TP04-MultiLayerPerceptron

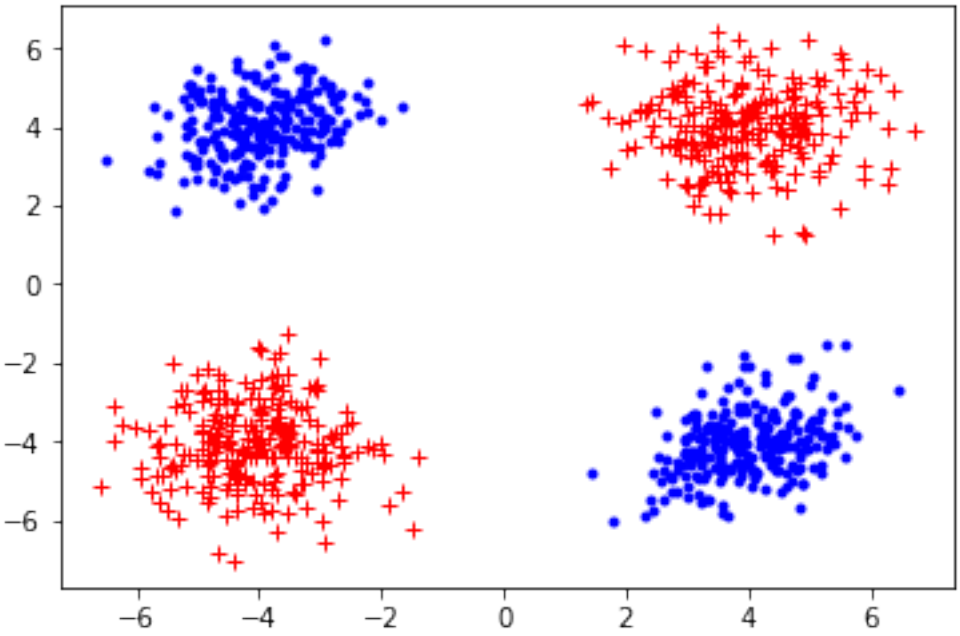
September 18, 2019

0.0.1 Multi Layer Perceptron

The following script generate a *Xor* dataset that can not be well separated by a logistic regression.

[1]:
```
import numpy as np
import tensorflow as tf

import matplotlib.pyplot as plt
%matplotlib inline
import datetime as dt


def generate_all_dataset():
    # --- Fake dataset ---

    np.random.seed(0)

    ntrain = 1000
    nvalid = 100
    ntest = 100

    mupos = np.array([4., 4.])
    sigmapos = np.array([[1., 0.], [0., 1.]])
    muneg = np.array([4., -4.])
    sigmaneg = np.array([[.7, .2], [.2, .7]])

    def generate_a_dataset(mupos, sigmapos, muneg, sigmaneg, n):
        nelem = int(n/4)
        npos1 = n-nelem*3
        npos2, nneg1, nneg2 = nelem, nelem, nelem

        Xpos1 = np.random.multivariate_normal(mupos, sigmapos, npos1)
        Ypos1 = np.stack((np.ones((npos1,)), np.zeros((npos1,))), axis=1)
        Xpos2 = np.random.multivariate_normal(-mupos, sigmapos, npos2)
        Ypos2 = np.stack((np.ones((npos2,)), np.zeros((npos2,))), axis=1)

        Xneg1 = np.random.multivariate_normal(muneg, sigmaneg, nneg1)
        Yneg1 = np.stack((np.zeros((nneg1,)), np.ones((nneg1,))), axis=1)
```

```
        Xneg2 = np.random.multivariate_normal(-muneg, sigmaneg, nneg2)
        Yneg2 = np.stack((np.zeros((nneg2,)), np.ones((nneg2,))), axis=1)

        X = np.concatenate((Xpos1, Xpos2, Xneg1, Xneg2))
        Y = np.concatenate((Ypos1, Ypos2, Yneg1, Yneg2))

        idx = np.arange(n)
        np.random.shuffle(idx)

        X, Y = X[idx], Y[idx]

        return np.array(X, dtype='float32'), np.array(Y, dtype='float32')

    Xtrain, Ytrain = generate_a_dataset(
        mupos, sigmapos, muneg, sigmaneg, ntrain)
    Xvalid, Yvalid = generate_a_dataset(
        mupos, sigmapos, muneg, sigmaneg, nvalid)
    Xtest, Ytest = generate_a_dataset(mupos, sigmapos, muneg, sigmaneg, ntest)

    return (Xtrain, Ytrain), (Xvalid, Yvalid), (Xtest, Ytest)


def plot_dataset(X, Y):
    plt.figure()
    idpos, = np.nonzero(Y[:, 0] == 1.)
    idneg, = np.nonzero(Y[:, 1] == 1.)
    plt.plot(X[idpos, 0], X[idpos, 1], 'r+')
    plt.plot(X[idneg, 0], X[idneg, 1], 'b.')
    plt.show()


def demo(trainset, validset, testset):
    plot_dataset(*trainset)


demo(*generate_all_dataset())
```



0.0.2 Model

In order to solve this problem you will need to add a second layer to the logistic regression model :

$$h = sigmoid(< x, A_1 > + b_1)$$
$$\hat{y} = softmax(< h, A_2 > + b_2)$$

where $h \in \mathbb{R}^n$ with n the number of hidden units.
It can still be learnt by the cross-entropy loss:

$$\mathcal{L}(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

In fact, that's a **Multi layer perceptron** !

0.0.3 Tensorboard histograms, parameters and gradients

Usually you don't want to track in the log file all the component of a parameter as it represents many scalars. That is why Tensorflow provides a facility to record histogram and distribrution of parameters:

For example, if you want to track the weights of a layer you can do:

```
asummary = tf.summary.histogram("A_parameter", self.A)
```

In order to also track gradients in Tensorboard, we need tensors representing them. Right now they are hidden inside the optimizer.

They can be obtain by the `compute_gradients` method on an optimizer:

```
        self.rawoptimizer = tf.train.GradientDescentOptimizer(
            self.learning_rate)
        self.optimizer = self.rawoptimizer.minimize(
            self.loss, var_list=var_list)
        self.grads = self.rawoptimizer.compute_gradients(
            self.loss, var_list=var_list)
```

`self.grads` is a list of tuple, each tuple consisting in (gradient,parameter).
Then summaries for parameters and gradients can be create by

```
        parametersummaries = []
        gradsummaries = []
        for gradient, parameter in self.grads:
            parametersummaries.append(
                tf.summary.histogram("parameters/" + parameter.name, parameter))
            gradsummaries.append(
                tf.summary.histogram("gradients/" + parameter.name, gradient))
        self.parametersummaries = tf.summary.merge(parametersummaries)
        self.gradsummaries = tf.summary.merge(gradsummaries)
```

Warnings

1) `parameter.name` relies on the `name` argument you give at the creation of parameters:

```
toto = tf.Variable(tf.zeros([5,5]), name="toto")
```

2) `self.gradsummaries` must be executed at the same *run* as `self.optimizer`, and only once per epoch:

```
 _,_,gradsummaryval = self.session.run([self.optimizer, self.lossmetric_update, self.gradsumma
```

0.0.4 Exercise

1) Copy the code of `LogisticRegression` class and rename the class to `MLPV1`
2) Modify the `__init__` method to take a number of hidden units as argument.
3) Modify `_build_network` method in order to have a 2-layer model. You will need `tf.nn.sigmoid`
4) Try your network with the following parameters:

```
nhiddens = 10
learning_rate = 1e-1
training_epochs = 100
batchsize = 5
```

5) Add parameter and gradient summaries
6) Analyze the training trough Tensorboard

What can be said about the evolution of the parameters (look at the distribution and histogram tabs in Tensorboard) ?

[2]:
```
class LogisticRegression:

    def __init__(self, dtype, ninputs, noutputs,
                 learning_rate=1e-1, training_epochs=100, batchsize=50):

        self.nc = ninputs
        self.no = noutputs

        self.dt_shapes = (tf.TensorShape((None, ninputs)),
                          tf.TensorShape((None, noutputs)))
        self.dt_types = (dtype, dtype)

        self.learning_rate = learning_rate
        self.training_epochs = training_epochs
        self.batchsize = batchsize

        self._build_network()

        # local and global variable initializers
        self.init_global = tf.initializers.global_variables()
        self.init_local = tf.initializers.local_variables()

    def _build_network(self):
        # Create ONLY ONE iterator base on types and shapes of one of the␣
↪dataset
        # both dataset should have the same types and shapes...
        self.dataset_iterator = tf.data.Iterator.from_structure(
            self.dt_types, self.dt_shapes)

        # Placeholders from the dataset iterator
        self.x, self.y = self.dataset_iterator.get_next()

        # Logistic regression parameters
        self.A = tf.Variable(tf.zeros([self.nc, self.no]))
        self.b = tf.Variable(tf.zeros([self.no]))
        # All parameters are gathered into var_list
        var_list = [self.A, self.b]

        # Actual logistic regression
        self.logit = tf.matmul(self.x, self.A) + self.b
        self.output = tf.nn.softmax(self.logit)
        self.pred = self.output > .5

        # Model loss
        self.loss = tf.losses.softmax_cross_entropy(self.y, self.logit)

        self.optimizer = tf.train.GradientDescentOptimizer(
```

```
            self.learning_rate).minimize(self.loss, var_list=var_list)

        # Metrics
        self.lossmetric, self.lossmetric_update = tf.metrics.mean(self.loss)
        self.accuracymetric,  self.accuracymetric_update = tf.metrics.accuracy(
            self.y, self.pred)

        # Summaries
        trainlosssummary = tf.summary.scalar("train_loss", self.lossmetric)
        trainaccuracysummary = tf.summary.scalar(
            "train_accuracy", self.accuracymetric)
        self.trainsummaries = tf.summary.merge(
            [trainlosssummary, trainaccuracysummary])

        validationlosssummary = tf.summary.scalar(
            "validation_loss", self.lossmetric)
        validationaccuracysummary = tf.summary.scalar(
            "validation_accuracy", self.accuracymetric)
        self.validationsummaries = tf.summary.merge(
            [validationlosssummary, validationaccuracysummary])

    def _prepareset(self, dataset, shuffle=True):
        if shuffle:
            dataset = dataset.shuffle(buffer_size=1000)
        dataset = dataset.batch(self.batchsize)
        return dataset

    def _compute_loss(self, set_iterator_init, nbatches):
        # Initialize all the metrics
        self.session.run(self.init_local)
        # Initialize the iterator
        self.session.run(set_iterator_init)
        # Loop
        for b in range(nbatches):
            self.session.run(
                [self.lossmetric_update, self.accuracymetric_update])
        return self.session.run([self.lossmetric, self.accuracymetric])

    def _compute_gradient_step(self, set_iterator_init, nbatches):
        self.session.run(self.init_local)
        self.session.run(set_iterator_init)
        for b in range(nbatches):
            self.session.run([self.optimizer, self.lossmetric_update])
        lossval = self.session.run(self.lossmetric)
        return lossval

    def _compute_pred_loss(self, set_iterator_init, nbatches):
```

```
        self.session.run(self.init_local)
        self.session.run(set_iterator_init)
        predval = None
        for b in range(nbatches):
            batch_pred, _, _ = self.session.run(
                [self.pred, self.lossmetric_update, self.accuracymetric_update])
            if predval is None:
                predval = batch_pred
            else:
                predval = np.concatenate((predval, batch_pred))
        lossval, accval = self.session.run(
            [self.lossmetric, self.accuracymetric])
        return predval, lossval, accval

    def train(self, trainset, validset, testset):

        (Xtrain, Ytrain) = trainset
        (Xvalid, Yvalid) = validset
        (Xtest, Ytest) = testset

        # --- Linear Regression ---

        ntrain, _ = Xtrain.shape
        ntest, _ = Xtest.shape
        nvalid, _ = Xvalid.shape

        trainset = self._prepareset(
            tf.data.Dataset.from_tensor_slices((Xtrain, Ytrain)))
        testset = self._prepareset(
            tf.data.Dataset.from_tensor_slices((Xtest, Ytest)), shuffle=False)
        validset = self._prepareset(
            tf.data.Dataset.from_tensor_slices((Xvalid, Yvalid)), shuffle=False)

        ntrainbatches = int(np.ceil(ntrain/self.batchsize))
        ntestbatches = int(np.ceil(ntest/self.batchsize))
        nvalidbatches = int(np.ceil(nvalid/self.batchsize))

        # Create one initializer per dataset
        training_init_op = self.dataset_iterator.make_initializer(trainset)
        test_init_op = self.dataset_iterator.make_initializer(testset)
        validation_init_op = self.dataset_iterator .make_initializer(validset)

        with tf.Session() as self.session:

            # We call the global initialization
            self.session.run(self.init_global)
```

```
            # Create the log writer
            logdir = "logs/logisticregression/" + dt.datetime.now().
↪strftime("%Y%m%d-%H%M%S")
            writer = tf.summary.FileWriter(logdir)

            # We compute the train and validation loss
            # Note that you just have to change the feed_dict to change the set

            trainloss, trainacc = self._compute_loss(
                training_init_op, ntrainbatches)
            validationloss, validacc = self._compute_loss(
                validation_init_op, nvalidbatches)

            trainsummariesval = self.session.run(self.trainsummaries)
            writer.add_summary(trainsummariesval, 0)
            validationsummariesval = self.session.run(self.validationsummaries)
            writer.add_summary(validationsummariesval, 0)

            print("Init\t\t train loss %f\t valid loss %f\t valid acc %f" %
                  (trainloss, validationloss, validacc))

            # We cycle on epochs
            for epoch in range(self.training_epochs):

                trainloss = self._compute_gradient_step(
                    training_init_op, ntrainbatches)
                validationloss, validacc = self._compute_loss(
                    validation_init_op,  nvalidbatches)

                trainsummariesval = self.session.run(self.trainsummaries)
                writer.add_summary(trainsummariesval, epoch+1)
                validationsummariesval = self.session.run(
                    self.validationsummaries)
                writer.add_summary(validationsummariesval, epoch+1)

                print("Epoch %03d\t train loss %f\t valid loss %f\t valid acc␣
↪%f" %
                      (epoch+1, trainloss, validationloss, validacc))

            # We compute the prediction and the test loss
            ytestpred, testloss, testacc = self._compute_pred_loss(
                test_init_op, ntestbatches)
            print("Test loss %f\t test acc %f" % (testloss, testacc))

        # Here session is closed automatically
        return ytestpred
```

[3]:
```
class MLPV1:

    def __init__(self, dtype, ninputs, nhiddens, noutputs,
                 learning_rate=1e-1, training_epochs=100, batchsize=50):

        self.nc = ninputs
        self.nh = nhiddens
        self.no = noutputs

        self.dt_shapes = (tf.TensorShape((None, ninputs)),
                          tf.TensorShape((None, noutputs)))
        self.dt_types = (dtype, dtype)

        self.learning_rate = learning_rate
        self.training_epochs = training_epochs
        self.batchsize = batchsize

        self._build_network()

        # local and global variable initializers
        self.init_global = tf.initializers.global_variables()
        self.init_local = tf.initializers.local_variables()

    def _build_network(self):
        # Create ONLY ONE iterator base on types and shapes of one of the␣
↪dataset
        # both dataset should have the same types and shapes...
        self.dataset_iterator = tf.data.Iterator.from_structure(
            self.dt_types, self.dt_shapes)

        # Placeholders from the dataset iterator
        self.x, self.y = self.dataset_iterator.get_next()

        # Parameters
        with tf.variable_scope("mlpv1"):
            with tf.variable_scope("layer1"):
                self.A1 = tf.Variable(tf.zeros([self.nc, self.nh]), name="A")
                self.b1 = tf.Variable(tf.zeros([self.nh]), name="b")

            with tf.variable_scope("layer2"):
                self.A2 = tf.Variable(tf.zeros([self.nh, self.no]), name="A")
                self.b2 = tf.Variable(tf.zeros([self.no]), name="b")

        # All parameters are gathered into var_list
        var_list = [self.A1, self.b1, self.A2, self.b2]

        # Actual logistic regression
```

```
        self.h = tf.nn.sigmoid(tf.matmul(self.x, self.A1) + self.b1)
        self.logit = tf.matmul(self.h, self.A2) + self.b2
        self.output = tf.nn.softmax(self.logit)
        self.pred = self.output > .5

        # Model loss
        self.loss = tf.losses.softmax_cross_entropy(self.y, self.logit)

        self.rawoptimizer = tf.train.GradientDescentOptimizer(
            self.learning_rate)
        self.optimizer = self.rawoptimizer.minimize(
            self.loss, var_list=var_list)
        self.grads = self.rawoptimizer.compute_gradients(
            self.loss, var_list=var_list)

        # Metrics
        self.lossmetric, self.lossmetric_update = tf.metrics.mean(self.loss)
        self.accuracymetric,  self.accuracymetric_update = tf.metrics.accuracy(
            self.y, self.pred)

        # Summaries

        parametersummaries = []
        gradsummaries = []
        for gradient, parameter in self.grads:
            parametersummaries.append(
                tf.summary.histogram("parameters/" + parameter.name, parameter))
            gradsummaries.append(
                tf.summary.histogram("gradients/" + parameter.name, gradient))
        self.parametersummaries = tf.summary.merge(parametersummaries)
        self.gradsummaries = tf.summary.merge(gradsummaries)

        trainlosssummary = tf.summary.scalar("train_loss", self.lossmetric)
        trainaccuracysummary = tf.summary.scalar(
            "train_accuracy", self.accuracymetric)
        self.trainsummaries = tf.summary.merge(
            [trainlosssummary, trainaccuracysummary])

        validationlosssummary = tf.summary.scalar(
            "validation_loss", self.lossmetric)
        validationaccuracysummary = tf.summary.scalar(
            "validation_accuracy", self.accuracymetric)
        self.validationsummaries = tf.summary.merge(
            [validationlosssummary, validationaccuracysummary])

    def _prepareset(self, dataset, shuffle=True):
        if shuffle:
```

```
            dataset = dataset.shuffle(buffer_size=1000)
        dataset = dataset.batch(self.batchsize)
        return dataset

    def _compute_loss(self, set_iterator_init, nbatches):
        # Initialize all the metrics
        self.session.run(self.init_local)
        # Initialize the iterator
        self.session.run(set_iterator_init)
        # Loop
        for b in range(nbatches):
            self.session.run(
                [self.lossmetric_update, self.accuracymetric_update])
        return self.session.run([self.lossmetric, self.accuracymetric])

    def _compute_gradient_step(self, set_iterator_init, nbatches):
        self.session.run(self.init_local)
        self.session.run(set_iterator_init)
        for b in range(nbatches):
            if b==nbatches-1:
                _,_,gradsummaryval = self.session.run([self.optimizer,
                                    self.lossmetric_update,
                                   self.gradsummaries])

                self.writer.add_summary(gradsummaryval, self.epoch+1)
            else:
                self.session.run([self.optimizer, self.lossmetric_update])
        lossval = self.session.run(self.lossmetric)

        return lossval

    def _compute_pred_loss(self, set_iterator_init, nbatches):
        self.session.run(self.init_local)
        self.session.run(set_iterator_init)
        predval = None
        for b in range(nbatches):
            batch_pred, _, _ = self.session.run(
                [self.pred, self.lossmetric_update, self.accuracymetric_update])
            if predval is None:
                predval = batch_pred
            else:
                predval = np.concatenate((predval, batch_pred))
        lossval, accval = self.session.run(
            [self.lossmetric, self.accuracymetric])
        return predval, lossval, accval

    def train(self, trainset, validset, testset):
```

```
        (Xtrain, Ytrain) = trainset
        (Xvalid, Yvalid) = validset
        (Xtest, Ytest) = testset

        # --- Linear Regression ---

        ntrain, _ = Xtrain.shape
        ntest, _ = Xtest.shape
        nvalid, _ = Xvalid.shape

        trainset = self._prepareset(
            tf.data.Dataset.from_tensor_slices((Xtrain, Ytrain)))
        testset = self._prepareset(
            tf.data.Dataset.from_tensor_slices((Xtest, Ytest)), shuffle=False)
        validset = self._prepareset(
            tf.data.Dataset.from_tensor_slices((Xvalid, Yvalid)), shuffle=False)

        ntrainbatches = int(np.ceil(ntrain/self.batchsize))
        ntestbatches = int(np.ceil(ntest/self.batchsize))
        nvalidbatches = int(np.ceil(nvalid/self.batchsize))

        # Create one initializer per dataset
        training_init_op = self.dataset_iterator.make_initializer(trainset)
        test_init_op = self.dataset_iterator.make_initializer(testset)
        validation_init_op = self.dataset_iterator .make_initializer(validset)

        with tf.Session() as self.session:

            # We call the global initialization
            self.session.run(self.init_global)

            # Create the log writer
            logdir = "logs/mlp/" + dt.datetime.now().strftime("%Y%m%d-%H%M%S")
            self.writer = tf.summary.FileWriter(logdir)

            # We compute the train and validation loss
            # Note that you just have to change the feed_dict to change the set

            trainloss, trainacc = self._compute_loss(
                training_init_op, ntrainbatches)
            validationloss, validacc = self._compute_loss(
                validation_init_op, nvalidbatches)

            trainsummariesval = self.session.run(self.trainsummaries)
            self.writer.add_summary(trainsummariesval, 0)
            validationsummariesval = self.session.run(self.validationsummaries)
```

```
            self.writer.add_summary(validationsummariesval, 0)

            parameterval = self.session.run(self.parametersummaries)
            self.writer.add_summary(parameterval, 0)

            print("Init\t\t train loss %f\t valid loss %f\t valid acc %f" %
                  (trainloss, validationloss, validacc))

            # We cycle on epochs
            for self.epoch in range(self.training_epochs):

                trainloss = self._compute_gradient_step(
                    training_init_op, ntrainbatches)
                validationloss, validacc = self._compute_loss(
                    validation_init_op,  nvalidbatches)

                trainsummariesval = self.session.run(self.trainsummaries)
                self.writer.add_summary(trainsummariesval, self.epoch+1)
                validationsummariesval = self.session.run(
                    self.validationsummaries)
                self.writer.add_summary(validationsummariesval, self.epoch+1)

                parameterval = self.session.run(self.parametersummaries)
                self.writer.add_summary(parameterval, self.epoch+1)

                print("Epoch %03d\t train loss %f\t valid loss %f\t valid acc␣
 ↪%f" %
                      (self.epoch+1, trainloss, validationloss, validacc))

            # We compute the prediction and the test loss
            ytestpred, testloss, testacc = self._compute_pred_loss(
                test_init_op, ntestbatches)
            print("Test loss %f\t test acc %f" % (testloss, testacc))

        # Here session is closed automatically
        return ytestpred


def mlpv1(trainset, validset, testset):
    # reset tensorflow
    tf.reset_default_graph()

    Xtrain, Ytrain = trainset
    _, ninputs = Xtrain.shape
    _, noutputs = Ytrain.shape

    nhiddens = 10
```

```
    lr = MLPV1(tf.float32, ninputs, nhiddens, noutputs,
               learning_rate=1e-1, training_epochs=100, batchsize=5)
    Ytestpred = lr.train(trainset, validset, testset)

    Xtest, Ytest = testset

    plot_dataset(Xtest, Ytestpred)


mlpv1(*generate_all_dataset())
```

```
WARNING:tensorflow:From /home/rherault/.local/venvs/spyder/lib/python3.7/site-
packages/tensorflow/python/framework/op_def_library.py:263: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /home/rherault/.local/venvs/spyder/lib/python3.7/site-
packages/tensorflow/python/ops/losses/losses_impl.py:209: to_float (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.cast instead.
INFO:tensorflow:Summary name parameters/mlpv1/layer1/A:0 is illegal; using
parameters/mlpv1/layer1/A_0 instead.
INFO:tensorflow:Summary name gradients/mlpv1/layer1/A:0 is illegal; using
gradients/mlpv1/layer1/A_0 instead.
INFO:tensorflow:Summary name parameters/mlpv1/layer1/b:0 is illegal; using
parameters/mlpv1/layer1/b_0 instead.
INFO:tensorflow:Summary name gradients/mlpv1/layer1/b:0 is illegal; using
gradients/mlpv1/layer1/b_0 instead.
INFO:tensorflow:Summary name parameters/mlpv1/layer2/A:0 is illegal; using
parameters/mlpv1/layer2/A_0 instead.
INFO:tensorflow:Summary name gradients/mlpv1/layer2/A:0 is illegal; using
gradients/mlpv1/layer2/A_0 instead.
INFO:tensorflow:Summary name parameters/mlpv1/layer2/b:0 is illegal; using
parameters/mlpv1/layer2/b_0 instead.
INFO:tensorflow:Summary name gradients/mlpv1/layer2/b:0 is illegal; using
gradients/mlpv1/layer2/b_0 instead.
Init             train loss 0.693146     valid loss 0.693147     valid acc
0.500000
Epoch 001        train loss 0.706059     valid loss 0.693172     valid acc
0.500000
Epoch 002        train loss 0.704355     valid loss 0.723053     valid acc
0.500000
Epoch 003        train loss 0.699262     valid loss 0.702432     valid acc
```

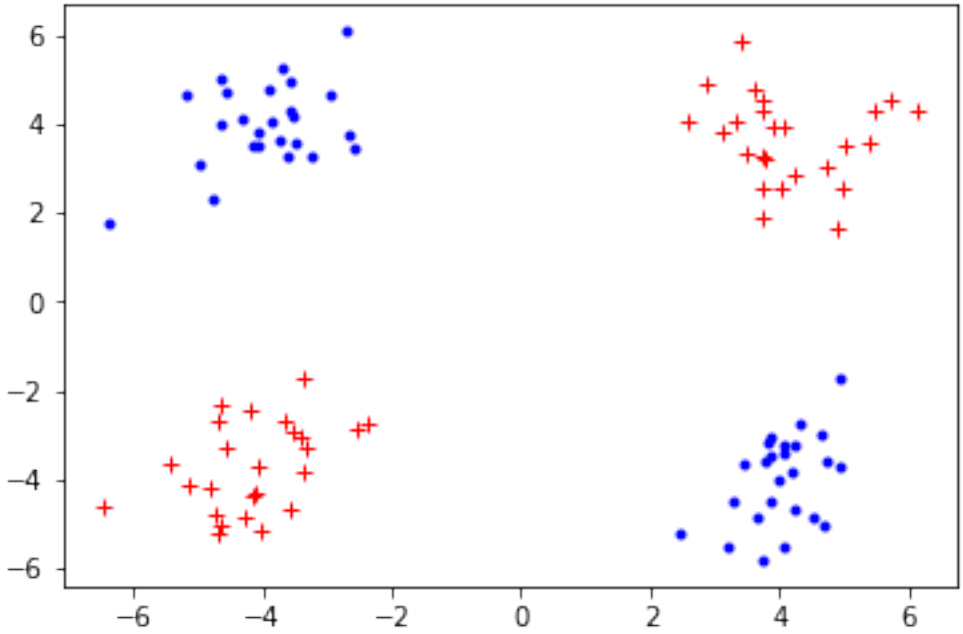
```
0.500000
Epoch 004       train loss 0.700361     valid loss 0.696511     valid acc
0.500000
Epoch 005       train loss 0.703234     valid loss 0.702387     valid acc
0.500000
Epoch 006       train loss 0.704216     valid loss 0.695846     valid acc
0.500000
Epoch 007       train loss 0.704442     valid loss 0.693675     valid acc
0.500000
Epoch 008       train loss 0.701749     valid loss 0.702520     valid acc
0.500000
Epoch 009       train loss 0.703050     valid loss 0.695143     valid acc
0.500000
Epoch 010       train loss 0.701098     valid loss 0.693148     valid acc
0.500000
Epoch 011       train loss 0.704482     valid loss 0.701009     valid acc
0.500000
Epoch 012       train loss 0.706172     valid loss 0.693829     valid acc
0.500000
Epoch 013       train loss 0.703780     valid loss 0.693267     valid acc
0.500000
Epoch 014       train loss 0.700646     valid loss 0.715416     valid acc
0.500000
Epoch 015       train loss 0.700366     valid loss 0.700329     valid acc
0.500000
Epoch 016       train loss 0.700186     valid loss 0.717031     valid acc
0.500000
Epoch 017       train loss 0.700532     valid loss 0.729367     valid acc
0.500000
Epoch 018       train loss 0.700518     valid loss 0.713111     valid acc
0.500000
Epoch 019       train loss 0.702148     valid loss 0.693894     valid acc
0.500000
Epoch 020       train loss 0.703244     valid loss 0.704454     valid acc
0.500000
Epoch 021       train loss 0.701998     valid loss 0.693152     valid acc
0.500000
Epoch 022       train loss 0.702637     valid loss 0.703354     valid acc
0.500000
Epoch 023       train loss 0.702529     valid loss 0.704868     valid acc
0.500000
Epoch 024       train loss 0.697876     valid loss 0.706933     valid acc
0.500000
Epoch 025       train loss 0.701567     valid loss 0.694027     valid acc
0.500000
Epoch 026       train loss 0.700151     valid loss 0.696434     valid acc
0.500000
Epoch 027       train loss 0.700016     valid loss 0.719469     valid acc
```

```
0.500000
Epoch 028        train loss 0.703630     valid loss 0.695819     valid acc
0.500000
Epoch 029        train loss 0.700636     valid loss 0.695602     valid acc
0.500000
Epoch 030        train loss 0.701318     valid loss 0.701571     valid acc
0.500000
Epoch 031        train loss 0.702468     valid loss 0.693203     valid acc
0.500000
Epoch 032        train loss 0.700893     valid loss 0.693976     valid acc
0.500000
Epoch 033        train loss 0.704225     valid loss 0.694125     valid acc
0.500000
Epoch 034        train loss 0.702393     valid loss 0.698208     valid acc
0.500000
Epoch 035        train loss 0.702025     valid loss 0.697670     valid acc
0.500000
Epoch 036        train loss 0.701573     valid loss 0.695063     valid acc
0.500000
Epoch 037        train loss 0.694622     valid loss 0.741551     valid acc
0.500000
Epoch 038        train loss 0.700420     valid loss 0.704326     valid acc
0.500000
Epoch 039        train loss 0.701491     valid loss 0.713455     valid acc
0.500000
Epoch 040        train loss 0.699980     valid loss 0.696007     valid acc
0.500000
Epoch 041        train loss 0.699084     valid loss 0.696933     valid acc
0.500000
Epoch 042        train loss 0.700882     valid loss 0.695817     valid acc
0.500000
Epoch 043        train loss 0.700839     valid loss 0.697926     valid acc
0.500000
Epoch 044        train loss 0.696428     valid loss 0.704837     valid acc
0.500000
Epoch 045        train loss 0.700676     valid loss 0.693310     valid acc
0.500000
Epoch 046        train loss 0.701062     valid loss 0.693589     valid acc
0.500000
Epoch 047        train loss 0.697194     valid loss 0.702370     valid acc
0.500000
Epoch 048        train loss 0.702956     valid loss 0.694061     valid acc
0.500000
Epoch 049        train loss 0.698304     valid loss 0.693646     valid acc
0.500000
Epoch 050        train loss 0.699876     valid loss 0.693622     valid acc
0.500000
Epoch 051        train loss 0.698990     valid loss 0.716102     valid acc
```

```
0.500000
Epoch 052       train loss 0.700654     valid loss 0.695092     valid acc
0.500000
Epoch 053       train loss 0.698548     valid loss 0.698030     valid acc
0.500000
Epoch 054       train loss 0.700147     valid loss 0.693693     valid acc
0.500000
Epoch 055       train loss 0.699356     valid loss 0.693177     valid acc
0.500000
Epoch 056       train loss 0.701796     valid loss 0.696749     valid acc
0.500000
Epoch 057       train loss 0.699871     valid loss 0.694431     valid acc
0.500000
Epoch 058       train loss 0.695408     valid loss 0.719627     valid acc
0.500000
Epoch 059       train loss 0.697946     valid loss 0.700913     valid acc
0.500000
Epoch 060       train loss 0.700717     valid loss 0.698220     valid acc
0.500000
Epoch 061       train loss 0.700008     valid loss 0.693065     valid acc
0.500000
Epoch 062       train loss 0.698212     valid loss 0.695234     valid acc
0.500000
Epoch 063       train loss 0.695636     valid loss 0.691885     valid acc
0.740000
Epoch 064       train loss 0.692986     valid loss 0.689223     valid acc
0.750000
Epoch 065       train loss 0.688272     valid loss 0.684620     valid acc
0.750000
Epoch 066       train loss 0.683886     valid loss 0.679972     valid acc
0.750000
Epoch 067       train loss 0.672716     valid loss 0.665617     valid acc
0.670000
Epoch 068       train loss 0.662507     valid loss 0.662059     valid acc
0.520000
Epoch 069       train loss 0.655503     valid loss 0.653916     valid acc
0.740000
Epoch 070       train loss 0.643067     valid loss 0.641715     valid acc
0.700000
Epoch 071       train loss 0.635835     valid loss 0.631851     valid acc
0.640000
Epoch 072       train loss 0.623579     valid loss 0.622815     valid acc
0.650000
Epoch 073       train loss 0.614849     valid loss 0.615547     valid acc
0.620000
Epoch 074       train loss 0.604770     valid loss 0.598534     valid acc
0.670000
Epoch 075       train loss 0.591114     valid loss 0.583410     valid acc
```

```
0.720000
Epoch 076       train loss 0.573093     valid loss 0.560154     valid acc
0.730000
Epoch 077       train loss 0.547252     valid loss 0.529248     valid acc
0.750000
Epoch 078       train loss 0.514075     valid loss 0.490873     valid acc
0.750000
Epoch 079       train loss 0.469669     valid loss 0.438316     valid acc
0.750000
Epoch 080       train loss 0.412703     valid loss 0.381552     valid acc
0.750000
Epoch 081       train loss 0.371369     valid loss 0.344595     valid acc
0.980000
Epoch 082       train loss 0.309772     valid loss 0.260702     valid acc
1.000000
Epoch 083       train loss 0.230830     valid loss 0.190508     valid acc
1.000000
Epoch 084       train loss 0.168838     valid loss 0.138202     valid acc
1.000000
Epoch 085       train loss 0.124882     valid loss 0.101169     valid acc
1.000000
Epoch 086       train loss 0.094768     valid loss 0.079652     valid acc
1.000000
Epoch 087       train loss 0.075172     valid loss 0.061602     valid acc
1.000000
Epoch 088       train loss 0.060642     valid loss 0.049922     valid acc
1.000000
Epoch 089       train loss 0.049608     valid loss 0.044184     valid acc
1.000000
Epoch 090       train loss 0.042387     valid loss 0.035266     valid acc
1.000000
Epoch 091       train loss 0.036603     valid loss 0.030292     valid acc
1.000000
Epoch 092       train loss 0.032418     valid loss 0.026335     valid acc
1.000000
Epoch 093       train loss 0.028505     valid loss 0.022973     valid acc
1.000000
Epoch 094       train loss 0.025667     valid loss 0.021843     valid acc
1.000000
Epoch 095       train loss 0.023469     valid loss 0.018676     valid acc
1.000000
Epoch 096       train loss 0.021110     valid loss 0.016672     valid acc
1.000000
Epoch 097       train loss 0.019515     valid loss 0.016140     valid acc
1.000000
Epoch 098       train loss 0.018039     valid loss 0.014332     valid acc
1.000000
Epoch 099       train loss 0.016705     valid loss 0.014057     valid acc
```

```
1.000000
Epoch 100        train loss 0.015600     valid loss 0.012663     valid acc
1.000000
Test loss 0.012045       test acc 1.000000
```



0.1 Weight initialization and regularization

A good start to speedup the training is to have a better weight initialization than 0 init.

Here is an example to get a variable initialize with a random normal distribution:

```
toto = tf.get_variable("toto", (5, 5), initializer=tf.random_normal_initializer())
```

Initializer available for Tensorflow are listed here: https://www.tensorflow.org/api_docs/python/tf/initializ

Moreover, weight regularization can prevent the network to be stuck into a local minimum at training start. Usually a L2 norm is used.

In Tensorflow, you can directly write the standard Tikhonov regularization scheme can expressed by adding a regularization term to the loss:

```
# losses
self.regularizers = tf.nn.l2_loss(self.A1) + tf.nn.l2_loss(self.A2)
self.model_loss = tf.losses.softmax_cross_entropy(self.y, self.logit)
beta = 5e-2
self.loss = self.model_loss + beta*self.regularizers
```

0.2 Variable scope

In order to structure your network, you may want to use `variable_scope` environment. It will automatically prepend the name of the variable by it's own name:

```
        # Parameters
        with tf.variable_scope("mlp"):
            with tf.variable_scope("layer1"):
                self.A1 = tf.get_variable("A", (self.nc, self.nh),
                    initializer=tf.random_normal_initializer())

                self.b1 = tf.Variable(tf.zeros([self.nh]), name="b")

            with tf.variable_scope("layer2"):
                self.A2 = tf.get_variable("A", (self.nh, self.no),
                    initializer=tf.random_normal_initializer())
                self.b2 = tf.Variable(tf.zeros([self.no]), name="b")
```

Here the first *A* variable will be named (approximatively) *mlp/layer1/A* and the second *mlp/layer2/A*

0.3 Exercise

Modify your code to add weight initialization/regularization and variable scope.

[4]:
```
class MLPV2(MLPV1):
    def _build_network(self):
        # Create ONLY ONE iterator base on types and shapes of one of the␣
↪dataset
        # both dataset should have the same types and shapes...
        self.dataset_iterator = tf.data.Iterator.from_structure(
            self.dt_types, self.dt_shapes)

        # Placeholders from the dataset iterator
        self.x, self.y = self.dataset_iterator.get_next()

        # Parameters
        with tf.variable_scope("mlpv2"):
            with tf.variable_scope("layer1"):
                self.A1 = tf.get_variable("A", (self.nc, self.nh),
                                          initializer=tf.
↪random_normal_initializer())

                self.b1 = tf.Variable(tf.zeros([self.nh]), name="b")

            with tf.variable_scope("layer2"):
                self.A2 = tf.get_variable("A", (self.nh, self.no),
                                          initializer=tf.
↪random_normal_initializer())
```

```
            self.b2 = tf.Variable(tf.zeros([self.no]), name="b")

        # All parameters are gathered into var_list
        var_list = [self.A1, self.b1, self.A2, self.b2]

        # Actual logistic regression
        self.h = tf.nn.sigmoid(tf.matmul(self.x, self.A1) + self.b1)
        self.logit = tf.matmul(self.h, self.A2) + self.b2
        self.output = tf.nn.softmax(self.logit)
        self.pred = self.output > .5

        # losses
        self.regularizers = tf.nn.l2_loss(self.A1) + tf.nn.l2_loss(self.A2)
        self.model_loss = tf.losses.softmax_cross_entropy(self.y, self.logit)
        beta = 5e-2
        self.loss = self.model_loss + beta*self.regularizers

        self.rawoptimizer = tf.train.GradientDescentOptimizer(
            self.learning_rate)
        self.optimizer = self.rawoptimizer.minimize(
            self.loss, var_list=var_list)
        self.grads = self.rawoptimizer.compute_gradients(
            self.loss, var_list=var_list)

        # Metrics
        self.lossmetric, self.lossmetric_update = tf.metrics.mean(self.loss)
        self.accuracymetric,  self.accuracymetric_update = tf.metrics.accuracy(
            self.y, self.pred)

        # Summaries

        parametersummaries = []
        gradsummaries = []
        for gradient, parameter in self.grads:
            parametersummaries.append(
                tf.summary.histogram("parameters/" + parameter.name, parameter))
            gradsummaries.append(
                tf.summary.histogram("gradients/" + parameter.name, gradient))
        self.parametersummaries = tf.summary.merge(parametersummaries)
        self.gradsummaries = tf.summary.merge(gradsummaries)

        trainlosssummary = tf.summary.scalar("train_loss", self.lossmetric)
        trainaccuracysummary = tf.summary.scalar(
            "train_accuracy", self.accuracymetric)
        self.trainsummaries = tf.summary.merge(
            [trainlosssummary, trainaccuracysummary])
```

```
        validationlosssummary = tf.summary.scalar(
            "validation_loss", self.lossmetric)
        validationaccuracysummary = tf.summary.scalar(
            "validation_accuracy", self.accuracymetric)
        self.validationsummaries = tf.summary.merge(
            [validationlosssummary, validationaccuracysummary])


def mlpv2(trainset, validset, testset):
    # reset tensorflow
    tf.reset_default_graph()

    Xtrain, Ytrain = trainset
    _, ninputs = Xtrain.shape
    _, noutputs = Ytrain.shape

    nhiddens = 10

    lr = MLPV2(tf.float32, ninputs, nhiddens, noutputs,
               learning_rate=1e-1, training_epochs=100, batchsize=5)
    Ytestpred = lr.train(trainset, validset, testset)

    Xtest, Ytest = testset

    plot_dataset(Xtest, Ytestpred)


mlpv2(*generate_all_dataset())
```

```
INFO:tensorflow:Summary name parameters/mlpv2/layer1/A:0 is illegal; using
parameters/mlpv2/layer1/A_0 instead.
INFO:tensorflow:Summary name gradients/mlpv2/layer1/A:0 is illegal; using
gradients/mlpv2/layer1/A_0 instead.
INFO:tensorflow:Summary name parameters/mlpv2/layer1/b:0 is illegal; using
parameters/mlpv2/layer1/b_0 instead.
INFO:tensorflow:Summary name gradients/mlpv2/layer1/b:0 is illegal; using
gradients/mlpv2/layer1/b_0 instead.
INFO:tensorflow:Summary name parameters/mlpv2/layer2/A:0 is illegal; using
parameters/mlpv2/layer2/A_0 instead.
INFO:tensorflow:Summary name gradients/mlpv2/layer2/A:0 is illegal; using
gradients/mlpv2/layer2/A_0 instead.
INFO:tensorflow:Summary name parameters/mlpv2/layer2/b:0 is illegal; using
parameters/mlpv2/layer2/b_0 instead.
INFO:tensorflow:Summary name gradients/mlpv2/layer2/b:0 is illegal; using
gradients/mlpv2/layer2/b_0 instead.
Init             train loss 1.589057     valid loss 1.591728     valid acc
0.500000
```

```
Epoch 001       train loss 1.012221     valid loss 0.776579     valid acc
0.740000
Epoch 002       train loss 0.716227     valid loss 0.692316     valid acc
0.500000
Epoch 003       train loss 0.631147     valid loss 0.590234     valid acc
1.000000
Epoch 004       train loss 0.573136     valid loss 0.550871     valid acc
1.000000
Epoch 005       train loss 0.546683     valid loss 0.536512     valid acc
1.000000
Epoch 006       train loss 0.535075     valid loss 0.539410     valid acc
1.000000
Epoch 007       train loss 0.532251     valid loss 0.525775     valid acc
1.000000
Epoch 008       train loss 0.528133     valid loss 0.528896     valid acc
1.000000
Epoch 009       train loss 0.528331     valid loss 0.522575     valid acc
1.000000
Epoch 010       train loss 0.525031     valid loss 0.523982     valid acc
1.000000
Epoch 011       train loss 0.523772     valid loss 0.522629     valid acc
1.000000
Epoch 012       train loss 0.524998     valid loss 0.522170     valid acc
1.000000
Epoch 013       train loss 0.525161     valid loss 0.522105     valid acc
1.000000
Epoch 014       train loss 0.523964     valid loss 0.528663     valid acc
1.000000
Epoch 015       train loss 0.523579     valid loss 0.536476     valid acc
1.000000
Epoch 016       train loss 0.522429     valid loss 0.525674     valid acc
1.000000
Epoch 017       train loss 0.523895     valid loss 0.521856     valid acc
1.000000
Epoch 018       train loss 0.523348     valid loss 0.520912     valid acc
1.000000
Epoch 019       train loss 0.521652     valid loss 0.528930     valid acc
1.000000
Epoch 020       train loss 0.524386     valid loss 0.521149     valid acc
1.000000
Epoch 021       train loss 0.522047     valid loss 0.523071     valid acc
1.000000
Epoch 022       train loss 0.523394     valid loss 0.521003     valid acc
1.000000
Epoch 023       train loss 0.523020     valid loss 0.521585     valid acc
1.000000
Epoch 024       train loss 0.522931     valid loss 0.521027     valid acc
1.000000
```

```
Epoch 025       train loss 0.523222     valid loss 0.520997     valid acc
1.000000
Epoch 026       train loss 0.523750     valid loss 0.520437     valid acc
1.000000
Epoch 027       train loss 0.524657     valid loss 0.521691     valid acc
1.000000
Epoch 028       train loss 0.524276     valid loss 0.520487     valid acc
1.000000
Epoch 029       train loss 0.524321     valid loss 0.521579     valid acc
1.000000
Epoch 030       train loss 0.524493     valid loss 0.520633     valid acc
1.000000
Epoch 031       train loss 0.523051     valid loss 0.523472     valid acc
1.000000
Epoch 032       train loss 0.522452     valid loss 0.521324     valid acc
1.000000
Epoch 033       train loss 0.523848     valid loss 0.522219     valid acc
1.000000
Epoch 034       train loss 0.522900     valid loss 0.523702     valid acc
1.000000
Epoch 035       train loss 0.523205     valid loss 0.521006     valid acc
1.000000
Epoch 036       train loss 0.524209     valid loss 0.522346     valid acc
1.000000
Epoch 037       train loss 0.523766     valid loss 0.520966     valid acc
1.000000
Epoch 038       train loss 0.521488     valid loss 0.528551     valid acc
1.000000
Epoch 039       train loss 0.524189     valid loss 0.520878     valid acc
1.000000
Epoch 040       train loss 0.523885     valid loss 0.525003     valid acc
1.000000
Epoch 041       train loss 0.523449     valid loss 0.520848     valid acc
1.000000
Epoch 042       train loss 0.523579     valid loss 0.521654     valid acc
1.000000
Epoch 043       train loss 0.524391     valid loss 0.521480     valid acc
1.000000
Epoch 044       train loss 0.524117     valid loss 0.520643     valid acc
1.000000
Epoch 045       train loss 0.521384     valid loss 0.536203     valid acc
1.000000
Epoch 046       train loss 0.525085     valid loss 0.523893     valid acc
1.000000
Epoch 047       train loss 0.524372     valid loss 0.524544     valid acc
1.000000
Epoch 048       train loss 0.523407     valid loss 0.520954     valid acc
1.000000
```

```
Epoch 049       train loss 0.523447     valid loss 0.534330     valid acc
1.000000
Epoch 050       train loss 0.522731     valid loss 0.524192     valid acc
1.000000
Epoch 051       train loss 0.522969     valid loss 0.521577     valid acc
1.000000
Epoch 052       train loss 0.522849     valid loss 0.520721     valid acc
1.000000
Epoch 053       train loss 0.524277     valid loss 0.521018     valid acc
1.000000
Epoch 054       train loss 0.523606     valid loss 0.523456     valid acc
1.000000
Epoch 055       train loss 0.524494     valid loss 0.521090     valid acc
1.000000
Epoch 056       train loss 0.523898     valid loss 0.521685     valid acc
1.000000
Epoch 057       train loss 0.523427     valid loss 0.534619     valid acc
1.000000
Epoch 058       train loss 0.524171     valid loss 0.527095     valid acc
1.000000
Epoch 059       train loss 0.523687     valid loss 0.521814     valid acc
1.000000
Epoch 060       train loss 0.522539     valid loss 0.523098     valid acc
1.000000
Epoch 061       train loss 0.521278     valid loss 0.522390     valid acc
1.000000
Epoch 062       train loss 0.522372     valid loss 0.521643     valid acc
1.000000
Epoch 063       train loss 0.523871     valid loss 0.528808     valid acc
1.000000
Epoch 064       train loss 0.523720     valid loss 0.520754     valid acc
1.000000
Epoch 065       train loss 0.523417     valid loss 0.521679     valid acc
1.000000
Epoch 066       train loss 0.523307     valid loss 0.523266     valid acc
1.000000
Epoch 067       train loss 0.524231     valid loss 0.524700     valid acc
1.000000
Epoch 068       train loss 0.523046     valid loss 0.521164     valid acc
1.000000
Epoch 069       train loss 0.524654     valid loss 0.523159     valid acc
1.000000
Epoch 070       train loss 0.523344     valid loss 0.520769     valid acc
1.000000
Epoch 071       train loss 0.523260     valid loss 0.520565     valid acc
1.000000
Epoch 072       train loss 0.523477     valid loss 0.523613     valid acc
1.000000
```

```
Epoch 073       train loss 0.522612     valid loss 0.525234     valid acc
1.000000
Epoch 074       train loss 0.523105     valid loss 0.525542     valid acc
1.000000
Epoch 075       train loss 0.521938     valid loss 0.525582     valid acc
1.000000
Epoch 076       train loss 0.523879     valid loss 0.521557     valid acc
1.000000
Epoch 077       train loss 0.522814     valid loss 0.535938     valid acc
1.000000
Epoch 078       train loss 0.523480     valid loss 0.524609     valid acc
1.000000
Epoch 079       train loss 0.522864     valid loss 0.521358     valid acc
1.000000
Epoch 080       train loss 0.523159     valid loss 0.520776     valid acc
1.000000
Epoch 081       train loss 0.522305     valid loss 0.536531     valid acc
1.000000
Epoch 082       train loss 0.523214     valid loss 0.521292     valid acc
1.000000
Epoch 083       train loss 0.523746     valid loss 0.521071     valid acc
1.000000
Epoch 084       train loss 0.523711     valid loss 0.523881     valid acc
1.000000
Epoch 085       train loss 0.521738     valid loss 0.521289     valid acc
1.000000
Epoch 086       train loss 0.524190     valid loss 0.524496     valid acc
1.000000
Epoch 087       train loss 0.524239     valid loss 0.525703     valid acc
1.000000
Epoch 088       train loss 0.522053     valid loss 0.524721     valid acc
1.000000
Epoch 089       train loss 0.523203     valid loss 0.521996     valid acc
1.000000
Epoch 090       train loss 0.522210     valid loss 0.520777     valid acc
1.000000
Epoch 091       train loss 0.522238     valid loss 0.522317     valid acc
1.000000
Epoch 092       train loss 0.523274     valid loss 0.520643     valid acc
1.000000
Epoch 093       train loss 0.521942     valid loss 0.521766     valid acc
1.000000
Epoch 094       train loss 0.522894     valid loss 0.521004     valid acc
1.000000
Epoch 095       train loss 0.523728     valid loss 0.520988     valid acc
1.000000
Epoch 096       train loss 0.523726     valid loss 0.528501     valid acc
1.000000
```

```
Epoch 097        train loss 0.523448     valid loss 0.520902     valid acc
1.000000
Epoch 098        train loss 0.522908     valid loss 0.521286     valid acc
1.000000
Epoch 099        train loss 0.523518     valid loss 0.520981     valid acc
1.000000
Epoch 100        train loss 0.523136     valid loss 0.524273     valid acc
1.000000
Test loss 0.521692       test acc 1.000000
```