Connectionist Computing
COMP 30230/41390

Gianluca Pollastri

office: E0.95, Science East.

email: gianluca.pollastri@ucd.ie

Credits

- **Geoffrey Hinton, University of Toronto.**
  - **borrowed some of his slides for "Neural Networks" and "Computation in Neural Networks" courses.**

- **Ronan Reilly, NUI Maynooth.**
  - **slides from his CS4018.**

- **Paolo Frasconi, University of Florence.**
  - **slides from tutorial on Machine Learning for structured domains.**

Lecture notes on Brightspace

- **Strictly confidential...**
- **Slim PDF version will be uploaded later, typically the same day as the lecture.**
- **If there is demand, I can upload onto Brightspace last year's narrated slides.. (should be very similar to this year's material)**

Books

- No book covers large fractions of this course.

- Parts of chapters 4, 6, (7), 13 of Tom Mitchell's "Machine Learning"
- Parts of chapter V of Mackay's "Information Theory, Inference, and Learning Algorithms", available online at:

http://www.inference.phy.cam.ac.uk/mackay/itprnn/book.html

- Chapter 20 of Russell and Norvig's "Artificial Intelligence: A Modern Approach", also available at:

http://aima.cs.berkeley.edu/newchap20.pdf

- More materials later..

Marking

- **3 landmark papers to read, and submit a 10-line summary on Brightspace about: each worth 6-7%**
- **a connectionist model to build and play with on some sets, write a report: 30%**
- **Final Exam in the RDS (50%)**

Error, or cost function

- **We use a squared error to define "fitting the data best":**

$$E = \frac{1}{2} \sum_{examples} \sum_{j} (t_j - y_j)^2$$

Learning algorithm

- **We start from some set of weights (possibly random), then we make small changes trying to minimise the cost:**

$$w_{ji} = w_{ji} + \Delta w_{ji}$$

Gradient descent

- **To figure out how to change the weights so that the error is reduced we can do gradient descent:**

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}}$$

- **This means computing the direction of steepest descent of the error and going there. We saw this already for associators.**

Delta rule

- **It turns out that this rule applies to networks with any number of layers.**
- **For any layer the gradient of the error (and hence the appropriate weight change) can be computed as:**

$$\Delta w_{ji}^{(m)} = -\eta \delta_j^{(m)} x_i^{(m)}$$

Delta rule

- **For the output layer, the deltas are:**

$$\delta_j^{(o)} = (t_j - y_j)$$

- **For any other layer:**

$$\delta_j^{(m)} = \sum_k \delta_k^{(m+1)} w_{kj}^{(m+1)} f'(z_j^{(m)})$$

Backpropagation algorithm

- **We just derived backpropagation.**
- **o is the number of layers, x is a global input, t is a desired output, y[] and z[] contain the outputs and the activations of all the layers in the network.**

```
y[0]=x;
for (i=1..o) {
  z[i] = w[i].y[i-1];
  y[i] = f (z[i]);
}
delta[o]=t-y[o];
for (i=o..1) {
  dw[i]= -η delta[i].y[i-1];
  delta[i-1] = (delta[i].w[i]) f'(z[i-1]);
}
```

Backpropagation as message passing

- **Backpropagation is a schedule for computing weight updates (according to gradient descent) in *layered* networks of neurons of any depth.**
- **Any layer can be seen as an independent processor passing messages forward and backwards.**

Forward, backwards

- **Forward: digest the input through the weights and produce an output.**
- **Backwards: digest deltas from the layer above, generate new deltas and pass them to the layer below.**

- **During backwards weight updates are also computed.**

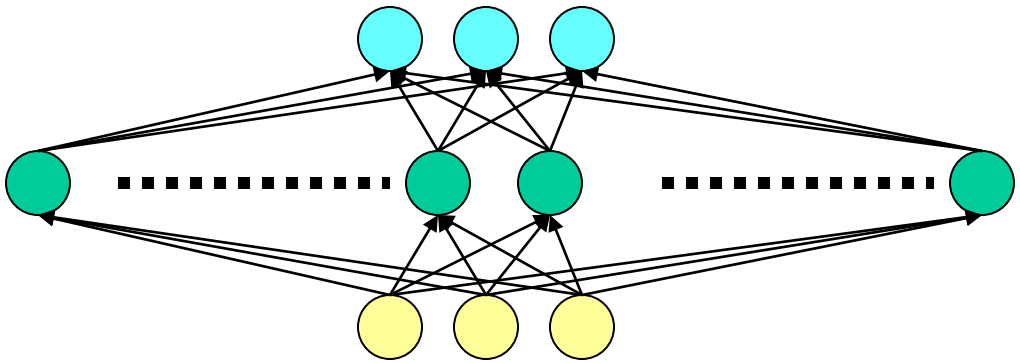
- **A layer doesn’t need to know anything about the network topology to do this. Excellent object.**

Backpropagation

- **Works on any Direct Acyclic Graph of continuous units: no binary-threshold (can't compute f'()).**
- **Loops acceptable only with time delays, we'll see this later.**
- **Very efficient:**
  - **O(|w|)**
  - **Large networks possible (~10^4-10^6 weights reported in many real world applications)**

Expressive power

- **Shortly:**
  - **A single hidden-layer network can approximate every input-output mapping (provided enough units in the hidden layer)**



Expressive power

- **In practice:**
- **If one hidden layer is enough to represent any function, this doesn't mean it is the most efficient configuration.**
- **It is possible that with multiple hidden layers more complex networks can be represented with fewer weights.**

Complexity

- **Loading problem:**
  - **given a network and a set of examples**
  - **Answer yes/no: is there a set of weights so that the network will be consistent with the examples?**
- **The loading problem is NP-complete**
- **In practice networks can be trained in a reasonable amount of time**

Complexity

- **Gradient descent is a *local* optimization approach**
- **Often local minima are not a big issue**
- **There are many techniques to try to avoid them, to speed up gradient descent: more about this in the next few lectures.**

VC dimension of MLPs

- **Given a DAG of M sigmoidal units and n inputs:**

$$\text{VC-dim}(\text{DAG}, \text{M}, \text{n}) \leq 2(n+1)M(1+\log M)$$

- **Hence, the sample complexity bound is**

$$m \geq \frac{1}{\varepsilon}(4\log_2(2/\delta) + 16(n+1)M\log(1+M)\log_2(13/\varepsilon))$$

VC dimension: Single and Multi-Layer Perceptrons

- **SLP performs linear separation. If there are n inputs**
  - **VC(SLP) = n+1.**
- **MLP is more powerful:**
  - **VC(MLP) = 2(n+1) M (1+logM)**

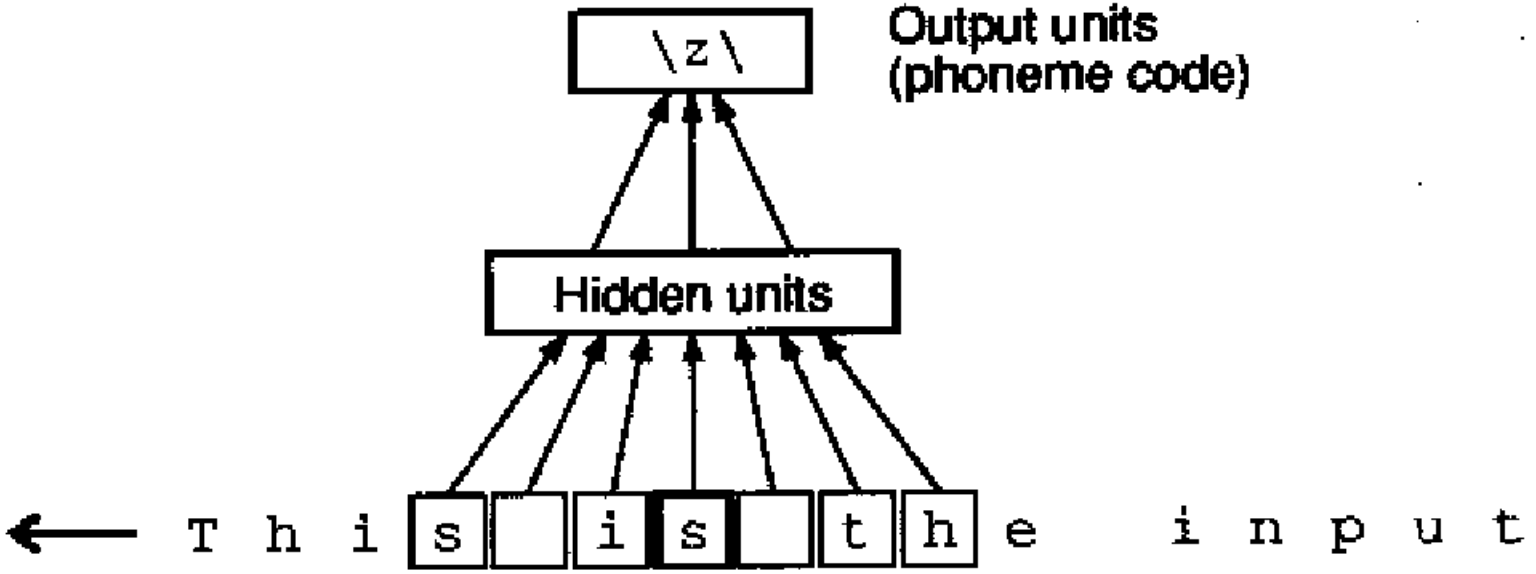
MLP applications: matching words and sounds

- **Sejnowski and Rosenberg, "NETtalk, a parallel network that learns to read aloud", Cognitive Science, 14, 179-211 (1986)**

- **Teaching an MLP how to pronounce English by backprop.**
- **The network was given a stream of words, with the corresponding phonemes.**
- **Once the network had learned, it was possible to make it read.**

The network

- **203x80x26 network**
- **input is sliding sequence of 7 characters**
- **80 hidden units**



Training

- **Trained on a corpus of 1024 words with associated phonemes.**
- **Intelligible speech after 10 epochs, 95% correct phonemes on *training* after 50 epochs.**
- **78% correct classification on test set (“overfits”)**
- **Damaging network produced “graceful degradation”.**

MLP applications: protein secondary structure prediction

- **Proteins are strings:**

  `FEFHGYARSGVIMNDSGASTKS`

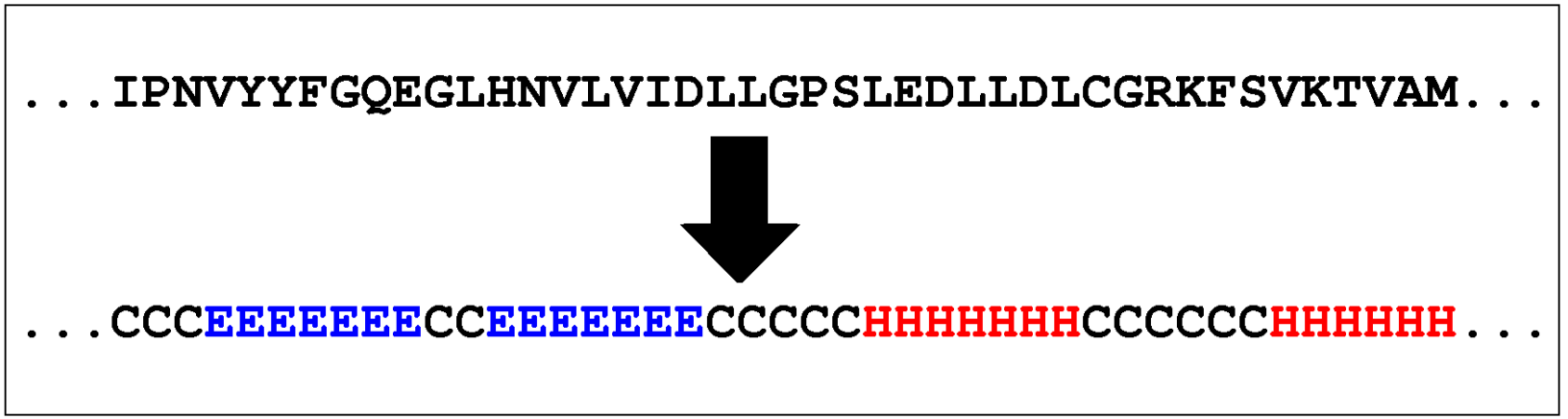
  `GAYITPAGETGGAIGRLGNQAD`

  `TYVEMNLEHKQTLDNG`

- **Structures too:**

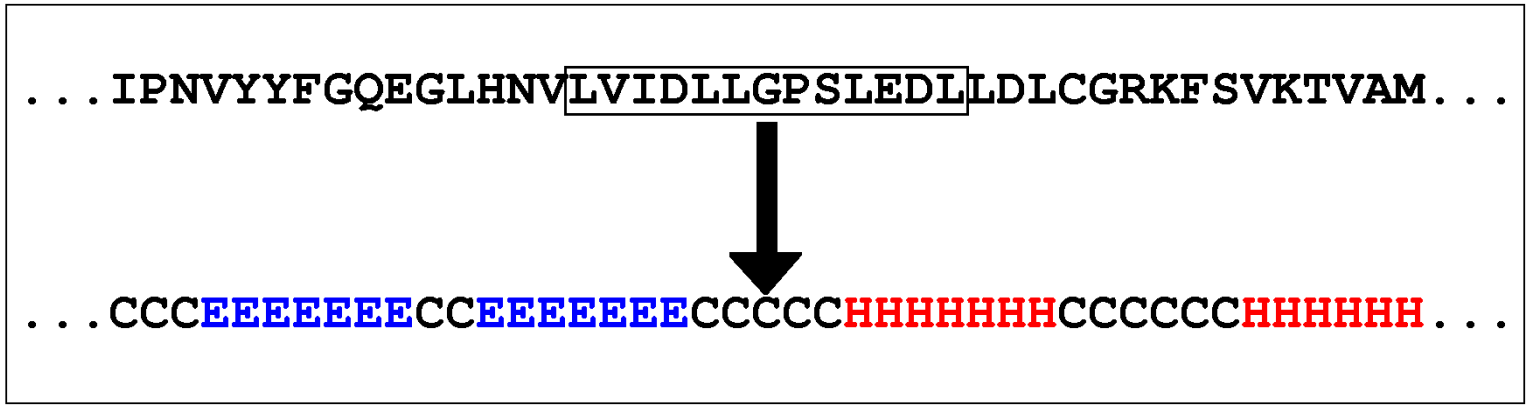
Secondary structure (SS) prediction

- **Label each amino acid as helix/strand/coil**
- **Predict this label from the amino acid sequence**



by NETtalk

- Qian and Sejnowski 1988.
- They used NETtalk to predict it
- Sliding window, stacked networks.



results

- **64% correct prediction**
- **not great but better than the other methods**
- **neural networks are nowadays the prime method for SS prediction**