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..

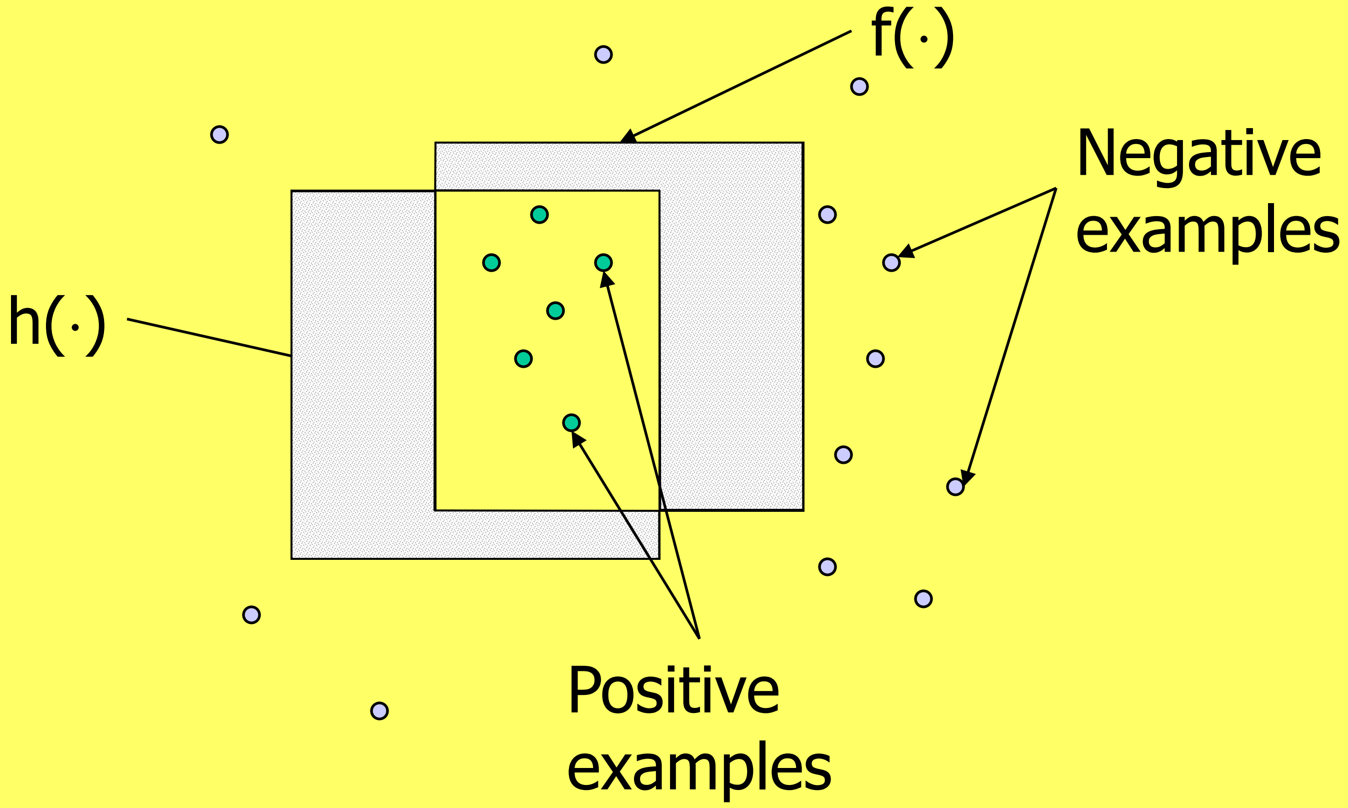
Marking

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

Learning as refinement

- **Start with a small hypothesis class (e.g., boolean conjunctions)**
  - **this means we need to *know a priori* something about the solution**
- **Use examples to infer the particular function (e.g. conjunction) in the class.**

Generalisation



$$\text{true error} = \int_X f(x) \oplus h(x) P_D(x)\,dx$$

Generalisation

- Are there boundaries on generalisation error?
- How expressive is a model, i.e. once we know how to learn:
  – what can we learn?
  – how many examples do we need to learn?
- For instance: how complex a task can we learn with N neurons / how many examples do we need to train them?

PAC learning

- Probably Approximately Correct (hopefully)
- H finite, unknown data distribution D, consistent learner
- Then we need at least

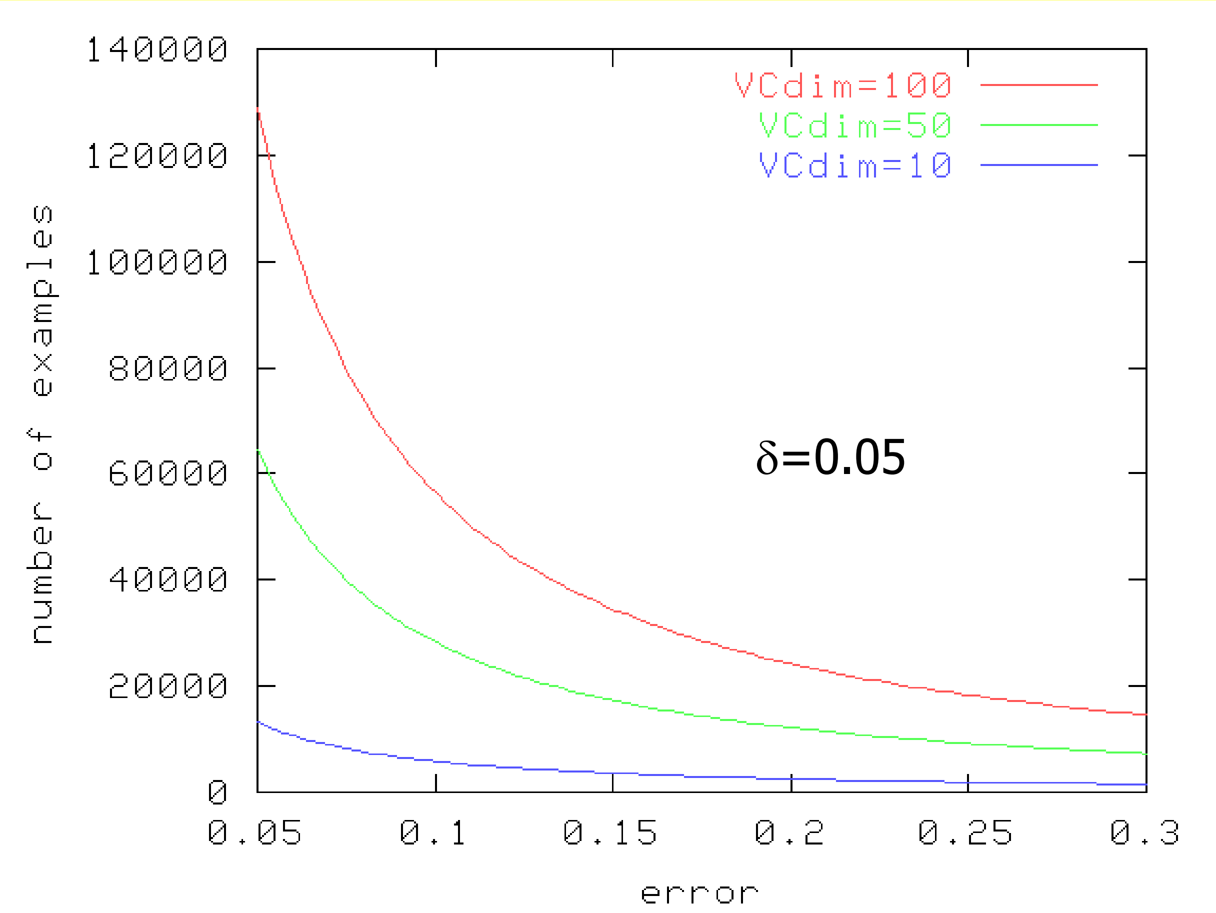
$$m \geq \frac{1}{\varepsilon}(\ln|H| + \ln(1/\delta))$$

training examples to guarantee that the true error will be < ε with probability > $(1-\delta)$

m is called *sample complexity*

PAC learning and VC dim

$$m \geq \frac{1}{\varepsilon}(4\log_2(2/\delta) + 8VC(H)\log_2(13/\varepsilon))$$



PAC learning for a perceptron

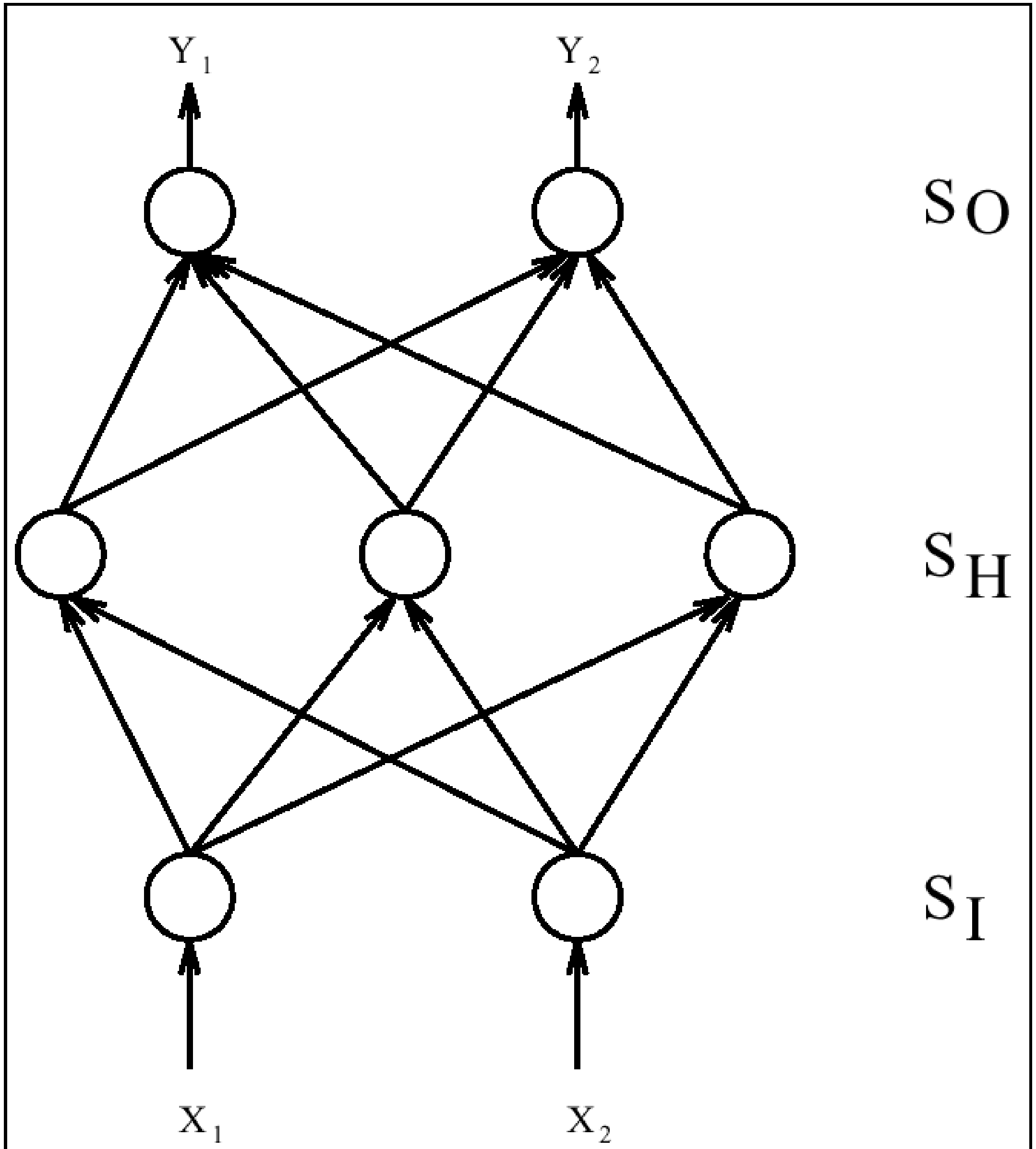
- n inputs -> VC(H)=n+1

- Example:
- 100 inputs, δ=1%, ε=1%
- $m \geq 100\ (4 \log 200 + 8\ 101 \log 1300) = 8388.8$

Multi-layer perceptrons

- So far we saw perceptrons where inputs and outputs are directly connected.
- In the case of Hopfield nets/Boltzmann machines we saw that it's possible to have hidden units, i.e. neurons that aren't directly connected to the inputs or the outputs.
- What about perceptrons with hidden units?

Three layer perceptron



S_O **Output layer**

S_H ***Hidden layer***

S_I **Input layer**

Multi-layer perceptrons

- **As usual, the outputs y are obtained as a linear combination of the inputs, and then passed through a *squashing function* (unspecified, for the moment):**

$$y_j^{(o)} = f(z_j^{(o)})$$

y: output
z: activation
x: inputs

$$z_j^{(o)} = \sum_i w_{ji}^{(o)} x_i^{(o)} + b_j^{(o)}$$

Multi-layer perceptrons

- **In this case though the input is the output of a lower *layer* of perceptrons:**

$$x_i^{(o)} = y_i^{(o-1)}$$

- **We will use the superscript to denote the layer:**

$$x_i^{(k)}, z_i^{(k)}, y_i^{(k)}, w_{ji}^{(k)}$$

Learning algorithm

- **Let's assume that we have a set of examples $(x_1,t_1), (x_2,t_2), .. , (x_n,t_n)$ (t for *target*, to distinguish from the output of the network).**
- **As usual we want to train the model on these examples, i.e. find the weights that fit the data best.**

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.

Multi-layer

- There are multiple layers here, so the problem is a bit trickier than for associators.
- The computations for the output layer of the network are really similar to those for associators though.

part 1

$$\frac{\partial E}{\partial w_{ji}^{(o)}} = \frac{\partial}{\partial w_{ji}^{(o)}}\Big(\frac{1}{2}\sum_{examples}\sum_{k}(t_k - y_k)^2\Big) =$$

$$\frac{1}{2}\sum_{examples}\sum_{k}\frac{\partial}{\partial w_{ji}^{(o)}}(t_k - y_k)^2 =$$

$$\sum_{examples}(t_j - y_j)\frac{\partial y_j}{\partial w_{ji}^{(o)}}$$

part 2

$$\frac{\partial y_j}{\partial w_{ji}^{(o)}} = \frac{\partial}{\partial w_{ji}^{(o)}} \left\{ f\left(\sum_l w_{jl}^{(o)} x_l^{(o)} + b_j^{(o)} \right) \right\} =$$

$$f'(z_j^{(o)}) \sum_l \frac{\partial}{\partial w_{ji}^{(o)}} \left(w_{jl}^{(o)} x_l^{(o)} + b_j^{(o)} \right) = f'(z_j^{(o)}) x_i^{(o)}$$

overall, for the output layer

- **Not too bad, in fact still fairly similar to Hebb's law.**

$$\frac{\partial E}{\partial w_{ji}^{(o)}} = \sum_{examples} (t_j - y_j) f'(z_j^{(o)}) x_i^{(o)}$$

- **Which I can write as:**

$$\sum_{examples} \delta_j^{(o)} x_i^{(o)}$$

f'()

- **Depends on f'().**
- **For instance, if f() is linear, f'()=1, so:**

$$\frac{\partial E}{\partial w_{ji}^{(o)}} = \sum_{examples} (t_j - y_j) x_i^{(o)}$$

- **(same law as for the associator)**

hidden layer

- **We can compute the derivative of the error w.r.t. the weights in the hidden layer now. The first part is this:**

$$\frac{\partial E}{\partial w_{ji}^{(o-1)}} = \frac{\partial}{\partial w_{ji}^{(o-1)}}\Big(\frac{1}{2}\sum_{examples}\sum_{k}(t_k - y_k)^2\Big) =$$

$$\frac{1}{2}\sum_{examples}\sum_{k}\frac{\partial}{\partial w_{ji}^{(o-1)}}(t_k - y_k)^2 =$$

$$\sum_{examples}\sum_{k}(t_k - y_k)\frac{\partial y_k}{\partial w_{ji}^{(o-1)}}$$

Hidden layer part 2

$$\frac{\partial y_k}{\partial w_{ji}^{(o-1)}} = \frac{\partial}{\partial w_{ji}^{(o-1)}} \left\{ f\left(\sum_l w_{kl}^{(o)} y_l^{(o-1)} + b_k^{(o)} \right) \right\} =$$

$$f'(z_k^{(o)}) w_{kj}^{(o)} \frac{\partial y_j^{(o-1)}}{\partial w_{ji}^{(o-1)}} =$$

$$f'(z_k^{(o)}) w_{kj}^{(o)} f'(z_j^{(o-1)}) x_i^{(o-1)}$$

Hidden layer, overall

$$\frac{\partial E}{\partial w_{ji}^{(o-1)}} = \sum_{examples} \sum_{k} (t_k - y_k) f'(z_k^{(o)}) w_{kj}^{(o)} f'(z_j^{(o-1)}) x_i^{(o-1)} =$$

$$\sum_{examples} \sum_{k} \delta_k^{(o)} w_{kj}^{(o)} f'(z_j^{(o-1)}) x_i^{(o-1)} =$$

$$\sum_{examples} \delta_j^{(o-1)} x_i^{(o-1)}$$

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]);
•  }
```