Deep Learning for Computer Vision

113-1/Fall 2024

https://cool.ntu.edu.tw/courses/41702 (NTU COOL)

http://vllab.ee.ntu.edu.tw/dlcv.html (Public website)

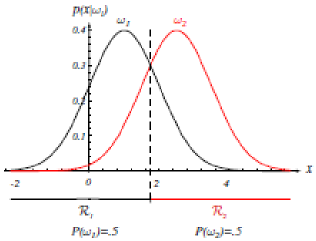
Yu-Chiang Frank Wang 王鈺強, Professor

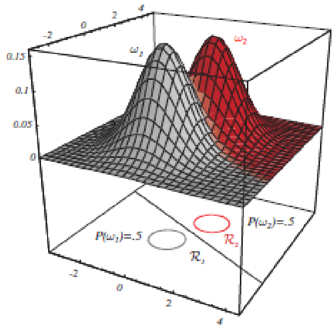
Dept. Electrical Engineering, National Taiwan University

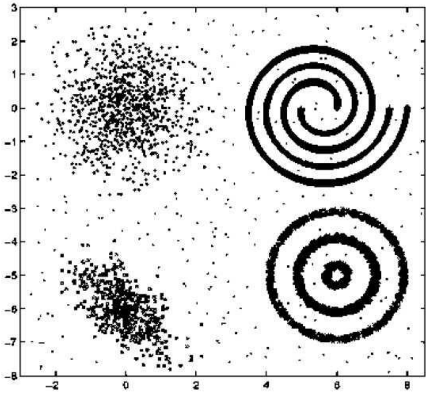
2024/09/03

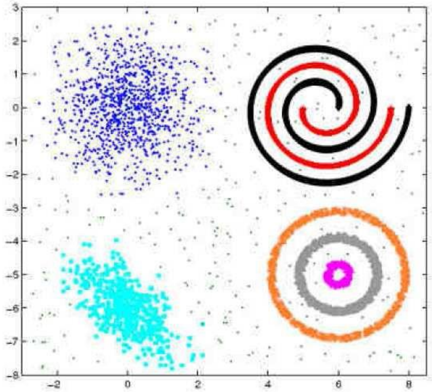
What's to Be Covered in This Lecture…

- Unsupervised vs. Supervised Learning
  - Clustering & Dimension Reduction
  - Training, testing, & validation
  - Linear Classification
  - From Linear Classifier to Neural Nets



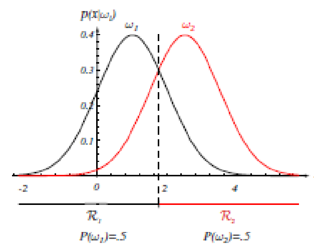


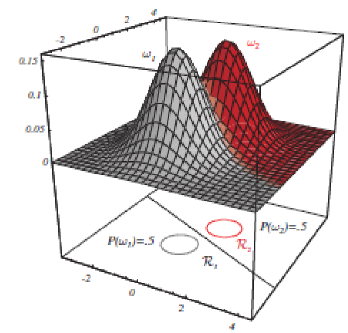


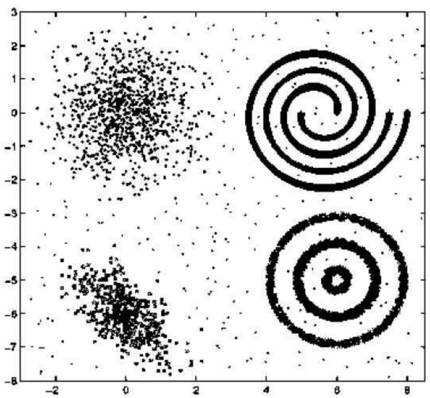


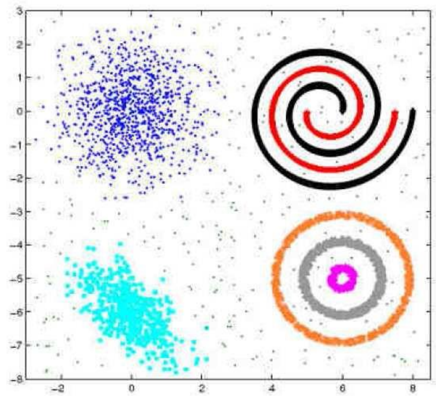
What’s to Be Covered Today…

- Unsupervised vs. Supervised Learning
    - Clustering & Dimension Reduction
    - Training, testing, & validation
    - Linear Classification
    - From Linear Classifier to Neural Nets

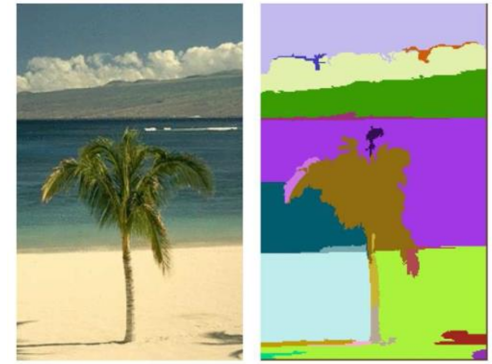




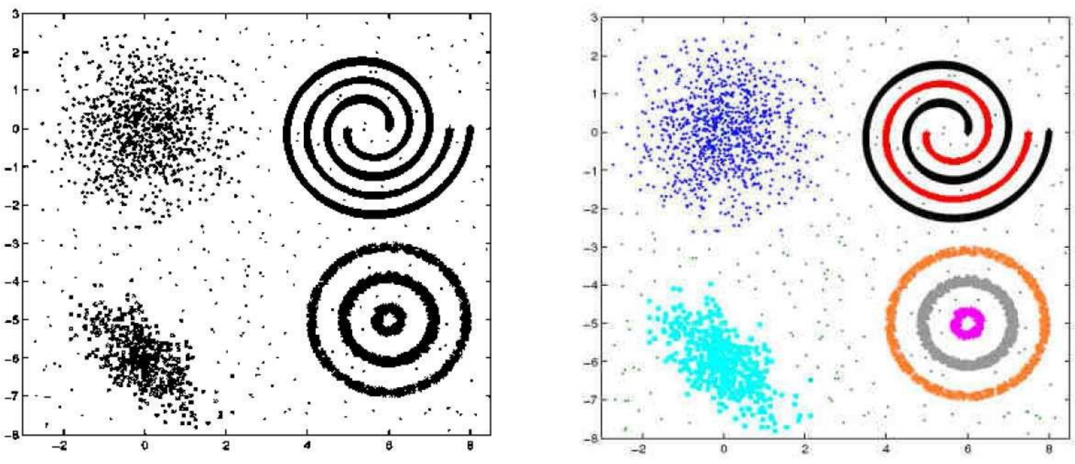




Clustering



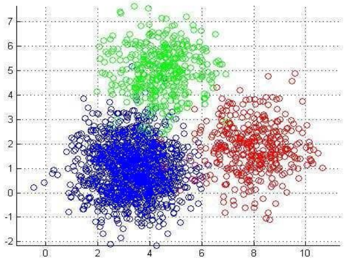
- Clustering is an unsupervised algorithm.
  - Given:
    a set of N unlabeled instances {$\boldsymbol{x}_1$, …, $\boldsymbol{x}_N$}; # of clusters K
  - Goal: group the samples into K partitions
- Remarks:
  - High within-cluster (intra-cluster) similarity
  - Low between-cluster (inter-cluster) similarity
  - But…how to determine a proper similarity measure?

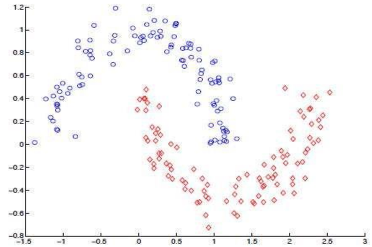


(a) Input data (b) Desired clustering

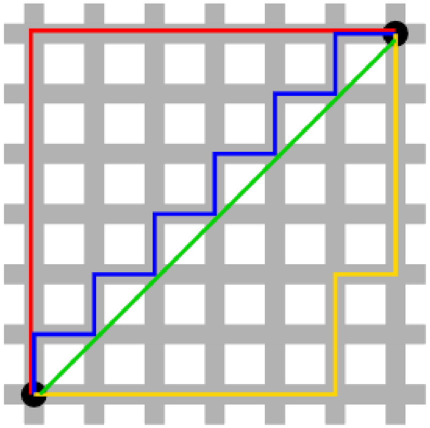
Similarity is NOT Always Objective…

Clustering (cont'd)





- Similarity:
  - A key component/measure to perform data clustering
  - Inversely proportional to distance
  - Example distance metrics:
    - Euclidean distance (L2 norm): $d(x,z) = \|x - z\|_2 = \sqrt{\sum_{i=1}^{D}(x_i - z_i)^2}$
    - Manhattan distance (L1 norm): $d(x,z) = \|x - z\|_1 = \sum_{i=1}^{D}|x_i - z_i|$
    - Note that *p*-norm of *x* is denoted as:

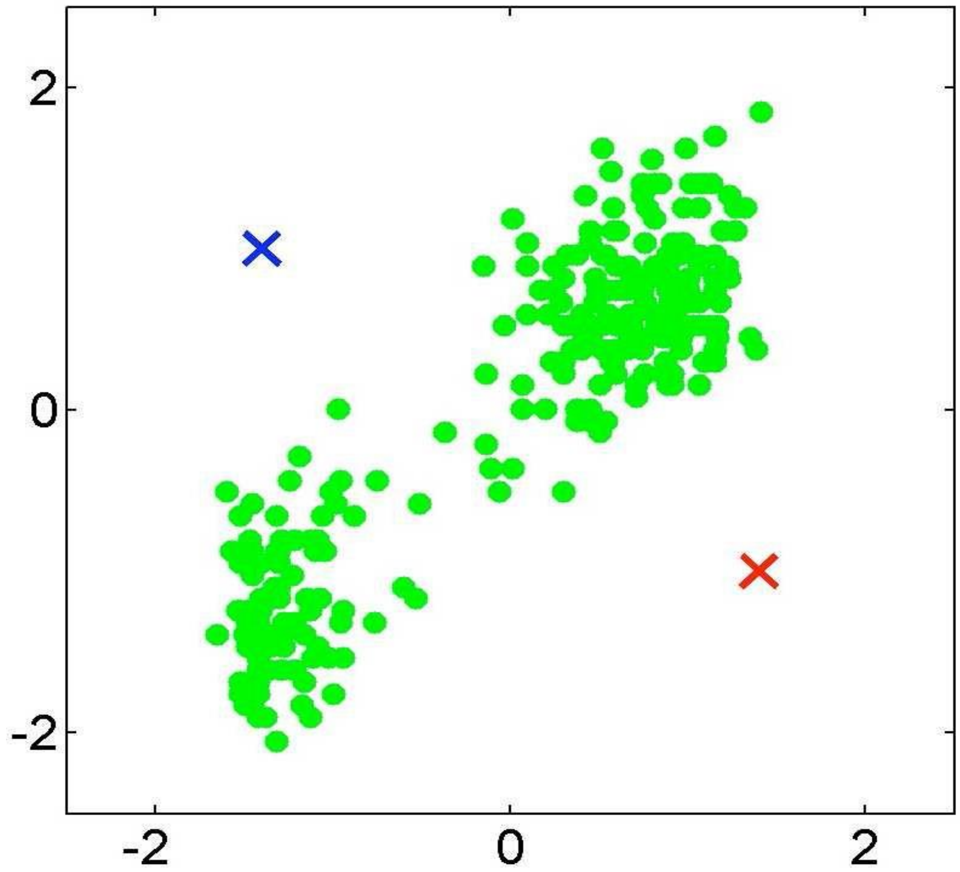


K-Means Clustering

- Input: N examples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ ($\boldsymbol{x}_n \in \mathrm{R}^D$); number of partitions K
- Initialize: K cluster centers $\mu_1, \ldots, \mu_K$. Several initialization options:
  - Randomly initialize $\mu_1, \ldots, \mu_K$ anywhere in R^D
  - Or, simply choose any K examples as the cluster centers
- Iterate:
  - Assign each of example $\boldsymbol{x}_\mathrm{n}$ to its closest cluster center
  - Recompute the new cluster centers μ_k (mean/centroid of the set C_k)
  - Repeat while not converge
- Possible convergence criteria:
  - Cluster centers do not change anymore
  - Max. number of iterations reached
- Output:
  - K clusters (with centers/means of each cluster)

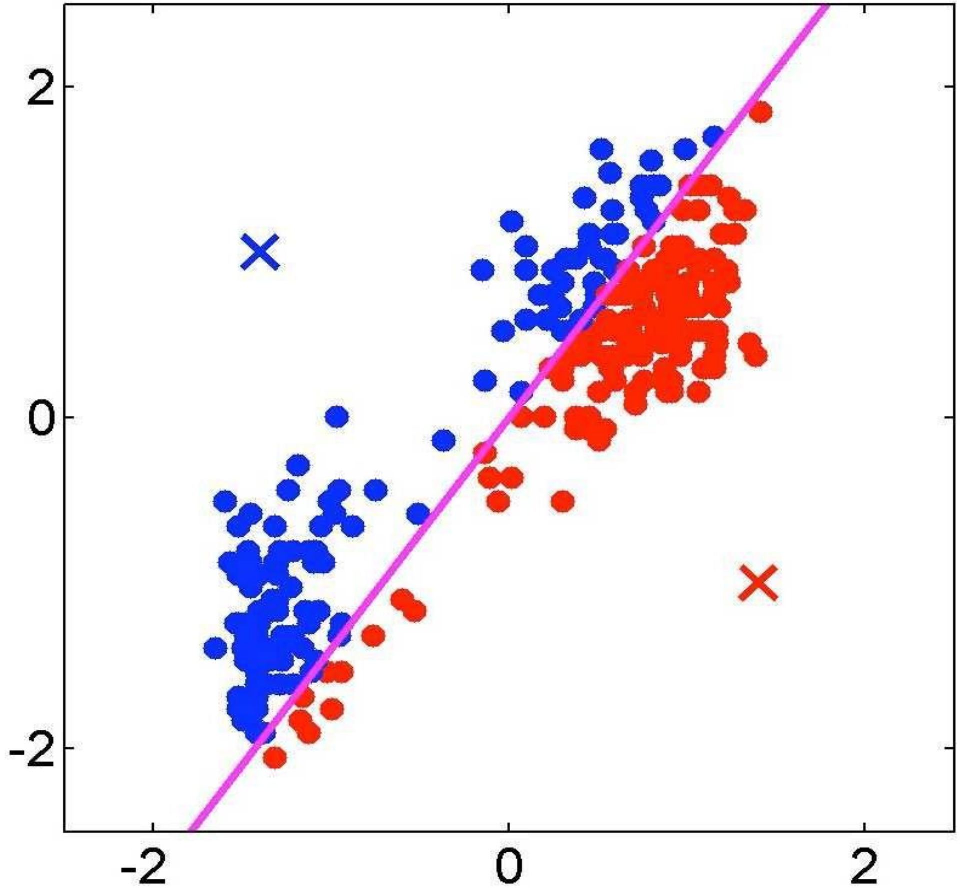
K-Means Clustering

- Example (K = 2): Initialization, iteration #1: pick cluster centers



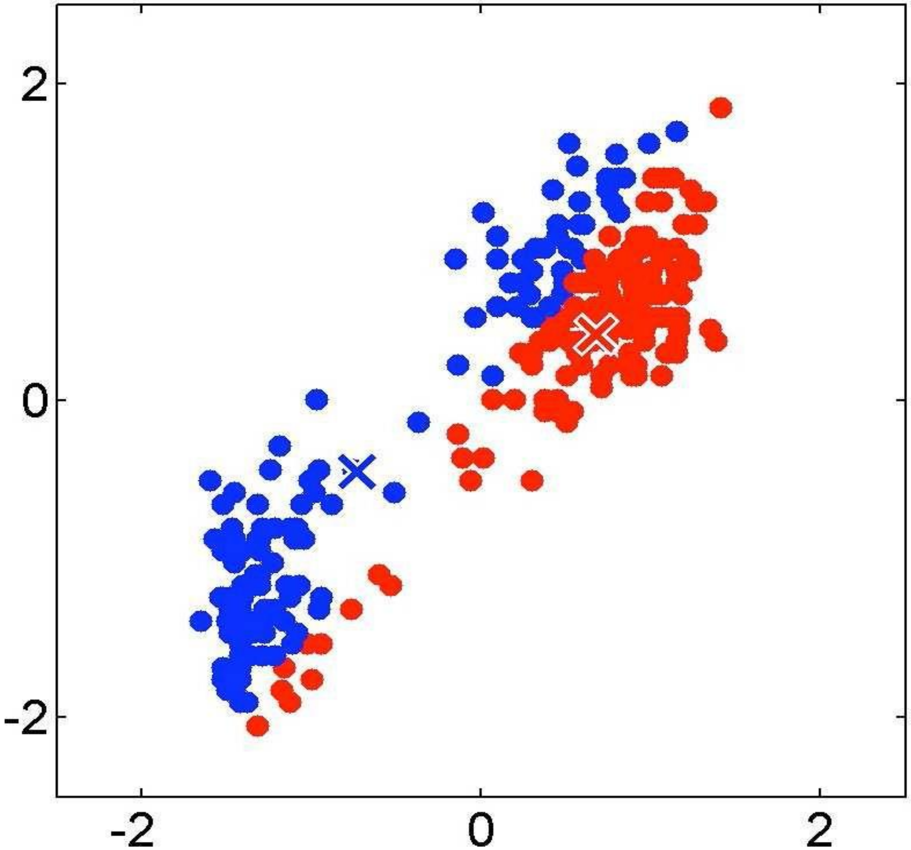
K-Means Clustering

- Example (K = 2): iteration #1-2, assign data to each cluster



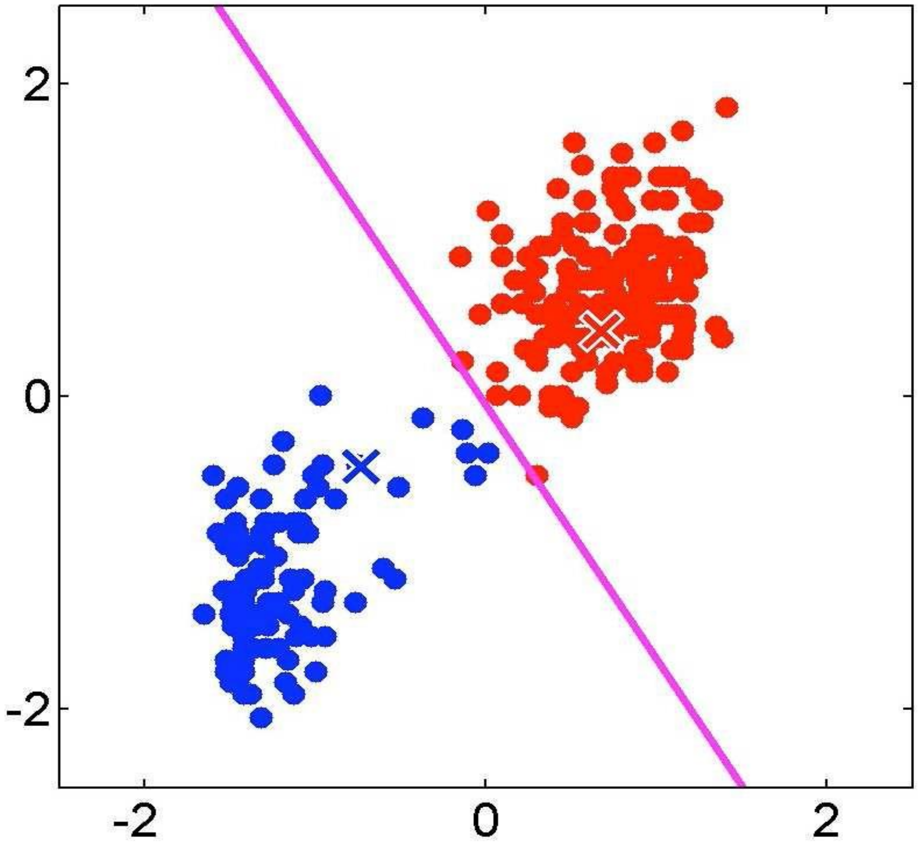
K-Means Clustering

- Example (K = 2): iteration #2-1, update cluster centers



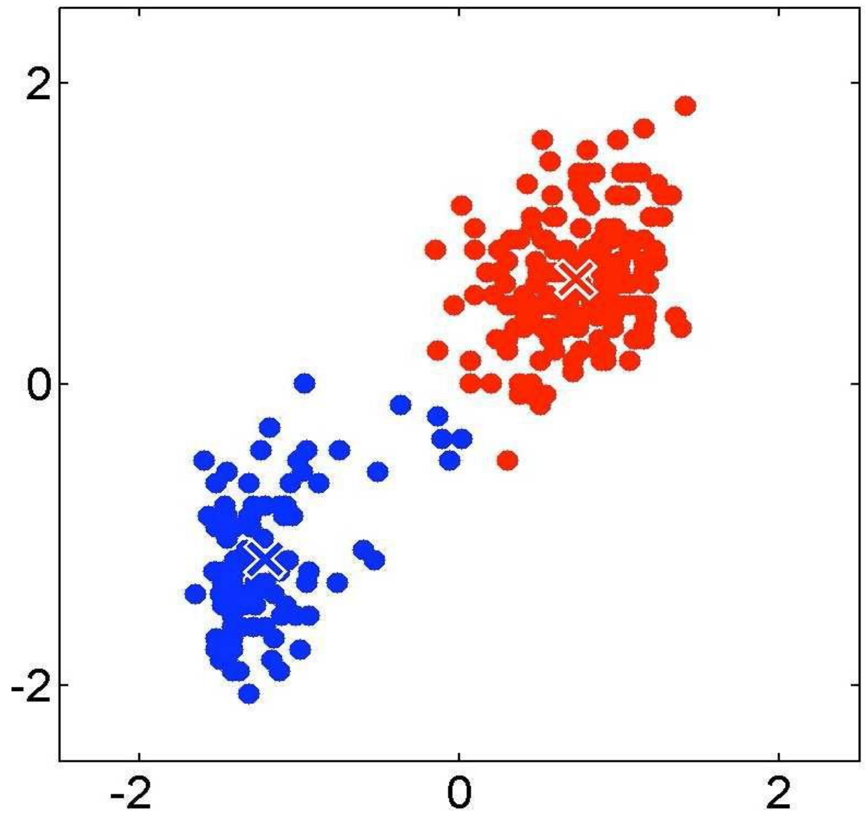
K-Means Clustering

- Example (K = 2): iteration #2, assign data to each cluster



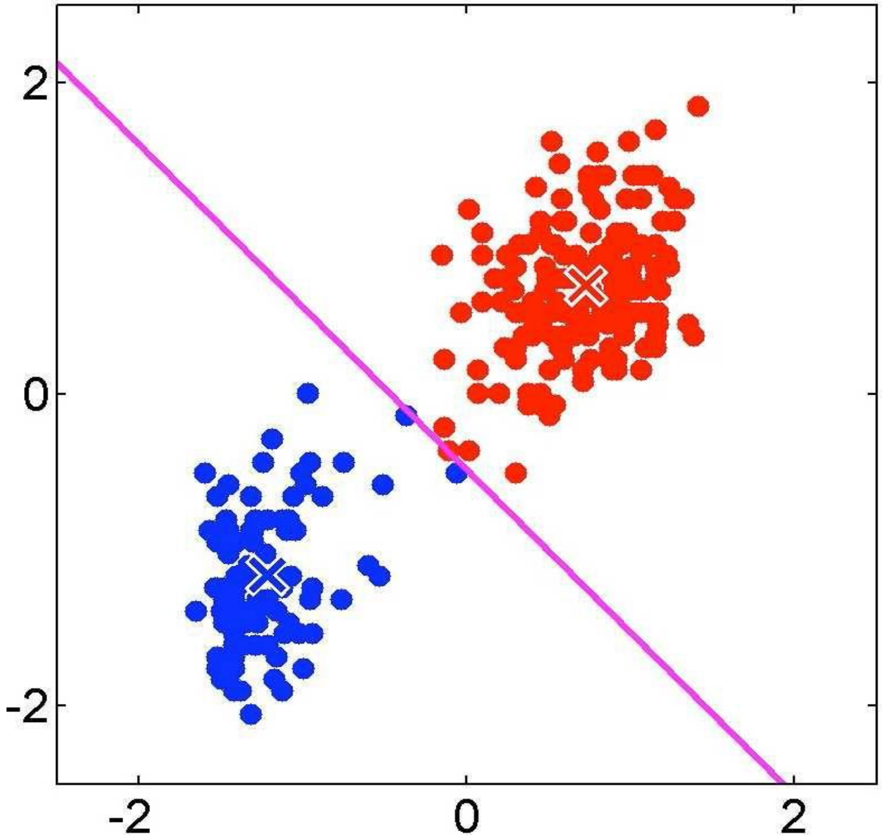
K-Means Clustering

- Example (K = 2): iteration #3-1



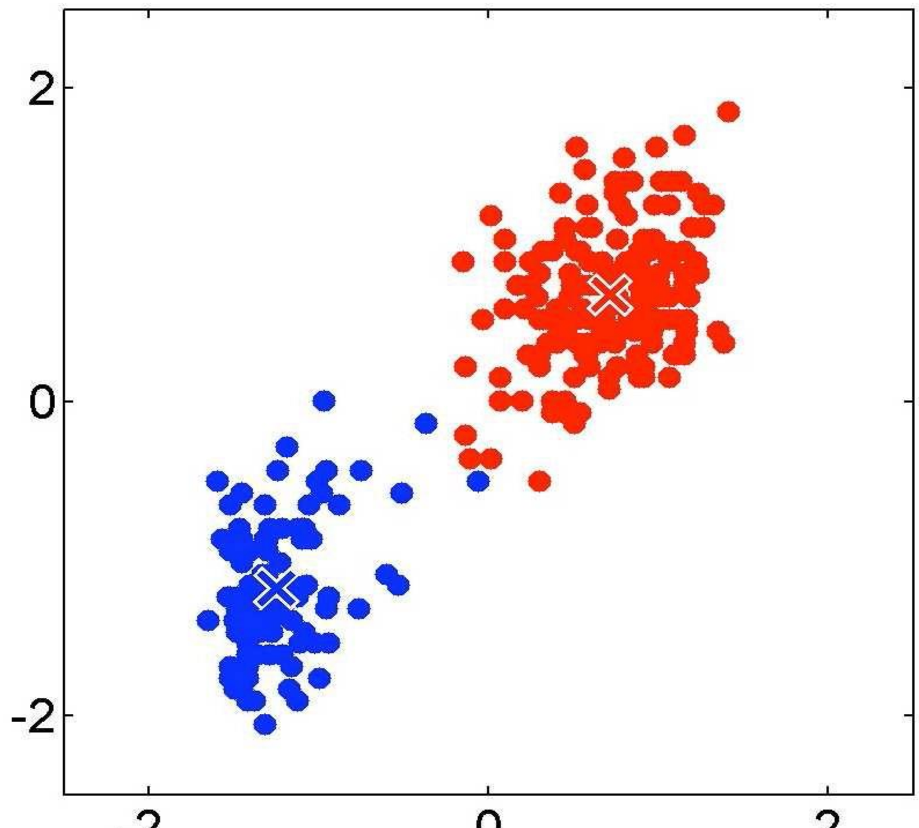
K-Means Clustering

- Example (K = 2): iteration #3-2



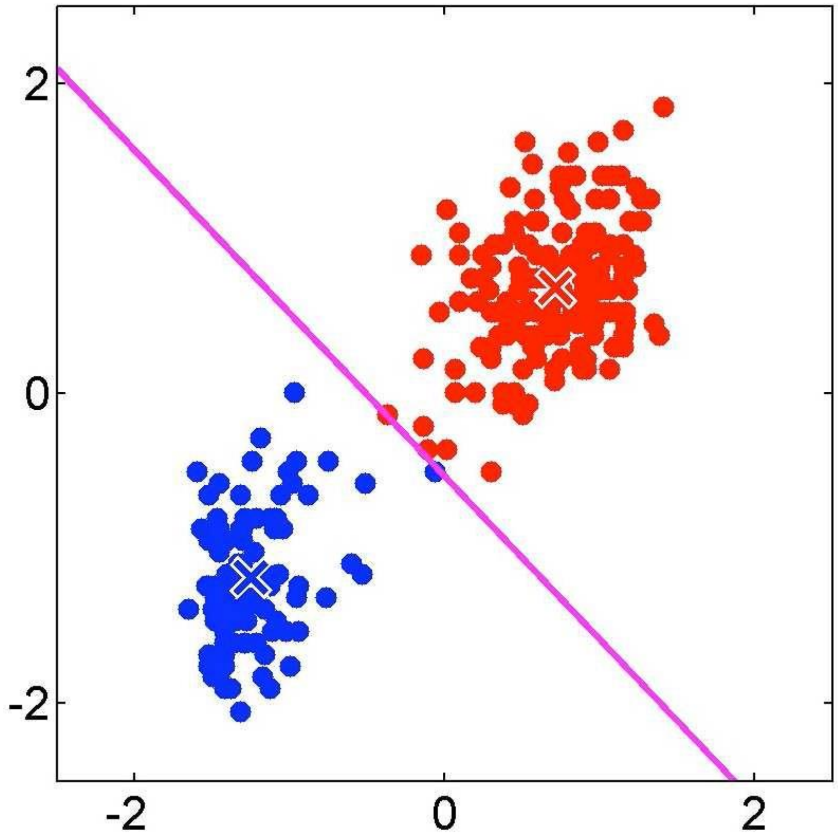
K-Means Clustering

- Example (K = 2): iteration #4-1



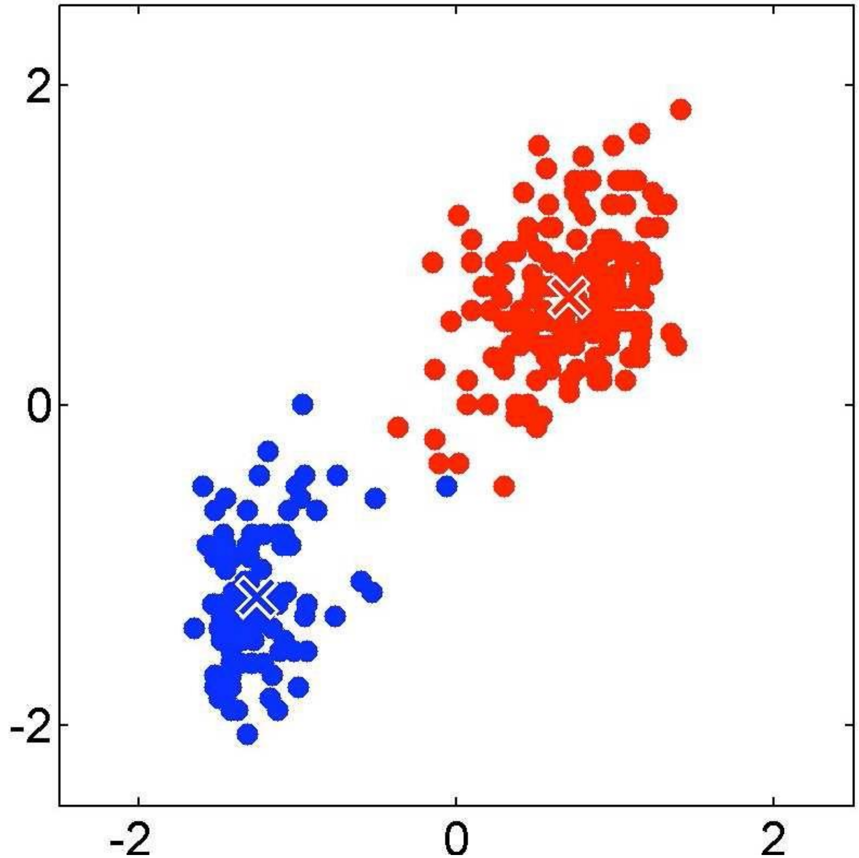
K-Means Clustering

- Example (K = 2): iteration #4-2



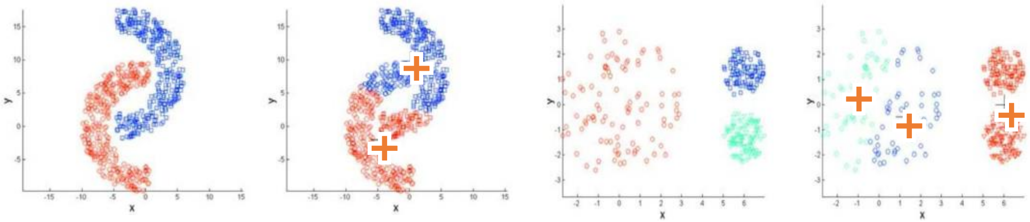
K-Means Clustering

- Example (K = 2): iteration #5, cluster means are not changed.



K-Means Clustering (cont'd)

- Easy to implement, but…
  - Preferable for round shaped clusters with similar sizes

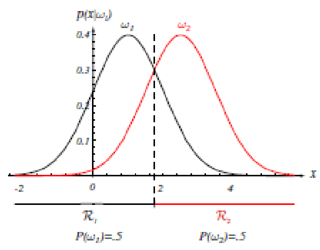


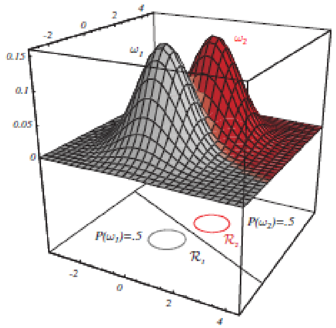
  - Limitations
    - Sensitive to initialization →
    - Sensitive to outliers →
    - Hard assignment only →

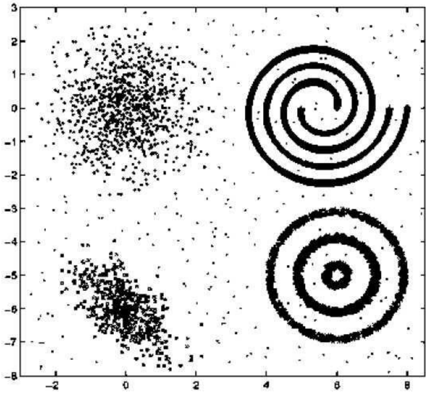
- Remarks
  - Expectation-maximization (EM) algorithm
  - Speed-up possible by hierarchical clustering (e.g., 100 = 10^2 clusters)

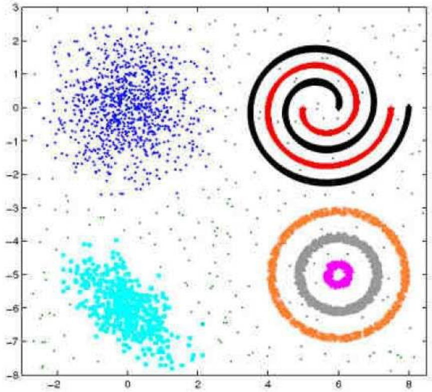
What's to Be Covered in This Lecture…

- Unsupervised vs. Supervised Learning
  - Clustering & Dimension Reduction
  - Training, testing, & validation
  - Linear Classification
  - From Linear Classifier to Neural Nets





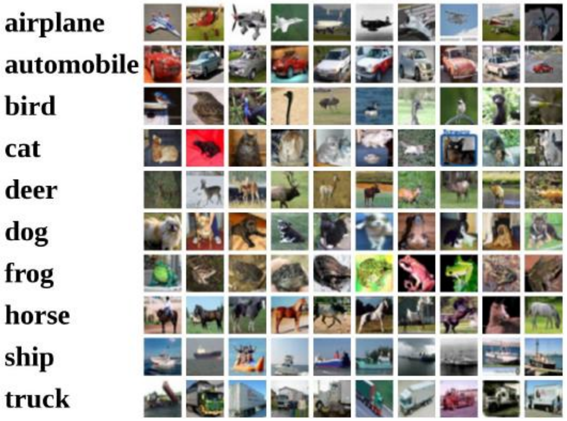




Linear Classification

- Linear Classifier
  - Consider that we have 10 object categories of interest
    - E.g., CIFAR10 with 50K training & 10K test images of 10 categories. And, each image is of size 32 x 32 x 3 pixels.

Linear Classification (cont’d)



- Linear Classifier
  - Let’s take the input image as **x**, and the linear classifier as **W**.
    We need **y** = **Wx** + **b** as a 10-dimensional output vector, indicating the score for each class.

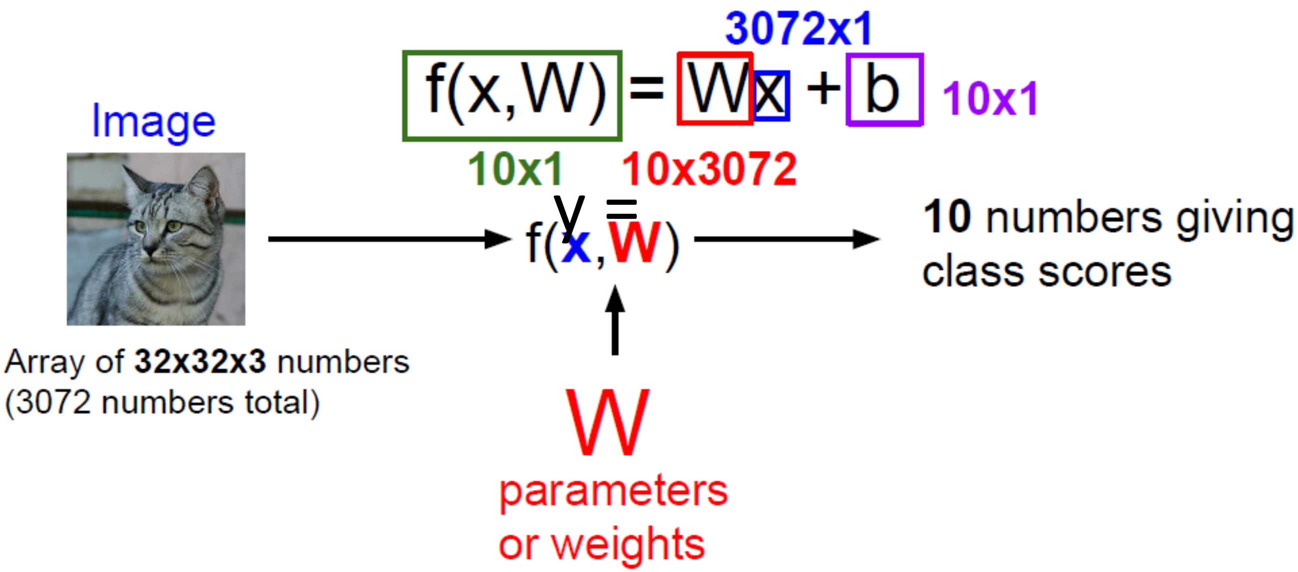


Image credit: Stanford CS231n

Linear Classification (cont'd)

- Linear Classifier
  - For example, an image with 2 x 2 pixels & 3 classes of interest we need to learn a linear classifier **W** (plus a bias **b**), so that desirable outputs **y** = **Wx** + **b** can be expected.

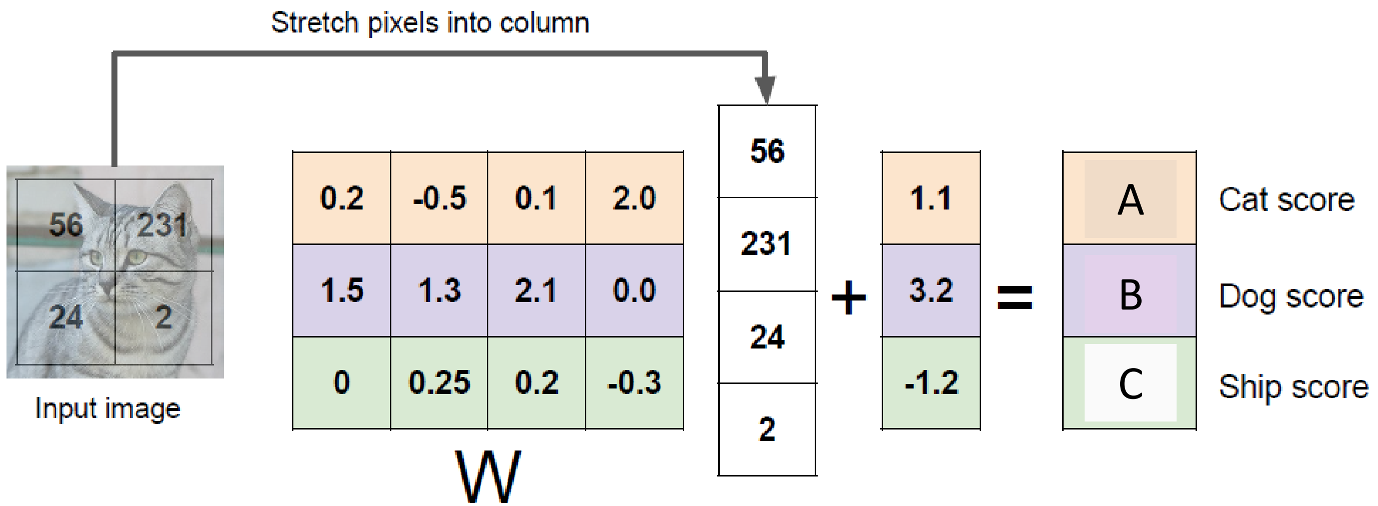
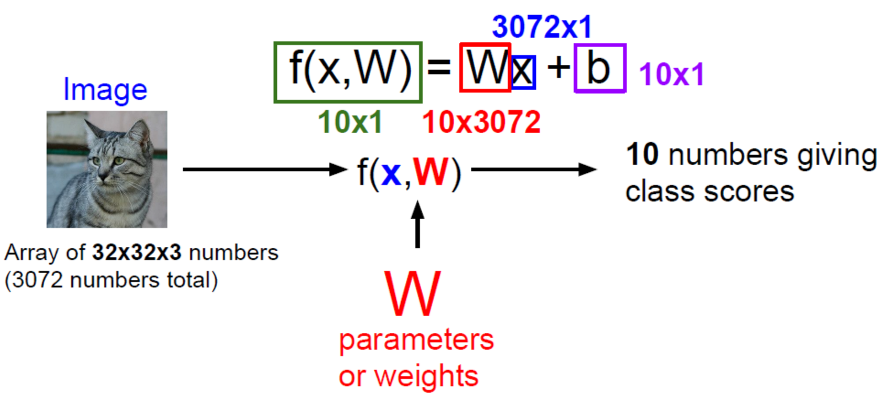
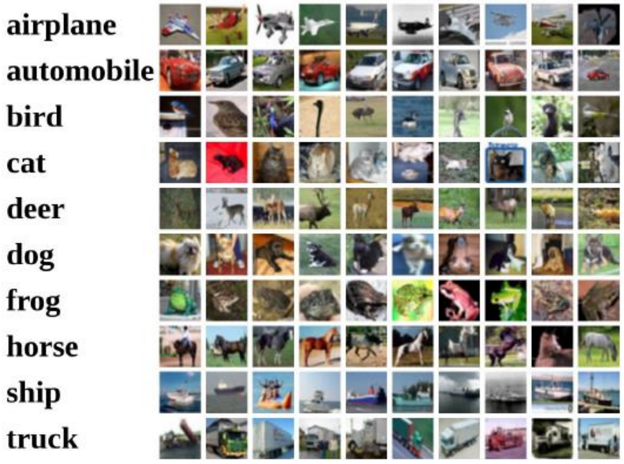


Image credit: Stanford CS231n

Final Remarks

- Interpreting **W** in **y** = **Wx** + **b**
  - Once **W** is learned for predicting **y**,
    each row in **W** can be viewed as an exemplar of the corresponding class.
  - Recall that, **Wx** basically performs inner product (or correlation) between the input **x** and the exemplar of each class -> *similarity* matters!
  - Any potential problem or limitation?





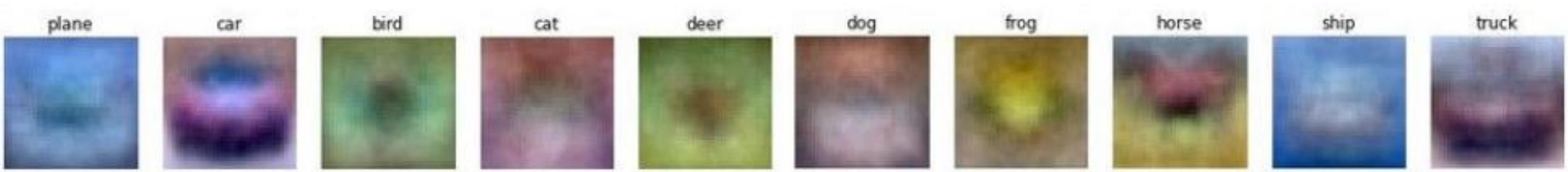


Image credit: Stanford CS231n

Loss Function (or Cost/Objective Function)

- **Loss is a function of model parameter W**
  - Tells us how good/bad our learned model **W** in **y** = **Wx** + **b** is.
    If loss function, the lower, the better!
  - Given a labeled dataset $\{(x_i, y_i)\}_{i=1}^{N}$
    where **x** and **y** indicate the input instance and its label, respectively,

    Loss of a single input instance is denoted as $L_i(f(x_i, W), y_i)$,

    and that for the entire dataset is the sum or average of per-instance losses:

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i) \; .$$

  - In practice, calculating full sum for L is expensive.
    - Approximate sum using a **minibatch** of instances (e.g., 32, 64, 128 samples, etc.)

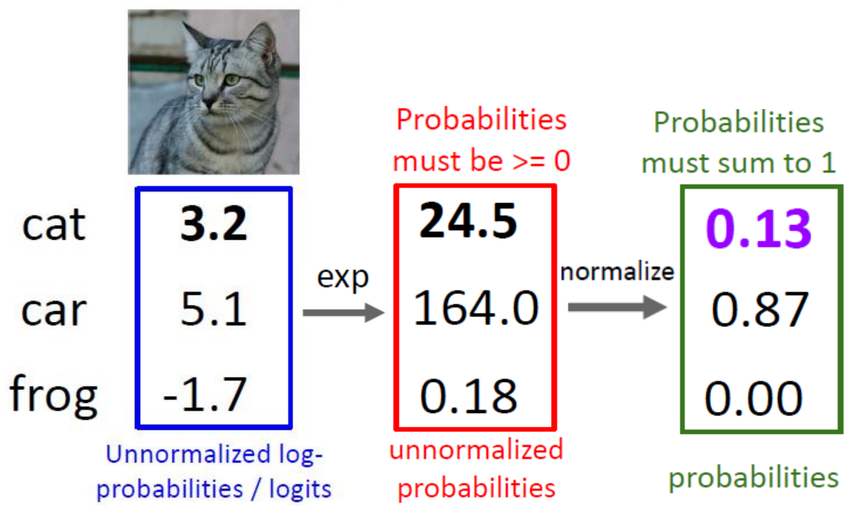
Slide credit: UMich EECS 498-007

Loss Function (cont'd)

- **Cross-Entropy Loss** (Multinomial Logistic Regression)
  - Interpret the classifier scores as **probabilities**
  - **Softmax function:**

$$P(Y = k \mid X = x_i) = \frac{\exp(s_k)}{\sum_j \exp(s_j)}$$ with $s = f(x_i; W)$ as the classifier output for input $\mathbf{x}_i$

  - See example below:



$$L_i = -\log P(Y = y_i \mid X = x_i)$$

$L_{cat} = -\log(0.13) = 2.04$

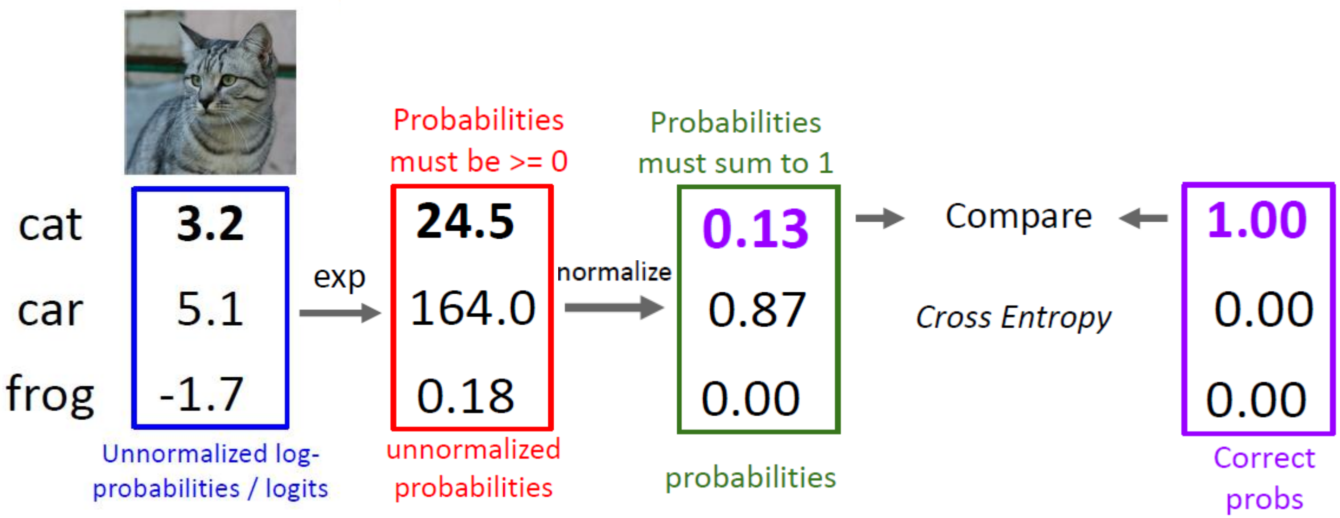
What about this L?
What are its possibly min/max value??

- **Cross-Entropy Loss** (cont'd)
  - **Softmax function:**

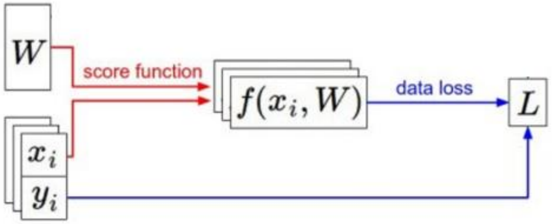
    $P(Y = k \mid X = x_i) = \frac{\exp(s_k)}{\sum_j \exp(s_j)}$ with $s = f(x_i; W)$ as the classifier output for input $\mathbf{x}_i$

    ➡ $L_i = -\log P(Y = y_i \mid X = x_i)$ or $L_i = -\log\left(\frac{\exp(s_{y_i})}{\sum_j \exp(s_j)}\right)$

  - **(Binary) Cross Entropy Loss** (or $\mathbf{L_{BCE}}$; see example below):

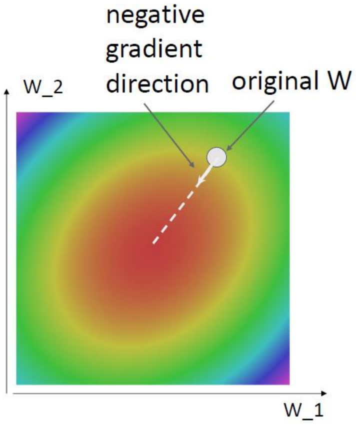


Slide credit: UMich EECS 498-007



- **Learning W from L_{BCE}**
  - Computing **gradients**:
    Following the slope to reach the (hopefully global) minimum for **W**.
  - **Gradient Descent** via numeric or analytic gradients:
    - Iteratively step in the direction of the negative gradient & search for **W**
    - Hyperparameters: weight initialization, # of steps, learning rate, etc.

```
# Vanilla gradient descent
w = initialize_weights()
for t in range(num_steps):
  dw = compute_gradient(loss_fn, data, w)
  w -= learning_rate * dw
```



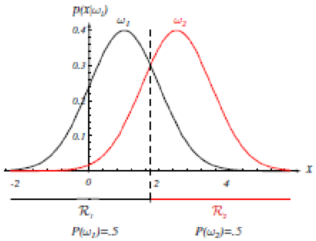
  - **Stochastic Gradient Descent**
    - Full sum in L is expensive when large N
    - *Approximate* sum using a minibatch of instances (e.g., 32, 64, 128, etc.)
    - Additional hyperparameters of batch size and data sampling

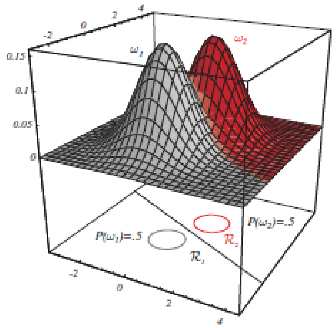
```
# Stochastic gradient descent
w = initialize_weights()
for t in range(num_steps):
  minibatch = sample_data(data, batch_size)
  dw = compute_gradient(loss_fn, minibatch, w)
  w -= learning_rate * dw
```

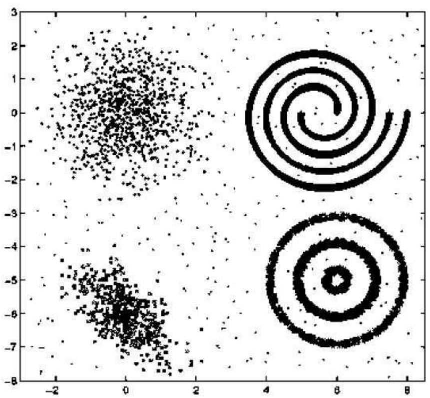
Slide credit: UMich EECS 498-007

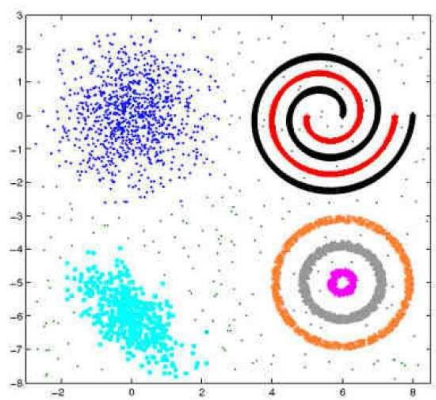
What's to Be Covered in This Lecture…

- Unsupervised vs. Supervised Learning
  - Clustering & Dimension Reduction
  - Training, testing, & validation
  - Linear Classification
  - From Linear Classifier to Neural Nets



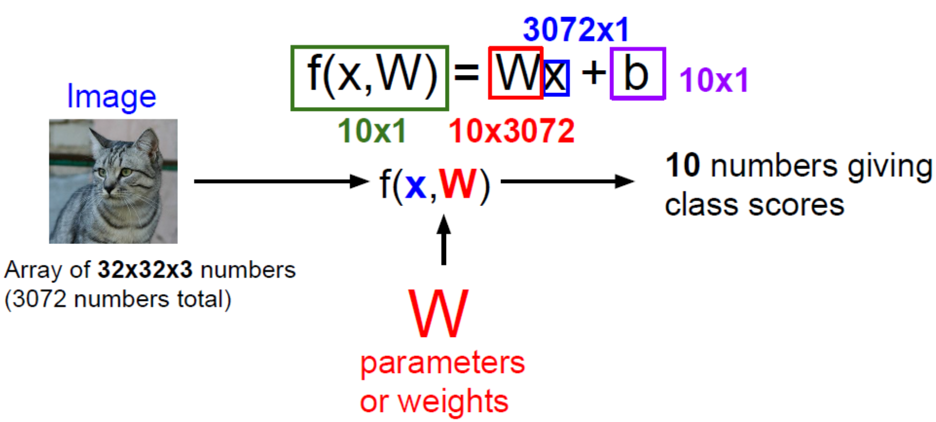






Recap: Linear Classification

- Linear Classifier
  - Let's take the input image as **x**, and the linear classifier as **W**.
    We need **y** = **Wx** + **b** as a 10-dimensional output vector, indicating the score for each class.
  - For example, an image with 2 x 2 pixels & 3 classes of interest
    we need to learn a linear classifier **W** (plus a bias **b**),
    so that desirable outputs **y** = **Wx** + **b** can be expected.



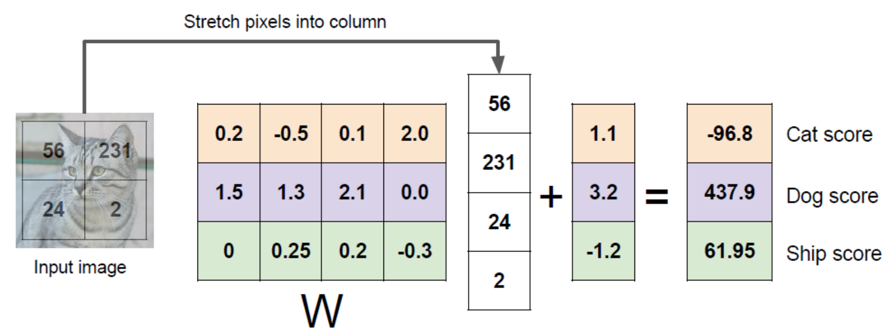
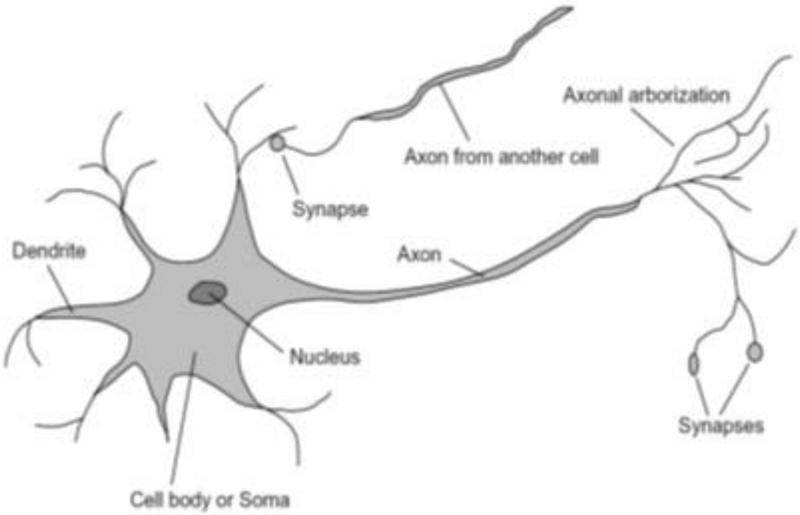
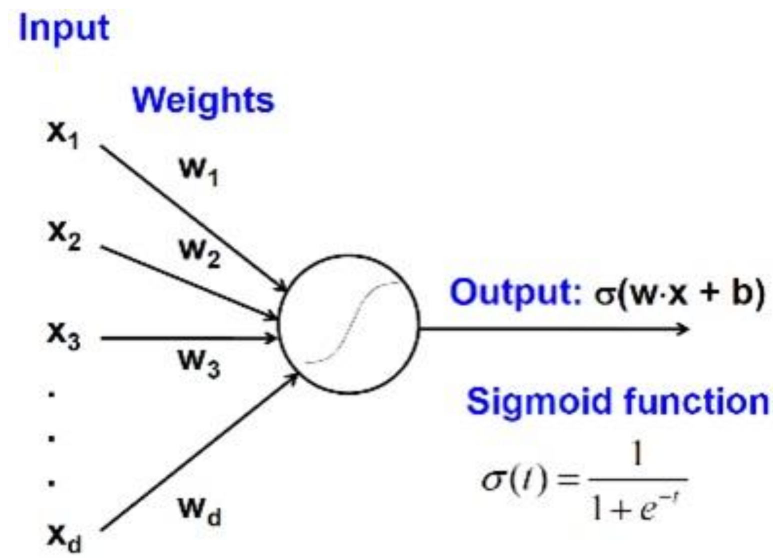


Image credit: Stanford CS231n

Biological neuron and Perceptrons

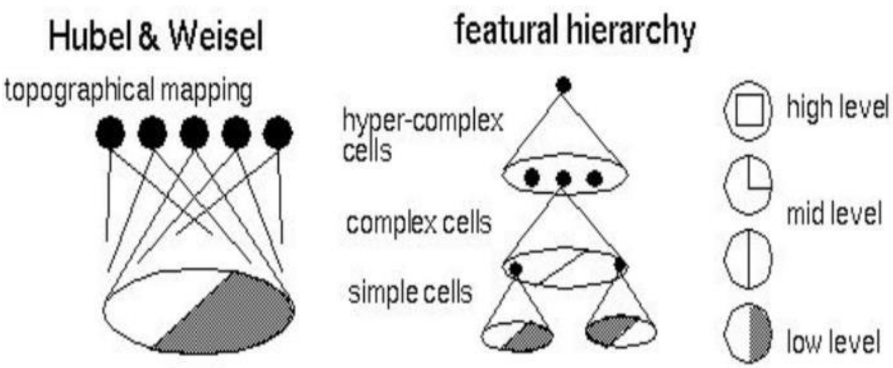


A biological neuron

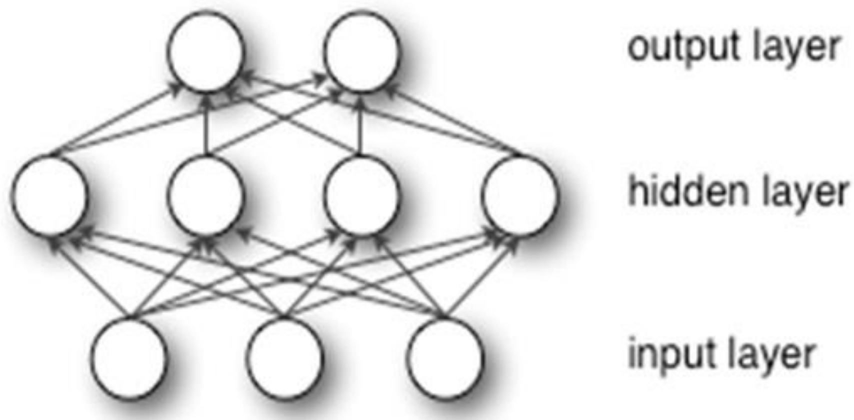


An artificial neuron (Perceptron)
- a linear classifier

Hubel/Wiesel Architecture and Multi-layer Neural Network



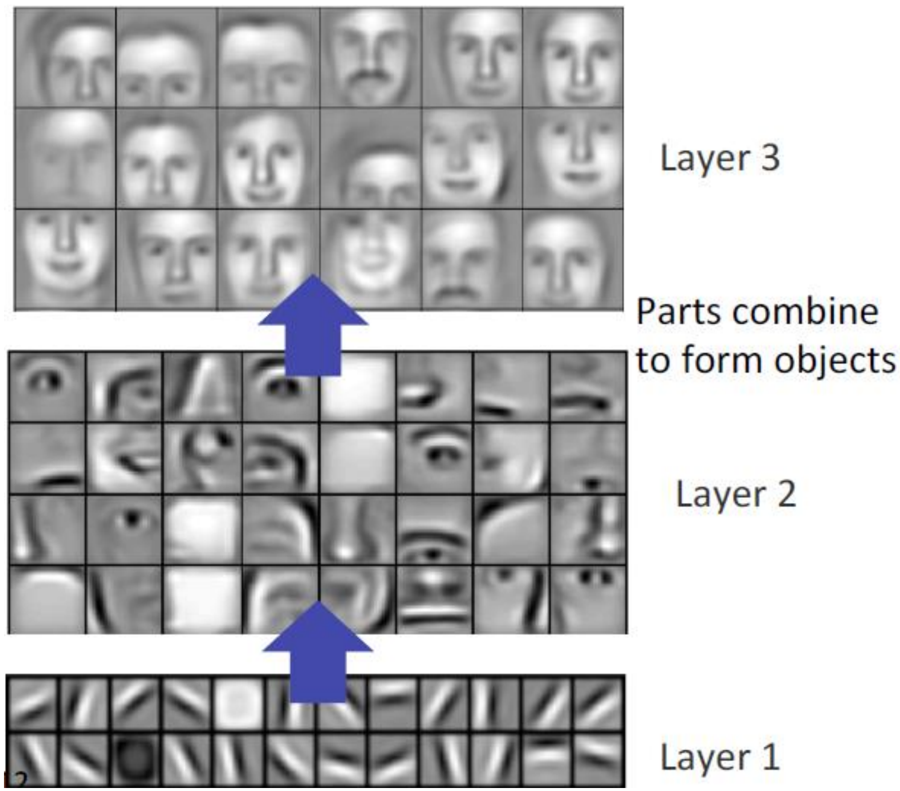
Hubel and Weisel's architecture

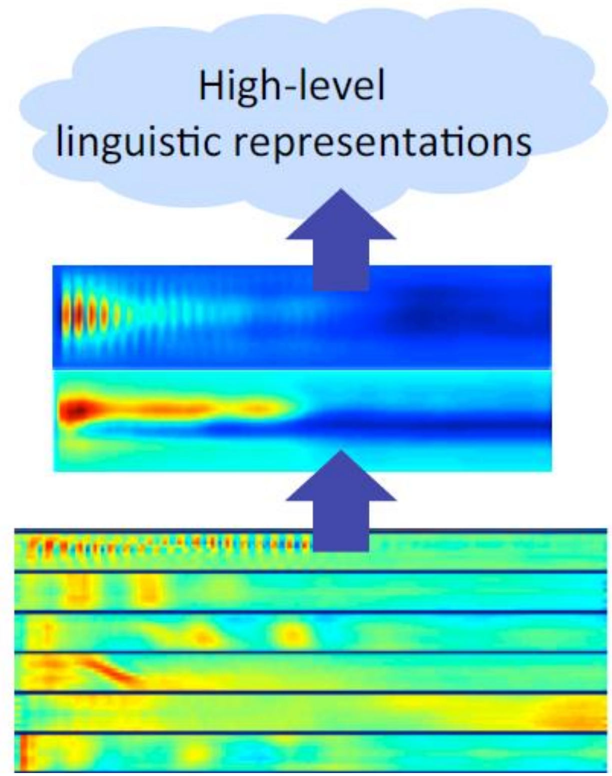


Multi-**L**ayer **P**erception or Neural Network
- A *non-linear* classifier

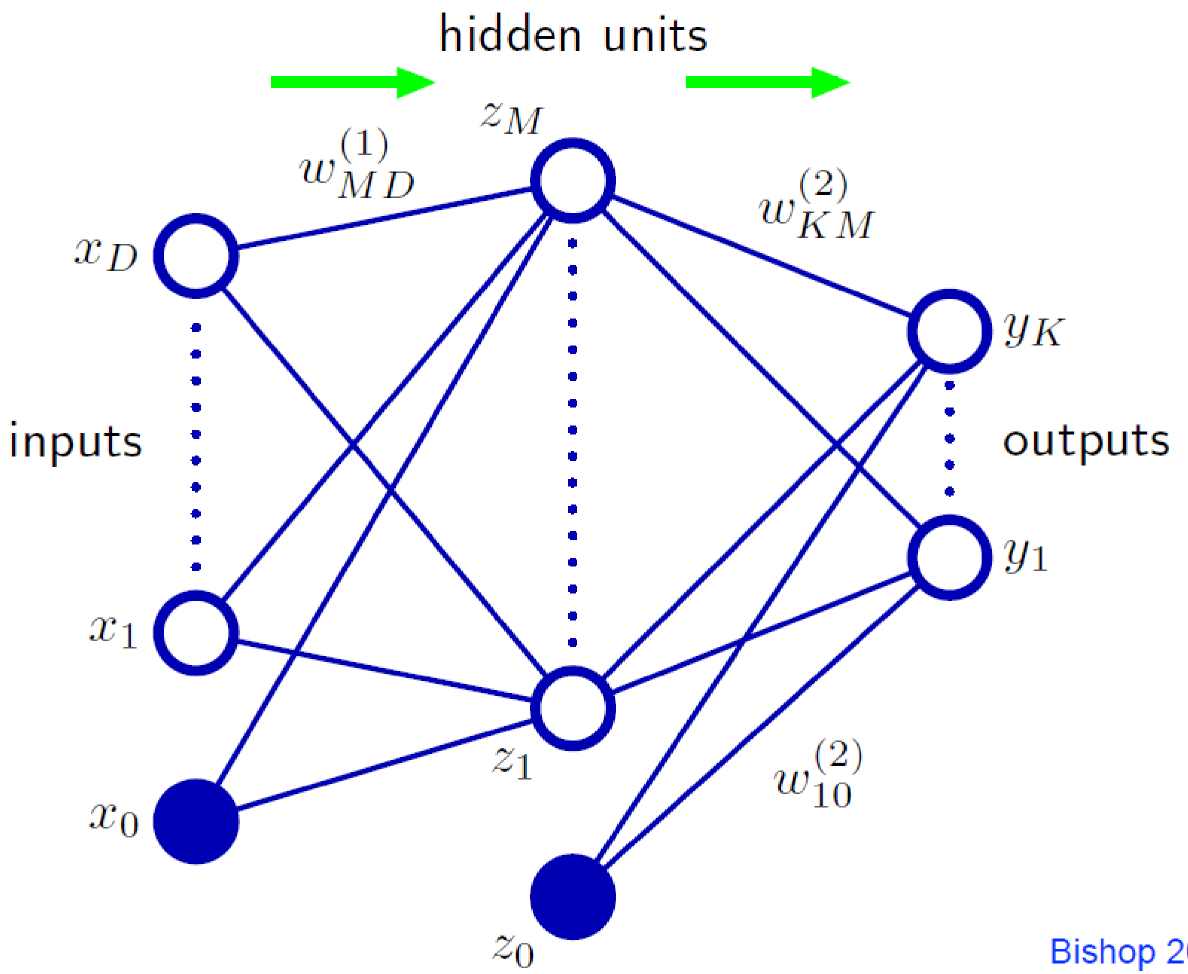
Hierarchical Representation Learning

- Successive model layers learn deeper intermediate representations.



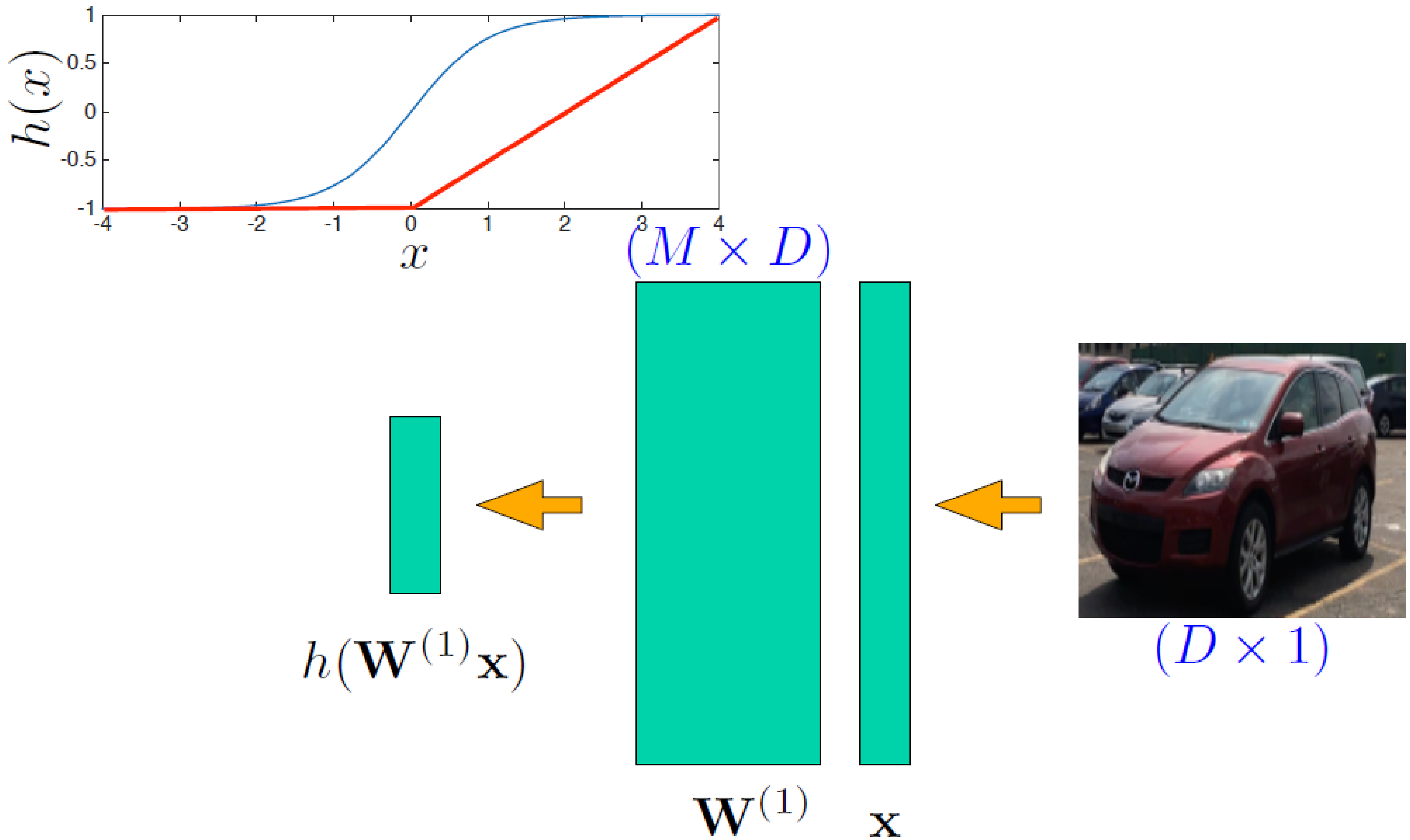


Multi-Layer Perceptron: A Nonlinear Classifier (cont'd)

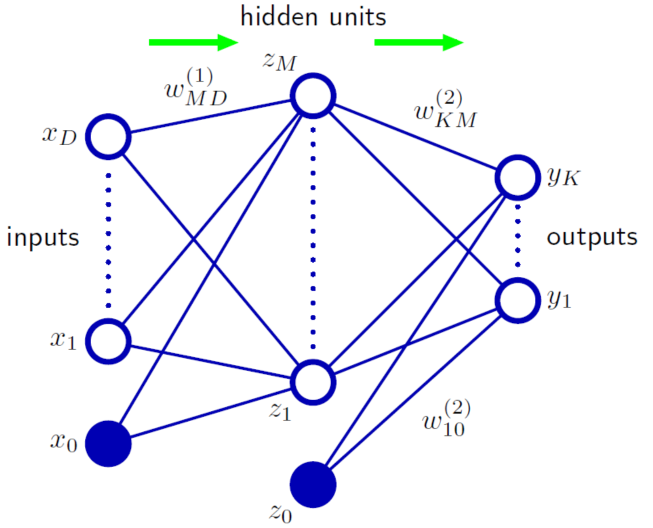


Bishop 2006

Multi-Layer Perceptron: A Nonlinear Classifier



Layer 1 in MLP



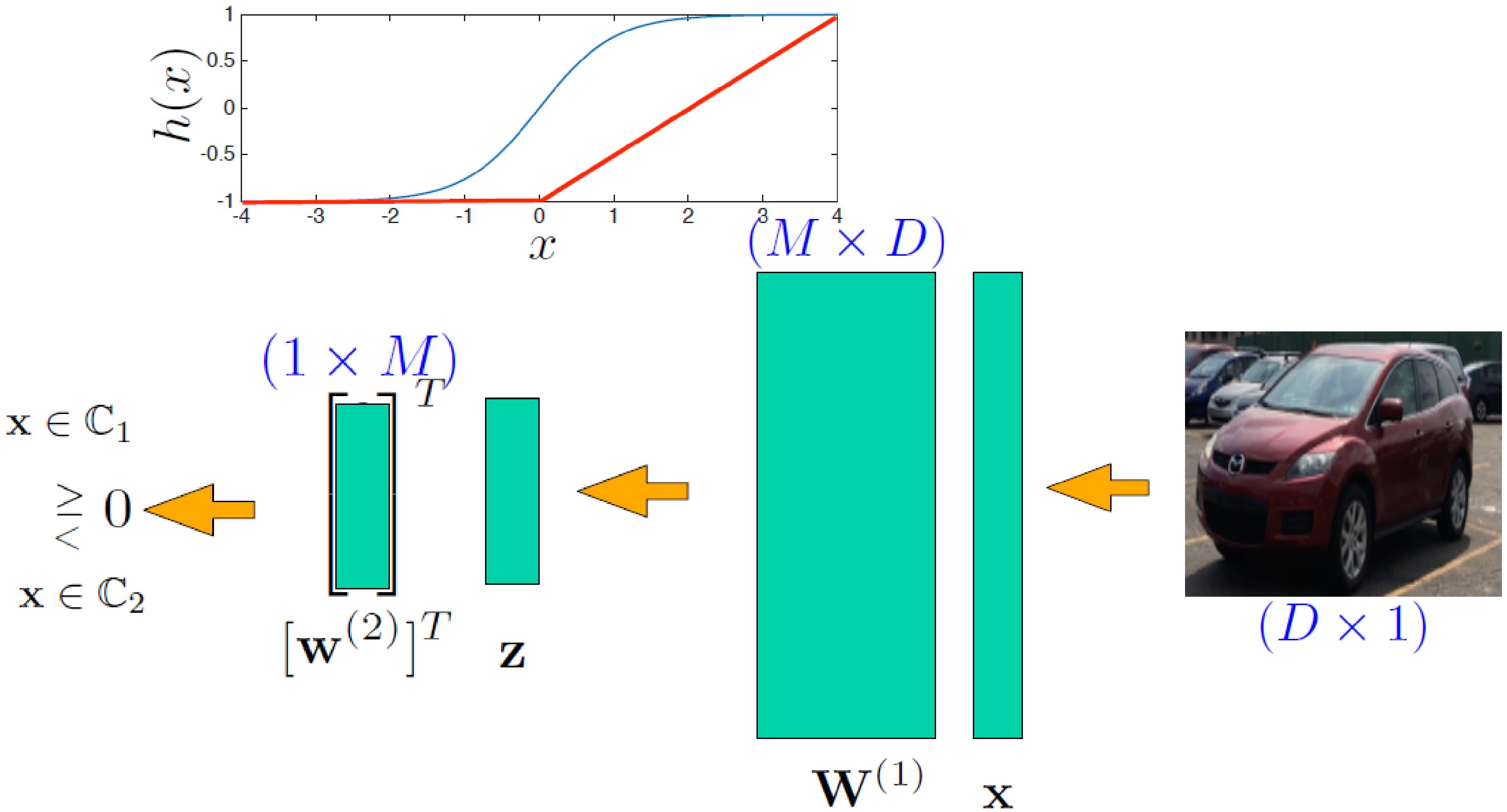
$$\mathbf{z} = \begin{bmatrix} z_1 \\ \vdots \\ z_M \end{bmatrix} \leftarrow \begin{bmatrix} h[\mathbf{x}^T \mathbf{w}_1^{(1)}] \\ \vdots \\ h[\mathbf{x}^T \mathbf{w}_M^{(1)}] \end{bmatrix}$$

$h() = \text{non-linear function}$

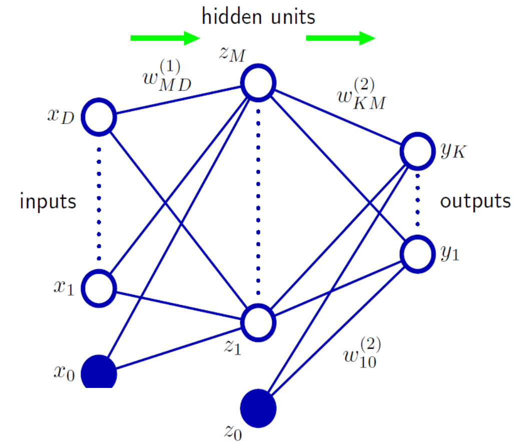
$[\mathbf{w}_1^{(1)}, \ldots, \mathbf{w}_M^{(1)}] = \text{1st layer's } D \times M \text{ weights}$

$\mathbf{x} = D \times 1 \text{ raw input}$

Multi-Layer Perceptron: A Nonlinear Classifier (cont'd)



Layer 2 in MLP



$$\mathbf{z} \in \mathbb{C}_1$$

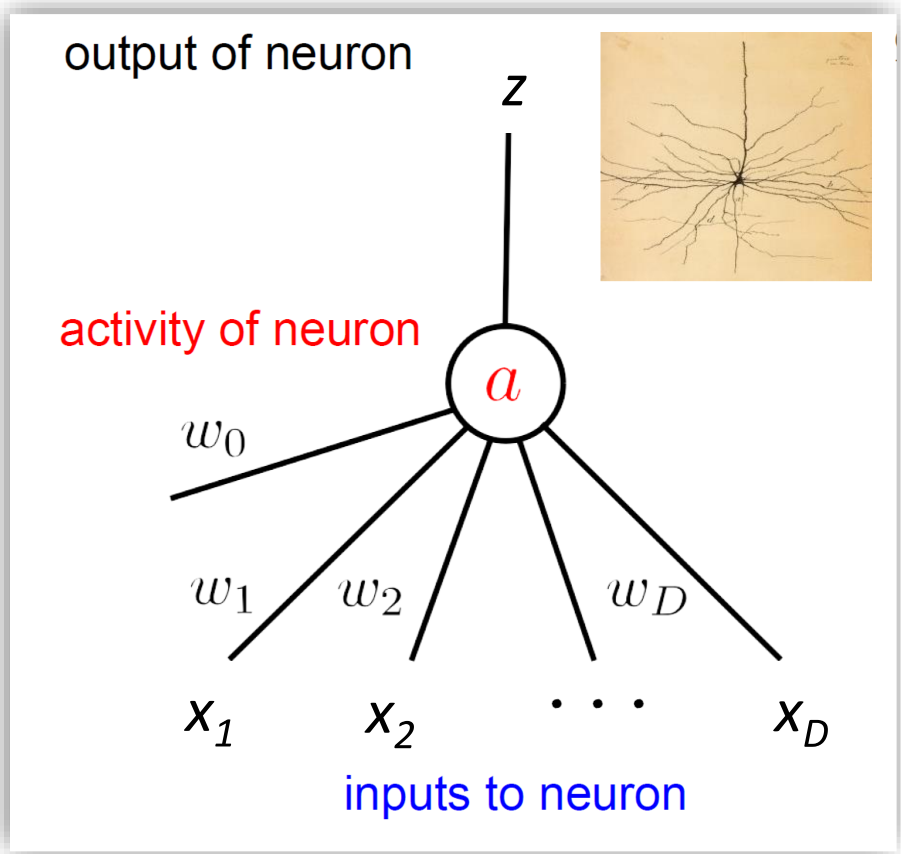
$$\mathbf{z}^T \mathbf{w}^{(2)} \quad \begin{matrix} \geq \\ < \end{matrix} \quad 0$$

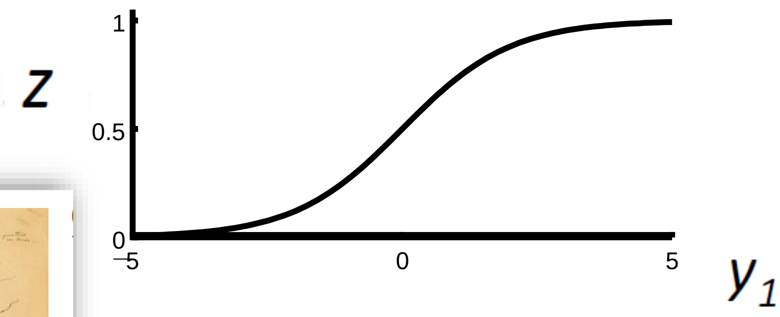
$$\mathbf{z} \in \mathbb{C}_2$$

$\mathbf{z} = M \times 1$ output of layer 1
$\mathbf{w}^{(2)} =$ 2nd layer's $M \times 1$ weight vector

Let's Take a Closer Look… Output $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$

- A single neuron

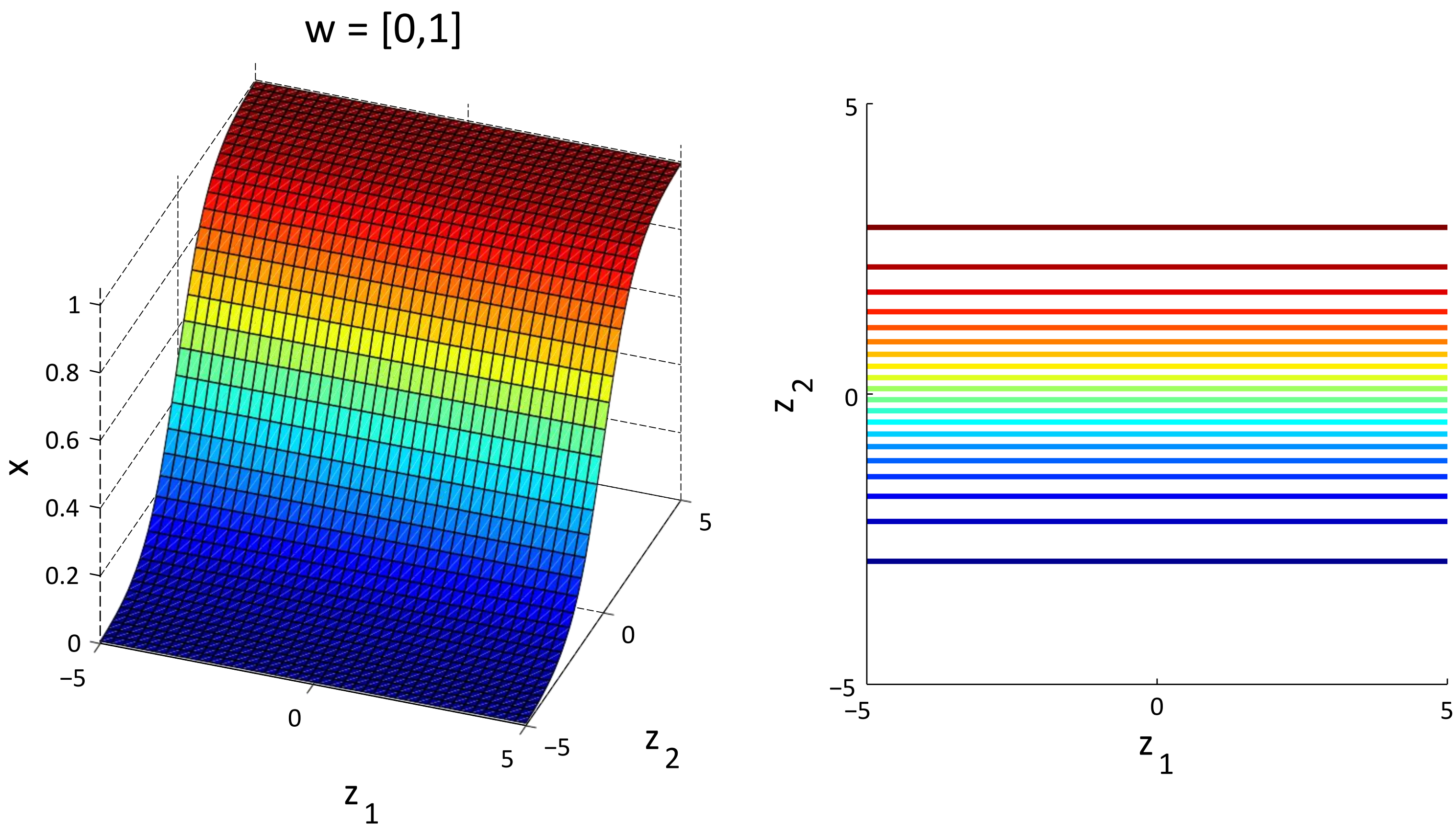




$$z(y_1) = \frac{1}{1 + \exp(-y_1)}, z \in (0,1)$$

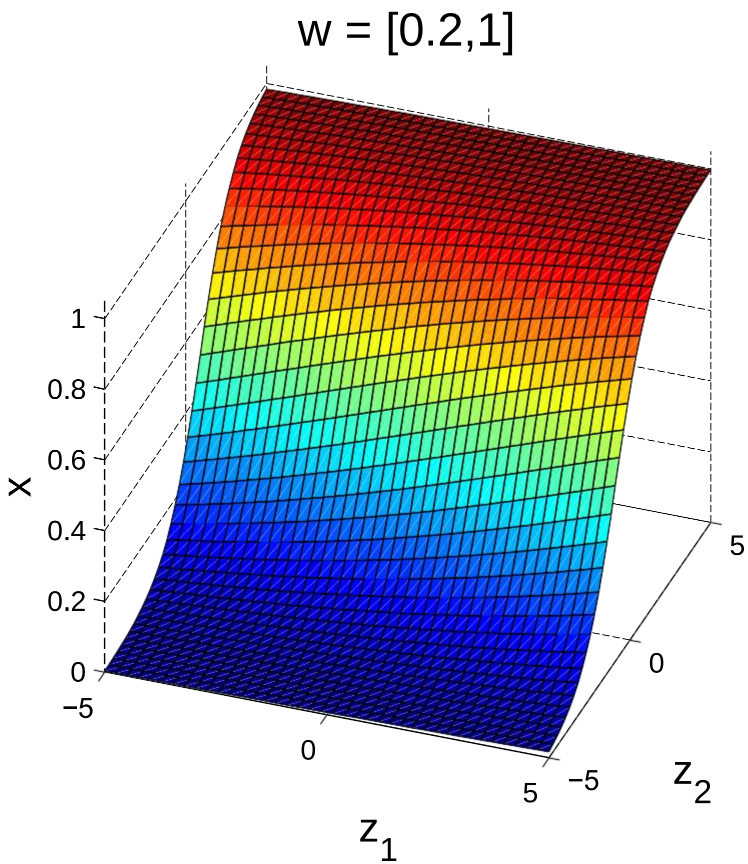
$$y_1 = w_0 + \sum_{d=1}^{D} w_d z_d$$
$$= \sum_{d=0}^{D} w_d z_d$$

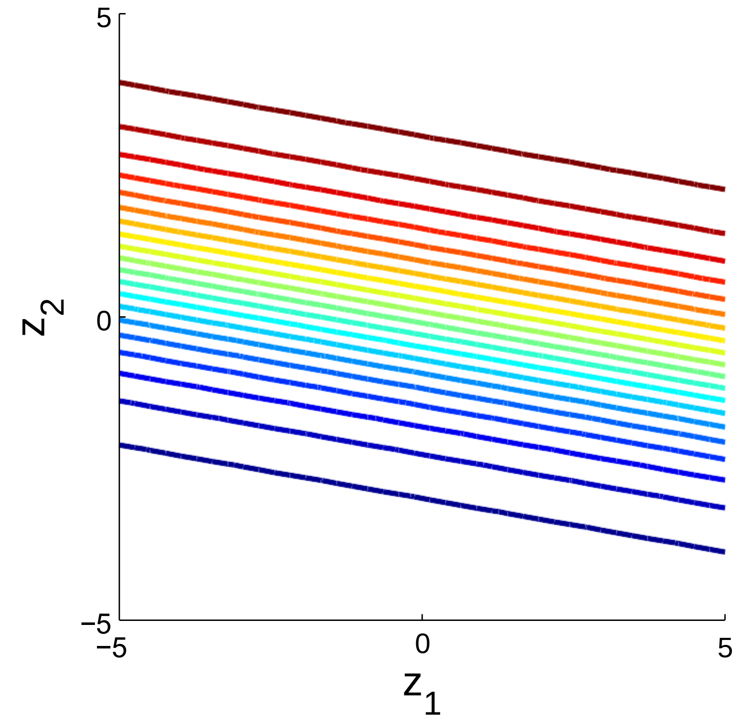
Input-Output Function of a Single Neuron



$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

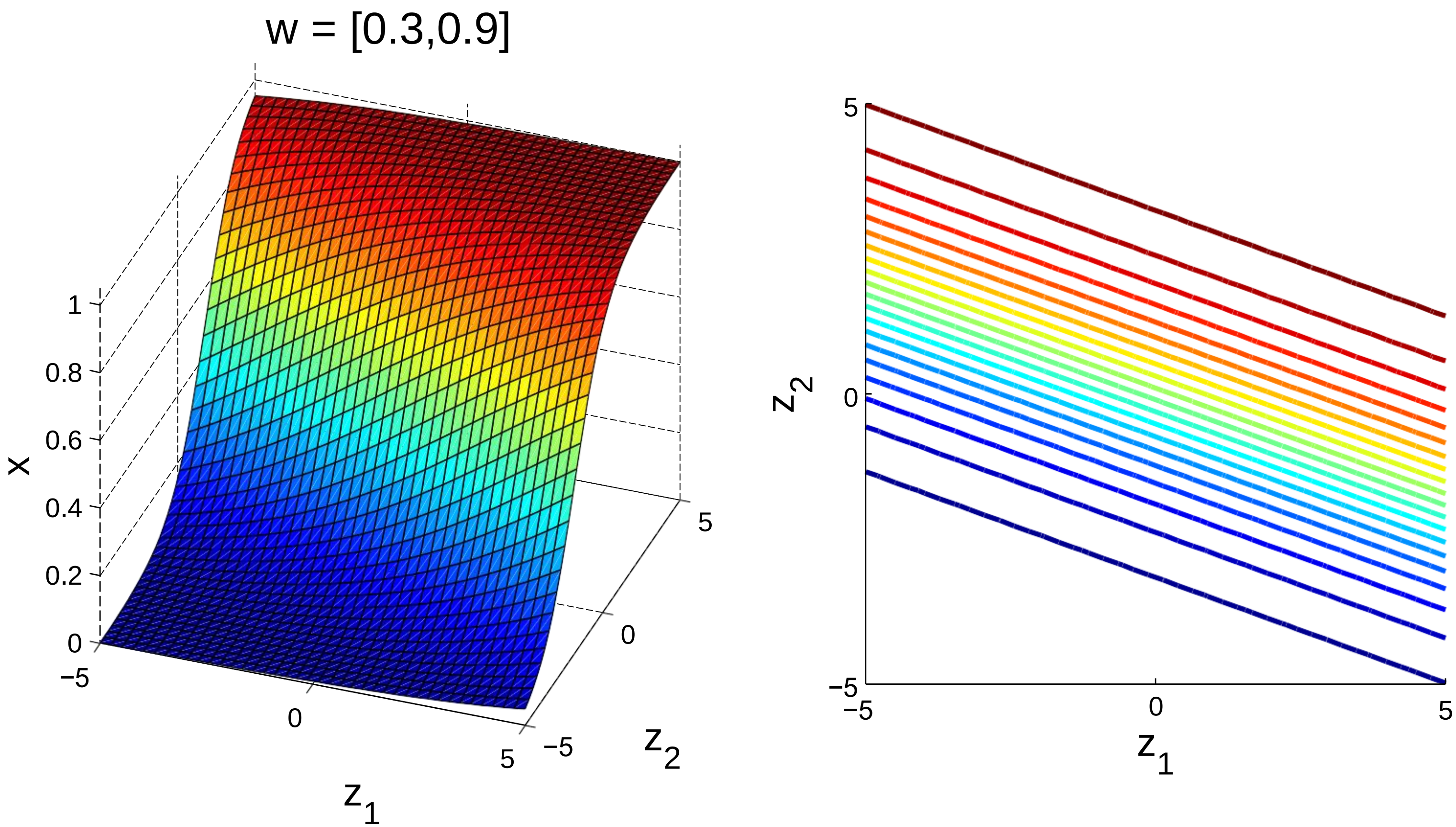
Input-Output Function of a Single Neuron





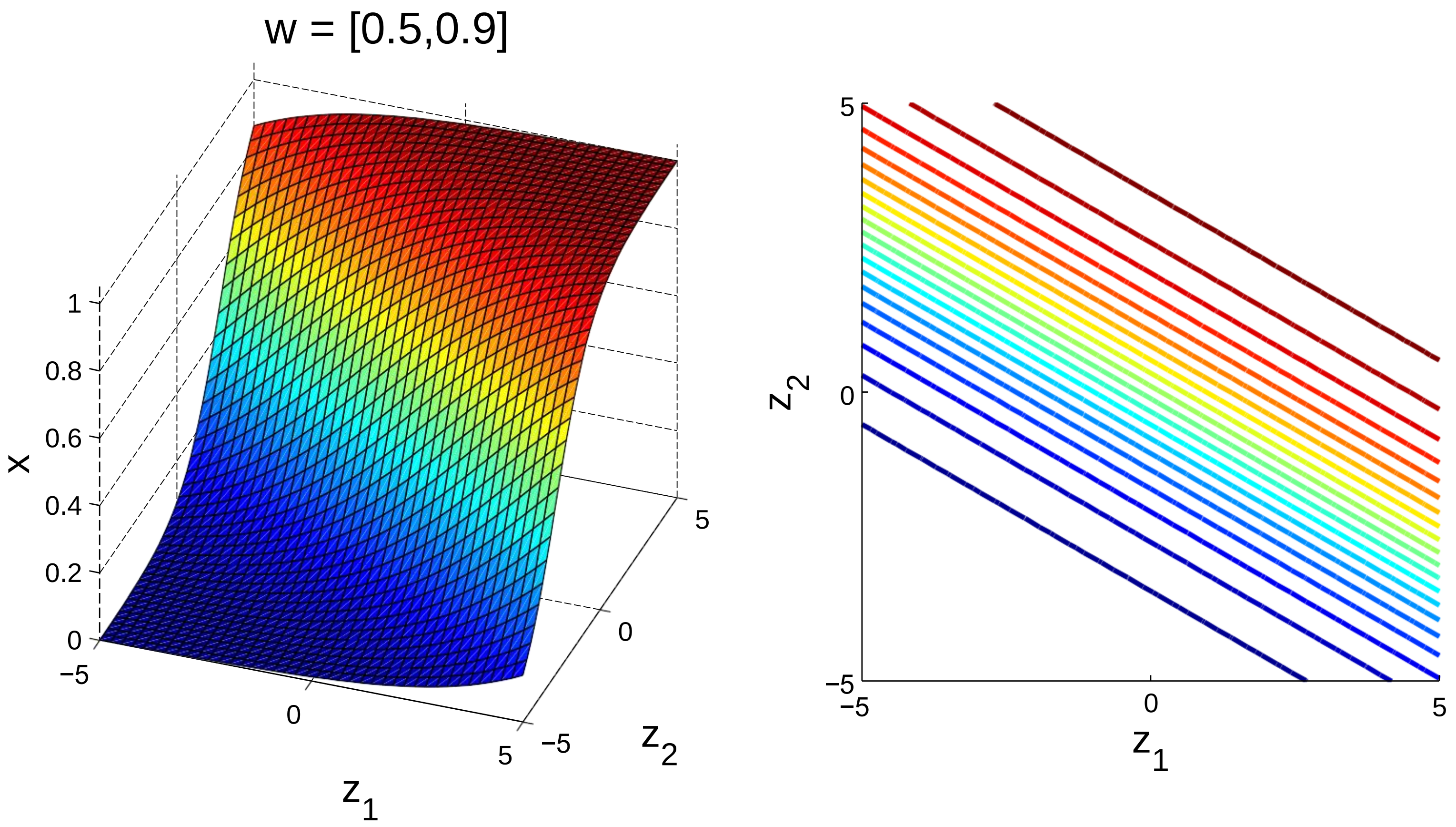
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

Input-Output Function of a Single Neuron



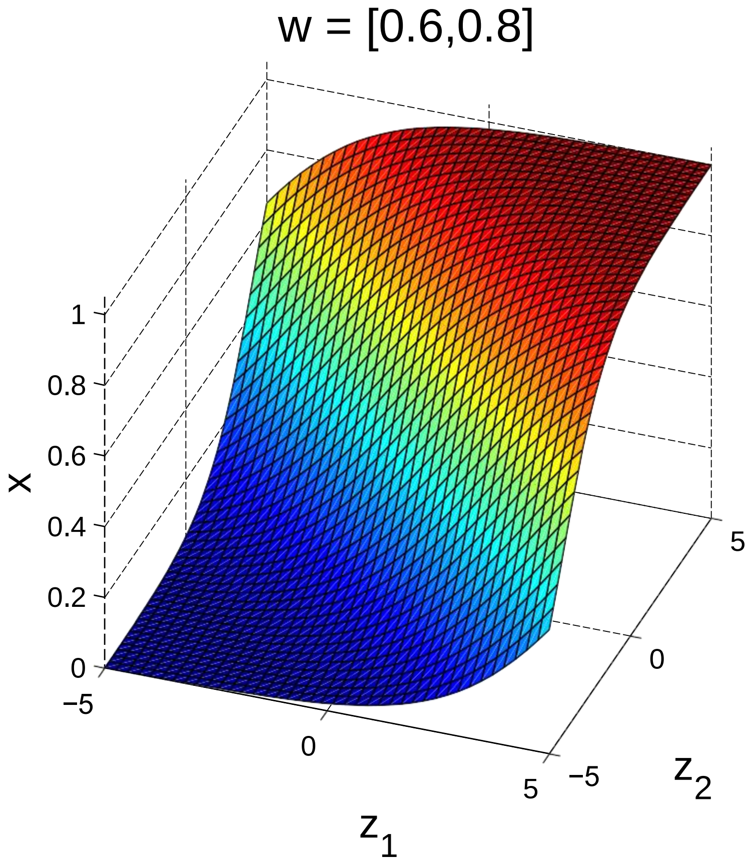
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

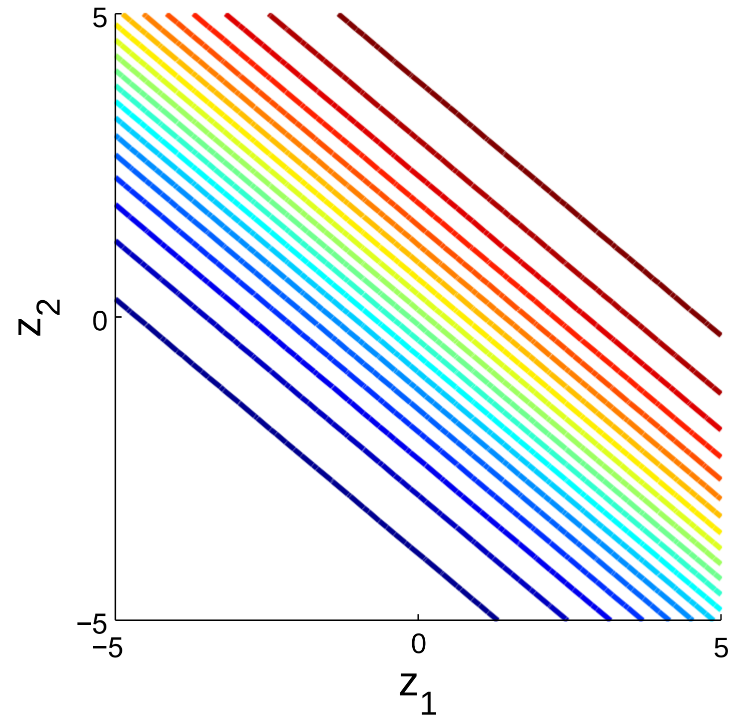
Input-Output Function of a Single Neuron



$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

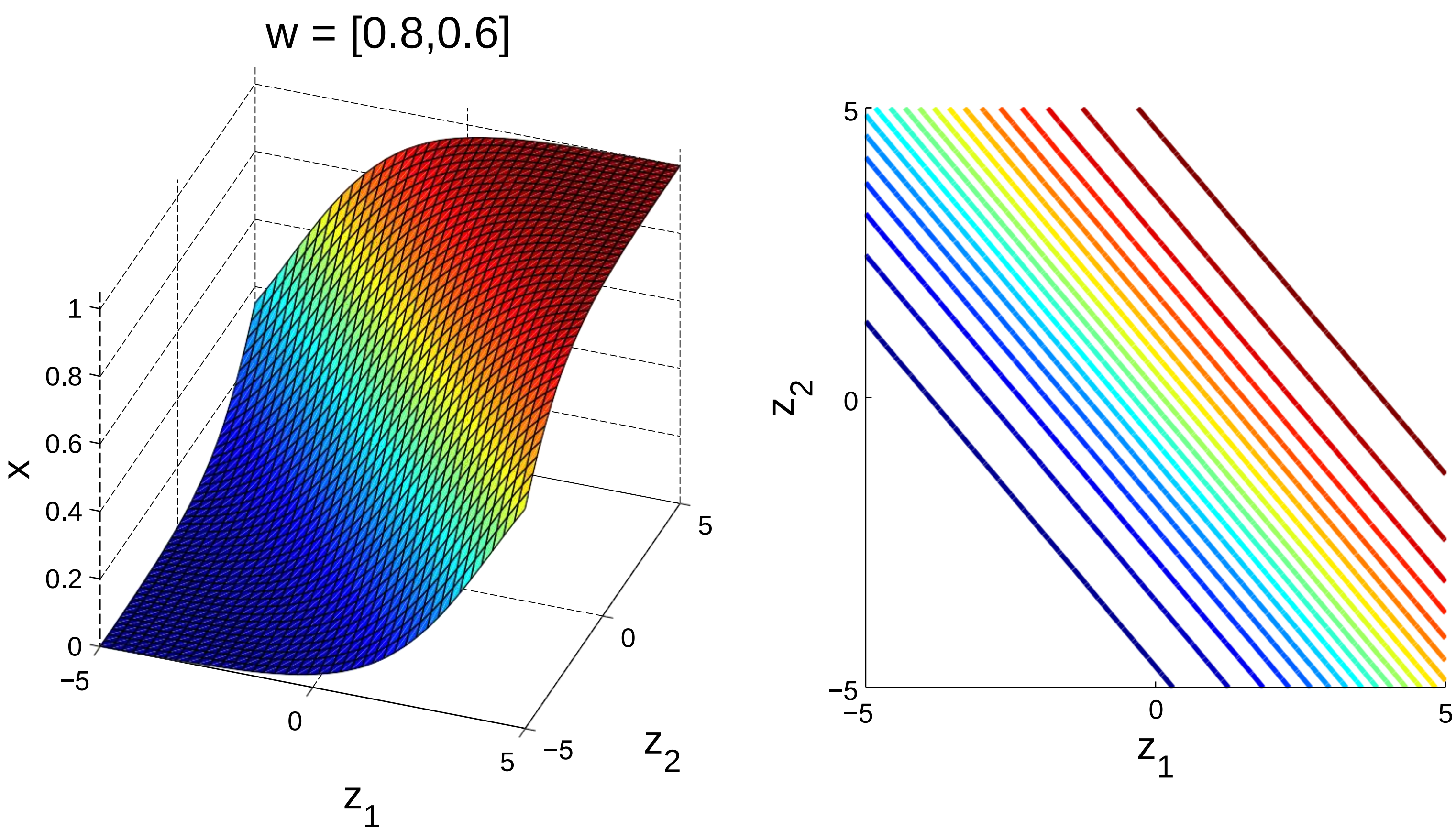
Input-Output Function of a Single Neuron





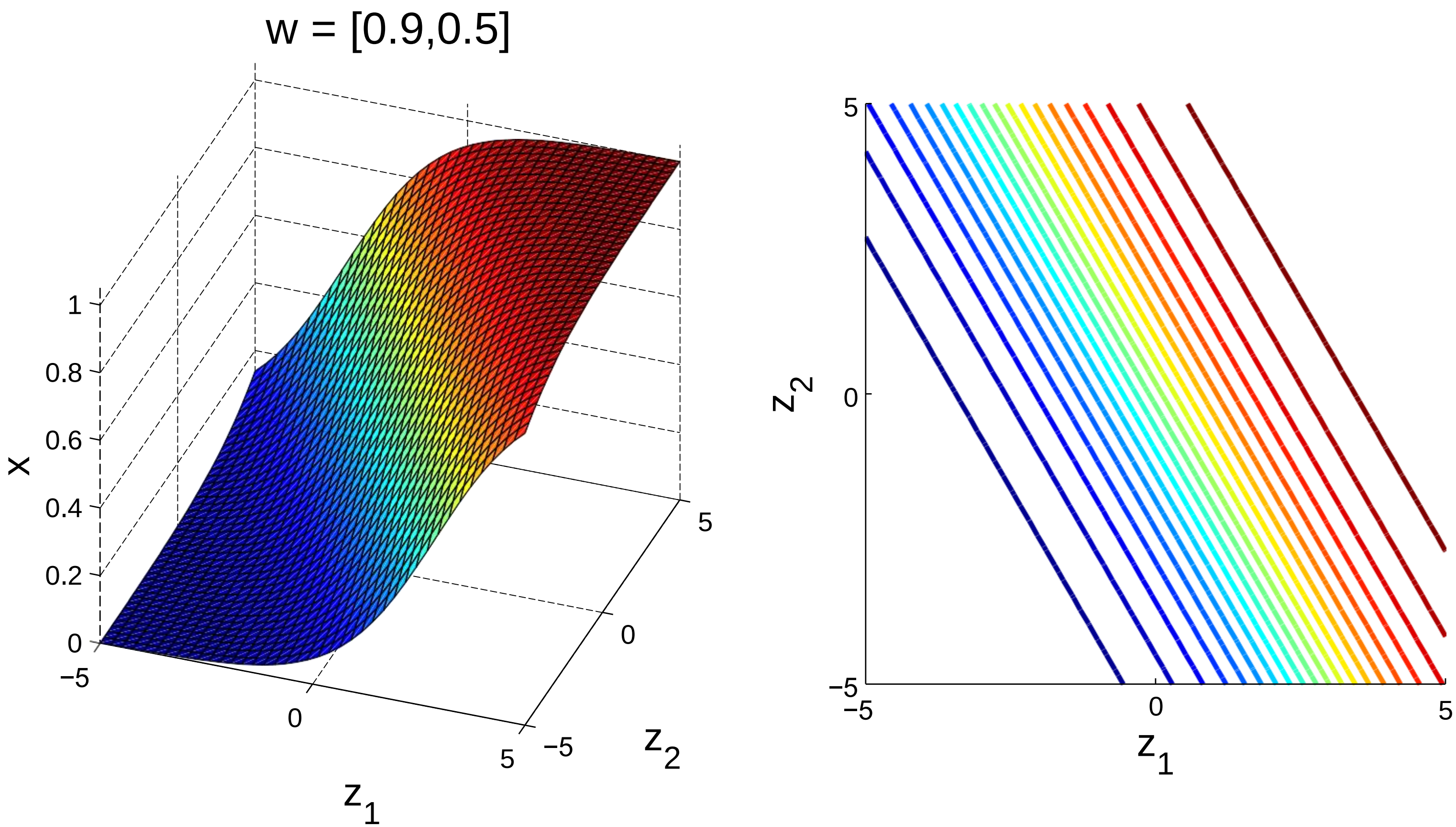
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

Input-Output Function of a Single Neuron



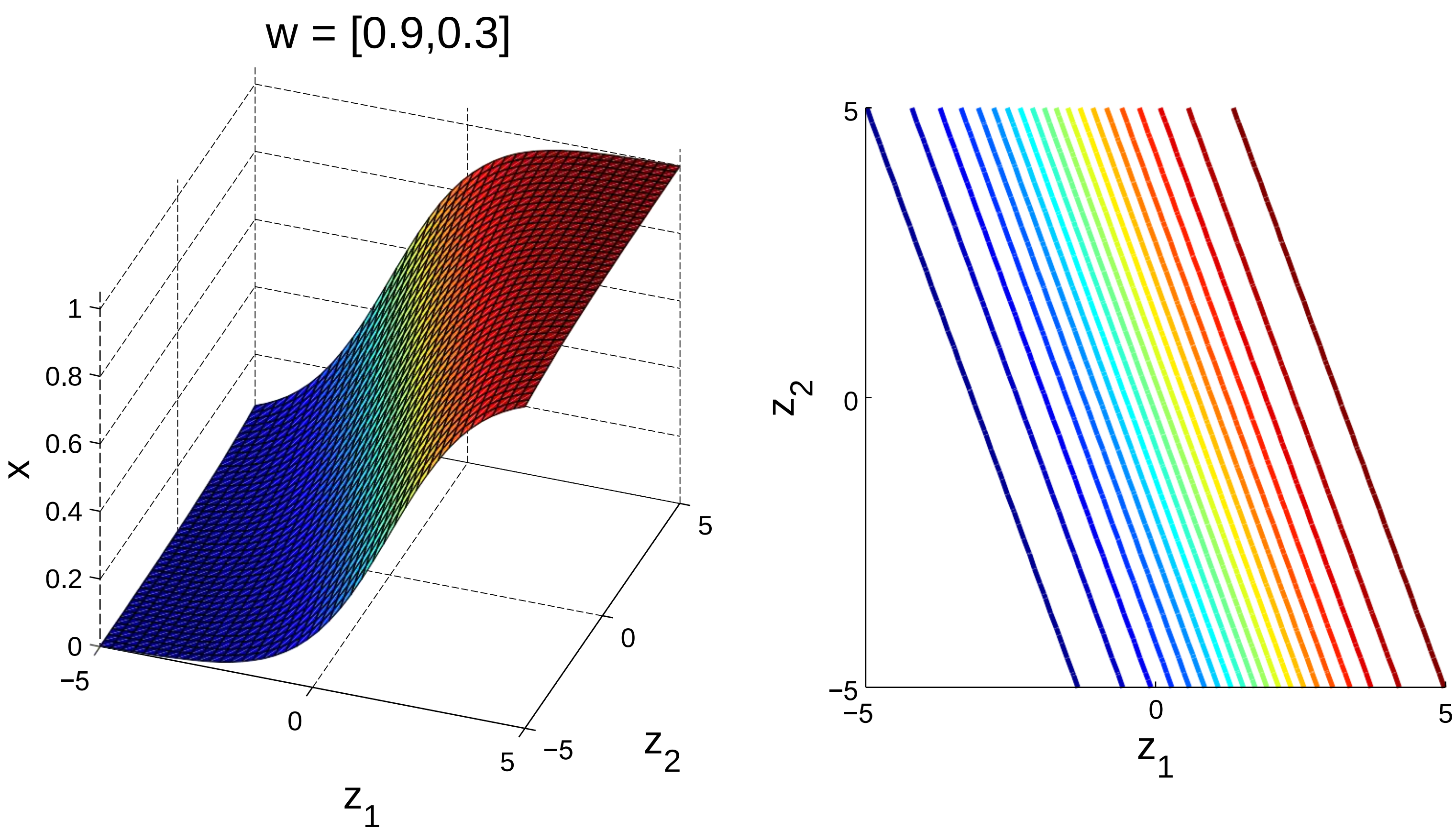
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

Input-Output Function of a Single Neuron



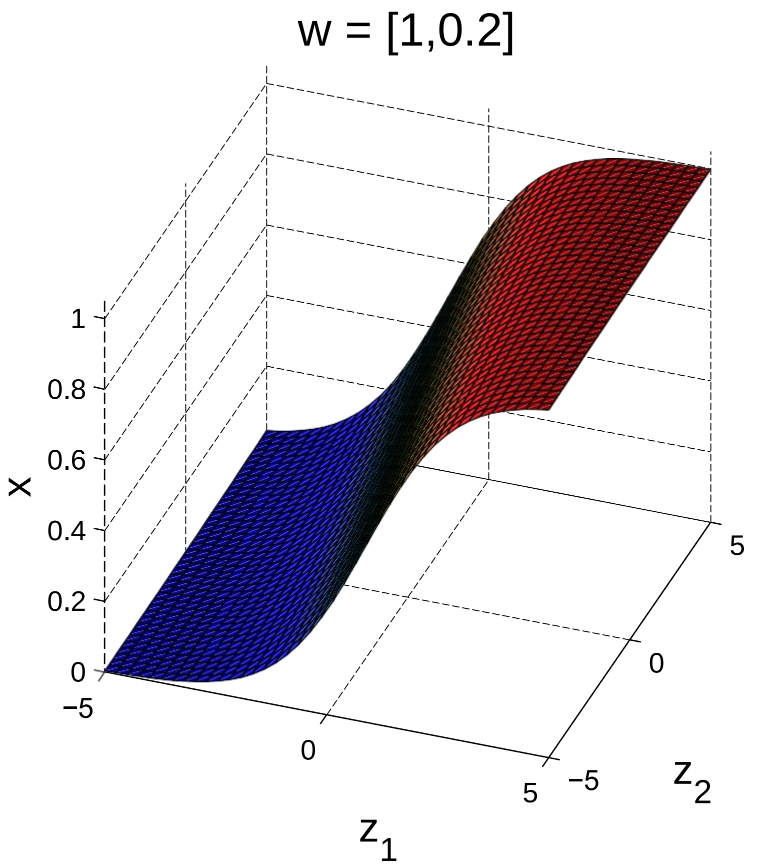
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

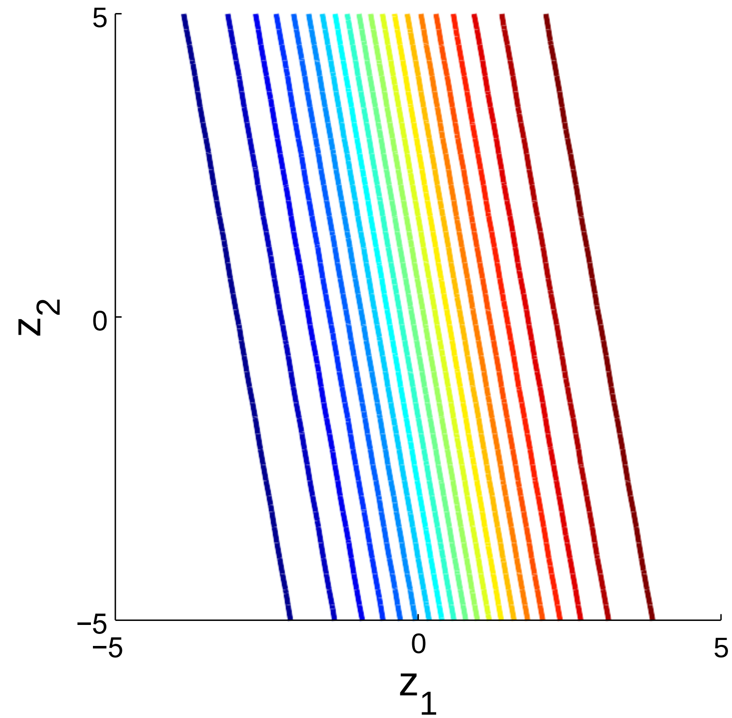
Input-Output Function of a Single Neuron



$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

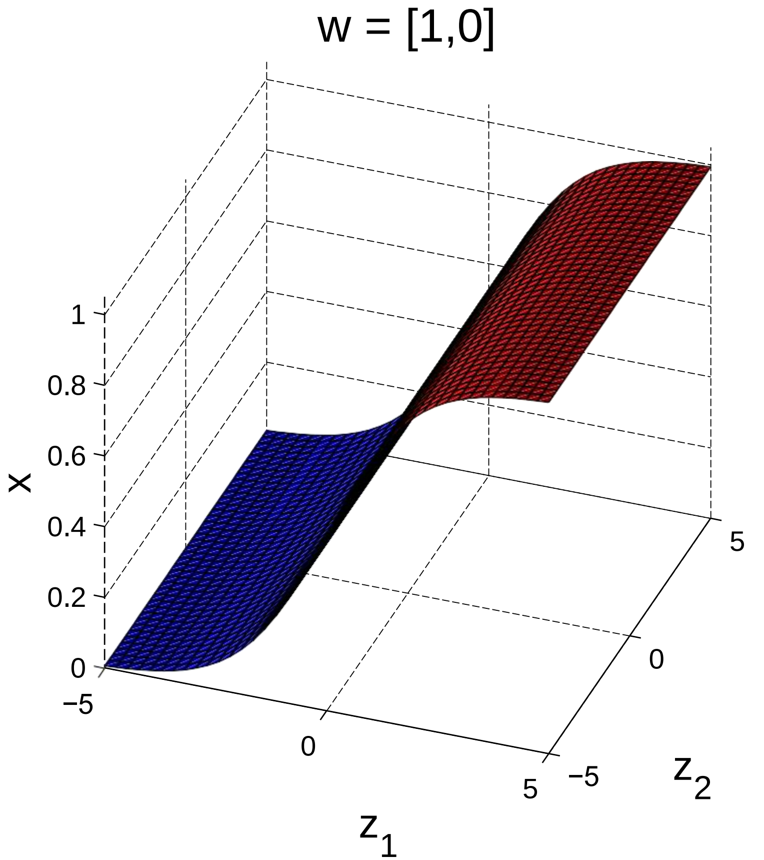
Input-Output Function of a Single Neuron

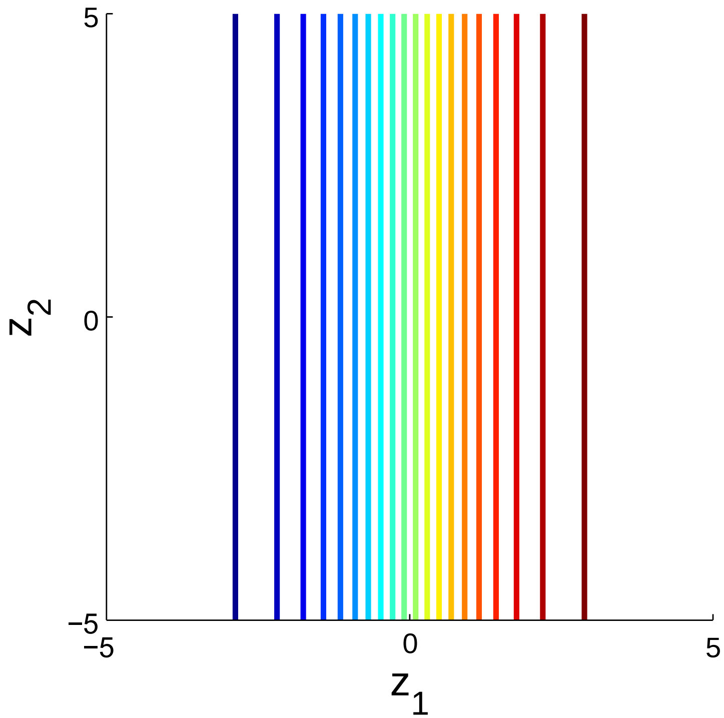




$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

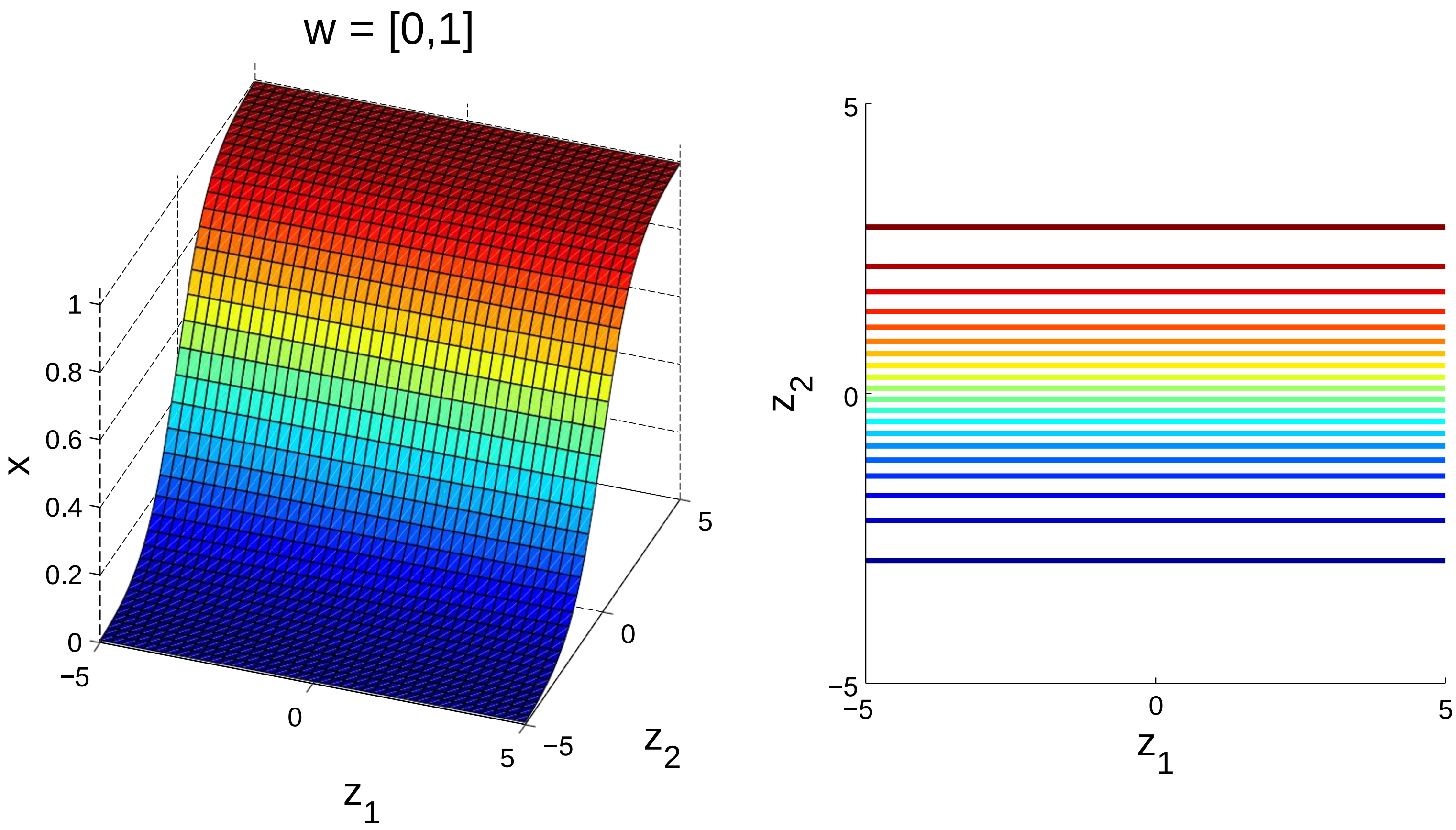
Input-Output Function of a Single Neuron





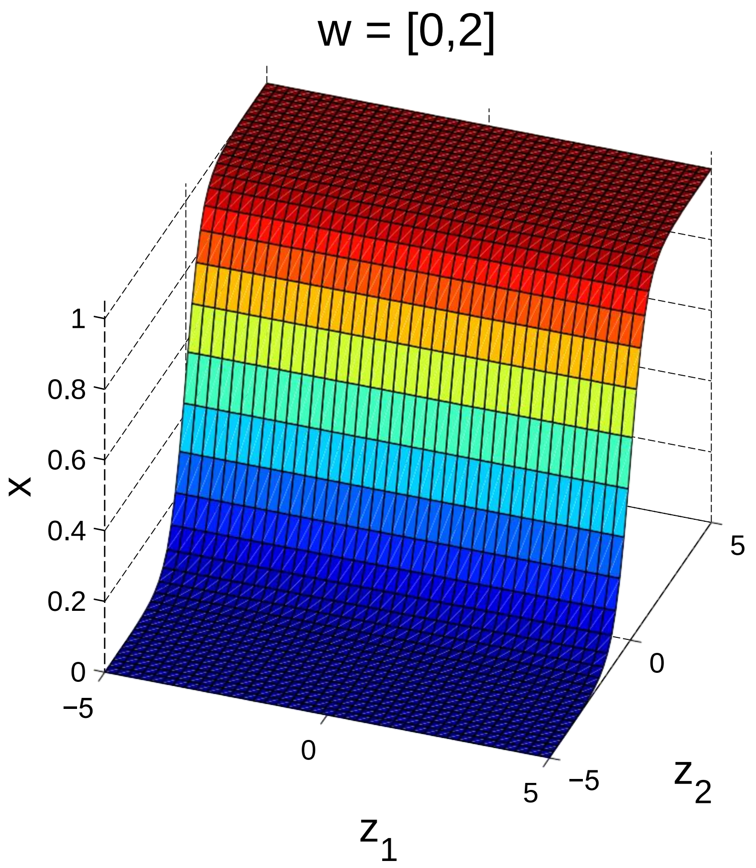
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

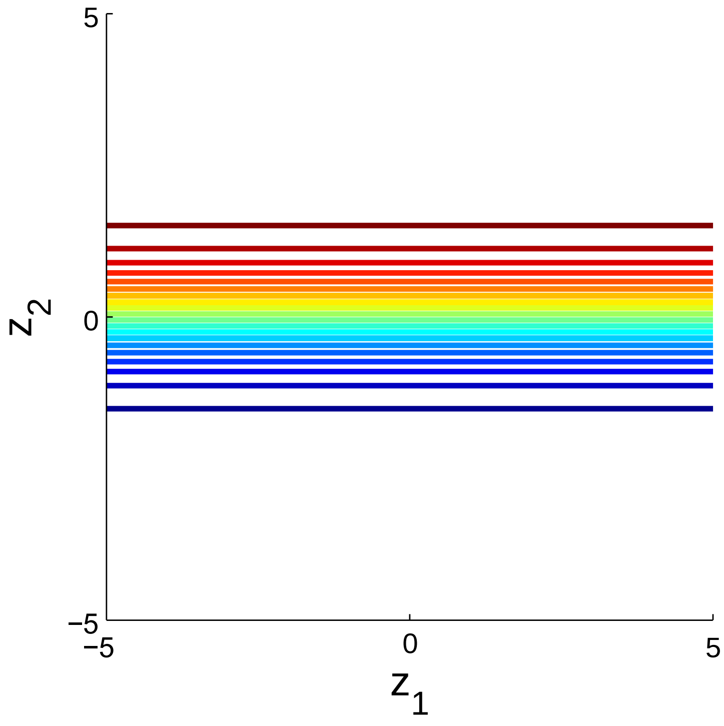
Input-Output Function of a Single Neuron (cont'd)



$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

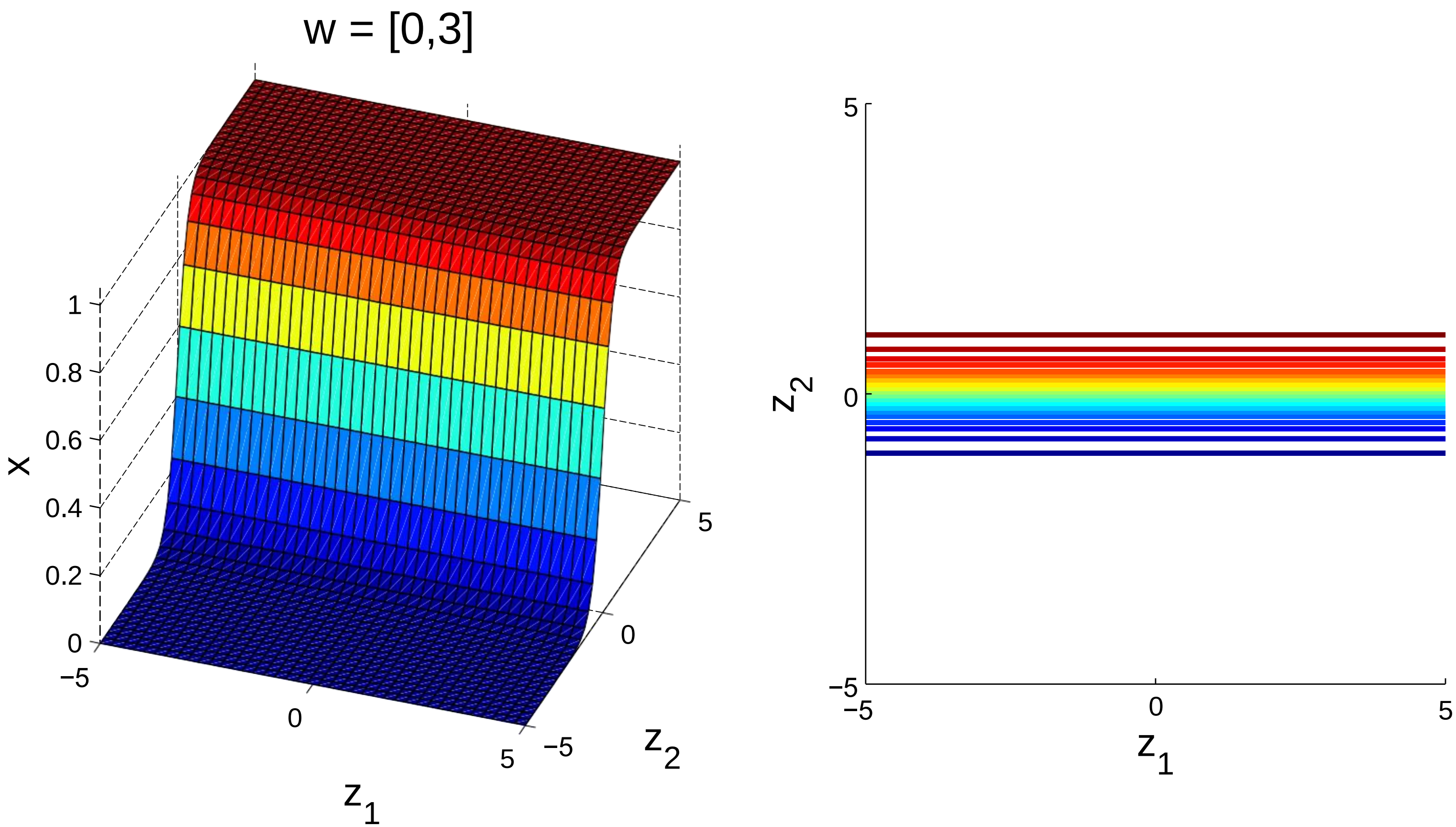
Input-Output Function of a Single Neuron (cont'd)





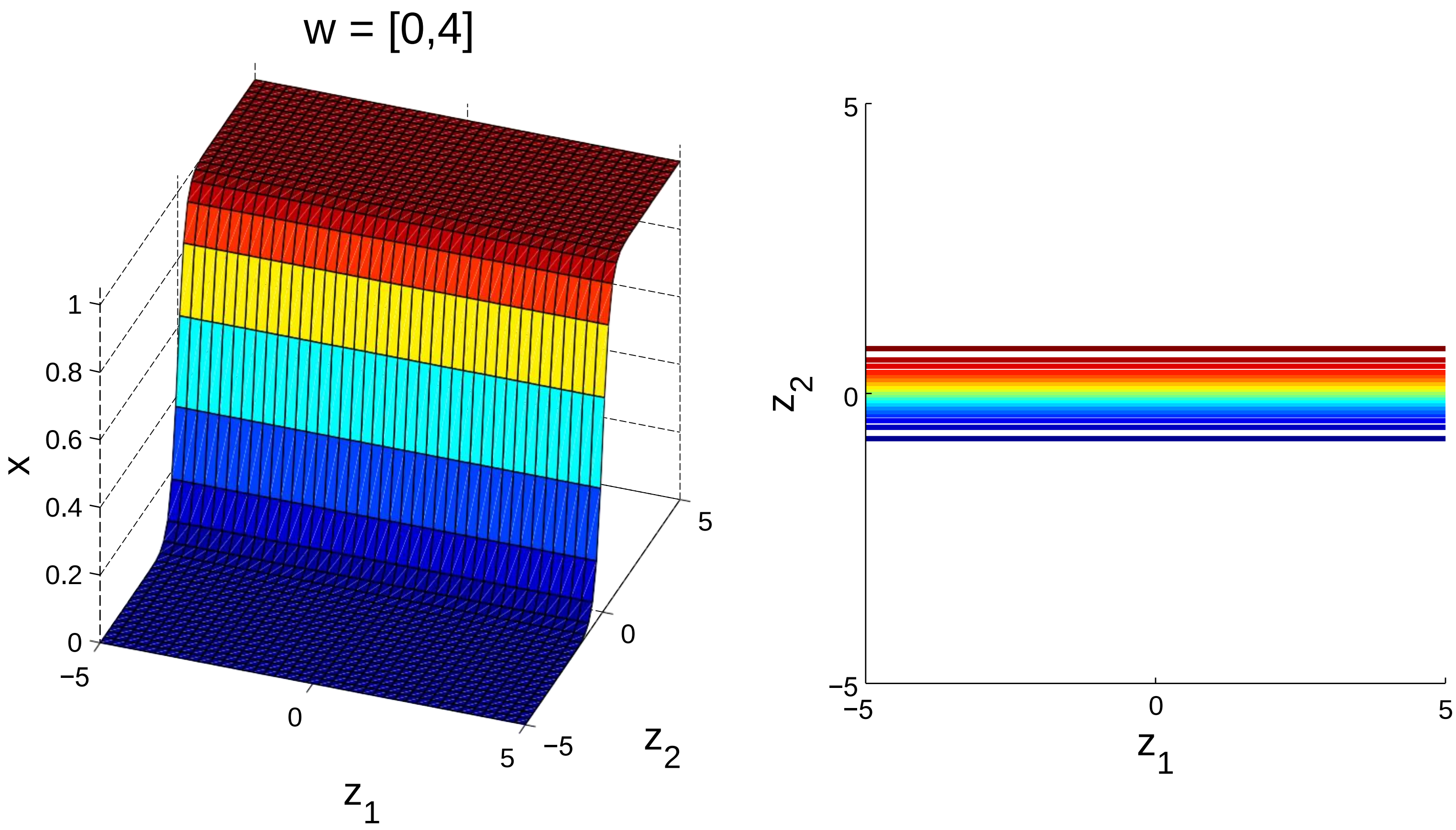
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

Input-Output Function of a Single Neuron (cont'd)



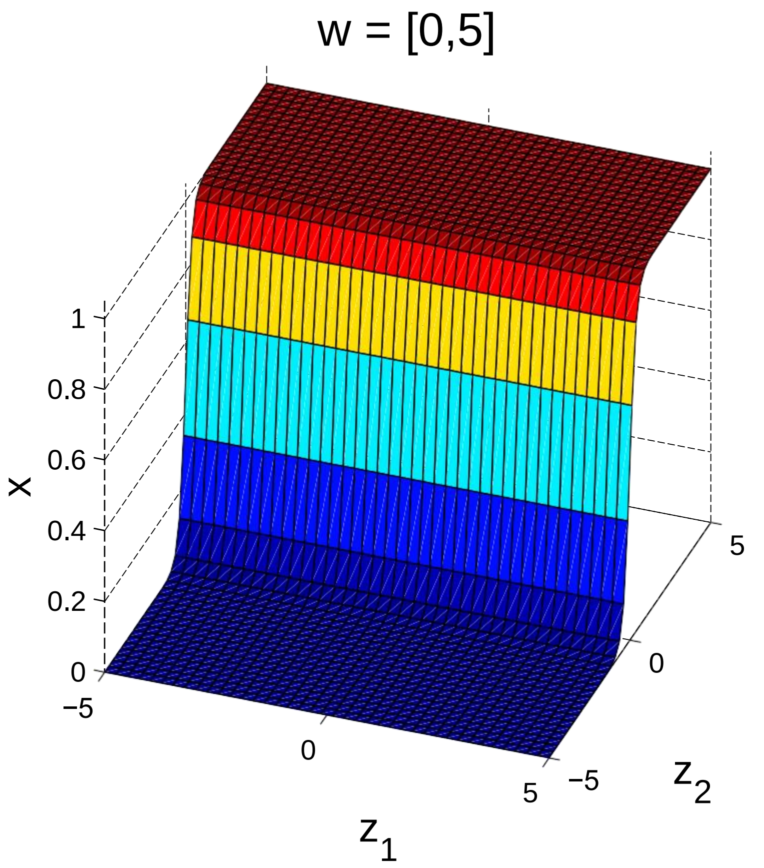
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

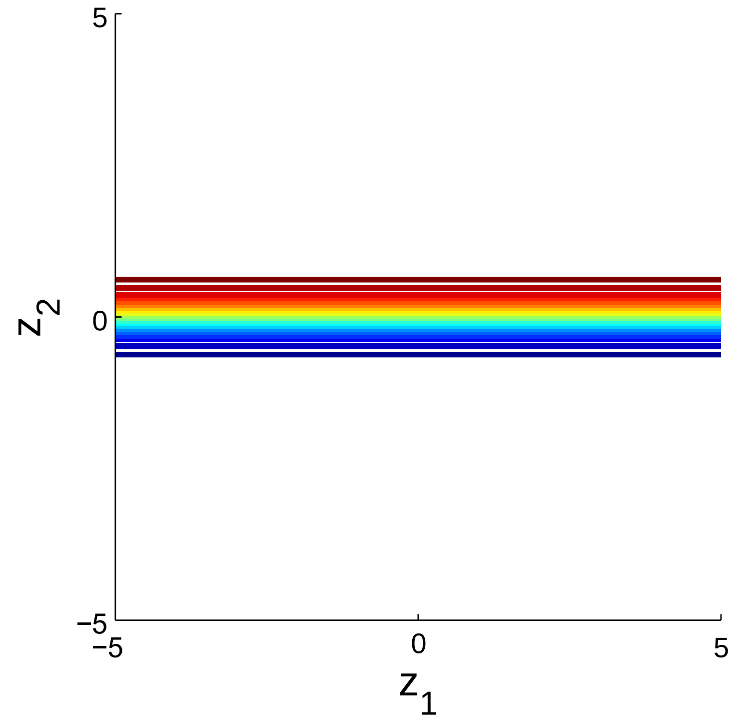
Input-Output Function of a Single Neuron (cont'd)



$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

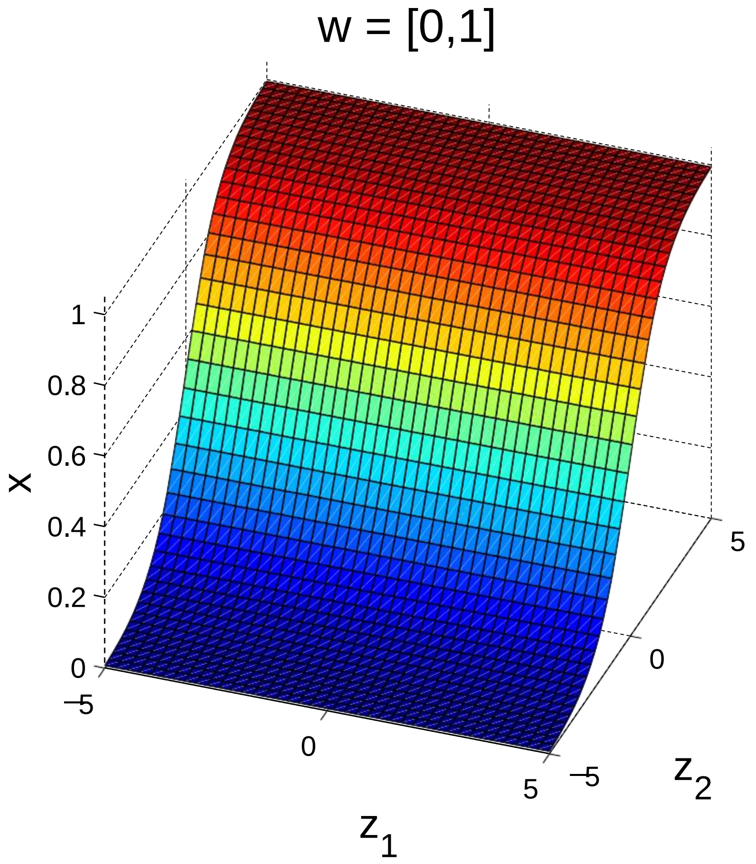
Input-Output Function of a Single Neuron (cont'd)

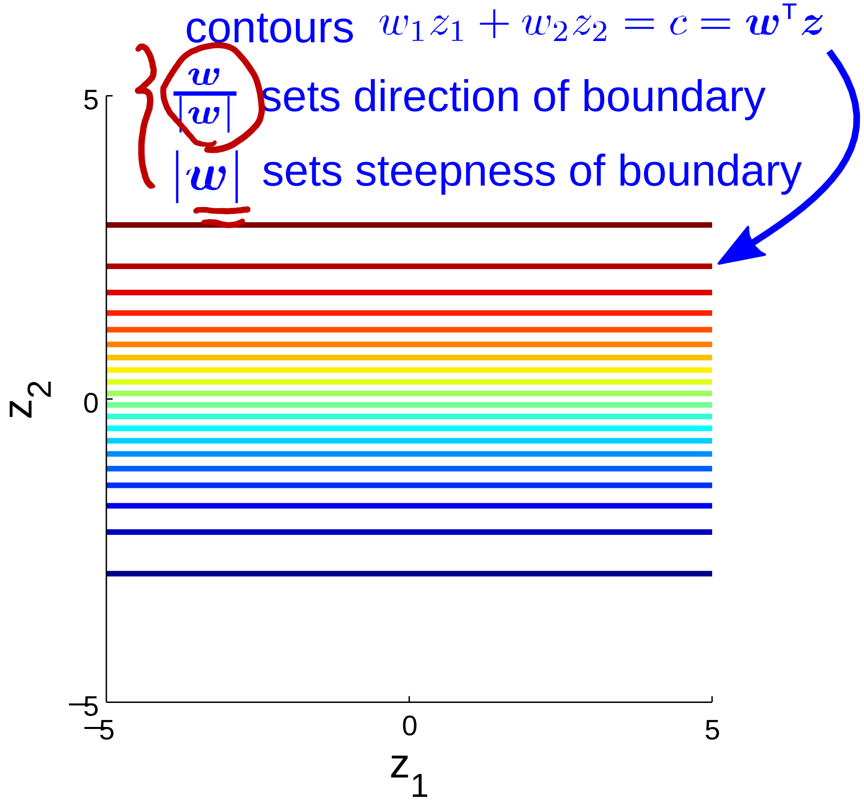




$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

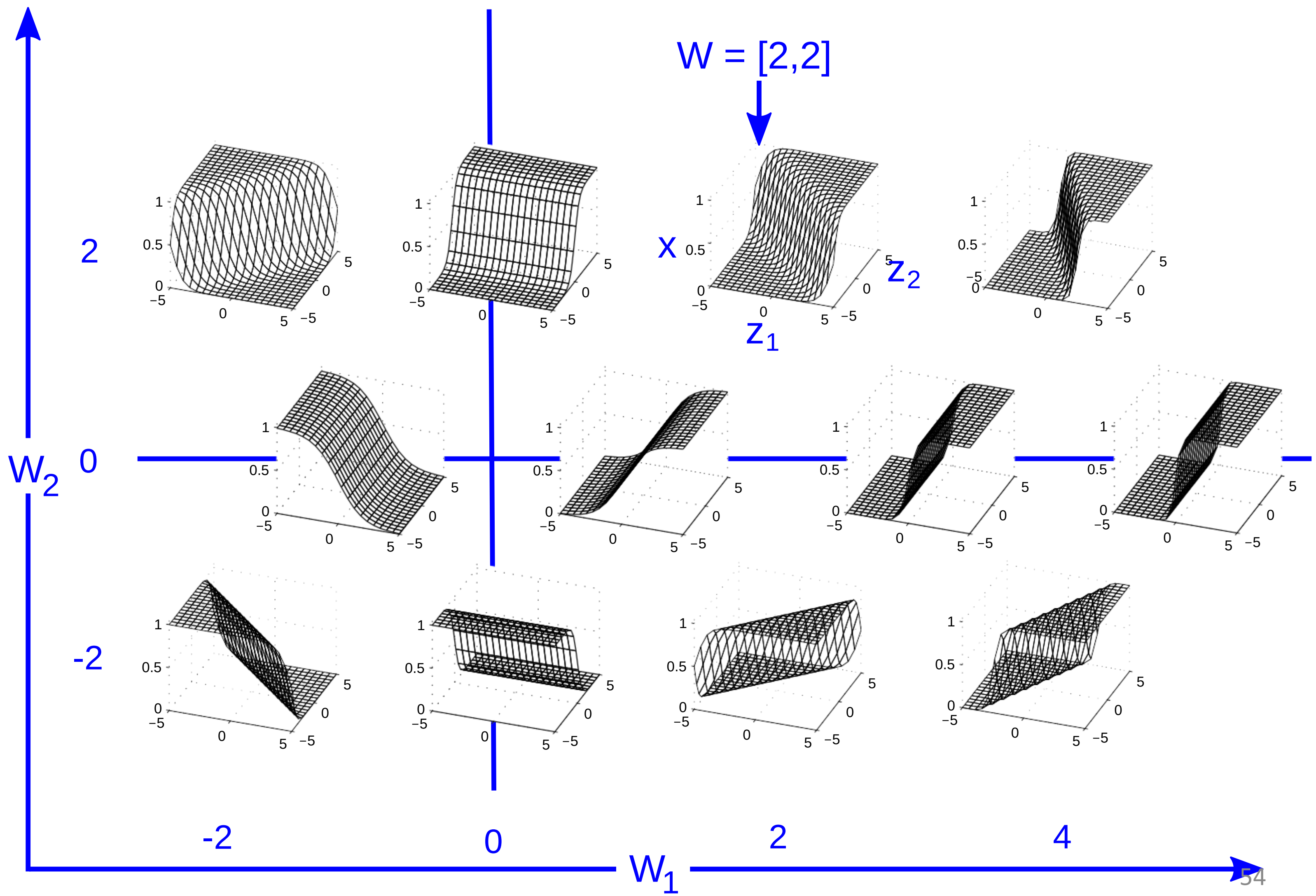
Input-Output Function of a Single Neuron (cont'd)



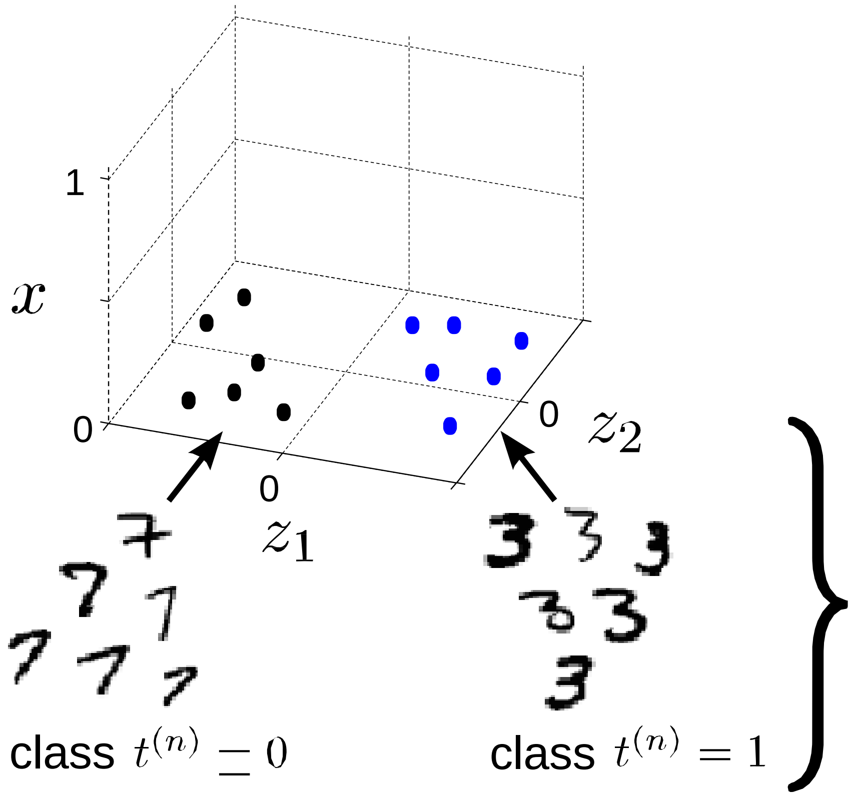


$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

Weight Space of a Single Neuron



Training a Single Neuron

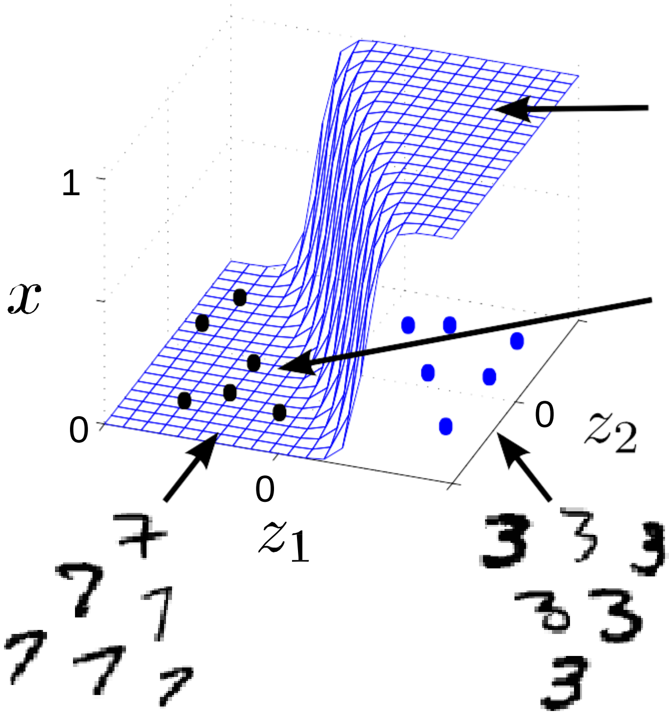


training data

$\{\boldsymbol{z}^{(n)}\}_{n=1}^{N}$ $\{t^{(n)}\}_{n=1}^{N}$

inputs class labels

Training a Single Neuron



desired result of training:

neuron outputs $\mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w}) \approx 1$ for $t^{(n)} = 1$

neuron outputs $\mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w}) \approx 0$ for $t^{(n)} = 0$

class $t^{(n)} = 0$

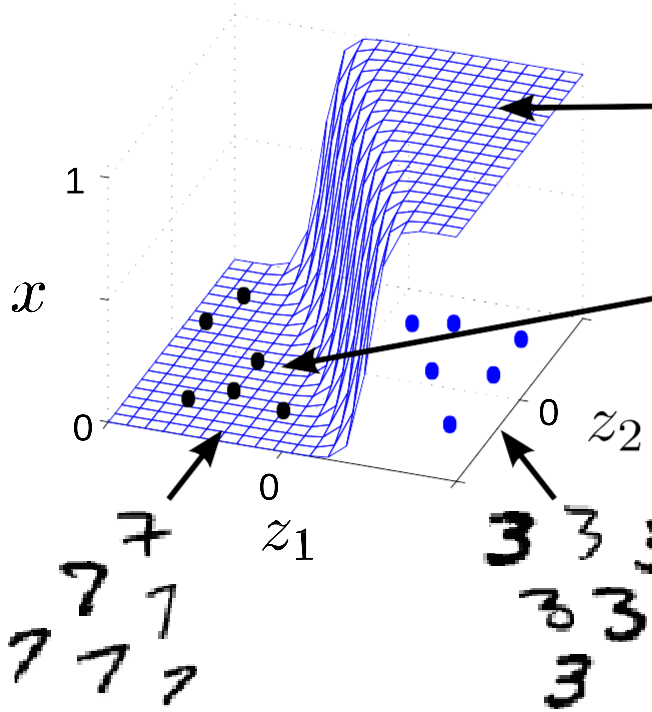
class $t^{(n)} = 1$

training data

$\{\boldsymbol{z}^{(n)}\}_{n=1}^{N}$ $\{t^{(n)}\}_{n=1}^{N}$

inputs class labels

Training a Single Neuron



desired result of training:

neuron outputs $\mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w}) \approx 1$ for $t^{(n)} = 1$

neuron outputs $\mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w}) \approx 0$ for $t^{(n)} = 0$

class $t^{(n)} = 0$ class $t^{(n)} = 1$

training data

$\{\boldsymbol{z}^{(n)}\}_{n=1}^{N}$ $\{t^{(n)}\}_{n=1}^{N}$

inputs class labels

objective function:

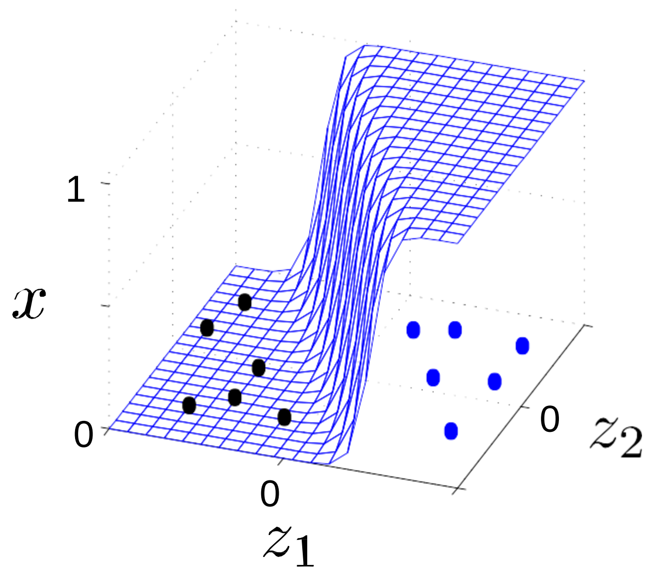
$$G(\boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w}) + (1 - t^n) \log \left(1 - \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w})\right)\right] \geq 0$$

surprise $-\log p(\text{outcome})$ when observing $t^{(n)}$

relative entropy between $\mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w})$ and $t^{(n)}$

} encourages neuron output to match training data

Training a Single Neuron



training data

$\{\boldsymbol{z}^{(n)}\}_{n=1}^{N}$ $\{t^{(n)}\}_{n=1}^{N}$

inputs class labels

objective function:

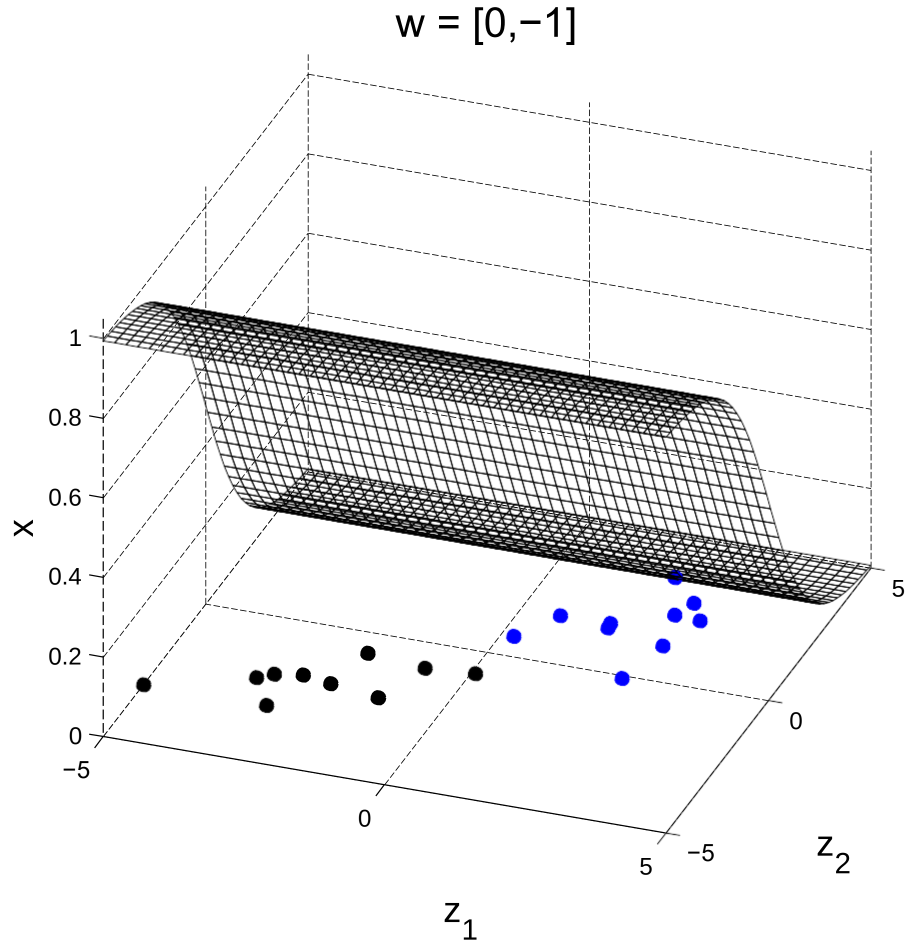
$$G(\boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w}) + (1 - t^n) \log \left(1 - \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w})\right)\right] \quad \geq 0$$

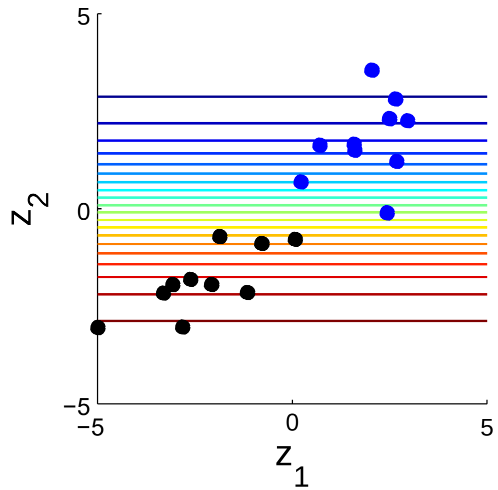
$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} G(\boldsymbol{w})$$ choose the weights that minimise the network's surprise about the training data

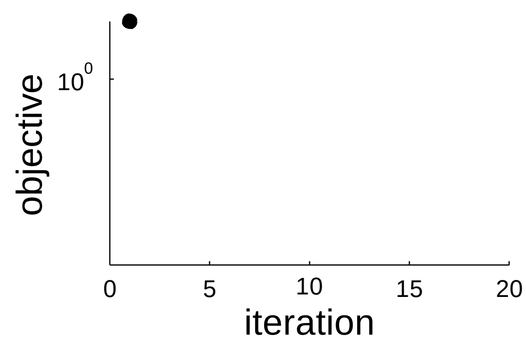
$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{w}} G(\boldsymbol{w}) = \sum_n \frac{\mathrm{d}G(\boldsymbol{w})}{\mathrm{d}x^{(n)}} \frac{\mathrm{d}x^{(n)}}{\mathrm{d}\boldsymbol{w}} = -\sum_n (t^{(n)} - x^{(n)}) \boldsymbol{z}^{(n)}$$ = prediction error x feature

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{w}} G(\boldsymbol{w})$$ iteratively step down the objective (gradient points up hill)

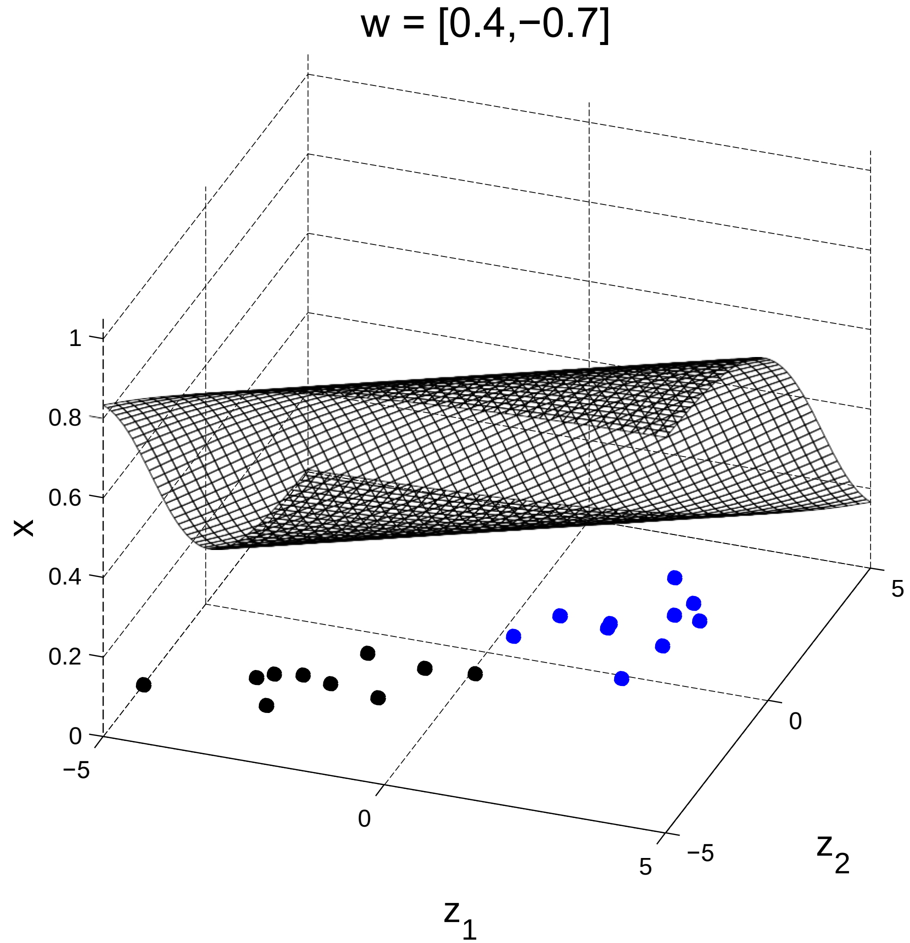
Training a Single Neuron

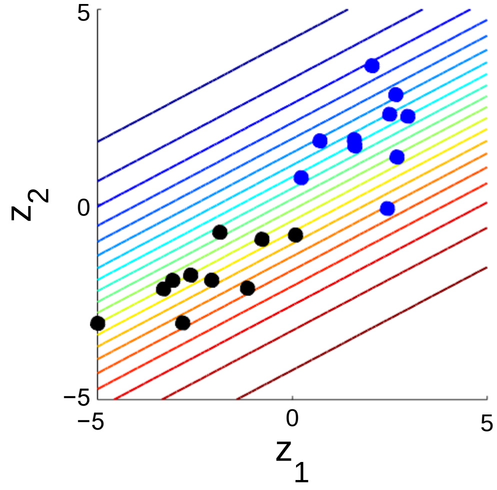


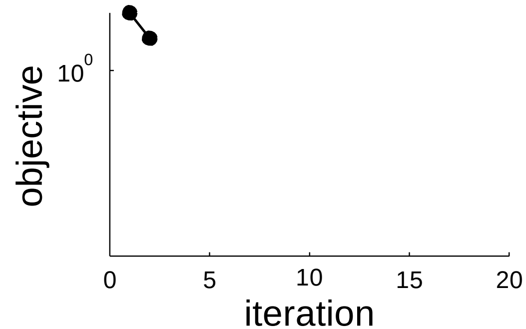




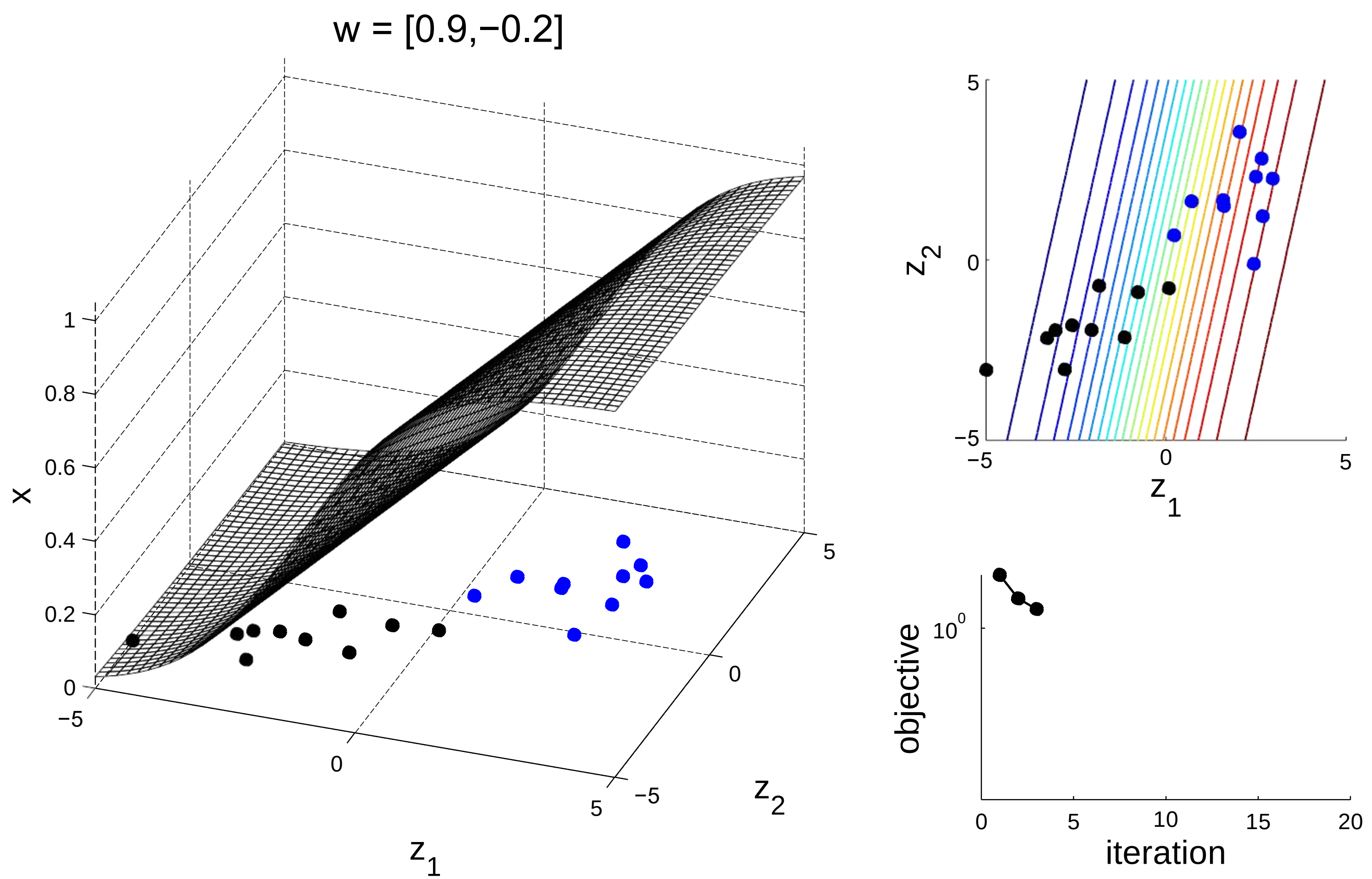
Training a Single Neuron



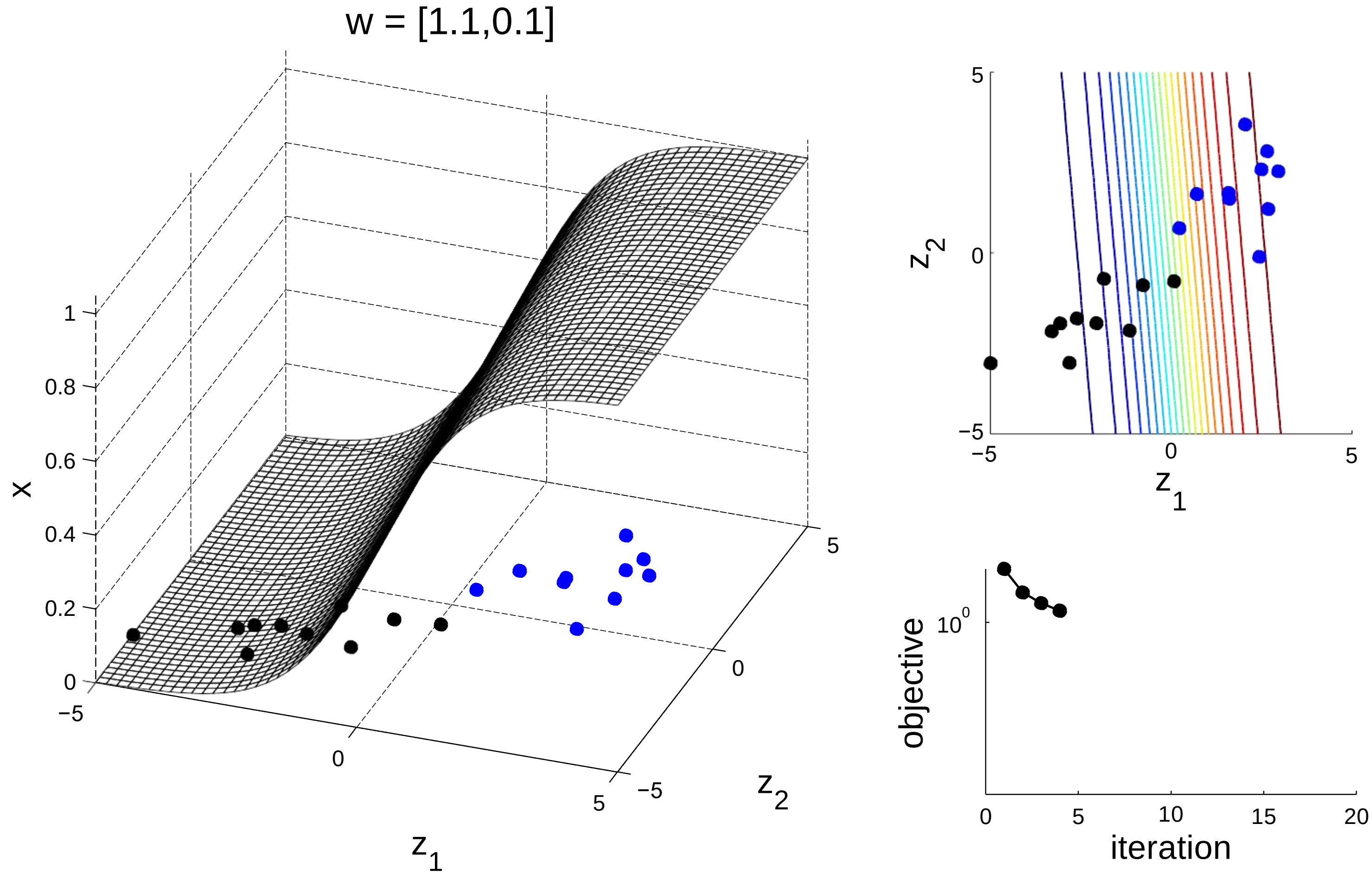




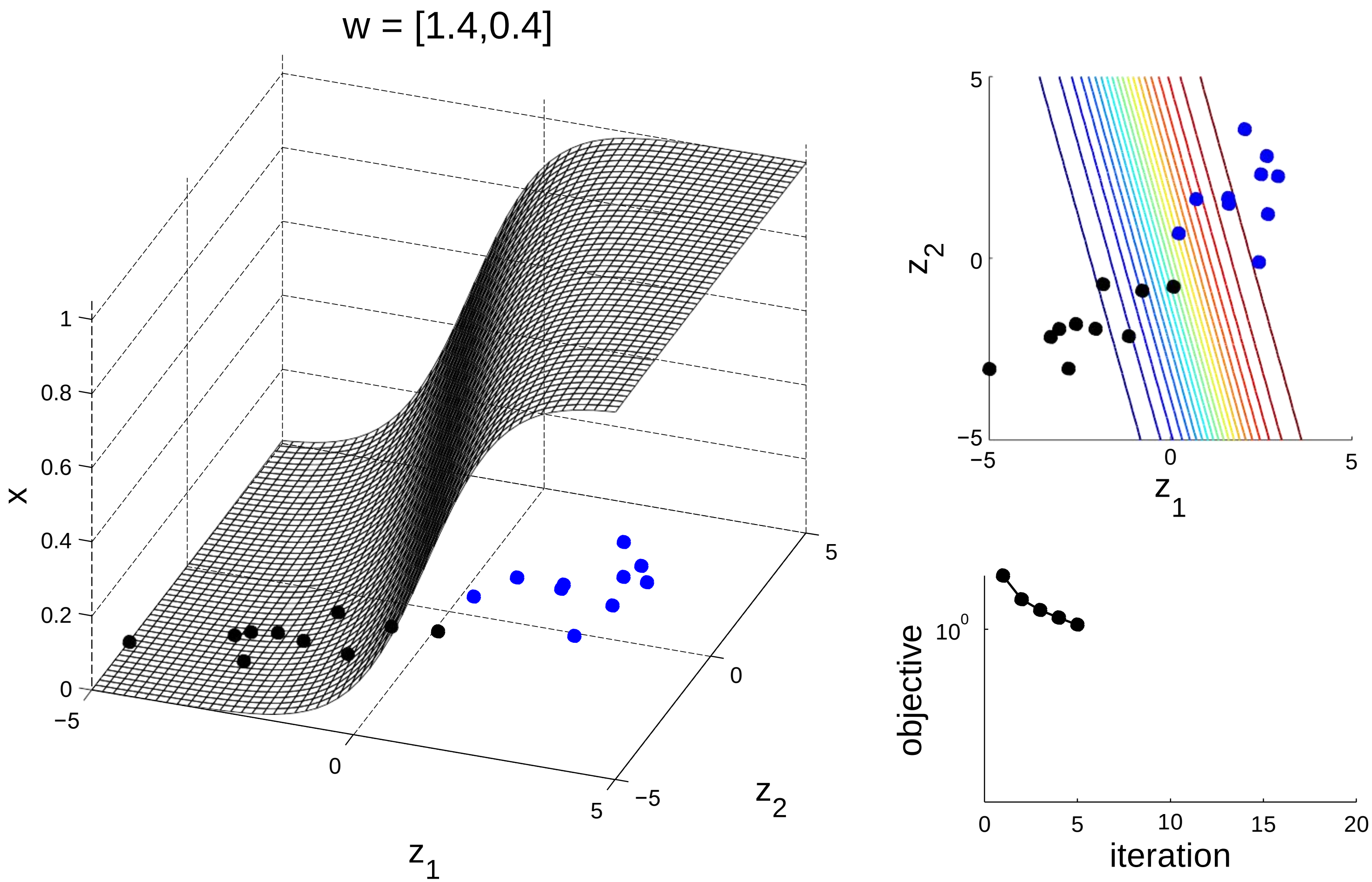
Training a Single Neuron



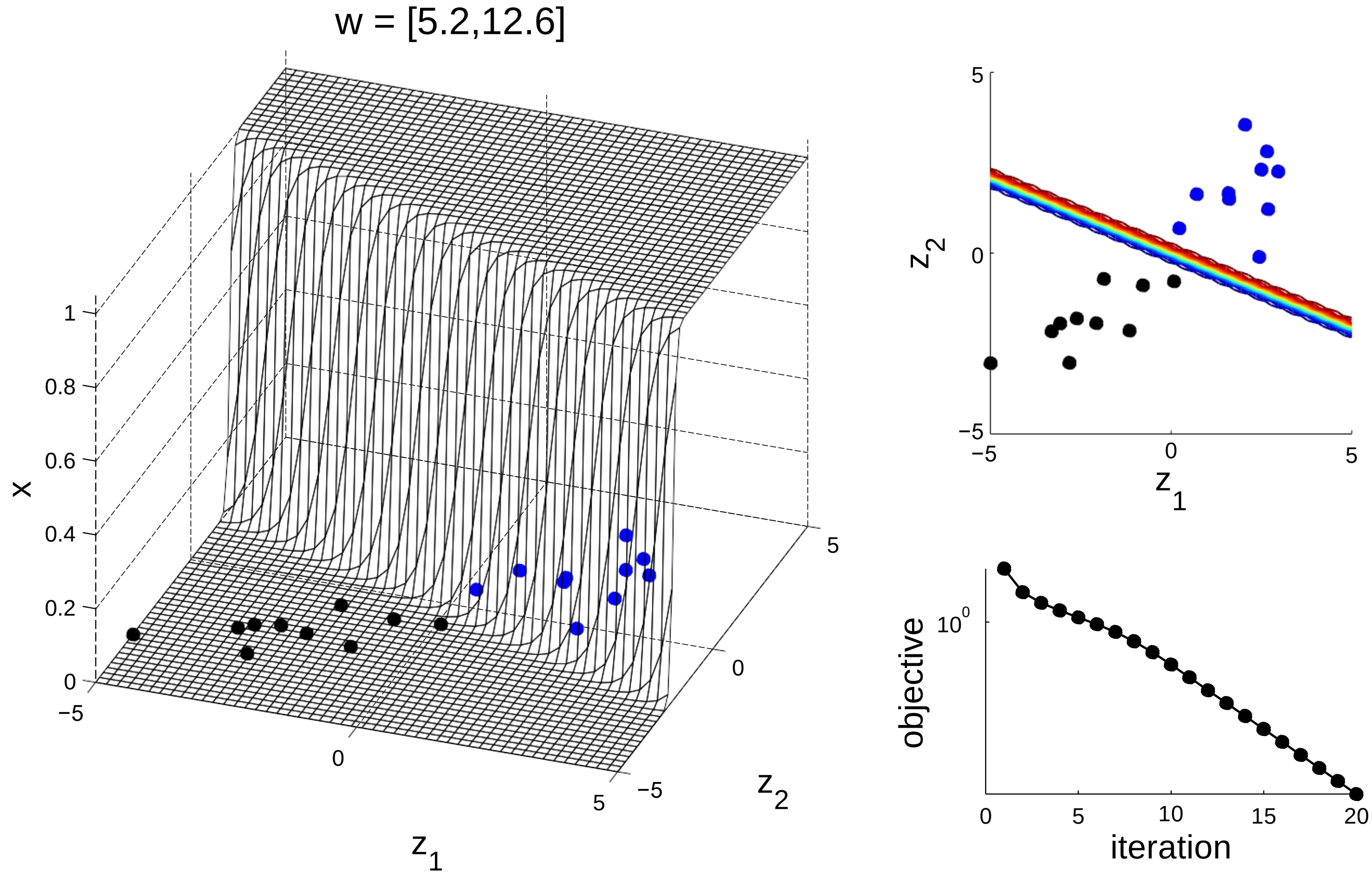
Training a Single Neuron



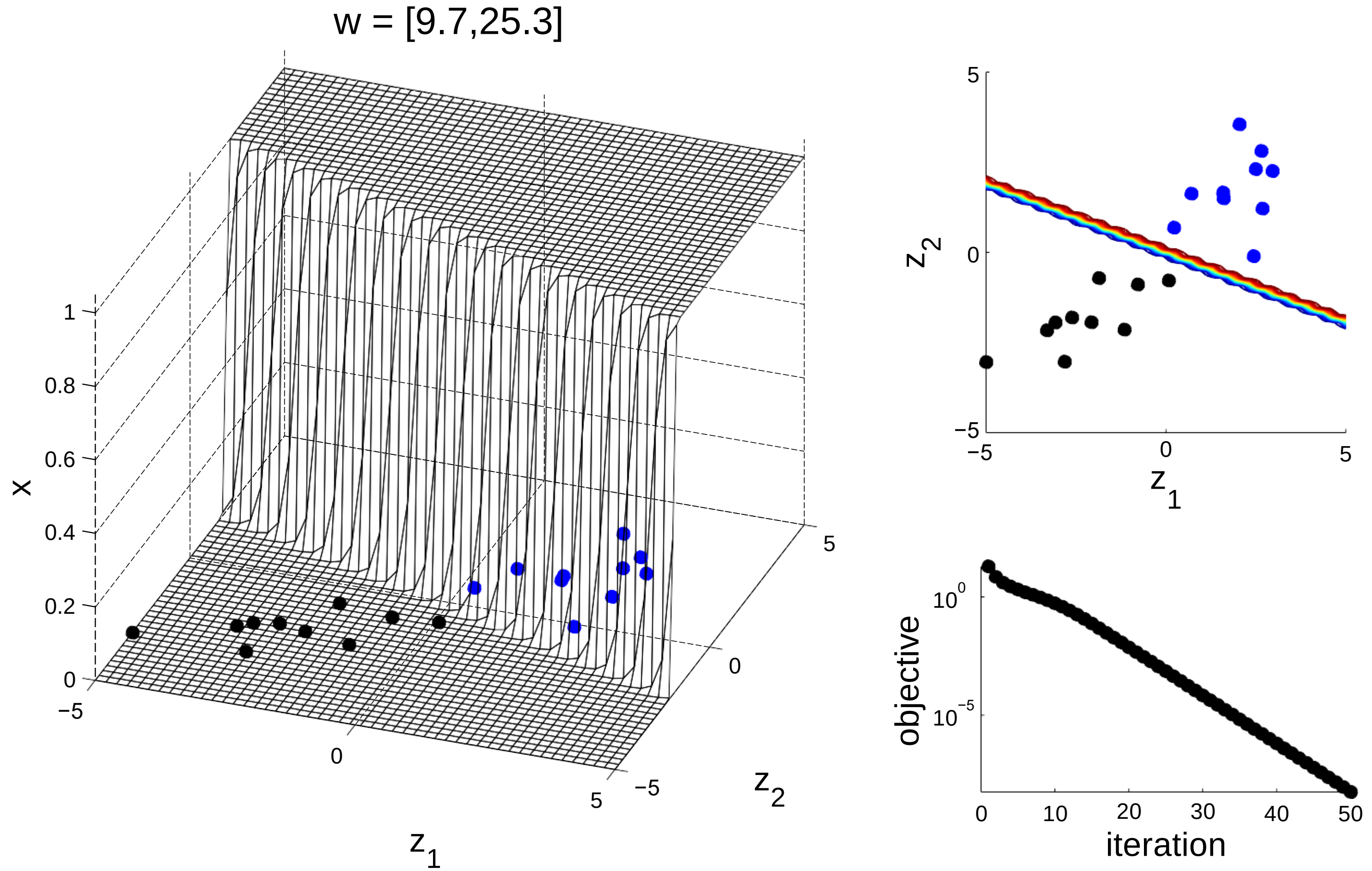
Training a Single Neuron



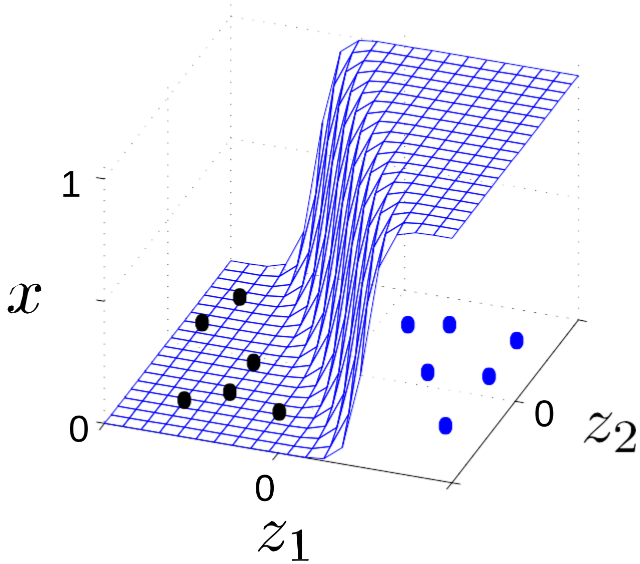
Training a Single Neuron



Training a Single Neuron



Overfitting and Weight Decay



training data

$\{\boldsymbol{z}^{(n)}\}_{n=1}^{N}$ $\{t^{(n)}\}_{n=1}^{N}$

inputs class labels

objective function:

$$G(\boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w}) + (1 - t^n) \log \left(1 - \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w})\right)\right]$$

$$E(\boldsymbol{w}) = \frac{1}{2} \sum_i w_i^2$$

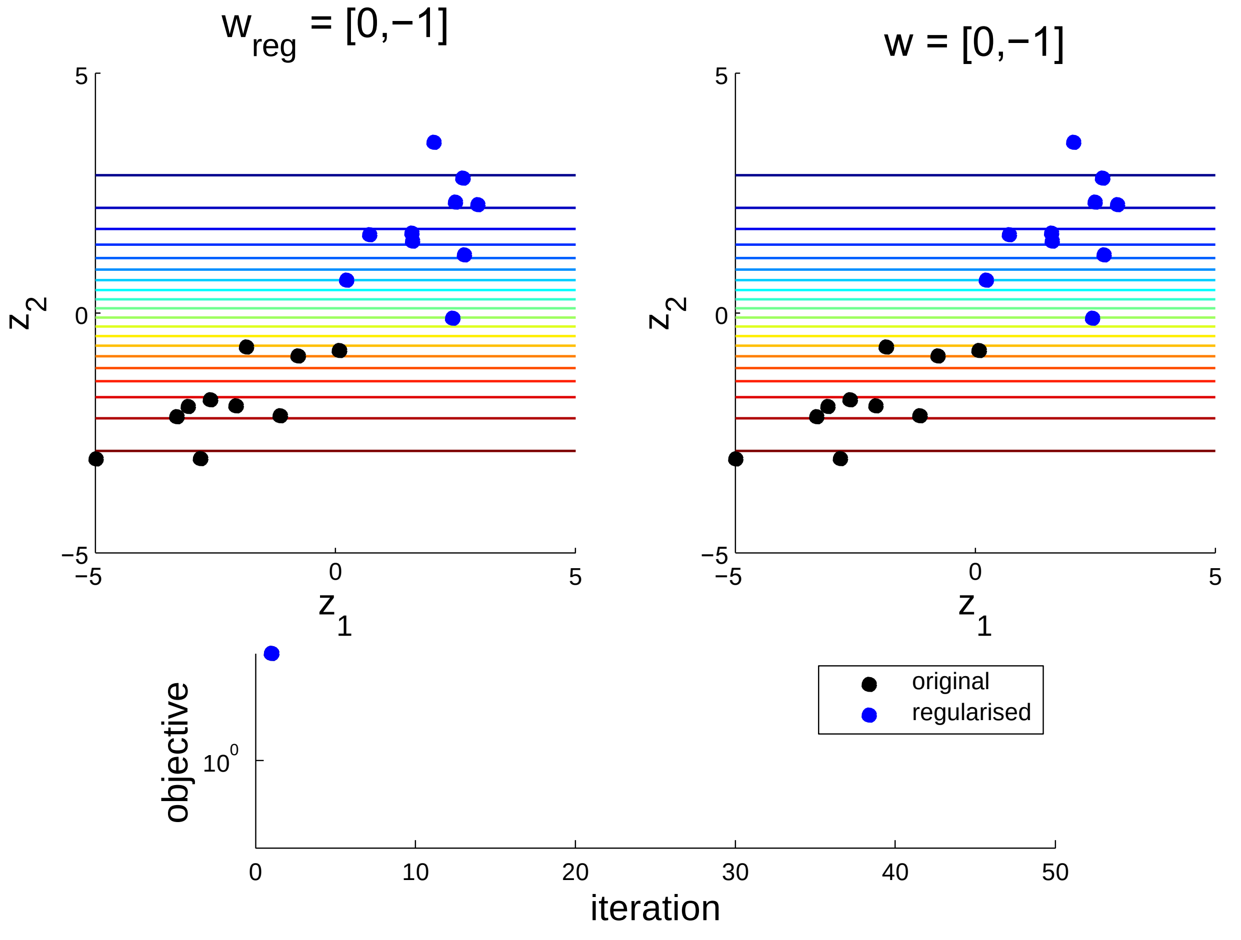
regulariser discourages the network using extreme weights

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} M(\boldsymbol{w}) = \arg\min_{\boldsymbol{w}} \left[G(\boldsymbol{w}) + \alpha E(\boldsymbol{w})\right]$$

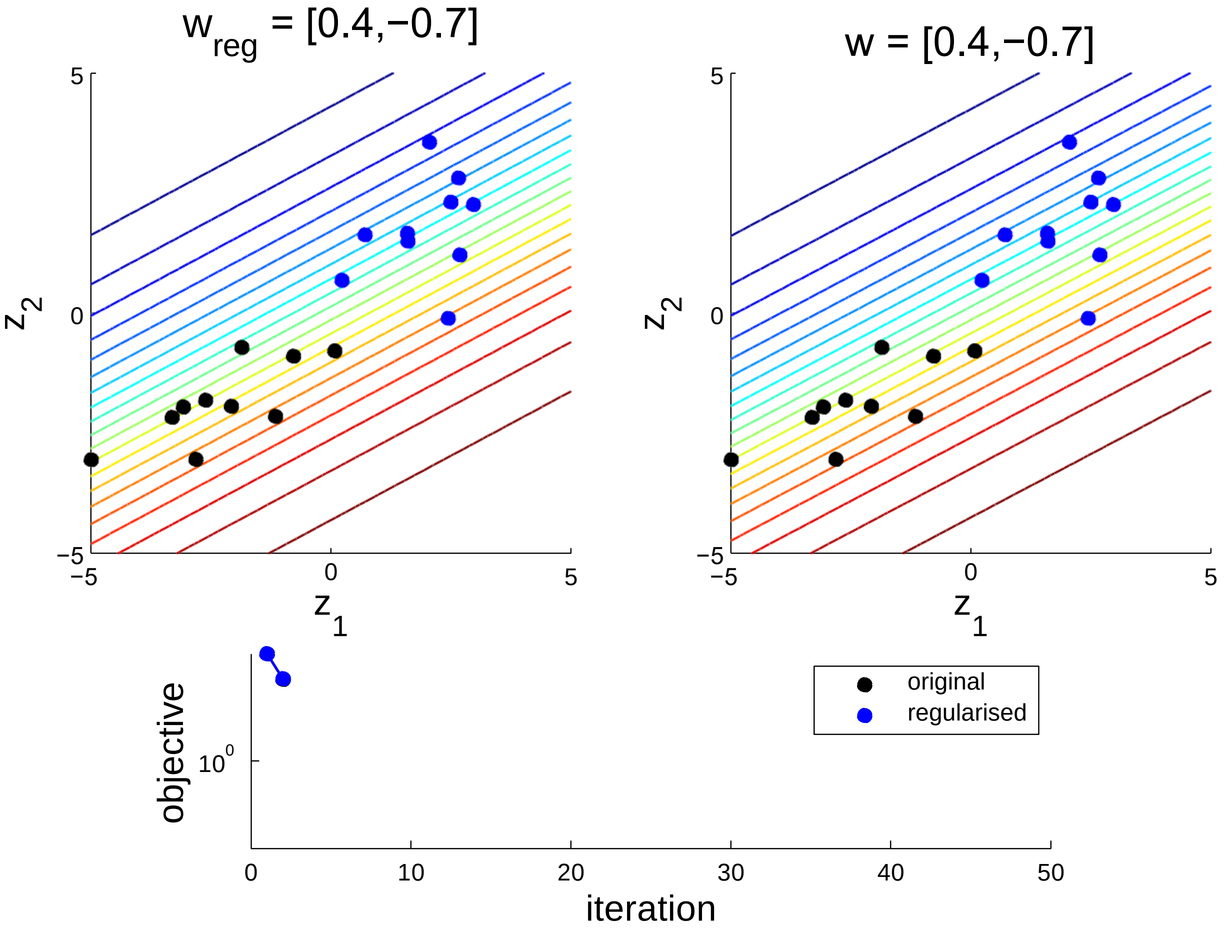
$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{w}} M(\boldsymbol{w}) = -\sum_n (t^{(n)} - x^{(n)}) \boldsymbol{z}^{(n)} + \alpha \boldsymbol{w}$$

weight decay - shrinks weights towards zero

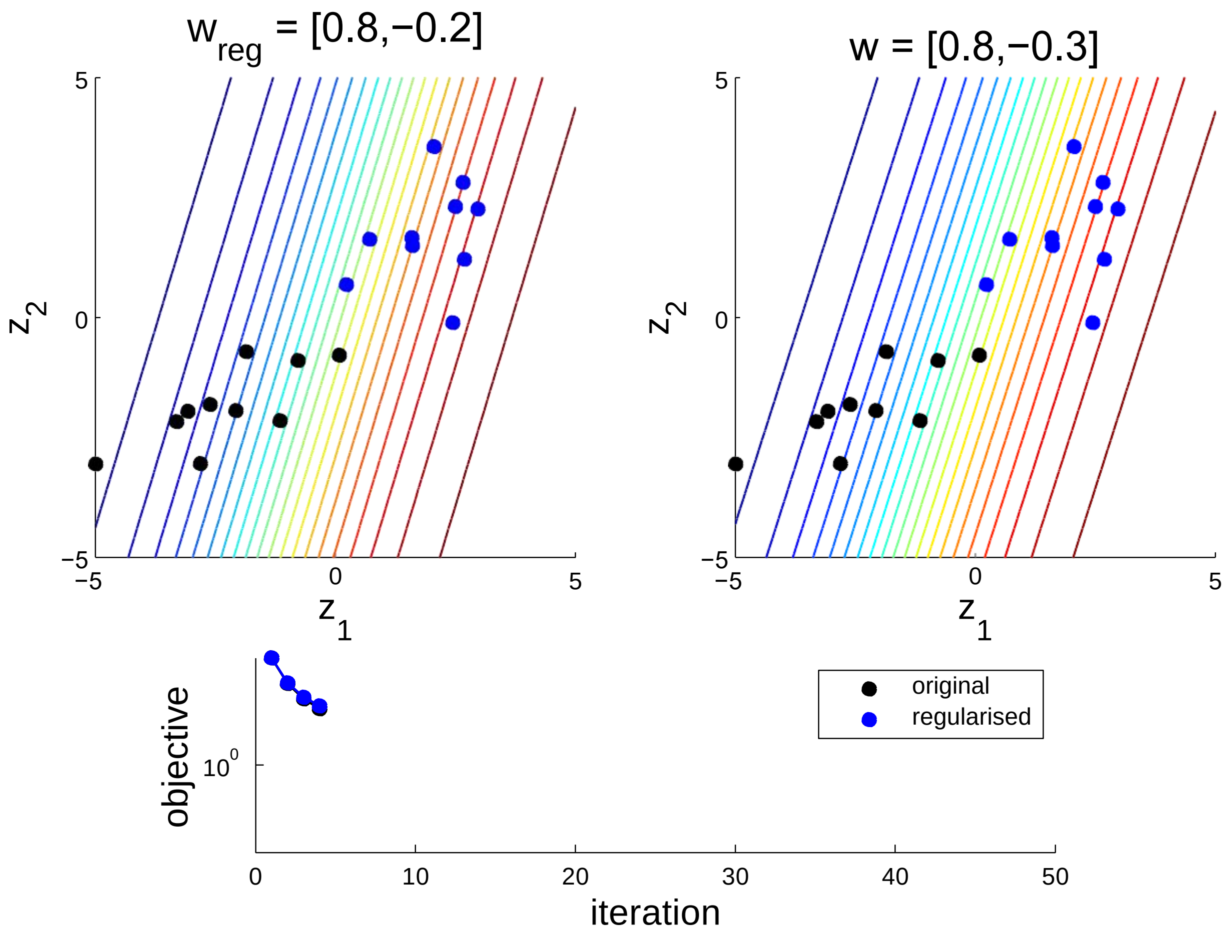
Training a Single Neuron (cont’d)



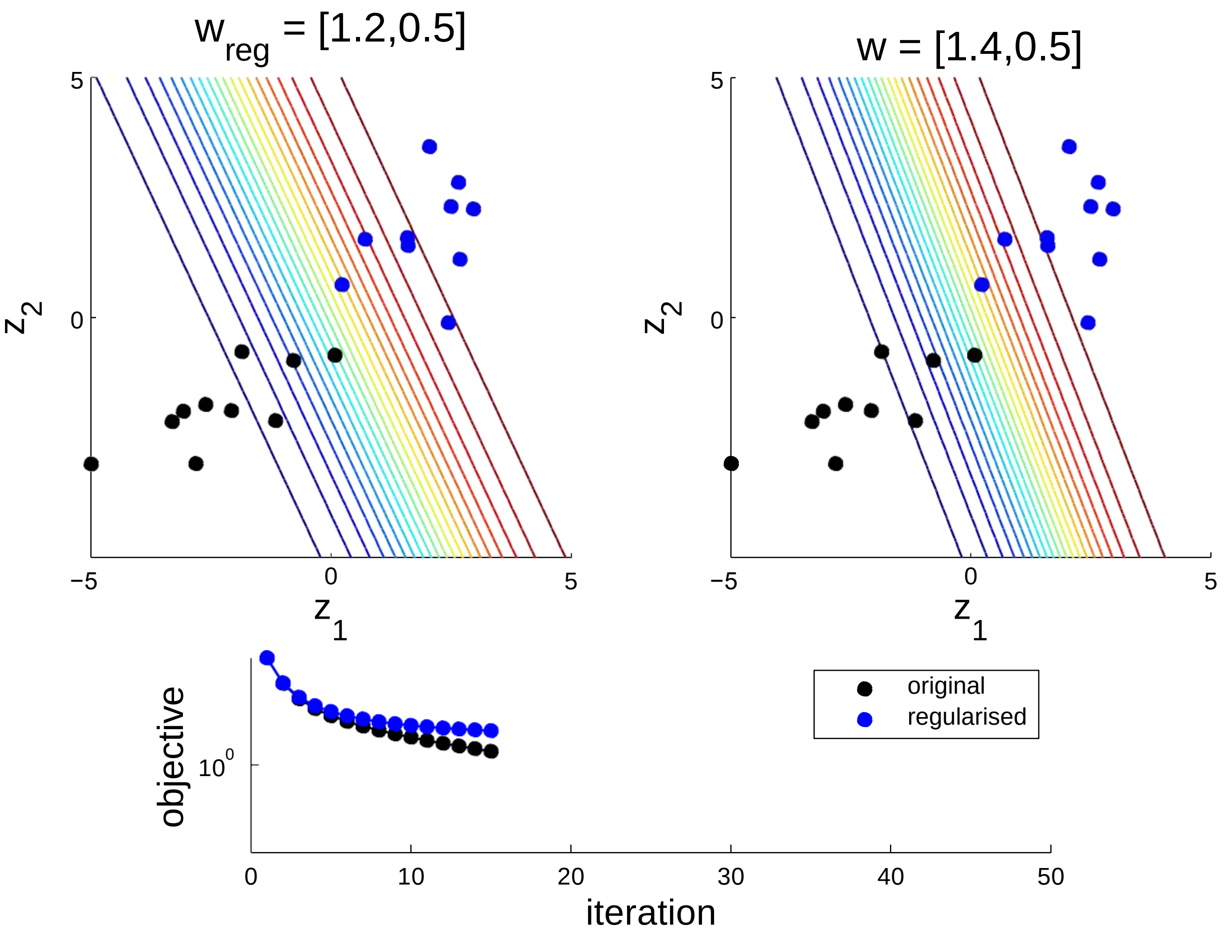
Training a Single Neuron (cont'd)



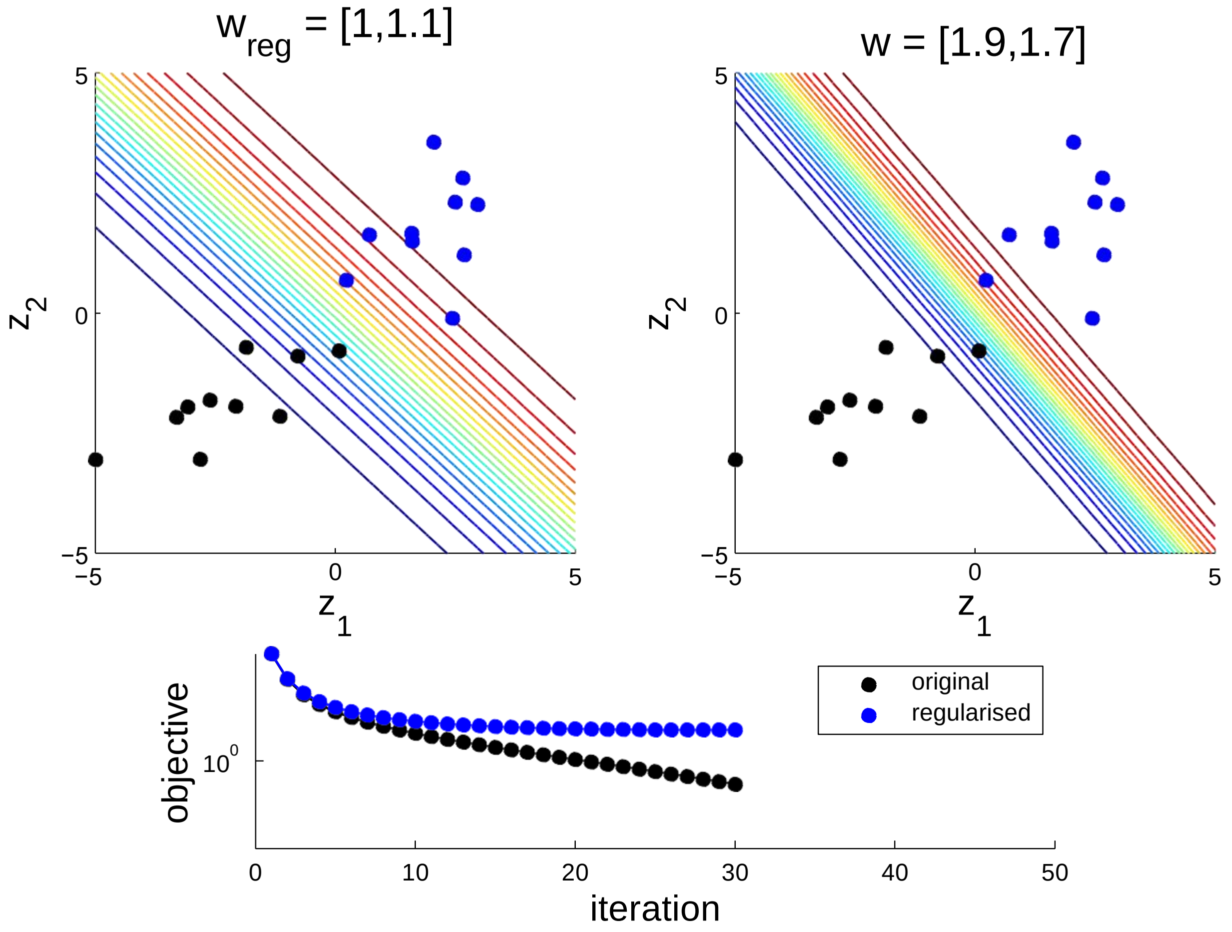
Training a Single Neuron (cont'd)



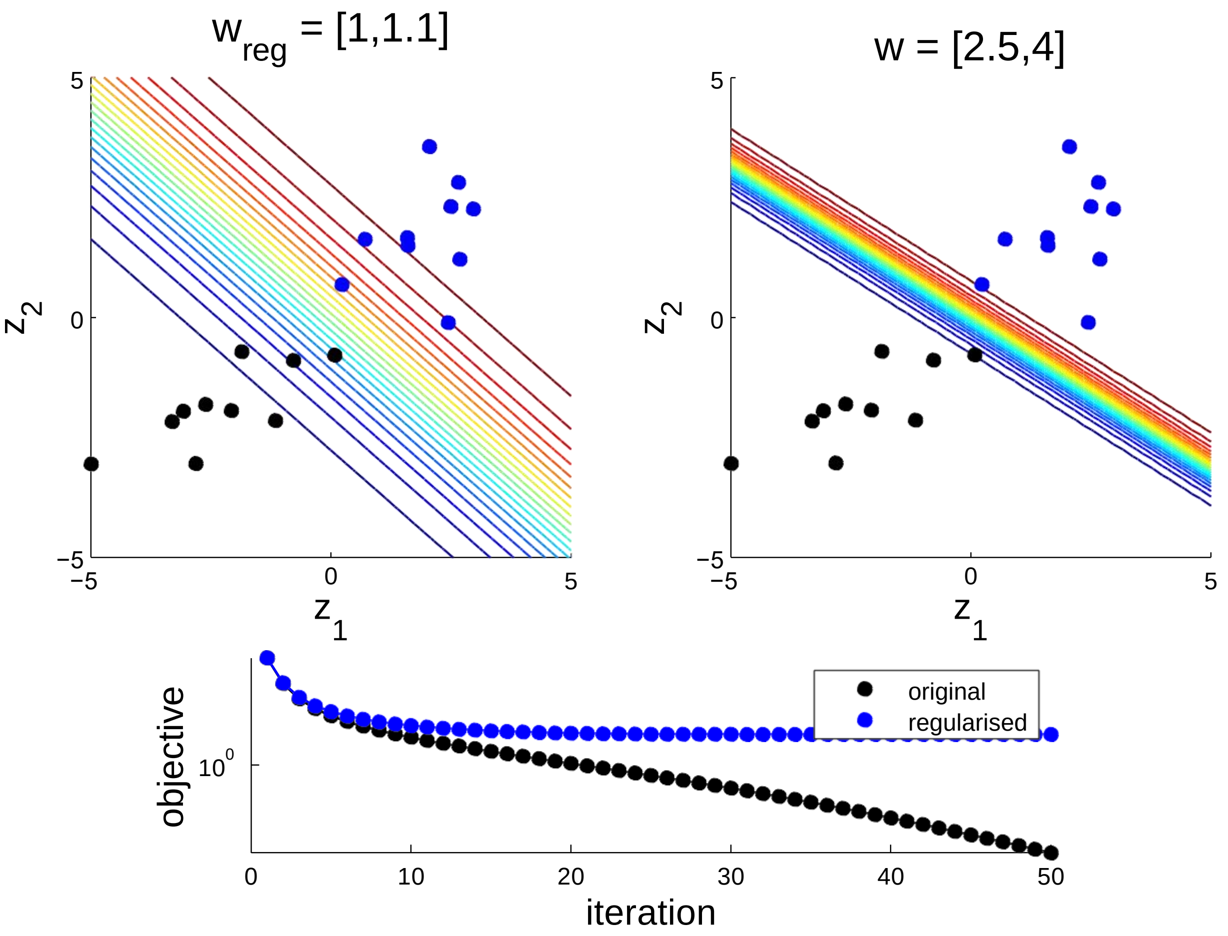
Training a Single Neuron (cont’d)



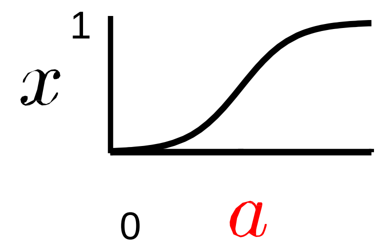
Training a Single Neuron (cont'd)

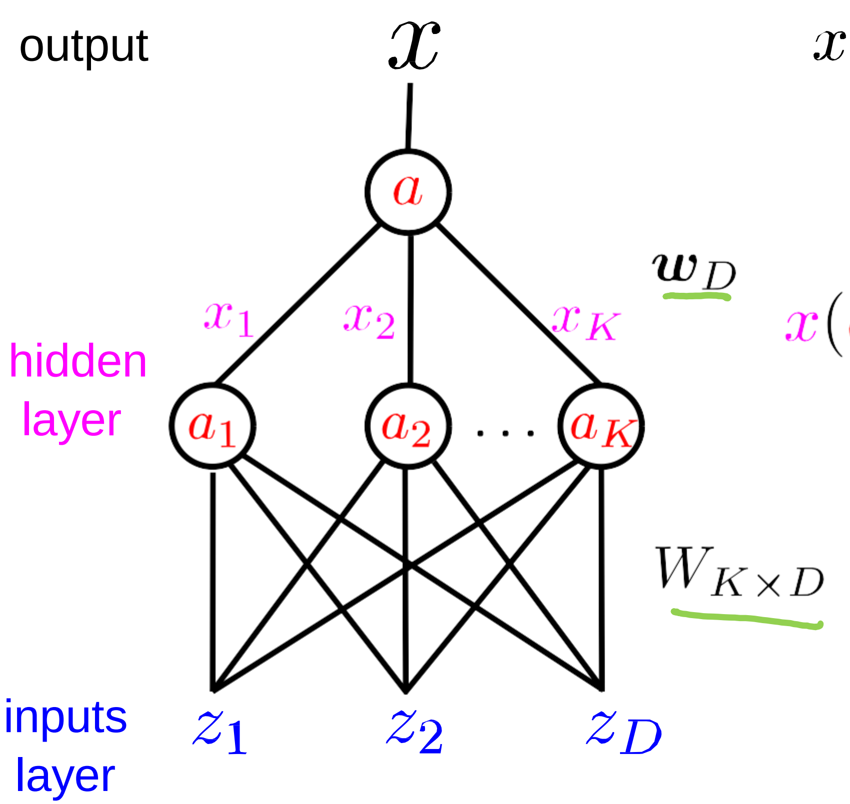


Training a Single Neuron (cont'd)



Single Hidden Layer Neural Networks





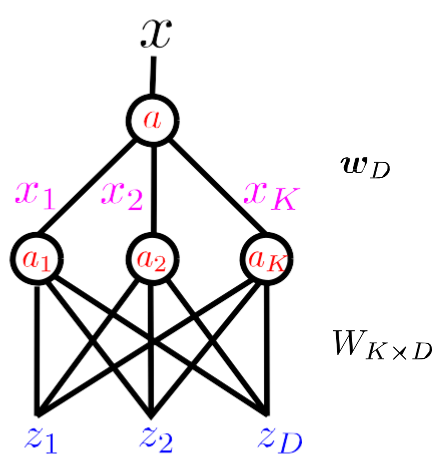
$$x(a) = \frac{1}{1+\exp(-a)}$$

$$a = \sum_{k=1}^{K} w_k x_k$$

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

$$a_k = \sum_{d=1}^{D} W_{k,d} z_d$$

Training a Neural Network with a Single Hidden Layer



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

$$a = \sum_{k=1}^{K} w_k x_k$$

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

$$a_k = \sum_{d=1}^{D} W_{k,d} z_d$$

objective function:

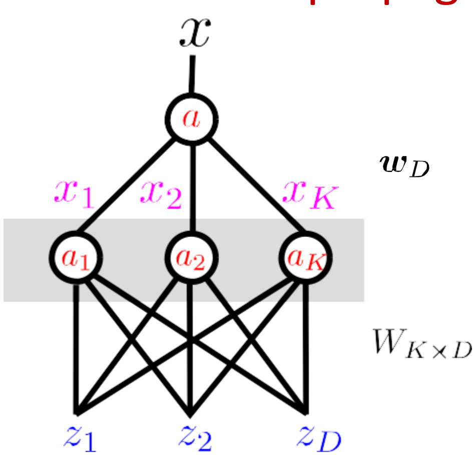
$$G(W, \boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}^{(n)} + (1 - t^n) \log \left(1 - \mathrm{x}^{(n)}\right)\right]$$ likelihood same as before

$$E(W, \boldsymbol{w}) = \frac{1}{2} \sum_i w_i^2 + \frac{1}{2} \sum_{ij} W_{ij}^2$$ regulariser discourages extreme weights

$$\{W, \boldsymbol{w}^*\} = \arg\min_{W,\boldsymbol{w}} M(W, \boldsymbol{w}) = \arg\min_{W,\boldsymbol{w}} \left[G(W, \boldsymbol{w}) + \alpha E(W, \boldsymbol{w})\right]$$

Training a Neural Network with a Single Hidden Layer

Networks with hidden layers can be fit using gradient descent using an algorithm called back-propagation.



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

$$a = \sum_{k=1}^{K} w_k x_k$$

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

$$a_k = \sum_{d=1}^{D} W_{k,d} z_d$$

objective function:

$$G(W, \boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}^{(n)} + (1 - t^n) \log \left(1 - \mathrm{x}^{(n)}\right)\right]$$ likelihood same as before

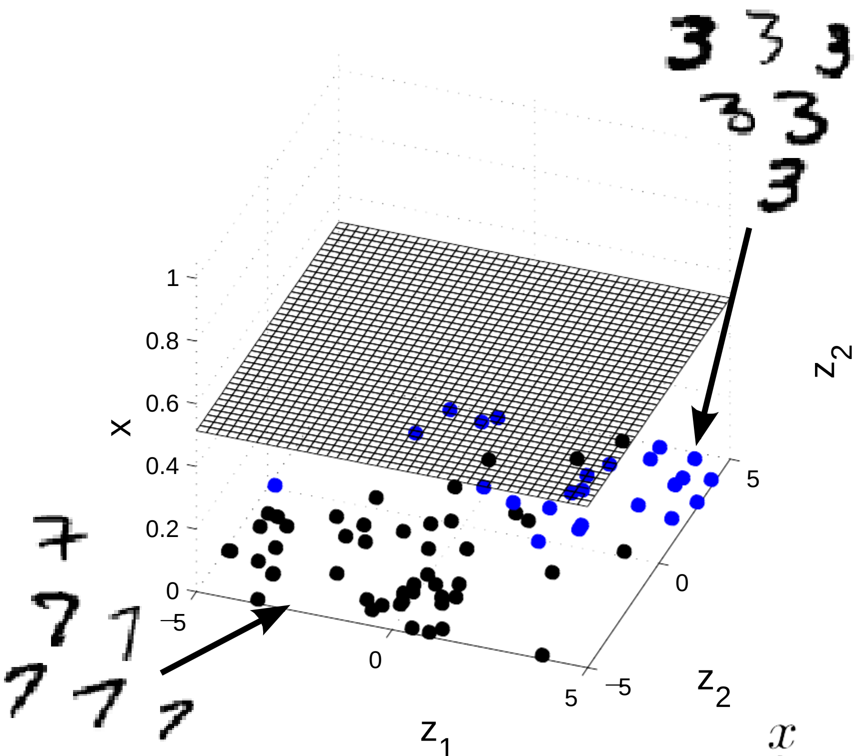
$$E(W, \boldsymbol{w}) = \tfrac{1}{2} \sum_i w_i^2 + \tfrac{1}{2} \sum_{ij} W_{ij}^2$$ regulariser discourages extreme weights

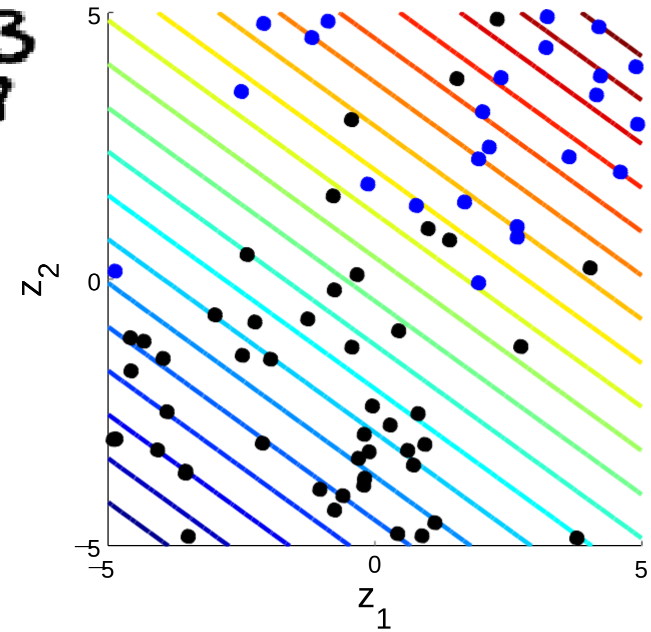
$$\{W, \boldsymbol{w}^*\} = \underset{W,\boldsymbol{w}}{\arg\min}\, M(W, \boldsymbol{w}) = \underset{W,\boldsymbol{w}}{\arg\min} \left[G(W, \boldsymbol{w}) + \alpha E(W, \boldsymbol{w})\right]$$

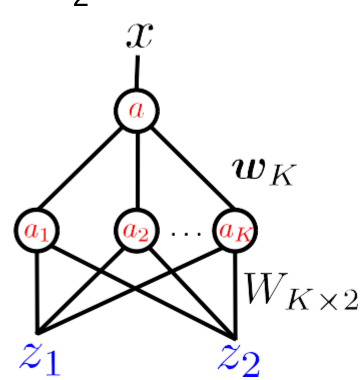
$$\frac{\mathrm{d}G(W, \boldsymbol{w})}{\mathrm{d}W_{ij}} = \sum_n \frac{\mathrm{d}G(W, \boldsymbol{w})}{\mathrm{d}x^{(n)}} \frac{\mathrm{d}x^{(n)}}{\mathrm{d}W_{ij}} = \sum_n \frac{\mathrm{d}G(W, \boldsymbol{w})}{\mathrm{d}x^{(n)}} \frac{\mathrm{d}x^{(n)}}{\mathrm{d}a^{(n)}} \frac{\mathrm{d}a^{(n)}}{\mathrm{d}W_{ij}}$$

$$= \sum_n \frac{\mathrm{d}G(W, \boldsymbol{w})}{\mathrm{d}x^{(n)}} \frac{\mathrm{d}x^{(n)}}{\mathrm{d}a^{(n)}} \frac{\mathrm{d}a^{(n)}}{\mathrm{d}x_i^{(n)}} \frac{\mathrm{d}x_i^{(n)}}{\mathrm{d}W_{ij}} = \sum_n \frac{\mathrm{d}G(W, \boldsymbol{w})}{\mathrm{d}x^{(n)}} \frac{\mathrm{d}x^{(n)}}{\mathrm{d}a^{(n)}} \frac{\mathrm{d}a^{(n)}}{\mathrm{d}x_i^{(n)}} \frac{\mathrm{d}x_i^{(n)}}{\mathrm{d}a_i^{(n)}} \frac{\mathrm{d}a_i^{(n)}}{\mathrm{d}W_{ij}}$$

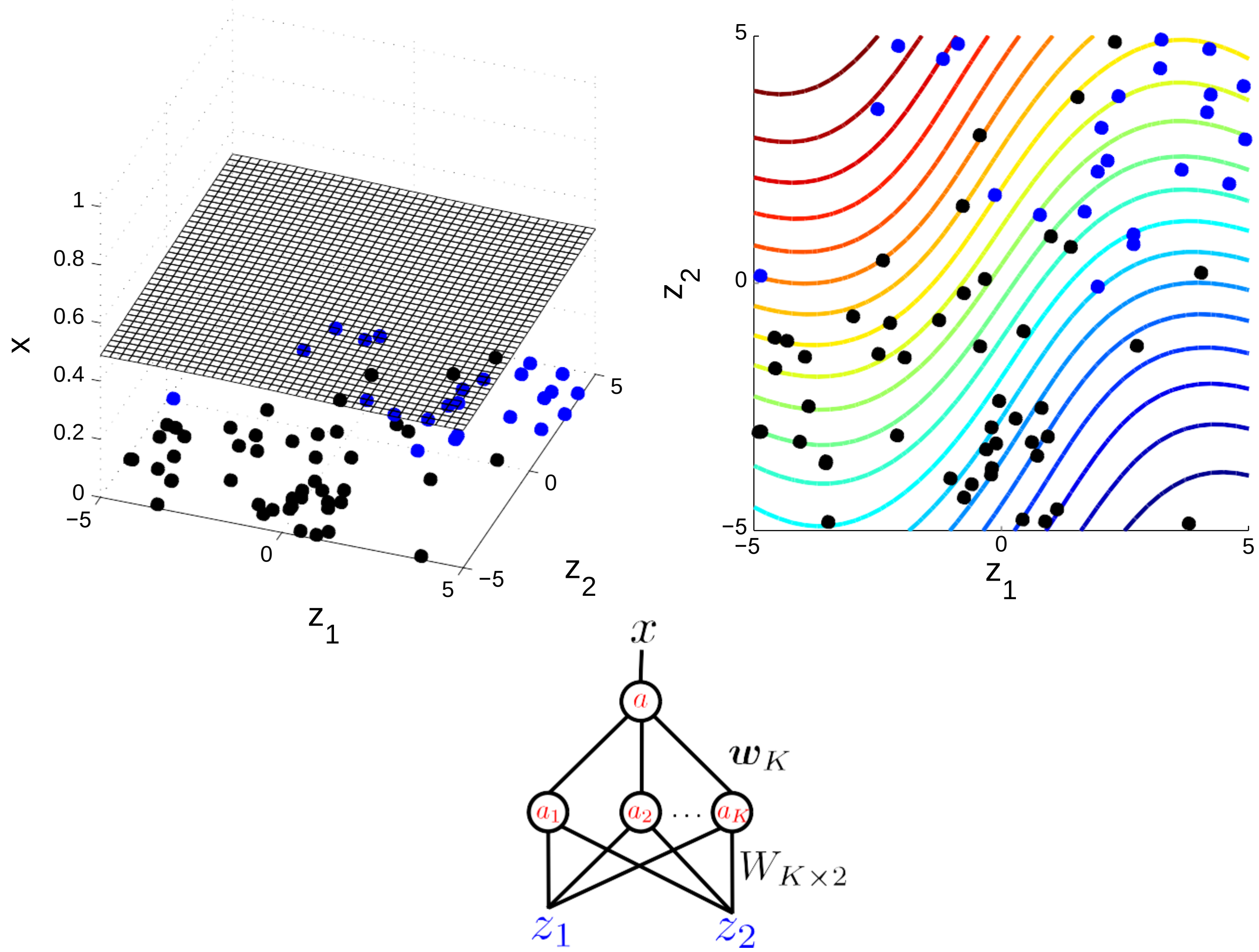
Training a Neural Network with a Single Hidden Layer



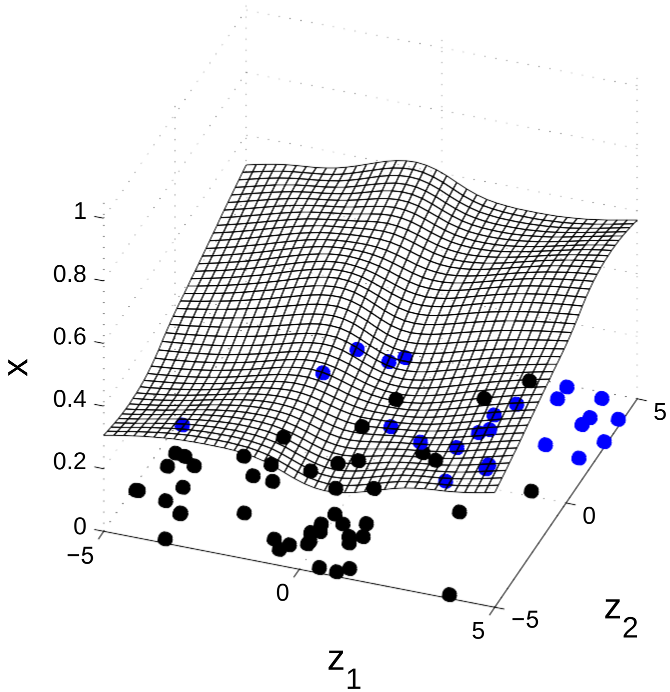


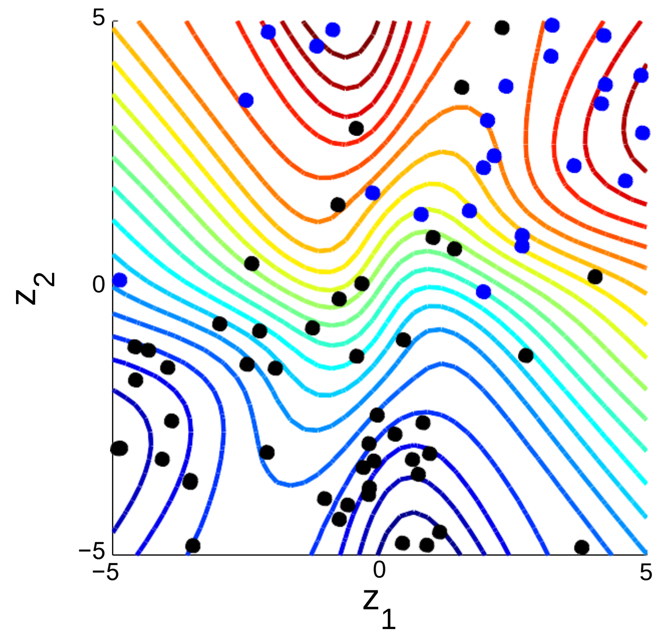


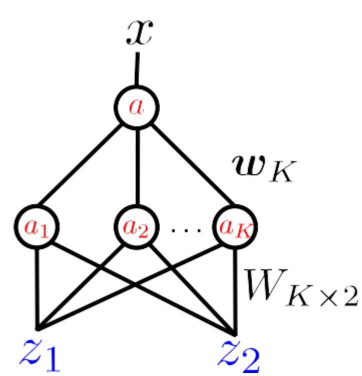
Training a Neural Network with a Single Hidden Layer



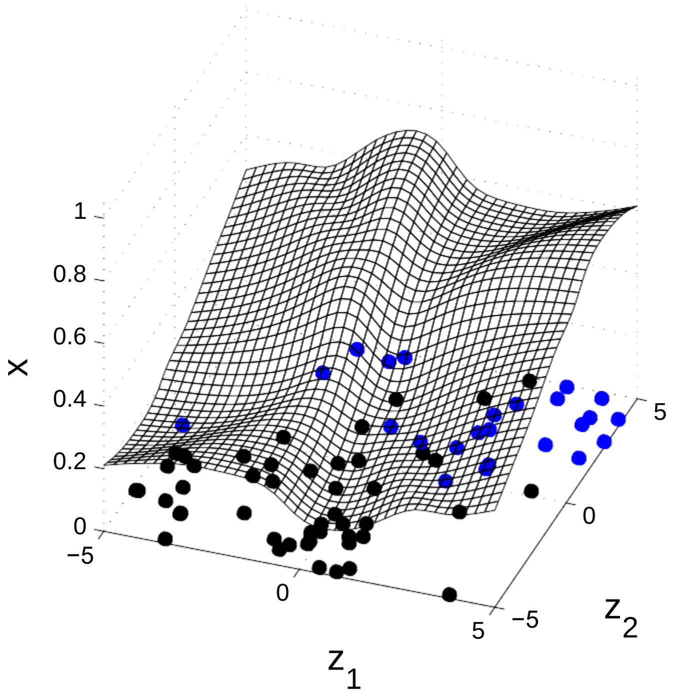
Training a Neural Network with a Single Hidden Layer

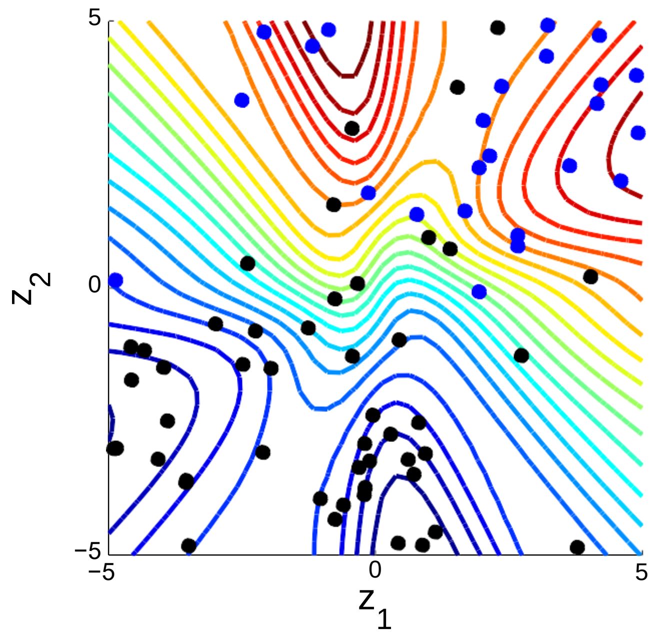


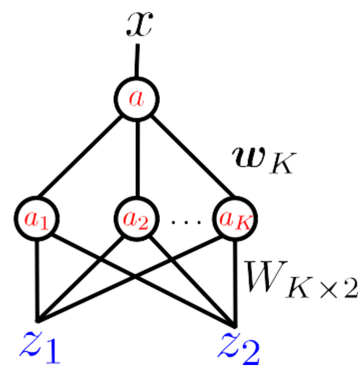




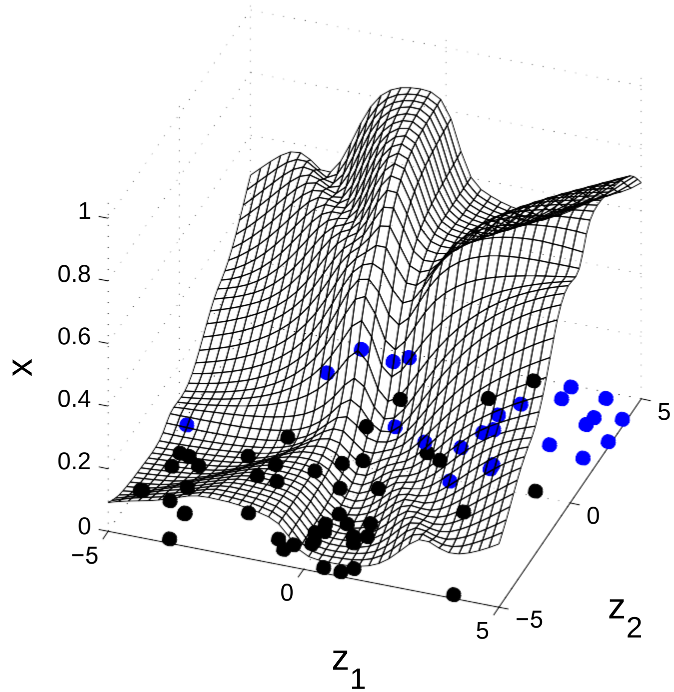
Training a Neural Network with a Single Hidden Layer

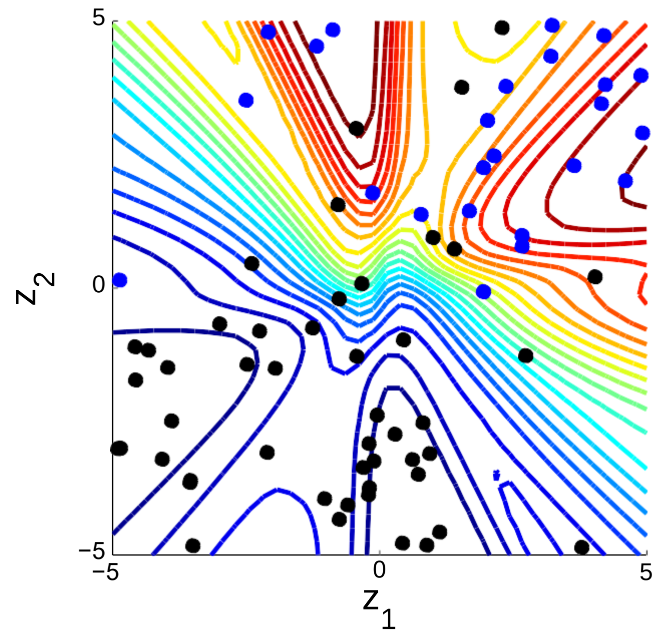


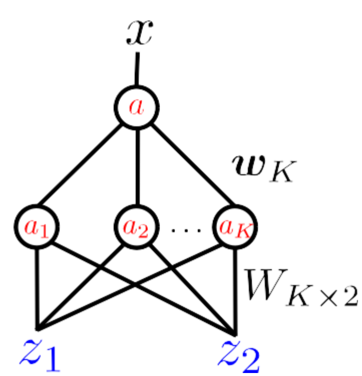




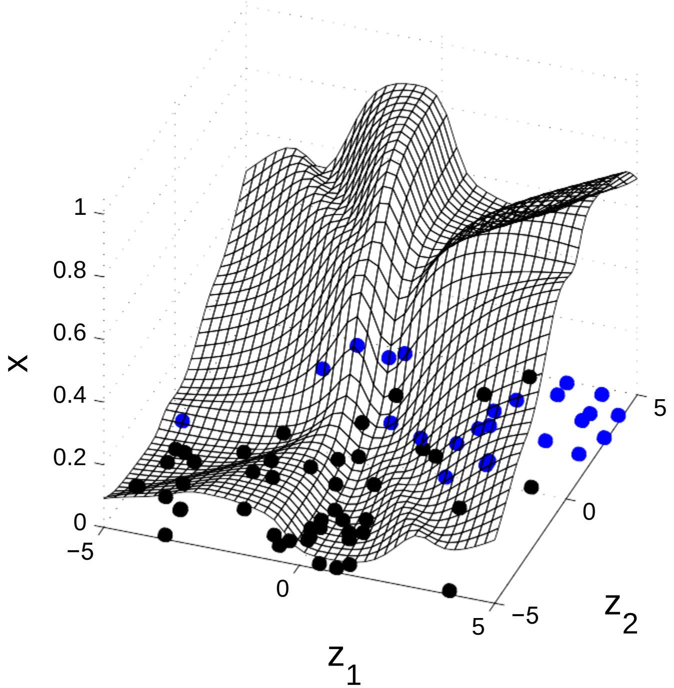
Training a Neural Network with a Single Hidden Layer

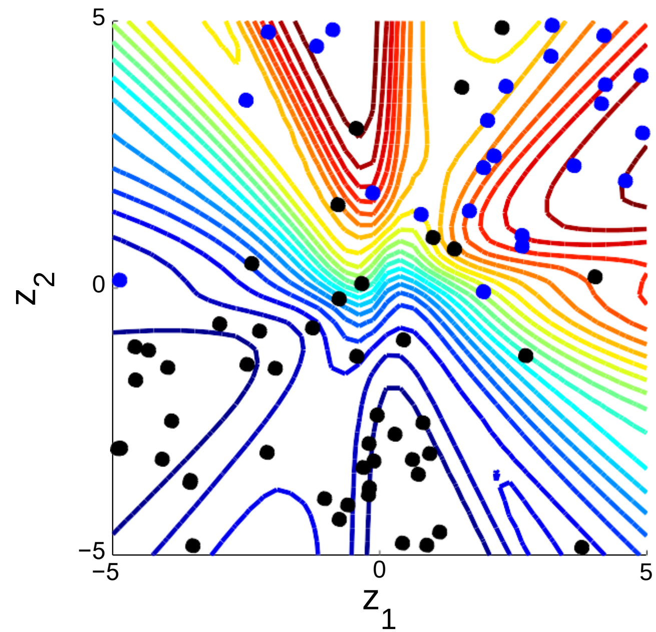


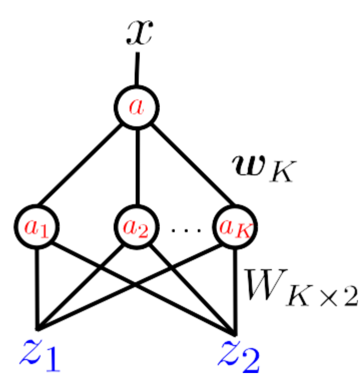




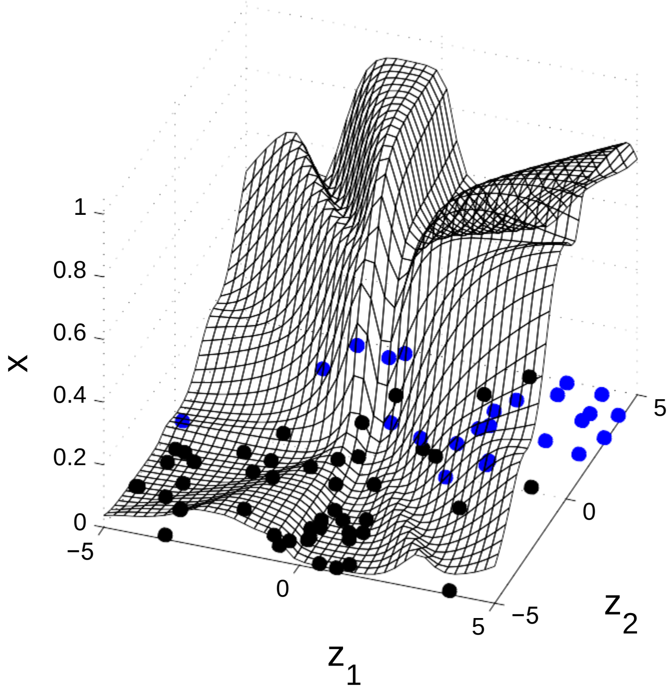
Training a Neural Network with a Single Hidden Layer

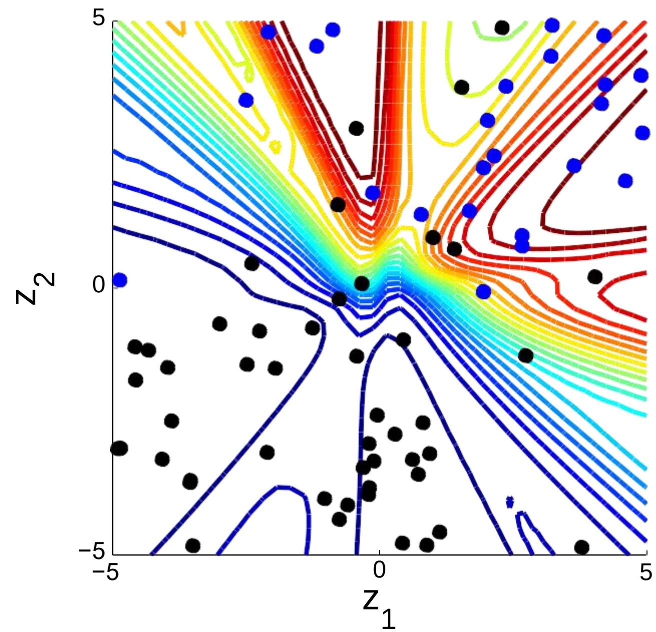


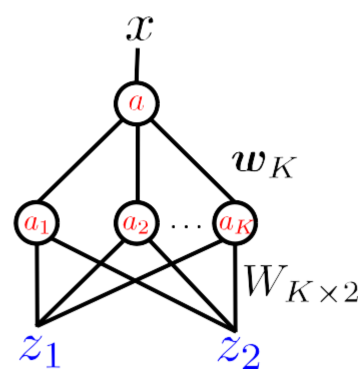




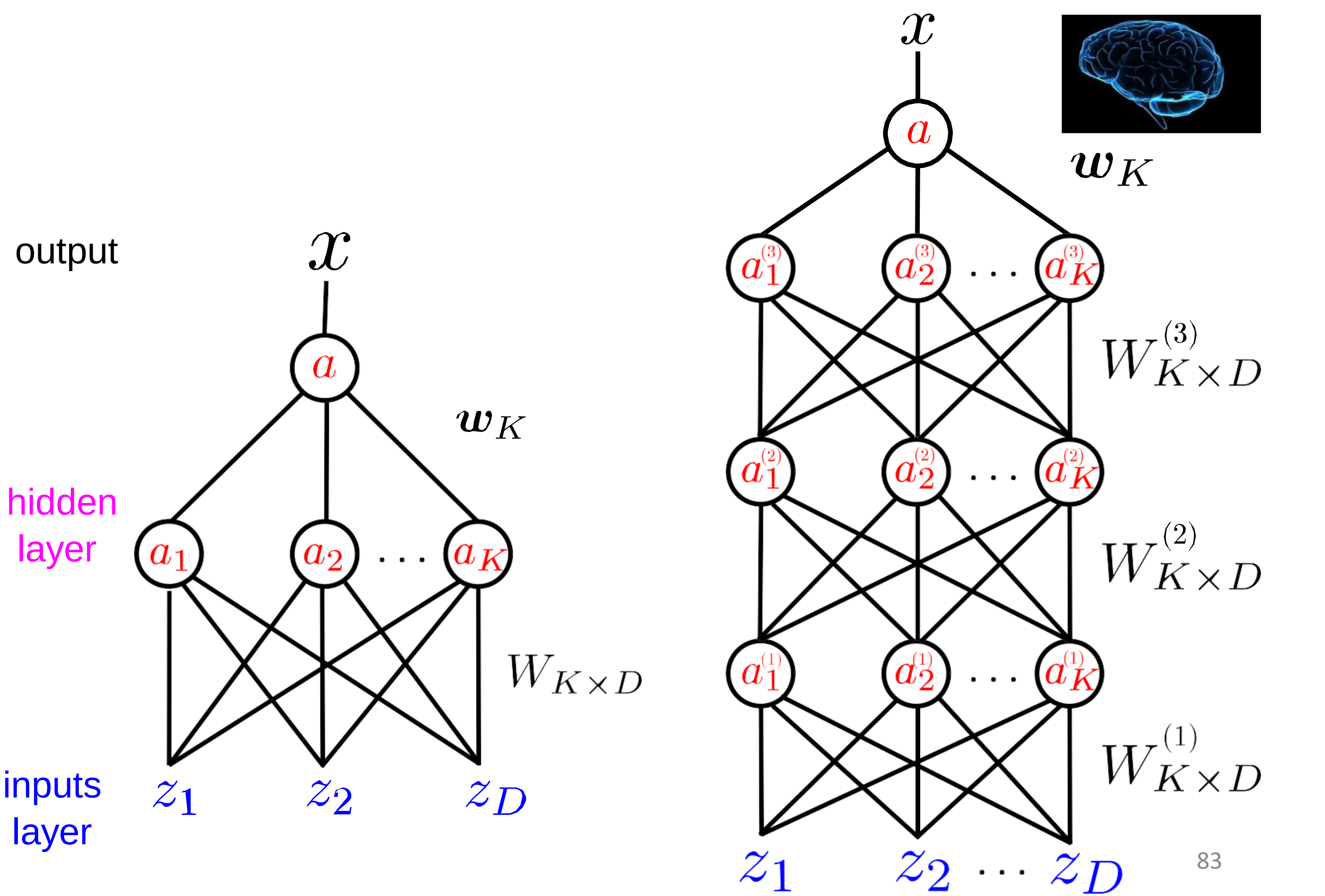
Training a Neural Network with a Single Hidden Layer







Hierarchical Models with Many Layers



What We've Covered Today…

- Unsupervised vs. Supervised Learning
  - Clustering & Dimension Reduction
  - Training, testing, & validation
  - Linear Classification
  - From Linear Classifier to Neural Nets