

Kapitola 11 - Přístupy k učení RBF sítě

Demonstrace použití různých způsobů učení RBF sítě.

Načtení knihovny NeuralNetworks

Nejdříve načteme knihovnu neuronových sítí.

In[102]:=

```
<< NeuralNetworks`
```

Pokud pracujete v Mathematic 8.0, vypněte ještě zobrazování chybové hlášky Remove::rmnsm. Tuto hlášku vyhazují funkce knihovny NeuralNetworks. Na funkci knihovny toto nemá žádný vliv.

In[103]:=

```
Off[Remove::rmnsm]
```

Import dat

■ Načtení dat ze souboru

Nastavíme si pracovní adresář na ten, kde máme uložen aktuální notebook, načítaná data musejí být ve stejném adresáři.

In[104]:=

```
SetDirectory[NotebookDirectory[]]
```

Out[104]=

```
D:\Dokumenty\BP
```

A načteme data.

In[105]:=

```
data = Import["iris.data"];
```

■ Načtení dat z internetu

Jiná možnost je importovat data přímo z internetu - příklad pro UCI databázi (stejná data jako při načítání ze souboru, jen načtena přímo z internetu).

In[106]:=

```
data =  
  Import["http://ftp.ics.uci.edu/pub/machine-learning-databases/iris/iris.data"];
```

Předzpracování dat

Provedeme předzpracování dat pomocí stejného postupu, jaký je popsán v kapitole 5 - Dopředná síť a Iris data.

In[107]:=

```
data2 = Drop[data, -1];
inData = data2[[All, 1 ;; 4]]; (*vstupní parametry*)
outDataTmp = data2[[All, 5]]; (*výstupní parametr*)
outVal = Tally[outDataTmp][[All, 1]];
encode = MapIndexed[#1 -> Normal[SparseArray[#2 -> 1, {Length[outVal]}]] &, outVal];
outData = Flatten[outDataTmp] /. encode;
```

Narozdíl od předchozího notebooku si data rozdělíme na trénovací a testovací množinu. Na trénovací množině budeme síť učit a na testovací množině budeme vyhodnocovat její úspěšnost.

Provedeme náhodné rozdělení dat v poměru 30% testovací a 70% trénovací.

In[113]:=

```
testPercent = 30; (*procent pro testovací data*)
inDataDimension = 4; (*dimenze vstupnich dat*)
classes = 3; (*pocet trid v datech*)
completeData = Join[inData, outData, 2];
completeData = RandomSample[completeData];
vectors = Dimensions[completeData][[1]];
testData = completeData[[1 ;; IntegerPart[vectors * testPercent / 100]]];
trainData = completeData[[IntegerPart[vectors * testPercent / 100] + 1 ;; vectors]];
```

Podívejme se kolik máme trénovacích a kolik testovacích vektorů.

In[121]:=

```
Dimensions[trainData][[1]]
Dimensions[testData][[1]]
```

Out[121]=

105

Out[122]=

45

Nyní ještě rozdělíme trénovací data na učící a validační množinu, v poměru 70% učící data a 30% validační data. Na učících datech budeme síť učit a na validačních datech budeme kontrolovat zda se na nich klasifikace zlepšuje. Pokud by se výsledky sítě na validačních datech začaly zhoršovat, přeručíme učení sítě. Tím zabráníme jejímu přeučení (přílišnému přizpůsobení se trénovacím datům).

In[123]:=

```
validatePercent = 30;
vectors2 = Dimensions[trainData][[1]];
validationData = trainData[[1 ;; IntegerPart[vectors2 * validatePercent / 100]]];
learnData = trainData[[IntegerPart[vectors2 * validatePercent / 100] + 1 ;; vectors2]];
```

Rozdělíme si každou vytvořenou množinu dat na vstupní a výstupní část.

In[127]:=

```
testDataIn = testData[[All, 1 ;; inDataDimension]]; (*vstupni parametry 1 až 4*)
testDataOut = testData[[All, inDataDimension + 1 ;; inDataDimension + classes]];
(*vystupni zakodovane parametry 5 až 7*)
trainDataIn = trainData[[All, 1 ;; inDataDimension]];
trainDataOut = trainData[[All, inDataDimension + 1 ;; inDataDimension + classes]];
validationDataIn = validationData[[All, 1 ;; inDataDimension]];
validationDataOut =
  validationData[[All, inDataDimension + 1 ;; inDataDimension + classes]];
learnDataIn = learnData[[All, 1 ;; inDataDimension]];
learnDataOut = learnData[[All, inDataDimension + 1 ;; inDataDimension + classes]];
```

Nyní máme připraveny data k učení sítě.

Výběr a naučení neuronové sítě

Protože nevíme přesně kolik neuronů v síti bude ideální a tipovat by bylo riskantní, použijeme křížovou validaci (podrobněji popsána v samostatné kapitole), která nám určí jaká z předložených sítí podává nejlepší výsledky. Vytvoříme tedy užší výběr tří sítí a ty budeme testovat pomocí křížové validace. Obecně se křížová validace používá k nastavení nejlepších parametrů algoritmu (např. počet neuronů), zde tedy k zjištění struktury sítě.

In[135]:=

```
net1 = InitializeRBFNet[inData, outData, 1];  
net2 = InitializeRBFNet[inData, outData, 3, Neuron -> Sigmoid];  
net3 = InitializeRBFNet[inData, outData, 6, Neuron -> Exp];
```

Zavedeme si funkci crossvalidate, jíž dáme jako parametry síť, vstupní data, výstupní data a počet foldů. Funkci je potom možné spustit příkazem crossvalidate[parametry]. Funkce vrátí seznam výsledků sítě pro jednotlivé foldy.

In[138]:=

```

crossvalidate[yourNet_, yourInData_, yourOutData_, yourFolds_] :=
  (Off[NeuralFit::StoppedSearch];
   yourVectors = Take[Dimensions[yourInData], 1];
  inDataDimensions = Take[Dimensions[yourInData], -1];
   outDataDimensions = Take[Dimensions[yourOutData], -1];
   yourVectorsInFold = IntegerPart[yourVectors / yourFolds];
   yourCompleteData = Join[yourInData, yourOutData, 2];
  yourCompleteData = RandomSample[yourCompleteData];
   yourBest = 1;
  yourWorst = 0;
  yourSum = 0;
   yourResults = {};
   For[i = 1, i <= yourFolds, i++, (*kod jednoho kola krossvalidace*)
  yourValidationData = yourCompleteData[[
    1 + (i - 1) * yourVectorsInFold[[1]] ;; i * yourVectorsInFold[[1]]]];
  yourLearnData = Drop[yourCompleteData,
    {1 + (i - 1) * yourVectorsInFold[[1]], i * yourVectorsInFold[[1]]}];
  yourValidationDataIn = yourValidationData[[All, 1 ;; inDataDimensions[[1]]]];
   yourValidationDataOut = yourValidationData[[All,
    inDataDimensions[[1]] + 1 ;; inDataDimensions[[1]] + outDataDimensions[[1]]]];
  yourLearnDataIn = yourLearnData[[All, 1 ;; inDataDimensions[[1]]]];
  yourLearnDataOut = yourLearnData[[All,
    inDataDimensions[[1]] + 1 ;; inDataDimensions[[1]] + outDataDimensions[[1]]]];
  {netX, record} = NeuralFit[yourNet, yourLearnDataIn, yourLearnDataOut,
    yourValidationDataIn, yourValidationDataOut,
    CriterionLog -> False, CriterionPlot -> False];
  current = Take[CriterionValidationValues /. record[[2]], -1];
  If[current[[1]] < yourBest, yourBest = current[[1]], yourBest = yourBest];
  If[current[[1]] > yourWorst, yourWorst = current[[1]], yourWorst = yourWorst];
  yourSum = yourSum + current;
   AppendTo[yourResults, current[[1]]];
   x = x + 1;
  (*Print[i ". fold = ", current[[1]]];*)
  (*odkomentujte pokud chcete vypsat vysledek pro kazdy fold*)
];
  Return[yourResults];
)

```

Pomocí funkce crossvalidate provedeme křížovou validaci našich třech sítí a výsledky, pro snadné porovnání, zobrazíme do jednoho grafu.

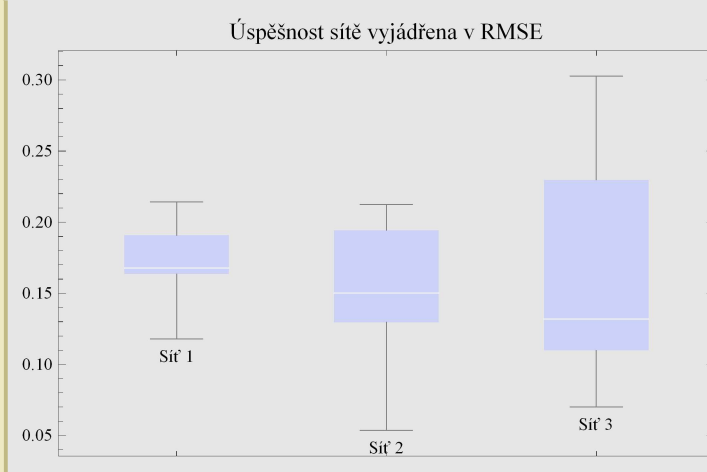
In[139]:=

```

folds = 10;
x = 0;
Monitor[
  g1 = crossvalidate[net1, trainDataIn, trainDataOut, folds];
  g2 = crossvalidate[net2, trainDataIn, trainDataOut, folds];
  g3 = crossvalidate[net3, trainDataIn, trainDataOut, folds];
  BoxWhiskerChart[{g1, g2, g3}, PlotLabel → "Úspěšnost sítě vyjádřena v RMSE",
    ChartLabels → Placed[{"Síť 1", "Síť 2", "Síť 3"}, Below],
    ProgressIndicator[Dynamic[x], {0, 3 * folds}]]

```

Out[141]=



Z grafu je vidět, že sítě podávají na těchto datech obdobné výkony, můžeme tedy vybrat libovolnou z nich. Vybereme tedy síť 2.

In[142]:=

```

bestNet = InitializeRBFNet[inData, outData, 3, Neuron → Sigmoid];

```

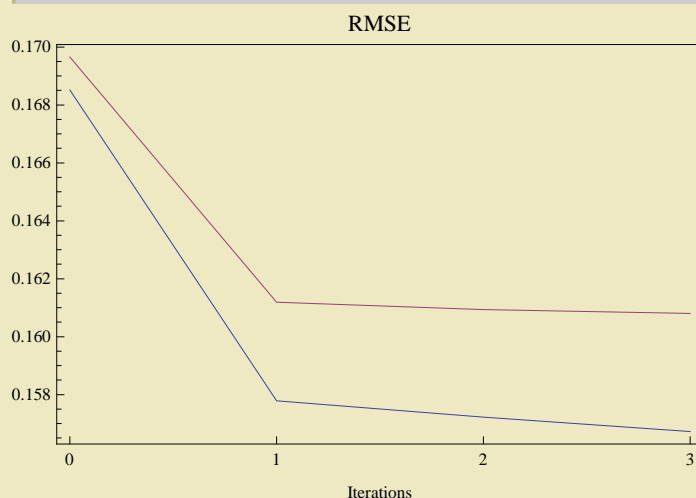
Naučíme tuto síť na učicích datech (podmnožina trénovacích dat) a její učení přerušíme v okamžiku kdy se její výsledky začnou na validační množině zhoršovat.

In[143]:=

```

{learnedBestNet, record2} = NeuralFit[bestNet,
  learnDataIn, learnDataOut, validationDataIn, validationDataOut];

```



Nyní máme naučenou síť, zbývá nám zjistit její skutečnou chybu. Hodnota RMSE, která nám reprezentuje chybovost sítě během učení, není pro člověka tak snadno uchopitelná jako hodnota úspěšnosti v procentech. Vypočteme si tedy skutečnou chybovost sítě v procentech. K tomu použijeme testovací

data. Necháme síť oklasifikovat testovací množinu dat a spočteme kolik procent dat klasifikovala správně.

Zavedeme si funkci `percentage`, která přijímá jako parametry neuronovou síť, vstupní testovací data, výstupní testovací data a vrací úspěšnost sítě v procentech.

In[144]:=

```
percentage[inNet_, testDatIn_, testDatOut_] := (
  testVectors = Dimensions[testDatIn][[1]];
  good = 0; (*pocet spravne oklasifikovanych*)
  classes = {};
  For[i = 1, i <= testVectors, i++,
    class = Round[inNet[testDatIn[[i]]]];
    AppendTo[classes, class];
    If[class == testDatOut[[i]], good = good + 1, good = good];
  ];
  Return[good / testVectors * 100];)
```

Použijeme tuto funkci k určení skutečné úspěšnosti naší sítě na testovacích datech.

In[145]:=

```
N[percentage[learnedBestNet, testDataIn, testDataOut], 5]
```

Out[145]=

```
91.111
```

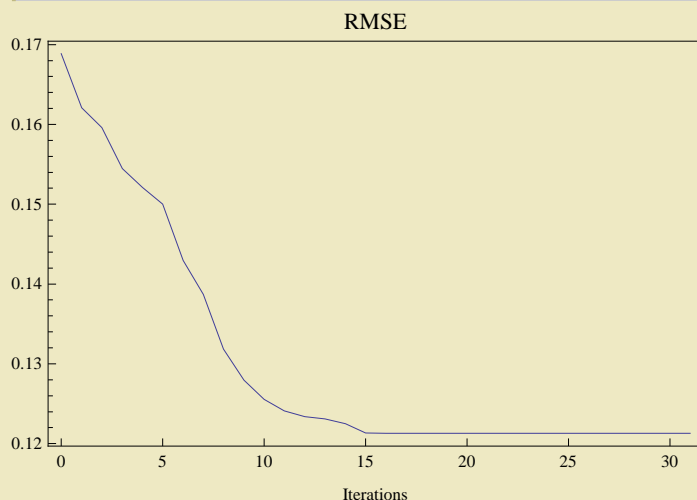
Jiný způsob učení sítě

Možná si říkáte, že rozdělovat trénovací data na učicí a validační je zbytečné. Následuje tedy ukázka postupu učení sítě jen s trénovací množinou a její zhodnocení pomocí testovací množiny.

Naučíme stejnou síť jakou jsme učili s validační množinou (i na počátku stejně zinicizovanou), ale naučíme ji na všech trénovacích datech.

In[146]:=

```
{learnedBestNet2, record2} = NeuralFit[bestNet, trainDataIn, trainDataOut, 50];
```



A podíváme se na její skutečnou úspěšnost na testovacích datech.

In[147]:=

```
N[percentage[learnedBestNet2, testDataIn, testDataOut], 5]
```

Out[147]=

91.111

Pokud bychom se rozhodli nerozdělovat data ani na testovací a trénovací množinu (viz. předchozí notebook) nemáme již možnost určit skutečnou úspěšnost sítě.

Prohlášení

Tento text byl vypracován jako součást bakalářské práce Adama Činčury "Demonstrační aplikace pro podporu kurzu neuronových sítí" na FEL ČVUT 2011.