Feed-forward Neural Networks

A neural network is a 2-stage regression (or classification) model

[A deep network has more than 2 stages]

This is typically represented by a network diagram

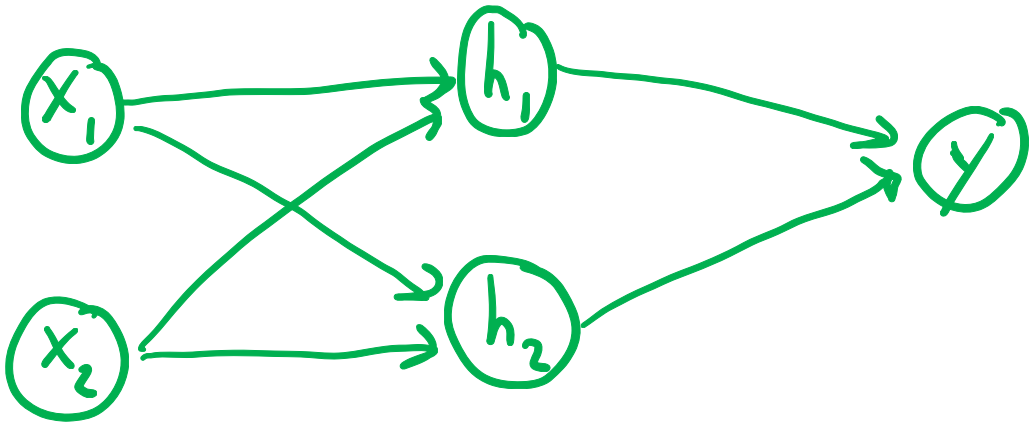
Recall: Parametric models are of the form $f(\cdot, \beta)$

Neural networks: $f(x, \beta) = f^{(2)}(f^{(1)}(x, \beta^{(1)}), \beta^{(2)})$

The generality of this form can make interpretation of the model hard

Terminology: "feed-forward" because information flows forward from x to y (through intermediate / hidden layers)

ex:

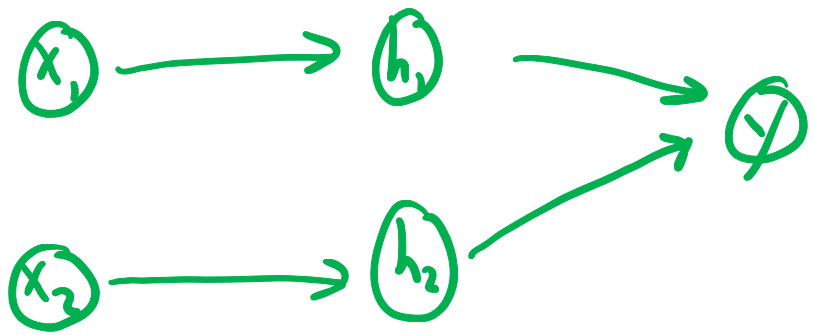


Mathematically:

$$y \approx f(x_1, x_2, \beta)$$
$$= f^{(2)}(h_1, h_2, \beta^{(2)})$$
$$h_1 = f_1^{(1)}(x_1, x_2, \beta_1^{(1)})$$
$$h_2 = f_2^{(1)}(x_1, x_2, \beta_2^{(1)})$$

$$\hookrightarrow y \approx f^{(2)}\left(f_1^{(1)}(x_1, x_2, \beta_1^{(1)}), f_2^{(1)}(x_1, x_2, \beta_2^{(1)}), \beta^{(2)}\right)$$

Question: What is the mathematical form for:



Answer: $y \approx f^{(2)}\left(f_1^{(1)}(x_1, \beta^{(1)}),\ f_2^{(1)}(x_2, \beta_2^{(1)}),\ \beta^{(2)} \right)$

h_1 does not depend on x_2

h_2 does not depend on x_1

Activation functions

Typically the regression functions $f_i^{(k)}$ are a nonlinear function (activation function) applied to a linear mapping

Take $\beta_i^{(k)} = (W^{ik}, b^{ik})$ are weights and bias

$\hookrightarrow f_i^{(k)}(x, \beta_i^{(k)}) = g_k(W^{ik}x + b^{ik})$ for activation function g_k

Common Activation Functions

1) ReLU: Rectified linear unit: $g(x) = \max(x, 0)$ 

2) Tanh: Hyperbolic tangent: $g(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ 

3) Linear: (As final activation only): $g(x) = x$ 

4) Sigmoid: Logistic function: $g(x) = (1 + e^{-x})^{-1}$ 

Hyperparameters

1) The shape of the network
   a) How many hidden layers are there? K
   b) How many nodes are in each layer? p_k for layer k
   c) How are these layers connected?
2) The form of the regression functions $f_i^{(k)}$
   ↳ Choice of activation functions
3) Parameter constraints (ex: $\beta \geq 0$) ← Rare

# of parameters: (dense network with activation function construction)

$$\sum_{k=0}^{K-1} (p_k + 1) p_{k+1}$$ where p_0 = # of predictors

This can be a very large number

Universal Approximation Theorem

Consider input space $\mathbb{X} \subseteq \mathbb{R}^p$ compact (all predictors x lie in $\mathbb{X}$)

Let $f^*: \mathbb{X} \to \mathbb{R}$ be the true relationship

↪ Assume f^* is continuous

Thm For every $\varepsilon > 0$, there exists a (shallow) neural network f that approximates f^* with error ε if the hidden layer activation function g_1 is not a polynomial

$$\inf_{\beta} \sup_{x \in \mathbb{X}} |f^*(x) - f(x, \beta)| < \varepsilon$$

Note:

- This theorem only says there exists a network large enough to achieve the desired degree of accuracy

  It does not say how large the network will be

- The error bound assumes we can perfectly calibrate the parameters (weights & biases) $\beta = (W, b)$

Fitting Neural Networks

Given the hyperparameters, we need to determine all of the weights W and biases b

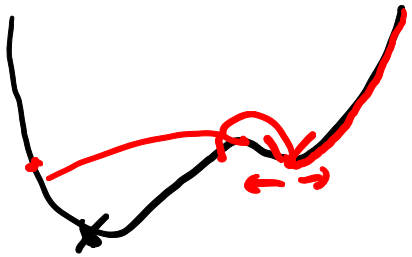
This is (often) accomplished through gradient descent

Briefly: $\beta_m = \beta_{m-1} - \gamma_m \nabla_\beta L(y, f(X, \beta_{m-1}))$

- β_m: new estimate
- β_{m-1}: old estimate
- γ_m: step length (multiple approaches)
- $\nabla_\beta L(y, f(X, \beta_{m-1}))$: gradient of the loss function at the old estimate

ex: $L(y, f(X, \beta)) = \sum_{i=1}^{N} (y_i - f(x_i, \beta))^2$

- "smart" selection of step length will converge to a (local) minimum



- Computing the gradient can be expensive

Issues in Training Neural Networks

- **Starting values**: The gradient descent requires starting values β_0
  - Where you start can impact the fitted parameters β^*
  - You do not know if you are at a local or global minimum
  - ↳ Generally, try **random** initial values near zero (but not equal to 0)
    - ↳ near 0, the neural network is close to a linear fit

- **Overfitting**: Because of the high number of parameters, neural networks can overfit
  - One approach is to use **weight decay** (analogous to ridge or LASSO)
  - ↳ add a penalty to the loss: $L(y, f(X, \beta)) + \lambda J(\beta)$ for $\lambda \geq 0$

- **Scaling of inputs**: If different predictors have different orders of magnitude the results of fitting can be off (give undue weight to larger values)
  - ↳ Often normalize so all predictors have mean 0 & standard dev. 1

- **Choosing hyperparameters**:
  Often determined by background knowledge of data and/or cross-validation

- **Multiple minima**: You may get stuck at a local minimum of the error
  - ↳ One solution is to use bagging or bumping of neural networks

[**Bagging**: $\hat{f}(x) = \frac{1}{B} \sum_{b=1}^{B} f(x, \beta_b)$

Boosting: Instead of the average, reweight the models in order to improve the fit]

Example Financial Applications

1) Financial market predictions

Using a fixed length memory of the market (& other factor returns ...)

try to predict future returns/values or direction of movement

Generalizes linear autoregressive / factor models

But lose interpretability

2) Pricing & hedging derivatives

Longstaff-Schwartz (2001) proposed a regression-based approach for pricing path-dependent options (ex: American options)

↳ called least squares Monte Carlo

ex: American option $V_0 = \sup_\tau \mathbb{E}^Q[e^{-r\tau}(S_\tau - K)^+]$

Discrete time: $t = 0, \Delta t, \ldots, T$

↳ $V_T(x) = (x-K)^+$

$$V_{t-\Delta t}(x) = \max\left\{(x-K)^+,\ \underbrace{\mathbb{E}^Q\left[e^{-r\Delta t} V_t(S_t) \mid S_{t-\Delta t} = x\right]}_{\text{needs to be approximated}}\right\}$$

Idea: Regress this value with data $(S^n_{t-\Delta t}, V_t(S^n_t))$ from paths from Monte Carlo

3) Credit / ESG Rating Prediction

Many bonds are not rated or costly to get the rating

Using known predictors (eg: balance sheet information) and fitted to true ratings, we want to predict new scores

This is a classification problem

↳ With so many possible predictors, model complexity + overfitting can be a problem

↳ feature selection can be an important component