An example

$$L_{CE} = -\sum_{i=1}^{n} t_i log(p_i)$$

Probabilities

	C1	C2	C3
S1	0.1	0.6	0.3
S2	0.1	0.0	0.9

Ground-truths

	L
S1	0
S2	2

Cross-Entropy Loss

Cross entropy loss for the first sample

$$L_{CE} = -\sum_{i=1}^{n} t_i log(p_i)$$

Probabilities

	C1	C2	C3
S1	$\boxed{0.1}$	0.6	0.3
S2	0.1	0.0	0.9

Ground-truths

	L
S1	$\boxed{0}$
S2	2

$$L_{CE}^{S_1} = -log(0.1) = 1$$

Cross-Entropy Loss

Cross entropy loss for the second sample

$$L_{CE} = -\sum_{i=1}^{n} t_i log(p_i)$$

Probabilities

	C1	C2	C3
S1	0.1	0.6	0.3
S2	0.1	0.0	$\boxed{0.9}$

Ground-truths

	L
S1	0
S2	$\boxed{2}$

$$L_{CE}^{S_2} = -log(0.9) = 0.05$$

Cross-Entropy Loss

$$L_{CE} = -\sum_{i=1}^{n} t_i log(p_i)$$

Cross entropy loss for the second sample

Probabilities

	C1	C2	C3
S1	0.1	0.6	0.3
S2	0.1	0.0	0.9

Ground-truths

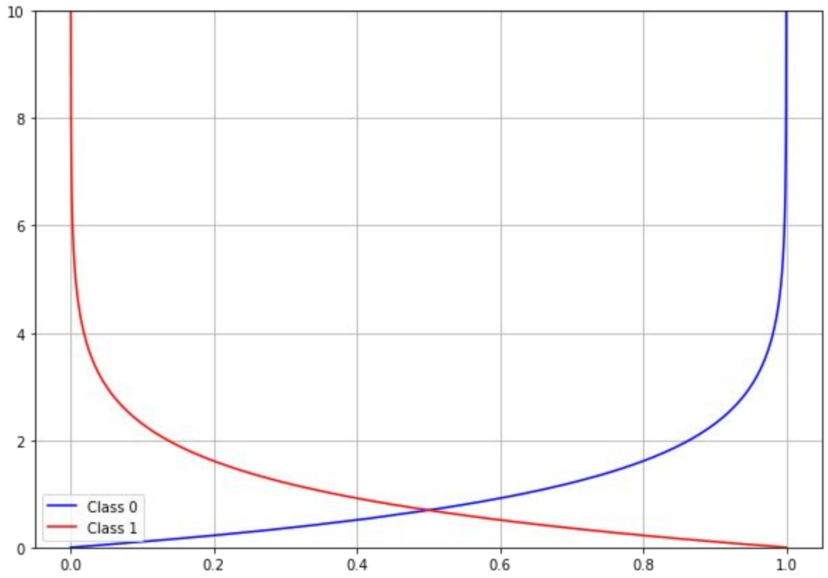
	L
S1	0
S2	2

$$L_{CE} = L_{CE}^{S_1} + L_{CE}^{S_2} = 1 + 0.05 = 1.05$$

Binary Cross-Entropy Loss

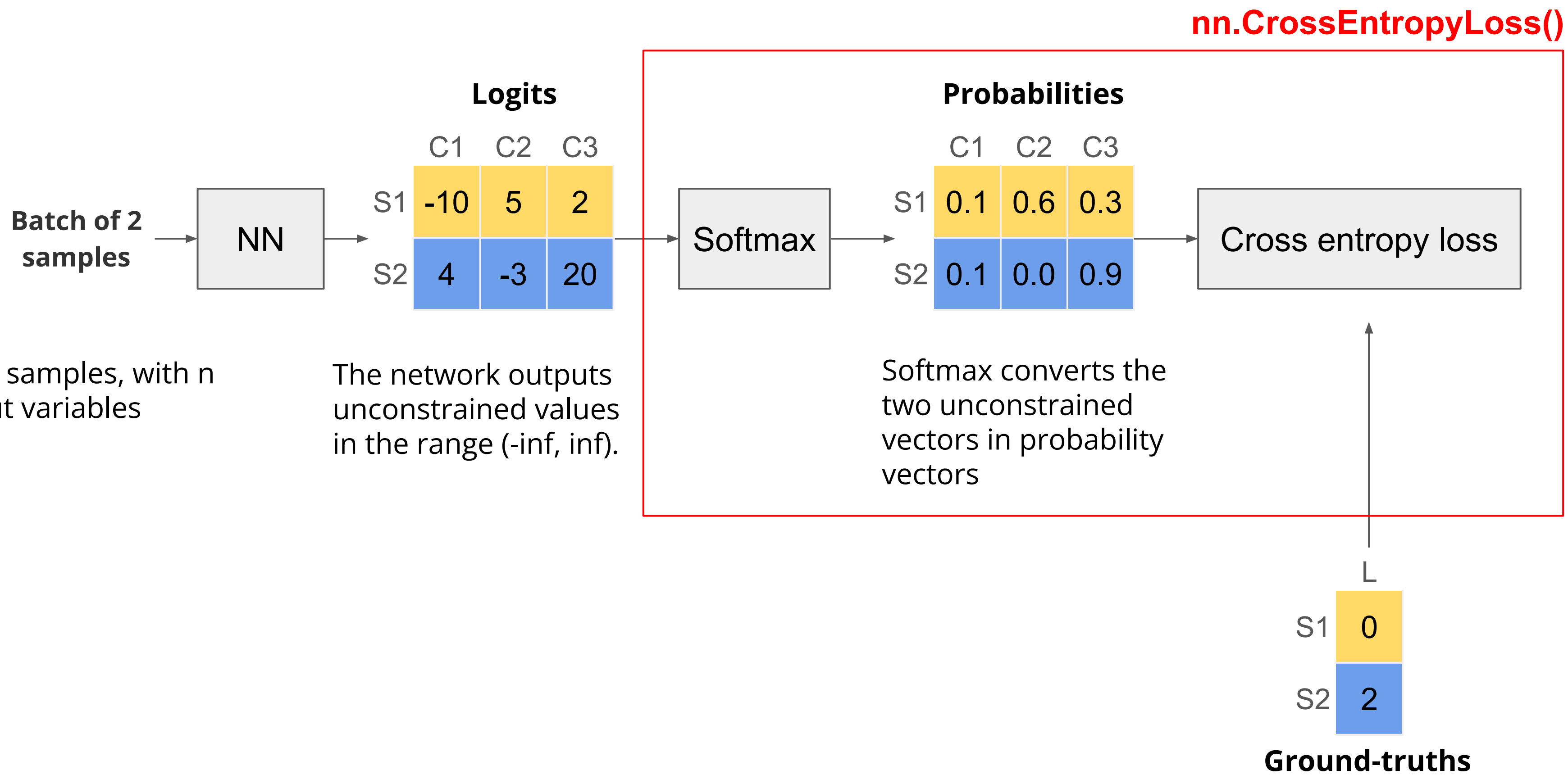
Easy computation of the cross entropy in case the classes are 2

$$L_{BCE} = -\frac{1}{N}\sum_{i=1}^{N} \boxed{y_i log(p(y_i))} + \boxed{(1-y_i)log(1-p(y_i))}$$



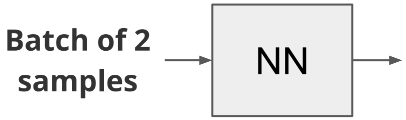
Pytorch implementation

In Pytorch, you can use directly nn.CrossEntropyLoss which combines softmax and cross entropy loss



So.. How do I get the predicted class in val/test phase?

In test phase we do not have vector probabilities but just logits

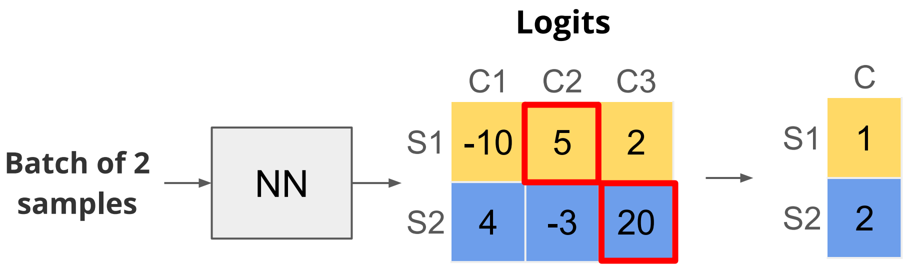


Logits

	C1	C2	C3
S1	-10	5	2
S2	4	-3	20

So.. How do I get the predicted class in val/test phase?

Very easy: the predicted class is the index of the maximum



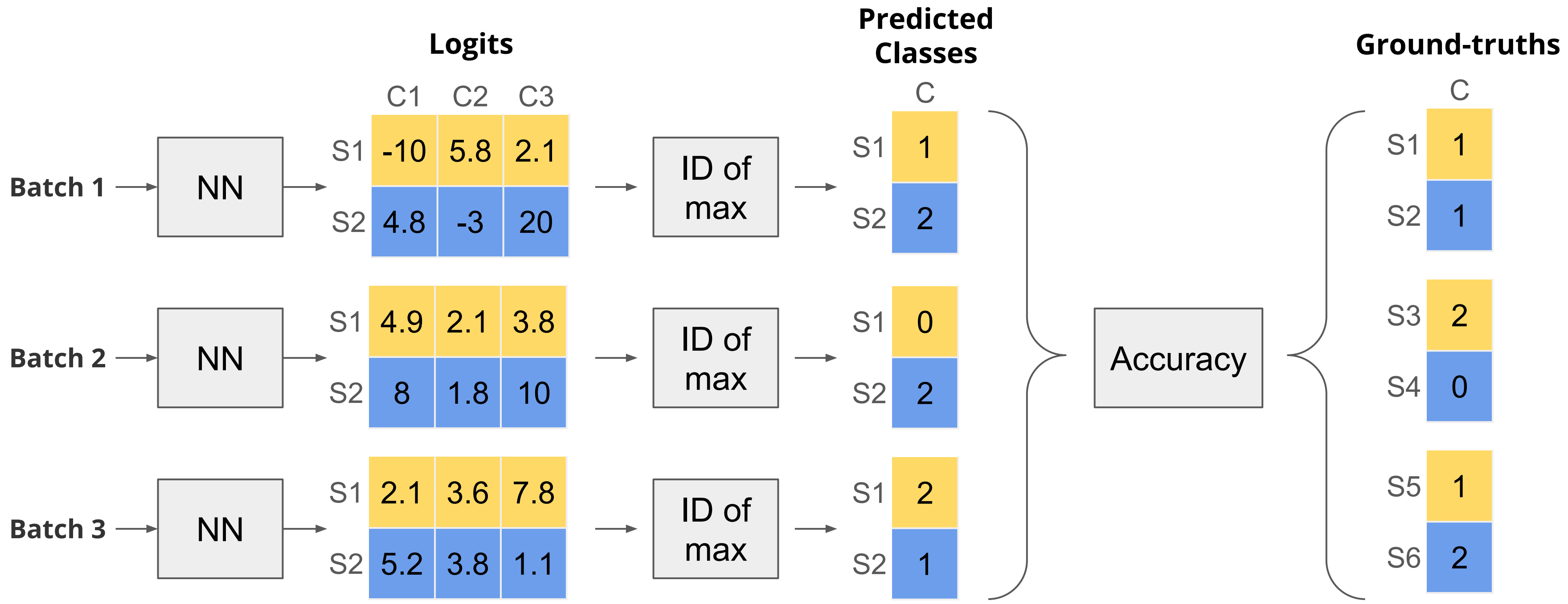

```python
# define the logits
logits = torch.tensor(
    [
        [-10, 5, 2],
        [4, -3, 20],
    ]
)

# let's find the maximum
vals, idx = torch.max(logits, axis=1)

# the predicted classes are the indexes of the max vals
print(idx)
```

How do we measure the performance of classification?

- Predict the class of every sample in every batch
- Compute accuracy



Accuracy

- The accuracy is:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

- In contrast to MAE, MSE, the higher the better

- How much is the accuracy in the example on the right?

	Predicted Classes	**Ground-truths**
	C	C
S1	1	1
S2	2	1
S3	0	0
S4	2	2
S5	2	2
S6	1	2

Accuracy

- The accuracy is:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

- In contrast to MAE, MSE, the higher the better

- How much is the accuracy in the example on the right?

66.66%

Correct predictions: 4
Total predictions: 6

- This metric is called "**macro accuracy**"

	Predicted Classes C	Ground-truths C
S1	1	1
S2	2	1
S3	0	0
S4	2	2
S5	2	2
S6	1	2

How to deal with imbalanced datasets?

- An imbalanced dataset is a data set with skewed class proportions
- Suppose to have the following situation:
  - a predictor that always predicts the same class
  - an imbalanced dataset
- How much is the accuracy?

Pred	GT
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	1
0	1

How to deal with imbalanced datasets?

- An imbalanced dataset is a data set with skewed class proportions
- Suppose to have the following situation:
  - a predictor that always predicts the same class
  - an imbalanced dataset
- How much is the accuracy?

80% !!!

Pred	GT
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	1
0	1

How to deal with imbalanced datasets?

Solution:

- compute accuracy weighted by the class cardinalities

$$\text{Accuracy} = \frac{1}{N}\sum_{c=1}^{N} \frac{\text{\# correct preds for class c}}{\text{\# samples of class c}}$$

- this metric is called “**Micro accuracy**”

Pred	GT
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	1
0	1

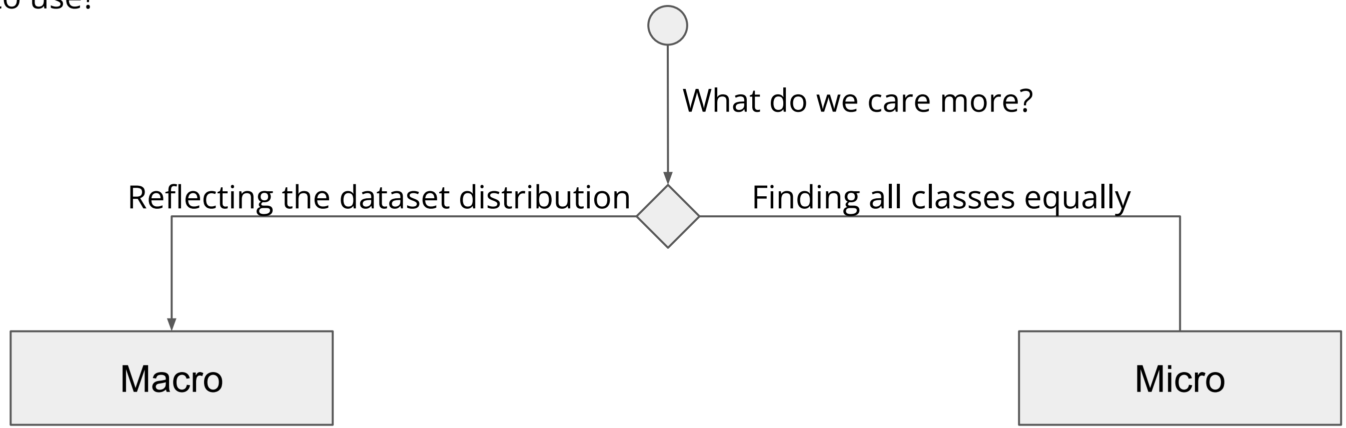
$Acc_0 = 1$

$Acc_1 = 0$

$Acc = 0.5$

Micro vs Macro Accuracy

- Which one to use?



- Suitable for many tasks
- Reflecting dataset distribution is often preferred

- Suitable for
  - anomaly detection tasks
  - tasks where all classes count equal

Torchmetrics

- Pytorch offers a library that simplifies the computation of the scores
- Documentation can be found at the link:

torchmetrics.readthedocs.io

- To install it:

pip install torchmetrics

Torchmetrics

- Usage

```
import torchmetrics

# define input and ground truth
inp = torch.tensor([0,0,0,0,0,0,0,0,0,0])
gt  = torch.tensor([0,0,0,0,0,0,0,0,1,1])

# define metric objects
acc_micro = torchmetrics.Accuracy(task = 'multiclass', num_classes = 2, average = 'micro')
acc_macro = torchmetrics.Accuracy(task = 'multiclass', num_classes = 2, average = 'macro')

# update metrics
acc_micro.update(inp, gt)
acc_macro.update(inp, gt)

# you can update the metrics with more batches ..

# at the end, compute the final score
micro = acc_micro.compute()
macro = acc_macro.compute()

# print
print(f'Micro accuracy is {micro:0.2f} while macro accuracy is {macro:0.2f}')

# reset the metric object (optional)
acc_micro.reset()
acc_macro.reset()
```

It will print: “Micro accuracy is 0.80 while macro accuracy is 0.50”

Torchmetrics

```
import torchmetrics

# define input and ground truth
inp = torch.tensor([0,0,0,0,0,0,0,0,0,0])
gt  = torch.tensor([0,0,0,0,0,0,0,0,1,1])

# define metric objects
acc_micro = torchmetrics.Accuracy(task = 'multiclass', num_classes = 2, average = 'micro')
acc_macro = torchmetrics.Accuracy(task = 'multiclass', num_classes = 2, average = 'macro')
```

Initialization

```
# update metrics
acc_micro.update(inp, gt)
acc_macro.update(inp, gt)

# you can update the metrics with more batches ..
```

Update of the metric.
One update for each batch.

```
# at the end, compute the final score
micro = acc_micro.compute()
macro = acc_macro.compute()
```

Final computation of the metric

```
# print
print(f'Micro accuracy is {micro:0.2f} while macro accuracy is {macro:0.2f}')

# reset the metric object (optional)
acc_micro.reset()
acc_macro.reset()
```

Reset of the metric

It will print: “Micro accuracy is 0.80 while macro accuracy is 0.50”

Exercises

Exercise 1 - manual evaluation of regression task

- Given the network of the previous exercise

    1. load the weights of the best model
    2. Set the network in evaluation mode (*net.eval()*)
    3. predict the estimation of the uber fare for each sample of the testset
        a. remember to accumulate all the estimations and ground truths
    4. compute the MAE and the MSE

Exercise 2 - evaluation of regression task with torchmetrics

- Perform the same task of exercise 1 but use torchmetrics

Exercise 3 - classification

A dataset contains tree observations from four areas of the Roosevelt National Forest in Colorado.

- All observations are cartographic variables (no remote sensing) from 30 meter x 30 meter sections of forest.
- There are ~ half a million measurements in total

- Is it possibile to build a model that predicts what types of trees grow in an area based on the surrounding characteristics?

- Download the dataset from elearning.
  - The dataset is already normalized.
  - The target variable is named “Cover_Type”
  - There are 7 classes

- Create all the code to perform training, validation and test.
- Use torchmetrics both for computing accuracy and for defining the confusion matrix