Intro ML
ML. 7.1. Neural nets
Intro and Shallow NNs

DataLab CSIC

Objectives and schedule

Introduce key concepts about neural networks, from shallow to deep. Study some currently important architectures: convolutional, recurrent, transformer, autoencoder, GANS. Sketch some of their major application domains. Introduce KERAS (and TensorFlow). Introduce stochastic gradient descent and variants.

Contents

- Introduction
- (Shallow) neural networks
- Deep neural networks
- Specific architectures

Schedule

App next 3.5 weeks. This week intro and shallow nets.

Today, case by Nuria Campillo (CNB+ICMAT) on Deep NNs for mutagenicity prediction

Bishop 5, CASI 18, Goodfellow et al, Chollet and Allaire (KERAS)

Lab for 7.1 and 7.2

- Basic example with neuralnet to understand concepts
- Comparison SVM-NN (with SVM winning, recal COSS talk)
- 1st example with Keras
- Comparison NN-Elasticnet (with NN winning)

NNs. Motivation

Motivation

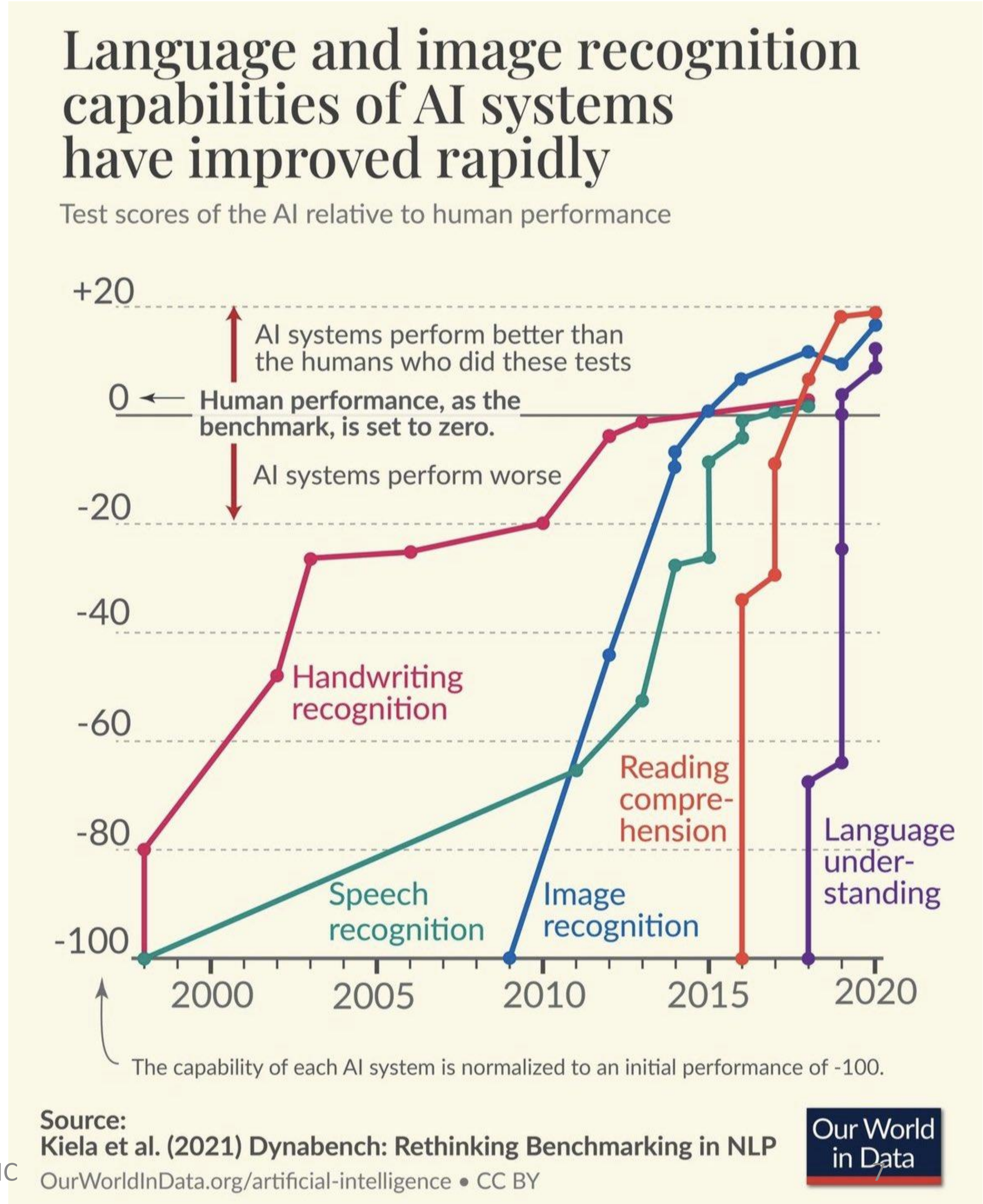
- AI is ultra cool because of deep learning
- ML is very cool because of deep learning
- Stats is pretty cool because of deep learning
- Annex 1 of EU AI Act

- Many exciting research questions
- Many exciting computational problems

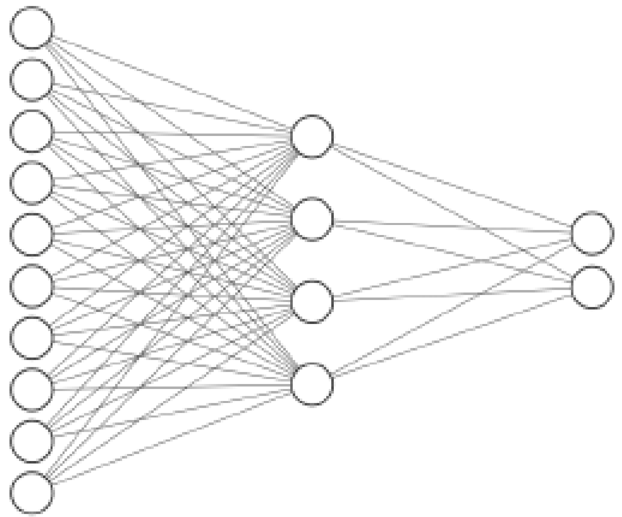
Brief history of NNs

When	What	Why	Why not
End of 50’s, Beg of 60’s	Rosenblatt’s perceptron	Efficiente scheme Good branding	Minsky& Papert (1968)
End of 80’s, Beg of 90’s	Cybenko’s representation Shallow NNs	Good branding Impulse from CS comm	Tech problems (vanishing gradient) Emergence of SVM and others
2010’s on	Deep learning, variants Outstanding aplications	Massive labeled data Rediscovery of SGD GPUs ReLUs et al Domain specific architectures Winning Imagenet comp	

Some benchmarks

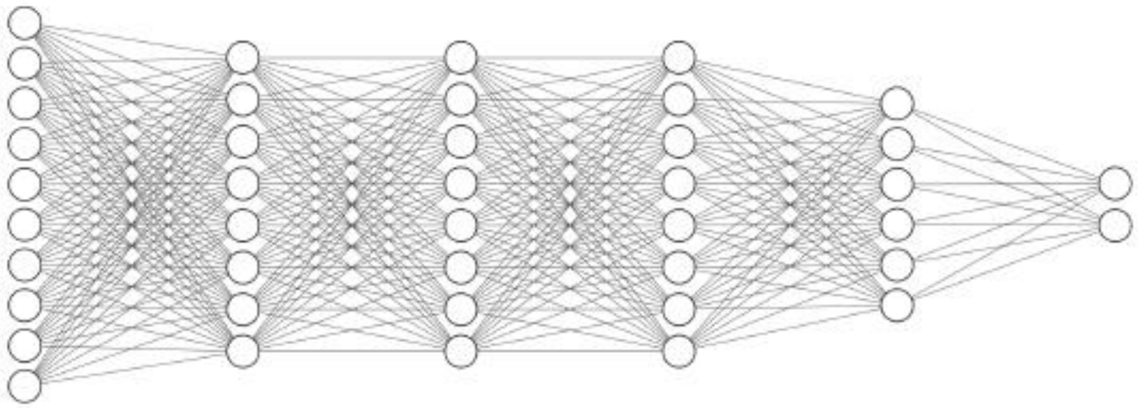


Concept



$$y = \sum_{j=1}^{m} \beta_j \psi(x'\gamma_j) + \epsilon$$
$$\epsilon \sim N(0, \sigma^2),$$
$$\psi(\eta) = \exp(\eta)/(1 + \exp(\eta))$$

(Shallow) Neural nets



$$\{f_0, f_1, ..., f_{L-1}\}$$

$$z_{l+1} = f_l(z_l, \gamma_l).$$

$$y = \sum_{j=1}^{m_L} \beta_j z_{L,j} + \epsilon$$
$$\epsilon \sim N(0, \sigma^2),$$

Deep neural nets

Some nice apps

https://playground.tensorflow.org/

https://www.i-am.ai/neural-numbers.html

https://www.i-am.ai/piano-genie.html

http://places2.csail.mit.edu/

https://modeldepot.github.io/tfjs-yolo-tiny-demo/

Structure

- Perceptron
- Shallow neural nets
- Deep neural nets
- Convolutional neural nets
- Recurrent neural nets (and Transformers)
- Autoencoders (and VAEs)
- GANs

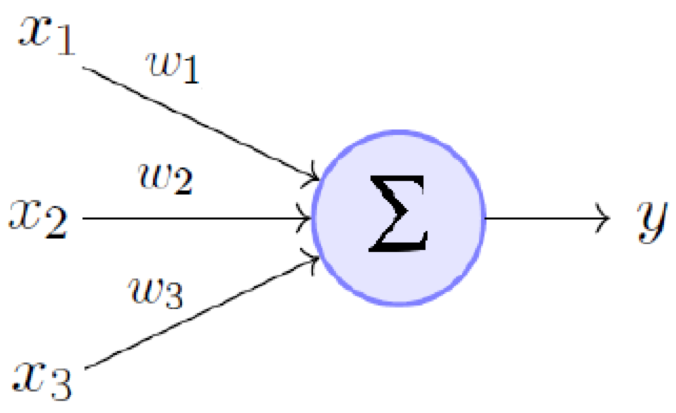
The perceptron

Concept

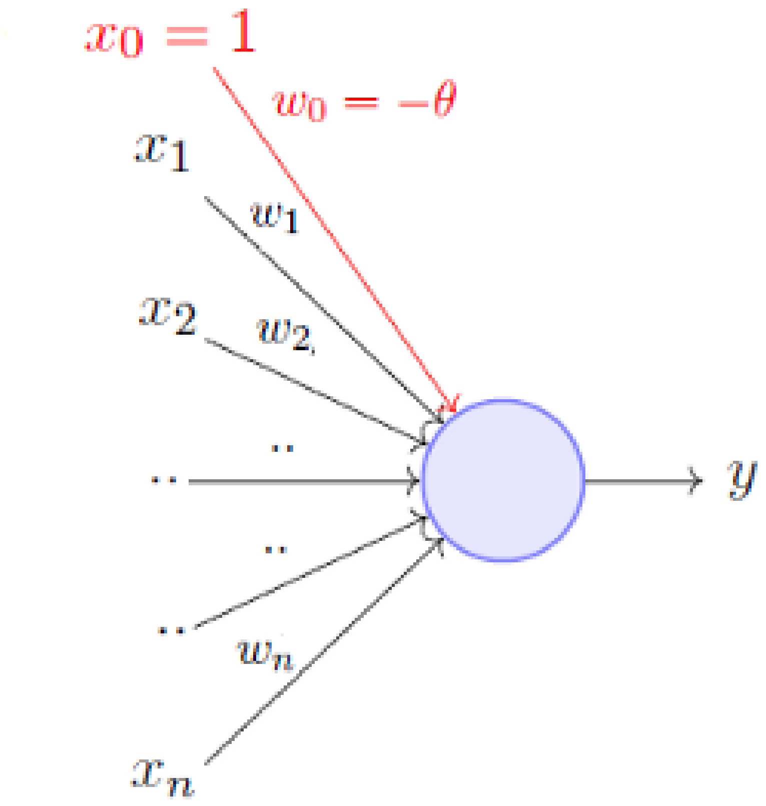
Rosenblatt (1958)

Linear combination of inputs

Nonlinear activation (e.g, step function)



Classifying with a perceptron



A more accepted convention,

$$
\begin{aligned}
y &= 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0 \\
&= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0
\end{aligned}
$$

$$where, \quad x_0 = 1 \quad and \quad w_0 = -\theta$$

Training

Algorithm: Perceptron Learning Algorithm

$P \leftarrow inputs \quad with \quad label \quad 1;$

$N \leftarrow inputs \quad with \quad label \quad 0;$

Initialize $\mathbf{w}$ randomly;

while $!convergence$ **do**

  Pick random $\mathbf{x} \in P \cup N$;

  if $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ **then**

    $\mathbf{w} = \mathbf{w} + \mathbf{x}$;

  end

  if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**

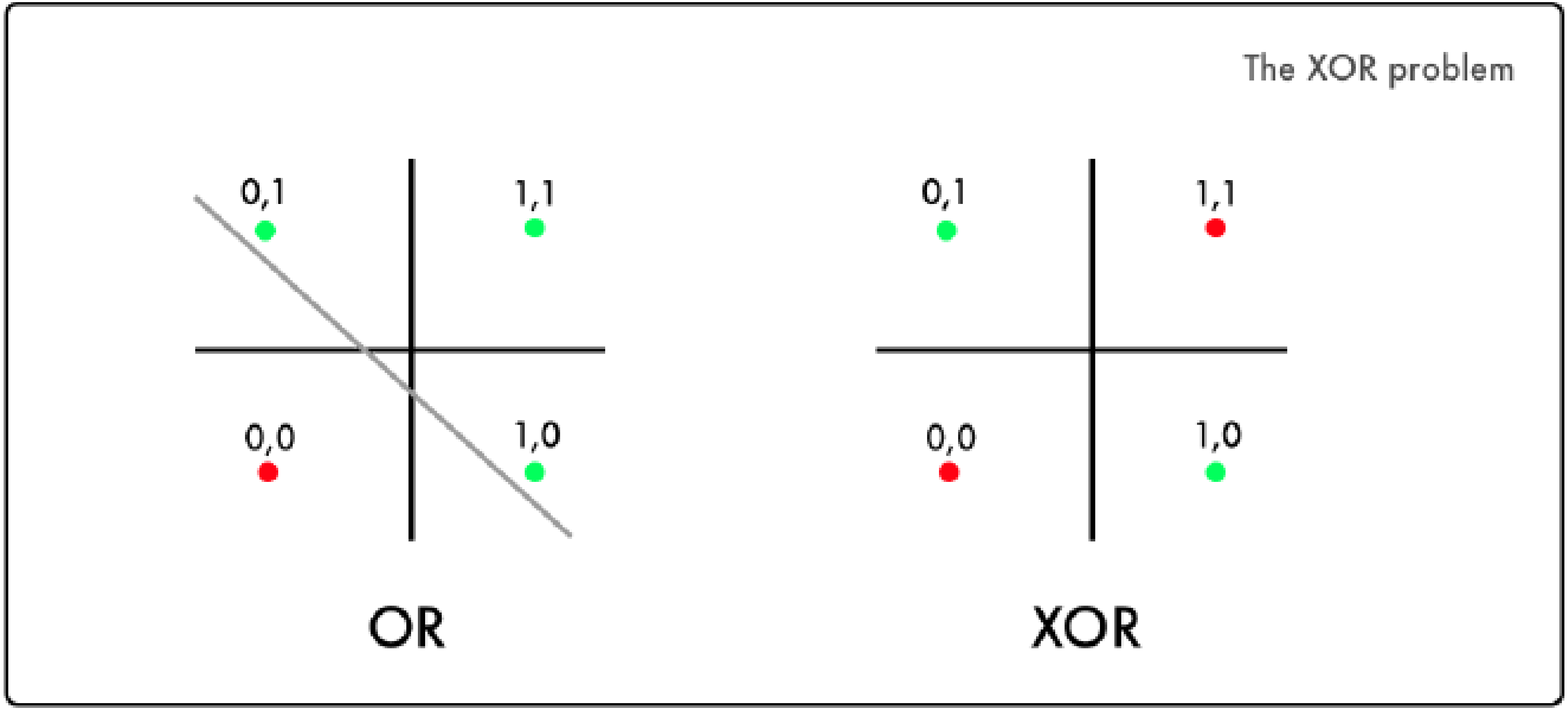
    $\mathbf{w} = \mathbf{w} - \mathbf{x}$;

  end

end

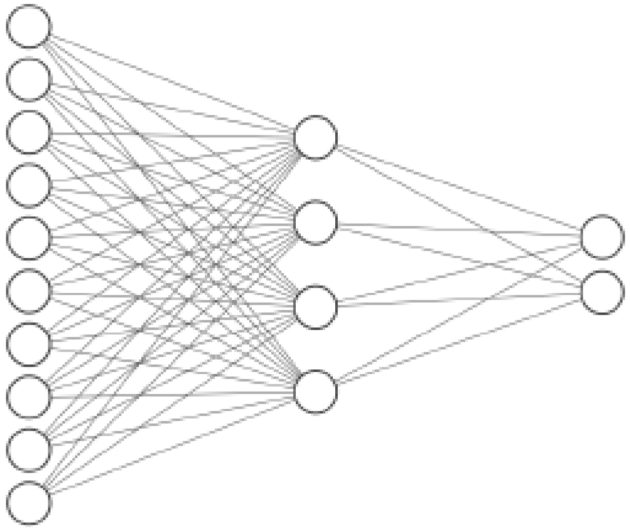
//the algorithm converges when all the inputs are classified correctly

A shortcoming



Shallow neural nets

Formulation



$$
\begin{aligned}
y &= \sum_{j=1}^{m} \beta_j \psi(x'\gamma_j) + \epsilon \\
&\epsilon \sim N(0, \sigma^2), \\
&\psi(\eta) = \exp(\eta)/(1 + \exp(\eta))
\end{aligned}
$$

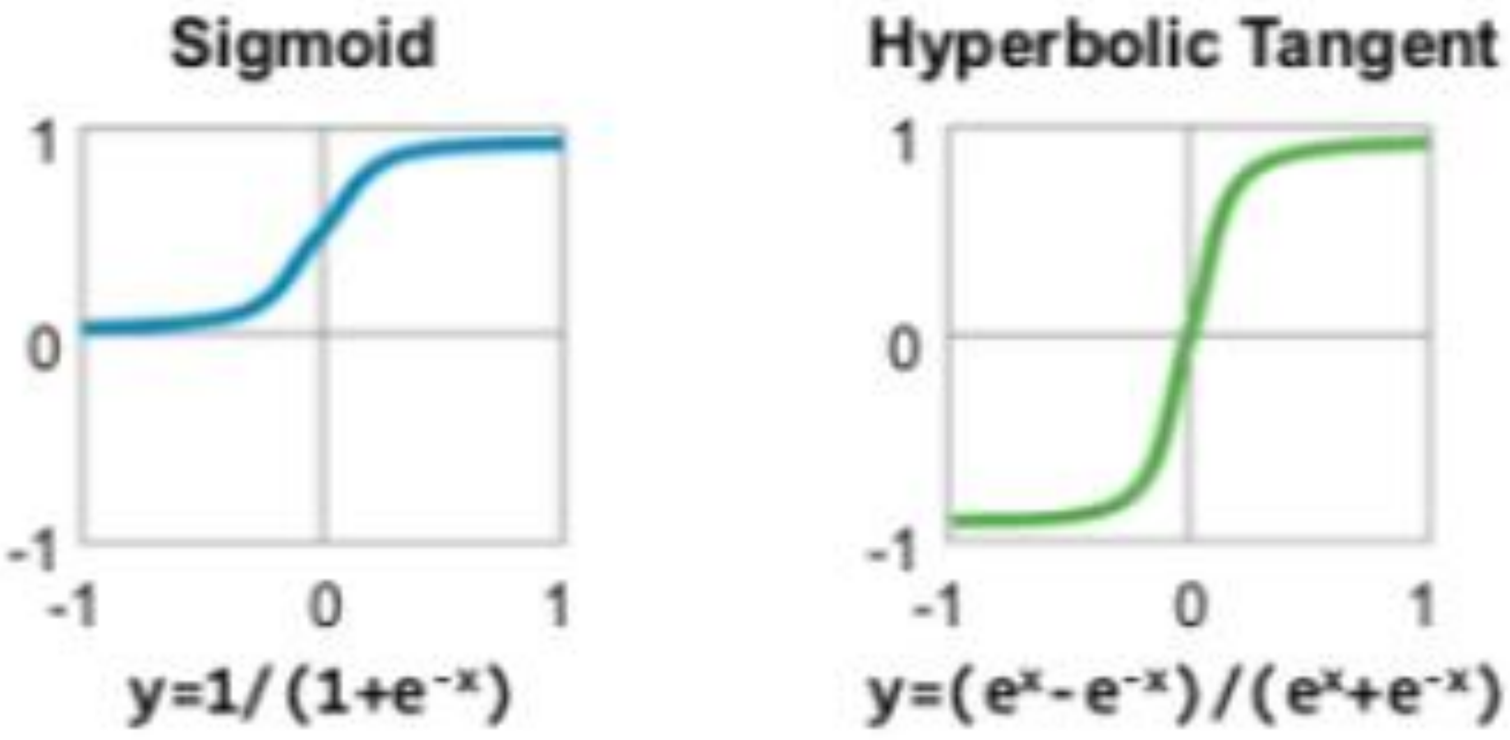
Linear in beta's, nonlinear in gamma's

Motivation. Cybenko's theorem

Any continuous function in the r-dimensional cube may be approximated by models of type $\sum_{j=1}^{m} \beta_j \psi(x'\gamma_j)$ when the activation function is sigmoidal

(as m goes to infty)

Early activation functions



Training

Given training data, maximise log-likelihood

$$\min_{\beta,\gamma} f(\beta,\gamma) = \sum_{i=1}^{n} f_i(\beta,\gamma) = \sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{m}\beta_j \psi(x_i'\gamma_j)\right)^2$$

Gradient descent

Backpropagation to estimate gradient

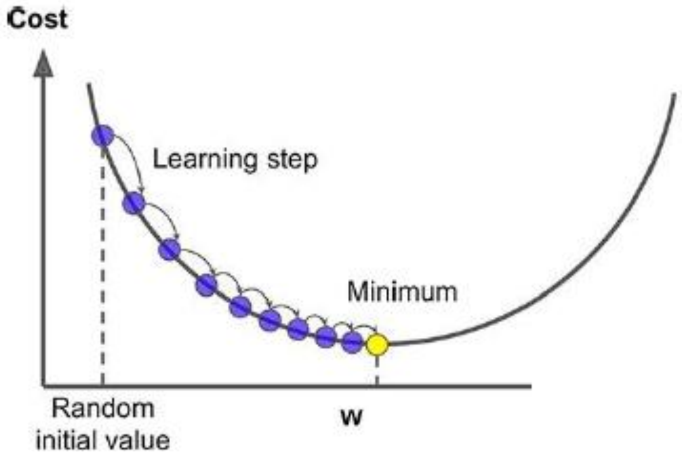
Training

Given training data, maximise log-likelihood

$$\min_{\beta,\gamma} f(\beta,\gamma) = \sum_{i=1}^{n} f_i(\beta,\gamma) = \sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{m}\beta_j\psi(x_i'\gamma_j)\right)^2$$

Gradient descent

Backpropagation to estimate gradient



Training

Given training data, maximise log-likelihood

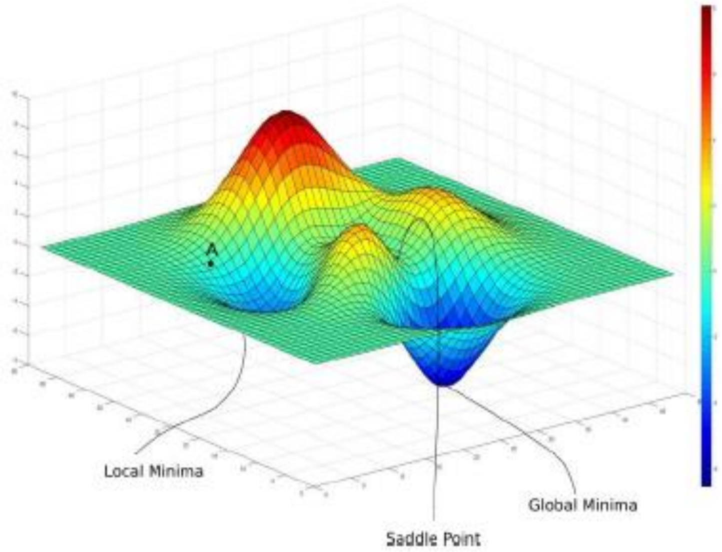
$$\min_{\beta,\gamma} f(\beta,\gamma) = \sum_{i=1}^{n} f_i(\beta,\gamma) = \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{m} \beta_j \psi(x_i' \gamma_j) \right)^2$$

Gradient descent

Backpropagation to estimate gradient

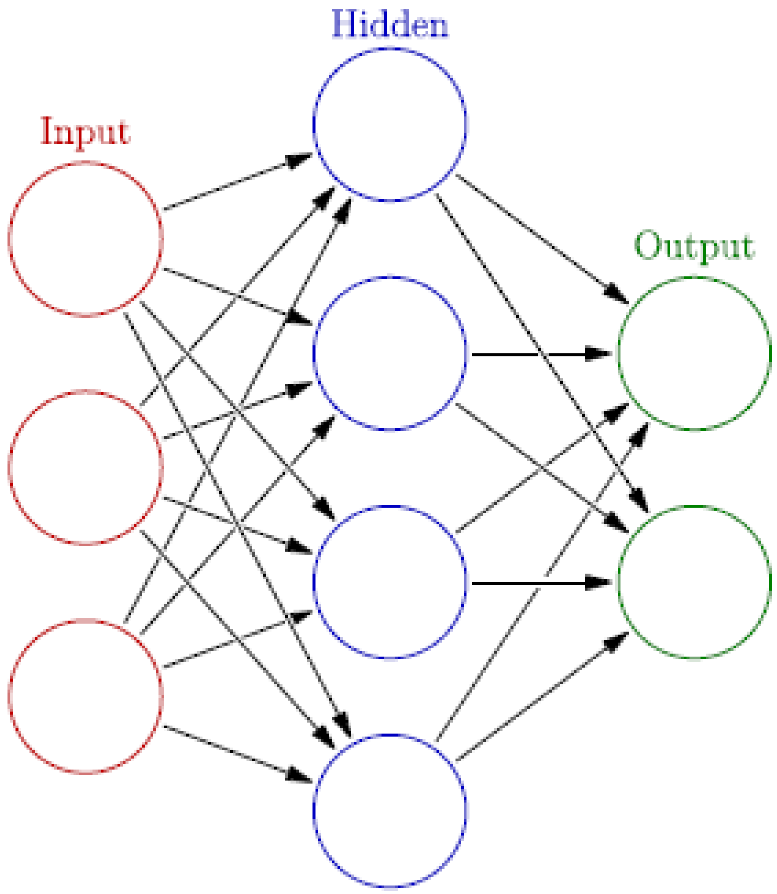
Training. Multiple local optima

- Node relabeling
- Inherent due to non-linearity, non-convexity
- Node duplicity



Look for a (hopefully good) local minimum

Example with Keras




```r
library(keras)

model <- keras_model_sequential()

# arquitectura
model %>%
  layer_dense(units = 4,
              activation = 'sigmoid',
              input_shape = c(3)) %>%
  layer_dense(units = 2,
              activation = 'linear')

# definir entrenamiento
model %>% compile(loss = "mse",
                  optimizer = optimizer_sgd())

# entrenamiento
model %>% fit(X_train, y_train,
              epochs = 10, batch_size = 128,
              validation_size = 0.2)

# error de test
model %>% evaluate(X_test)
```

Training with regularisation

$$\min_{\beta,\gamma} f(\beta,\gamma) = \sum_{i=1}^{n} f_i(\beta,\gamma) = \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{m} \beta_j \psi(x_i' \gamma_j) \right)^2$$

$$\min g(\beta,\gamma) = f(\beta,\gamma) + h(\beta,\gamma),$$

Weight decay

$$h(\beta,\gamma) = \lambda_1 \sum \beta_i^2 + \lambda_2 \sum \sum \gamma_{ji}^2$$

Ridge

Gradient descent

$$(\beta,\gamma)_{k+1} = (\beta,\gamma)_k - \eta\nabla g((\beta,\gamma)_k)$$

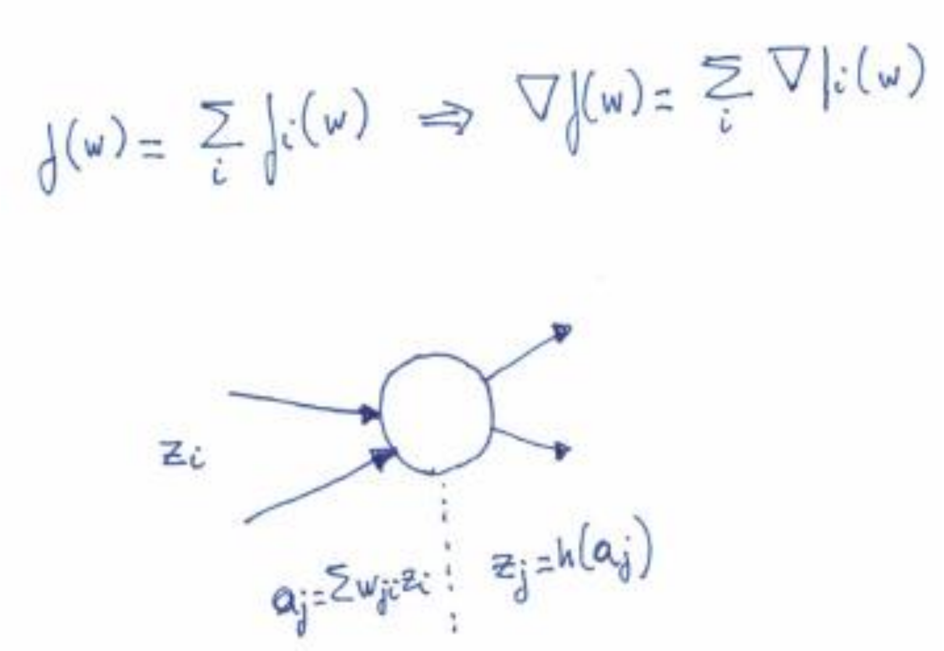
$$\nabla g((\beta,\gamma)) = \sum_{i=1}^{n} \nabla f_i(\beta,\gamma) + \nabla h(\beta,\gamma)$$

$$(\nabla_\beta f_i)_k = -2\left(y_i - \sum_{j=1}^{m} \beta_j \psi(x_i'\gamma_j)\right)\psi(x_i'\gamma_k)$$

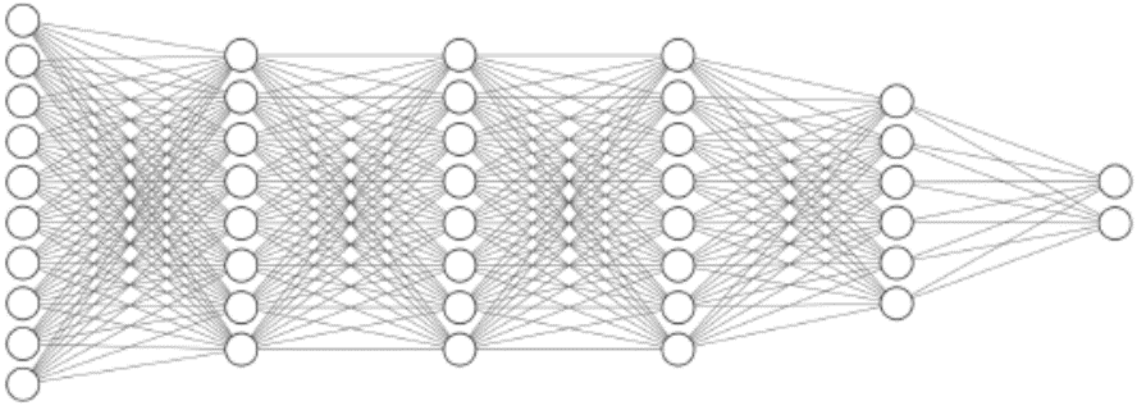
$$(\nabla_\gamma f_i)_{k,l} = -2\left(y_i - \sum_{j=1}^{m} \beta_j \psi(x_i'\gamma_j)\right)\beta_l\psi(x_i'\gamma_l)(1-\psi(x_i'\gamma_l))x_k$$

$$(\nabla_\beta h)_k = 2\lambda_1\beta_k \qquad (\nabla_\gamma h)_{k,l} = 2\lambda_2\gamma_{k,l}.$$

Backpropagation. Forward pass



In a ***forward*** pass, each node accumulates its input and output



Backpropagation. Backward pass I

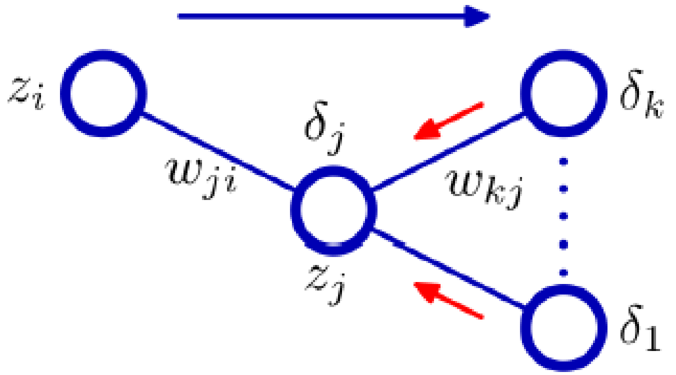
$$\frac{\partial J_n}{\partial w_{ji}} = \frac{\partial J_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} := \delta_j z_i$$

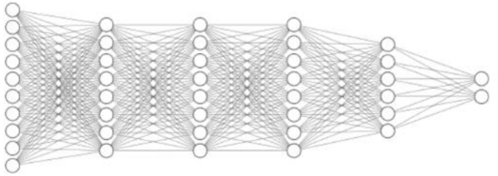
$$\delta_j = \frac{\partial J_n}{\partial a_j}$$

$$\delta_j = \frac{\partial J_n}{\partial a_j} = \sum_k \frac{\partial J_n}{\partial a_k}\frac{\partial a_k}{\partial a_j}$$

$$\delta_j = h'(a_j)\sum_k w_{kj}\,\delta_k$$

In a hidden node f_n is function of aj only through ak of following layer





Computation trivial for output layer

Backpropagation (CASI 18, care with notation)

Algorithm 18.1 BACKPROPAGATION

1 Given a pair x, y, perform a "feedforward pass," computing the activations $a_\ell^{(k)}$ at each of the layers $L_2, L_3, \ldots, L_K$; i.e. compute $f(x; \mathcal{W})$ at x using the current $\mathcal{W}$, saving each of the intermediary quantities along the way.

2 For each output unit ℓ in layer L_K, compute

$$\begin{aligned}\delta_\ell^{(K)} &= \frac{\partial L[y, f(x, \mathcal{W})]}{\partial z_\ell^{(K)}} \\ &= \frac{\partial L[y, f(x; \mathcal{W})]}{\partial a_\ell^{(K)}} \dot{g}^{(K)}(z_\ell^{(K)}),\end{aligned} \tag{18.10}$$

where $\dot{g}$ denotes the derivative of $g(z)$ wrt z. For example for $L(y, f) = \frac{1}{2}\|y - f\|_2^2$, (18.10) becomes $-(y_\ell - f_\ell) \cdot \dot{g}^{(K)}(z_\ell^{(K)})$.

3 For layers $k = K-1,\ K-2, \ldots, 2$, and for each node ℓ in layer k, set

$$\delta_\ell^{(k)} = \left(\sum_{j=1}^{p_{k+1}} w_{j\ell}^{(k)} \delta_j^{(k+1)} \right) \dot{g}^{(k)}(z_\ell^{(k)}). \tag{18.11}$$

4 The partial derivatives are given by

$$\frac{\partial L[y, f(x; \mathcal{W})]}{\partial w_{\ell j}^{(k)}} = a_j^{(k)} \delta_\ell^{(k+1)}. \tag{18.12}$$

Backpropagation efficiency

Complexity is O(w) , w number of parameters

Sounds nice… but recall number of parameters can be pretty big…

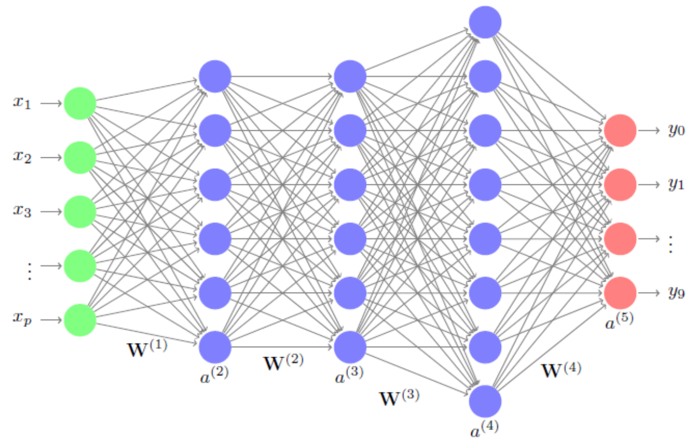
More later when talking about automatic differentiation

So how big is pretty big???

Images 28x28

784 inputs
1024
1024
2048
9 outputs

4M parameters

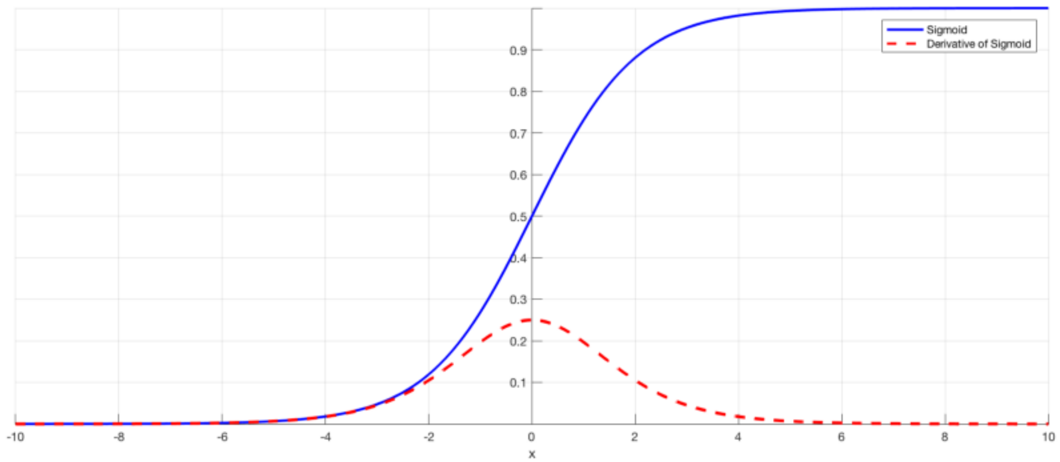


Vanishing gradient problem

- When using sigmoid activation functions, as the derivative is in (0,1), if we pile up several layers derivatives rapidly vanish, eventually, blocking training,…

$$\psi(z)' = \psi(z)(1-\psi(z))$$

$$\psi(z) = \frac{1}{1+e^{-z}}$$

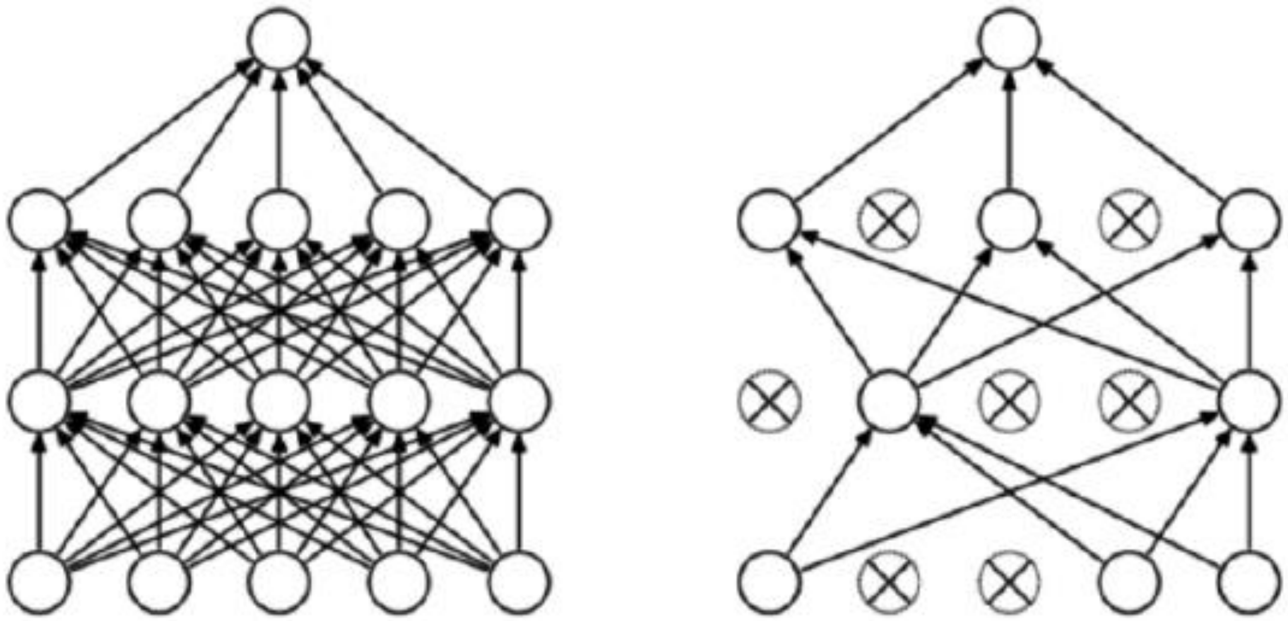


Regularisation II. Early stopping

- Training is iterative
- Preserve validation set. After each iteration, compute validation error.
- Typically, validation error reduces and then grows (due to overfitting)
- Stop before this happens-→ Early stopping
- May reduce network complexity

Regularisation III. Dropout

- At each iteration, each neuron is switched off with probability 1-p



Bayesian analysis of shallow neural nets

Recall MLE+regularisation --→ MAP !!!!

Bayesian analysis of shallow neural nets (fixed arch)

$$y = \sum_{j=1}^{m} \beta_j \psi(x'\gamma_j) + \epsilon$$

$$\epsilon \sim N(0, \sigma^2),$$

$$\psi(\eta) = \exp(\eta)/(1 + \exp(\eta))$$

$$\beta_i \sim N(\mu_\beta, \sigma_\beta^2) \text{ and } \gamma_i \sim N(\mu_\gamma, S_\gamma^2)$$

$$\mu_\beta \sim N(a_\beta, A_\beta),\ \mu_\gamma \sim N(a_\gamma, A_\gamma),\ \sigma_\beta^{-2} \sim \quad Gamma(c_b/2, c_b C_b/2)$$

$$S_\gamma^{-1} \sim Wish(c_\gamma, (c_\gamma C_\gamma)^{-1}) \text{ and } \sigma^{-2} \sim Gamma(s/2, sS/2)$$

Bayesian analysis of shallow neural nets (fixed arch)

Start with arbitrary (β, γ, ν).
while *not convergence* **do**
  Given current (γ, ν), draw β from $p(\beta|\gamma, \nu, y)$ (a multivariate normal).
  for $j = 1, \ldots, m$, *marginalizing in* β *and given* ν **do**
    Generate a candidate $\tilde{\gamma}_j \sim g_j(\gamma_j)$.
    Compute $a(\gamma_j, \tilde{\gamma}_j) = \min\left(1, \frac{p(D|\tilde{\gamma}, \nu)}{p(D|\gamma, \nu)}\right)$ with $\tilde{\gamma} = (\gamma_1, \gamma_2, \ldots, \tilde{\gamma}_i, \ldots, \gamma_m)$.
    With probability $a(\gamma_j, \tilde{\gamma}_j)$ replace γ_j by $\tilde{\gamma}_j$. If not, preserve γ_j.
  end
  Given β and γ, replace ν based on their posterior conditionals:
  $p(\mu_\beta|\beta, \sigma_\beta)$ is normal; $p(\mu_\gamma|\gamma, S_\gamma)$, multivariate normal; $p(\sigma_\beta^{-2}|\beta, \mu_\beta)$, Gamma; $p(S_\gamma^{-1}|\gamma, \mu_\gamma)$, Wishart; $p(\sigma^{-2}|\beta, \gamma, y)$, Gamma.
end

Bayesian analysis of shallow neural nets (var arch)

$$
\begin{aligned}
y &= x_i' a + \sum_{j=1}^{m^*} d_j \beta_j \psi(x' \gamma_j) + \epsilon \\
& \epsilon \sim N(0, \sigma^2), \\
& \psi(\eta) = \exp(\eta)/(1 + \exp(\eta)), \\
Pr(d_j = k | d_{j-1} = 1) &= (1-\alpha)^{1-k} \times \alpha^k, k \in \{0, 1\}
\end{aligned}
$$

$$
\beta_i \sim N(\mu_b, \sigma_\beta^2),\ a \sim N(\mu_a, \sigma_a^2),\ \gamma_i \sim N(\mu_\gamma, \Sigma_\gamma).
$$

Reversible jump algo

NNs in other contexts

Classification

Use a multinomial likelihood

$$p(y|x, \beta, \gamma) = Mult(n = 1, p_1(x, \beta, \gamma), \ldots, p_K(x, \beta, \gamma)),$$

Use softmax to compute class probabilities

$$p_k = \frac{\exp\{\beta_k \psi(x' \gamma_k)\}}{\exp\left\{\sum_{k=1}^{K} \beta_k \psi(x' \gamma_k)\right\}}$$

Other

Non-linear autoregression

Semi-parametric regression

(Gaussian process)

$$y = \sum_{j=1}^{m} \beta_j \psi(x'\gamma_j) + \epsilon$$

$$\epsilon \sim N(0, \sigma^2),$$

$$\psi(\eta) = \exp(\eta)/(1 + \exp(\eta))$$

Final comments

If n is large, $$\nabla g((\beta,\gamma)) = \sum_{i=1}^{n} \nabla f_i(\beta,\gamma) + \nabla h(\beta,\gamma)$$

If more than 1 hidden layer, VG exacerbates

If more than hidden layer, backprop chains get longer….

Seems we are not in good shape for DL….

If n is large, $$\nabla g((\beta,\gamma)) = \sum_{i=1}^{n} \nabla f_i(\beta,\gamma) + \nabla h(\beta,\gamma)$$

If more than 1 hidden layer, VG exacerbates

If more than hidden layer, backprop chains get longer….

Seems we are not in good shape for DL….