Computer Vision:
Neural networks

Siheng Chen 陈思衡

Neural networks

Intro to Neural networks

Backpropagation

Roadmap

Image processing

- filtering, Hough transform, corner detection, SIFT

Machine learning

- PCA, K-means clustering, logistic regression

Deep learning

- **backpropagation**, training tricks, CNN, Transformer

CV tasks

- Object detection, segmentation, tracking, …

A recipe for machine learning

1. Given training data: $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$

2. Design your model: $\widehat{\mathbf{y}}_i = f_\theta(\mathbf{x}_i)$

3. Choose your loss function: $L(\widehat{\mathbf{y}}_i, \mathbf{y}_i) \in \mathbb{R}$

4. Formulate an optimization: $\theta^* = \arg\min_\theta J(\theta) = \sum_{i=1}^{N} L(\widehat{\mathbf{y}}_i, \mathbf{y}_i)$

5. Train your model: $\theta^{(t+1)} = \theta^{(t)} - \alpha \frac{\partial J}{\partial \theta}$

Logistic regression

1. Given training data: $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$

2. Design your model: $\hat{\mathbf{y}}_i = f_\theta(\mathbf{x}_i)$

   $\hat{y} = \sigma(\mathbf{w}^T \mathbf{x} + b)$

3. Choose your loss function: $L(\hat{\mathbf{y}}_i, \mathbf{y}_i) \in \mathbb{R}$

   $L(\hat{y}, y) = -(y \log \hat{y} + (1-y) \log(1-\hat{y}))$

4. Formulate an optimization: $\theta^* = \arg\min_\theta J(\theta) = \sum_{i=1}^{N} L(\hat{\mathbf{y}}_i, \mathbf{y}_i)$

5. Train your model: $\theta^{(t+1)} = \theta^{(t)} - \alpha \frac{\partial J}{\partial \theta}$

Logistic regression

m training samples

$$J(\mathbf{w}, b) = \frac{1}{m}\sum_{i=1}^{m} L(a^{(i)}, y^{(i)}) \quad a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(\mathbf{w}^T\mathbf{x}^{(i)} + b)$$

Partial derivative computation

$$\frac{\partial J}{\partial \mathbf{w}} = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial \mathbf{w}} L(a^{(i)}, y^{(i)})$$

$$= \frac{1}{m}\sum_{i=1}^{m} (a^{(i)} - y^{(i)})\mathbf{x}^{(i)}$$

$$\frac{\partial J}{\partial b} = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial b} L(a^{(i)}, y^{(i)})$$

$$= \frac{1}{m}\sum_{i=1}^{m} (a^{(i)} - y^{(i)})$$

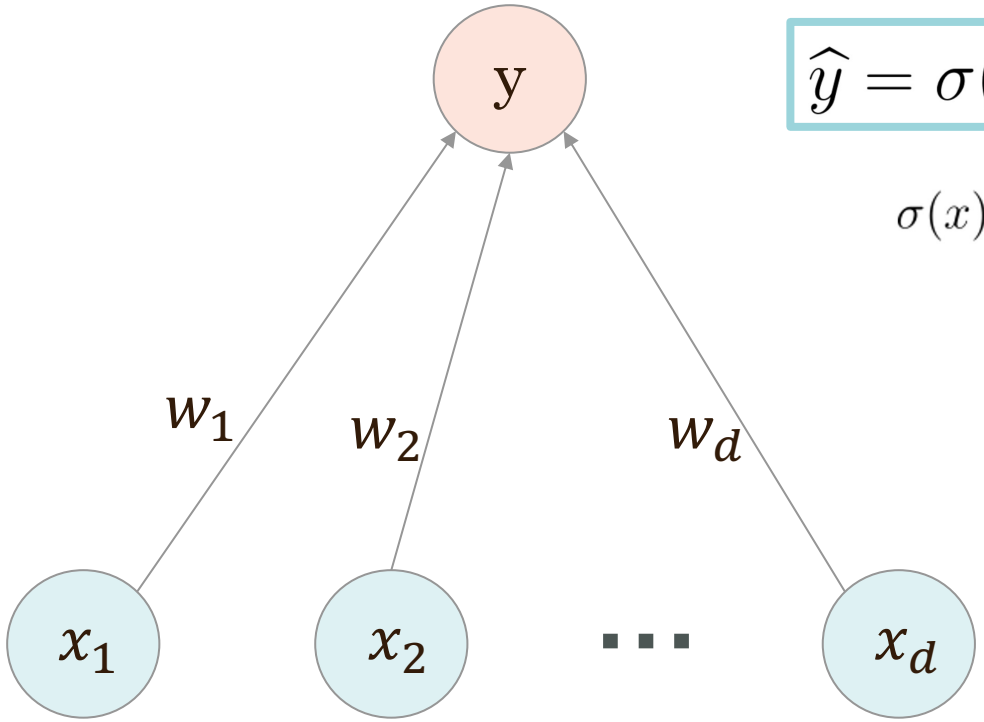
update

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha\frac{\partial J}{\partial \mathbf{w}} \qquad b \leftarrow b - \alpha\frac{\partial J}{\partial b}$$

Logistic regression

output

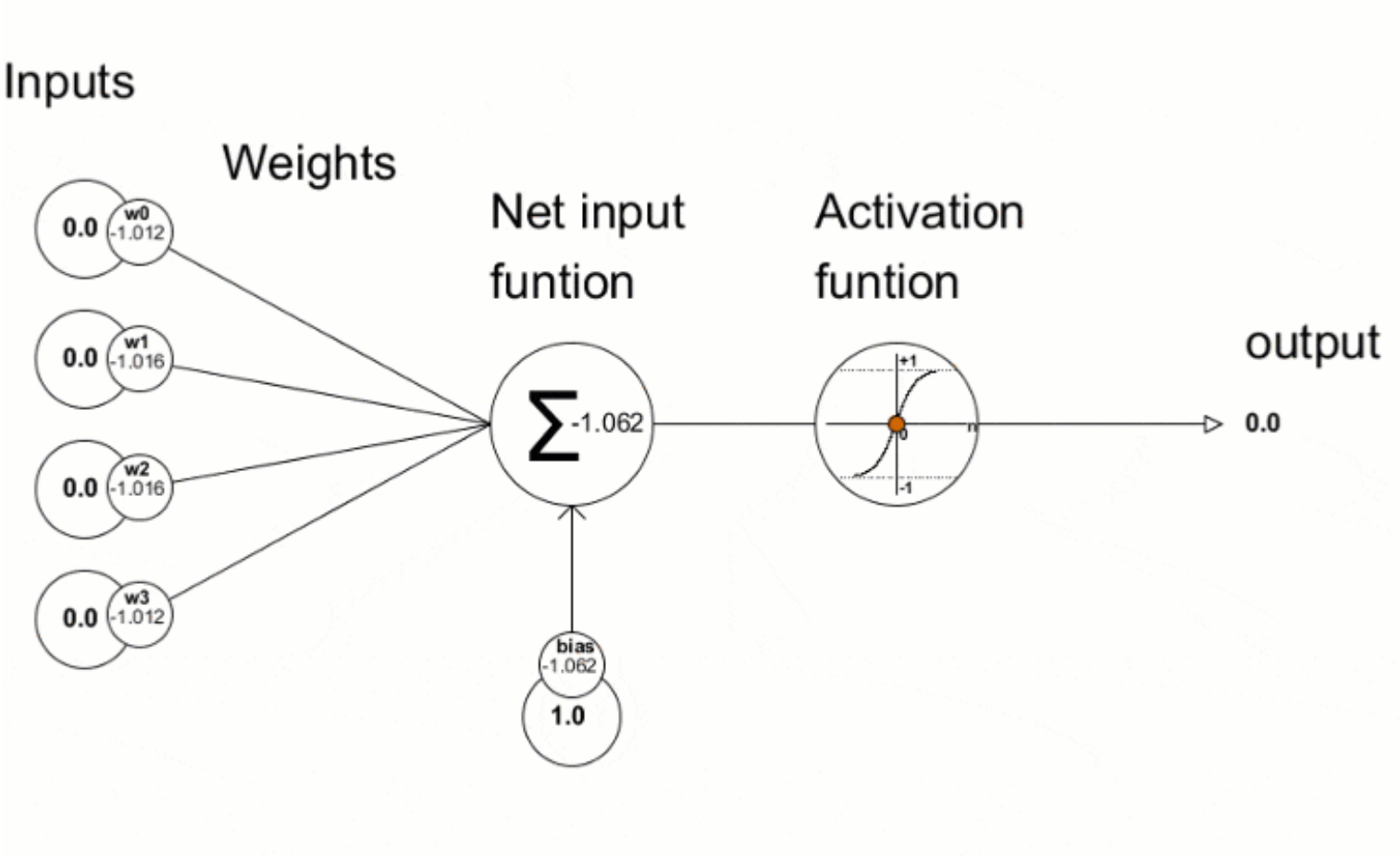
input



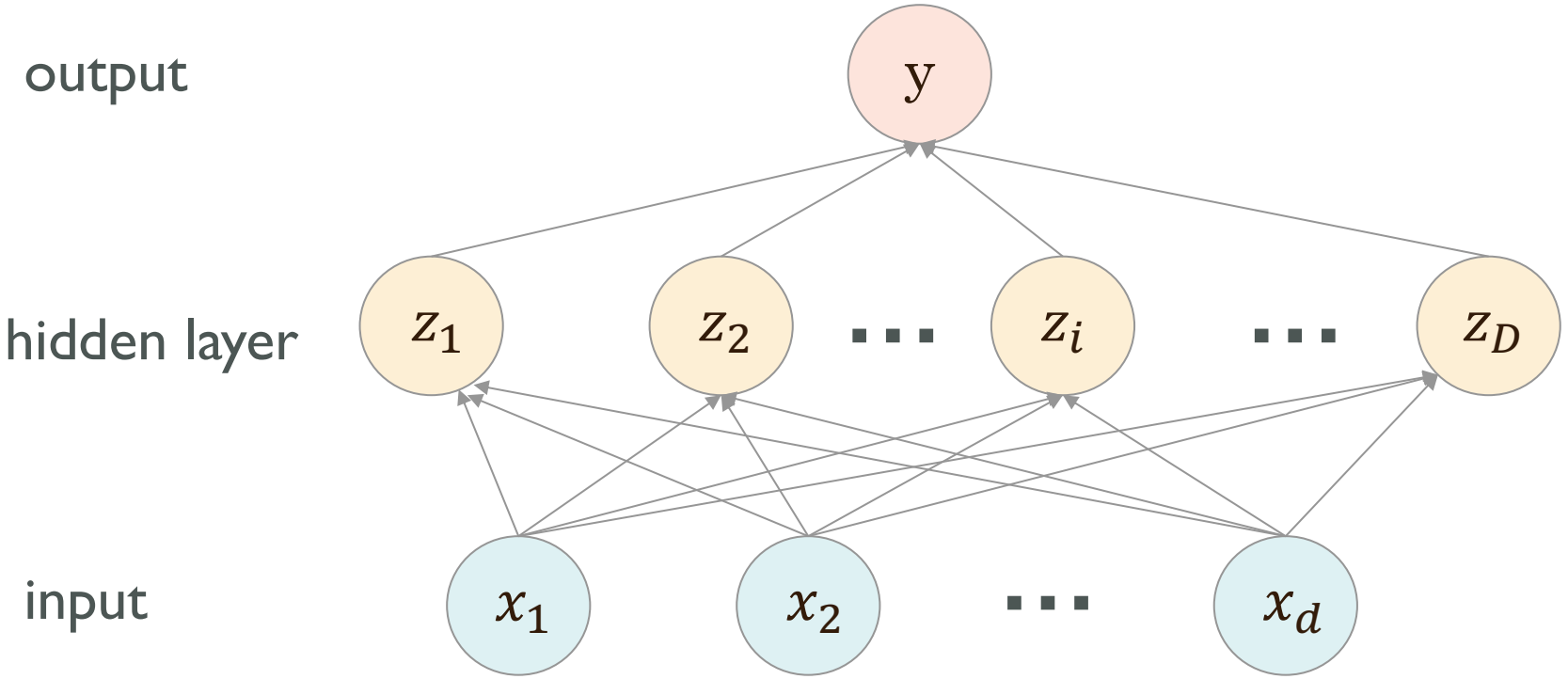
$$\hat{y} = \sigma(\mathbf{w}^T\mathbf{x} + b)$$

$$\sigma(x) = \frac{1}{1+\exp^{-x}}$$

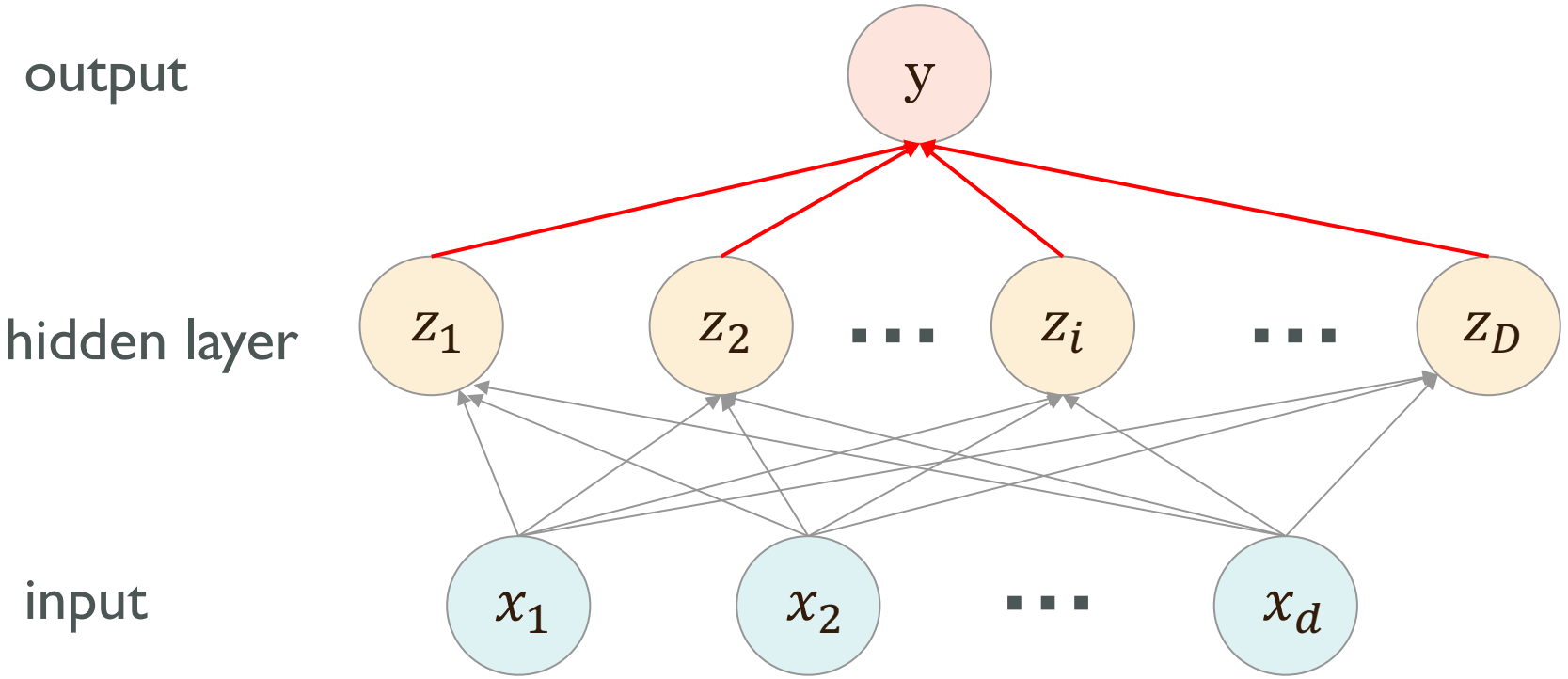
Logistic regression



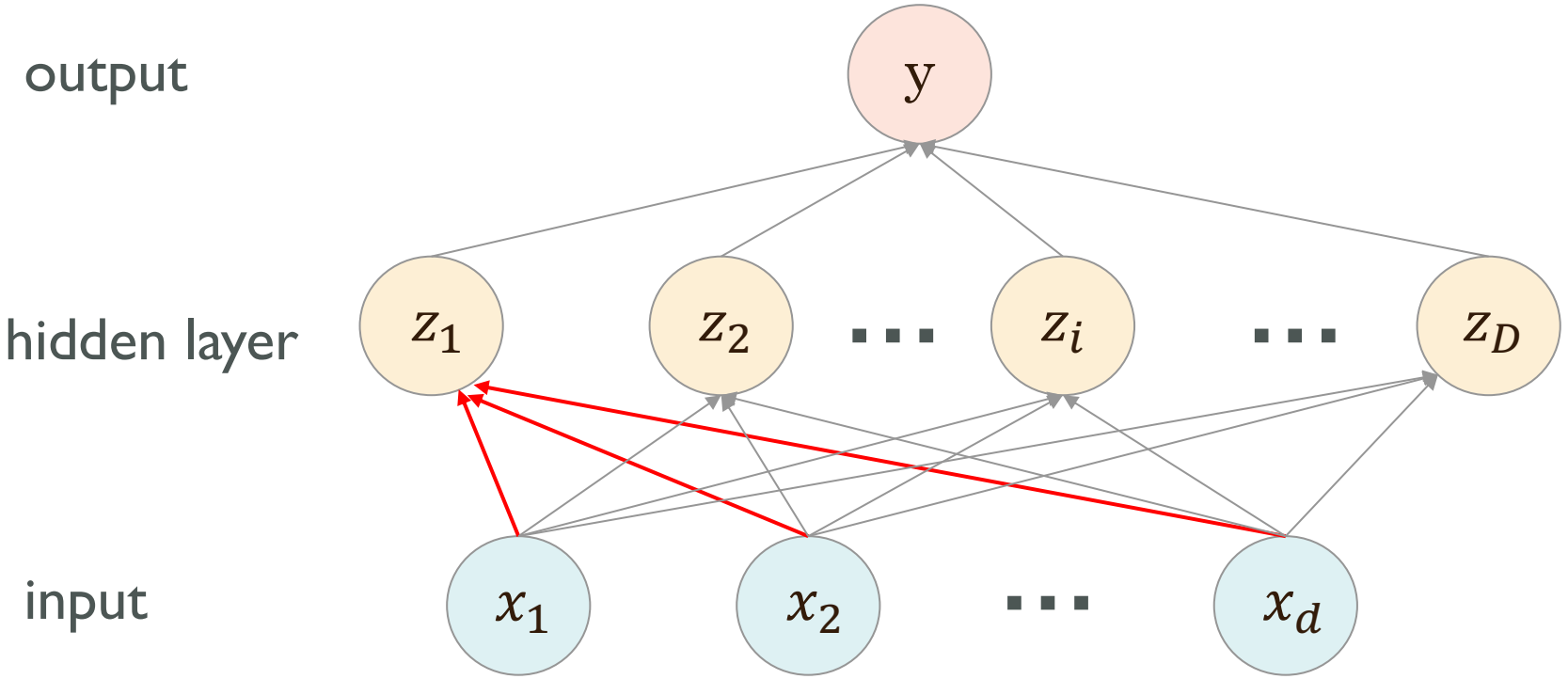
Neural networks



Neural networks

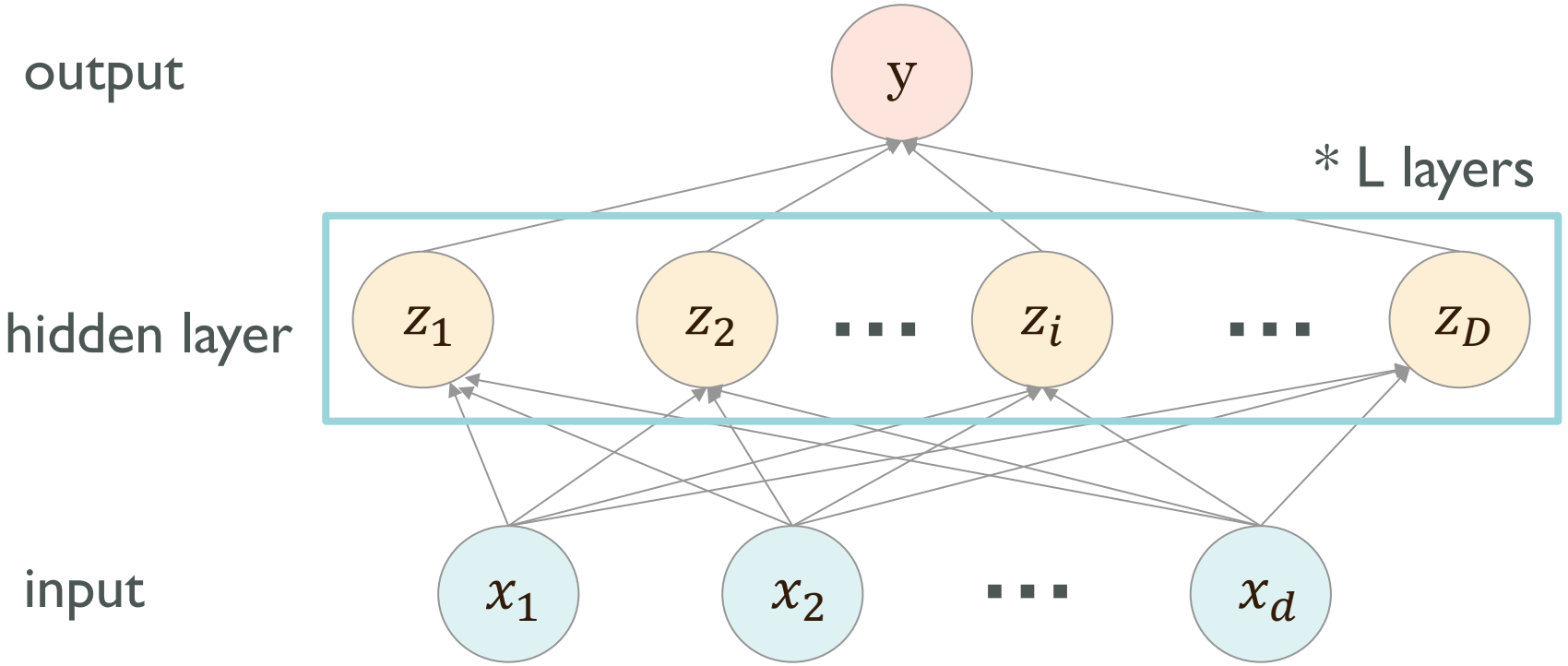


Neural networks



The computation of each neural network unit resembles binary logistic regression.

Deep neural networks

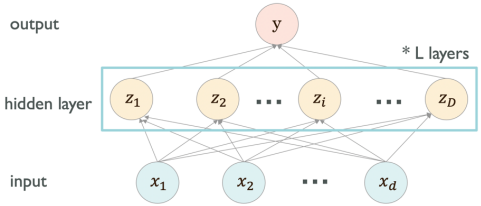


The computation of each neural network unit resembles binary logistic regression.

A recipe for deep neural networks

1. Given training data: $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$

2. Design your model: $\hat{\mathbf{y}}_i = f_\theta(\mathbf{x}_i)$



Design a computation graph

3. Choose your loss function: $L(\hat{\mathbf{y}}_i, \mathbf{y}_i) \in \mathbb{R}$

Task specific: Classification? Regression?

4. Formulate an optimization: $\theta^* = \arg\min_\theta J(\theta) = \sum_{i=1}^{N} L(\hat{\mathbf{y}}_i, \mathbf{y}_i)$

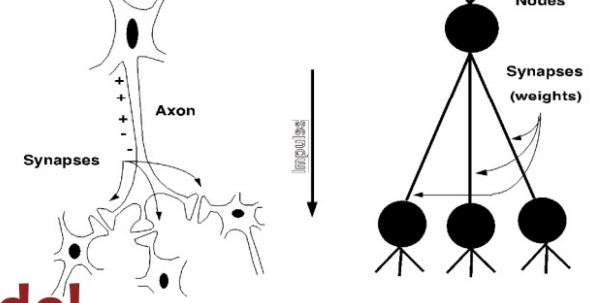
Regularization? Specific priors?

5. Train your model: $\theta^{(t+1)} = \theta^{(t)} - \alpha \frac{\partial J}{\partial \theta}$

Solver? GD? SGD? Adam?

From Biological to Artificial

The motivation for Artificial Neural Networks comes from biology…



Biological "Model"

- **Neuron:** an excitable cell
- **Synapse:** connection between neurons
- A *neuron* sends an **electrochemical pulse** along its *synapses* when a sufficient voltage change occurs
- **Biological Neural Network:** collection of neurons along some pathway through the brain

Artificial Model

- **Neuron:** node in a directed acyclic graph (DAG)
- **Weight:** multiplier on each edge
- **Activation Function:** nonlinear thresholding function, which allows a neuron to "fire" when the input value is sufficiently high
- **Artificial Neural Network:** collection of neurons into a DAG, which define some differentiable function

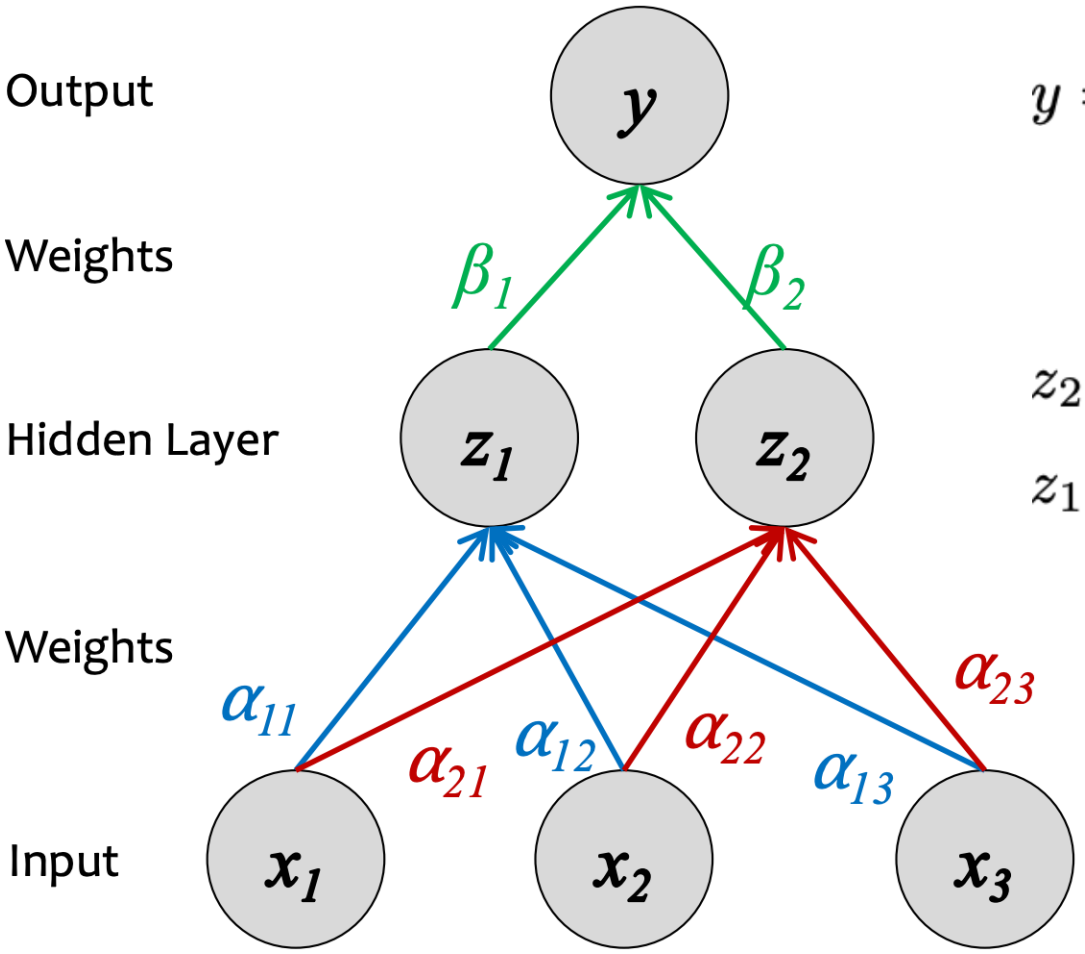
Biological "Computation"

- Neuron switching time : ~ 0.001 sec
- Number of neurons: ~ 10^{10}
- Connections per neuron: ~ 10^{4-5}
- Scene recognition time: ~ 0.1 sec

Artificial Computation

- Many neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed processes

Neural networks

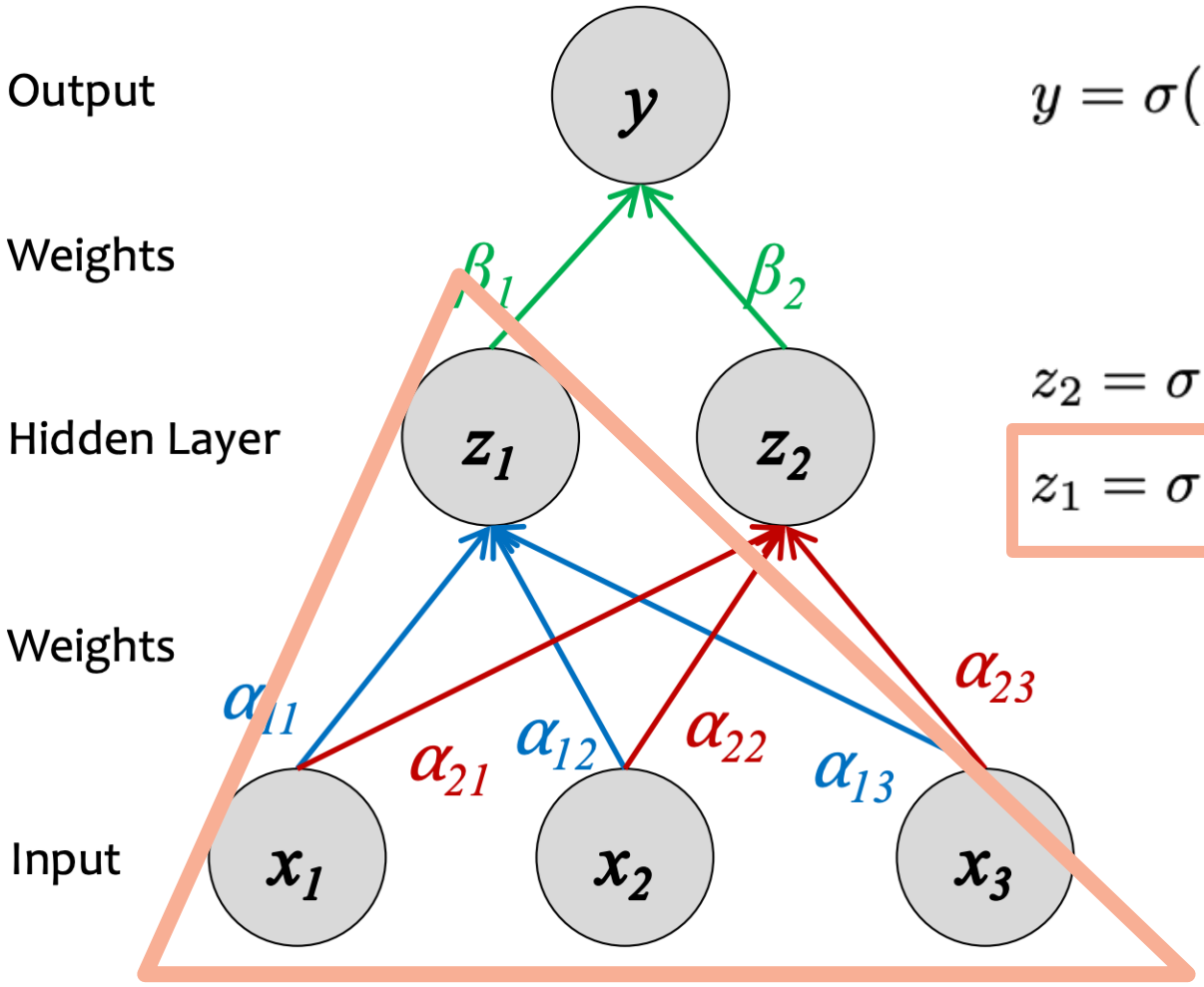


$$y = \sigma(\beta_1 z_1 + \beta_2 z_2)$$

$$z_2 = \sigma(\alpha_{21} x_1 + \alpha_{22} x_2 + \alpha_{23} x_3)$$

$$z_1 = \sigma(\alpha_{11} x_1 + \alpha_{12} x_2 + \alpha_{13} x_3)$$

Neural networks

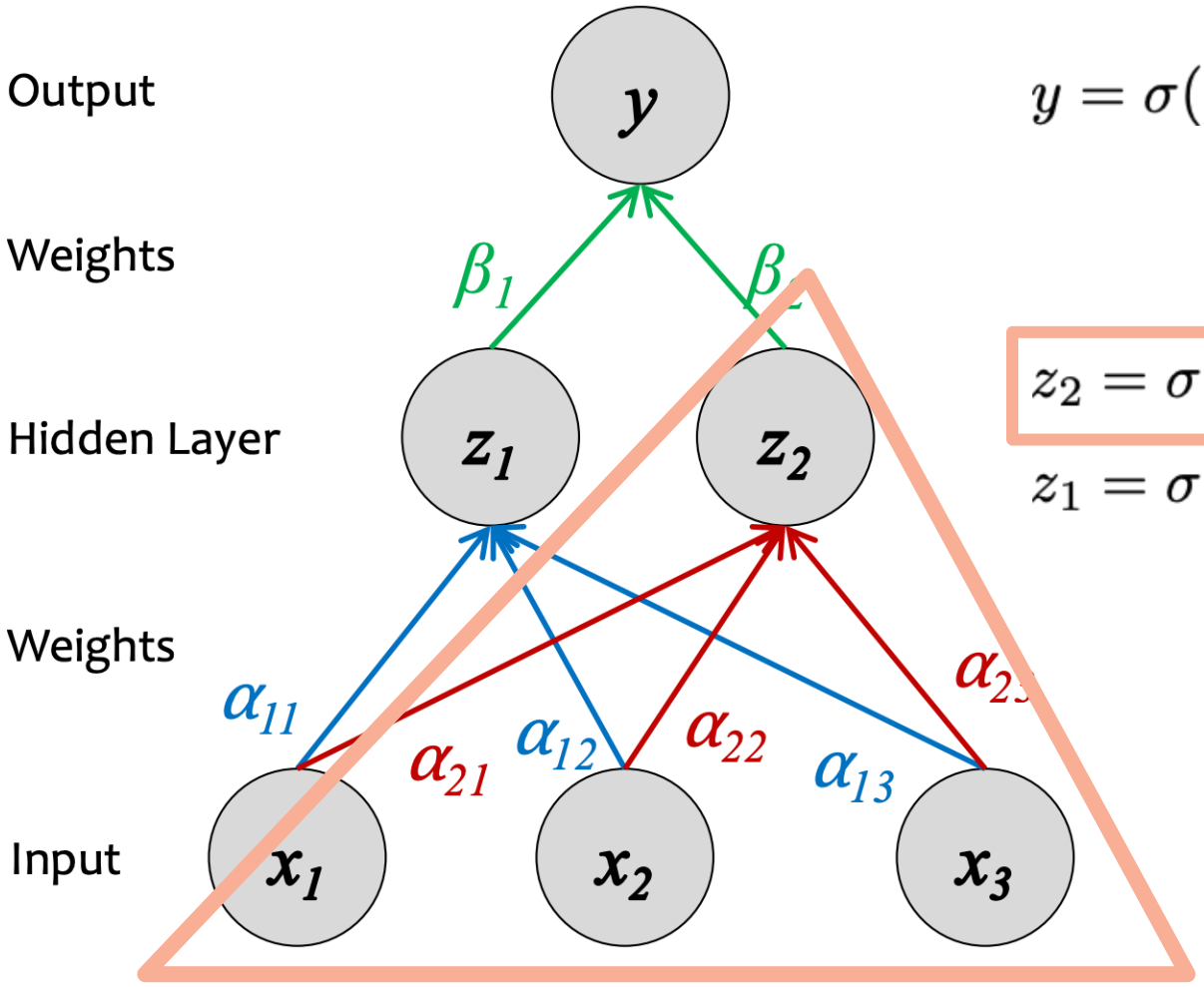


$$y = \sigma(\beta_1 z_1 + \beta_2 z_2)$$

$$z_2 = \sigma(\alpha_{21} x_1 + \alpha_{22} x_2 + \alpha_{23} x_3)$$

$$z_1 = \sigma(\alpha_{11} x_1 + \alpha_{12} x_2 + \alpha_{13} x_3)$$

Neural networks

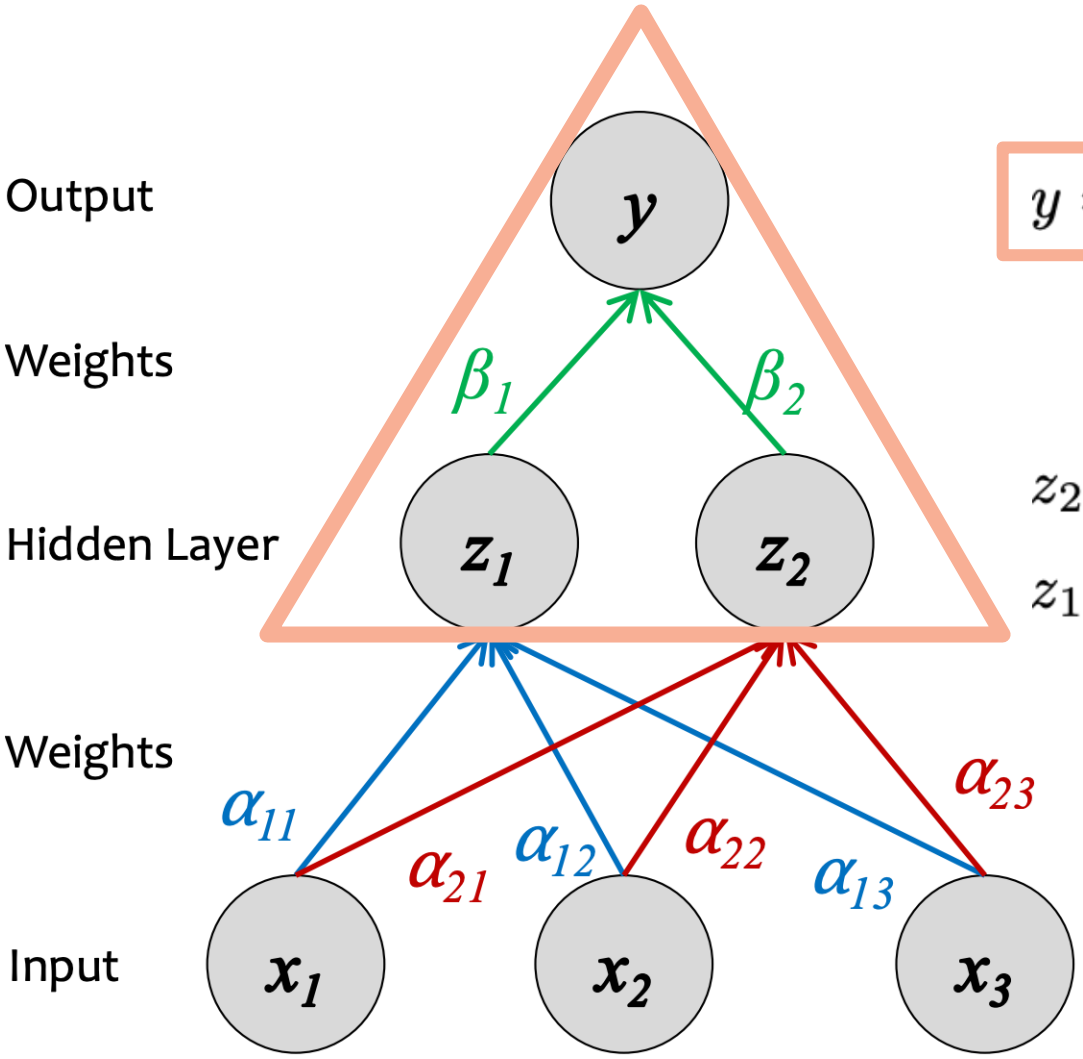


$$y = \sigma(\beta_1 z_1 + \beta_2 z_2)$$

$$z_2 = \sigma(\alpha_{21} x_1 + \alpha_{22} x_2 + \alpha_{23} x_3)$$

$$z_1 = \sigma(\alpha_{11} x_1 + \alpha_{12} x_2 + \alpha_{13} x_3)$$

Neural networks

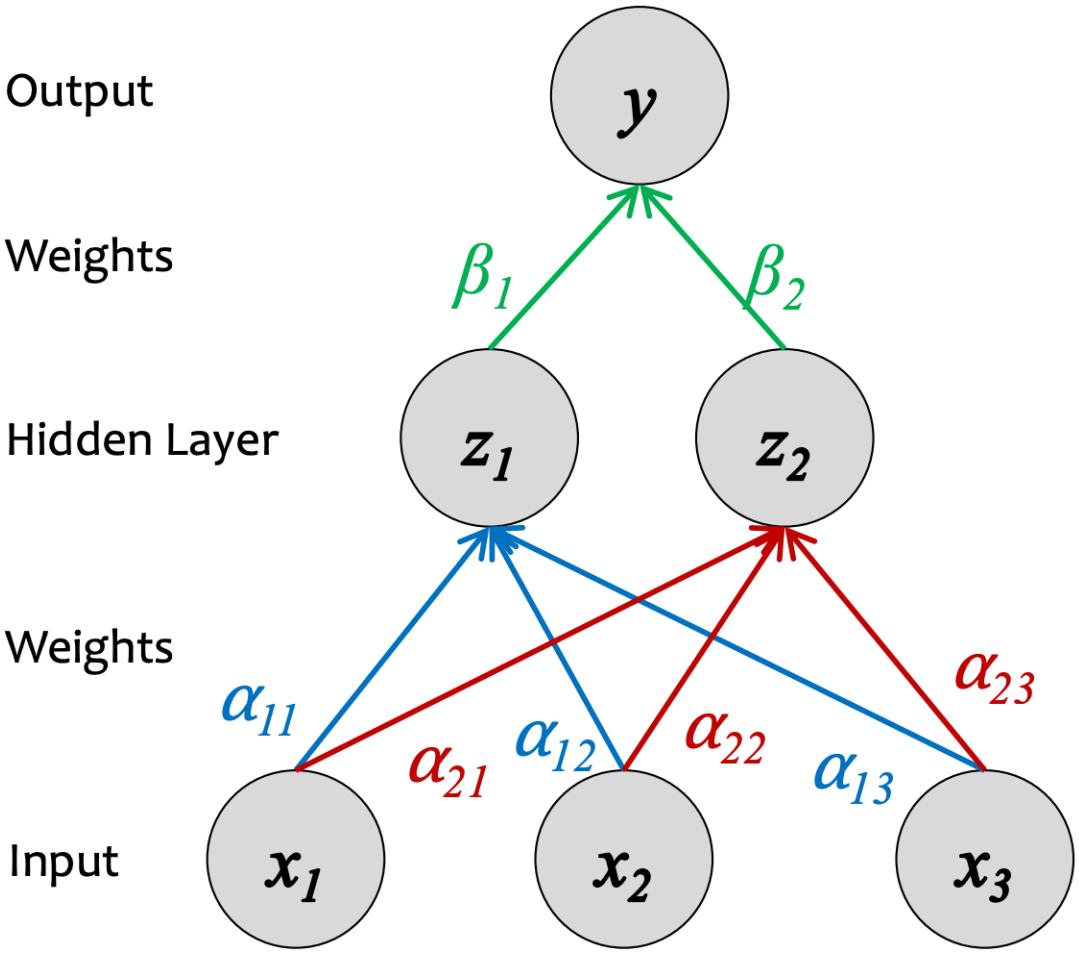


$$y = \sigma(\beta_1 z_1 + \beta_2 z_2)$$

$$z_2 = \sigma(\alpha_{21} x_1 + \alpha_{22} x_2 + \alpha_{23} x_3)$$

$$z_1 = \sigma(\alpha_{11} x_1 + \alpha_{12} x_2 + \alpha_{13} x_3)$$

Neural networks



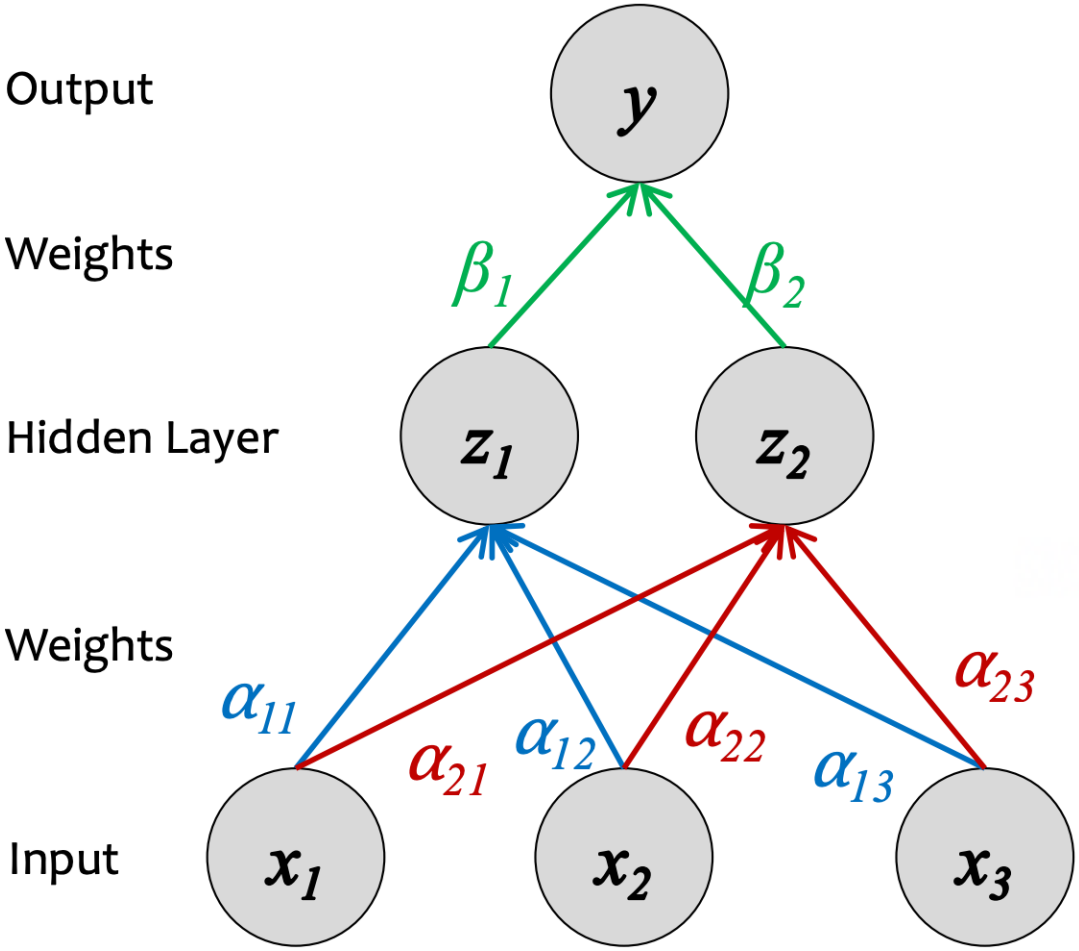
$$y = \sigma(\boldsymbol{\beta}^T \mathbf{z})$$

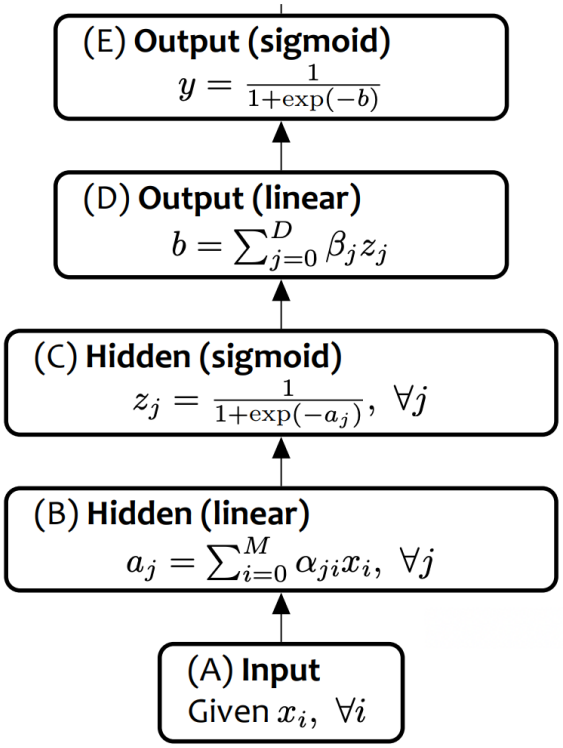
$$z_2 = \sigma(\boldsymbol{\alpha}_{2,\cdot}^T \mathbf{x})$$

$$z_1 = \sigma(\boldsymbol{\alpha}_{1,\cdot}^T \mathbf{x})$$

http://www.cs.cmu.edu/~mgormley/courses/10601/slides/lecture11-nn.pdf

Neural networks





http://www.cs.cmu.edu/~mgormley/courses/10601/slides/lecture11-nn.pdf

Neural networks

Many design decisions to make:

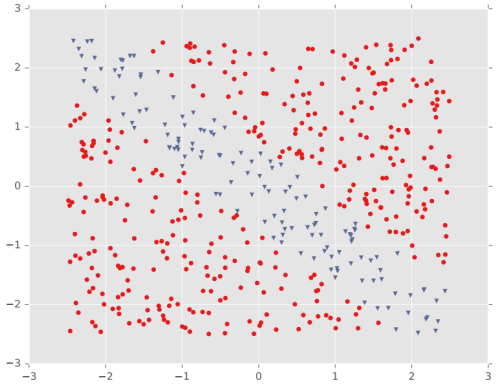
1. # of hidden layers (depth)
2. # of units per hidden layer (width)
3. Type of activation function (nonlinearity)
4. Form of objective function
5. How to initialize the parameters

http://www.cs.cmu.edu/~mgormley/courses/10601/slides/lecture11-nn.pdf

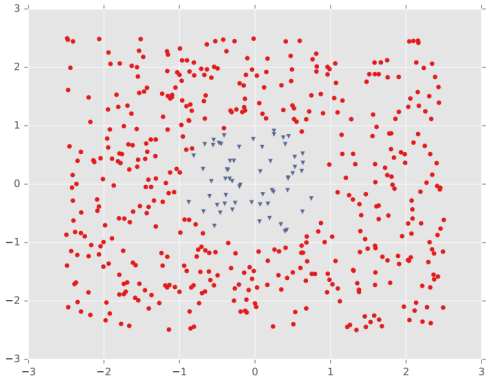
Neural networks

Can logistic regression handle these cases?

Example #1: Diagonal Band

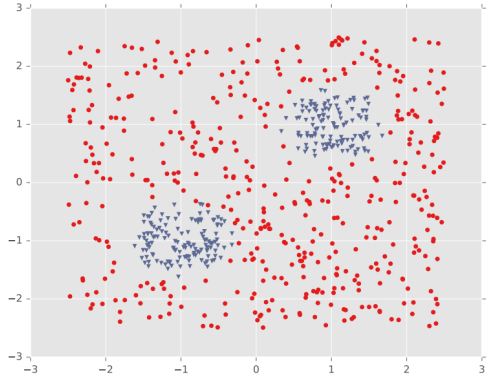


Example #2: One Pocket



Example #3: Four Gaussians

Example #4: Two Pockets



Neural networks

Example 3: Four Gaussians

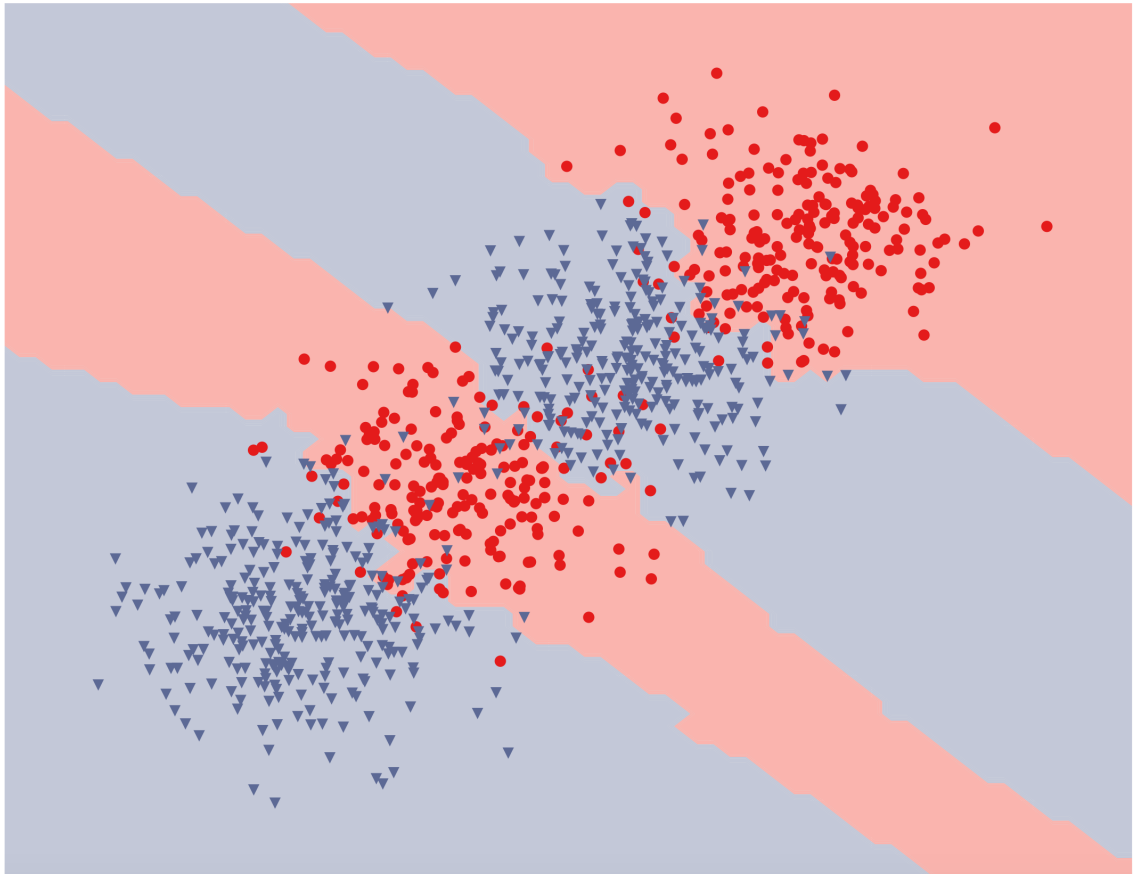
Neural networks

Example 3: Four Gaussians

Logistic regression

Neural networks

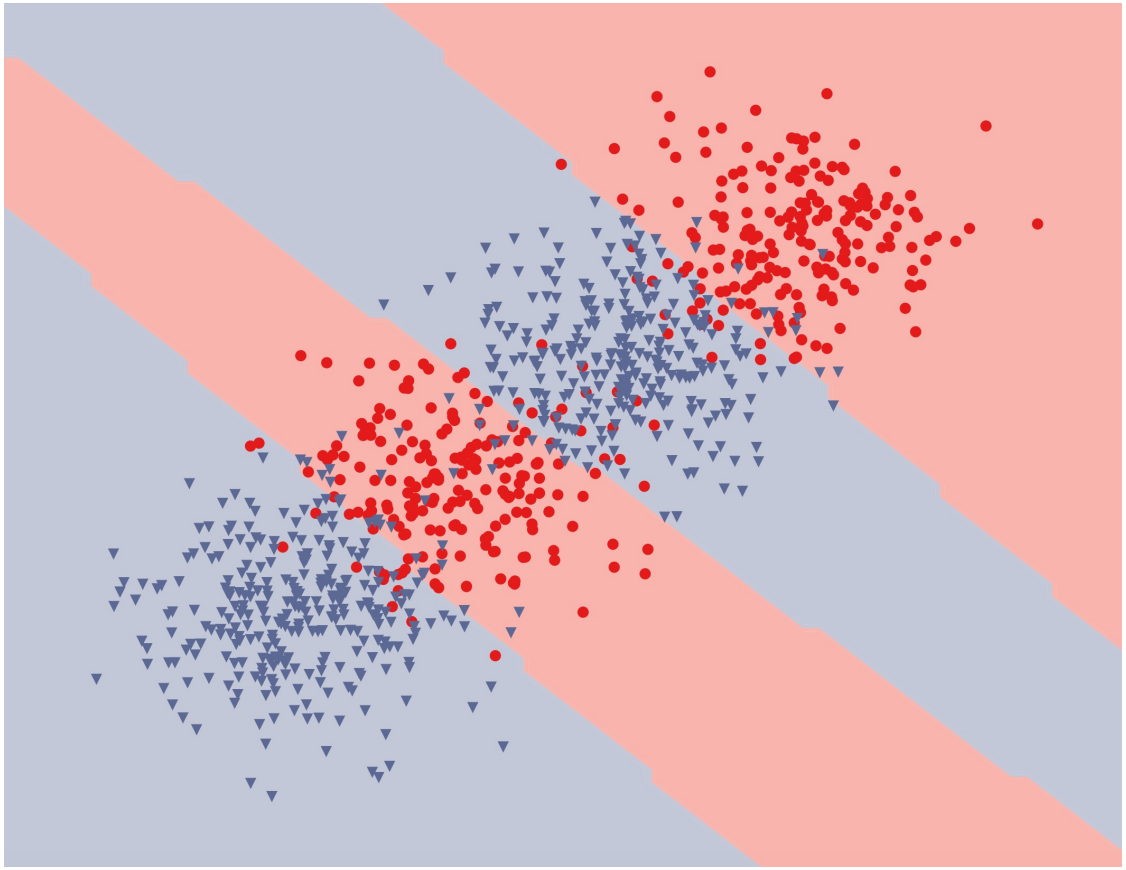
Example 3: Four Gaussians



k-NN (k=5, metric=Euclidean)

Neural networks

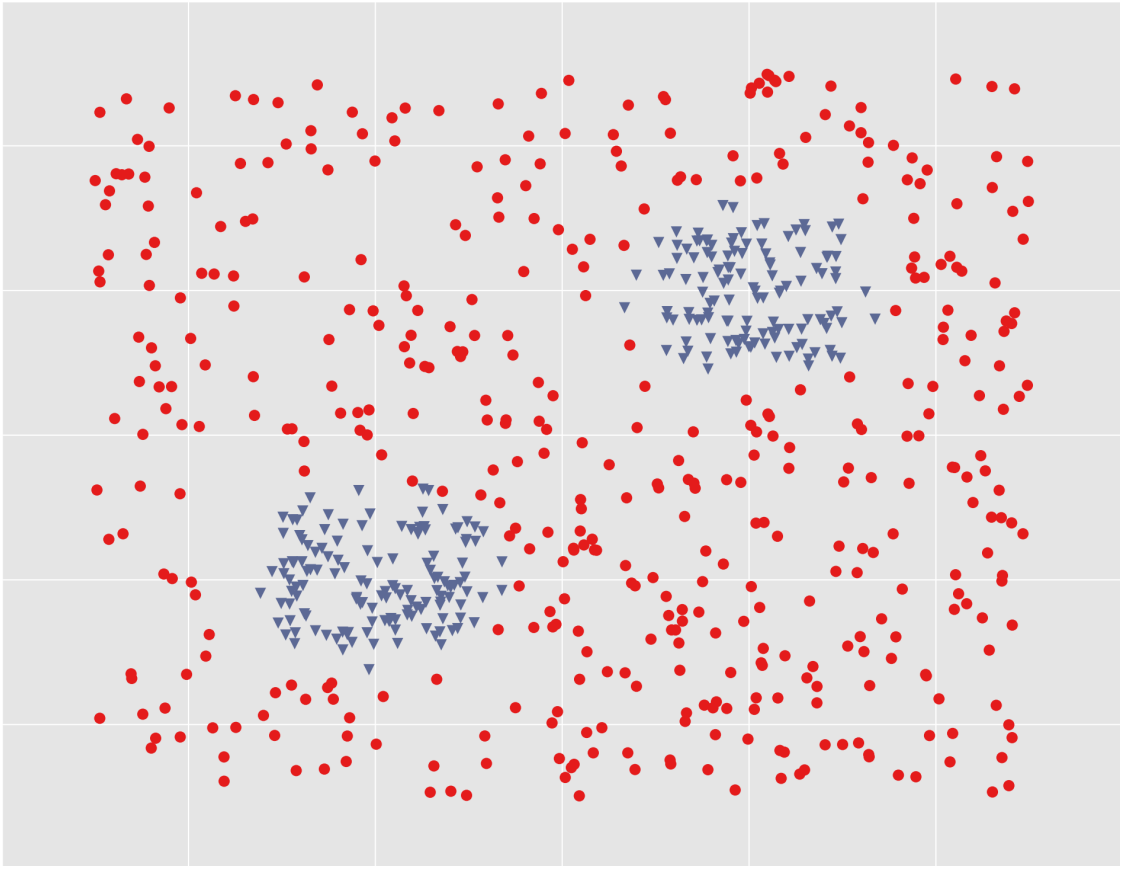
Example 3: Four Gaussians



Neural network (hidden=2, activation=logistic)

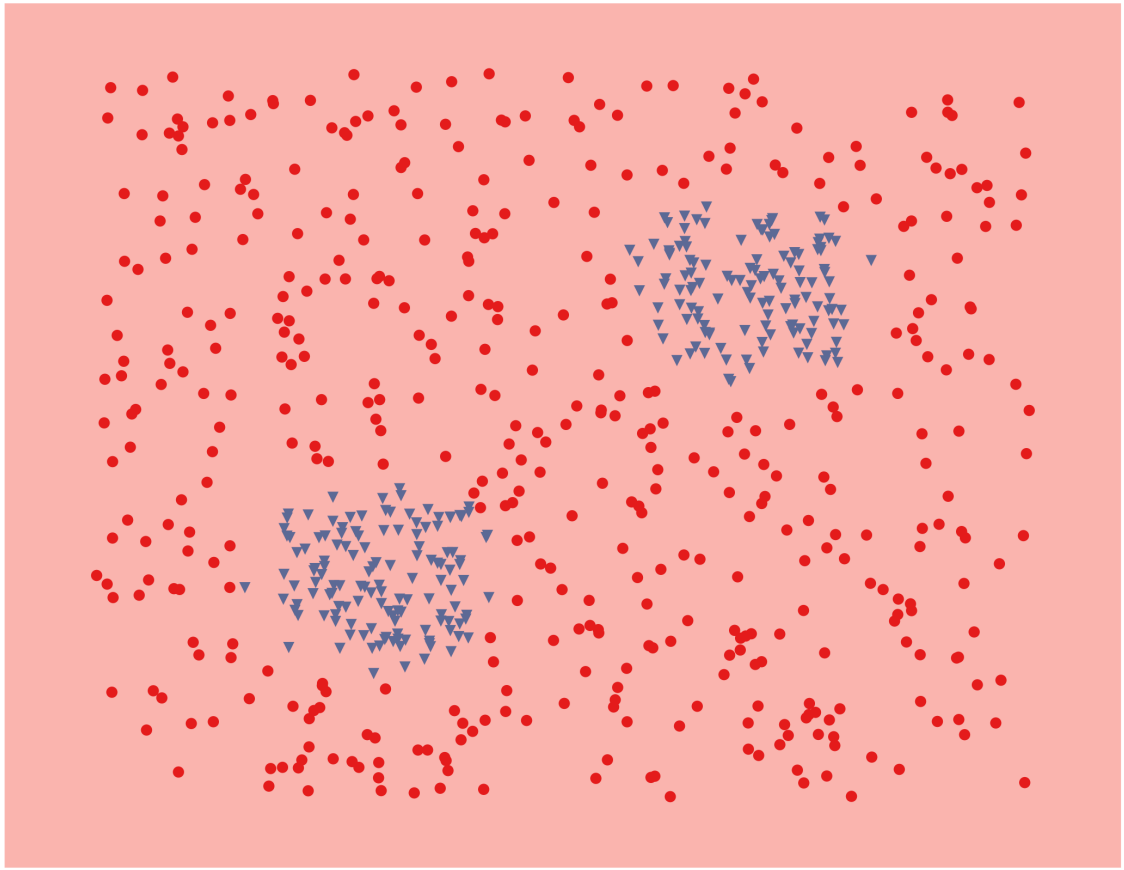
Neural networks

Example 4: Two Pockets



Neural networks

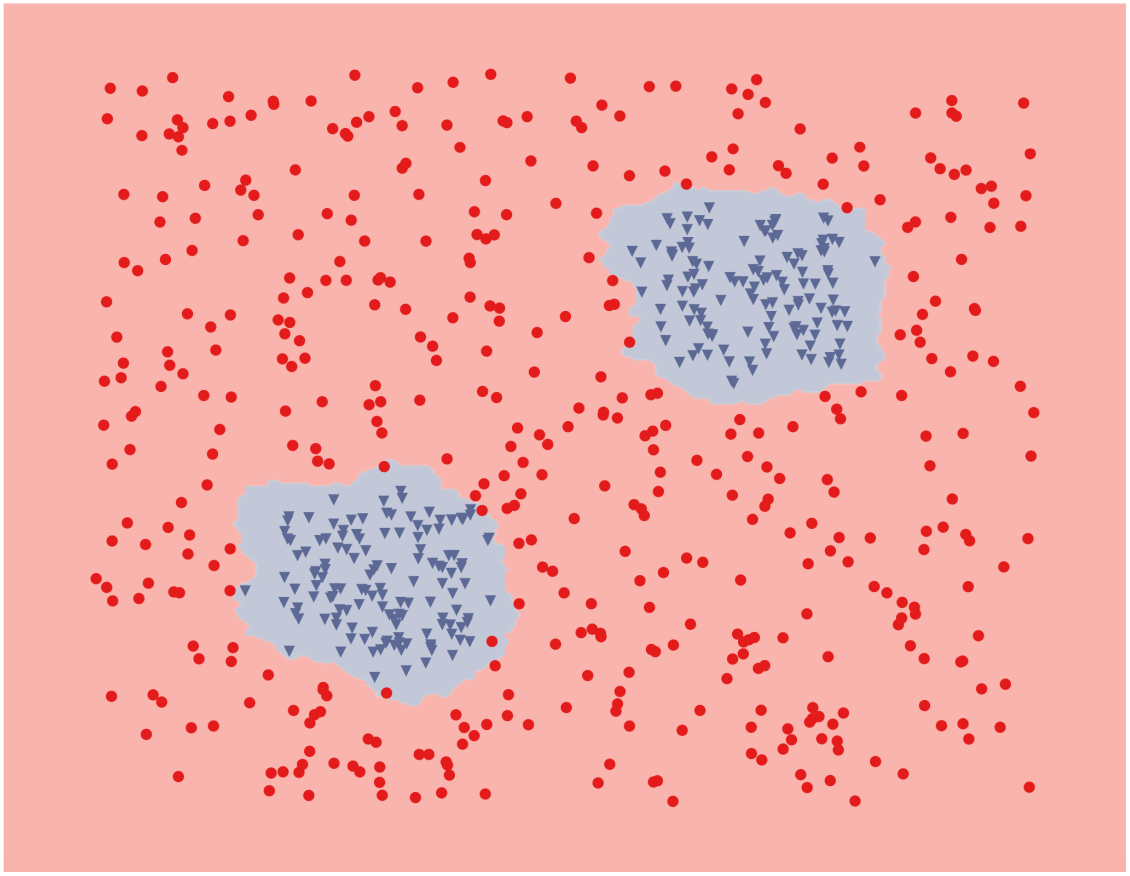
Example 4: Two Pockets



Logistic regression

Neural networks

Example 4: Two Pockets

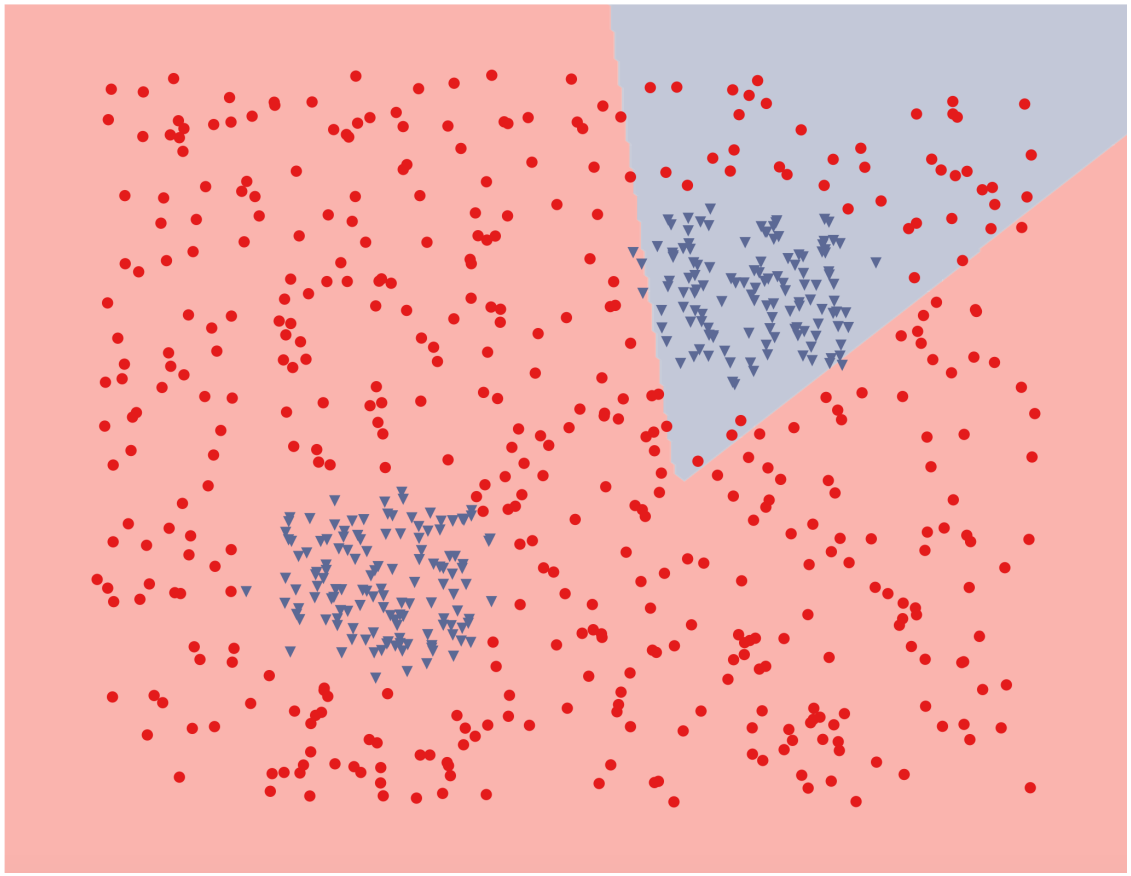


k-NN (k=5, metric=Euclidean)

http://www.cs.cmu.edu/~mgormley/courses/10601/slides/lecture11-nn.pdf

Neural networks

Example 4: Two Pockets

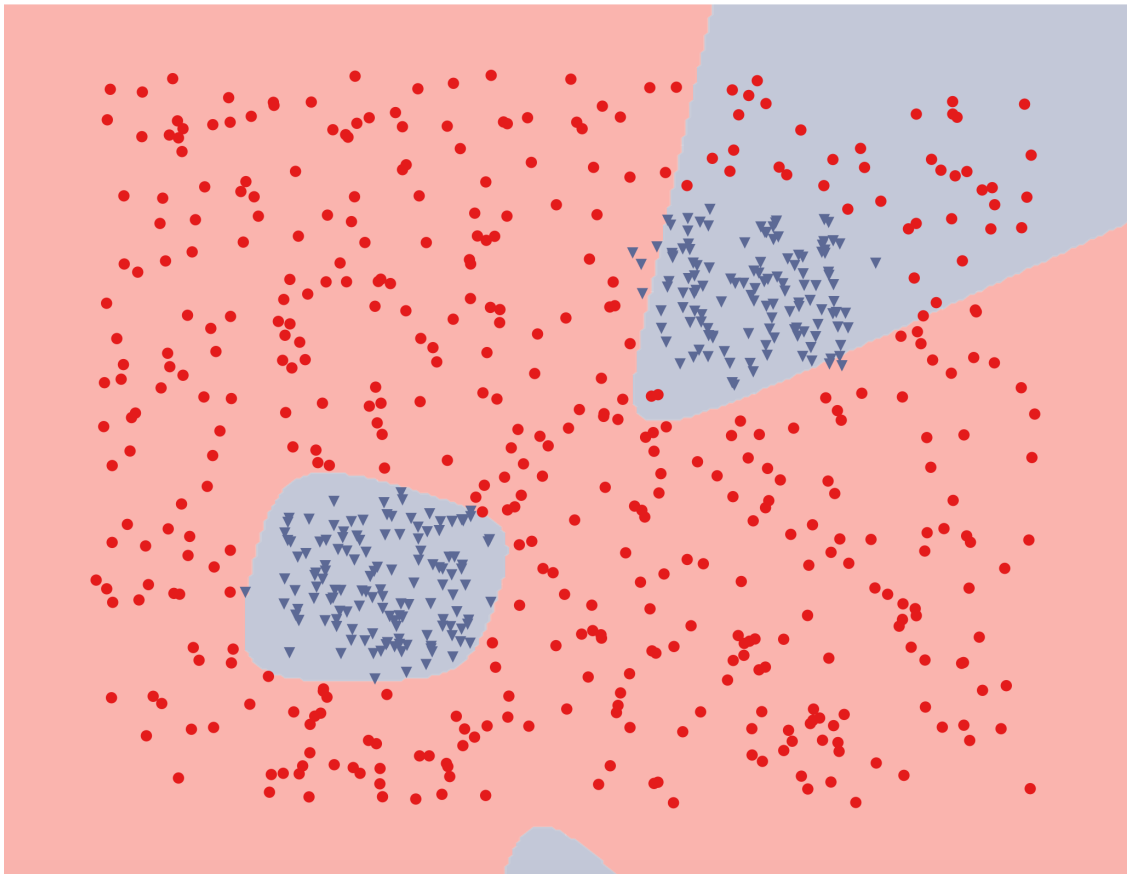


Neural network (hidden=2, activation=logistic)

http://www.cs.cmu.edu/~mgormley/courses/10601/slides/lecture11-nn.pdf

Neural networks

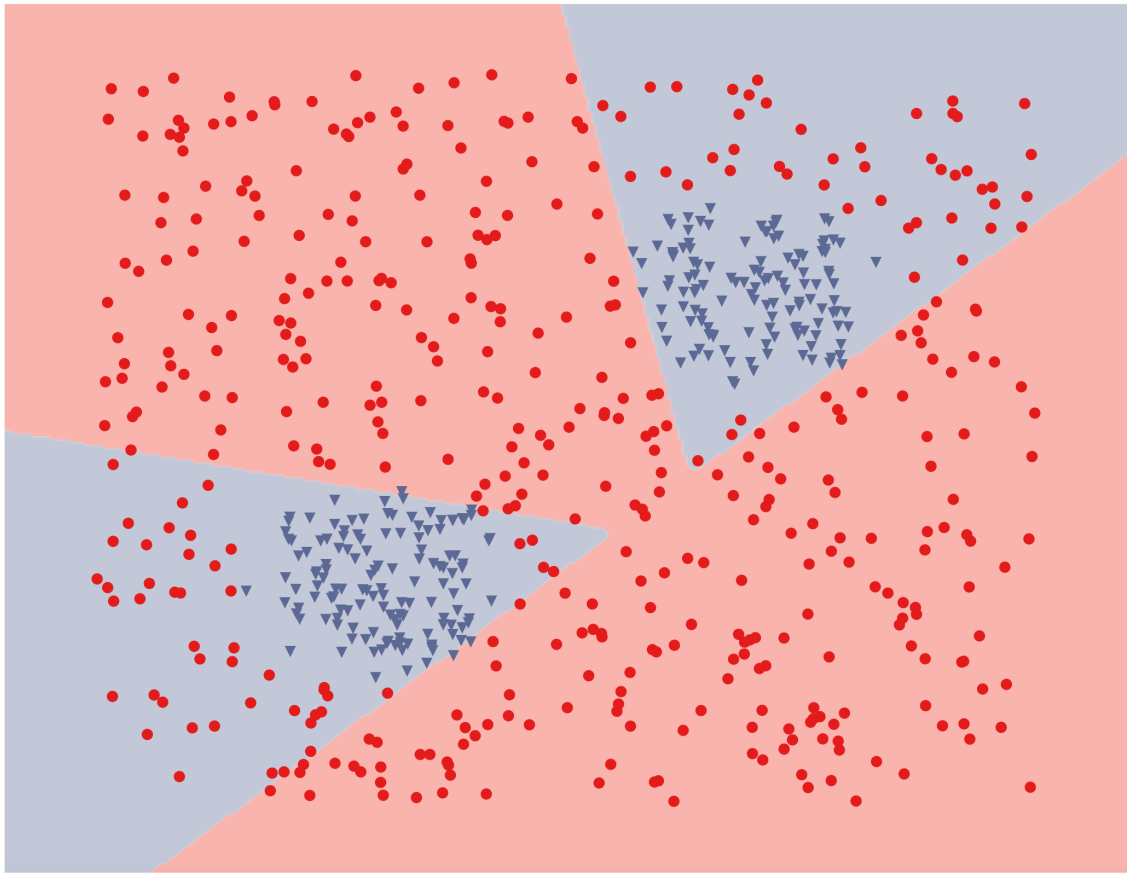
Example 4: Two Pockets



Neural network (hidden=3, activation=logistic)

Neural networks

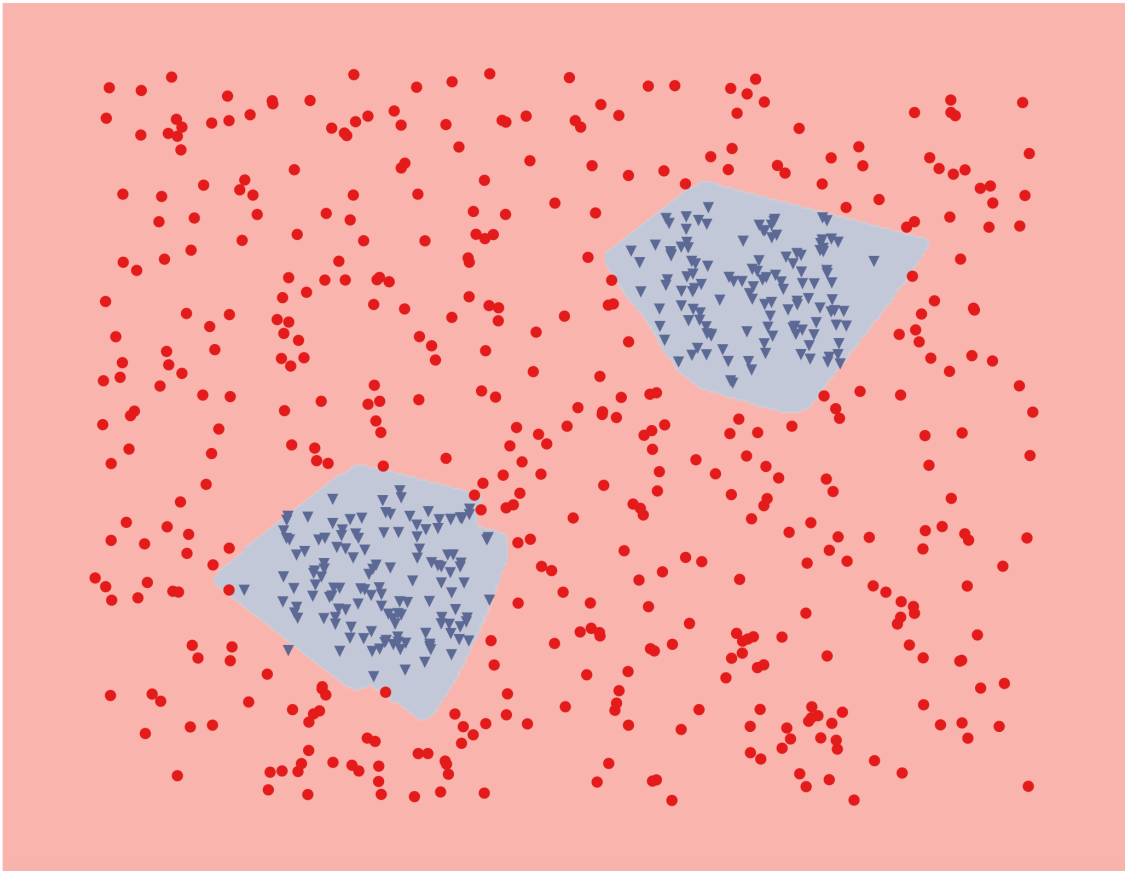
Example 4: Two Pockets



Neural network (hidden=4, activation=logistic)

Neural networks

Example 4: Two Pockets



Neural network (hidden=10, activation=logistic)

Neural networks

feature extraction → classifier → 1 (human face) vs. 0 (non human face)

After 2012
Feature learning

Machine learning

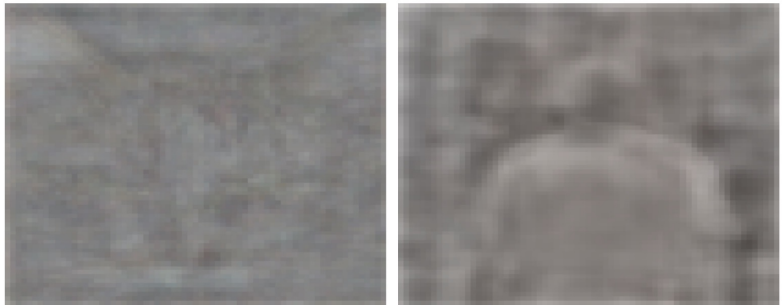
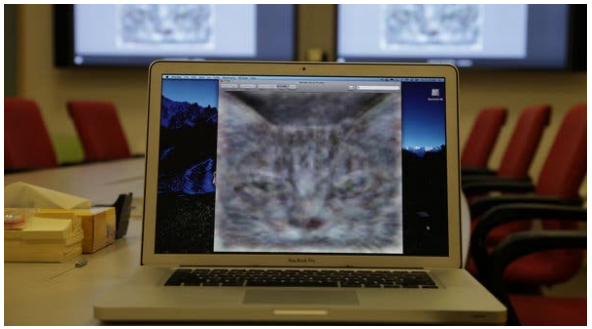


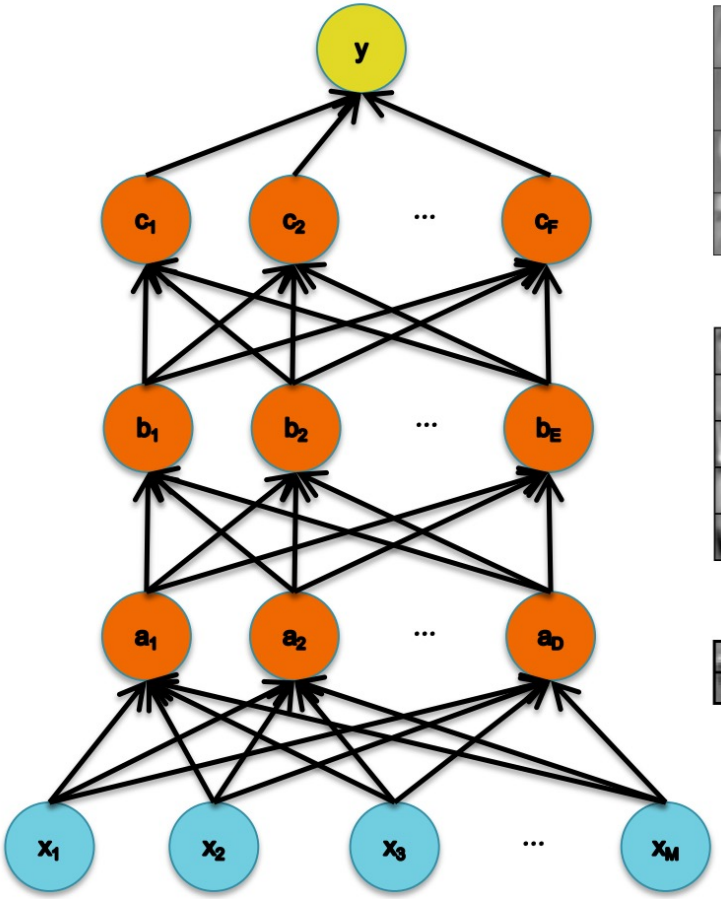
Figure 6. Visualization of the cat face neuron (left) and human body neuron (right).

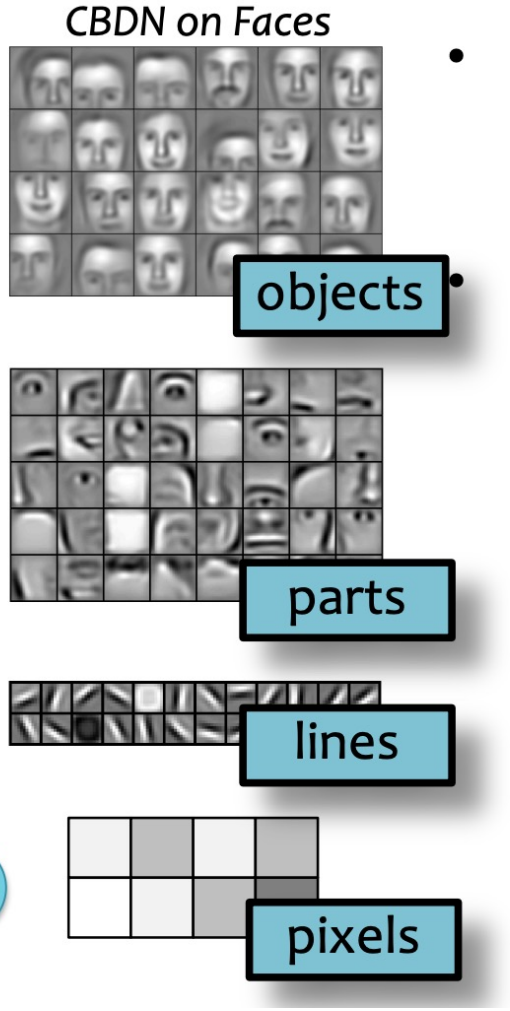


New York Times

Quoc V. Le, et al, Building High-level Features Using Large Scale Unsupervised Learning, ICML 2012
https://www.nytimes.com/2012/06/26/technology/in-a-big-network-of-computers-evidence-of-machine-learning.html?pagewanted=all

Neural networks





- **Traditional feature engineering:** build up levels of abstraction by hand
- **Deep networks** (e.g. convolution networks): learn the increasingly higher levels of abstraction from data
  - each layer is a learned feature representation
  - sophistication increases in higher layers

Quoc V. Le, et al, Building High-level Features Using Large Scale Unsupervised Learning, ICML 2012
https://www.nytimes.com/2012/06/26/technology/in-a-big-network-of-computers-evidence-of-machine-learning.html?pagewanted=all

Neural networks

How many layers should we use?

- **In theory:** A neural network with 1 hidden layer is a universal function approximator

Theorem 3. *Let σ be a continuous sigmoidal function. Let f be the decision function for any finite measurable partition of I_n. For any $\varepsilon > 0$, there is a finite sum of the form*

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma(y_j^T x + \theta_j)$$

and a set $D \subset I_n$, so that $m(D) \geq 1 - \varepsilon$ and

$$|G(x) - f(x)| < \varepsilon \qquad \text{for} \quad x \in D.$$

G. Cybenko, Approximation by Superpositions of a Sigmoidal Function, Math. Control Signals Systems (1989) 2:303-314

Neural networks

How many layers should we use?

- **Empirically:**
  - Before 2006: "Deep networks (e.g. 3 or more hidden layers) are too hard to train"
  - Before 2015: "Deep networks (e.g. 20 or more hidden layers) are too hard to train"
  - After 2015: "Deep networks (e.g. 1000 or more hidden layers) are too expensive to train"

Neural networks

Intro to Neural networks

Backpropagation

Backpropagation

Simple Example: The goal is to compute $J = \cos(\sin(x^2) + 3x^2)$ on the forward pass and the derivative $\frac{dJ}{dx}$ on the backward pass.

Forward

$J = cos(u)$

$u = u_1 + u_2$

$u_1 = sin(t)$

$u_2 = 3t$

$t = x^2$

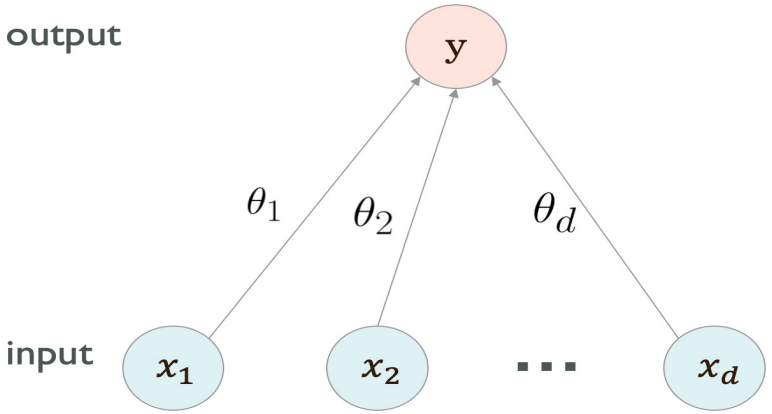
Backpropagation

Simple Example: The goal is to compute $J = \cos(\sin(x^2) + 3x^2)$ on the forward pass and the derivative $\frac{dJ}{dx}$ on the backward pass.

Forward	Backward
$J = cos(u)$	$\frac{dJ}{du} = -sin(u)$
$u = u_1 + u_2$	
$u_1 = sin(t)$	
$u_2 = 3t$	
$t = x^2$	

Backpropagation

Logistic regression



Forward

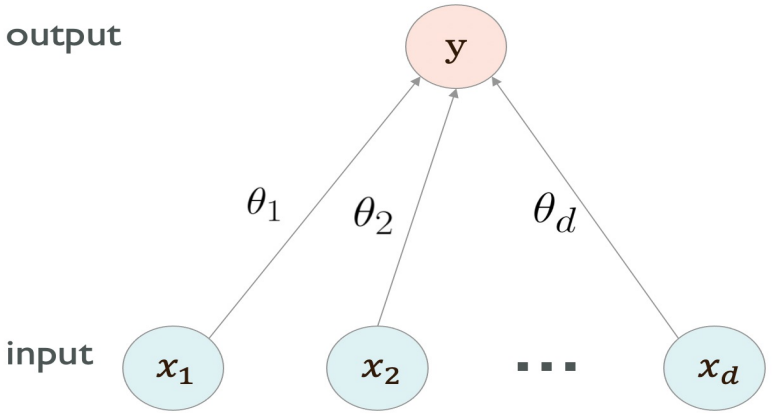
$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-a)}$$

$$a = \sum_{j=0}^{D} \theta_j x_j$$

Backpropagation

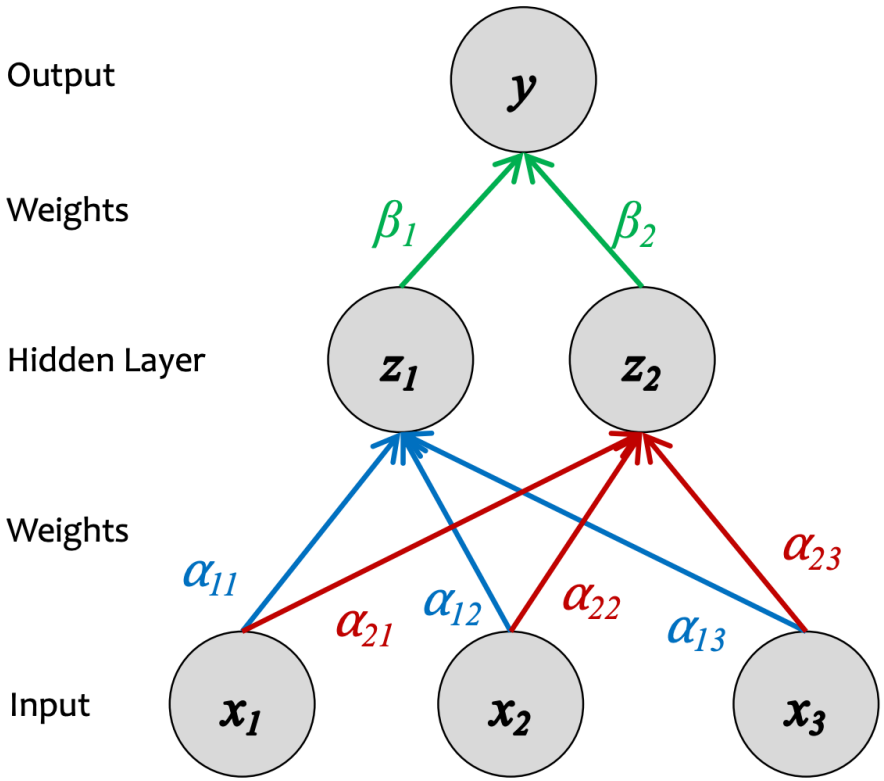
Logistic regression

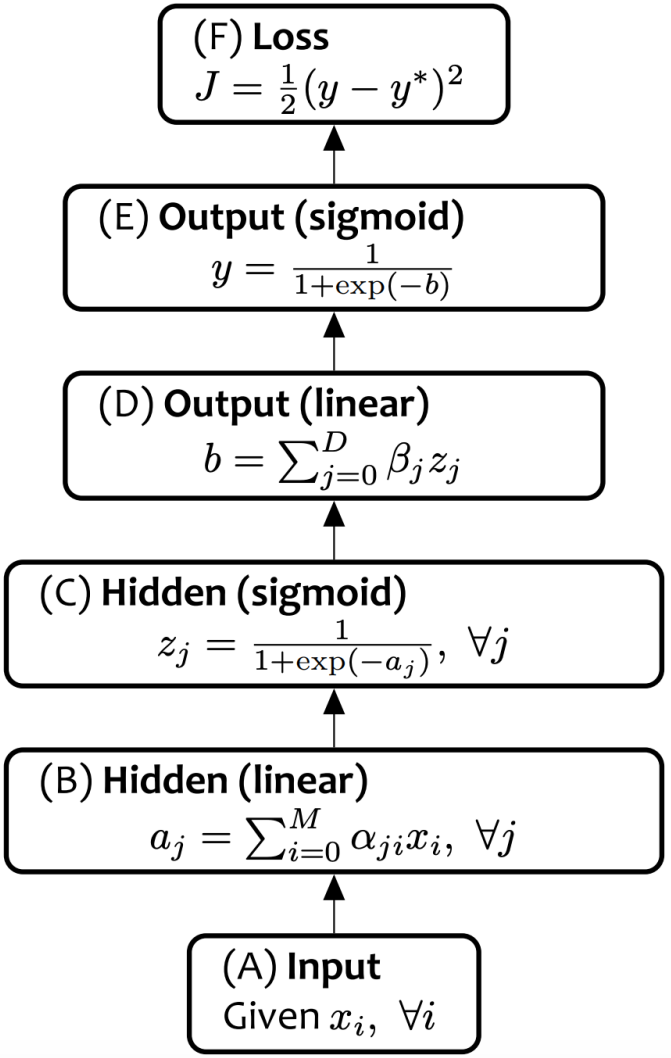


Forward	Backward
$J = y^* \log y + (1 - y^*) \log(1 - y)$	
$y = \frac{1}{1 + \exp(-a)}$	
$a = \sum_{j=0}^{D} \theta_j x_j$	

Backpropagation

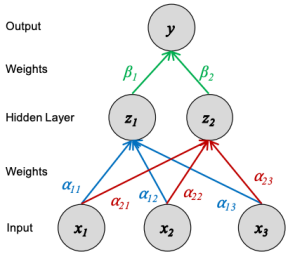
Neural networks





Backpropagation

Case 2: Neural Network



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^{D} \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$$

Backward

Backpropagation

Neural networks

Case 2:

	Forward	Backward
Loss	$J = y^* \log y + (1 - y^*) \log(1 - y)$	$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$
Sigmoid	$y = \frac{1}{1 + \exp(-b)}$	$\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{db}, \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$
Linear	$b = \sum_{j=0}^{D} \beta_j z_j$	$\frac{dJ}{d\beta_j} = \frac{dJ}{db}\frac{db}{d\beta_j}, \frac{db}{d\beta_j} = z_j$ $\frac{dJ}{dz_j} = \frac{dJ}{db}\frac{db}{dz_j}, \frac{db}{dz_j} = \beta_j$
Sigmoid	$z_j = \frac{1}{1 + \exp(-a_j)}$	$\frac{dJ}{da_j} = \frac{dJ}{dz_j}\frac{dz_j}{da_j}, \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$
Linear	$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$	$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \frac{da_j}{d\alpha_{ji}} = x_i$ $\frac{dJ}{dx_i} = \sum_{j=0}^{D} \frac{dJ}{da_j}\frac{da_j}{dx_i}, \frac{da_j}{dx_i} = \alpha_{ji}$

Backpropagation

Algorithm 1 Gradient Descent (GD)

procedure GD(Training data $\mathcal{D}$, test data $\mathcal{D}_t$)
  Initialize parameters $\boldsymbol{\alpha}, \boldsymbol{\beta}$
  for $e \in \{1, 2, \ldots, E\}$ **do**
    for $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$ **do**
      Compute neural network layers:
      $\mathbf{o} = \texttt{object}(\mathbf{x}, \mathbf{a}, \mathbf{b}, \mathbf{z}, \hat{\mathbf{y}}, J) = \text{NNFORWARD}(\mathbf{x}, \mathbf{y}, \boldsymbol{\alpha}, \boldsymbol{\beta})$
      Compute gradients via backprop:
      $\left.\begin{array}{l} \mathbf{g}_{\boldsymbol{\alpha}} = \nabla_{\boldsymbol{\alpha}} J \\ \mathbf{g}_{\boldsymbol{\beta}} = \nabla_{\boldsymbol{\beta}} J \end{array}\right\} = \text{NNBACKWARD}(\mathbf{x}, \mathbf{y}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{o})$
      Update parameters:
      $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} - \gamma \mathbf{g}_{\boldsymbol{\alpha}}$
      $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \gamma \mathbf{g}_{\boldsymbol{\beta}}$
    Evaluate training mean cross-entropy $J_{\mathcal{D}}(\boldsymbol{\alpha}, \boldsymbol{\beta})$
    Evaluate test mean cross-entropy $J_{\mathcal{D}_t}(\boldsymbol{\alpha}, \boldsymbol{\beta})$
  return parameters $\boldsymbol{\alpha}, \boldsymbol{\beta}$

feed-forward

Backpropagation

Algorithm 1 Gradient Descent (GD)

procedure GD(Training data $\mathcal{D}$, test data $\mathcal{D}_t$)
  Initialize parameters $\boldsymbol{\alpha}, \boldsymbol{\beta}$
  for $e \in \{1, 2, \ldots, E\}$ **do**
    for $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$ **do**
      Compute neural network layers:
      $\mathbf{o} = \texttt{object}(\mathbf{x}, \mathbf{a}, \mathbf{b}, \mathbf{z}, \hat{\mathbf{y}}, J) = \text{NNFORWARD}(\mathbf{x}, \mathbf{y}, \boldsymbol{\alpha}, \boldsymbol{\beta})$
      Compute gradients via backprop:
      $\left.\begin{array}{l} \mathbf{g}_{\boldsymbol{\alpha}} = \nabla_{\boldsymbol{\alpha}} J \\ \mathbf{g}_{\boldsymbol{\beta}} = \nabla_{\boldsymbol{\beta}} J \end{array}\right\} = \text{NNBACKWARD}(\mathbf{x}, \mathbf{y}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{o})$
      Update parameters:
      $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} - \gamma \mathbf{g}_{\boldsymbol{\alpha}}$
      $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \gamma \mathbf{g}_{\boldsymbol{\beta}}$
    Evaluate training mean cross-entropy $J_{\mathcal{D}}(\boldsymbol{\alpha}, \boldsymbol{\beta})$
    Evaluate test mean cross-entropy $J_{\mathcal{D}_t}(\boldsymbol{\alpha}, \boldsymbol{\beta})$
  return parameters $\boldsymbol{\alpha}, \boldsymbol{\beta}$

backward

Backpropagation

Algorithm 1 Gradient Descent (GD)

procedure GD(Training data $\mathcal{D}$, test data $\mathcal{D}_t$)
  Initialize parameters $\boldsymbol{\alpha}, \boldsymbol{\beta}$
  for $e \in \{1, 2, \ldots, E\}$ **do**
    for $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$ **do**
      Compute neural network layers:
      $\mathbf{o} = \texttt{object}(\mathbf{x}, \mathbf{a}, \mathbf{b}, \mathbf{z}, \hat{\mathbf{y}}, J) = \text{NNFORWARD}(\mathbf{x}, \mathbf{y}, \boldsymbol{\alpha}, \boldsymbol{\beta})$
      Compute gradients via backprop:
      $\left.\begin{array}{l} \mathbf{g}_{\boldsymbol{\alpha}} = \nabla_{\boldsymbol{\alpha}} J \\ \mathbf{g}_{\boldsymbol{\beta}} = \nabla_{\boldsymbol{\beta}} J \end{array}\right\} = \text{NNBACKWARD}(\mathbf{x}, \mathbf{y}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{o})$

update

      Update parameters:
      $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} - \gamma \mathbf{g}_{\boldsymbol{\alpha}}$
      $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \gamma \mathbf{g}_{\boldsymbol{\beta}}$
    Evaluate training mean cross-entropy $J_{\mathcal{D}}(\boldsymbol{\alpha}, \boldsymbol{\beta})$
    Evaluate test mean cross-entropy $J_{\mathcal{D}_t}(\boldsymbol{\alpha}, \boldsymbol{\beta})$
  return parameters $\boldsymbol{\alpha}, \boldsymbol{\beta}$

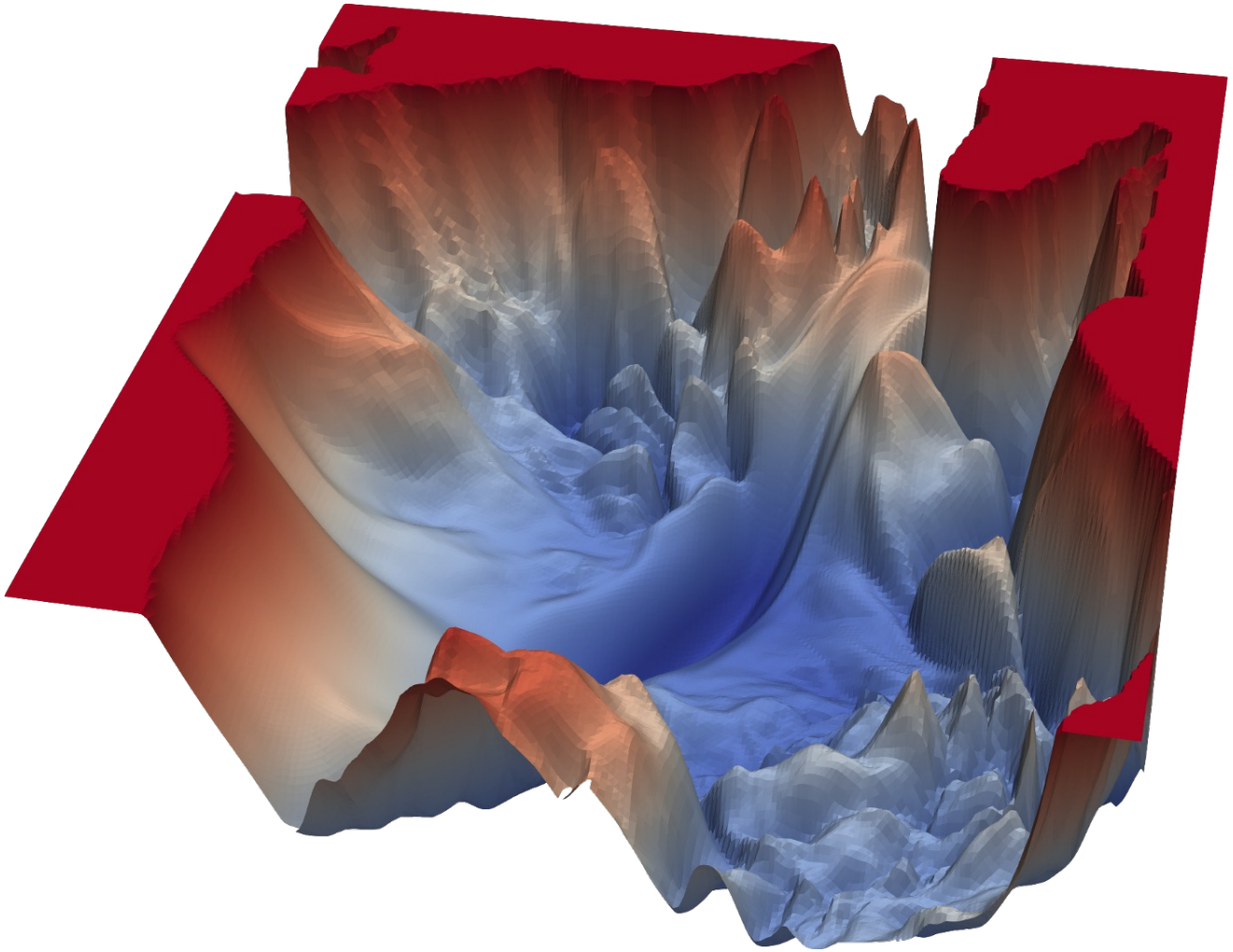
Backpropagation

Algorithm 1 Gradient Descent (GD)

procedure GD(Training data $\mathcal{D}$, test data $\mathcal{D}_t$)
  Initialize parameters $\boldsymbol{\alpha}, \boldsymbol{\beta}$
  for $e \in \{1, 2, \ldots, E\}$ **do**
    for $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$ **do**
      Compute neural network layers:
      $\mathbf{o} = \texttt{object}(\mathbf{x}, \mathbf{a}, \mathbf{b}, \mathbf{z}, \hat{\mathbf{y}}, J) = \text{NNFORWARD}(\mathbf{x}, \mathbf{y}, \boldsymbol{\alpha}, \boldsymbol{\beta})$
      Compute gradients via backprop:
      $$\left.\begin{aligned} \mathbf{g}_{\boldsymbol{\alpha}} &= \nabla_{\boldsymbol{\alpha}} J \\ \mathbf{g}_{\boldsymbol{\beta}} &= \nabla_{\boldsymbol{\beta}} J \end{aligned}\right\} = \text{NNBACKWARD}(\mathbf{x}, \mathbf{y}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{o})$$
      Update parameters:
      $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} - \gamma \mathbf{g}_{\boldsymbol{\alpha}}$
      $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \gamma \mathbf{g}_{\boldsymbol{\beta}}$
    Evaluate training mean cross-entropy $J_{\mathcal{D}}(\boldsymbol{\alpha}, \boldsymbol{\beta})$
    Evaluate test mean cross-entropy $J_{\mathcal{D}_t}(\boldsymbol{\alpha}, \boldsymbol{\beta})$
  return parameters $\boldsymbol{\alpha}, \boldsymbol{\beta}$

evaluation

Next lecture: Training neural networks



Thank you very much!

sihengc@sjtu.edu.cn