# Statistical Machine Learning
# Hilary Term 2020

**Pier Francesco Palamara**
Department of Statistics
University of Oxford

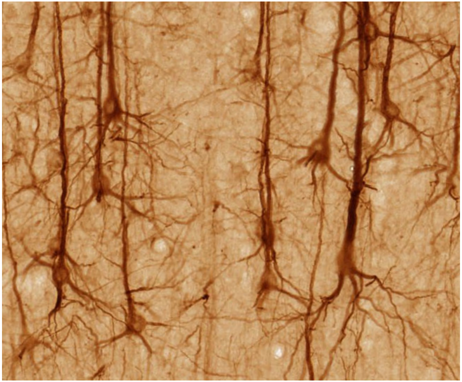
Slide credits and other course material can be found at:
http://www.stats.ox.ac.uk/~palamara/SML20.html

February 27, 2020

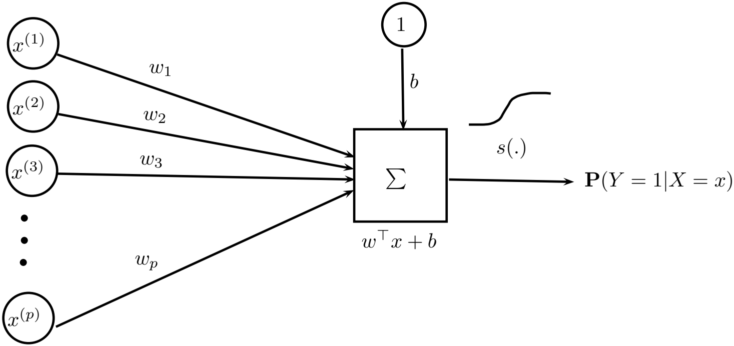
# Neural Networks

# Biological inspiration

- Basic computational elements: neurons.
- Receives signals from other neurons via dendrites.
- Sends processed signals via axons.
- Axon-dendrite interactions at synapses.
- $10^{10} - 10^{11}$ neurons.
- $10^{14} - 10^{15}$ synapses.

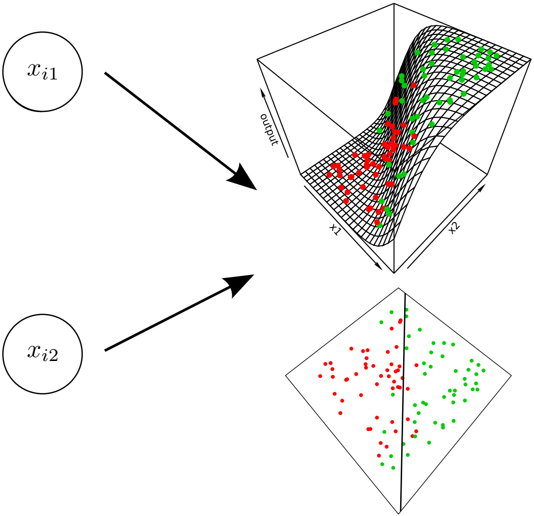


# Single Neuron Classifier



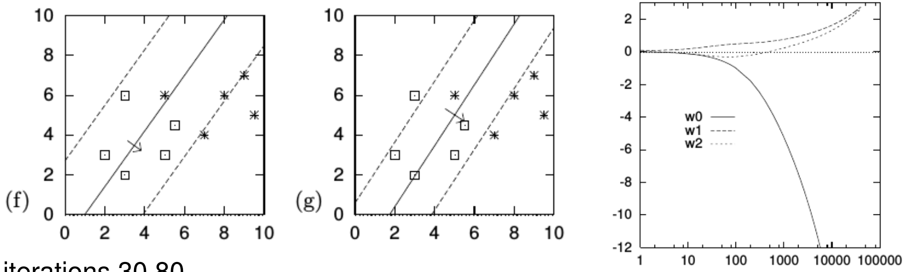

- **activation** $w^\top x + b$ (linear in **inputs** $x$)
- **activation/transfer function** $s$ gives the **output/activity** (potentially nonlinear in $x$)
- $b$ often called **bias** (not to be confused with other biases we discussed!)
- common nonlinear activation function $s(a) = \frac{1}{1+e^{-a}}$: **logistic regression**
- learn $w$ and $b$ via gradient descent

# Single Neuron Classifier




$i$ is the index of a training point.

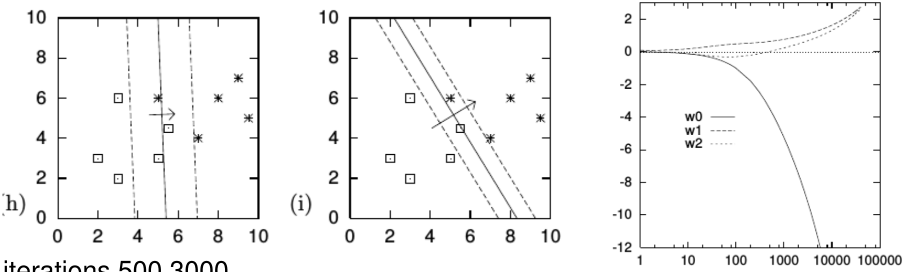
# Overfitting




iterations 30,80

Figures from D. MacKay, **Information Theory, Inference and Learning Algorithms**

# Overfitting

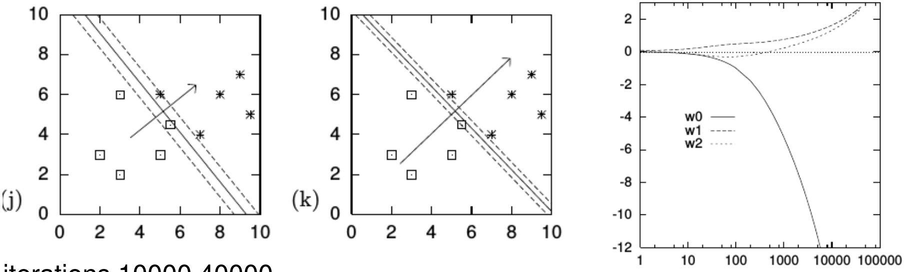



iterations 500,3000



Figures from D. MacKay, **Information Theory, Inference and Learning Algorithms**

# Overfitting




iterations 10000,40000

prevent overfitting by:

- **early stopping**: just halt the gradient descent
- regularization: $L_2$-regularization called **weight decay** in neural networks literature.

Figures from D. MacKay, **Information Theory, Inference and Learning Algorithms**

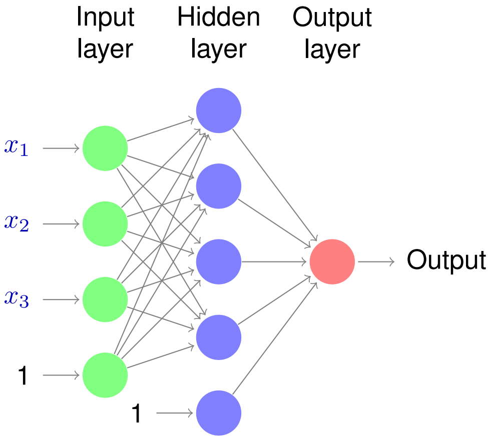
# Multilayer Networks

- Data vectors $\boldsymbol{x} \in \mathbb{R}^p$, binary labels $y \in \{0, 1\}$.
- **inputs** $\boldsymbol{x} = [x_1, \ldots, x_p]^\top$
- **output** $\hat{y} = \mathbb{P}(Y = 1 | X = \boldsymbol{x})$
- **hidden unit activities** $h_1, \ldots, h_m$
  - Compute **hidden unit activities**:

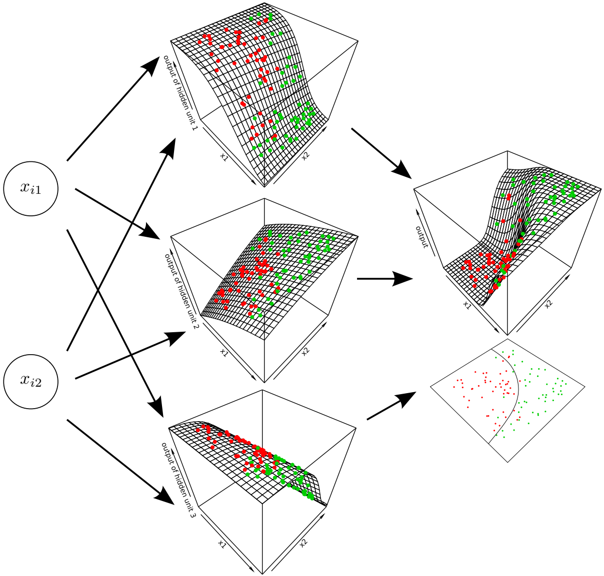
$$h_\ell = s\left(b_\ell^h + \sum_{j=1}^{p} w_{j\ell}^h x_j\right)$$

  - Compute **output probability**:

$$\hat{y} = s\left(b^o + \sum_{\ell=1}^{m} w_k^o h_\ell\right)$$



# Multilayer Networks



# Training a Neural Network

- Objective function: $L_2$-regularized log-loss

$$J = -\sum_{i=1}^{n} y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i) + \frac{\lambda}{2} \left( \sum_{jl} (w_{jl}^h)^2 + \sum_{l} (w_l^o)^2 \right)$$

where

$$\hat{y}_i = s\left(b^o + \sum_{l=1}^{m} w_l^o h_{il}\right) \qquad h_{il} = s\left(b_l^h + \sum_{j=1}^{p} w_{jl}^h x_{ij}\right)$$

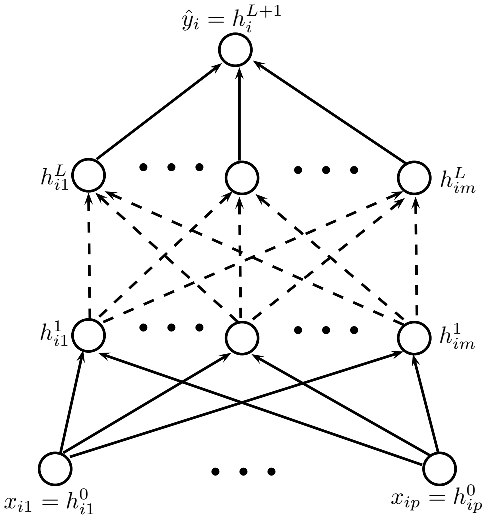
- Optimize parameters $\theta = \{b^h, w^h, b^o, w^o\}$, where $b^h \in \mathbb{R}^m$, $w^h \in \mathbb{R}^{p \times m}$, $b^o \in \mathbb{R}$, $w^o \in \mathbb{R}^m$ with gradient descent.

$$\frac{\partial J}{\partial w_l^o} = \lambda w_l^o + \sum_{i=1}^{n} \frac{\partial J}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial w_l^o} = \lambda w_l^o + \sum_{i=1}^{n} (\hat{y}_i - y_i) h_{il},$$

$$\frac{\partial J}{\partial w_{jl}^h} = \lambda w_{jl}^h + \sum_{i=1}^{n} \frac{\partial J}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial h_{il}} \frac{\partial h_{il}}{\partial w_{jl}^h} = \lambda w_{jl}^h + \sum_{i=1}^{n} (\hat{y}_i - y_i) w_l^o h_{il}(1 - h_{il}) x_{ij}.$$

- $L_2$-regularization often called **weight decay**.
- Multiple hidden layers: **Backpropagation** algorithm

# Multiple hidden layers



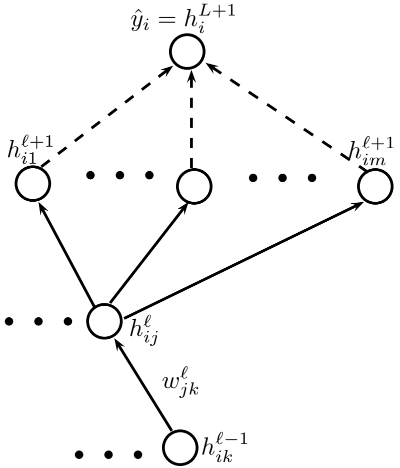

$$h_i^{\ell+1} = \underline{s}\left(W^{\ell+1} h_i^\ell\right)$$

- $W^{\ell+1} = \left(w_{jk}^\ell\right)_{jk}$: weight matrix at the $(\ell+1)$-th layer, weight $w_{jk}^\ell$ on the edge between $h_{ik}^{\ell-1}$ and $h_{ij}^\ell$
- $\underline{s}$: entrywise (logistic) transfer function

$$\hat{y}_i = \underline{s}\left(W^{L+1}\underline{s}\left(W^L\left(\cdots \underline{s}\left(W^1 x_i\right)\right)\right)\right)$$

- **Many** hidden layers can be used: they are usually thought of as forming a hierarchy from low-level to high-level features.

# Backpropagation

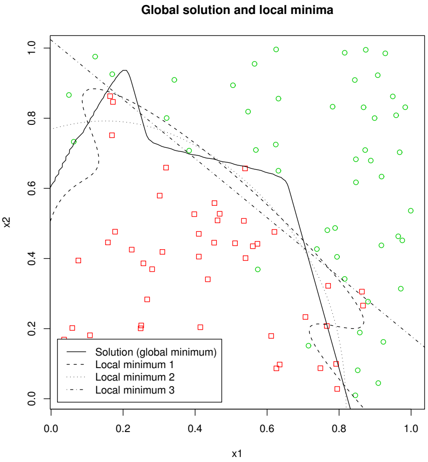


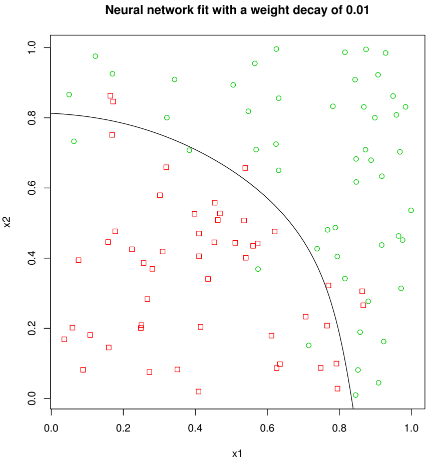

$$J = -\sum_{i=1}^{n} y_i \log h_i^{L+1} + (1-y_i)\log(1-h_i^{L+1})$$

- Gradients wrt $h_{ij}^{\ell}$ computed by recursive applications of chain rule, and propagated through the network backwards.

$$\begin{aligned}
\frac{\partial J}{\partial h_i^{L+1}} &= -\frac{y_i}{h_i^{L+1}} + \frac{1-y_i}{1-h_i^{L+1}} \\
\frac{\partial J}{\partial h_{ij}^{\ell}} &= \sum_{r=1}^{m} \frac{\partial J}{\partial h_{ir}^{\ell+1}} \frac{\partial h_{ir}^{\ell+1}}{\partial h_{ij}^{\ell}} \\
\frac{\partial J}{\partial w_{jk}^{\ell}} &= \sum_{i=1}^{n} \frac{\partial J}{\partial h_{ij}^{\ell}} \frac{\partial h_{ij}^{\ell}}{\partial w_{jk}^{\ell}}
\end{aligned}$$

# Neural Networks




R package implementing neural networks with a single hidden layer: `nnet`.

# Dropout Training of Neural Networks

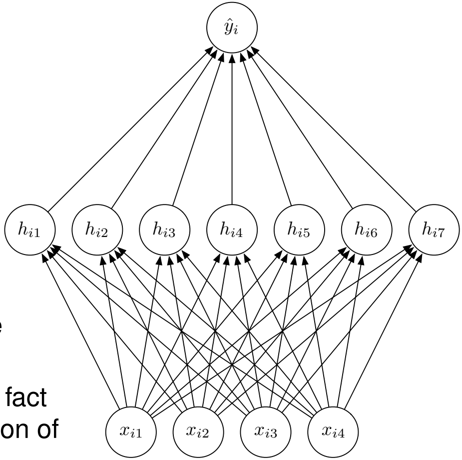
- Neural network with single layer of hidden units:
  - **Hidden unit activations**:

$$h_{ik} = s\left(b_k^h + \sum_{j=1}^{p} W_{jk}^h x_{ij}\right)$$

  - **Output probability**:

$$\hat{y}_i = s\left(b^o + \sum_{k=1}^{m} W_k^o h_{ik}\right)$$

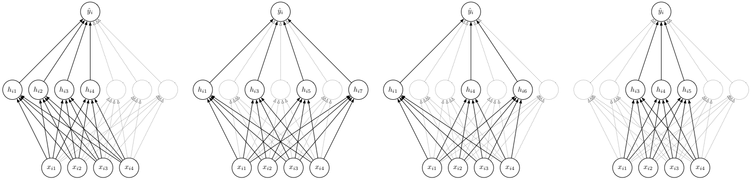
- Large, overfitted networks often have co-adapted hidden units.
- What each hidden unit learns may in fact be useless, e.g. predicting the negation of predictions from other units.
- Can prevent co-adaptation by randomly **dropping out** units from network.




Hinton et al (2012).

# Dropout Training of Neural Networks

- Model as an **ensemble** of networks (more on ensembles later):




- **Weight-sharing** among all networks: each network uses a subset of the parameters of the full network (corresponding to the retained units).
- Training by stochastic gradient descent: at each iteration a network is sampled from ensemble, and its subset of parameters are updated.
- Biological inspiration: $10^{14}$ weights to be fitted in a lifetime of $10^{9}$ seconds
  - Poisson spikes as a regularization mechanism which prevents co-adaptation: Geoff Hinton on Brains, Sex and Machine Learning

# Dropout Training of Neural Networks

Classification of phonemes in speech.

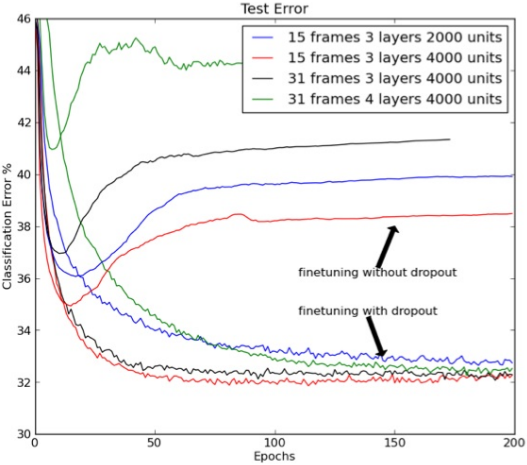



Figure from Hinton et al.

# Neural Networks – Variations

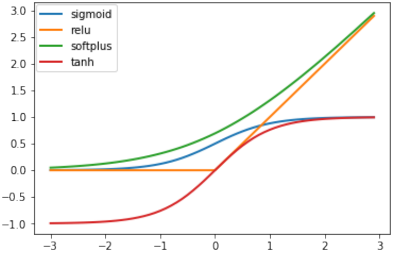
- Other loss functions can be used, e.g. for regression:

$$\sum_{i=1}^{n} |y_i - \hat{y}_i|^2$$

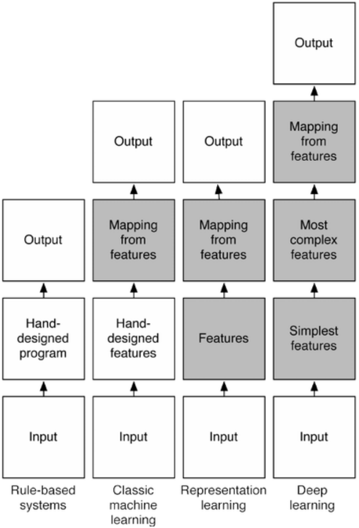
For multiclass classification, use **softmax** outputs:

$$\hat{y}_{ik} = \frac{\exp(b_k^o + \sum_\ell w_{lk}^o h_{i\ell})}{\sum_{k'=1}^{K} \exp(b_{k'}^o + \sum_\ell w_{lk'}^o h_{i\ell})} \qquad L(y_i, \hat{y}_i) = -\sum_{k=1}^{K} \mathbb{1}(y_i = k) \log \hat{y}_{ik}$$

- Other activation functions can be used:
  - **rectified linear unit (ReLU)**: $s(z) = \max(0, z)$
  - **softplus**: $s(z) = \log(1 + \exp(z))$
  - **tanh**: $s(z) = \tanh(z)$



# Deep learning intuition




Source: http://rinuboney.github.io/2015/10/18/theoretical-motivations-deep-learning.html

# Deep learning demo

http://playground.tensorflow.org/

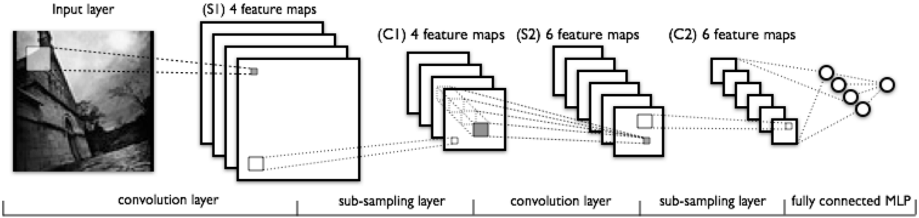
# Convolutions

| Operation | Filter | Convolved Image |
|---|---|---|
| Identity | $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ |  |
| Edge detection | $\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$ |  |
| | $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ |  |
| | $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$ |  |
| Sharpen | $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ |  |
| Box blur (normalized) | $\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ |  |
| Gaussian blur (approximation) | $\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ |  |

Click for animation.

Source: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

# Deep Convolutional Neural Networks




- Input is a 2D image, $X \in \mathbb{R}^{p \times q}$.
- **Convolution**: detects simple object parts or features Weights $W^m$ now correspond to a **filter** to be learned - typically much smaller than the input thus encouraging sparse connectivity.
- **Pooling and Sub-sampling**: replace the output with a summary statistic of the nearby outputs, e.g. max-pooling (allows invariance to small translations in the input).

LeCun et al, Krizhevsky et al.

# Neural Networks – Discussion

- Nonlinear hidden units introduce modelling flexibility, hierarchical representations.
- In contrast to user-introduced nonlinearities, features are global, and can be learned to maximize predictive performance.
- Neural networks with a single hidden layer and sufficiently many hidden units can model arbitrarily complex functions.
- Highly flexible framework, with many variations to solve different learning problems and introduce domain knowledge.
- Optimization problem is **not convex**, and objective function can have many local optima, plateaus and ridges.
- On large scale problems, often use **stochastic gradient descent**, along with a whole host of techniques for optimization, regularization, and initialization.
- Explosion of interest in the field recently and **many new developments** not covered here, especially by Geoffrey Hinton, Yann LeCun, Yoshua Bengio, Andrew Ng and others. See also `http://deeplearning.net/`.