HLS4ML
Prakticky

Miroslav Skrbek

Datová množina IRIS

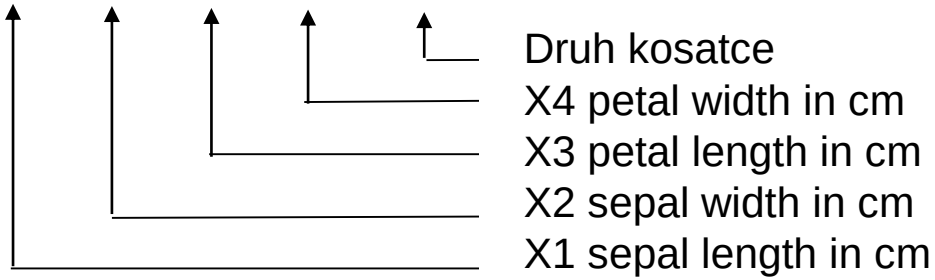
Nejpopulárnější databáze užívaná k testování algoritmů vůbec.

Zdroj dat: UCI databaze (http://archive.ics.uci.edu/ml/datasets)

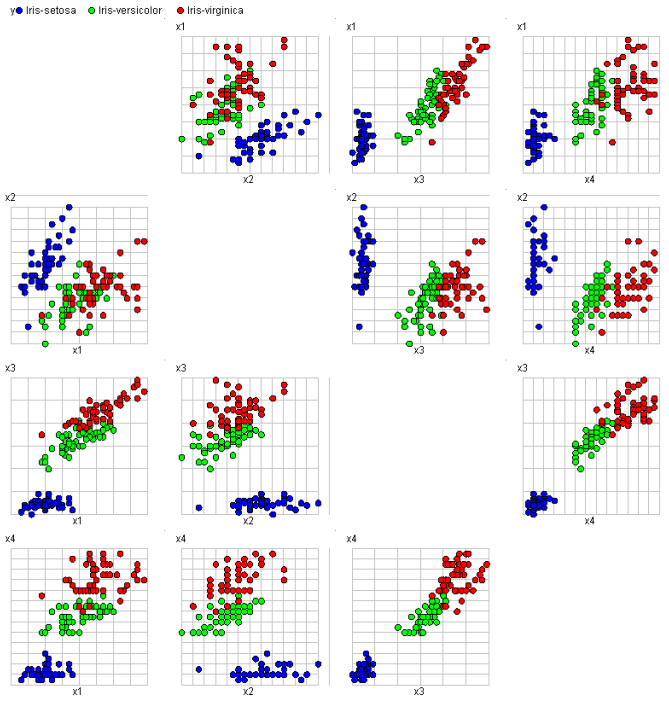
Fisher,R.A. "The use of multiple measurements in taxonomic problems. Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to Mathematical Statistics" (John Wiley, NY, 1950)..

Cílem je vytvořit neuronový klasifikátor pro tuto datovou množinu

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
7.0,3.2,4.7,1.4,Iris-versicolor
6.4,3.2,4.5,1.5,Iris-versicolor
7.7,2.6,6.9,2.3,Iris-virginica
6.0,2.2,5.0,1.5,Iris-virginica
```

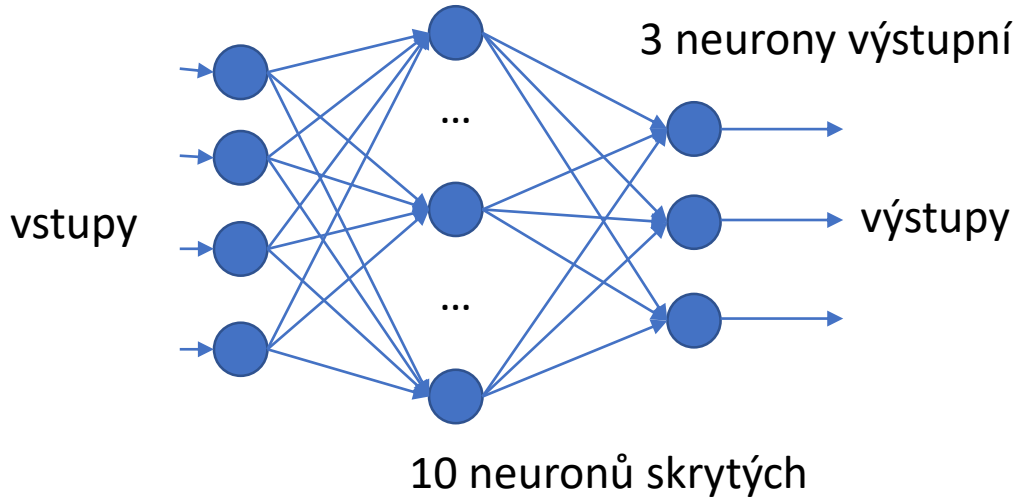


Druh kosatce
X4 petal width in cm
X3 petal length in cm
X2 sepal width in cm
X1 sepal length in cm



Neuronová síť

Dopředná neuronová síť 2 vrstvy



KERAS model v Pythonu

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Input(shape=(4,)))
model.add(tf.keras.layers.Dense(10, activation=tf.nn.sigmoid))
model.add(tf.keras.layers.Dense(3, activation=tf.nn.sigmoid))
```

Učení neuronové sítě

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.5),
              loss="mean_squared_error", metrics=['accuracy'])

model.fit(x_train_norm, y_train,
          batch_size=len(x_train_norm), epochs= 1000, verbose=2)

y_predict = model.predict(x_train_norm)

print("Confussion matrix")
m = np.array([0.0,0.0,0.0])
mz = np.outer(m,m)
n = len(y_train)
for i in range(n):
    p = np.round(y_predict[i])
    t = np.round(y_train[i])
    mz += np.outer(p,t)
print(mz)
```

```
Confussion matrix
      se    vi   ve
se [[49.   1.   0.]
vi  [ 1.  49.   0.]
ve  [ 0.   0.  50.]]
```

Transformace KERAS modelu do zdrojového kódu v jazyce C

Datový typ pro vstupy, výstupy, váhy a prahy

```
config = hls4ml.utils.config_from_keras_model(model, granularity='model',
default_precision='float')

hls_model = hls4ml.converters.convert_from_keras_model(
                    model, hls_config=config,
                    project_name='hls_iris', output_dir='hls_iris_prj',
fpga_part='xc7z010clg400-1')
hls_model.config.config['IOType'] = 'io_serial'
hls_model.write()
```

HLS4ML extrahuje dataflow graf, kde vrstvy jsou uzly grafu, v našem případě je to žetízek vrstev od vstupní po výstupní.

Typ rozhraní TOP funkce (v tomto případě axis)

Vitis projekt (hls_iris.cpp)

TOP funkce pro syntézu

```
void hls_iris(
    input_t input_1[N_INPUT_1_1],
    hls::stream<stream_data_t>& output_1,
    unsigned short &const_size_in_1,
    unsigned short &const_size_out_1
) {
        #pragma HLS INTERFACE s_axilite port=return
        #pragma HLS INTERFACE axis port=input_1, output_1
        #pragma HLS DATAFLOW

        const_size_in_1 = N_INPUT_1_1;
        const_size_out_1 = N_LAYER_4;
```

Tělo TOP funkce (pokračování)

`net::dense` a `net::sigmoid` jsou funkce definované v knihovnách HLS4ML

```
// NETWORK INSTANTIATION

layer2_t layer2_out[N_LAYER_2];
#pragma HLS ARRAY_PARTITION variable=layer2_out complete dim=0
nnet::dense<input_t, layer2_t, config2>(input_1, layer2_out, w2, b2); // dense

layer3_t layer3_out[N_LAYER_2];
#pragma HLS ARRAY_PARTITION variable=layer3_out complete dim=0
nnet::sigmoid<layer2_t, layer3_t, sigmoid_config3>(layer2_out, layer3_out); // dense_sigmoid

layer4_t layer4_out[N_LAYER_4];
#pragma HLS ARRAY_PARTITION variable=layer4_out complete dim=0
nnet::dense<layer3_t, layer4_t, config4>(layer3_out, layer4_out, w4, b4); // dense_1

result_t layer5_out[N_LAYER_4];
#pragma HLS ARRAY_PARTITION variable=layer5_out complete dim=0
nnet::sigmoid<layer4_t, result_t, sigmoid_config5>(layer4_out, layer5_out); // dense_1_sigmoid
```

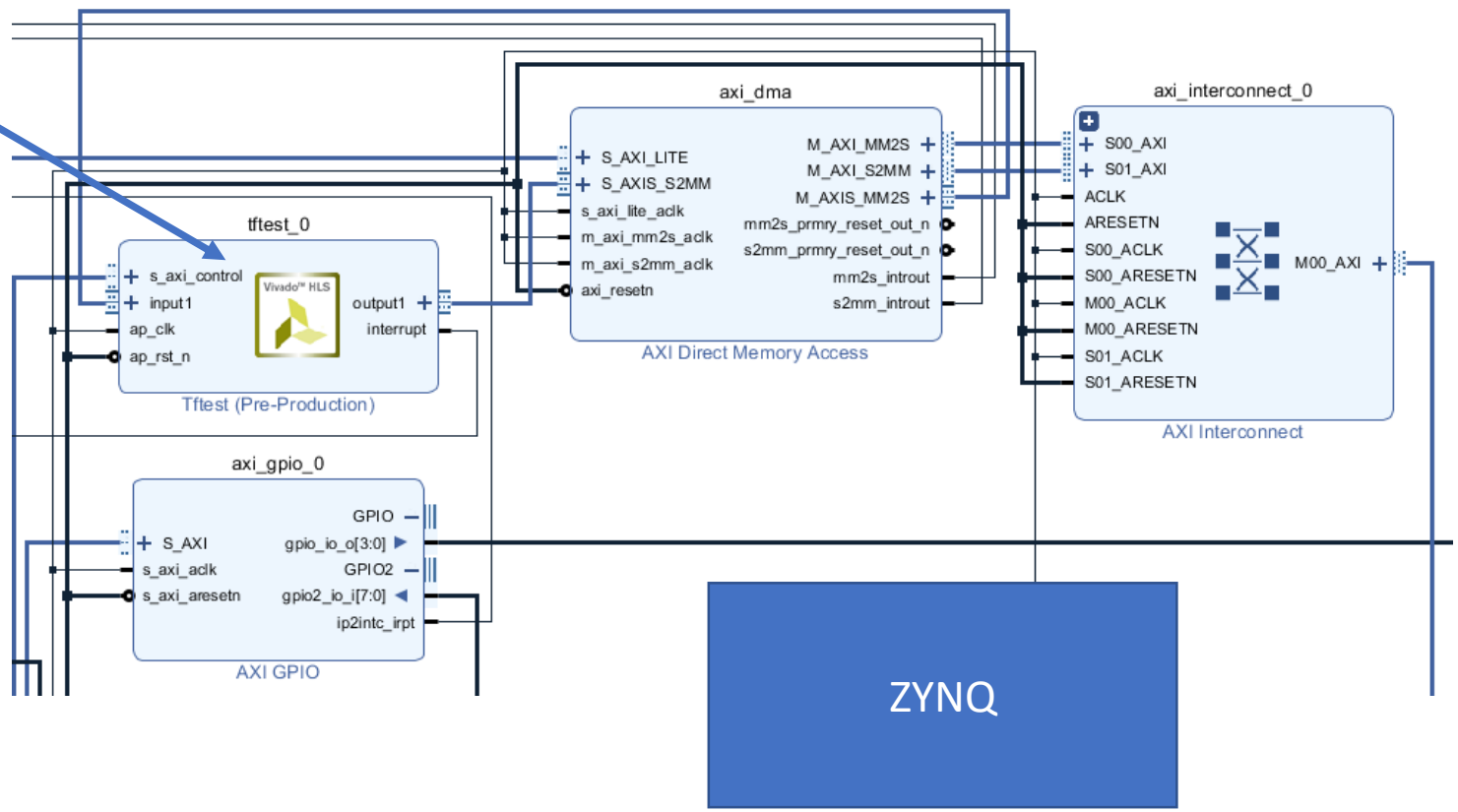
Tělo TOP funkce (pokračování)

Odeslání dat do výstupního streamu a
zajištění generování signálů TKEEP, TVALID

```
    int i;
    for(i = 0; i < N_LAYER_4; i++) {
          stream_data_t r;
          r.set_data(*((int*)&layer5_out[i]));
          r.set_last(i == (N_LAYER_4 - 1));
          r.keep_all();
          output_1.write(r);
    }
    // konec TOP funkce hls_iris
}
```

Vitis generuje IP Core

DMA blok zajišťuje přenosy dat mezi akcelerátorem a ARM systémem



Interface akcelerátoru v PYTHONU (PYNQ)

Inicializace

```
def __init__(self, overlay_name):
                o = Overlay(overlay_name)
               self.dma_nn = o.axi_dma
                self.acc_nn = o.tftest_0
                self.xlnk = Xlnk()
                self.nn_in = self.xlnk.cma_array(shape=(4,), dtype=np.float32)
                self.nn_out = self.xlnk.cma_array(shape=(3,), dtype=np.float32)
```

Alokace DMA bufferů

Predikce

```
def predict(self, x):
        for i in range(len(x)):
                self.nn_in[i] = x[i]
        self.dma_nn.sendchannel.transfer(self.nn_in)
        self.dma_nn.recvchannel.transfer(self.nn_out)
        self.acc_nn.mmio.write(0,1)
        if self.dma_nn.sendchannel.running:
                self.dma_nn.sendchannel.wait()
        if self.dma_nn.recvchannel.running:
                self.dma_nn.recvchannel.wait()
        return np.array(self.nn_out)
```

Přenos dat do vstupního bufferu

Start akcelerátoru

Čekání na dokončení DMA přenosů

Děkuji za pozornost