proteinformatics

Machine Learning

Lecture:
https://renedominik.github.io/teaching/machine-learning/

Dr. René Staritzbichler

2021

Practical part

- Toy example: learning sinus
- Biological example: helix propensity for a sequence

First example

- Create data: random pairs of x and sin(x)
  - $ cd ann_sinus/
  - Open create_data.py, x should range from -50 to 50
    - Check which lines need to be commented out
  - $ ./create_data.py > train_data.txt
- Train artificial neural network
  - Open train_ann.py, hidden should be (5,8,5), 200 iterations
  - $ ./train_ann.py

$ANN(x) = y_{pred}$, $ANN : \mathbb{R} \rightarrow \mathbb{R}$

First example

- Visualize outcome
  - $ gnuplot
  - > plot “test_predictions.txt” using 1:2 with points pt 6, “” us 1:3 with points pt 4
- Re-train artificial neural network
  - Open train_ann.py, set hidden (55,85,55), 1000 iterations
  - $ time ./train_ann.py

First example

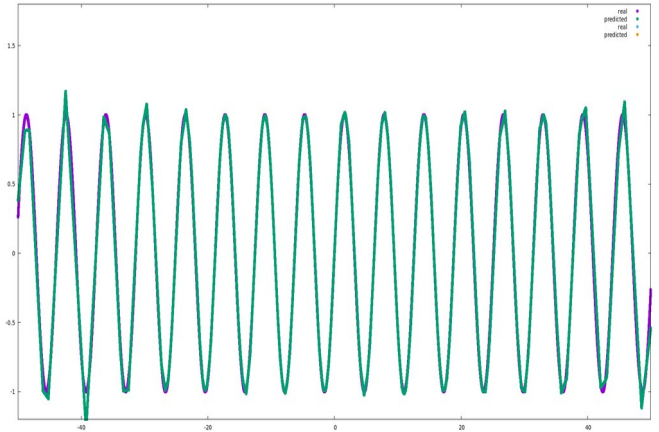
- Visualize outcome as above
  - Sinus should be described pretty well
- Test predictive power
  - So far, our test was only using random values in the same range of x
  - Predictions are interested in the “future”
    - Stock markets
    - Weather forecast
    - MANY more...
- Open create_data.py, and plot 500 data points in the range of 50 to 70
  - $ ./create_data.py > outside_test.txt

First example

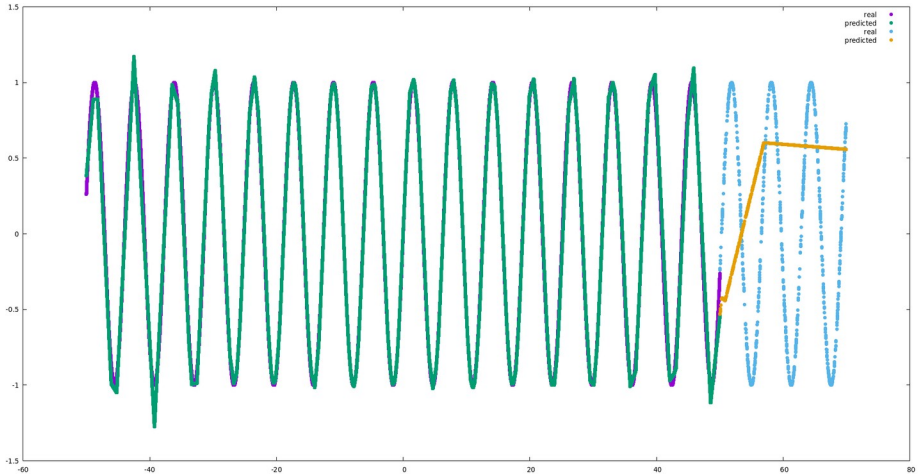
- Predict for the new values
  - $ ./predict.py
- Visualize ‘old’ and ‘new’ data together
  - $ gnuplot
  - > plot “test_predictions.txt” us 1:2 w p pt 4, “” us 1:3 w p pt 5, “outside_predictions.txt” us 1:2 w p pt 4, “” us 1:3 w p pt 5
- How would you judge the predictive power?

proteinformatics

First example



Looks nice for training intervall …



but it has absolutely no predictive value!!!

Conclusions

- The first setup has absolutely no predictive power
- It learned the shape by heart, but no principle

- One has to ask the right question
- A main part of training learning algorithms is to find the right perspective
- The way data is presented is crucial !
- What would you change?

Second approach

- Before we used x as input and predicted y

$$ANN(x) = y_{pred}, \quad ANN : \mathbb{R} \rightarrow \mathbb{R}$$

- We need to know the history to predict the future
- Actually x does not matter that much
  - Example: when predicting stocks (x is then time) you don't care primarily about when a certain trend appears
- Next we use (y_1, …, y_n) as input and predict y_{n+1}

$$ANN(y_1, \ldots, y_n) = y_{n+1,\, pred}, \quad ANN : \mathbb{R}^n \rightarrow \mathbb{R}$$

Second approach

- $ cd ../sinus_array
- Open create_data.py
  - Starting from a similar random vector we will create arrays of 11 subsequent points
  - 10 will serve as input and the last is used as reference to compare to predicted values
  - Note that we need much fewer data points (1000 instead of 100000)
- $ ./create_data.py

Second approach

- Open train.py
  - Note, that we can use a much smaller network ((10,15,10) instead of (55,85,55))
  - Note, that we need much fewer iterations (100 instead of 1000)
  - Reading of the data differs, otherwise the same training
  - Evaluation is done systematically from -50 to 70
- $ ./train_ann.py
  - Note the speed
- Visualize as before

Conclusions

- Clearly, this way of presenting the data has improved
  - Training efficiency
  - Accuracy
  - Predictive power

Biological example

- Predicting helices from a protein sequence
- Challenging task – LA has to ‘understand biochemistry’
- Accuracy cannot be expected as high as in toy example
- Already data collection is significantly more complex
- Many ways of presenting data (‘ALA’: 0.34, ‘ASP’: -2.73, ..)
- Real topic

Helix predictor

- Reference data: PDB
  - PDB files contain both sequence as well as helix information
- Sequence to vector:
  - translate amino acids to descriptors
- Train ANN:
  - Input: descriptors
  - Output: helix probability

Prediction procedure

- convert_pisces.py
  - Translate individual amino acids into features
- create_db.py:
  - Create data matrix X for sequence windows with helix probability as last value
- train_ann.py
  - Input data matrix X
  - Output predictions y

NOTE: if you change anything in convert or create, you have to adjust **ALL** subsequent steps !!!

Helix predictor :: get data

- $ cd ../helix_predictor/
- Non-redundant list of proteins
  - Pisces server: max 50% identity
  - $ less cullpdb_pc50_res3.0_R1.0_d200828_chains29260
- Copy PDB files of Pisces list (use one of the following options):
  - USB stick: $ cp -r /media/USER/Bigbelly/pdb.tgz .
  - http://proteinformatics.uni-leipzig.de/document_server/ download ‘helix predictor data’
- $ tar xzf pdb.tgz

Helix predictor :: translate data

- Open ‘convert_pisces.py’
  - Select pair of mode and output directory
- Translate pdbs to profiles (features)

  $ ./convert_pisces.py # calls pdb2dat.py
- Open pdb2dat.py
  - It extracts from each PDB both sequence and helix information
  - **Translates AA type to descriptors**, different choices:
    - sorted by hydrophobicity scale
    - grouping into hydrophobic (aromatic,others), polar, neg, pos, special cases
    - profile: id, polar, pos/neg, aromatic, rest/gly/cys/his/pro , (kyte-doolittle)
- Task: think of other simple ways of translating amino acids into descriptors

Crucial!

Helix predictor :: training data

$ ./create_db.py

- Predictions will be performed for sequence of fixed size
  - Sliding window for longer sequences
  - ANN: predict helix propensity for a given sequence window
- We have to collect a training database that contains
  - *n*l* descriptor vector

    (*n* number of descriptors per AA, *l* length of sequence window)
  - Binary classifier: helix or not
    - Related to center AA!!

14 19 1 18 6 11 19 13 3 20 11 0

Helix predictor :: setup ANN

- Open train_ann.py
  - Reads sequence descriptors into matrix X
  - Reads reference classifications (helix/other) into vector y
  - Splits both X and y into train and test subsets
  - Calculates scaling by x_train, applies the same to x_test (normalization!)
  - Setup of ANN
    - Architecture given as ‘hidden = (350)’
    - Iterations

Helix predictor :: train and evaluate

- $ ./train_ann.py
- Understand the quality of the model: confusion matrix

		Actual class	
		P	**N**
Predicted class	**P**	**TP**	FP
	N	FN	**TN**

Correct predictions: true positive, true negative

False predictions: false positive, false negative

Sensitivity: $\frac{TP}{P} = \frac{TP}{TP + FN}$

Specificity: $\frac{TN}{N} = \frac{TN}{TN + FP}$

Accuracy: $\frac{TP + TN}{P + N} = \frac{TP + TN}{TP + TN + FP + FN}$

Sklearn outputs the confusion matrix with actual categories as rows, predicted as cols !!

Things you could try

- Modify used data
  - Use different scales, features in convert_pisces.py
  - Window size, increment in create_db.py
- Adjust algorithm in train_ann.py
  - Number and size of hidden layers
  - Iterations
  - With without data scaling, regularization
- Aim is to improve prediction quality

The End