# The Perceptron

Nigel Goddard
School of Informatics

# A Simple Linear Algorithm

- Can we do something simpler than logistic regression? And still be linear?
- For logistic regression we had this squashing function

$$f(z) = \sigma(z) \equiv 1/(1 + \exp(-z))$$

- What if we just have a step function?

$$f(z) = \text{sign}[z] = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

- Notice that we call the classes $y \in \{-1, 1\}$. This is just for convenience later on.
- This architecture is called a *perceptron*, and has a very long history.

# Classifying Using a Perceptron

- Like any other linear classifier. Given $\tilde{\mathbf{w}}$, $w_0$ and a $\mathbf{x}$ to classify, do

$$\hat{y} = \begin{cases} 1 & \text{if } \tilde{\mathbf{w}}^T\mathbf{x} + w_0 \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

- This is OK, but how are you going to train it?
- The problem is that you can't use gradient descent anymore.

# The Perceptron Learning Rule

- The following rule was studied by Rosenblatt (1956)

  **repeat**
  **for** $i$ in $1, 2, \ldots n$
  $\hat{y} \leftarrow \text{sign}[\mathbf{w}^T \mathbf{x}_i]$
  **if** $\hat{y} \neq y_i$
  $\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i$
  **until** all training examples correctly classified

- Why does this make sense? Use same reasoning as logistic regression gradient.
- Say $y_i = 1$ and $\hat{y} = 0$. Then, after the update $\mathbf{w}^T \mathbf{x}_i$ gets bigger.

# The Perceptron Learning Rule

- Amazing fact: If the data is linearly separable, the above algorithm always converges to a weight vector that separates the data.
- If the data is not separable, algorithm does not converge. Need to somehow pick which weight vector to go with.
- There are ways to do this (not examinable), such as the *averaged perceptron* and *voted perceptron*.
- This algorithm is a bit old and frumpy, but can still be very useful. Especially when you add kernels, to get the *kernel perceptron* algorithm.
- Also can be seen as a very simple neural network. see later.

# Artificial Neural Networks

Charles Sutton
School of Informatics

# Outline

- Why multilayer artificial neural networks (ANNs)?
- Representation Power of ANNs
- Training ANNs: backpropagation
- Learning Hidden Layer Representations
- Examples

# What's wrong with the IAML course

*When we write programs that "learn", it turns out that we do and they don't. —Alan Perlis*

- Many of the methods in this course are linear. All of them depend on *representation*, i.e., having good features.
- What if we want to learn the features?
- This lecture: Nonlinear regression and nonlinear classification
- Can think of this as: A linear method where we learn the features
- These are motivated by a (weak) analogy to the human brain, hence the name *artificial neural networks*

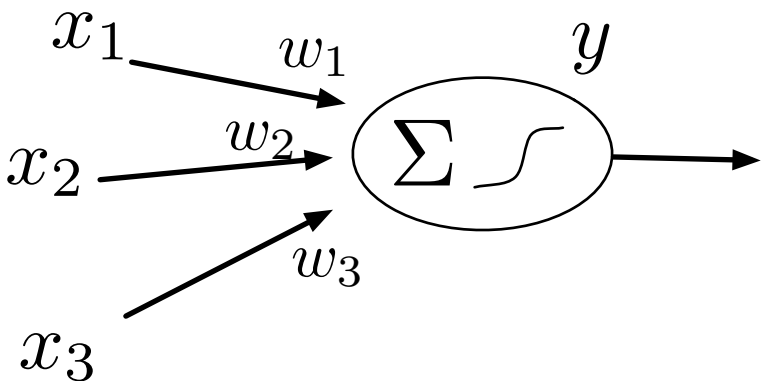
# How artificial neural networks fit into the course

| | Supervised | | Unsupervised | |
|---|---|---|---|---|
| | Class. | Regr. | Clust. | D. R. |
| Naive Bayes | ✓ | | | |
| Decision Trees | ✓ | | | |
| *k*-nearest neighbour | ✓ | | | |
| Linear Regression | | ✓ | | |
| Logistic Regression | ✓ | | | |
| SVMs | ✓ | | | |
| *k*-means | | | ✓ | |
| Gaussian mixtures | | | ✓ | |
| PCA | | | | ✓ |
| Evaluation | | | | |
| **ANNs** | ✔ | ✔ | | |

# Artificial Neural Networks (ANNs)

- The field of neural networks grew up out of simple models of neurons
- Each single neuron looks like a linear unit
- (In fact, *unit* is name for “simulated neuron”.)
- A network of them is nonlinear

# Classification Using a Single Neuron




Take a single input $\mathbf{x} = (x_1, x_2, \ldots x_d)$. To compute a class label

1. Compute the neuron's activation
   $a = \mathbf{x}^\top \mathbf{w} + w_0 = \sum_{d=1}^{D} x_d w_d + w_0$
2. Set the neuron output $y$ as a function of its activation
   $y = g(a)$. For now let's say

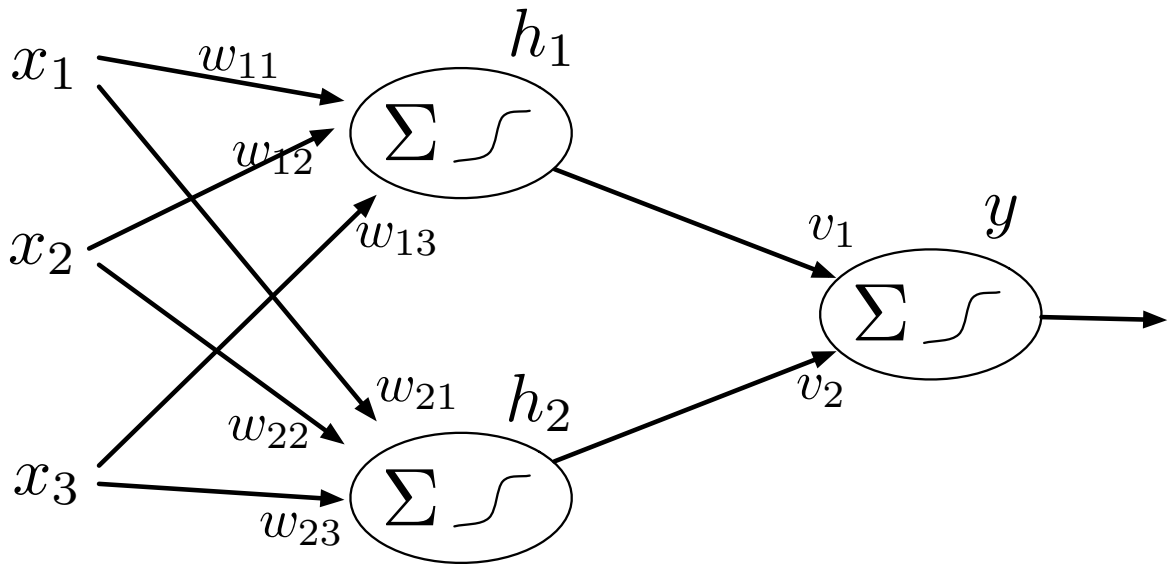
$$g(a) = \sigma(a) = \frac{1}{1 + e^{-a}} \qquad \text{i.e., sigmoid}$$

3. If $y > 0.5$, assign $\mathbf{x}$ to class 1. Otherwise, class 0.

# Why we need multilayer networks

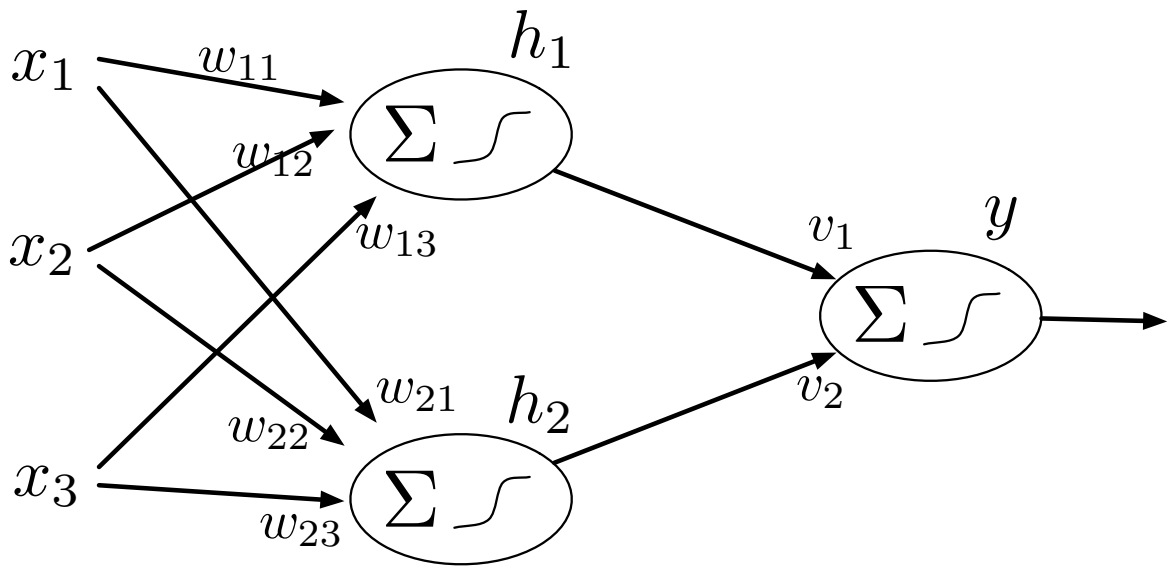
- We haven't done anything new yet.
- This is just a very strange way of presenting logistic regression
- Idea: Use recursion. Use the output of some neurons as input to another neuron that actually predicts the label

# A Slightly More Complex ANN: The Units



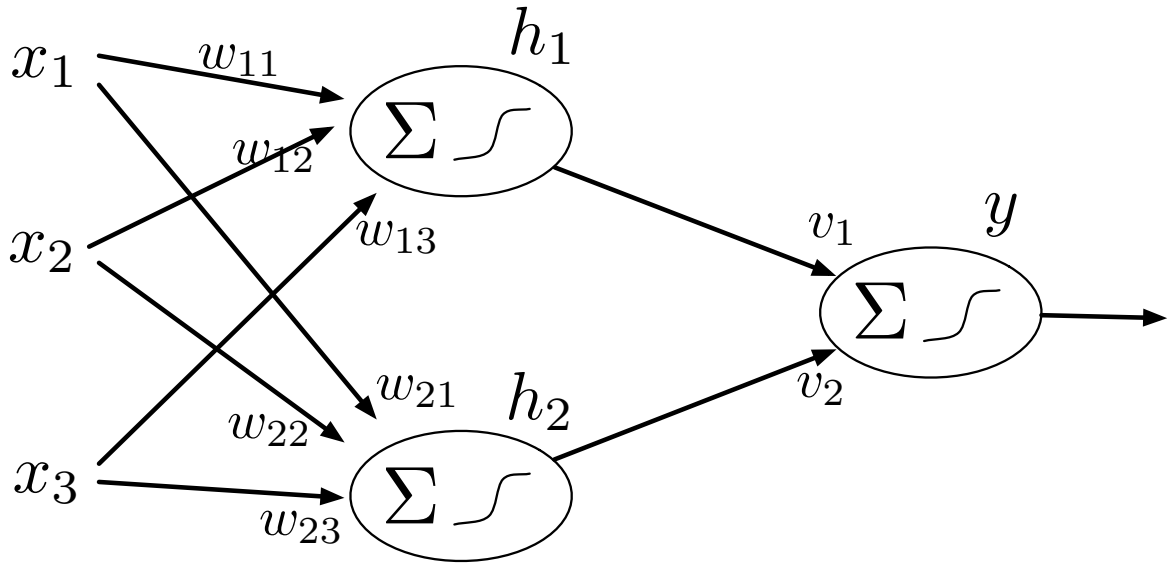

- $x_1$, $x_2$, and $x_3$ are the input features, just like always.
- $y$ is the output of the classifier. In an ANN this is sometimes called an *output unit*.
- The units $h_1$ and $h_2$ don't directly correspond to anything int the data. They are called hidden units

# A Slightly More Complex ANN: The Weights




- Each unit gets its own weight vector.
- $\mathbf{w}_1 = (w_{11}, w_{12}, w_{13})$ are the weights for $h_1$.
- $\mathbf{w}_2 = (w_{21}, w_{22}, w_{23})$ are the weights for $h_2$.
- $\mathbf{v} = (v_1, v_2)$ are the weights for $y$.
- Also each unit gets a "bias weight" $w_{10}$ for unit $h_1$, $w_{20}$ for unit $h_2$ and $v_0$ for unit $y$.
- Use $\mathbf{w} = (\mathbf{w}_1, \mathbf{w}_2, \mathbf{v}, w_{10}, w_{20}, v_0)$ to refer to all of the weights stacked into one vector.

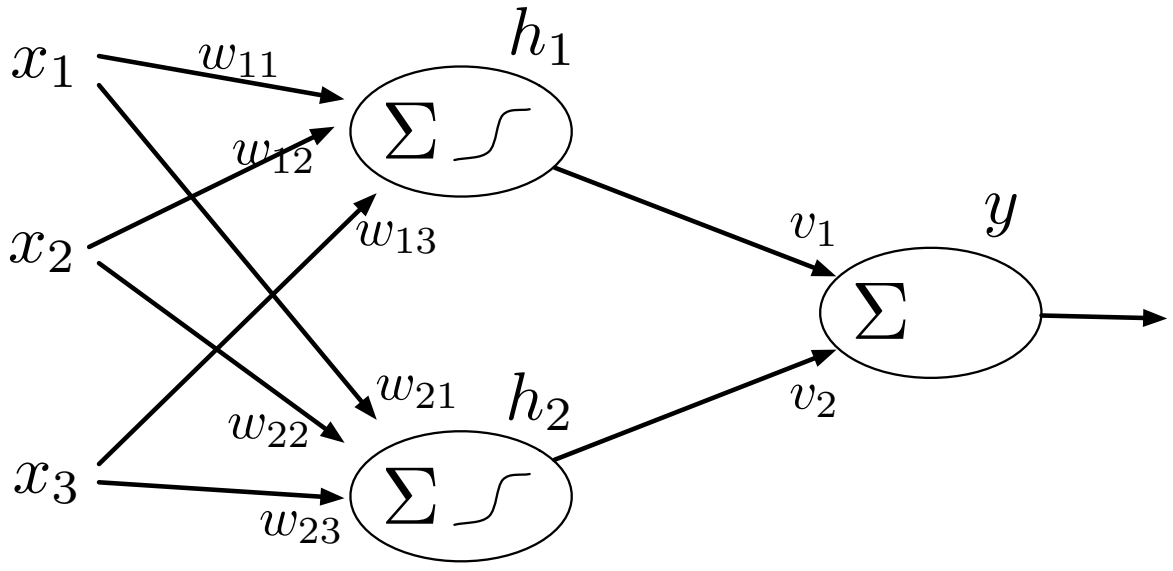
# A Slightly More Complex ANN: Predicting




Here is how to compute a class label in this network:

1. $h_1 \leftarrow g(\mathbf{w}_1^T \mathbf{x} + w_{10}) = g(\sum_{d=1}^{D} w_{1d} x_d + w_{10})$
2. $h_2 \leftarrow g(\mathbf{w}_2^T \mathbf{x} + w_{20}) = g(\sum_{d=1}^{D} w_{2d} x_d + w_{20})$
3. $y \leftarrow g(\mathbf{v}^T \begin{pmatrix} h_1 \\ h_2 \end{pmatrix} + v_0) = g(v_1 h_1 + v_2 h_2 + v_0)$
4. If $y > 0.5$, assign to class 1, i.e., $f(\mathbf{x}) = 1$. Otherwise $f(\mathbf{x}) = 0$.

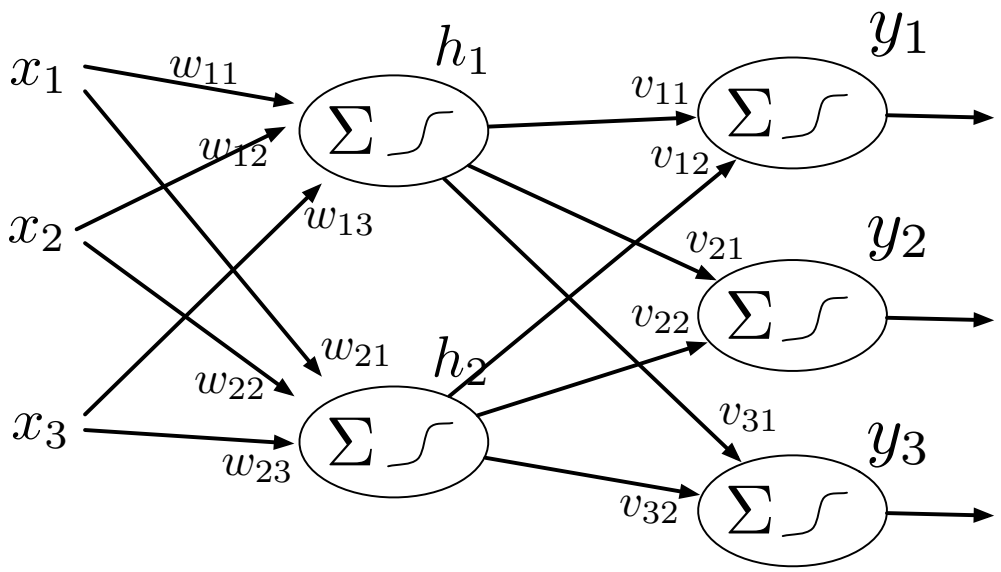
# ANN for Regression




If you want to do regression instead of classification, it's simple. Just don't squash the output. Here is how to make a real-valued prediction:

1. $h_1 \leftarrow g(\mathbf{w}_1^T \mathbf{x} + w_{10}) = g(\sum_{d=1}^{D} w_{1d} x_d + w_{10})$
2. $h_2 \leftarrow g(\mathbf{w}_2^T \mathbf{x} + w_{20}) = g(\sum_{d=1}^{D} w_{2d} x_d + w_{20})$
3. $y \leftarrow g_3(\mathbf{v}^T \begin{pmatrix} h_1 \\ h_2 \end{pmatrix} + v_0) = g_3(v_1 h_1 + v_2 h_2 + v_0)$
   where $g_3(a) = a$ the identity function.
4. Return $f(\mathbf{x}) = y$ as the prediction of the real-valued output

# ANN for Multiclass Classification



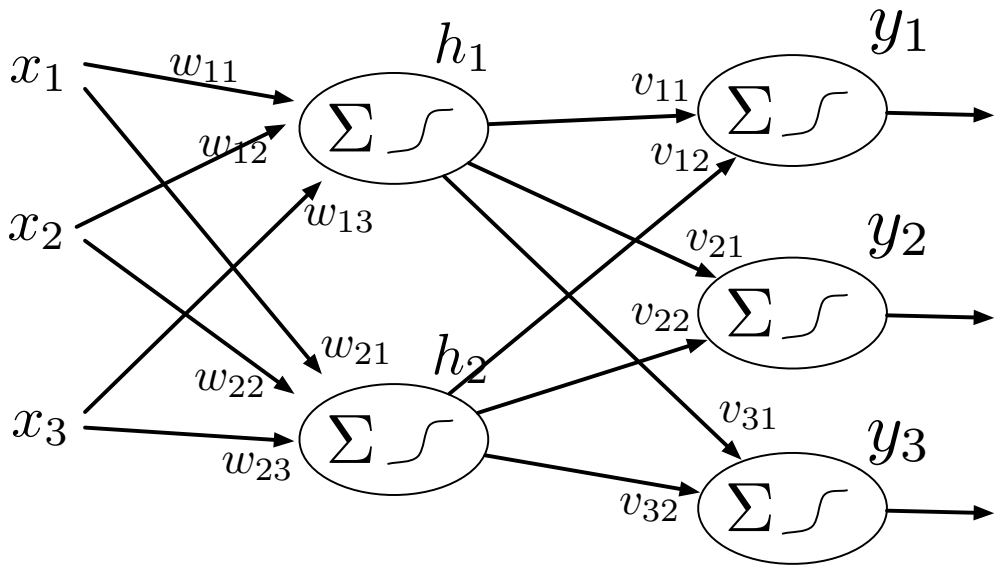

More than two classes? No problem. Only change is to output layer. Define one output unit for each class.

$$y_i \leftarrow \text{how likely it is that } \mathbf{x} \text{ in class } i, (i = 1 \ldots M)$$

Then convert to probabilities using a softmax function.

$$p(y = m|\mathbf{x}) = \frac{e^{y_m}}{\sum_{k=1}^{M} e^{y_k}}$$

# Multiclass ANN: Making a Prediction

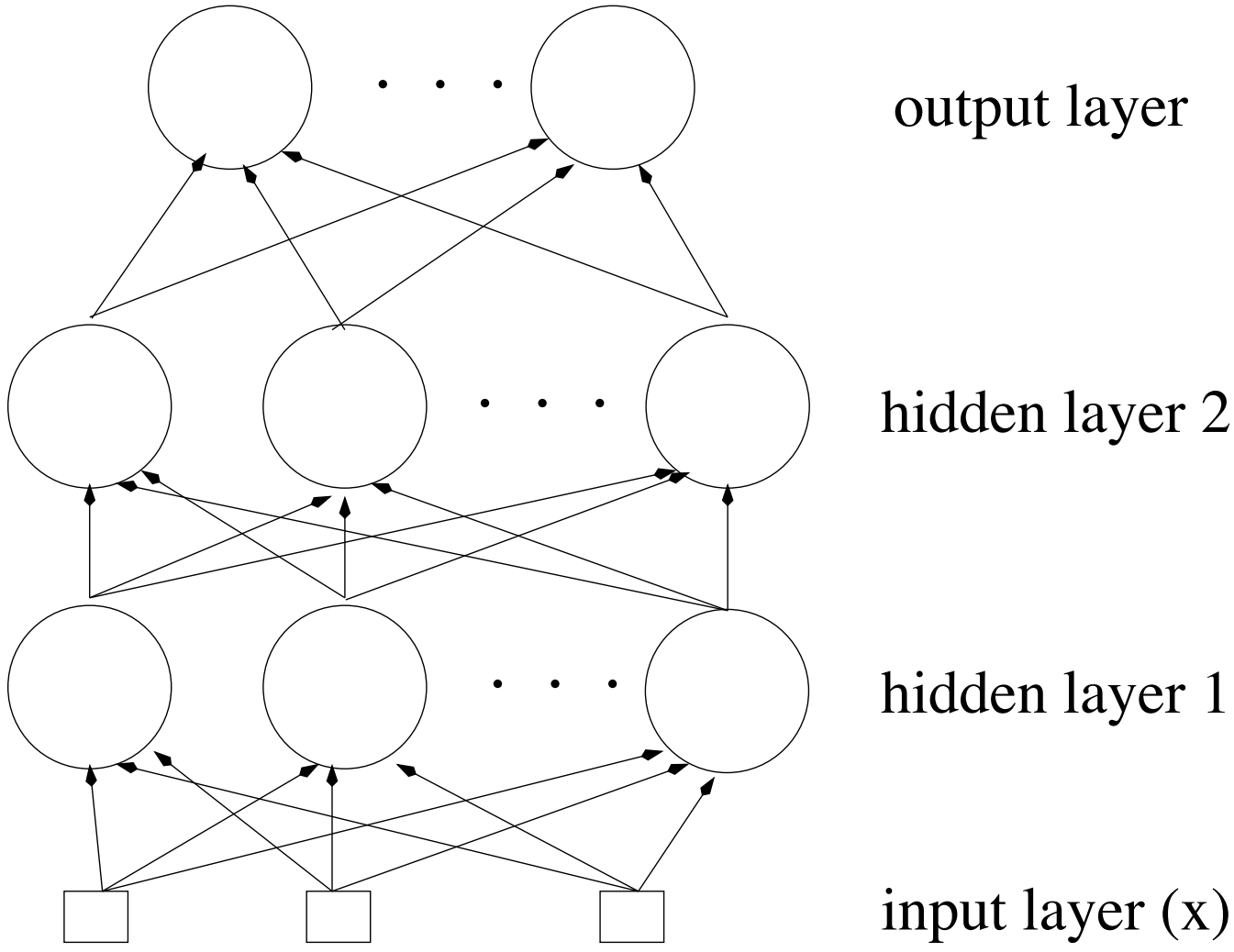



1. $h_1 \leftarrow g(\mathbf{w}_1^T \mathbf{x} + w_{10}) = g(\sum_{d=1}^{D} w_{1d} x_d + w_{10})$
2. $h_2 \leftarrow g(\mathbf{w}_2^T \mathbf{x} + w_{20}) = g(\sum_{d=1}^{D} w_{2d} x_d + w_{20})$
3. for all $m \in 1, 2, \ldots M$,

   $$y_m \leftarrow \mathbf{v}_m^T \begin{pmatrix} h_1 \\ h_2 \end{pmatrix} + v_{m0} = v_{m1} h_1 + v_{m2} h_2 + v_{m0}$$
4. Prediction $f(\mathbf{x})$ is the class with the highest probability

$$p(y = m|\mathbf{x}) = \frac{e^{y_m}}{\sum_{k=1}^{M} e^{y_k}} \qquad f(\mathbf{x}) = \max_{m=1}^{M} p(y = m|\mathbf{x})$$

You can have more hidden layers and more units. An example network with 2 hidden layers



- There can be an arbitrary number of hidden layers
- The networks that we have seen are called *feedforward* because the structure is a directed acyclic graph (DAG).
- Each unit in the first hidden layer computes a non-linear function of the input $\mathbf{x}$
- Each unit in a higher hidden layer computes a non-linear function of the outputs of the layer below

# Things that you get to tweak

- The structure of the network: How many layers? How many hidden units?
- What activation function $g$ to use for all the units.
- For the output layer this is easy:
  - $g$ is the identity function for a regression task
  - $g$ is the logistic function for a two-class classification task
- For the hidden layers you have more choice:

$$g(a) = \sigma(a) \qquad \text{i.e., sigmoid}$$

$$g(a) = \tanh(a)$$

$$g(a) = a \qquad \text{linear unit}$$

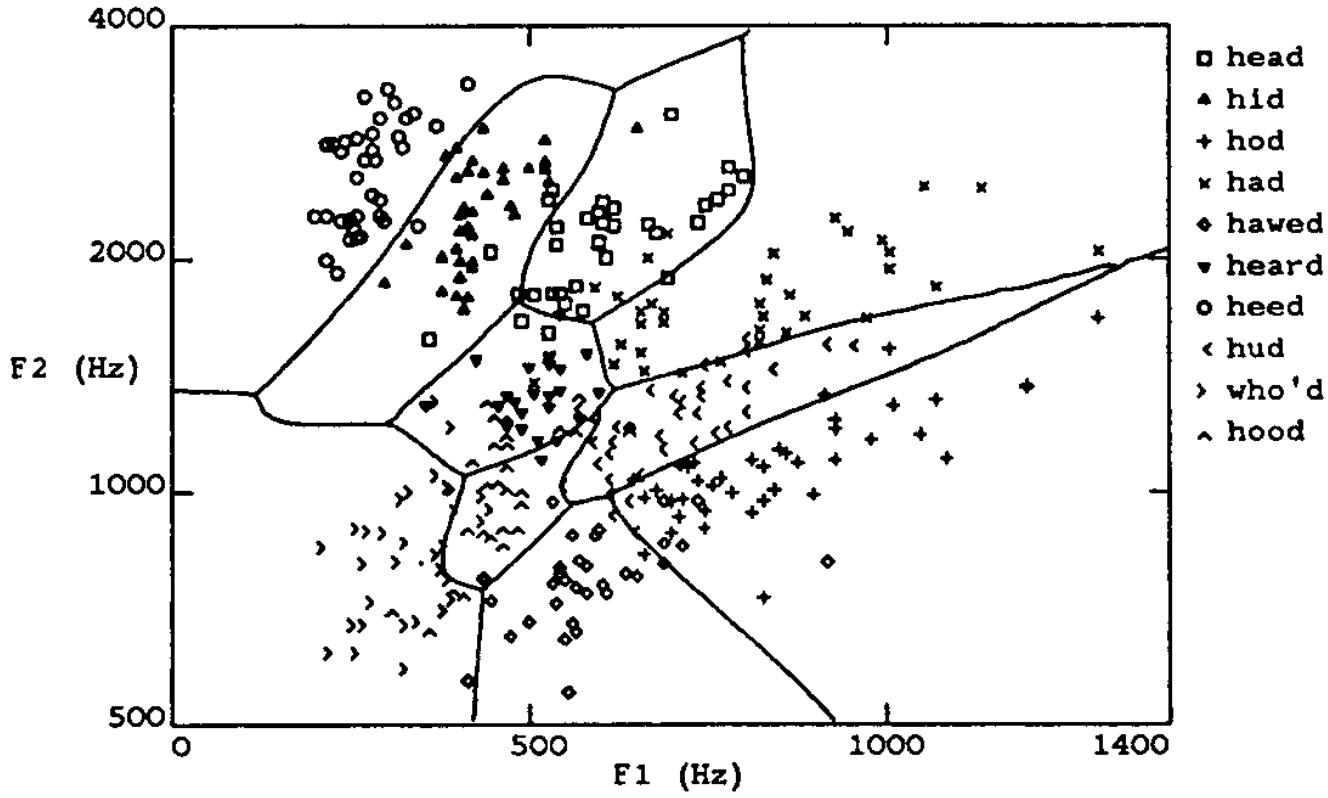
$$g(a) = \text{Gaussian density} \qquad \text{radial basis network}$$

$$g(a) = \Theta(a) = \begin{cases} 1 & \text{if } a \geq 0 \\ -1 & \text{if } a < 0 \end{cases} \qquad \text{threshold unit}$$

- Tweaking all of these can be a black art

# Representation Power of ANNs

- Boolean functions:
  - Every boolean function can be represented by network with single hidden layer
  - but might require exponentially many (in number of inputs) hidden units
- Continuous functions:
  - Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer [Cybenko 1989; Hornik et al. 1989]
  - Any function can be approximated to arbitrary accuracy by a network with two hidden layers [Cybenko 1988]. This follows from a famous result of Kolmogorov.
  - Neural Networks are *universal approximators*.
  - But again, if the function is complex, two hidden layers may require an extremely large number of units
- Advanced (non-examinable): For more on this see,
  - F. Girosi and T. Poggio. “Kolmogorov’s theorem is irrelevant.” Neural Computation, 1(4):465469, 1989.
  - V. Kurkova, “Kolmogorov’s Theorem Is Relevant”, Neural Computation, 1991, Vol. 3, pp. 617622.




# ANN predicting 1 of 10 vowel sounds based on formats F1 and F2

Figure from Mitchell (1997)

# Training ANNs

- Training: Finding the best weights for each unit
- We create an error function that measures the agreement of the target $y_i$ and the prediction $f(\mathbf{x})$
- Linear regression, squared error: $E = \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i))^2$
- Logistic regression (0/1 labels): $E = \sum_{i=1}^{n} y_i \log f(\mathbf{x}_i) + (1 - y_i) \log(1 - f(\mathbf{x}_i))$
- It can make sense to use a regularization penalty (e.g. $\lambda |\mathbf{w}|^2$) to help control overfitting; in the ANN literature this is called *weight decay*
- The name of the game will be to find $\mathbf{w}$ so that $E$ is minimized.
- For linear and logistic regression the optimization problem for $\mathbf{w}$ had a unique optimum; this is no longer the case for ANNs (e.g. hidden layer neurons can be permuted)

# Backpropagation

- As discussed for logistic regression, we need the gradient of $E$ wrt all the parameters $\mathbf{w}$, i.e. $\mathbf{g}(\mathbf{w}) = \frac{\partial E}{\partial \mathbf{w}}$
- There is a clever recursive algorithm for computing the derivatives. It uses the chain rule, but stores some intermediate terms. This is called *backpropagation*.
- We make use of the layered structure of the net to compute the derivatives, heading backwards from the output layer to the inputs
- Once you have $\mathbf{g}(\mathbf{w})$, you can use your favourite optimization routines to minimize $E$; see discussion of gradient descent and other methods in Logistic Regression slides

# Convergence of Backpropagation

- Dealing with local minima. Train multiple nets from different starting places, and then choose best (or combine in some way)
- Initialize weights near zero; therefore, initial networks are near-linear
- Increasingly non-linear functions possible as training progresses

# Training ANNs: Summary

- Optimize over vector of all weights/biases in a network
- All methods considered find *local* optima
- Gradient descent is simple but slow
- In practice, second-order methods (*conjugate gradients*) are used for batch learning
- Overfitting can be a problem

# Applications of Neural Networks

- Recognizing handwritten digits on cheques and post codes (LeCun and Bengio, 1995)
- Language modelling: Given a partial sentence “Neural networks are”, predict the next word (Y Bengio et al, 2003)
- Financial forecasting
- Speech recognition

# ANNs: Summary

- Artificial neural networks are a powerful nonlinear modelling tool for classification and regression
- These were never very good models of the brain. But as classifiers, they work.
- The hidden units are new representation of the original input. Think of this as learning the features
- Trained by optimization methods making use of the backpropagation algorithm to compute derivatives
- Local optima in optimization are present, cf linear and logistic regression (and kernelized versions thereof, e.g. SVM)
- Ability to automatically discover useful hidden-layer representations