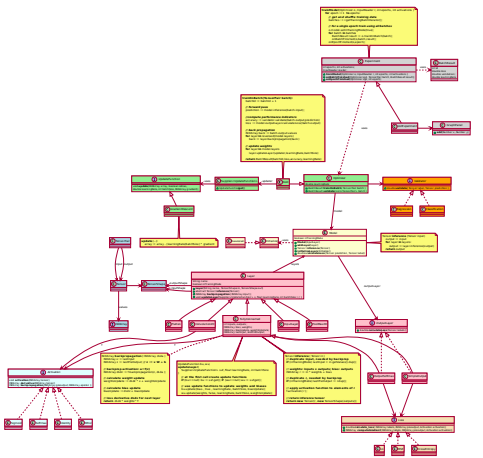
2ID90-DL – package

Huub van de Wetering
TU/e, 2ID90
2018-2019-Q3



Overview

- Introduction
- Sample Usage
- UML diagrams

Introduction 2ID90-DL

- package for handling Artificial neural networks
- support for
  - Fully connected networks
  - Convolutional networks
  - Backpropagation
  - Gradient Descent
- GUI
  - showing details of the learning process
- Based on Nd4j
  - an optimized library for handling high-dimensional vectors, matrices, tensor, …
  - based on BLAS (Basic Linear Algebra Subprograms)

Sample Usage - 1

MyExperiment.java

READ DATA

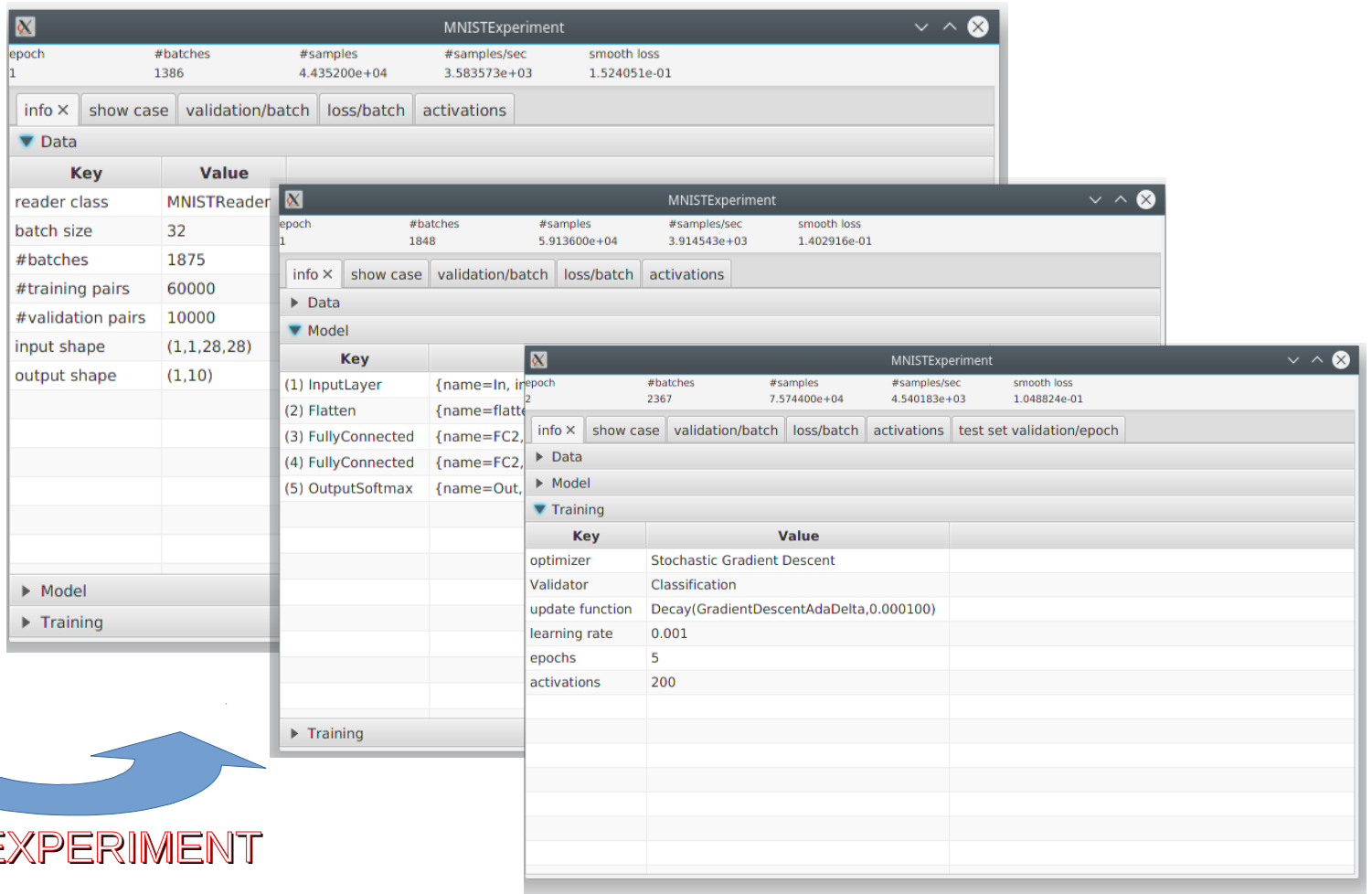
CREATE MODEL

INITIALIZE SGD

START TRAINING

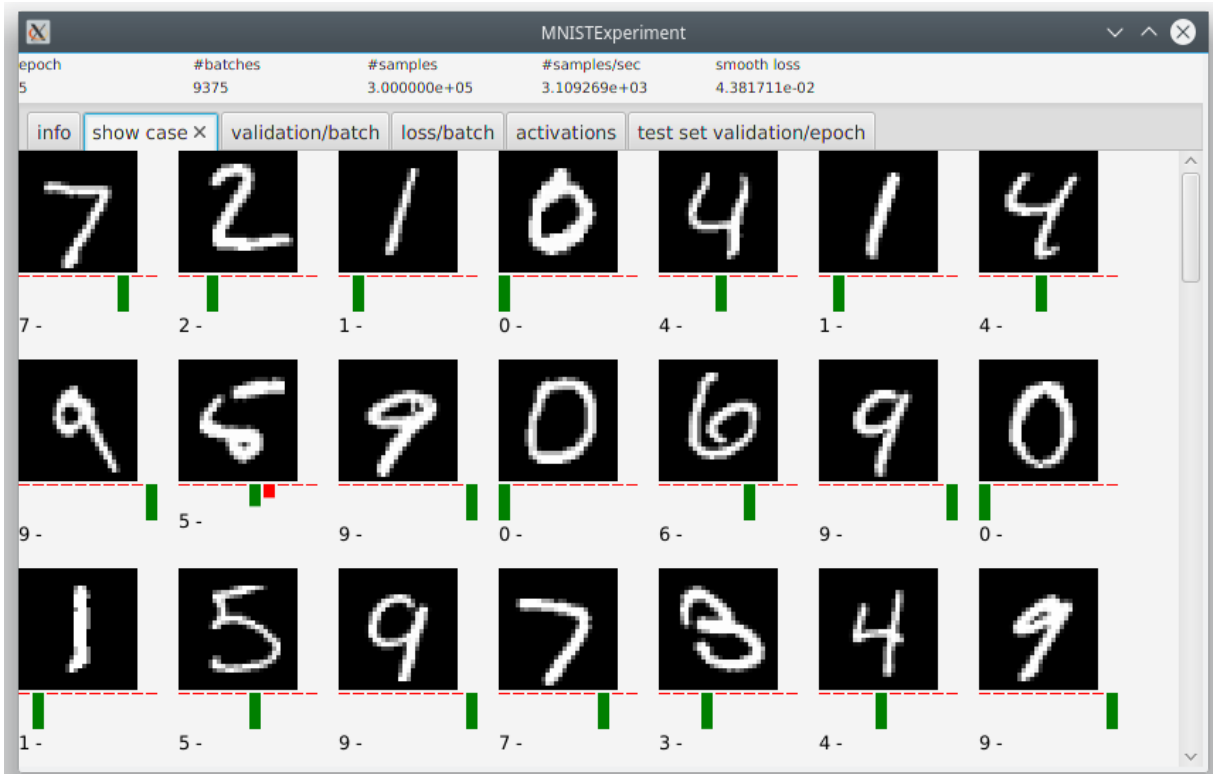
```
int outputs = reader.getOutputShape().getNeuronC
Model model = createModel(inputs, outputs);
                                    model);

// Training: create and configure SGD && train m
Optimizer sgd = SGD.builder()
        .model(model)
        .learningRate(reg.getLearningRate)
        .validation(new Regression())
        .updateFunction(() -> new Decay(Gradient
        .build();
System.out.println("\nOptimizer info:\n" + sgd.t
```

```
Model createModel(int inputs, int outputs) {
    InputLayer input = new InputLayer("In", new Tens
    Model model = new Model(input);
    model.addLayer(new FullyConnected("fc1", new Ten
    model.addLayer(new FullyConnected("fc2", new Ten
    model.addLayer(new FullyConnected("fc3", new Ten
    model.addLayer(new FullyConnected("fc4", new Ten
    model.addLayer(new FullyConnected("fc5", new Ten
    model.addLayer(new FullyConnected("fc6", new Ten
    model.addLayer(new SimpleOutput("Out", new Tenso
    return model;
```



RUN EXPERIMENT

Sample Usage - 2

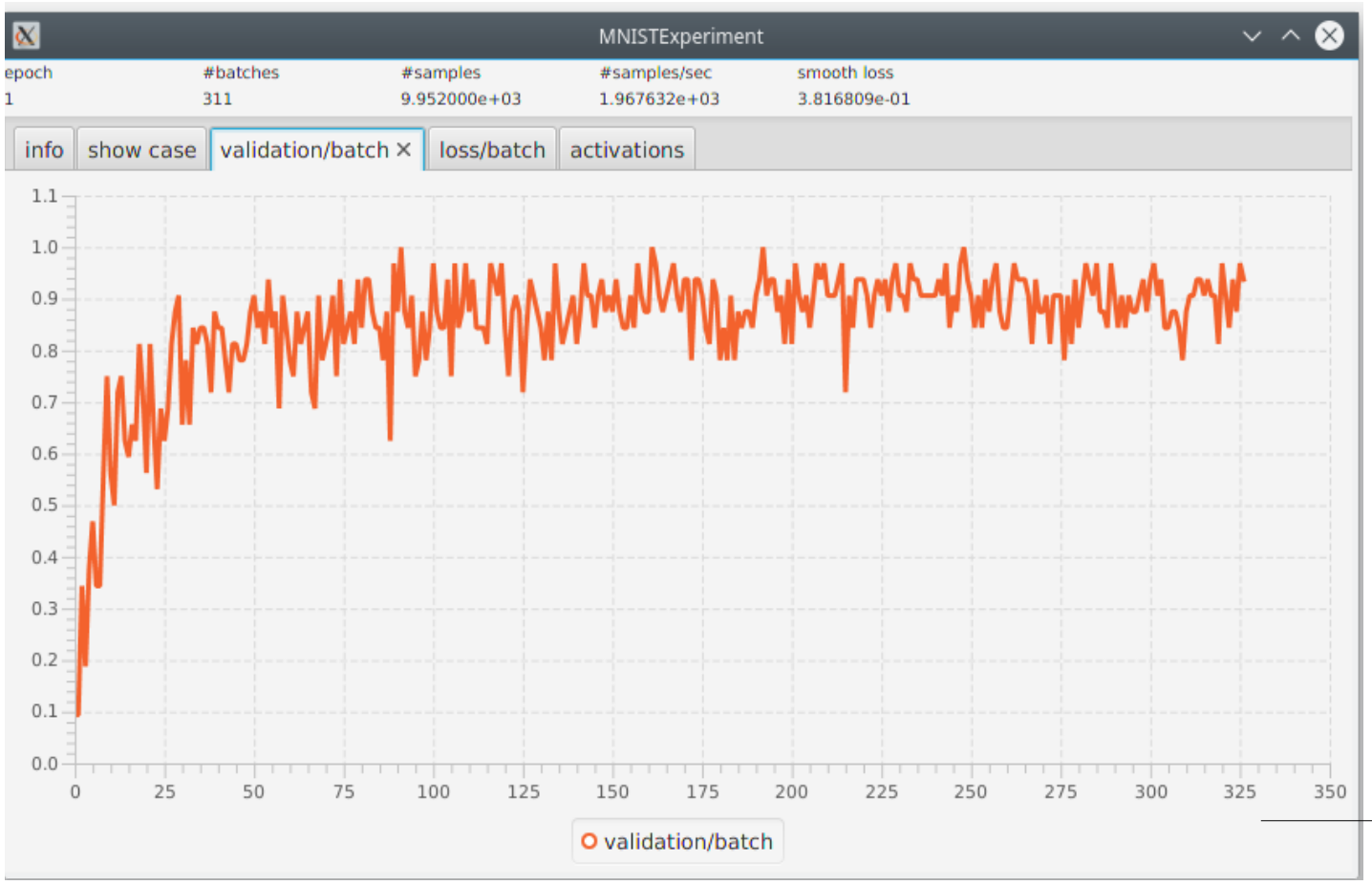


Show case: shows samples from the test data annotated with the current network's classification.

All samples are correctly classified, but the network hesitates whether one of the fives is actually a six.

Sample Usage - 2

Validation

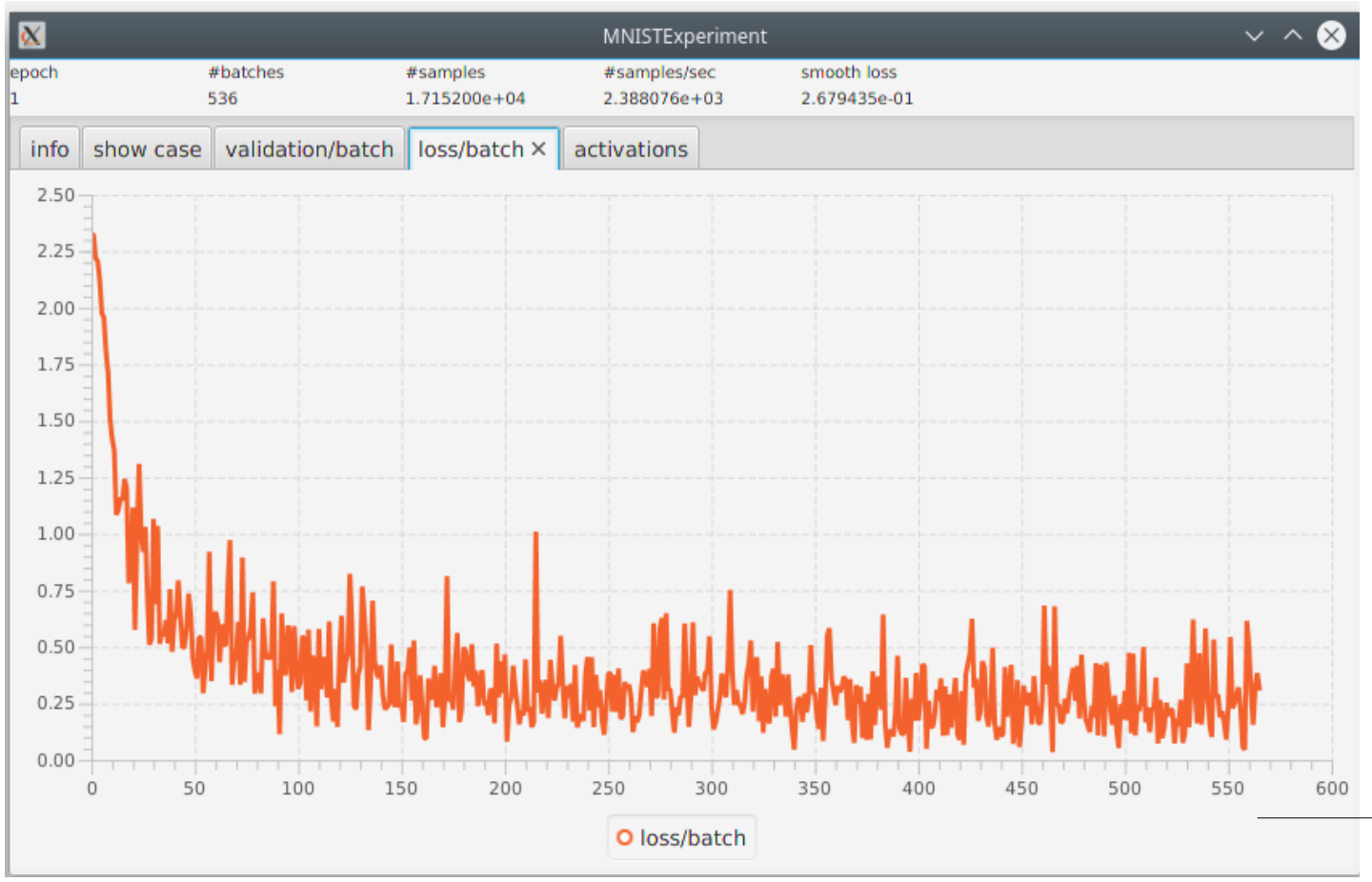


Validation per batch: here fraction of correctly classified samples in the batch.

Batch Id

Sample Usage - 3

Loss

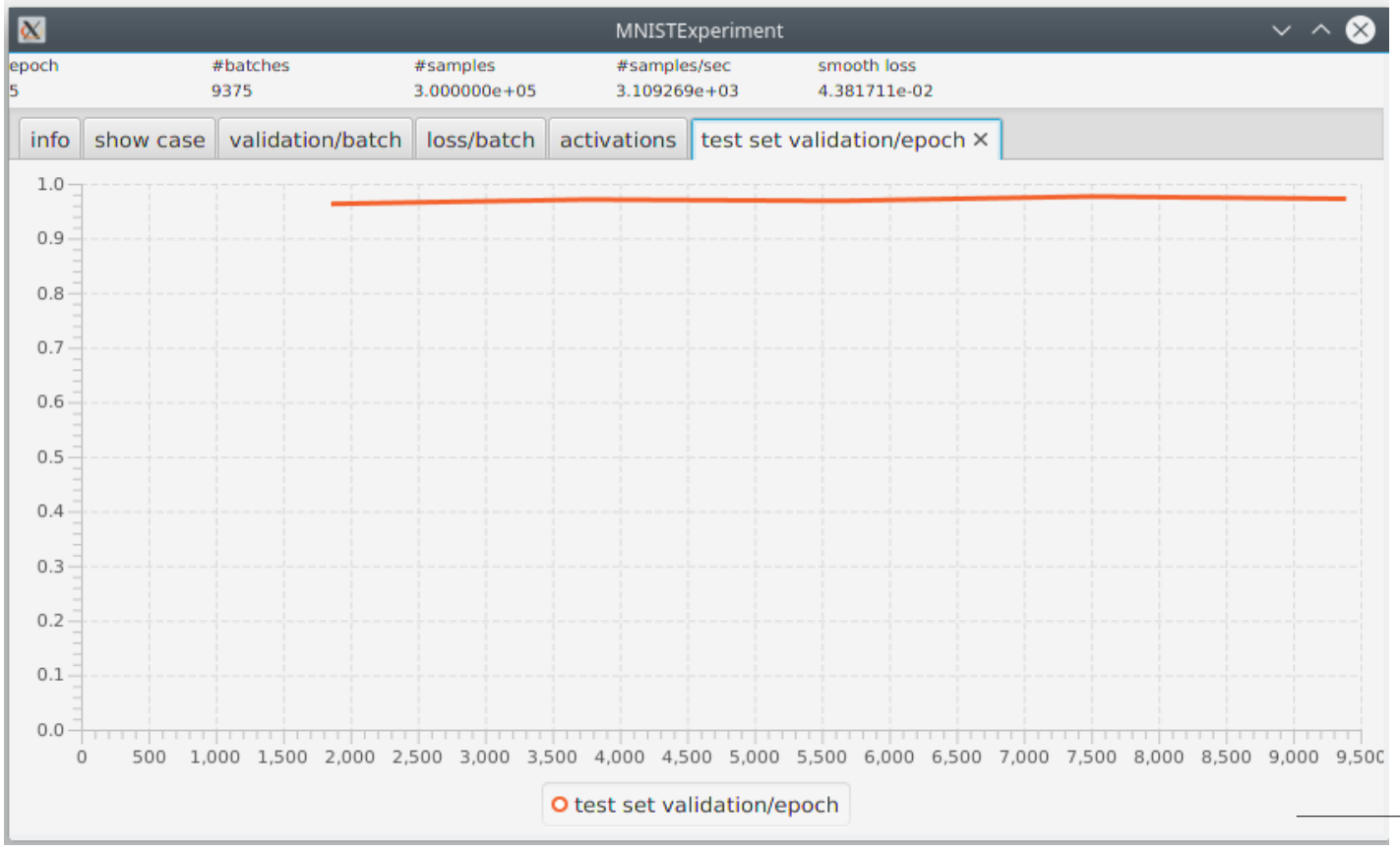


Loss per batch:
here the average error made in classifying the samples in a batch.

Batch Id

Sample Usage - 4

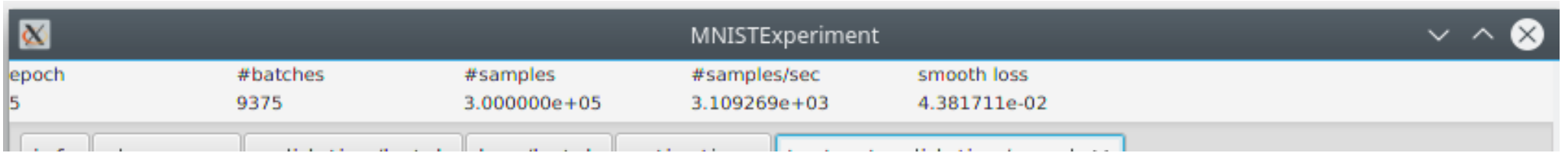
Validation



Test set validation per epoch
- computed at end of epoch
- validation based on test data
- here: percentage correctly classified

Batch Id

Sample Usage - 5

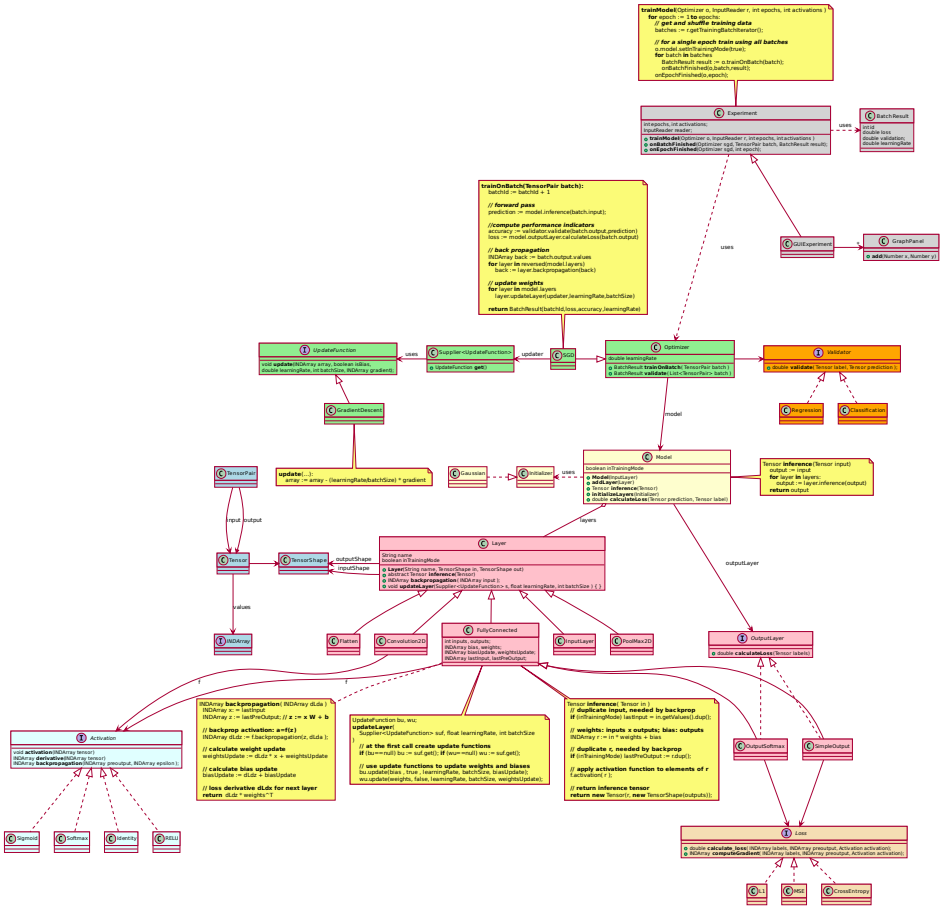


Current status of the training:

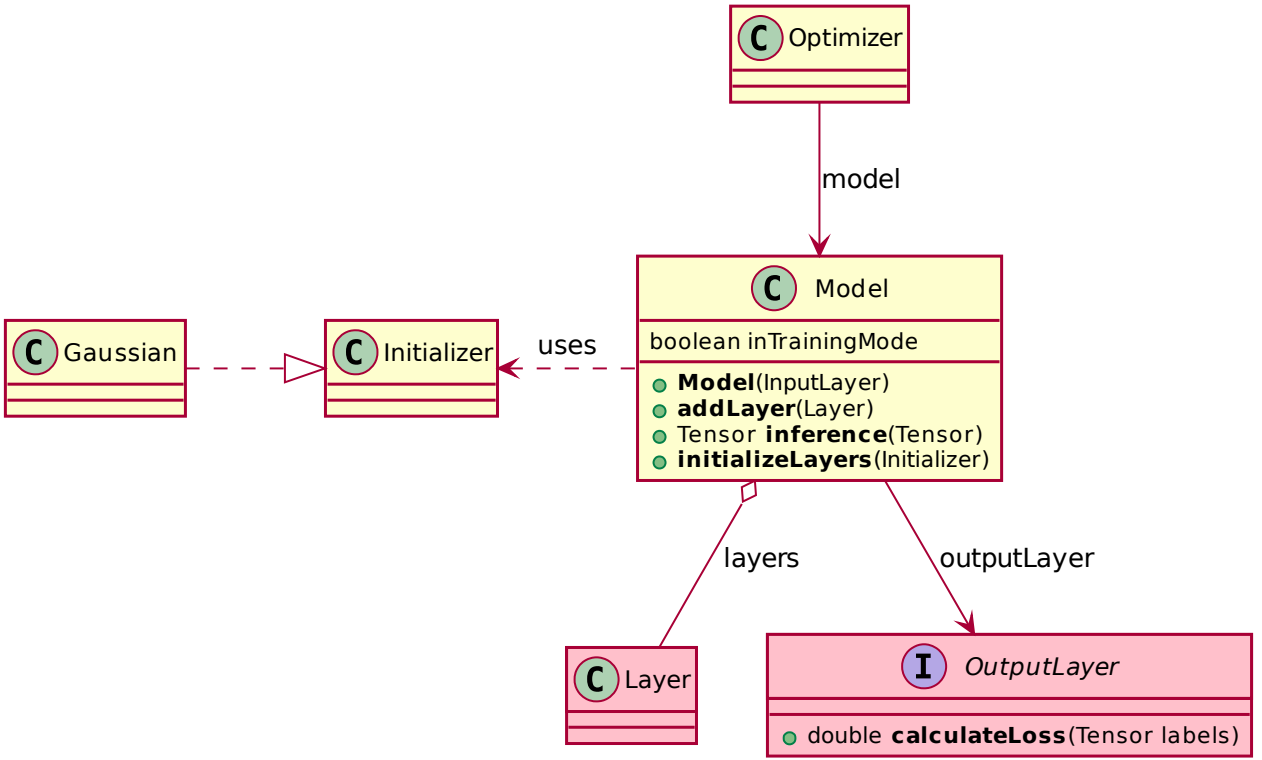
- current epoch
- total number of processed batches
- total number of processed samples = #batches * batchsize
- #samples/second : gives an indication of efficiency of network
- smooth loss is updated after each batch according to

  smoothLoss := 0.99 * smoothLoss + 0.01 * lossOfLastbatch

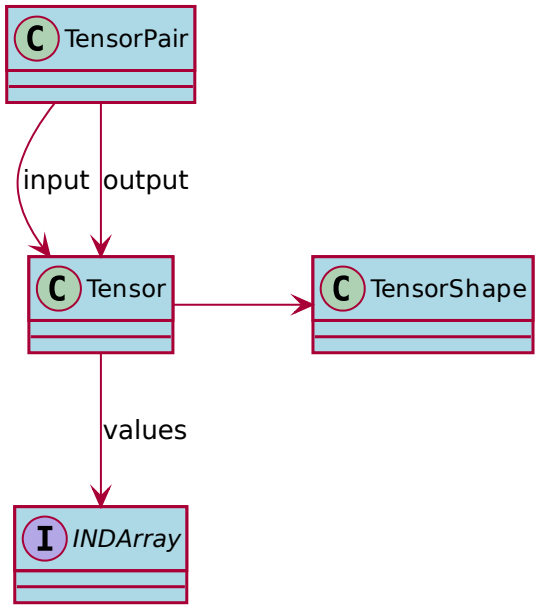
UML - Overview



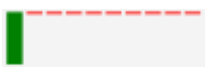
Network Model



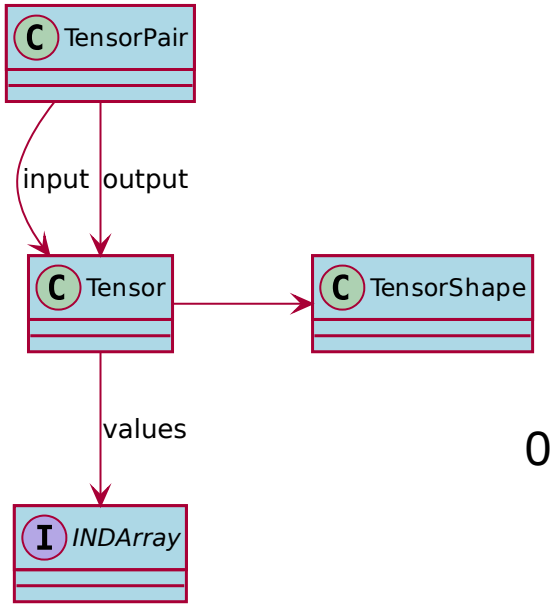
Tensors



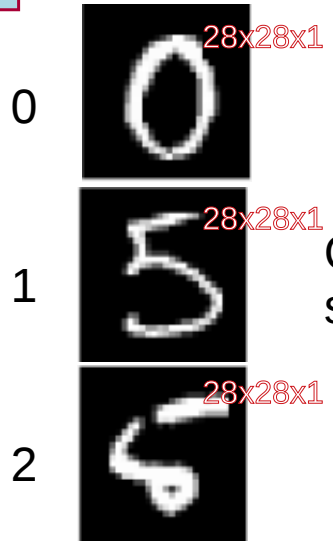
- Input network: List of TensorPairs
- TensorPair: (input,output)
- Tensor has a shape, e.g.
  - input shape: **28x28x1** (image)
    - greyscale image of size 28x28
  - output shape: **10**
    - Probability distribution over ten possible class values (0 ..9)



Tensors



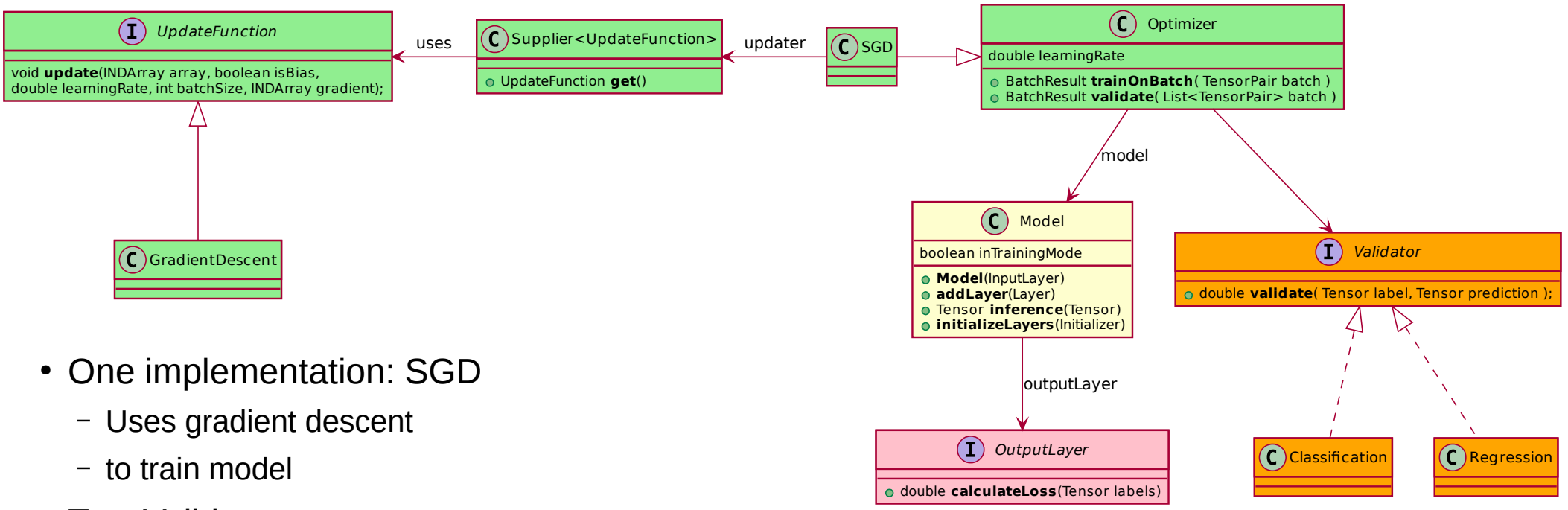
- Input shape of a network: **nx28x28x1**
  - n greyscale images of size 28x28
- Output shape of a network: **nx10**
  - n probability distributions
- Network then maps: **nx28x28x1** Tensor on **nx10** Tensor



One tensor with shape 3x28x28x1.

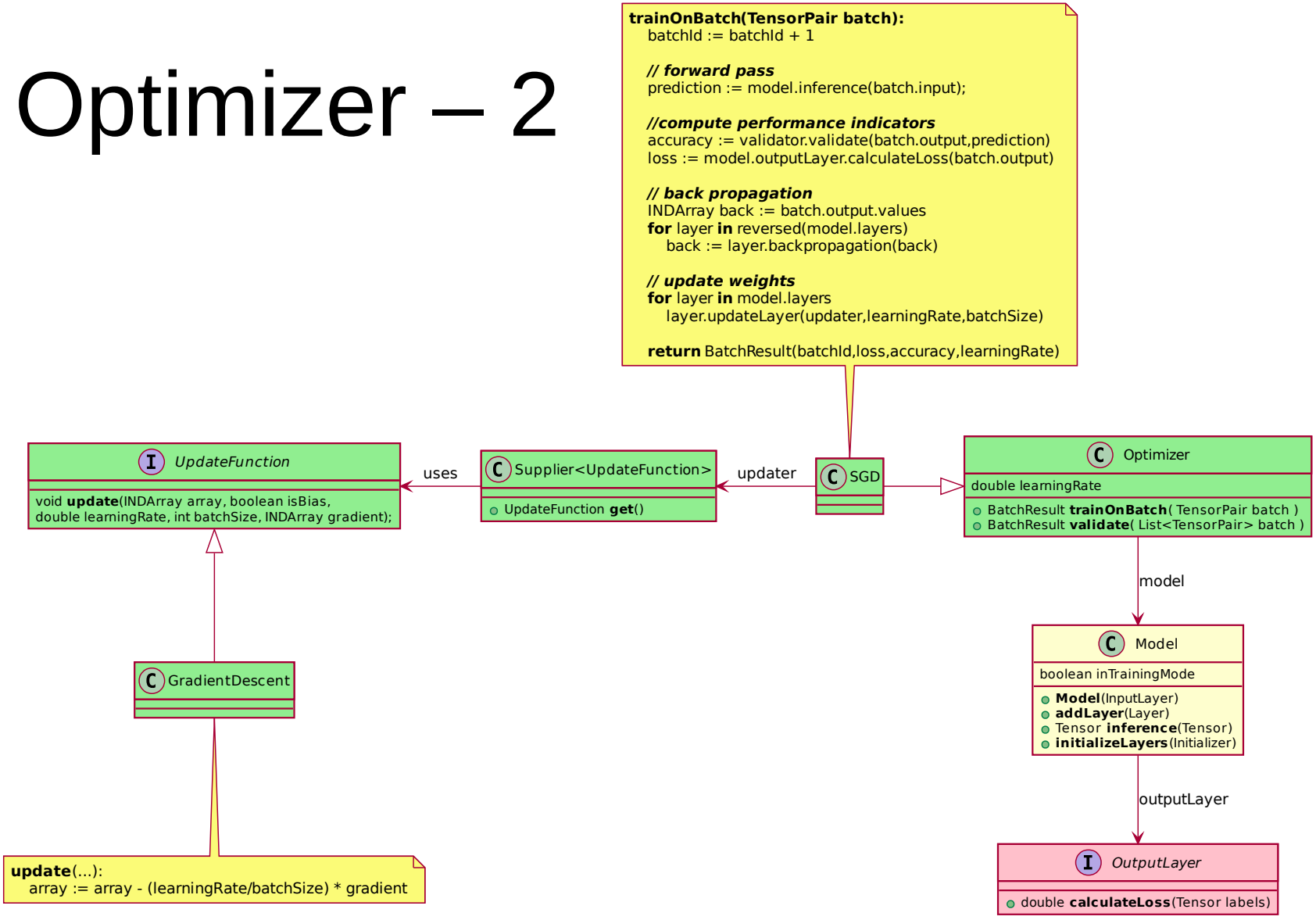
One tensor with shape 3x10.

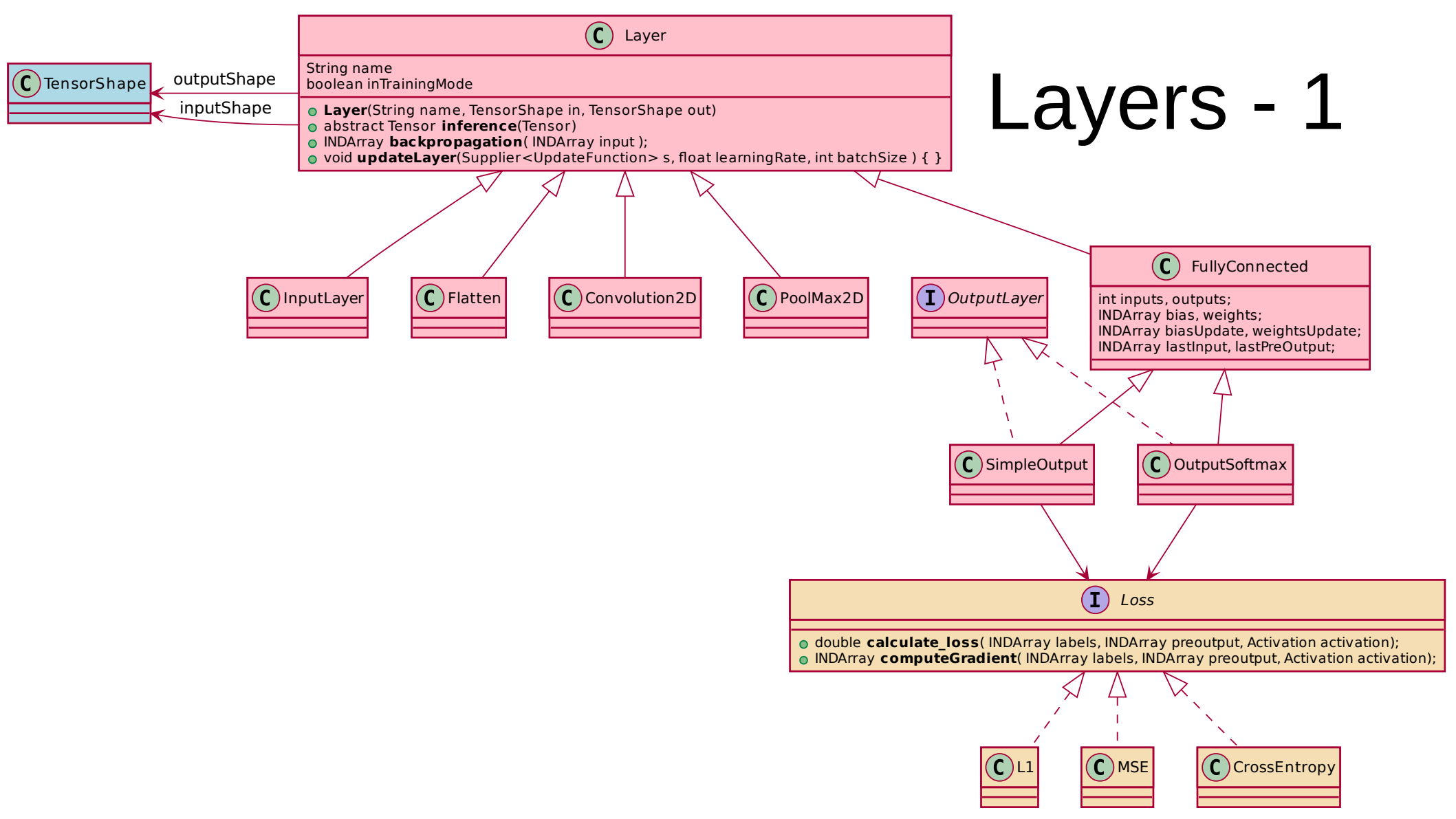
Optimizer - 1

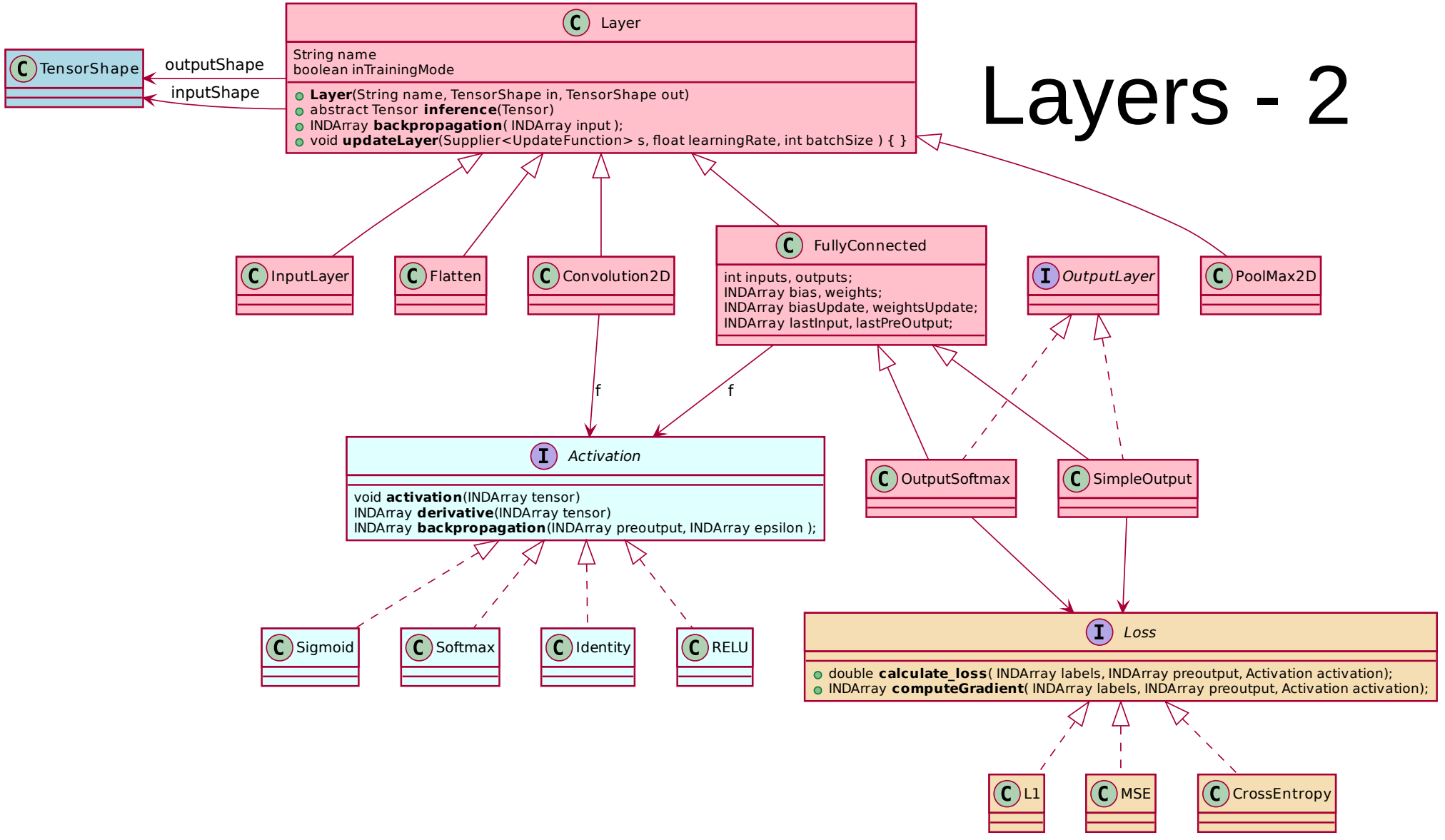


- One implementation: SGD
  - Uses gradient descent
  - to train model
- Two Validators
  - Regression: MSE
  - Classification: fraction of correctly classified samples

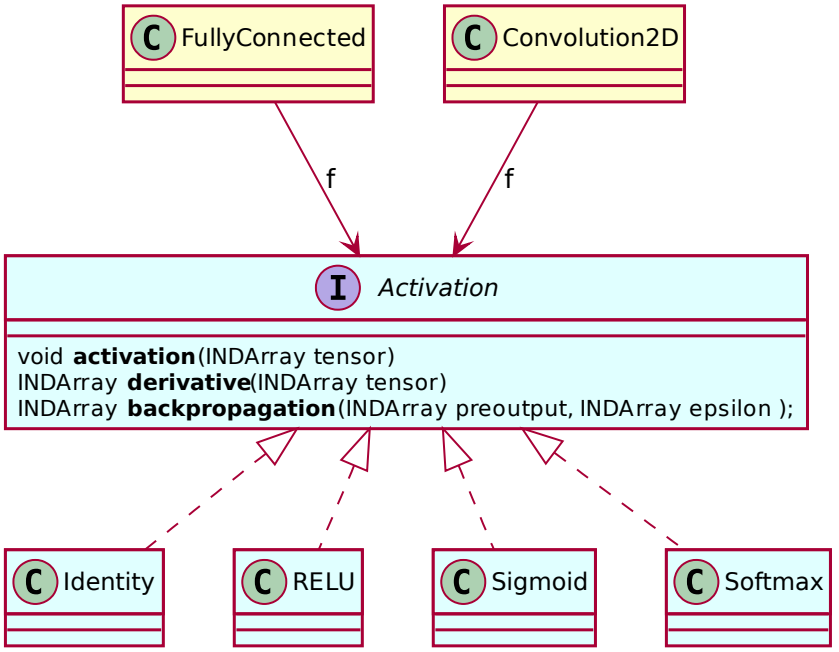
Optimizer – 2


trainOnBatch(TensorPair batch):
batchId := batchId + 1
// forward pass
prediction := model.inference(batch.input);
//compute performance indicators
accuracy := validator.validate(batch.output,prediction)
loss := model.outputLayer.calculateLoss(batch.output)
// back propagation
INDArray back := batch.output.values
for layer in reversed(model.layers)
back := layer.backpropagation(back)
// update weights
for layer in model.layers
layer.updateLayer(updater,learningRate,batchSize)
return BatchResult(batchId,loss,accuracy,learningRate)
UpdateFunction
void update(INDArray array, boolean isBias,
double learningRate, int batchSize, INDArray gradient);
uses
Supplier<UpdateFunction>
UpdateFunction get()
updater
SGD
Optimizer
double learningRate
BatchResult trainOnBatch(TensorPair batch)
BatchResult validate(List<TensorPair> batch)
model
Model
boolean inTrainingMode
Model(InputLayer)
addLayer(Layer)
Tensor inference(Tensor)
initializeLayers(Initializer)
outputLayer
OutputLayer
double calculateLoss(Tensor labels)
GradientDescent
update(...):
array := array - (learningRate/batchSize) * gradient


Layers - 1
Layer
String name
boolean inTrainingMode
Layer(String name, TensorShape in, TensorShape out)
abstract Tensor inference(Tensor)
INDArray backpropagation(INDArray input);
void updateLayer(Supplier<UpdateFunction> s, float learningRate, int batchSize) { }
TensorShape
outputShape
inputShape
InputLayer
Flatten
Convolution2D
PoolMax2D
OutputLayer
FullyConnected
int inputs, outputs;
INDArray bias, weights;
INDArray biasUpdate, weightsUpdate;
INDArray lastInput, lastPreOutput;
SimpleOutput
OutputSoftmax
Loss
double calculate_loss(INDArray labels, INDArray preoutput, Activation activation);
INDArray computeGradient(INDArray labels, INDArray preoutput, Activation activation);
L1
MSE
CrossEntropy

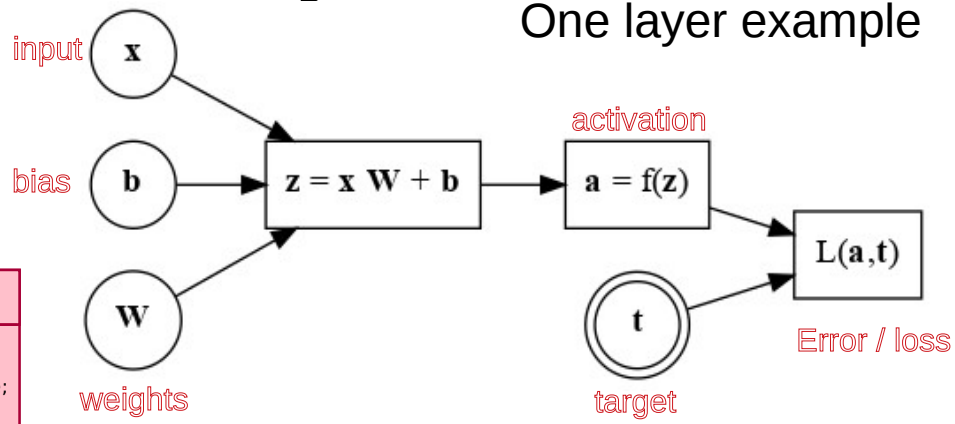

Layers - 2
Layer
String name
boolean inTrainingMode
Layer(String name, TensorShape in, TensorShape out)
abstract Tensor inference(Tensor)
INDArray backpropagation(INDArray input);
void updateLayer(Supplier<UpdateFunction> s, float learningRate, int batchSize) { }
TensorShape
outputShape
inputShape
InputLayer
Flatten
Convolution2D
FullyConnected
int inputs, outputs;
INDArray bias, weights;
INDArray biasUpdate, weightsUpdate;
INDArray lastInput, lastPreOutput;
OutputLayer
PoolMax2D
f
f
Activation
void activation(INDArray tensor)
INDArray derivative(INDArray tensor)
INDArray backpropagation(INDArray preoutput, INDArray epsilon);
OutputSoftmax
SimpleOutput
Sigmoid
Softmax
Identity
RELU
Loss
double calculate_loss(INDArray labels, INDArray preoutput, Activation activation);
INDArray computeGradient(INDArray labels, INDArray preoutput, Activation activation);
L1
MSE
CrossEntropy

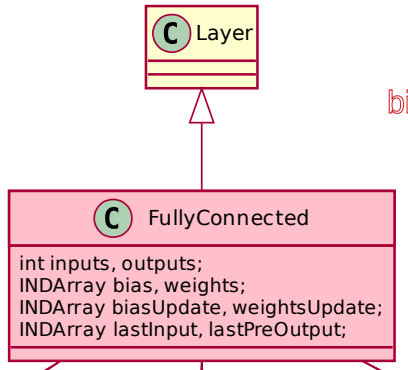
Activations



FullyConnected layer

One layer example





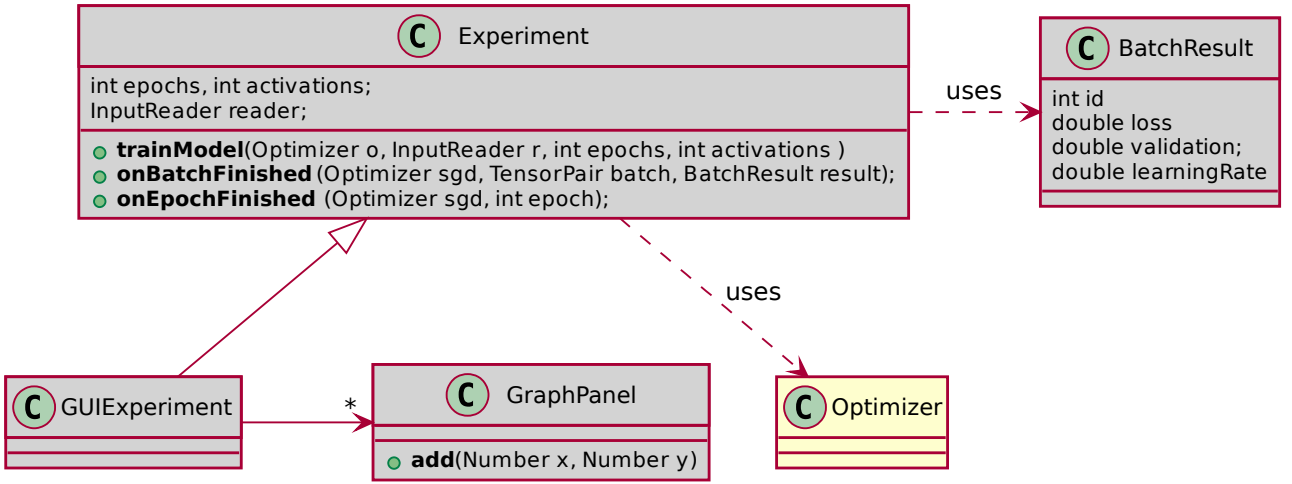

```
Tensor inference(Tensor in)
 // duplicate input, needed by backprop
 if (inTrainingMode) lastInput = in.getValues().dup();

 // weights: inputs x outputs; bias: outputs
 INDArray z := in * weights + bias

 // duplicate z, needed by backprop
 if (inTrainingMode) lastPreOutput := z.dup();

 // apply activation function to elements of z
 f.activation( z );

 // return inference tensor
 return new Tensor(z, new TensorShape(outputs));
```

```
INDArray backpropagation(INDArray dLda)
 INDArray x: = lastInput
 INDArray z := lastPreOutput; // z := x W + b

 // backprop activation: a=f(z)
 INDArray dLdz := f.backpropagation(z, dLda );

 // calculate weight update
 weightsUpdate := dLdz * x + weightsUpdate

 // calculate bias update
 biasUpdate := dLdz + biasUpdate

 // loss derivative dLdx for next layer
 return  dLdz * weights^T
```

```
UpdateFunction bu, wu;
updateLayer(
    Supplier<UpdateFunction> suf, float learningRate, int batchSize
)
    // at the first call create update functions
    if (bu==null) bu := suf.get(); if (wu==null) wu := suf.get();

    // use update functions to update weights and biases
    bu.update(bias , true , learningRate, batchSize, biasUpdate);
    wu.update(weights, false, learningRate, batchSize, weightsUpdate);
```

Experiment - 1

- Create experiment
- Read data
- Create model
  - Initialize model
- Build optimizer
- Call trainModel
  - React on onBatch/EpochFinished events
    - to fill plots
    - ...



Experiment - 2

```
trainModel(Optimizer o, InputReader r, int epochs, int activations )
    for epoch := 1 to epochs:
        // get and shuffle training data
        batches := r.getTrainingBatchIterator();

        // for a single epoch train using all batches
        o.model.setInTrainingMode(true);
        for batch in batches
            BatchResult result := o.trainOnBatch(batch);
            onBatchFinished(o,batch,result);
        onEpochFinished(o,epoch);
```