Artificial Neural Networks

Topics in Cognitive Modelling
Jan. 21, 2014

John Lee, Chris Lucas
School of Informatics
University of Edinburgh
{jlee,clucas2}@inf.ed.ac.uk

Connectionism and ANNs

- Last time, we discussed connectionist philosophy:
  - “Biologically inspired” empiricism.
  - The brain has powerful general-purpose learning and processing capabilities, due to its architecture: distributed parallel processing and distributed representations.
  - These can be modelled using artificial neural networks.
- Today, some more technical details about ANNs, and a critique.

Artificial neural networks

- ANNs reproduce what Elman et al. (in *Rethinking Innateness*) believe to be the critical aspects of neural structure:
  - Distributed computation using small computational elements.
  - Each element accesses only local information.
  - Information is represented in a distributed way.
  - Responses are nonlinear.

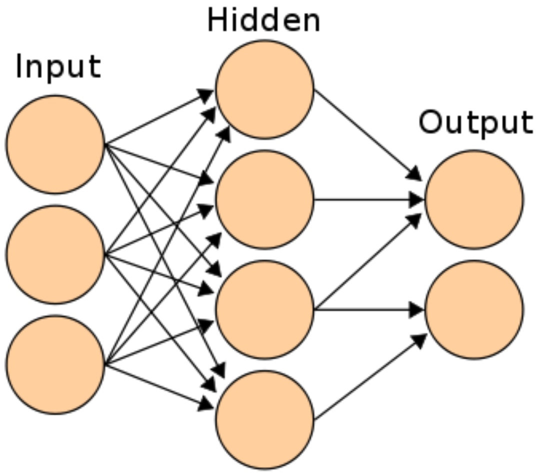
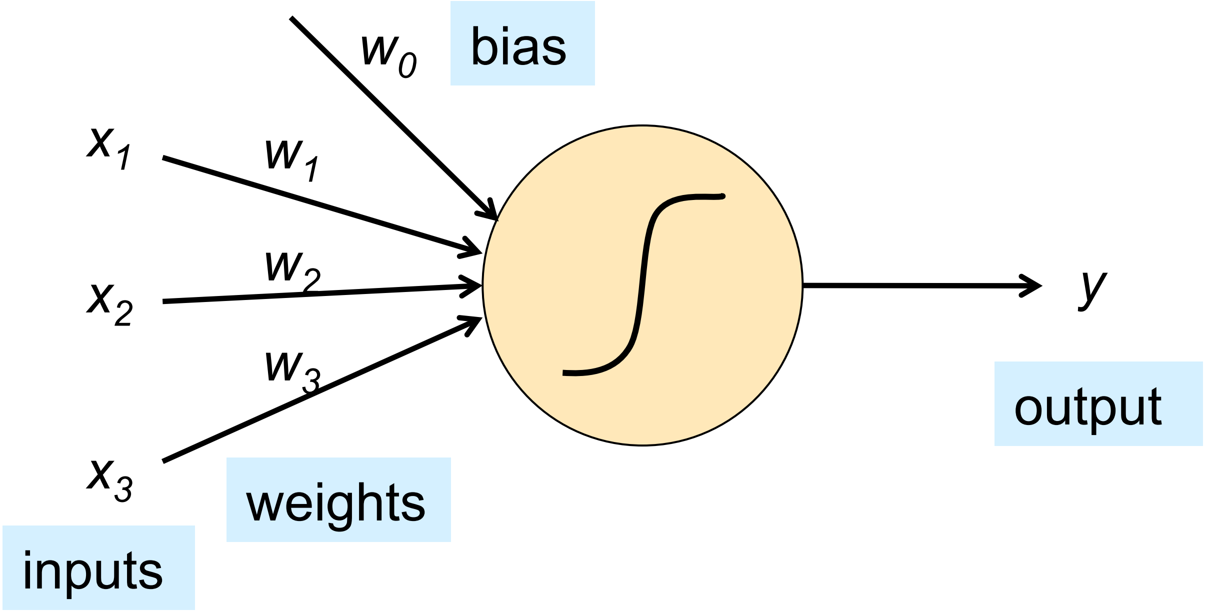


Figure: http://en.wikipedia.org/wiki/Artificial_neural_network

The perceptron

- Simple model of a neuron, building block of ANNs.



bias

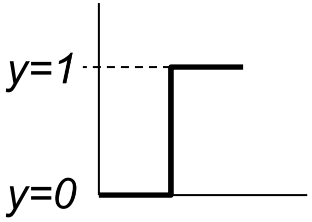
output

weights

inputs

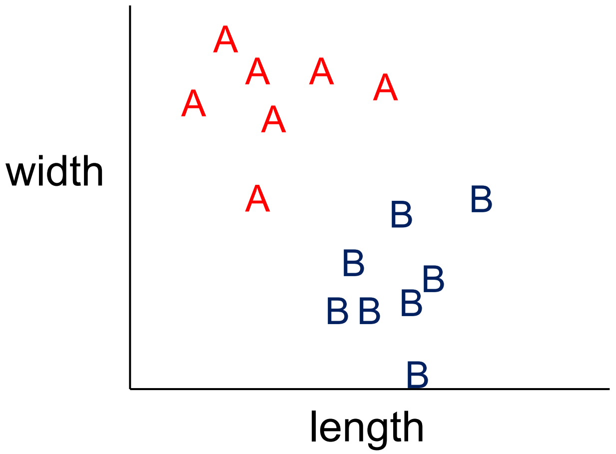
$$y = F\left(w_0 + \sum_{i=1}^{n} w_i x_i\right)$$

Activation function F is usually sigmoid, a smooth approximation to a step function:



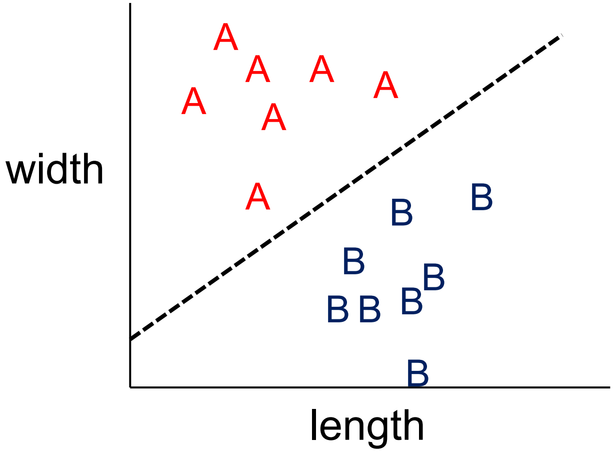
Example computation

- Does an object belong to category A or B?
  - Inputs: features of the objects (e.g., length, width)
  - Output: A or B



Example computation

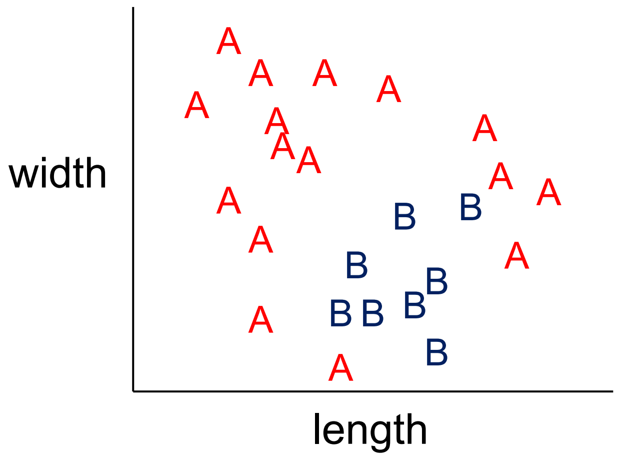
- Does an object belong to category A or B?
  - Inputs: features of the objects (e.g., length, width)
  - Output: A or B



- Perceptron can be trained to classify.
  - Given example inputs with labels, adjust weights to learn a decision boundary (more on training later).

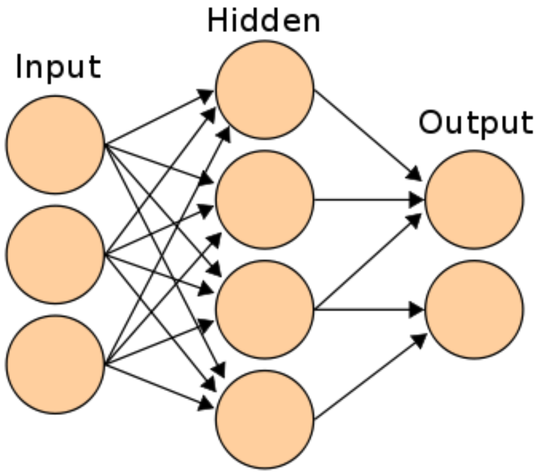
Limitations of the perceptron

- A single perceptron can only learn linear decision boundaries.
- What if we want to learn a more complex function?



Multilayer perceptron

- MLP networks have one or more hidden layers, which allows them to learn more complex (non-linear) functions.



- Probably the most commonly used feedforward network.
- Usually each layer is fully connected to the next layer.
- Types of functions that can be computed depends on number of nodes, layers, and connections (and the activation function).

Representation

- Task: learn whether each animal is dangerous or not
- Localized input representation:
  - One node active per input.

Dog: `[1 0 0 0 ... 0]`
Cat: `[0 1 0 0 ... 0]`
Lion: `[0 0 1 0 ... 0]`
`...`
Rhino: `[0 0 0 0 ... 1]`

- Distributed input representation:
  - Several nodes active per input.
  - Often feature-based.

Dog: `[0 1 1 1 ... 0]`
Cat: `[0 1 1 1 ... 1]`
Lion: `[1 1 1 0 ... 1]`
`...`
Rhino: `[1 0 0 0 ... 0]`

- Cf. http://cognet.mit.edu/library/erefs/mitecs/van_gelder1.html

Training

- MLPs are trained using backpropagation.
  - Minimizes the training error $J = \frac{1}{2} ||\mathbf{d}-\mathbf{y}||^2$ between desired output **d** and actual output **y** (assumes correct outputs are known).
  - Uses gradient descent (requires differentiating the activation function – that's why sigmoid is preferred over step function).

Backpropagation sketch

- Repeat for each training example until convergence:
  - Compute the output **y** for this input.
  - Determine for each w_{ij} (weight from unit *i* to *j*) in output layer how changing w_{ij} affects training error *J*, i.e. compute

$$\frac{\partial J}{\partial w_{ij}}$$

  - Perform a similar computation for the weights in the previous layer(s) to propagate the error signal backward.
  - Update all weights (η is the learning rate parameter):

$$w_{ij} = w_{ij} + \eta \frac{\partial J}{\partial w_{ij}}$$

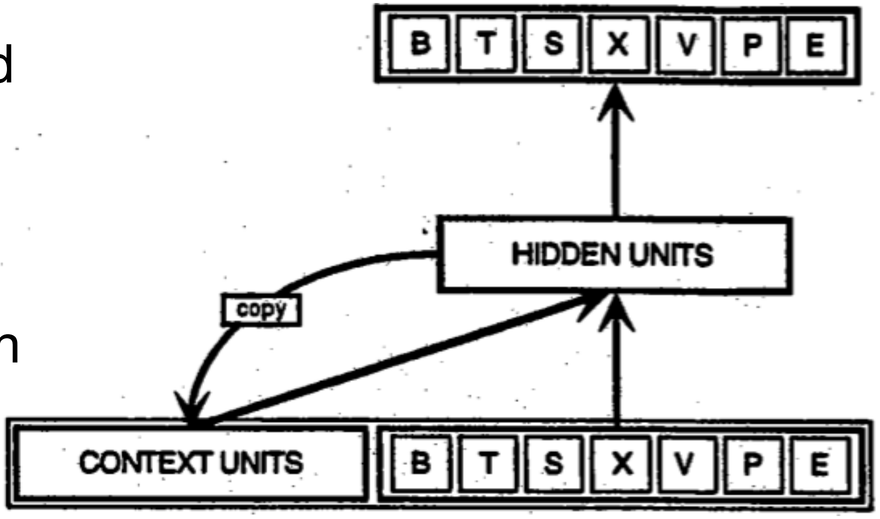
For more details, see Elman et al. (1996) Ch. 2 or your favourite machine learning textbook.

Supervised vs. unsupervised

- Backpropagation requires training data with known (input, output) pairs.
- Normally, this means supervised training.
  - Ex. Categorization problem, with category labels provided.
- Sometimes desired outputs may be available in the world, without supervision.
  - Ex. Given view of object, predict alternate view. (Requires supervised data for computer, but not for human with real objects.)
- Other types of networks usually used for unsupervised learning problems.

Simple recurrent networks (Elman nets)

- Goal: predict the next input in a sequence (e.g., characters in a string of English text).
- Prediction is based on current input, plus additional input provided by context units.
  - Hidden unit values are copied into context units after each timestep.
  - Trained using modified back-propagation (backpropagation through time).



- Cf. http://www.stanford.edu/group/pdplab/pdphandbook/handbookch8.html

Figure: Servan-Schreiber et al. (1991).

Self-organizing maps (Kohonen nets)

- Goal: Nearby output units respond to similar inputs.
  - Project inputs into lower dimensional space.
  - Unsupervised clustering method.
- Uses 2D layer of output units with competitive learning.
  - Input is compared to weight vector of each unit.
  - Most similar unit “wins”; weights are updated to be even closer to input, and nearby units are similarly updated.
  - Hence nearby locations in the map represent inputs with similar properties.

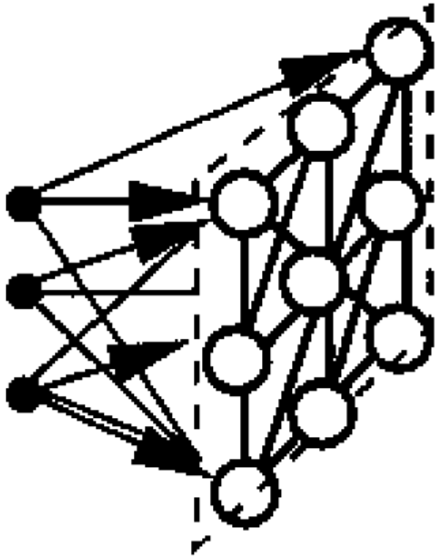


Figure: Jain et al. (1996).

Summing Up

- ANNs are made of collections of perceptrons. Important properties include:
    - Activation function: threshold, sigmoid, Gaussian, etc.
    - Connections: feedforward vs. feedback (recurrent), local and global topology.
    - Representation of input and output.
    - Training algorithm: backpropagation, competitive learning, etc.
- Different kinds of functions can be learned depending on the choices made for these properties.

ANNs in cognitive modelling

- Examples of what ANN models may be used for:
  - Showing that rule-like behaviour (e.g., in language) is possible without explicit mental representation of rules.
  - Showing that certain representations or features of the data are useful for learning (e.g., by comparing success of networks using different kinds of input).
  - Showing that X is learnable in principle (by exhibiting a network that learns X).

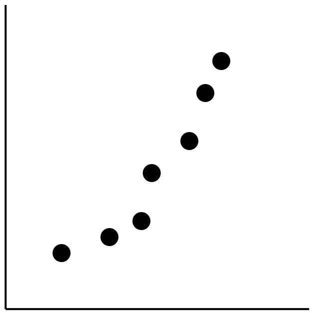
Hypothetical question

- If we are trying to prove that X is learnable, why not just use the most powerful possible ANN?
  - An ANN with a gazillion nodes and two jillion hidden layers should be able to learn anything, right?*

*Technically, one hidden layer is enough for an MLP to learn any function, if we are allowed to use arbitrary activation functions and enough nodes.

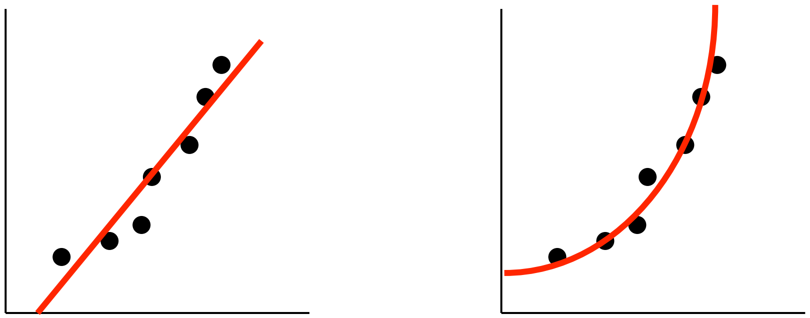
Function learning

- Suppose we are trying to predict some response *y* given an input *x*.
  - If I push with *x* force, how far (*y*) does an object move?
  - If I add *x* grams of salt, how good (*y*) does my food taste?
- Observe some (*x*,*y*) pairs, want to learn a function to correctly predict new (*x*,*y*) pairs (i.e., regression).



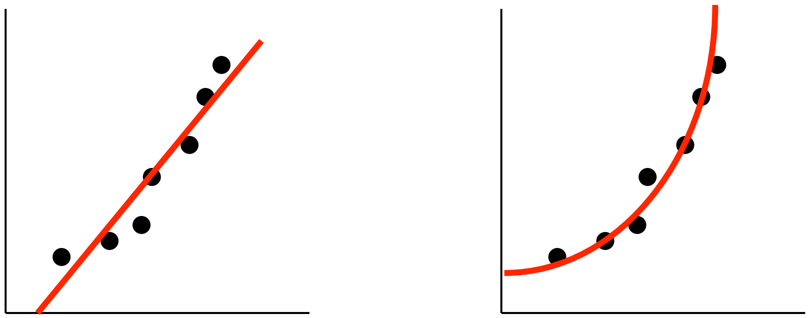
Function learning:

- But which function is right?

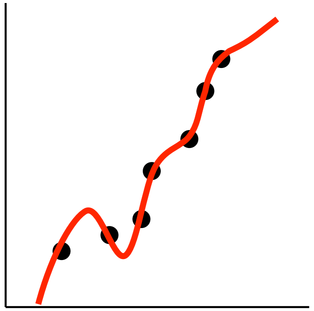


Function learning:

- But which function is right?



- What about this one?



The bias-variance tradeoff

- Allowing the learner to posit complex functions (e.g., 7-degree poly) means estimates have more variance.
    - Small perturbations in the data will cause large changes in estimated function.
    - The learner can overfit the data, causing poor generalization.
    - More data is required to accurately estimate the function.

The bias-variance tradeoff

- Limiting possibilities to simpler function classes (e.g. linear) reduces variance, but increases bias.
  - If the true function is in the allowed simpler class, then overfitting is avoided and generalization improves.
  - But some functions cannot be learned, because not in the simpler class.
  - If the true function is not in this allowed class, the fit will be bad and probably so will generalization.
  - So there's a limit to how far we can reduce both bias and variance together.

No Free Lunch theorem (Wolpert, 1996)

- No learning algorithm is inherently “better” for all data.
  - An algorithm whose bias matches the distribution of the data will learn faster and more accurately than other algorithms.
  - But this algorithm will not necessarily be good at learning from other kinds of data.

Hypothetical question

- If we are trying to prove that X is learnable, why not just use the most powerful possible ANN?
  - An ANN with a gazillion nodes and two jillion hidden layers should be able to learn anything, right?
  - *Well, yes, and that's the problem.* Since it can learn anything, it will overfit the data it sees, and not generalize well. It will also require a lot more data to get close to the right solution, perhaps more than humans are exposed to.

Hypothetical question

- If we are trying to prove that X is learnable, why not just use the most powerful possible ANN?
  - An ANN with a gazillion nodes and two jillion hidden layers should be able to learn anything, right?
  - *Well, yes, and that's the problem.* Since it can learn anything, it will overfit the data it sees, and not generalize well. It will also require a lot more data to get close to the right solution, perhaps more than humans are exposed to.
- So we are back to the fact that ANNs do and should impose constraints on learning.
  - (Some form of innateness after all …)
  - But what exactly are these constraints?

Implicit vs. explicit constraints

- The constraints imposed by ANNs are implicit.
    - Different architectures can learn different kinds of things.
    - In many cases it's hard to make really clear the relationship between the architecture and what can be learned.
- If we want to study human learning biases, maybe we should be explicit about modelling them.
    - This is (part of) the philosophy of the Bayesian approach to cognitive modelling... Stay tuned.

References

Elman, J., Bates, E., Johnson, M., Karmiloff-Smith, A., Parisi, D., and Plunkett, K. 1996. *Rethinking innateness: a connectionist perspective on development*. Cambridge, MA: MIT Press.

Jain, A.K., Mao, J. and Mohiuddin, K.M. 1996. Artificial neural networks: a tutorial. *Computer* 29(3), 31-44.

Servan-Schreiber, D., Cleeremans, A., and McClelland, J. L. 1991. Graded state machines: The representation of temporal contingencies in simple recurrent networks. *Machine Learning*, 7:161–193.

Thomas, M. S. C. and McClelland, J. L. 2008. Connectionist Models of Cognition, pp. 23-30. In *Handbook of Computational Psychology*, Ron Sun, (ed.) Cambridge: Cambridge University Press.

Wolpert, D. H., 1996. The lack of *a priori* distinctions between learning algorithms. *Neural Computation*, 8(7), 1341–1390.