Data Science: Principles and Practice

Lecture 7

Guy Emerson

Today’s Lecture

- Neural networks:
  - Architectures
  - Training
- Overfitting

Features

input ⟶ features ⟶ prediction

Features

input ⟶ features ⟶ prediction

engineered

Features

input ⟶ features ⟶ prediction

engineered trained

Features

input ⟶ features ⟶ prediction

trained trained

Features

input → features → prediction

trained trained

- Engineering at a more abstract level

Feedforward Networks

$$x \mapsto f_1(x) \mapsto f_2(f_1(x))$$

Feedforward Networks

$$x \mapsto f_1(x) \mapsto f_2(f_1(x))$$

- Linear: $f(x) = Ax$

Feedforward Networks

$$x \mapsto f_1(x) \mapsto f_2(f_1(x))$$

- Linear: $f(x) = Ax$
- but can simplify matrix multiplication $AB = C$

Feedforward Networks

$$x \mapsto f_1(x) \mapsto f_2(f_1(x))$$

- Nonlinear: $f(x) = g(Ax)$

Feedforward Networks

$$x \mapsto f_1(x) \mapsto f_2(f_1(x))$$

- Nonlinear: $f(x) = g(Ax)$
  (g applied componentwise)

Feedforward Networks

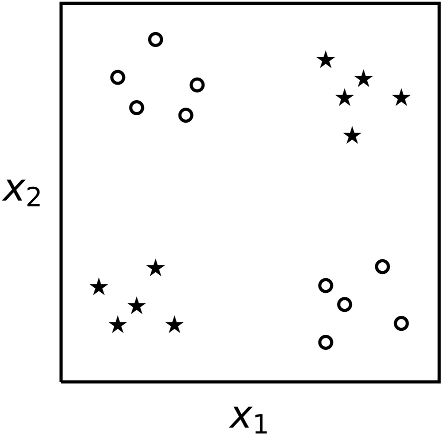
$$x \mapsto f_1(x) \mapsto f_2(f_1(x))$$

- Nonlinear: $f(x) = g(Ax)$
  (g applied componentwise)
- Can approximate any function

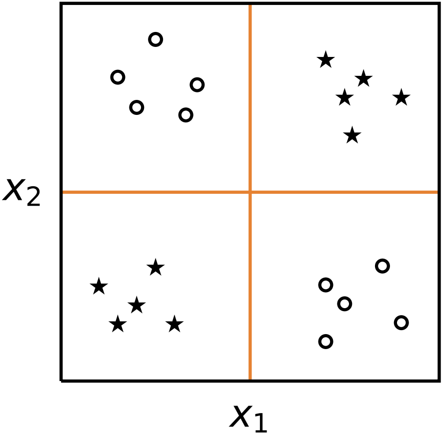
Nonlinear Activation Functions

- $\frac{1}{1+e^{-x}}$ "sigmoid" (cf. logistic regression)
- $\frac{1-e^{-2x}}{1+e^{-2x}}$ "tanh"
- $\max\{x, 0\}$ "rectified linear"
- $\log(1+e^x)$ "softplus"

Nonlinear Decision Boundaries

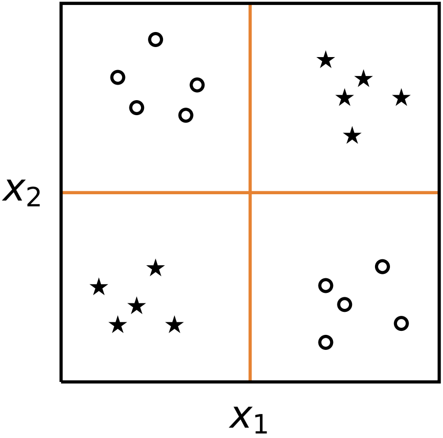


Nonlinear Decision Boundaries



Can be done with a decision tree

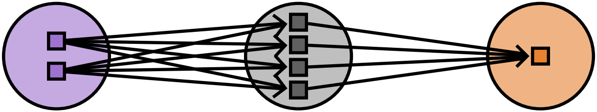
Nonlinear Decision Boundaries



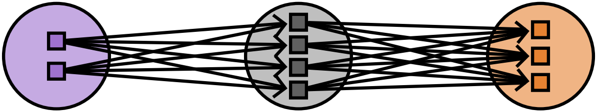
Rectified linear units:

$$
\begin{aligned}
&r(\;\;x_1 + x_2 - 2) \\
+\,&r(-x_1 - x_2 + 2) \\
-\,&r(\;\;x_1 - x_2) \\
-\,&r(-x_1 + x_2)
\end{aligned}
$$

Feedforward Networks

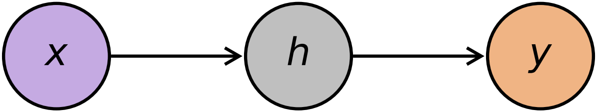


Feedforward Networks

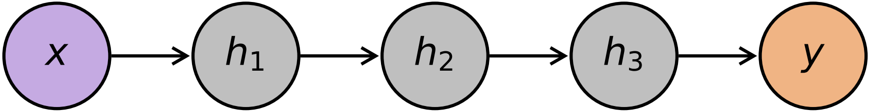


Multiple classes: “softmax”
(multiclass logistic regression)

Feedforward Networks



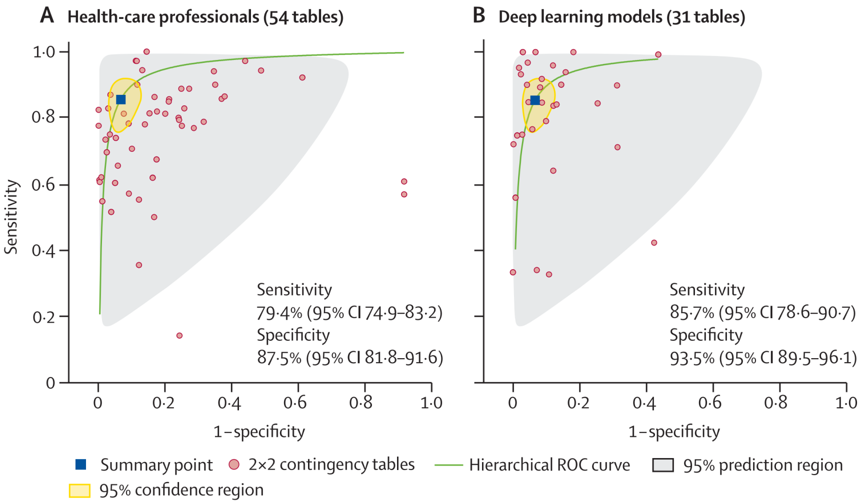
“Deep” Feedforward Networks



AlphaGo


THE ULTIMATE GO CHALLENGE
GAME 3 OF 3
27 MAY 2017
VS
AlphaGo
Winner of Match 3
Ke Jie
RESULT B + Res

Diagnosis from medical imaging



Sequence Labelling

Every picture tells a story

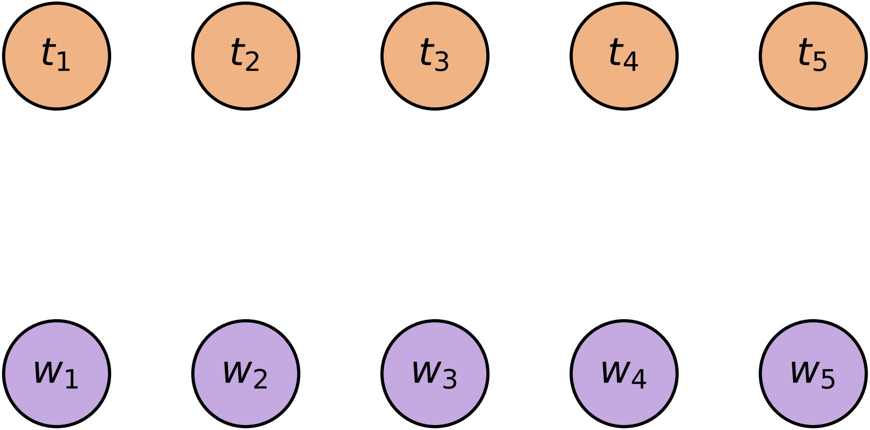
Sequence Labelling

article *noun* *verb* *article* *noun*

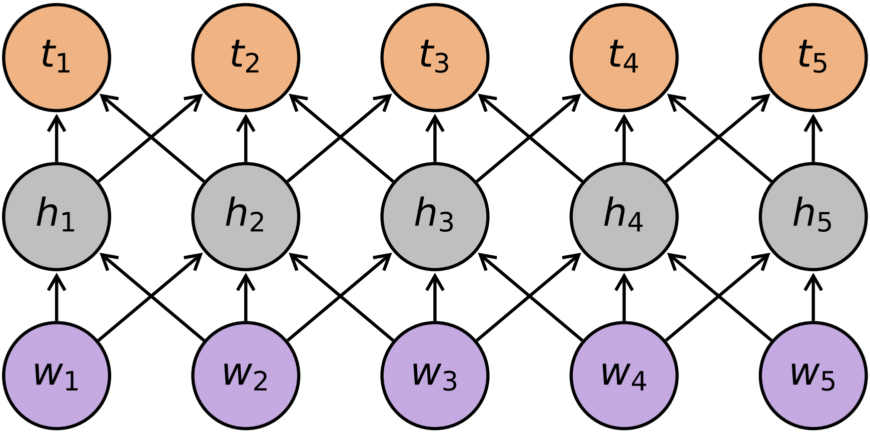
↑

Every picture tells a story

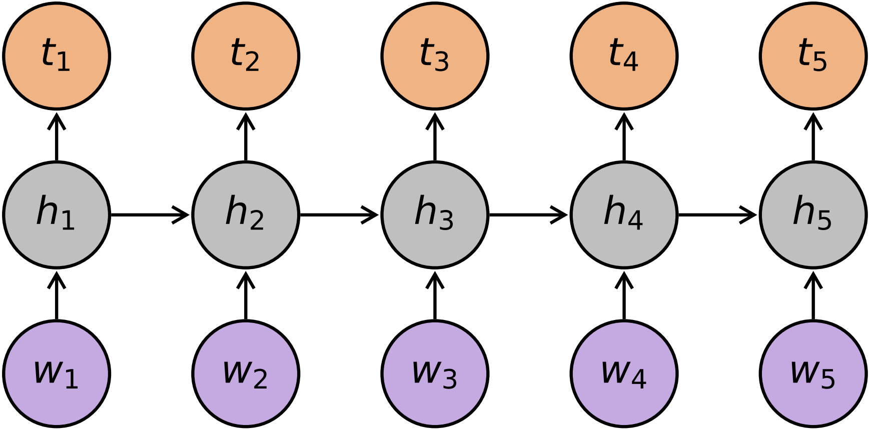
Sequence Labelling



Convolutional Neural Net



Recurrent Neural Net



Training a Network

- Loss function: $\mathcal{L}(\hat{y}, y)$

Training a Network

- Loss function: $\mathcal{L}(\hat{y}, y)$
- Gradient wrt parameters: $\frac{d}{d\theta}\left(\mathcal{L}(\hat{y}, y)\right)$

Training a Network

- Loss function: $\mathcal{L}(\hat{y}, y)$
- Gradient wrt parameters: $\frac{d}{d\theta}\left(\mathcal{L}(\hat{y}, y)\right)$
- Update: $\theta \leftarrow \theta - \alpha \frac{d}{d\theta}\left(\mathcal{L}(\hat{y}, y)\right)$

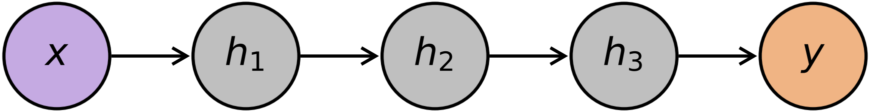
Backpropagation

- Chain rule: $\frac{d\mathcal{L}}{d\theta} = \frac{d\mathcal{L}}{du}\frac{du}{d\theta}$

Backpropagation

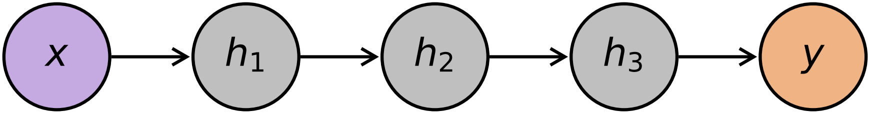
- Chain rule: $\frac{d\mathcal{L}}{d\theta} = \frac{d\mathcal{L}}{du}\frac{du}{d\theta}$
- Backprop: efficient chain rule

Backpropagation



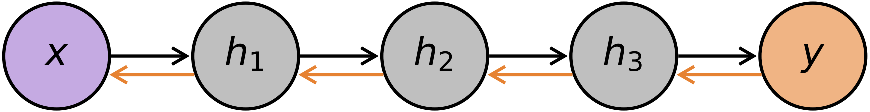
Backpropagation

Forward pass



Backpropagation

Forward pass



Backward pass
(calculate gradients with chain rule)

Training Hyperparameters

- Large learning rate:
  - Faster training
- Small learning rate:
  - More stable training

Gradient Descent

- Loss per datapoint: $\mathcal{L}(\hat{y}_i, y_i)$

Gradient Descent

- Loss per datapoint: $\mathcal{L}(\hat{y}_i, y_i)$
- Total training loss: $\sum_{i=1}^{N} \mathcal{L}(\hat{y}_i, y_i)$

Gradient Descent

- Loss per datapoint: $\mathcal{L}(\hat{y}_i, y_i)$
- Total training loss: $\sum_{i=1}^{N} \mathcal{L}(\hat{y}_i, y_i)$
- Ideal gradient: $\frac{d}{d\theta}\left(\sum_{i=1}^{N} \mathcal{L}(\hat{y}_i, y_i)\right)$

Stochastic Gradient Descent

- Ideal gradient: $\sum_{i=1}^{N} \frac{d}{d\theta} \mathcal{L}(\hat{y}_i, y_i)$
- Stochastic gradient: $\frac{d}{d\theta} \mathcal{L}(\hat{y}_i, y_i)$
  for $i = 1, 2, 3, \ldots$

Stochastic Gradient Descent

- Ideal gradient: $\sum_{i=1}^{N} \frac{d}{d\theta} \mathcal{L}(\hat{y}_i, y_i)$
- Stochastic gradient: $\frac{d}{d\theta} \mathcal{L}(\hat{y}_i, y_i)$ for $i = 1, 2, 3, \ldots$
- Minibatch gradient: $\sum_{i=j+1}^{j+b} \frac{d}{d\theta} \mathcal{L}(\hat{y}_i, y_i)$ for $j = 0, b, 2b, \ldots$

Training Hyperparameters

- Small batch size, large learning rate:
  - Faster training
- Large batch size, small learning rate:
  - More stable training

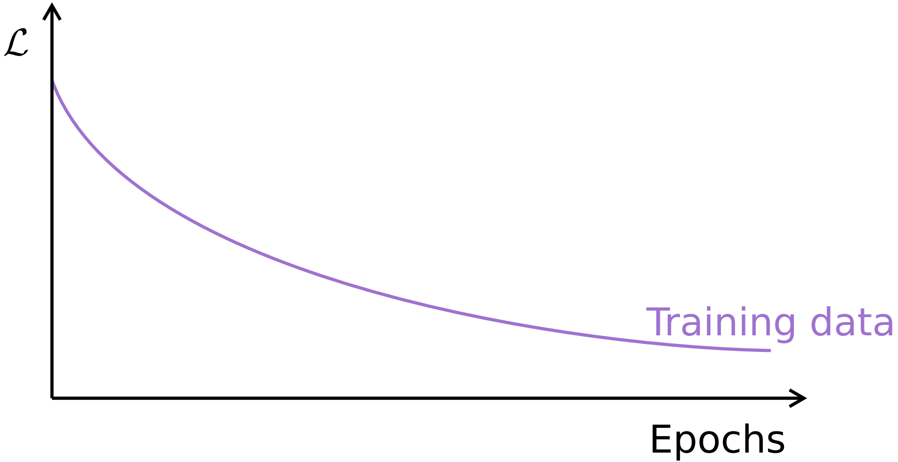
Overfitting

- Neural nets have many parameters
- Easy to overfit

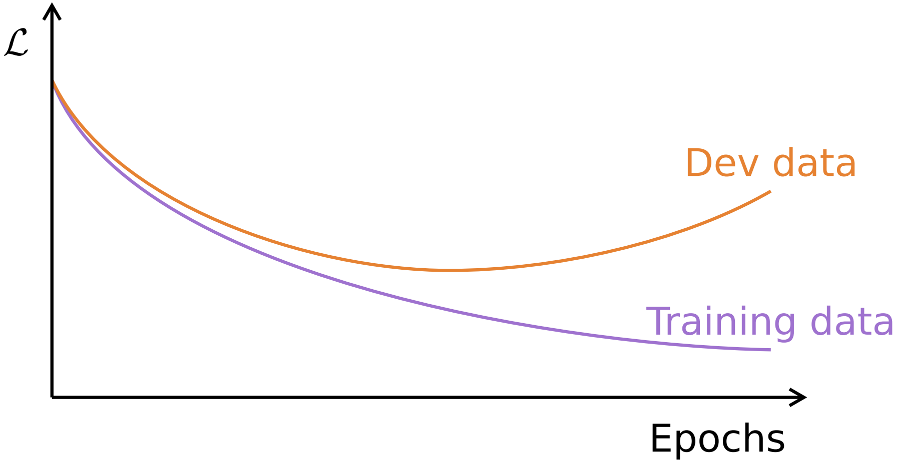
Overfitting

- Neural nets have many parameters
- Easy to overfit
- Some solutions:
  - Early stopping
  - Regularisation
  - Dropout

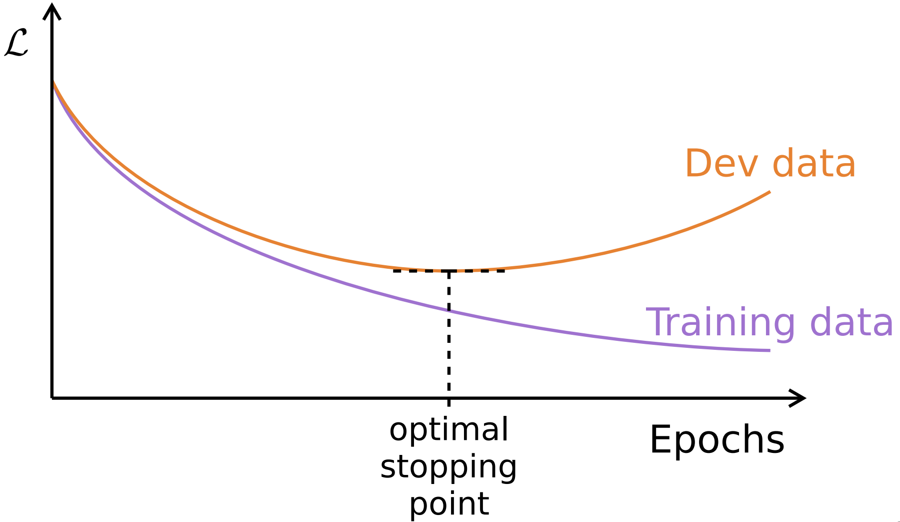
Early stopping



Early stopping



Early stopping



Regularisation

- Penalise “bad” parameters:

$$\mathcal{L} = \mathcal{L}_{\text{err}}(\hat{y}_i, y_i) + \mathcal{L}_{\text{reg}}(\theta)$$

Regularisation

- Penalise “bad” parameters:

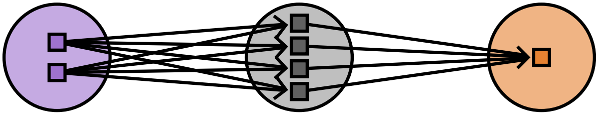
$$\mathcal{L} = \mathcal{L}_{\text{err}}(\hat{y}_i, y_i) + \mathcal{L}_{\text{reg}}(\theta)$$

- For example:

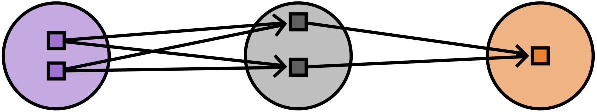
$$\mathcal{L}_1(\theta) = \sum_i |\theta_i|$$

$$\mathcal{L}_2(\theta) = \sum_i |\theta_i|^2$$

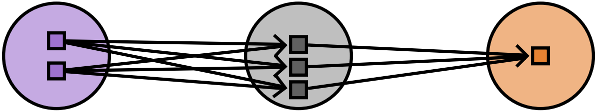
Dropout



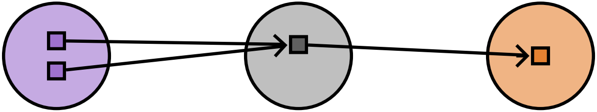
Dropout



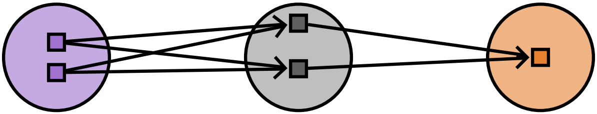
Dropout



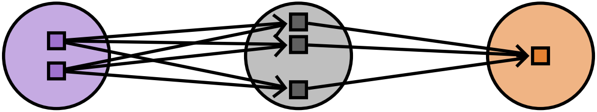
Dropout



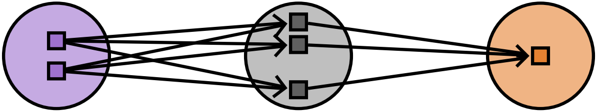
Dropout



Dropout



Dropout



- Less dependent on specific units
- More robust to noise

What we’ve covered

- Neural nets, activation functions
- Architectures: CNNs, RNNs
- Training by gradient descent
- Early stopping, regularisation, dropout