Artificial Neural Networks

Jeffrey Mei

ANN Accomplishments

- Atari Games (2013)
- AlphaGo (2015)
- AlphaStar: Starcraft II (2019)
- Open AI Five: Dota II (2019)
- Midjourney: AI Art (2022)
- ChatGPT: advanced chatbot (2022)

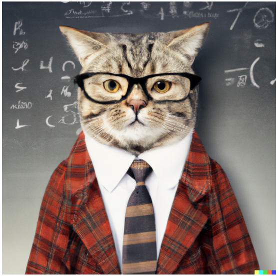


Figure: AI generated image in Dall-E

History

- Perceptrons: first models resembling ANNs (Rosenblatt, 1957)
  - algorithm that finds separating hyperplane
- AI Winter: perceptrons cannot model XOR (Minsky, 1969)
  - no linear separation possible

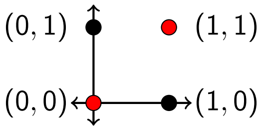


Figure: XOR circuit

History

- multilayer perceptron models (ANNs) able to solve XOR issue
- back-propagation led to computationally feasible solutions (Rumelhart, Hinton; 1986)
- improved computing power in the 1990s
  - more complex models
  - more data
- ANNs have been popular ever since

Why are ANNs Successful?

- automatic feature engineering
- flexible models
  - faster computers
  - bountiful data
  - universal approximation property

Projection Pursuit

Linear Regression: function must be linear

$$f(X) = X\beta + \varepsilon \tag{1}$$

Nonparametric Additive Models: handles nonlinearities

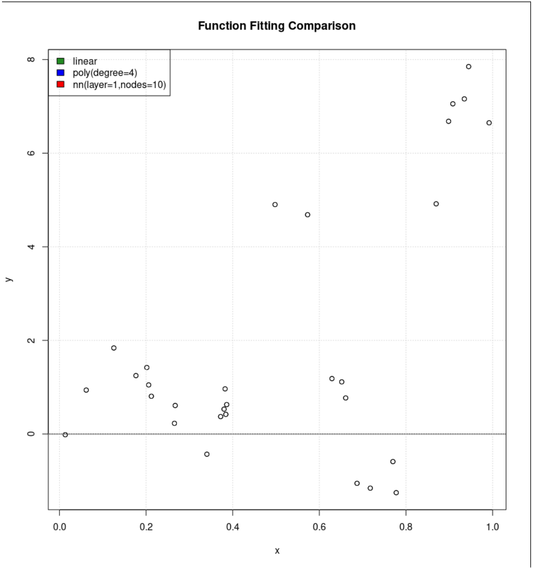
$$f(X) = \sum_{m=1}^{p} g_m(X_m) + \varepsilon \tag{2}$$

Projection Pursuit:

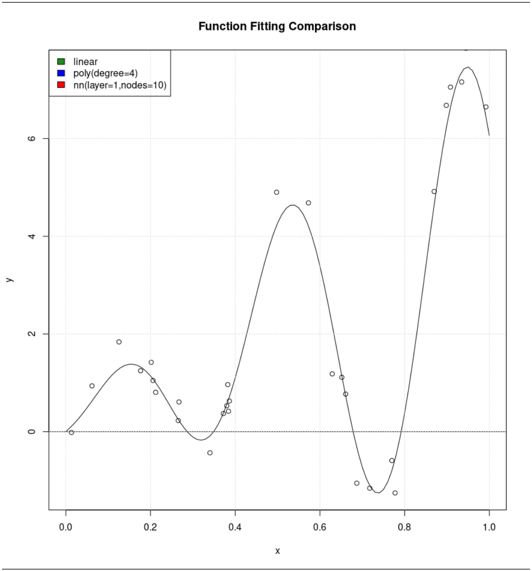
$$f(X) = \sum_{m=1}^{M} g_m(w_m^T X) + \varepsilon \tag{3}$$

- developed in nonparametric statistics literature (Huber, 1985)
- generalization of one-layer ANN
- possesses universal approximation property

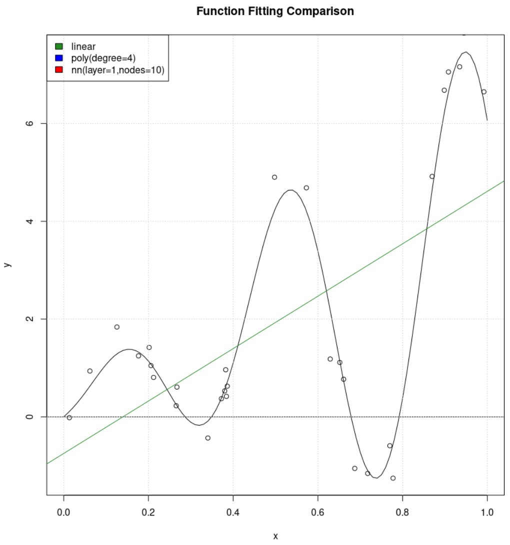
Universal Approximation Property



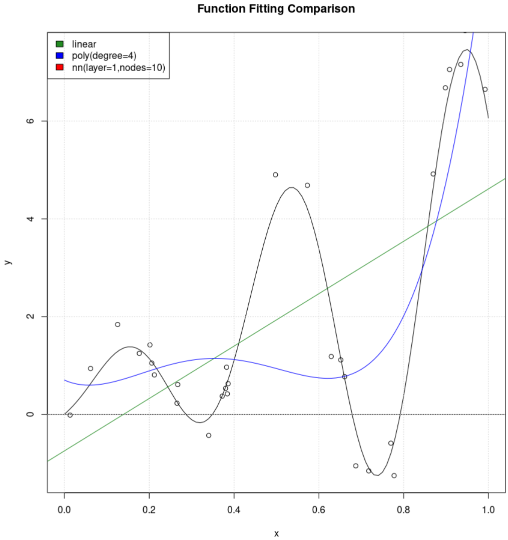
Universal Approximation Property



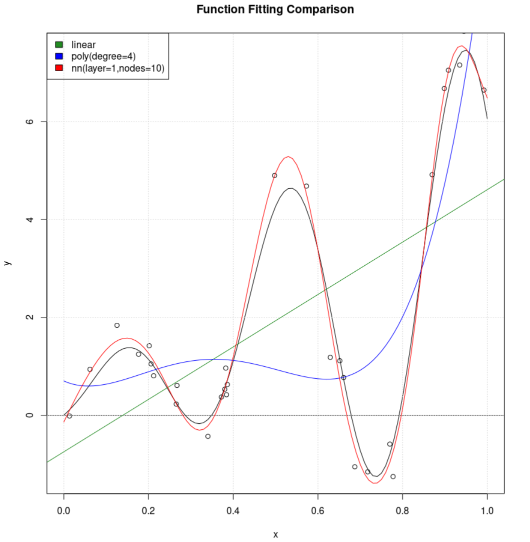
Universal Approximation Property



Universal Approximation Property



Universal Approximation Property



Sigmoid Function

Sigmoid Function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \qquad (4)$$

Softmax Function

$$g_k(x) = \frac{e^x}{\sum_{j=1}^{K} e^x} \qquad (5)$$

- sigmoid function is logistic regression
- softmax is multinomial logistic regression

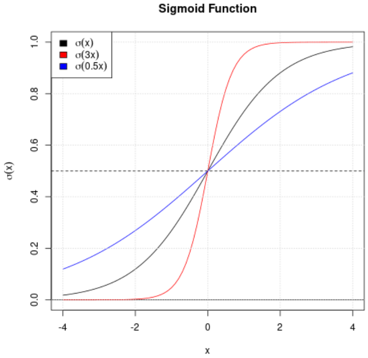
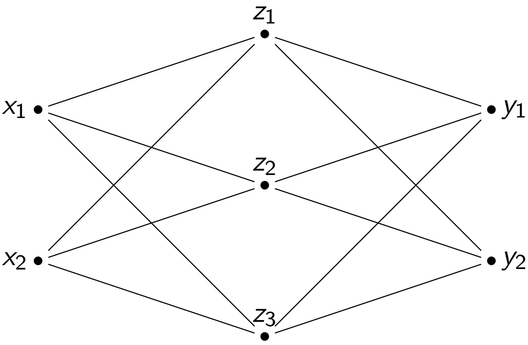
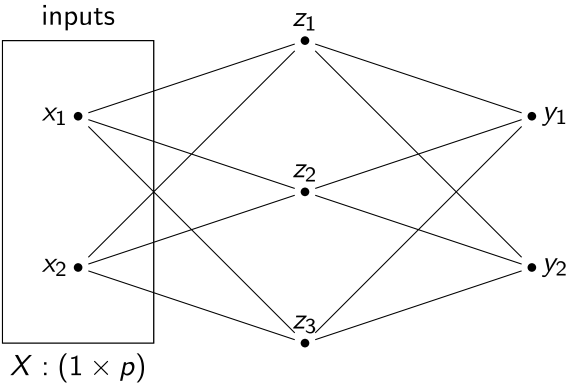


Figure: Sigmoid Function

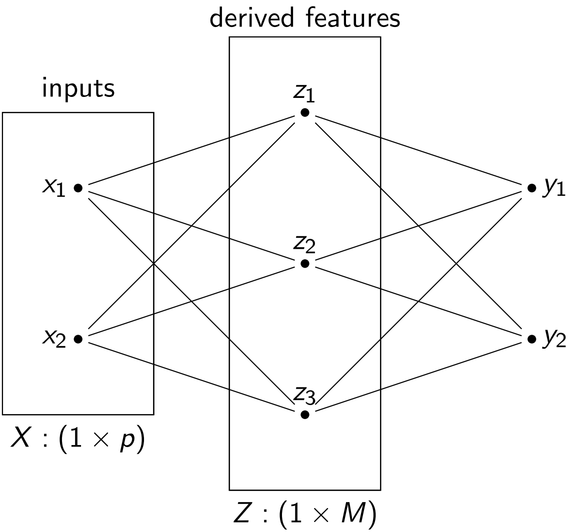
Network Diagram: ANN



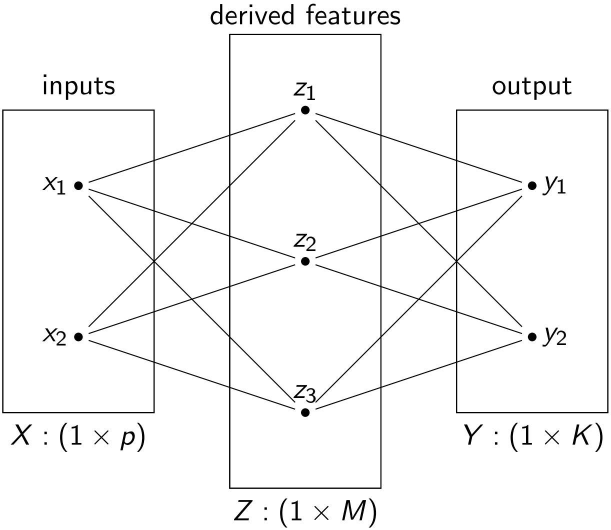
Network Diagram: ANN



Network Diagram: ANN

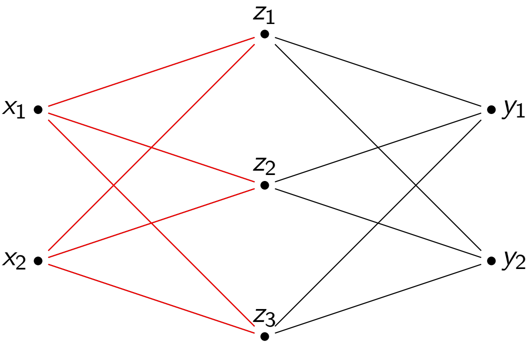


Network Diagram: ANN

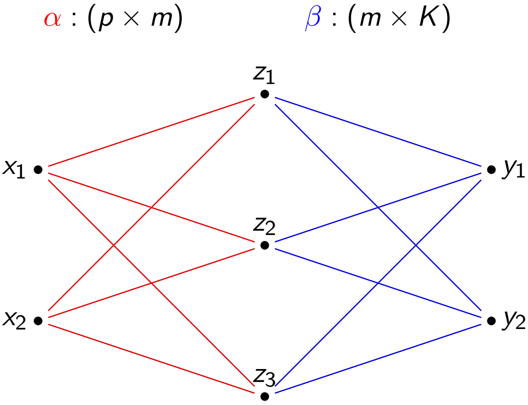


Network Diagram: ANN

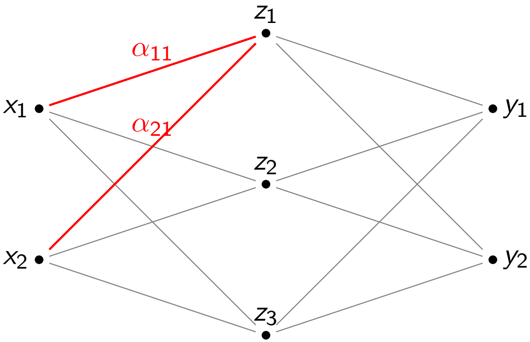
$\alpha : (p \times m)$



Network Diagram: ANN



Weights (α)

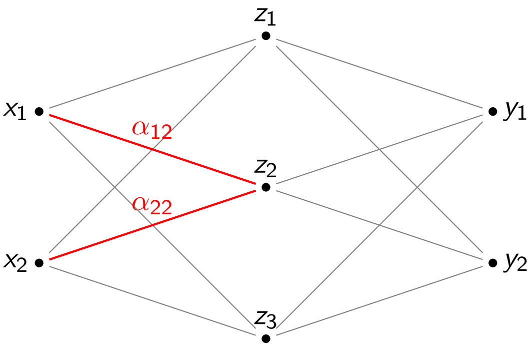


$$\overbrace{\begin{bmatrix} x_1 & x_2 \end{bmatrix}}^{X} \overbrace{\begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \end{bmatrix}}^{\alpha}$$

$$\underbrace{\phantom{\alpha_{11}}}_{\alpha_1} \quad \underbrace{\phantom{\alpha_{12}}}_{\alpha_2} \quad \underbrace{\phantom{\alpha_{13}}}_{\alpha_3}$$

$$z_1 = \sigma(X\alpha_1)$$

Weights (α)

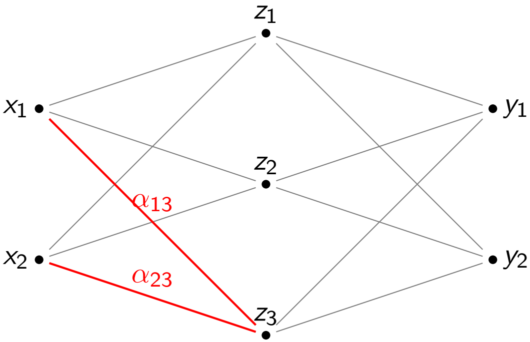


$$\overbrace{\begin{bmatrix} x_1 & x_2 \end{bmatrix}}^{X} \overbrace{\begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \end{bmatrix}}^{\alpha}$$

$$\underbrace{\quad}_{\alpha_1} \quad \underbrace{\quad}_{\alpha_2} \quad \underbrace{\quad}_{\alpha_3}$$

$$z_2 = \sigma(X\alpha_2)$$

Weights (α)

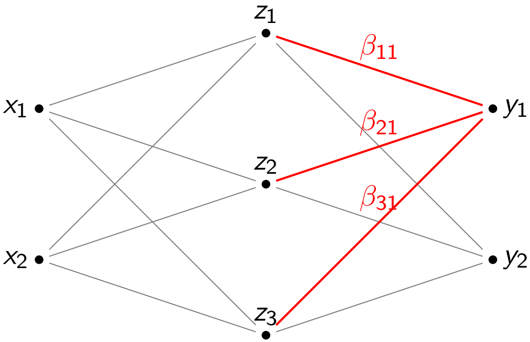


$$\overbrace{\begin{bmatrix} x_1 & x_2 \end{bmatrix}}^{X} \overbrace{\begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \end{bmatrix}}^{\alpha}$$

$$\underbrace{\phantom{\alpha_{11}}}_{\alpha_1} \quad \underbrace{\phantom{\alpha_{12}}}_{\alpha_2} \quad \underbrace{\phantom{\alpha_{13}}}_{\alpha_3}$$

$$z_3 = \sigma(X\alpha_3)$$

Weights (β)

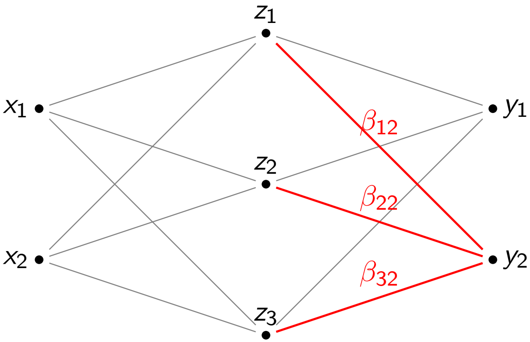


$$\overbrace{\begin{bmatrix} z_1 & z_2 & z_3 \end{bmatrix}}^{Z} \overbrace{\begin{bmatrix} \beta_{11} & \beta_{12} \\ \beta_{21} & \beta_{22} \\ \beta_{31} & \beta_{23} \end{bmatrix}}^{\beta}$$

$$\underbrace{\quad}_{\beta_1} \quad \underbrace{\quad}_{\beta_2}$$

$$y_1 = g(Z\beta_1)$$

Weights (β)

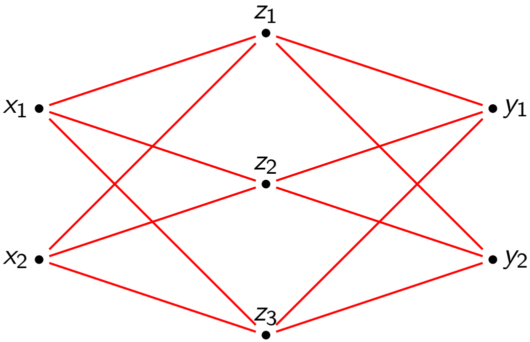


$$\overbrace{\begin{bmatrix} z_1 & z_2 & z_3 \end{bmatrix}}^{Z} \overbrace{\begin{bmatrix} \underbrace{\begin{matrix} \beta_{11} \\ \beta_{21} \\ \beta_{31} \end{matrix}}_{\beta_1} & \underbrace{\begin{matrix} \beta_{12} \\ \beta_{22} \\ \beta_{23} \end{matrix}}_{\beta_2} \end{bmatrix}}^{\beta}$$

$$y_2 = g(Z\beta_2)$$

Summary

$$\overbrace{X}^{1 \times p} \quad \overbrace{\alpha}^{p \times M} \quad \overbrace{Z}^{1 \times M} \quad \overbrace{\beta}^{M \times K} \quad \overbrace{Y}^{1 \times K}$$



$$Z = \sigma(X\alpha)$$

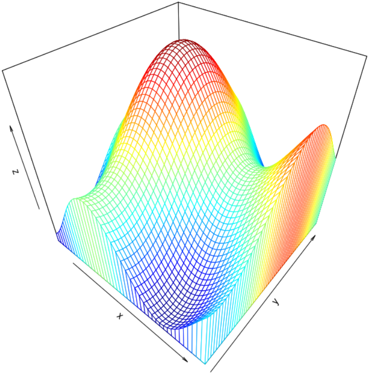
$$Y = g(Z\beta)$$

$$f(X) = g(Z\beta)$$

Gradient Descent

$$R(\theta) = \sum_{i=1}^{n} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2 \tag{6}$$

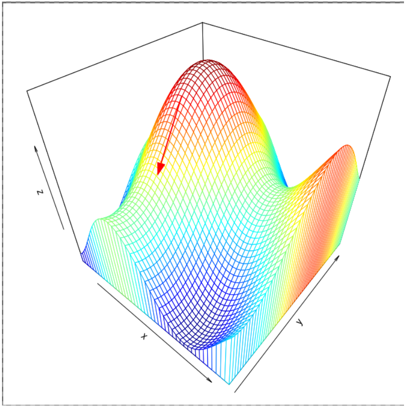
- forward propagation: evaluates error (height of surface)
- back propagation: readjusts weights to reduce error (vector)



Gradient Descent

$$R(\theta) = \sum_{i=1}^{n} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2 \tag{6}$$

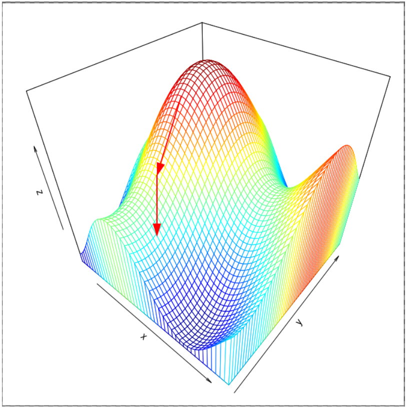
- forward propagation: evaluates error (height of surface)
- back propagation: readjusts weights to reduce error (vector)



Gradient Descent

$$R(\theta) = \sum_{i=1}^{n} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2 \tag{6}$$

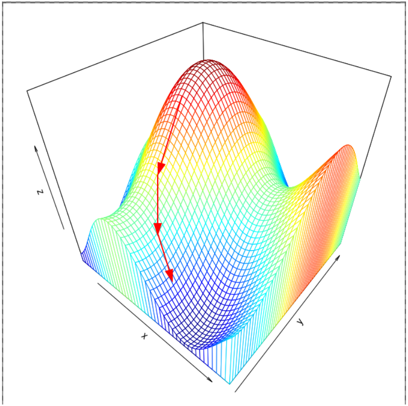
- forward propagation: evaluates error (height of surface)
- back propagation: readjusts weights to reduce error (vector)



Gradient Descent

$$R(\theta) = \sum_{i=1}^{n} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2 \tag{6}$$

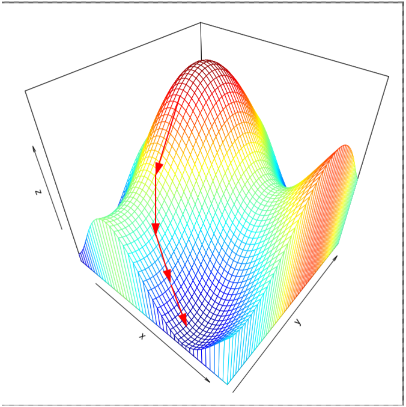
- forward propagation: evaluates error (height of surface)
- back propagation: readjusts weights to reduce error (vector)



Gradient Descent

$$R(\theta) = \sum_{i=1}^{n} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2 \tag{6}$$

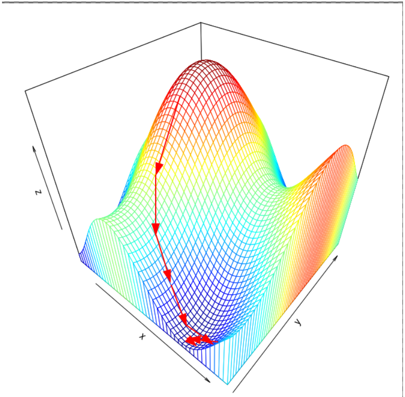
- forward propagation: evaluates error (height of surface)
- back propagation: readjusts weights to reduce error (vector)



Gradient Descent

$$R(\theta) = \sum_{i=1}^{n} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2 \tag{6}$$

- forward propagation: evaluates error (height of surface)
- back propagation: readjusts weights to reduce error (vector)



Gradient Descent

Let γ_r be the *learning rate* at step r. We update parameters by

$$\beta_{mk}^{(r+1)} = \beta_{mk}^{(r)} - \gamma_r \sum_{i=1}^{n} \frac{\partial R_i}{\partial \beta_{mk}^{(r)}}$$

$$\alpha_{jm}^{(r+1)} = \alpha_{jm}^{(r)} - \gamma_r \sum_{i=1}^{n} \frac{\partial R_i}{\partial \alpha_{jm}^{(r)}}$$

Back Propagation (β)

$$
\begin{aligned}
R_i &= \sum_{k=1}^{K} \left(y_{ik} - f_k(X_i)\right)^2 && \text{defn. RSS} \\
\frac{\partial}{\partial \beta_{mk}} R_i &= \sum_{k=1}^{K} -2\left(y_{ik} - f_k(X_i)\right) f_k'(X_i) && \text{take derivative} \\
&= \sum_{k=1}^{K} -2\left(y_{ik} - f_k(X_i)\right) g_k'(Z_i\beta) \frac{\partial}{\partial \beta_{mk}} (Z_i\beta) && \text{sub. } f_k(X_i) = g_k(Z_i\beta) \\
&= -2\left(y_{ik} - f_k(X_i)\right) g_k'(Z_i\beta) z_{im} && \text{derivative cancels terms}
\end{aligned}
$$

Back Propagation (α)

$$
\begin{aligned}
R_i &= \sum_{k=1}^{K} (y_{ik} - f_k(X_i))^2 && \text{defn. RSS} \\
\frac{\partial}{\partial \alpha_{jm}} R_i &= \sum_{k=1}^{K} -2\,(y_{ik} - f_k(X_i))\, f_k'(X_i) && \text{take derivative} \\
&= \sum_{k=1}^{K} -2\,(y_{ik} - f_k(X_i))\, g_k'(Z_i\beta) \frac{\partial}{\partial \alpha_{jm}} (Z_i\beta) && \text{sub. } f_k(X_i) = g_k(Z_i\beta) \\
&= \sum_{k=1}^{K} -2\,(y_{ik} - f_k(X_i))\, g_k'(Z_i\beta) \frac{\partial}{\partial \alpha_{jm}} (\sigma(X_i\alpha)\beta) && \text{sub. } Z_i = \sigma(X_i\alpha) \\
&= \sum_{k=1}^{K} -2\,(y_{ik} - f_k(X_i))\, g_k'(Z_i\beta) \beta_{mk} \sigma'(X_i\alpha) x_{ij} && \text{derivative cancels terms}
\end{aligned}
$$

Back-Propagation Equations

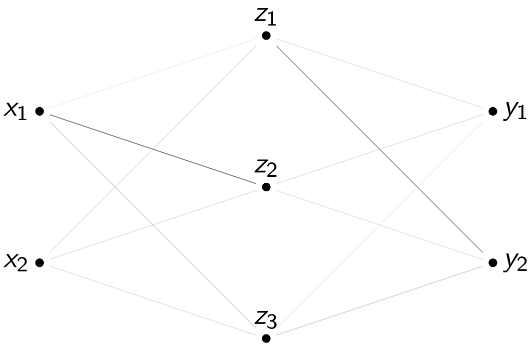
$$\frac{\partial}{\partial \beta_{mk}} R_i = \overbrace{-2\left(y_{ik} - f_k(X_i)\right) g_k'(Z_i\beta)}\, z_{im}$$

$$\frac{\partial}{\partial \alpha_{jm}} R_i = \sum_{k=1}^{K} \underbrace{-2\left(y_{ik} - f_k(X_i)\right) g_k'(Z_i\beta)}\, \beta_{mk}\sigma'(X_i\alpha)x_{ij}$$

- notice the compositional nature of the back-propagation equations
- yields significant computational savings

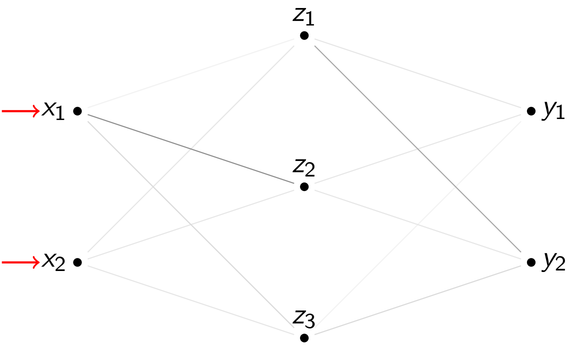
Training ANNs

1. randomize weights
2. forward propagation: calculate error
3. back propagation: adjust weights
4. repeat until convergence



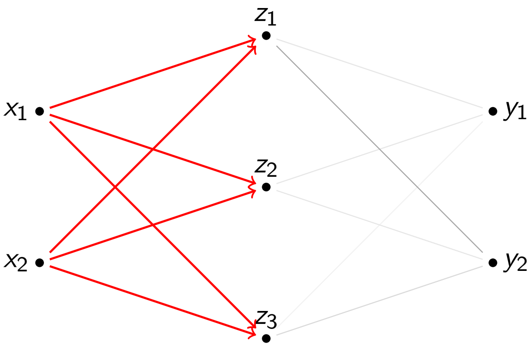
Training ANNs

1. randomize weights
2. forward propagation: calculate error
3. back propagation: adjust weights
4. repeat until convergence



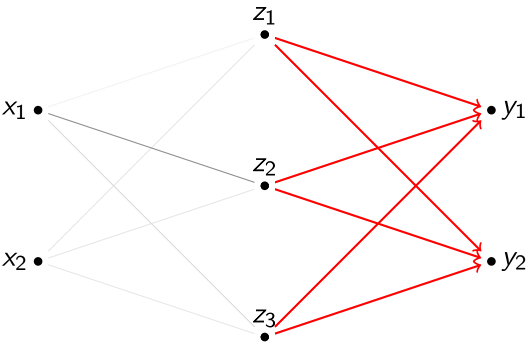
Training ANNs

1. randomize weights
2. forward propagation: calculate error
3. back propagation: adjust weights
4. repeat until convergence



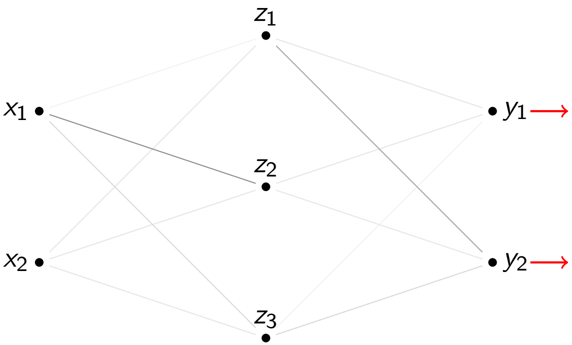
Training ANNs

1. randomize weights
2. forward propagation: calculate error
3. back propagation: adjust weights
4. repeat until convergence



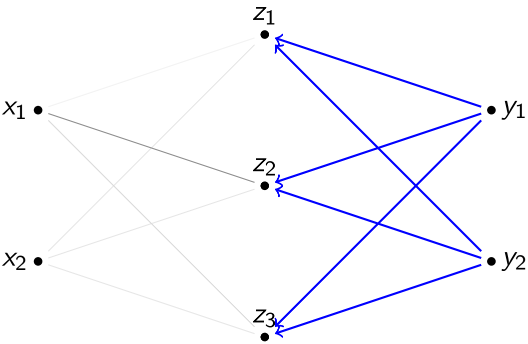
Training ANNs

1. randomize weights
2. forward propagation: calculate error
3. back propagation: adjust weights
4. repeat until convergence



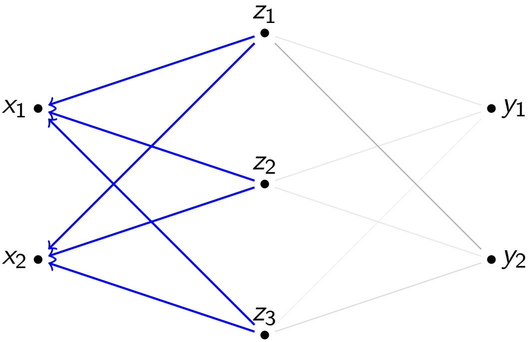
Training ANNs

1. randomize weights
2. forward propagation: calculate error
3. back propagation: adjust weights
4. repeat until convergence



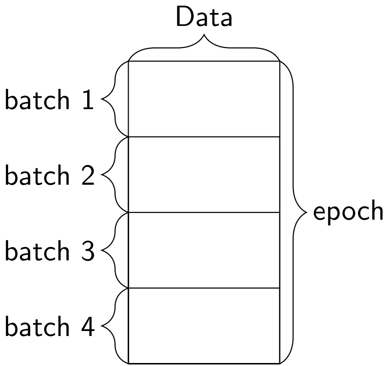
Training ANNs

1. randomize weights
2. forward propagation: calculate error
3. back propagation: adjust weights
4. repeat until convergence



Terminology

- **batch:** number of samples processed before updating parameters
- **online learning:** batch size of 1; useful for large data sets
- **epoch:** one sweep through training data; seems less arbitrary



Overfitting

- early stopping: stop before reaching global minimum
- penalization
  - MSE can be split into bias and variance
  - inflate bias to improve variance
  - flexible models require regularization to prevent overfitting

$$\min_{\theta} R(\theta) + \lambda J(\theta) \tag{7}$$

where penalty $J(\theta)$ is defined as

$$J(\theta) = \sum_{m,k} \frac{\beta_{mk}^2}{1 + \beta_{mk}^2} + \sum_{j,m} \frac{\alpha_{jm}^2}{1 + \alpha_{jm}^2}$$

Tuning

- training ANNs is an art; no hard and fast rules to training
- number of nodes (universal approximation)
  - in theory, a single layer with large number of nodes should be able to approximate any function
- number of hidden layers (abstraction)

Conclusion

- found great success in complex tasks
  - image recognition
  - natural language processing
- black-box approach: opaque by complexity
  - lack of interpretability prohibits ANNs' usefulness in medicine and science
- resource hungry
  - requires lots of data
  - requires lots of computational power
- no free lunch: simpler models are often more effective

Questions