Manifold Learning (or Nonlinear Dimension Reduction) with Laplacian eigenmaps and more (?)

Krzysztof Czarnowski

October 3rd, 2016

(Contains some joint work with Aleksandra Hernik)

Sources

This is based mainly on papers of Niyogi, Belkin and others from 2002-2008 and later. (Regretfully, Partha Niyogi passed in 2010 at the age of 47. His main interests in the general field of AI were manifold learning and evolutionary linguistics. Interestingly some of his late papers were co-authored by Steven Smale (sic!))

[1] *Laplacian Eigenmaps for Dimensionality Reduction and Data Representation*, M. Belkin and P. Niyogi. Neural Computation, 15(6):1373-1396, June 2003. `http://people.cs.uchicago.edu/~niyogi/papersps/BNeigenmap.pdf`

[2] *Manifold Regularization: a Geometric Framework for Learning from Examples*, M. Belkin, V. Sindhwani and P. Niyogi, University of Chicago CS Tech. Report , TR-2004-06, 2004 Journal of Machine Learning Research, Vol 7, 2399-2434, 2006 `http://people.cs.uchicago.edu/~niyogi/papersps/BNSJMLR.pdf`

Sources continued . . .

[3] *Intrinsic Fourier Analysis on the Manifold of Speech Sounds*, A. Jansen and P. Niyogi. Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, Toulose, France, 2006. `http://people.cs.uchicago.edu/~niyogi/papersps/JanNiyicassp.pdf`

But apart from Laplacian spectral methods (sometimes called ISA, Intrinsic Spectral Analysis) there are numerous other approaches! Isomap, LLE (Locally Linear Embedding), Manifold Alignment, HLLE, MLLE, LTSA . . .

Manifold concept

M
manifold
variété
mannigfaltigkeit
rozmaitość

- A model for a “curved space” of arbitrary *internal* dimension (usually finite), a “hypersurface” (like hyperspace but well . . . possibly curved).
- Examples: circle, sphere, sphere with handles, torus, Möbius strip, Klein bottle . . .
- Usually visualized as embedded in some $\mathbb{R}^H$ (preferably $\mathbb{R}^3$ ☺)
- But may be defined regardless of the embedding (abstract) and studied internally for it’s geometry, analytical properties . . .

Manifold learning basic idea

- Meaningfull data often falls in some high dimensional *observation space* $\mathbb{R}^H$...
- ... but usually groups along low dimensional structures.
- For example think about pose estimation

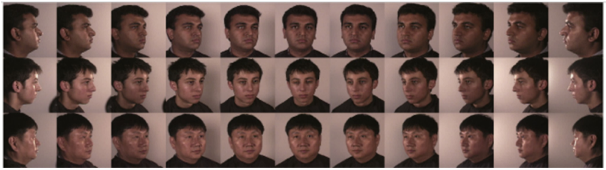


Fig. 2. The head images in the FacePix dataset.

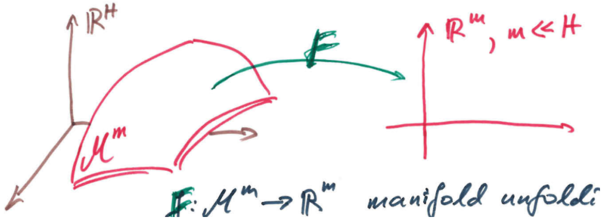
- It's natural to consider these are some low dimensional manifolds embedded in $\mathbb{R}^H$ (like circles for images above).
- Modelling such sub-manifolds is kind of *nonlinear PCA*.
- Dimensionality reduction is usually (?) good...

What we want?

- We want to construct mapping

$$F : \mathbb{R}^H \supset \mathcal{M} \rightarrow \mathbb{R}^m$$

where $m \ll H$ which is *faithfull* to the geometry of $\mathcal{M}$



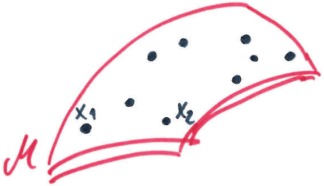
- or a bunch of functions:

$$F = (f_1, f_2, \dots f_m), \quad f_i : \mathcal{M} \rightarrow \mathbb{R}$$

- Some kind of proper flattening of $\mathcal{M}$, not necessarily unfolding.

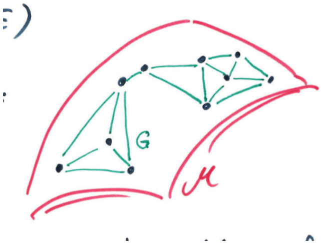
Graph on training data-set

- Training data $(x_1, x_2, \ldots x_N)$ is probing the manifold $\mathcal{M}$ (with noise).



- To get the geometry we build K-nearest neighbours graph $G = (V, E)$ on the training data.

$V = \{x_1, x_2, \ldots x_N\}$
$(x_i, x_j) \in E \Leftrightarrow$
x_i is among K-nearest neighbours of x_j or x_j is among K-nearest neighbours of x_i



Weights on the edges

- We usually apply weights to edges:

$$W_{ij} = e^{-|x_i - x_j|^2/t} \quad \text{if } (x_i, x_j) \in E$$

with some $t > 0$ and $W_{ij} = 0$ otherwise.
- Weighted adjacency matrix W is symmetric, possibly large but sparse.
- With weights we also get (generalized/weighted) vertex degrees:

$$D_i = \sum_k W_{ik}$$

but we prefer to put them into a diagonal degree matrix:

$$D_{ij} = \delta_{ij} \sum_k W_{ik}$$

(δ_{ij} is the Kronecker symbol).

Let's flatten the graph

- $F = (f_1, f_2, \dots f_m)$, let's focus on one of the f_l's and call it f.
- On the graph it's just a sequence of N values (one for each training vector x_i): $f : x_1 \mapsto y_1, x_2 \mapsto y_2, \dots x_N \mapsto y_N$
- To make f smooth on the graph (faithfull to the geometry) we want to minimize the following sum over the y_i's:

$$\sum_{ij} (y_i - y_j)^2 W_{ij}$$

- Obviously we need to prevent the collapse, so the actual optimization criterium for y_i's will be

$$\arg\min_{y,\ y^T D y = 1} \sum_{ij} (y_i - y_j)^2 W_{ij},\ y = [y_1, \dots y_N]^T,\ D_{ij} = \delta_{ij} \sum_k W_{ik}$$

(D is the graph vertex degree matrix).

transforming the sum

$$\sum_{ij}(y_i - y_j)^2 W_{ij} =$$

$$\sum_{ij} y_i^2 W_{ij} + \sum_{ij} y_j^2 W_{ij} - \sum_{ij} 2 y_i y_j W_{ij} =$$

$$2\left(\sum_i y_i^2 \sum_j W_{ij} - \sum_{ij} y_i y_j W_{ij}\right) =$$

$$2\left(\sum_{ij} y_i y_j \delta_{ij} \sum_k W_{ik} - \sum_{ij} y_i y_j W_{ij}\right) =$$

$$2\sum_{ij} y_i y_j \left(D_{ij} - W_{ij}\right) = 2y^T L y$$

where $D_{ij} = \delta_{ij} \sum_k W_{ik}, \; L = D - W$

Eigenvalue problem

- $L = D - W$ — graph laplacian, L is symmetric and positive-semidefinite (from the above calculation).
- So the optimization problem takes the form:

$$\underset{y,\ y^T Dy=1}{\arg\min}\ y^T Ly \tag{OP}$$

- The problem is related to the following eigenvalue problem:

$$Ly = \lambda Dy \tag{EP}$$

- Well... strictly speaking the solution to (OP) is the eigenvector which belongs to the smallest eigenvalue of (EP).
- However...

But not zero!

- However for each i:

$$
\begin{aligned}
(L\mathbb{1})_i &= \sum_j L_{ij} = \sum_j \left(\delta_{ij} \sum_k W_{ik} - W_{ij} \right) \\
&= \sum_k W_{ik} - \sum_j W_{ij} = 0
\end{aligned}
$$

  so the smallest eigenvalue is 0 and the corresponding eigenvector is „constant".

- We are not interested in so smooth the mapping, so we skip this one.
- (If the graph is not connected, then the zero eigenvalue is degenerate: one eigenspace dimension for every graph component. We would skip all of the eigenvectors, of course. However, let's assume the graph is connected.)

We like small but positive!

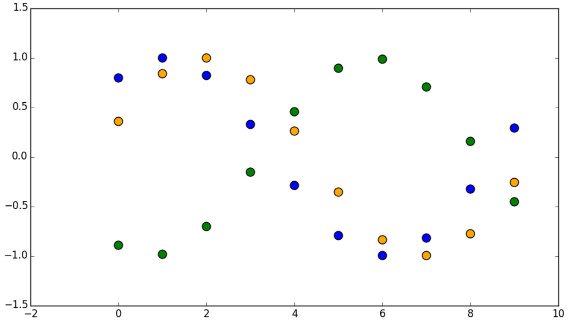
- Then we have the smallest positive eigenvalue λ_1 and its eigenvector
$$y^{(1)} = \left(y_1^{(1)}, y_2^{(1)}, \dots y_N^{(1)}\right)$$
which is orthogonal to the eigenspace of zero. This we take as the best flattening of the graph to one dimension:
$$x_1 \mapsto y_1^{(1)},\ x_2 \mapsto y_2^{(1)}, \dots\ x_N \mapsto y_N^{(1)}$$
- If we think that we need two dimensions, we add the eigenvector $y^{(2)}$ of the second smallest eigenvalue (orthogonal to all the previous eigenspaces) and we have the following mapping:
$$x_1 \mapsto (y_1^{(1)}, y_1^{(2)}),\ x_2 \mapsto (y_2^{(1)}, y_2^{(2)}), \dots\ x_N \mapsto (y_N^{(1)}, y_N^{(2)})$$
- And so on — take as many dimensions as you want!

A toy example

- Joint work with Ola Hernik.
- The idea was to try manifold learning on simple signals with natural low dimensional structures.
- And have fun, experiment and try ideas . . .
- and then maybe build some non-toy applications ☺

tone-phase data

Three example data vectors — tones of frequency $0.1 \cdot f_s$, 10-sample windows (exactly one period sampled 10 times) with random phase.

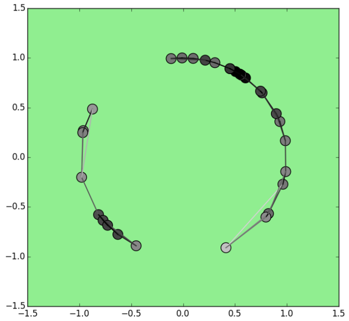


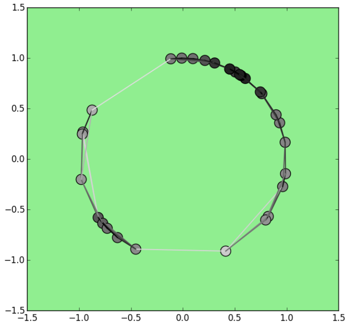
Observation space is 10-dimensional, but the actual observations fall on a one-dimensional submanifold — a circle.

graph and ground truth

Example graphs for a dataset of 30 random vectors:

- left: $K = 3$, $t = 2.0$ — graph has two components
- right: $K = 4$, $t = 2.0$ — graph is connected

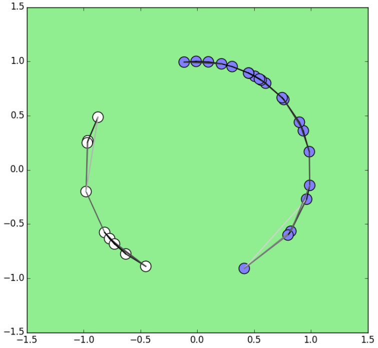


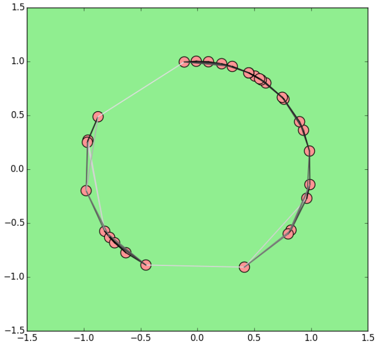


Vertices are plotted at "ground truth" points (corresponding to the actual signal phase) but distances are computed in the data space. Edge weights and vertex degrees are showed in shades of grey.

0-eigenfunction (first) on ground truth

For the same examples the first eigenfunction of the zero eigenvalue (first of two for the left graph, the only one for the right graph).

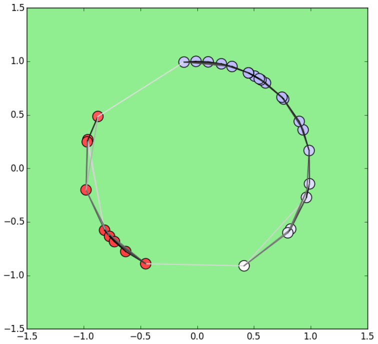


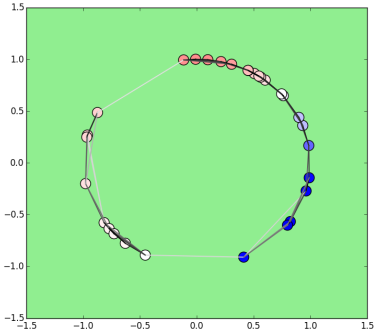


Vertex color shows the value for the vertex (edges as before).

first and second eigenfunctions

First and second eigenfunctions (for smallest positive eigenvalues) for the connected graph.

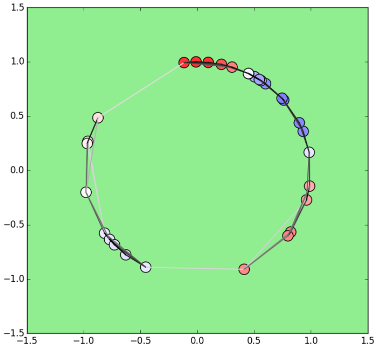


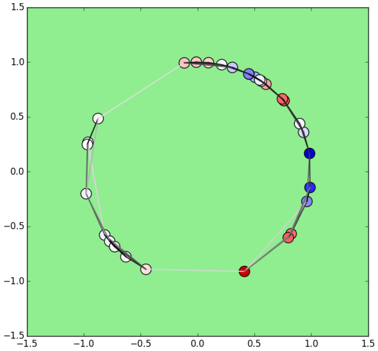


Vertex color shows the value for the vertex (edges as before).

for the curious . . .

Looking further ahead: three and six (we are counting from zero).

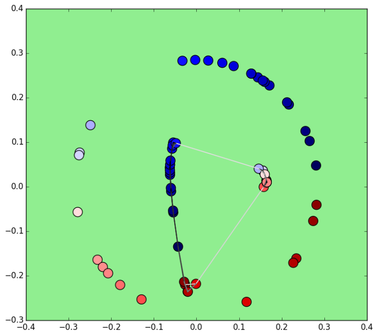


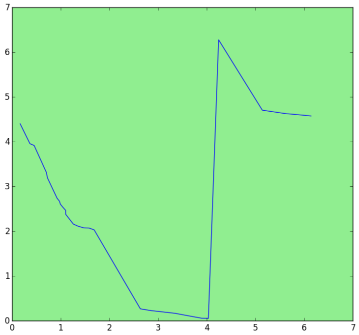


We are not interested in these . . . Probably . . .

let's celebrate!

Finally eigenfunctions 1 and 2 plotted against ground-truth on the left ☺ and resulting phase estimate
(vertical axis) plotted against the true phase (horizontal) on the right.

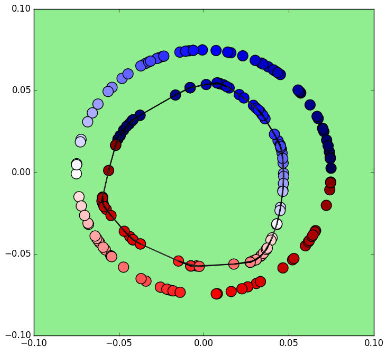


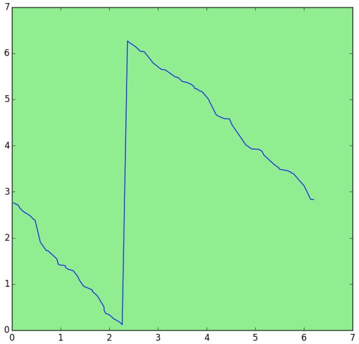


Maybe even nicer that expected?

More training data!

For 100 training points it's much better, as expected.





An example from [1]

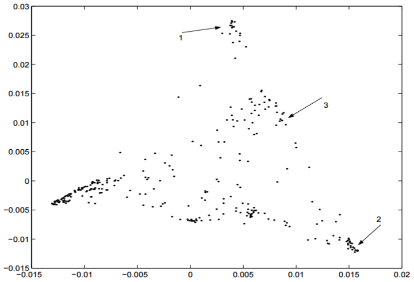
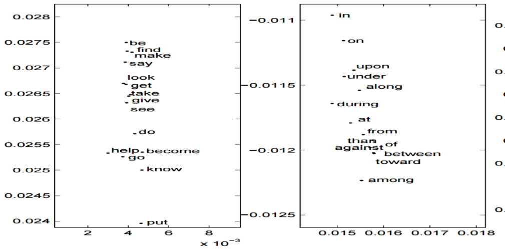


Figure 4: 300 most frequent words of the Brown corpus represented in the spectral domain.

An example from [1], continued



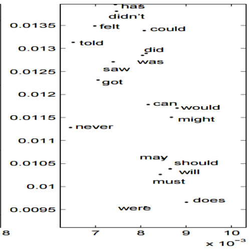


Figure 5: Fragments labelled by arrows, from left to right. The first is exclusively infinitives of verbs, the second contains prepositions and the third mostly modal and auxiliary verbs. We see that syntactic structure is well-preserved.

Another example from [1]

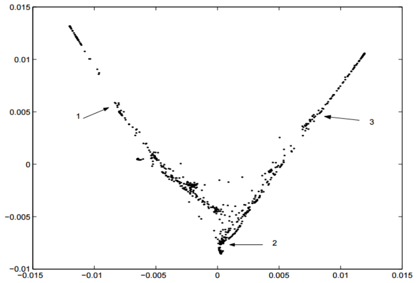


Figure 6: 685 speech data points plotted in the two dimensional Laplacian spectral representation.

Another example from [1], continued

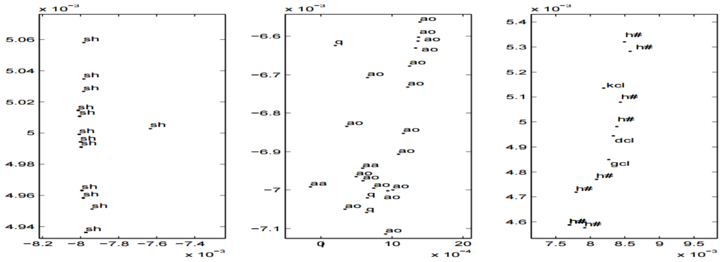


Figure 7: A blowup of the three selected regions (1,2,3) left to right. Notice the phonetic homogeneity of the chosen regions. The data points corresponding to the same region have similar phonetic identity though they may (and do) arise from occurrences of the same phoneme at different points in the utterance. The symbol “sh” stands for the fricative in the word she; “aa”,”ao” stand for vowels in the words dark and all respectively; “kcl”,”dcl”,”gcl” stand for closures preceding the stop consonants “k”,”d”,”g” respectively. “h#” stands for silence.

But hey! We need to go out-of-sample!

- So far we have no way to make predictions on unseen data!
- But we usually want to develop an estimator of, say, camera angle. Or a signal phase ...
- In [2] they extend eigenmaps from the graph to the ambient space using kernel regularizer and RKHS Representer Theorem methods. This leads to yet another optimization problem and an associated eigenvalue problem. (No details today.)
- The extended embedding functions are obtained as kernel train vector's sections' linear combinations

$$f(\cdot) = \sum_i \alpha_i K(x_i, \cdot), \quad \alpha \in \mathbb{R}^N$$

  where α is an eigenvector of the smallest, not yet utilized, positive eigevalue of the underlying eigenvalue problem.
- So same story again ...
- Not quite! The matrix is not sparse any more! Ooops! ☹
- But sometimes it's doable ...

An example from [2]

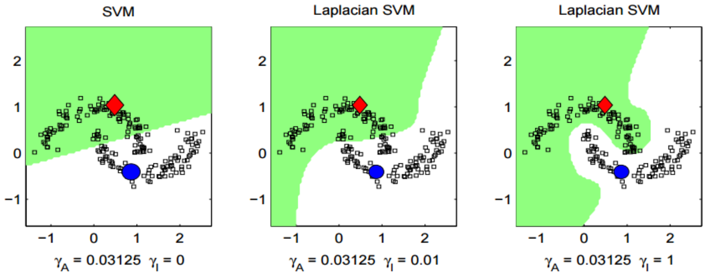


Figure 2: Laplacian SVM with RBF kernels for various values of γ_I. Labeled points are shown in color, other points are unlabeled.

A variant of manifold learning for semi-supervised classification problem — toy example ($K(x, y) = \exp(|x - y|^2/(2\sigma^2))$).

Speech ISA from [3]

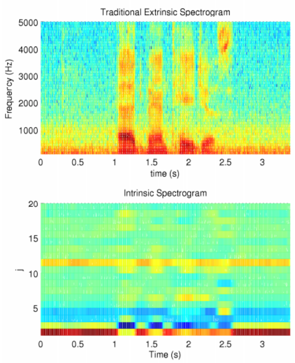


Fig. 1. Traditional (top) and intrinsic (bottom) spectrograms for the word “advantageous”.

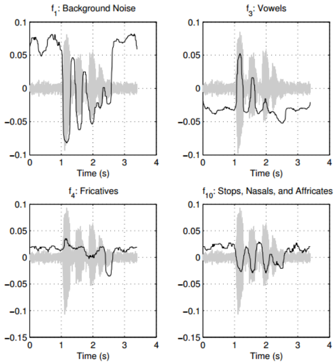


Fig. 2. Four components of the intrinsic spectrogram (shown in black) overlaid on the wave form.

10 random examples of each of 58 phonemes (TIMIT) to build embedding of 50-dim spectrograms (6 neighbours, linear kernel $K(x, y) = x^T y$). Most activity in first 10 components. Example spectrogram is converted to intrinsic representation by applying (*).

Two posters on Interspeech 2015

- *Under-resourced Speech Recognition based on the Speech Manifold*, R. Sahraeian, D. Van Compernolle, F. de Wet
  The main point is that ISA based accoustic models are better when transferred between similar languages (Flemish and Afrikaans example). They use exactly the Laplacian approach (ISA).
- Something on speech denoising — different approach. I must find it (TODO).

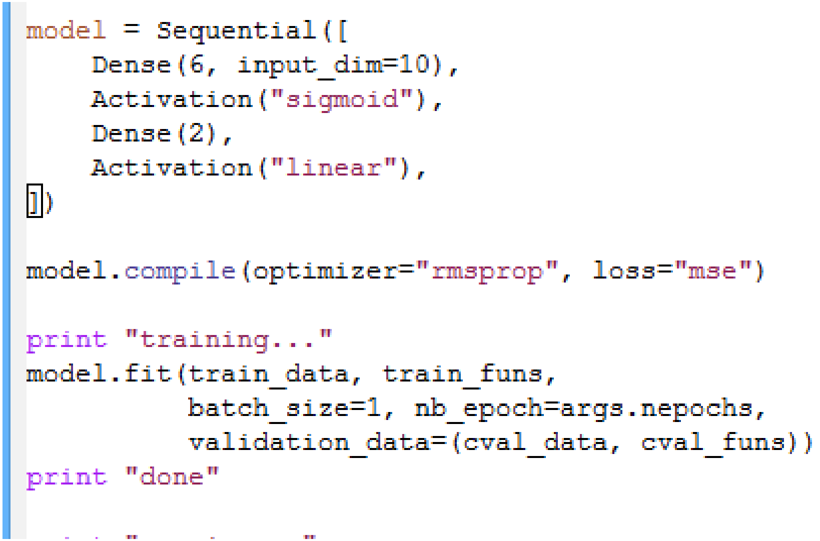
Some research ideas

- How to improve the embedding quality? Optimize weights? Any possibility for developing some general approach?
- Shouldn't graph construction be also kernel based? Why (possibly) different notions of similarity at both stages? I'm not sure I understand how these should interact.
- But this complexity … Even with kernels this should be somehow sparse! Influence of distant training points should be possible to cut-off and regain sparsity.
- More of them …
- But my best idea …

But my best idea

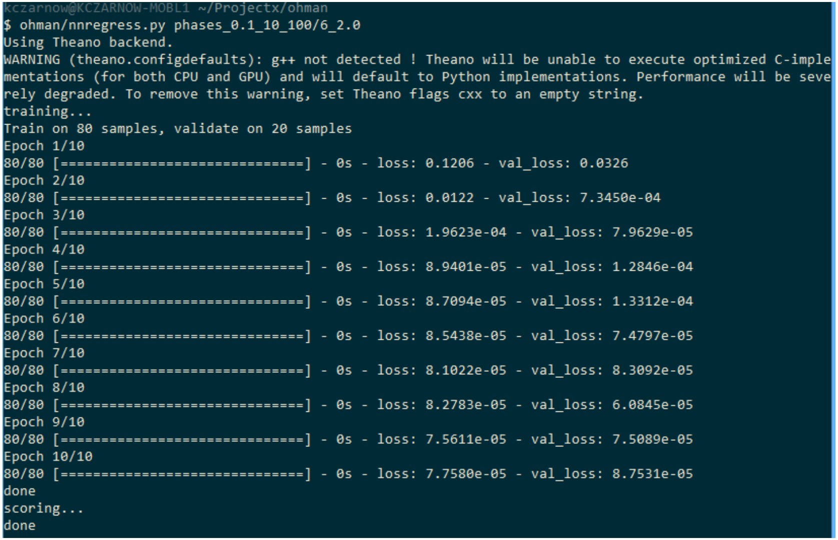
- Forget kernels, use NNs to train the embedding!
- Use graph Laplacian eigenfunctions as regression training objectives. (For the tone-phase data this is work in progress of Ola and me.)
- So this is basically a cooperation proposition . . .
- However, it's hard to believe that the NN idea is very original. Strange I haven't seen it anywhere . . .
- Anyway, even if so, I think there is a lot to explore!
- Can be done with NN's from the very start?
- Anything from the audience?

Training on graph embedding — the network



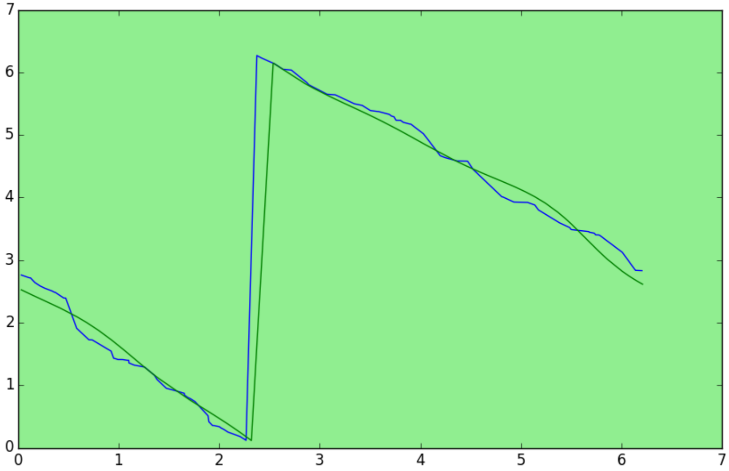
```
model = Sequential([
    Dense(6, input_dim=10),
    Activation("sigmoid"),
    Dense(2),
    Activation("linear"),
])

model.compile(optimizer="rmsprop", loss="mse")

print "training..."
model.fit(train_data, train_funs,
          batch_size=1, nb_epoch=args.nepochs,
          validation_data=(cval_data, cval_funs))
print "done"
```

Training on graph embedding — run it!



```
kczarnow@KCZARNOW-MOBL1 ~/Projectx/ohman
$ ohman/nnregress.py phases_0.1_10_100/6_2.0
Using Theano backend.
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute optimized C-imple
mentations (for both CPU and GPU) and will default to Python implementations. Performance will be seve
rely degraded. To remove this warning, set Theano flags cxx to an empty string.
training...
Train on 80 samples, validate on 20 samples
Epoch 1/10
80/80 [==============================] - 0s - loss: 0.1206 - val_loss: 0.0326
Epoch 2/10
80/80 [==============================] - 0s - loss: 0.0122 - val_loss: 7.3450e-04
Epoch 3/10
80/80 [==============================] - 0s - loss: 1.9623e-04 - val_loss: 7.9629e-05
Epoch 4/10
80/80 [==============================] - 0s - loss: 8.9401e-05 - val_loss: 1.2846e-04
Epoch 5/10
80/80 [==============================] - 0s - loss: 8.7094e-05 - val_loss: 1.3312e-04
Epoch 6/10
80/80 [==============================] - 0s - loss: 8.5438e-05 - val_loss: 7.4797e-05
Epoch 7/10
80/80 [==============================] - 0s - loss: 8.1022e-05 - val_loss: 8.3092e-05
Epoch 8/10
80/80 [==============================] - 0s - loss: 8.2783e-05 - val_loss: 6.0845e-05
Epoch 9/10
80/80 [==============================] - 0s - loss: 7.5611e-05 - val_loss: 7.5089e-05
Epoch 10/10
80/80 [==============================] - 0s - loss: 7.7580e-05 - val_loss: 8.7531e-05
done
scoring...
done
```

Phase estimate on unseen data



0
1
2
3
4
5
6
7
0
1
2
3
4
5
6
7