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>)

Programming assignment

- Implement a Multi-Layer Perceptron, test it.

- The description on Brightspace.

- Submit through Brightspace code and test results by Dec the 5th at 23:59, any time zone of your choice (Baker Island?).
- 30% of the overall mark
- One third of a grade down every day late, that is: if you deserve an A and you're 1 day late you get an A-, 2 days late a B+, etc.

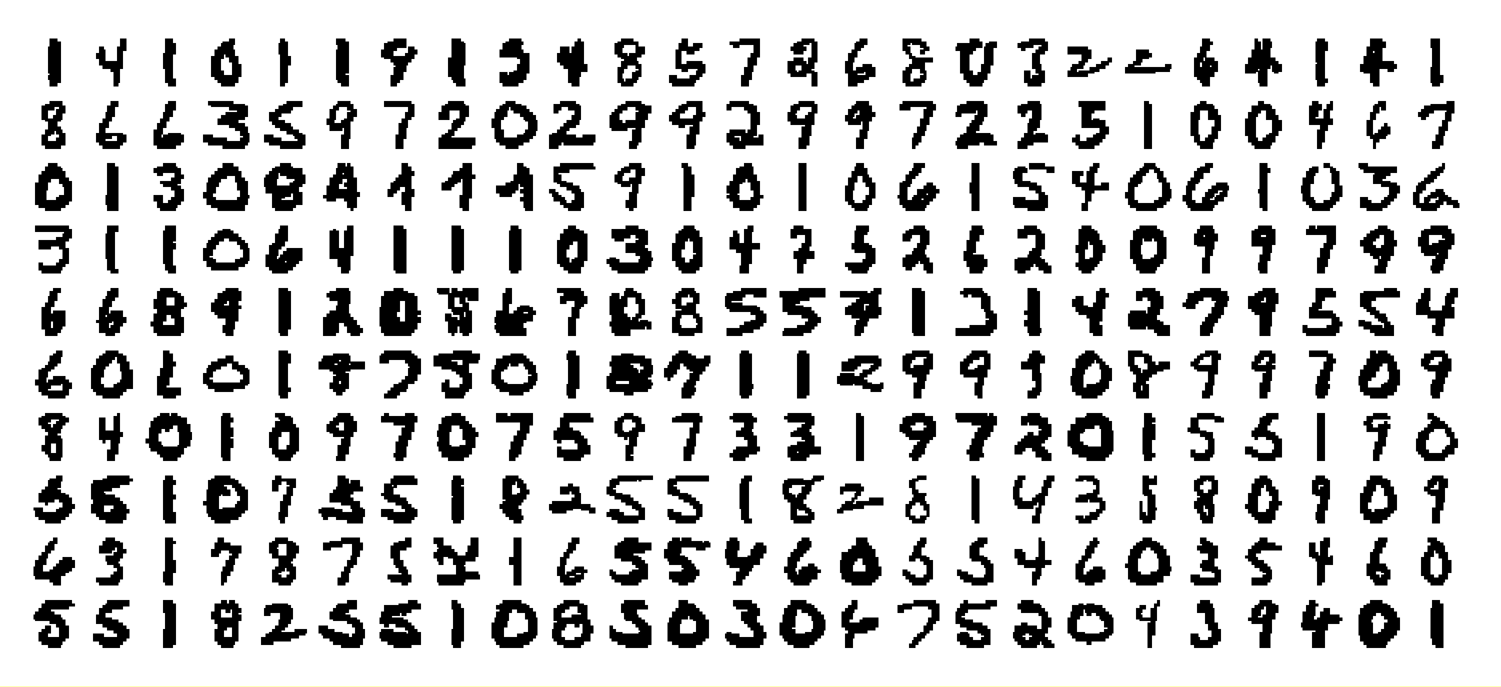
MLP applications: handwritten digit recognition

- “Hand-written digit recognition with a back-propagation network”, Le Cun et al. 1990

- Multi-layer perceptron applied to handwritten digits.
- Relatively little non-connectionist preprocessing: digits split, centred, normalised.

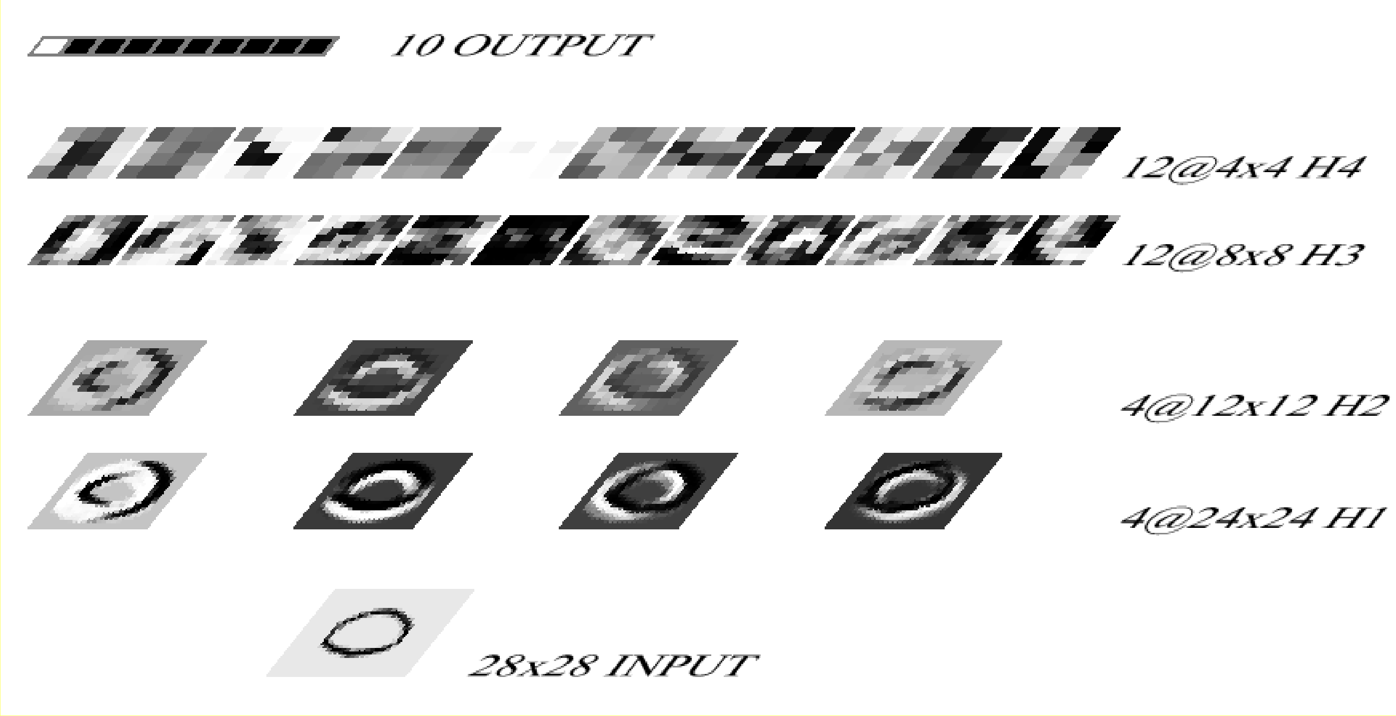
preprocessing

- **Enough to obtain digits in this format, each one is a 16x16 greyscale image:**



deep network

- **4 hidden layers:**



Results

- After 30 epochs the error on the training set was 1.1% and the squared error 0.017.
- On the test set: 3.4% and 0.024

- To get 1% error: 5.7% rejection (9% on just handwritten)
- A lot of these were actually caused by preprocessing. Some of those that weren’t, were ambiguous even to humans.

Invariances

- In Le Cun's paper we saw translation invariance was introduced into the network by weight sharing.
- Teaching neural networks invariances is a general problem.

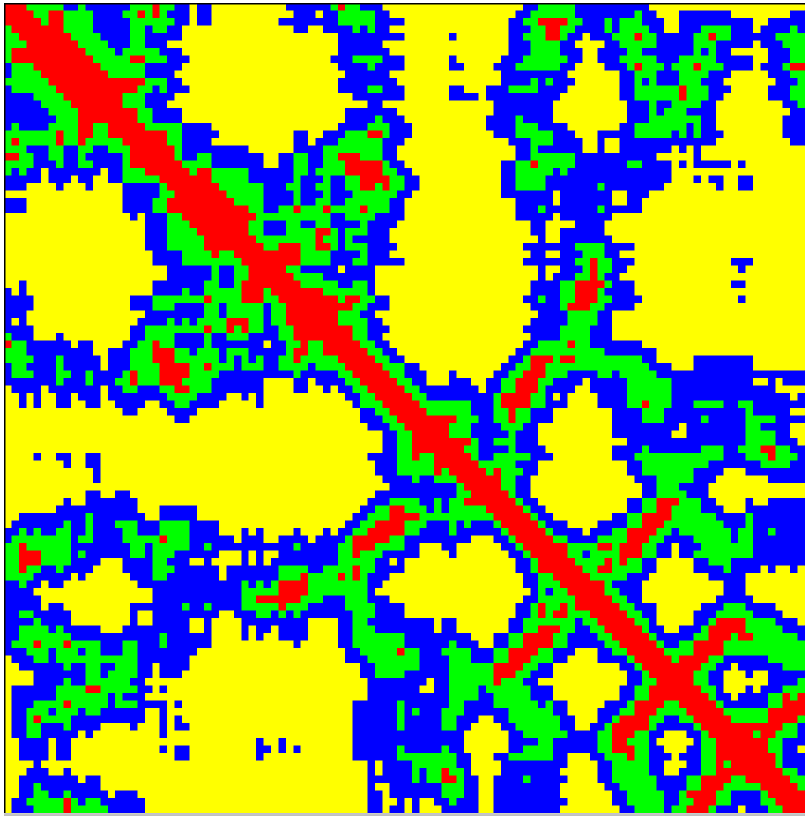
The invariance problem

- **Our perceptual systems are very good at dealing with invariances**
  - **translation, rotation, scaling**
  - **deformation, contrast, lighting, rate**
- **We are so good at this that it’s hard to appreciate how difficult it is.**
  - **It’s one of the main difficulties in making computers perceive.**
  - **We still don’t have generally accepted solutions.**

Invariances: using features

- Instead of representing an object directly, extract whatever-invariant features first.
- For instance, if we want roto-translational invariance, represent an object by the distances between parts instead of xyz coordinates.

example



What features?

- Some features do not affect the information content of an instance (e.g. distance map vs. xyz coordinates).

- Some features, while informative, might involve some information loss.

Cost of features

- All features have to be designed.
- This process may range from completely trivial to years-long and involving the consultation of experts and significant costs.

Invariances: normalisation

- For instance put a box around an object, then scale it to a fixed size: same as preprocessing digits in Le Cun et al.
- Eliminates degrees of freedom.
- Not always trivial how to choose the box.

Invariances: brute force

- We can tackle invariances by:
  - constraining network weights
  - using features
  - normalising
- But computer scientists should be lazy and impatient. Wouldn’t it be great if we could let the network do all the job?
- Brute force: to create invariance to trasformation X, for each example generate a lot of new other examples by applying X to it. Then train a large network on a fast computer.

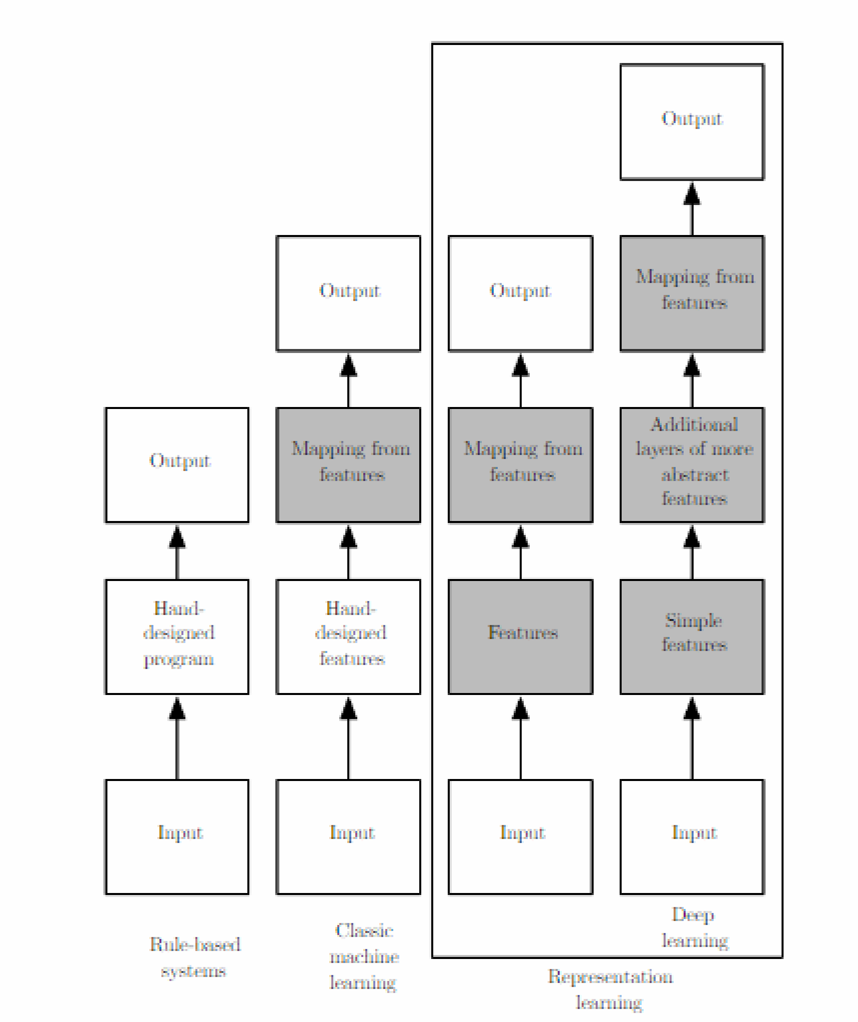
Invariances: brute force

- For example, translate and rotate a digit in a lot of different ways and train a large network to recognise it.
- It generally works well, if the transformations aren't too large: do approximate (easy) normalisation first.

Brute force and Deep Learning

- Incidentally, using brute force often also involves adding layers that can tackle a less processed input.
- This is in many ways one of the defining elements of Deep Learning, though it's been done plenty of times before Deep Learning was named.

A matter of depth



Summary: invariances

- Often as tough a problem as learning after invariances are tackled.
- Possible solutions:
  - network design
  - features
  - normalisation
  - brute force

Problems with squared error

- So far, for gradient descent, we used:
  - Error: squared
  - Output function: linear or sigmoid (binary won’t work)
- There are tricky problems with squared error. For instance if the desired output is 1 and the actual output is very close to 0 there is almost no gradient.

Problems with squared error

- **These are the deltas:**

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

- **And these are f and f':**

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

$$f'(x) = \frac{1}{2 + e^{-x} + e^{x}}$$

Alternatives: softmax and relative entropy

$$f(z_i) = softmax(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

$$E = -\sum t_j log y_j$$

- **Non-local non linearity.**
- **Outputs add up to 1 (can be interpreted as the probability of the output given the input).**

Gradient descent with softmax

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}}\Big(-\sum_k t_k log y_k\Big) =$$

$$-\sum_k \frac{t_k}{y_k}\frac{\partial y_k}{\partial w_{ji}} =$$

$$-\sum_k \frac{t_k}{y_k}\frac{\partial}{\partial w_{ji}}\frac{e^{z_k}}{\sum_v e^{z_v}} =$$

$$-\sum_k \frac{t_k}{y_k}\frac{e^{z_k}\frac{\partial z_k}{\partial w_{ji}}\sum_v e^{z_v} - e^{z_k}\sum_v e^{z_v}\frac{\partial z_v}{\partial w_{ji}}}{\left(\sum_v e^{z_v}\right)^2} =?$$

derivative of the activation

- **If I define:**

$$d(x) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{if } x \neq 0 \end{cases}$$

- **Then:**

$$\frac{\partial z_v}{\partial w_{ji}} = d(v - j)x_i$$

$$-\sum_k \frac{t_k}{y_k} \frac{e^{z_k} \frac{\partial z_k}{\partial w_{ji}} \sum_v e^{z_v} - e^{z_k} \sum_v e^{z_v} \frac{\partial z_v}{\partial w_{ji}}}{\left(\sum_v e^{z_v}\right)^2} =$$

$$-\sum_k \frac{t_k}{y_k} \frac{e^{z_k} d(k-j) x_i}{\sum_v e^{z_v}} + \sum_k \frac{t_k}{y_k} \frac{e^{z_k} \sum_v e^{z_v} d(v-j) x_i}{\left(\sum_v e^{z_v}\right)^2} =$$

$$-\sum_k \frac{t_k}{y_k} y_k d(k-j) x_i + \sum_k \frac{t_k}{y_k} \frac{e^{z_k} e^{z_j} x_i}{\left(\sum_v e^{z_v}\right)^2} =$$

$$-t_j x_i + \sum_k \frac{t_k}{y_k} y_k y_j x_i =$$

$$-t_j x_i + y_j x_i \sum_k t_k =$$

$$(y_j - t_j) x_i$$

Gradient descent with softmax

- No f'() : the steepness of the cost function balances the flatness of the output non-linearity.

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

More squashing functions

- **ReLU**
- **Smooth ReLU**

- **Also: Leaky ReLU, parametric ReLU, ..**