DATA MINING CLASSIFICATION

Nearest Neighbor Classifier
Support Vector Machines (SVM)
Logistic Regression
Neural Networks
Word Embeddings
Supervised Learning Pipeline

NEAREST NEIGHBOR CLASSIFICATION

Instance-Based Classifiers

Set of Stored Cases

Atr1	……...	AtrN	Class
			A
			B
			B
			C
			A
			C
			B

- Store the training records
- Use training records to predict the class label of unseen cases

Unseen Case

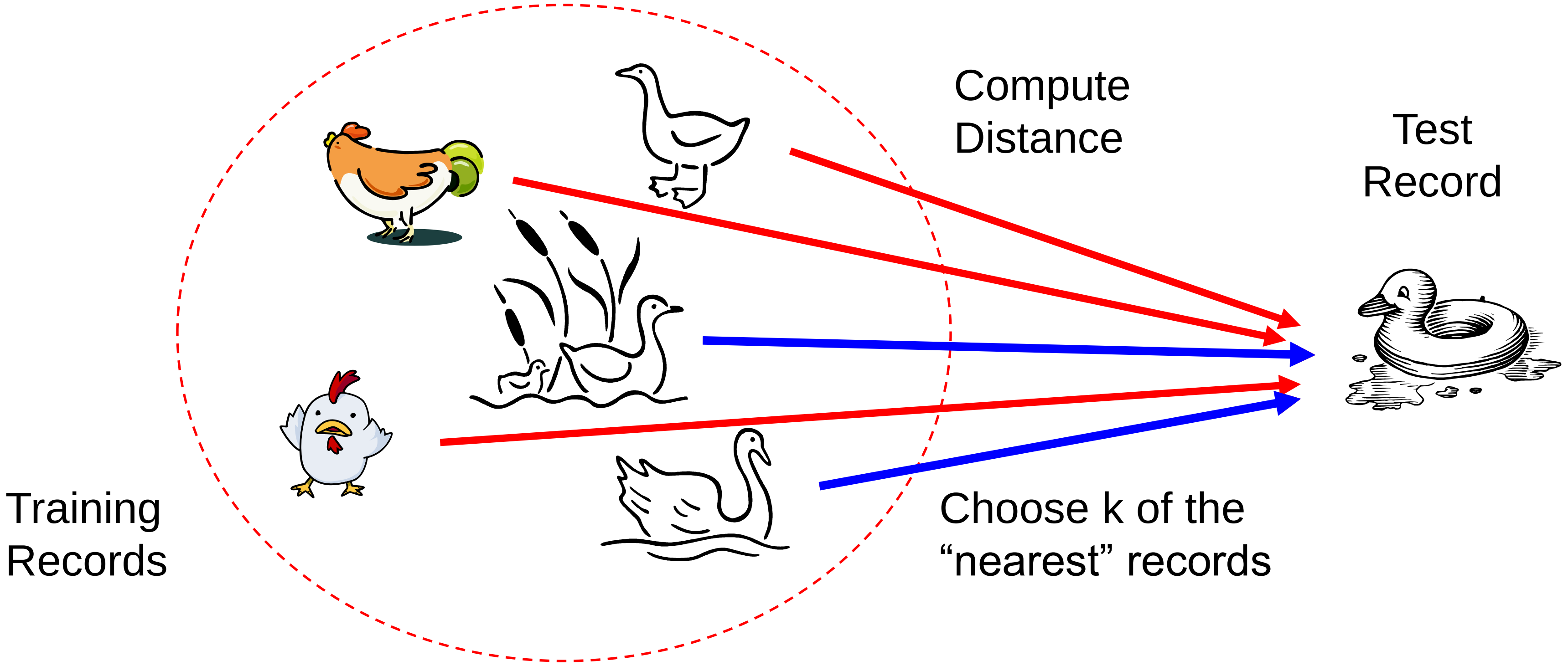
Atr1	……...	AtrN

Instance Based Classifiers

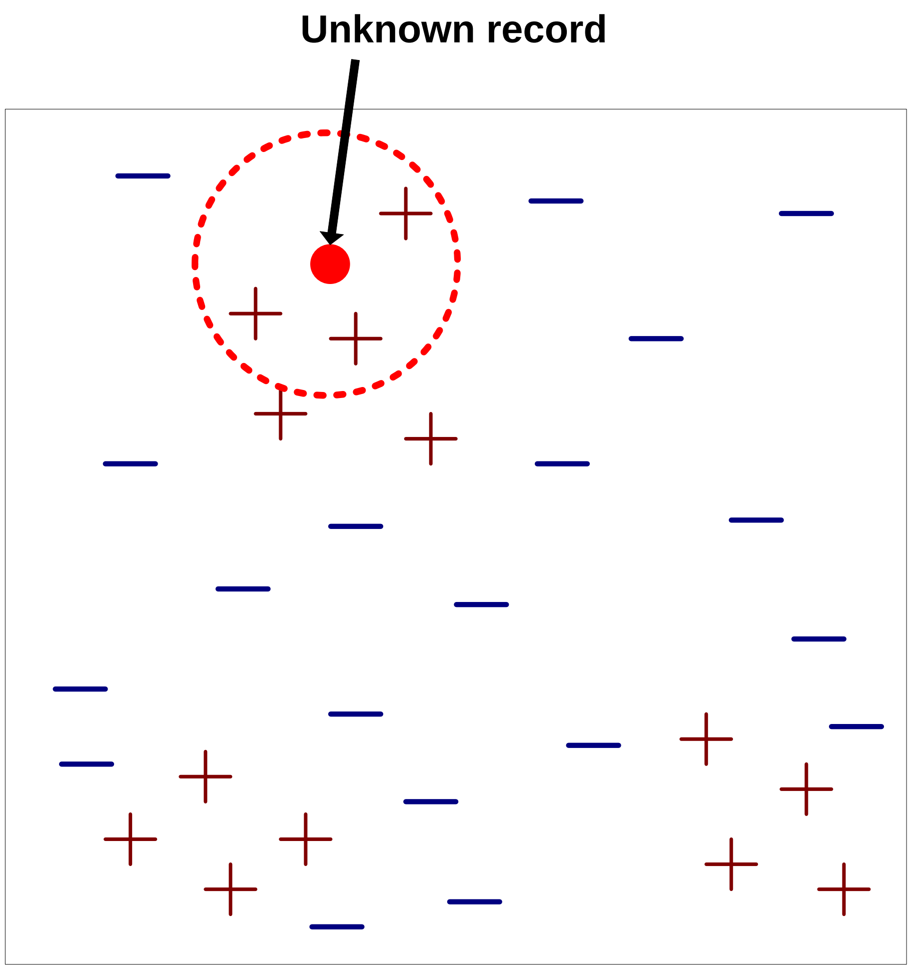
- Examples:
  - Rote-learner
    - Memorizes entire training data and performs classification only if attributes of record match one of the training examples exactly
  - Nearest neighbor classifier
    - Uses k “closest” points (nearest neighbors) for performing classification

Nearest Neighbor Classifiers

- Basic idea:
  - *"If it walks like a duck, quacks like a duck, then it's probably a duck"*



Nearest-Neighbor Classifiers



- Requires three things
  - The set of stored records
  - Distance Metric to compute distance between records
  - The value of *k*, the number of nearest neighbors to retrieve

- To classify an unknown record:
  1. Compute distance to other training records
  2. Identify *k* nearest neighbors
  3. Use class labels of nearest neighbors to determine the class label of unknown record (e.g., by taking majority vote)

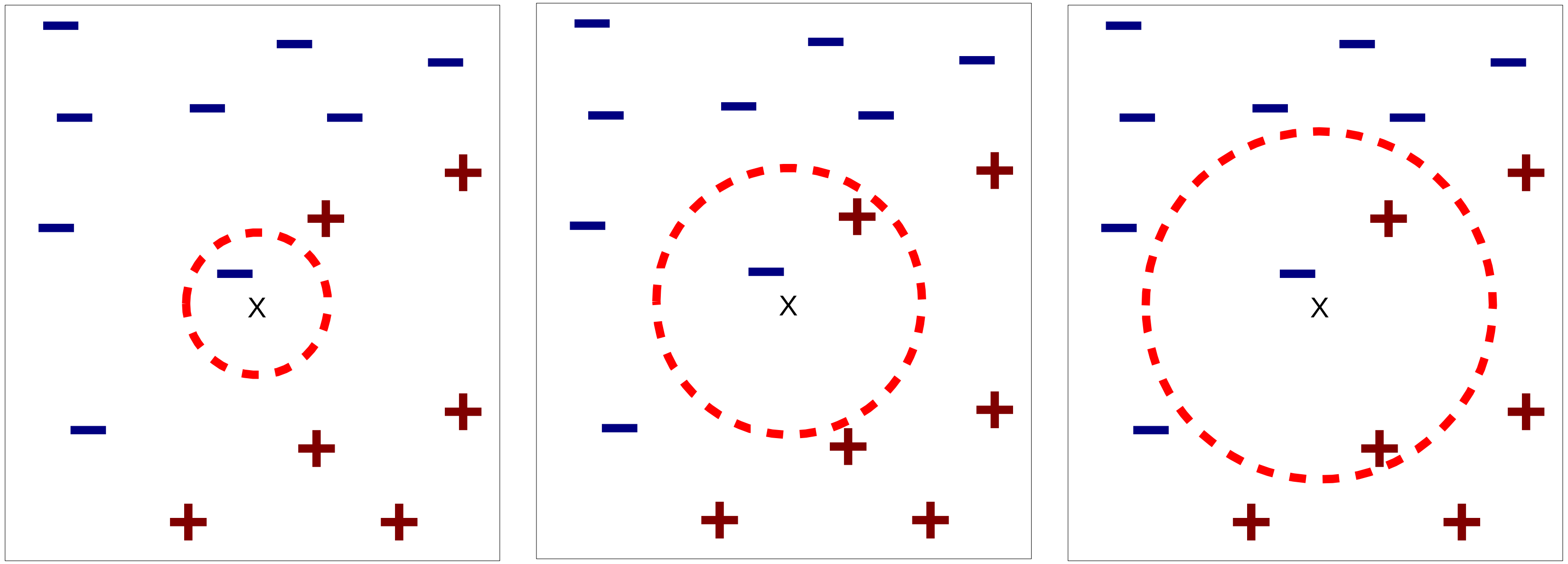
Nearest Neighbor Classification

- Compute distance between two points:
  - Euclidean distance

$$d(p,q) = \sqrt{\sum_i (p_i - q_i)^2}$$

- Determine the class from nearest neighbor list
  - take the majority vote of class labels among the k-nearest neighbors
  - Weigh the vote according to distance
    - weight factor, $w = 1/d^2$

Definition of Nearest Neighbor

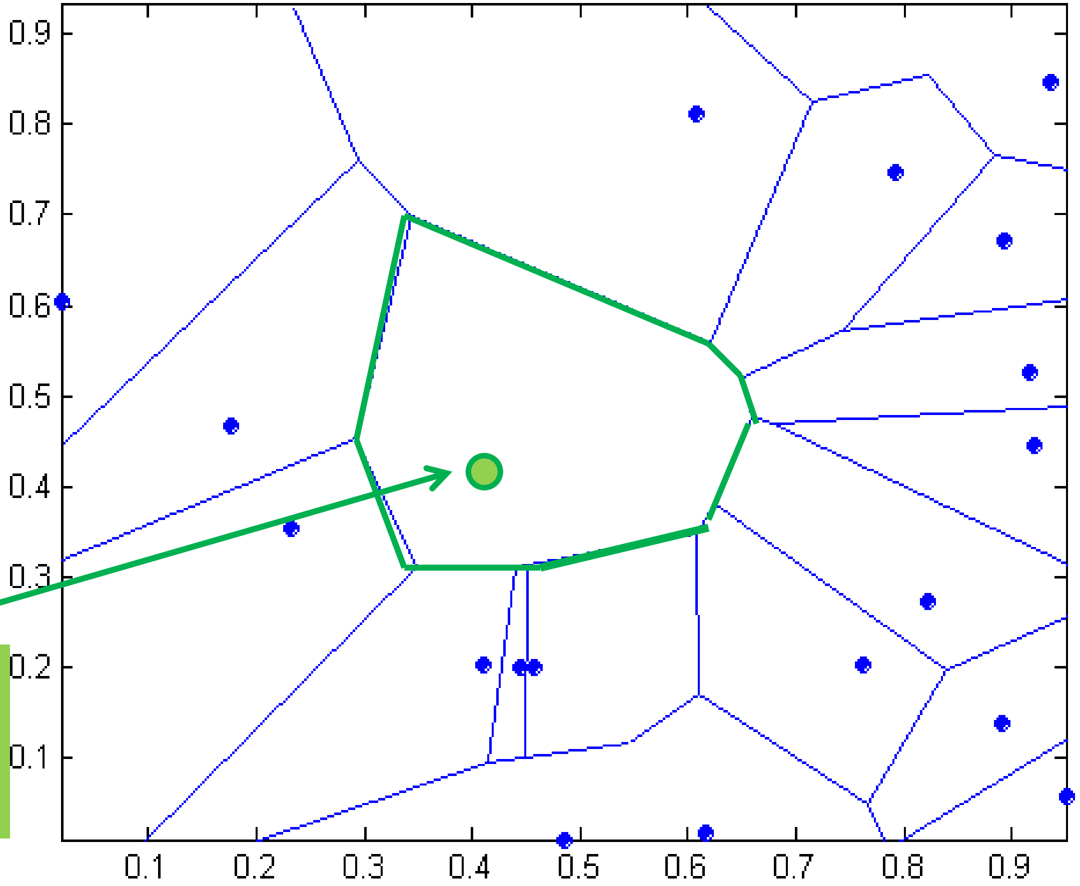


(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

K-nearest neighbors of a record x are data points that have the k smallest distance to x

1 nearest-neighbor

Voronoi Diagram defines the classification boundary



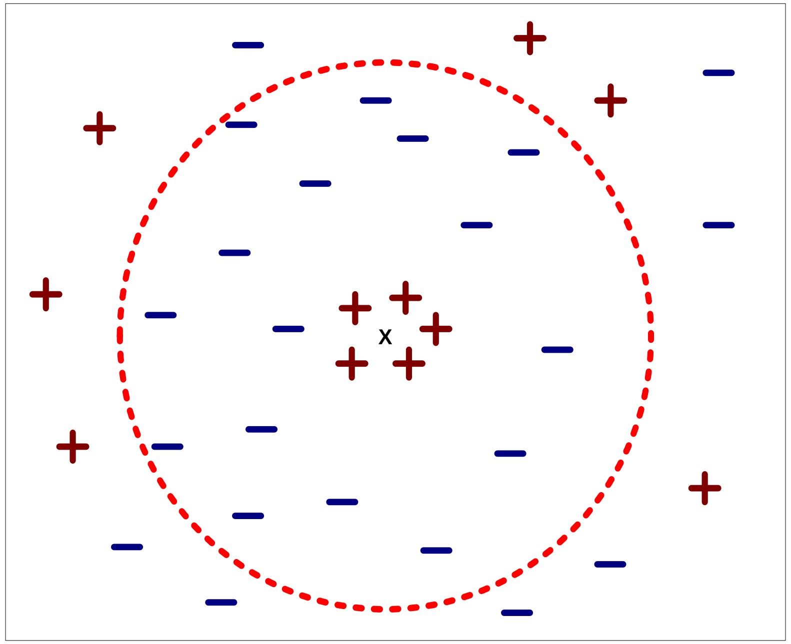
The area takes the class of the green point

Nearest Neighbor Classification…

- Choosing the value of k:
  - If k is too small, sensitive to noise points
  - If k is too large, neighborhood may include points from other classes

The value of k determines the complexity of the model

Lower k produces more complex models



Example

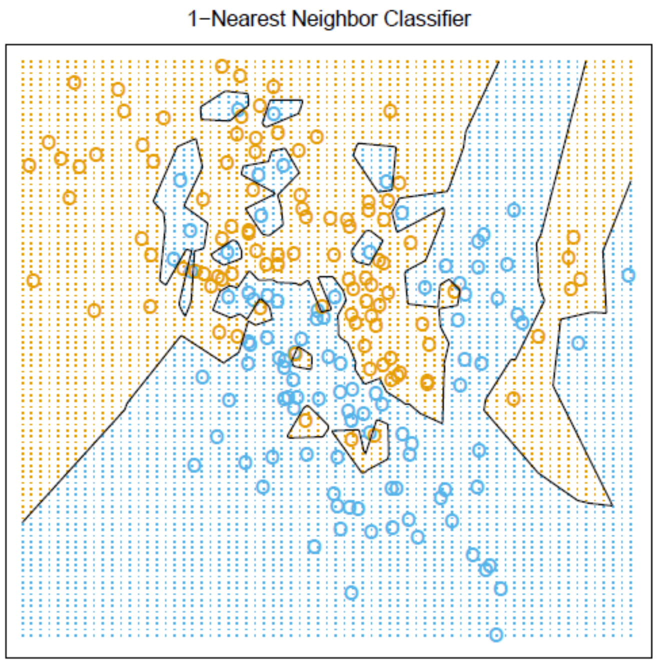


FIGURE 2.3. *The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable (BLUE = 0, ORANGE = 1), and then predicted by 1-nearest-neighbor classification.*

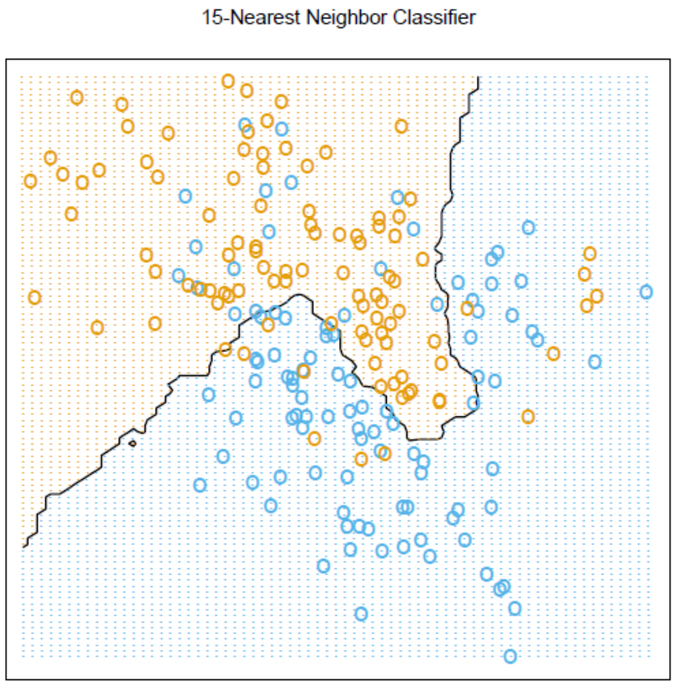


FIGURE 2.2. *The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable (BLUE = 0, ORANGE = 1) and then fit by 15-nearest-neighbor averaging as in (2.8). The predicted class is hence chosen by majority vote amongst the 15-nearest neighbors.*

Nearest Neighbor Classification…

- Problem with Euclidean measure:
  - High dimensional data
    - curse of dimensionality
  - Can produce counter-intuitive results

1 1 1 1 1 1 1 1 1 1 1 0	vs	1 0 0 0 0 0 0 0 0 0 0 0	
0 1 1 1 1 1 1 1 1 1 1 1		0 0 0 0 0 0 0 0 0 0 0 1	
d = 1.4142		d = 1.4142	

- Solution: Normalize the vectors to unit length

Nearest neighbor Classification…

- k-NN classifiers are lazy learners
  - It does not build models explicitly
  - Unlike eager learners such as decision trees
- Classifying unknown records is relatively expensive
  - Naïve algorithm: O(n)
  - Need for structures to retrieve nearest neighbors fast.
    - The Nearest Neighbor Search problem.
    - Also, Approximate Nearest Neighbor Search

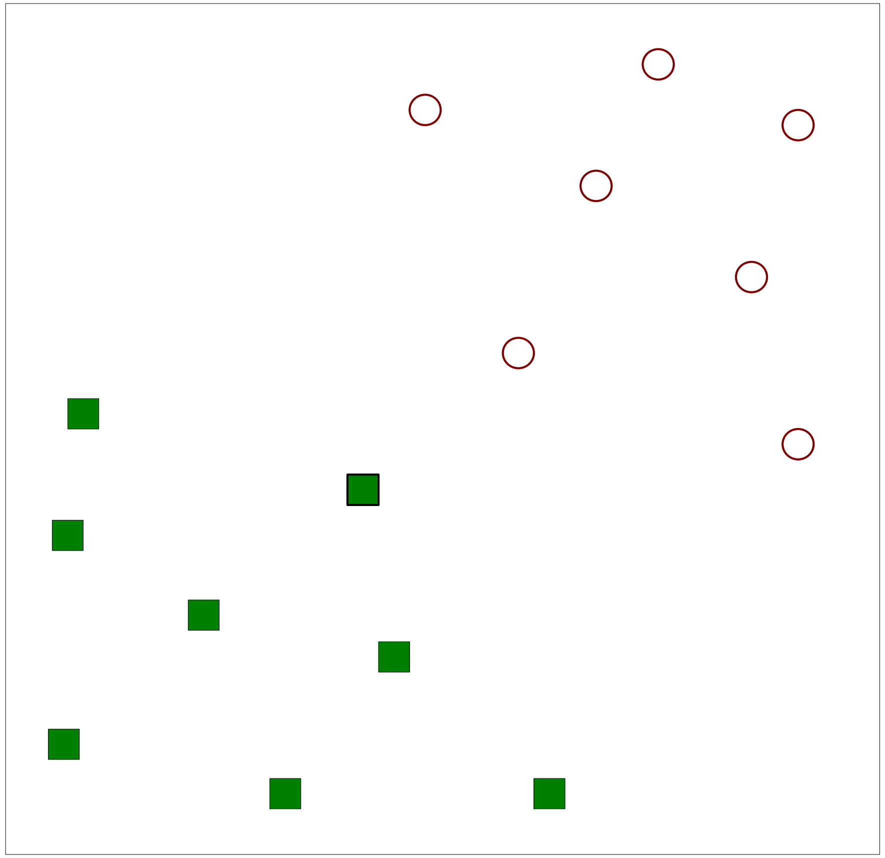
- Issues with distance in very high-dimensional spaces

SUPPORT VECTOR MACHINES

Linear classifiers

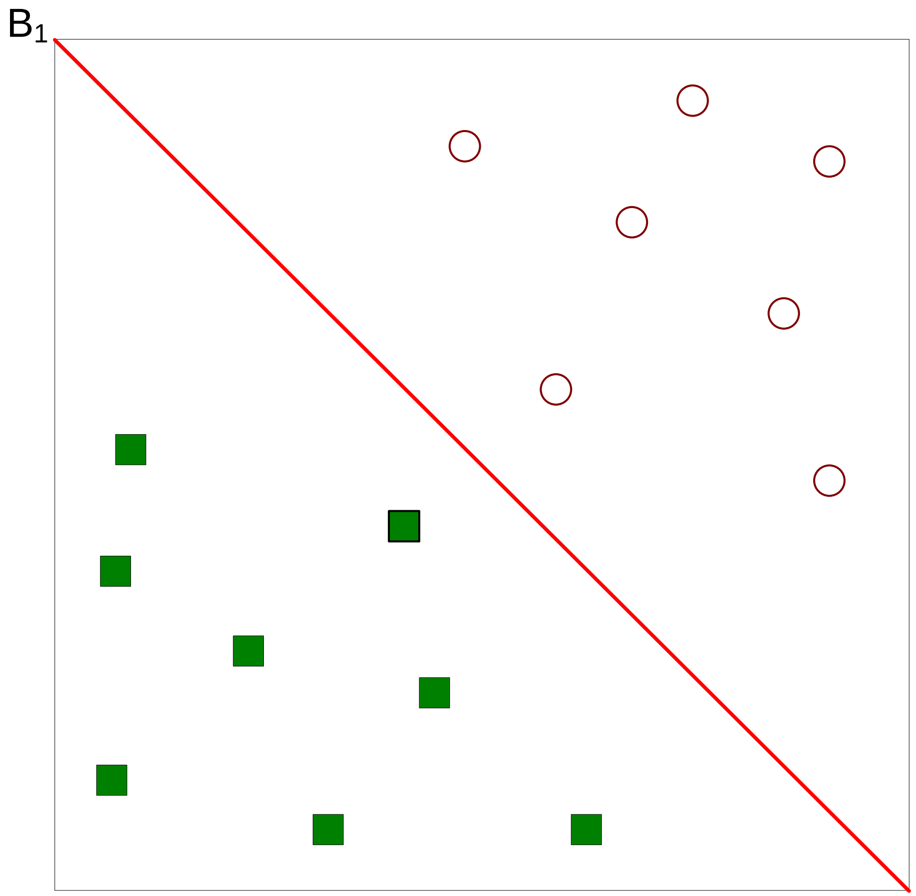
- SVMs are part of a family of classifiers that assumes that the classes are linearly separable
- That is, there is a hyperplane that separates (approximately, or exactly) the instances of the two classes.
- The goal is to find this hyperplane

Support Vector Machines



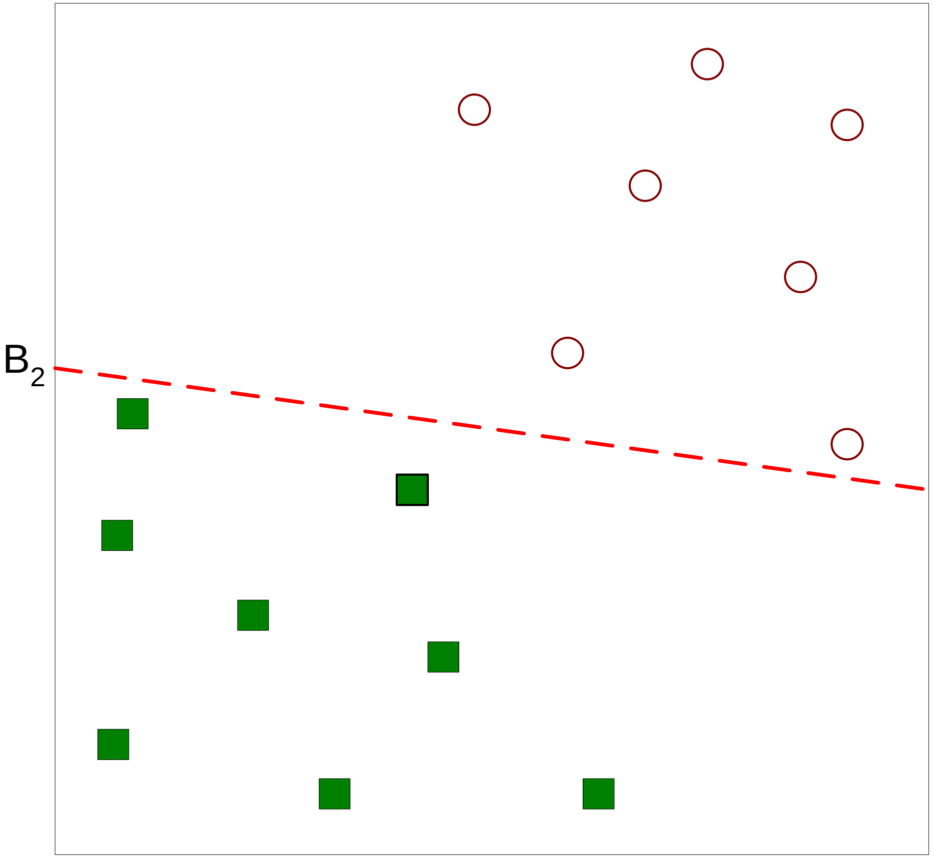
- Find a linear hyperplane (decision boundary) that will separate the data

Support Vector Machines



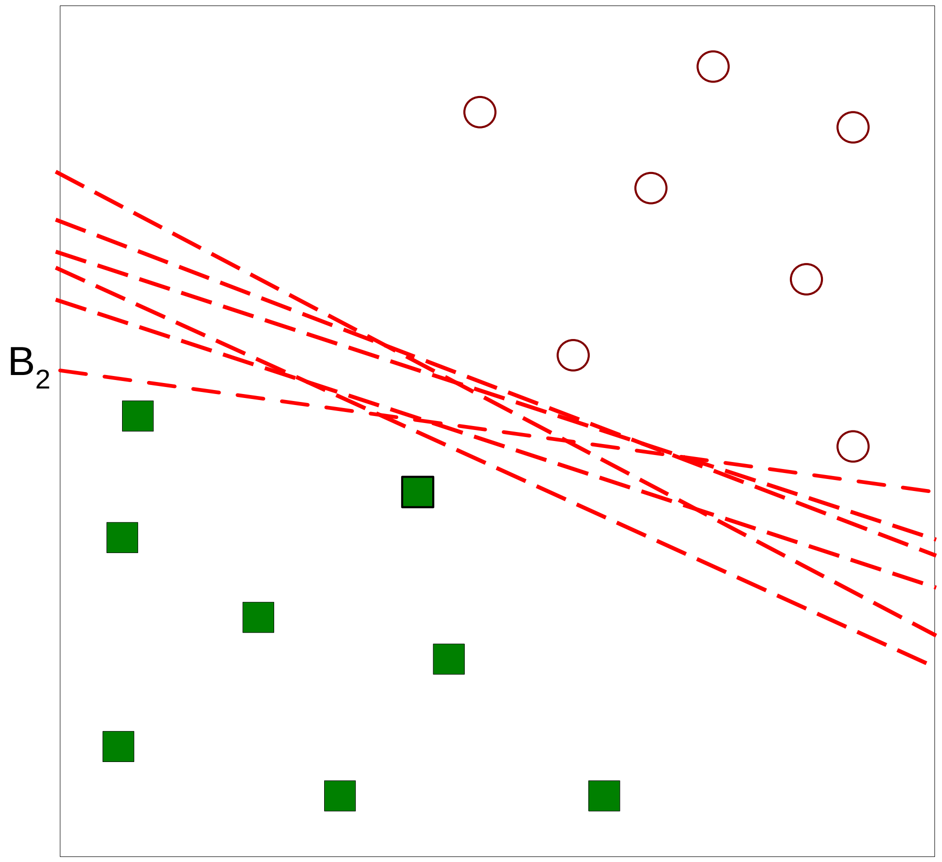
- One Possible Solution

Support Vector Machines



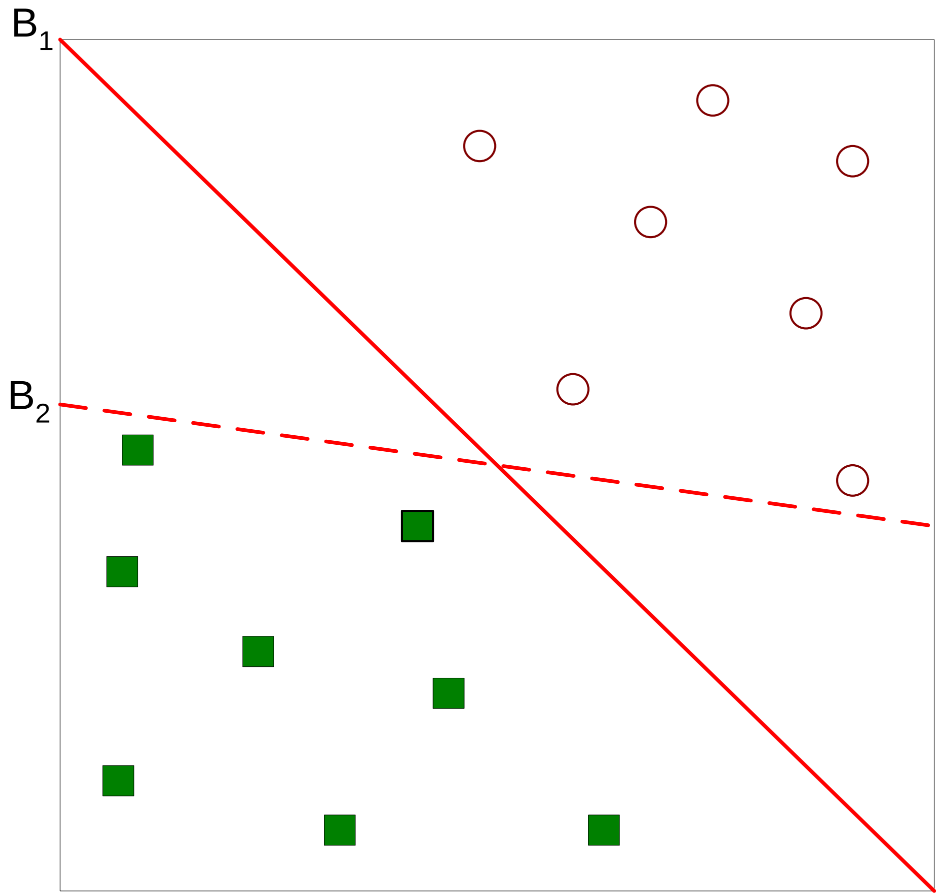
- Another possible solution

Support Vector Machines



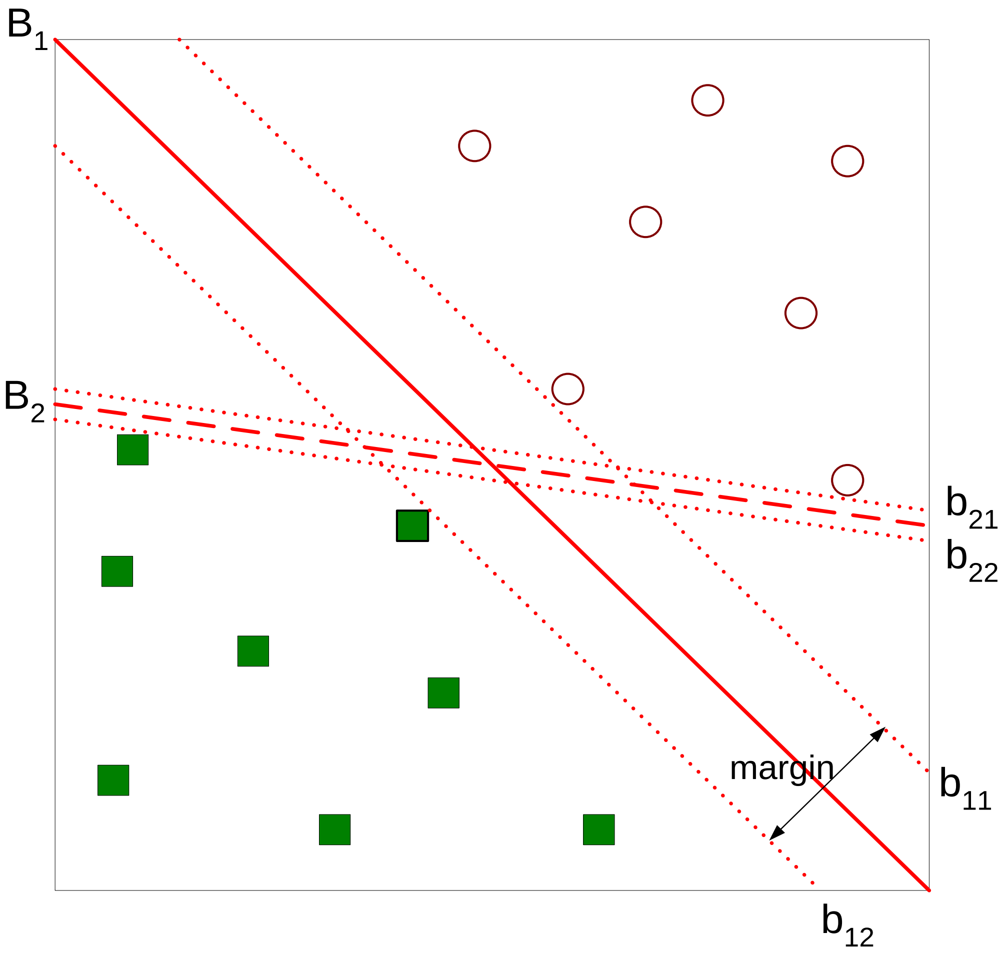
- Other possible solutions

Support Vector Machines



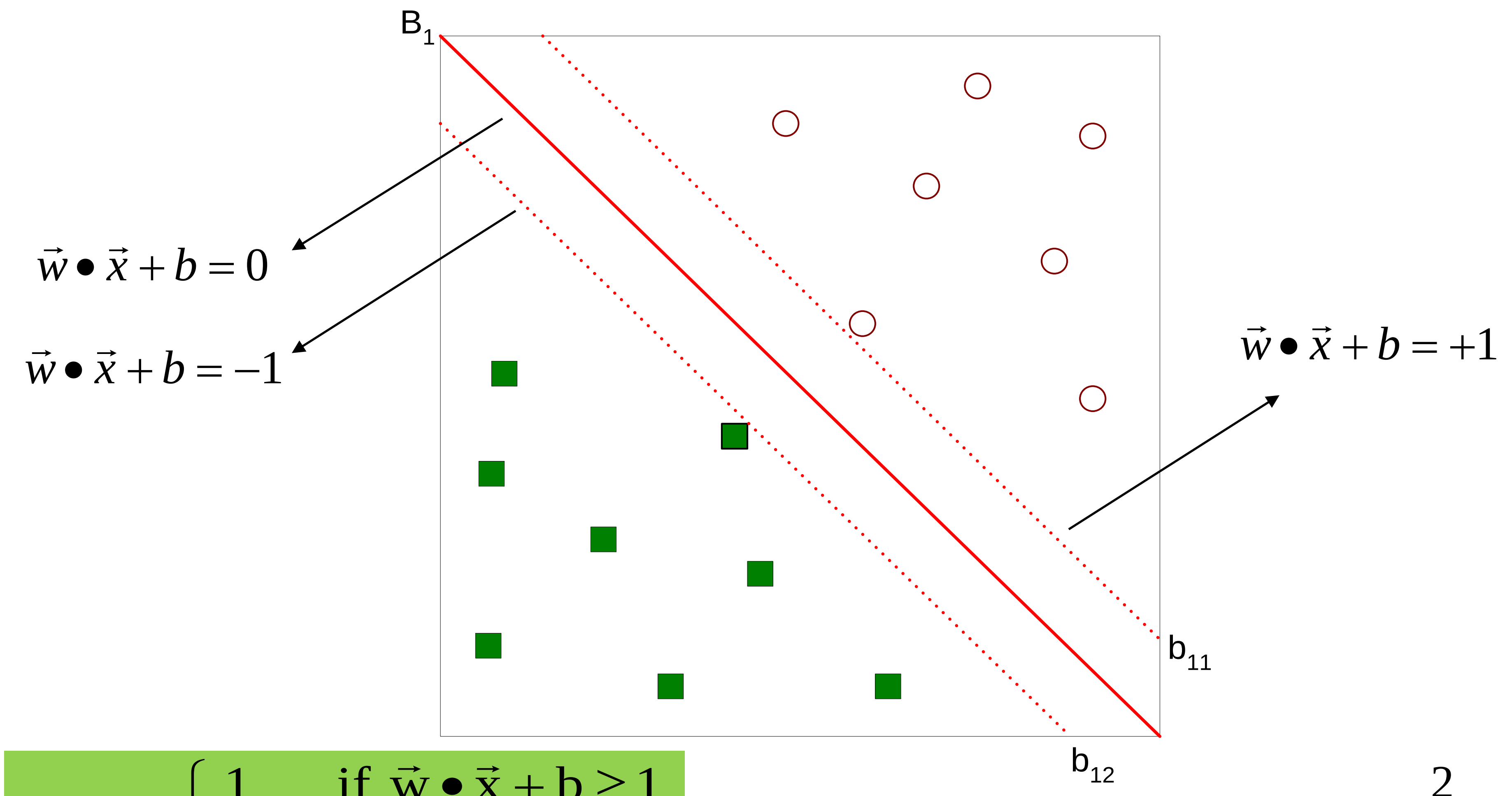
- Which one is better? B1 or B2?
- How do you define better?

Support Vector Machines



- Find hyperplane maximizes the margin : B1 is better than B2

Support Vector Machines



$$f(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \bullet \vec{x} + b \ge 1 \\ -1 & \text{if } \vec{w} \bullet \vec{x} + b \le -1 \end{cases}$$

$$\text{Margin} = \frac{2}{\|\vec{w}\|}$$

Support Vector Machines

- We want to maximize: $Margin = \frac{2}{\|\vec{w}\|}$

- Which is equivalent to minimizing: $L(\vec{w}) = \frac{\|\vec{w}\|}{2}$

- But subjected to the following constraints:

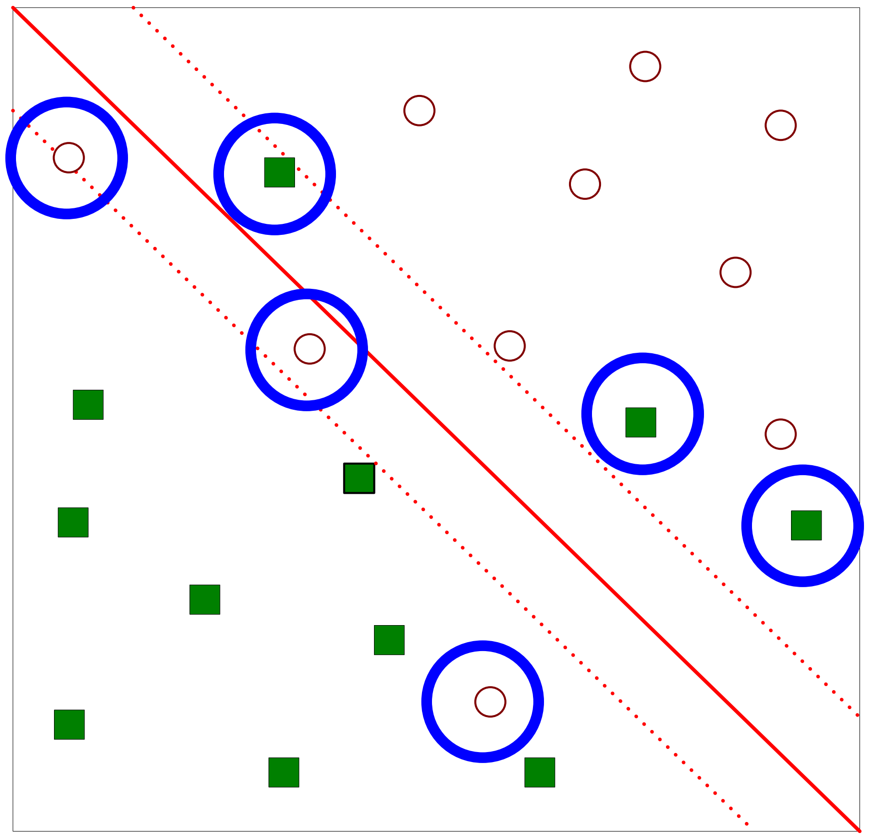
$$\vec{w} \cdot \vec{x_i} + b \geq 1 \text{ if } y_i = 1$$
$$\vec{w} \cdot \vec{x_i} + b \leq -1 \text{ if } y_i = -1$$

Concisely:
$y_i(\vec{w} \cdot \vec{x_i} + b) \geq 1$

- This is a constrained optimization problem
  - Numerical approaches to solve it (e.g., quadratic programming)

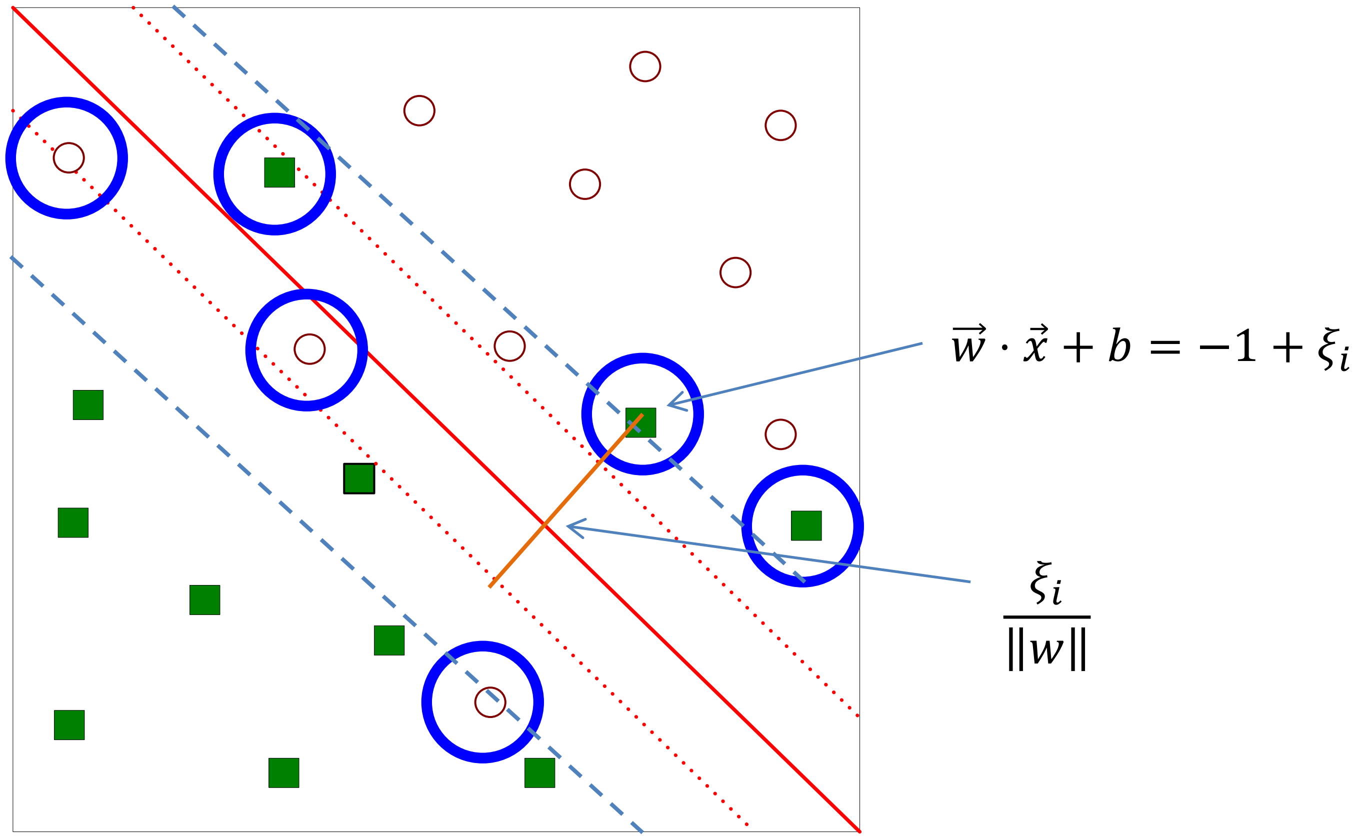
Support Vector Machines

- What if the problem is not linearly separable?



Support Vector Machines

- What if the problem is not linearly separable?



Support Vector Machines

- What if the problem is not linearly separable?
- Introduce slack variables
  - Minimize:

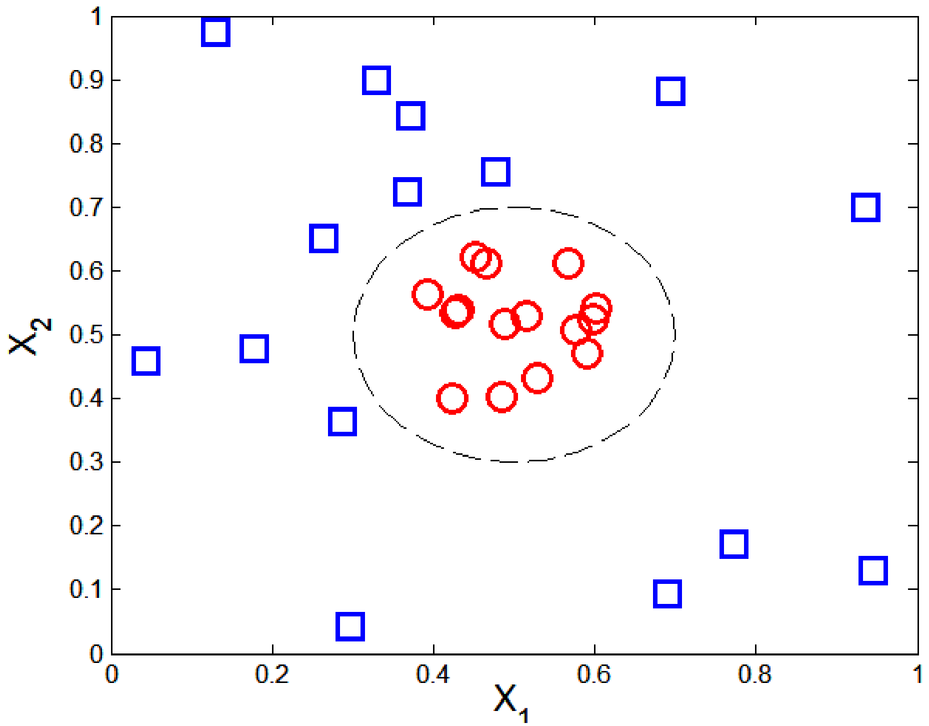
$$L(w) = \frac{\|\vec{w}\|}{2} + C\left(\sum_{i=1}^{N} \xi_i^k\right)$$

  - Subject to:

$$\vec{w} \cdot \vec{x_i} + b \geq 1 - \xi_i \text{ if } y_i = 1$$
$$\vec{w} \cdot \vec{x_i} + b \leq -1 + \xi_i \text{ if } y_i = -1$$

Nonlinear Support Vector Machines

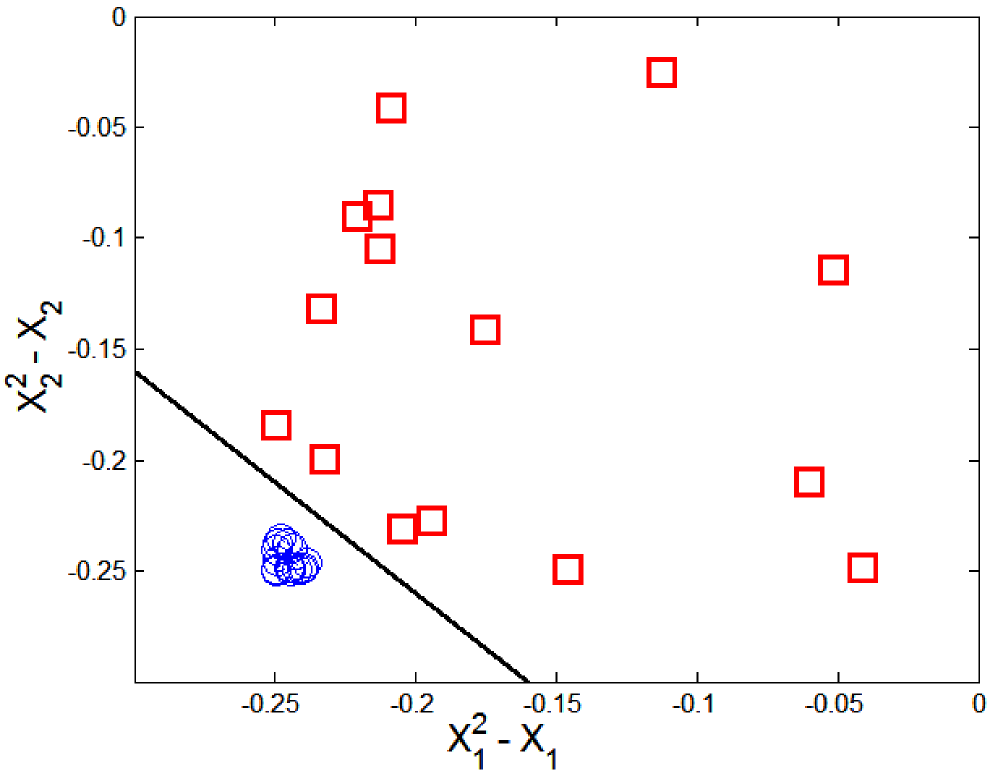
- What if decision boundary is not linear?



$$y(x_1, x_2) = \begin{cases} 1 & \text{if } \sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2} > 0.2 \\ -1 & \text{otherwise} \end{cases}$$

Nonlinear Support Vector Machines

- Trick: Transform data into higher dimensional space



$$x_1^2 - x_1 + x_2^2 - x_2 = -0.46.$$

$$\Phi : (x_1, x_2) \longrightarrow (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, 1).$$

$$w_4 x_1^2 + w_3 x_2^2 + w_2\sqrt{2}x_1 + w_1\sqrt{2}x_2 + w_0 = 0.$$

Decision boundary:

$$\vec{w} \cdot \Phi(\vec{x}) + b = 0$$

Learning Nonlinear SVM

- Optimization problem:

$$\min_{w} \frac{\|\mathbf{w}\|^2}{2}$$

$$subject\ to \qquad y_i(w \cdot \Phi(x_i) + b) \geq 1,\ \ \forall\{(x_i, y_i)\}$$

- Which leads to the same set of equations (but involve $\Phi(x)$ instead of x)

$$L_D = \sum_{i=1}^{n} \lambda_i - \frac{1}{2}\sum_{i,j} \lambda_i \lambda_j y_i y_j \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$

$$\mathbf{w} = \sum_i \lambda_i y_i \Phi(\mathbf{x}_i)$$

$$\lambda_i\{y_i(\sum_j \lambda_j y_j \Phi(\mathbf{x}_j) \cdot \Phi(\mathbf{x}_i) + b) - 1\} = 0,$$

$$f(\mathbf{z}) = sign(\mathbf{w} \cdot \Phi(\mathbf{z}) + b) = sign(\sum_{i=1}^{n} \lambda_i y_i \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{z}) + b).$$

Learning NonLinear SVM

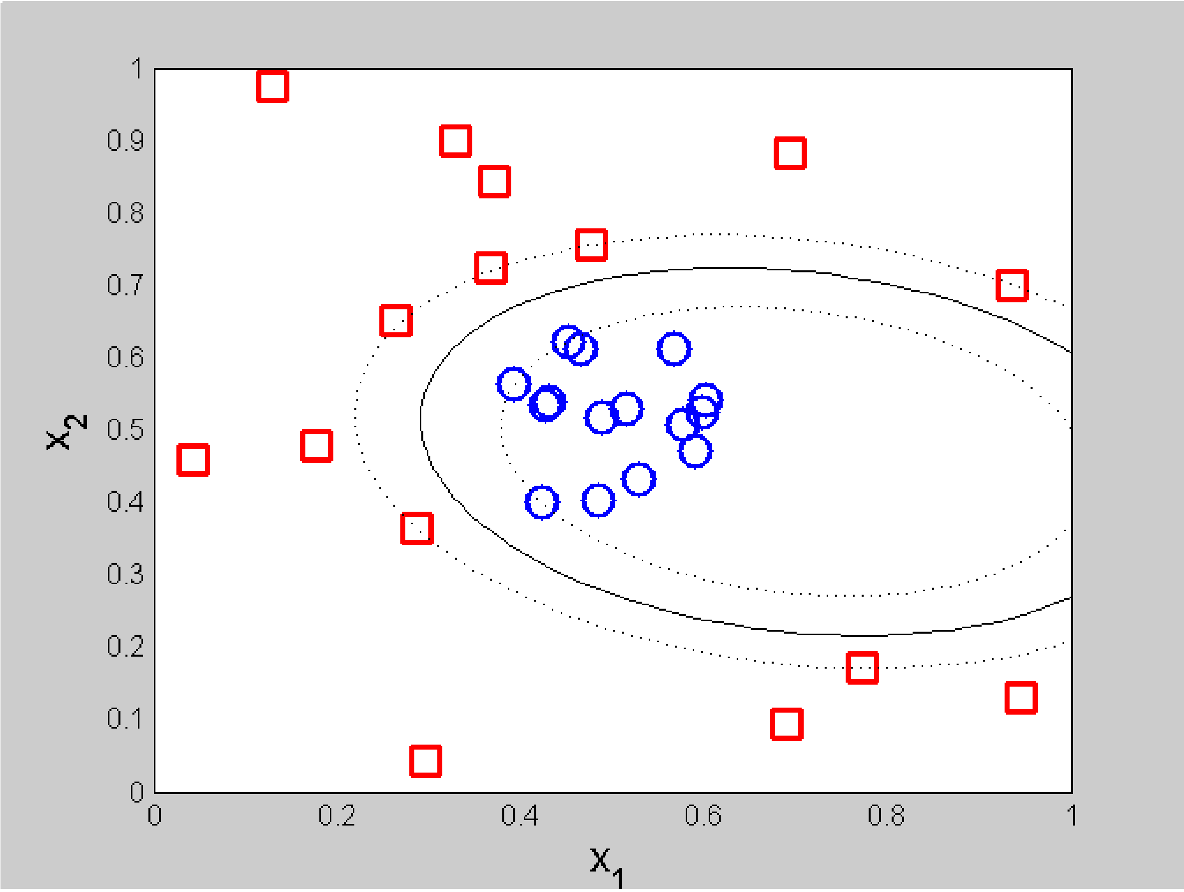
- Issues:
  - What type of mapping function Φ should be used?
  - How to do the computation in high dimensional space?
    - Most computations involve dot product $\Phi(x_i) \cdot \Phi(x_j)$
    - Curse of dimensionality?

Learning Nonlinear SVM

- Kernel Trick:
  - $\Phi(x_i) \cdot \Phi(x_j) = K(x_i, x_j)$
  - $K(x_i, x_j)$ is a kernel function (expressed in terms of the coordinates in the original space)
    - Examples:

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + 1)^p$$
$$K(\mathbf{x}, \mathbf{y}) = e^{-\|\mathbf{x}-\mathbf{y}\|^2/(2\sigma^2)}$$
$$K(\mathbf{x}, \mathbf{y}) = \tanh(k\mathbf{x} \cdot \mathbf{y} - \delta)$$

Example of Nonlinear SVM



SVM with polynomial degree 2 kernel

$$K(x_i, x_j) = (x_i \cdot x_j + 1)^2$$

Learning Nonlinear SVM

- Advantages of using kernel:
  - Don't have to know the mapping function Φ
  - Computing dot product $\Phi(x_i) \cdot \Phi(x_j)$ in the original space avoids curse of dimensionality
- Not all functions can be kernels
  - Must make sure there is a corresponding Φ in some high-dimensional space
  - Mercer's theorem (see textbook)

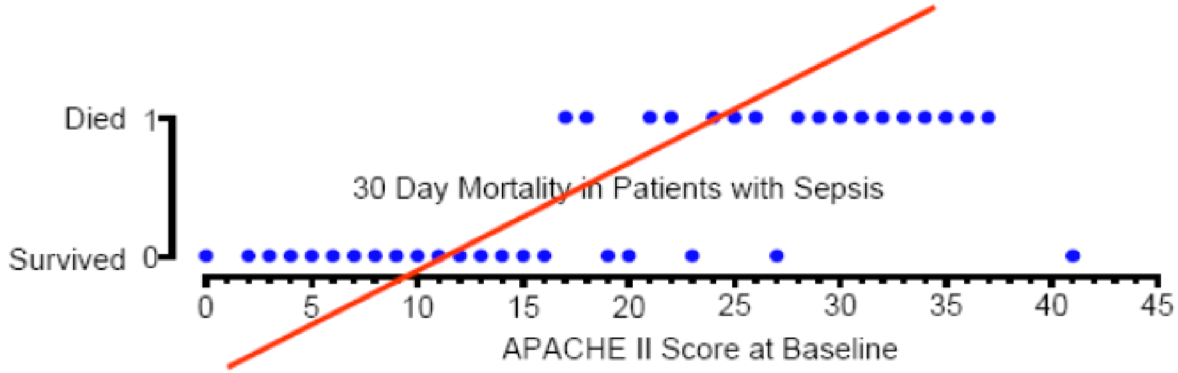
LOGISTIC REGRESSION

Classification via regression

- Instead of predicting the class of a record we want to predict the probability of the class given the record
- Transform the classification problem into a regression problem.
- But how do you define the probability that you want to predict?

Linear regression

- A simple approach: use linear regression to learn a linear function that predicts 0/1 values
  - Not good: It may produce negative probabilities, or probabilities that are greater than 1.
  - Also the probabilities it produces are not what we want. We want probability close to zero for small values, and close to 1 for large, and a transition from 0 to 1 around the value 20

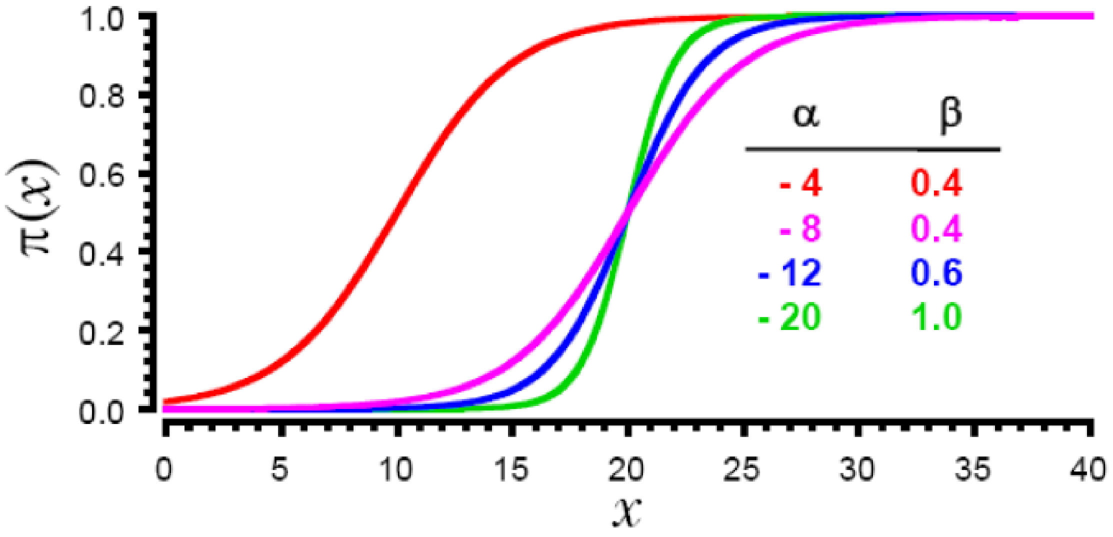


The logistic function

$$f(x) = \frac{1}{1 + e^{-\alpha-\beta x}}$$

β controls the slope
α controls the position of the turning point

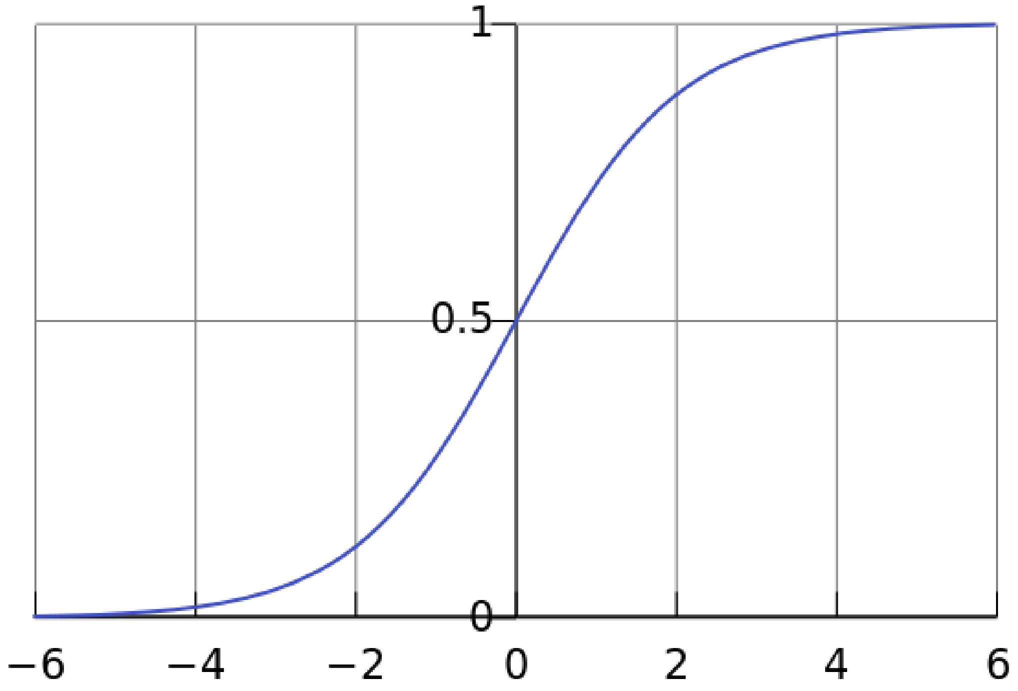
$$\pi(x) = \exp(\alpha + \beta x) / (1 + \exp(\alpha + \beta x))$$



When $x = -\alpha/\beta$, $\alpha + \beta x = 0$ and hence $\pi(x) = 1/(1+1) = 0.5$

Logistic Regression

$$f(x) = \frac{1}{1+e^{-x}}$$



Class Probabilities

$$P(C_+|x) = \frac{1}{1+e^{-\beta x - a}}$$

$$P(C_-|x) = \frac{e^{-\beta x - a}}{1+e^{-\beta x - a}}$$

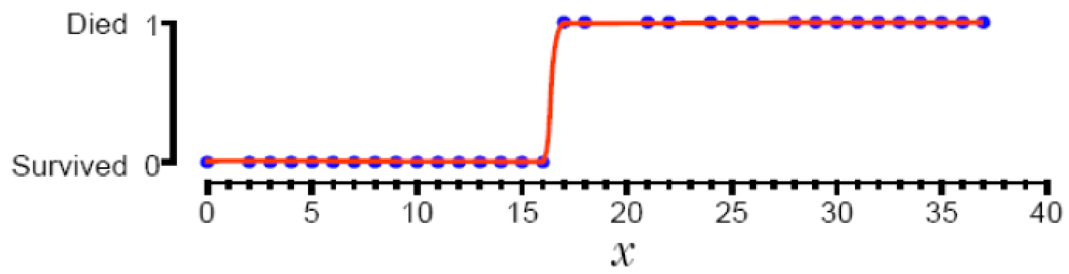
Logistic Regression: Find the values β, α that maximize the probability of the observed data

$$\log \frac{P(C_+|x)}{P(C_-|x)} = \beta x + a$$

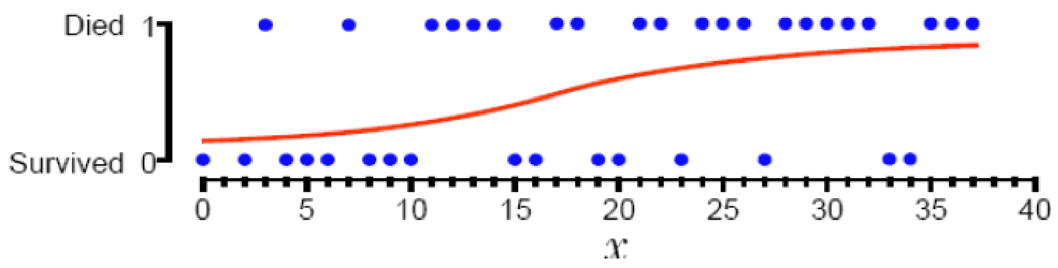
Linear regression on the log-odds ratio

Logistic Regression in one dimension

Data that has a sharp survival cut off point between patients who live or die should have a large value of β.



Data with a lengthy transition from survival to death should have a low value of β.



Logistic Regression in one dimension

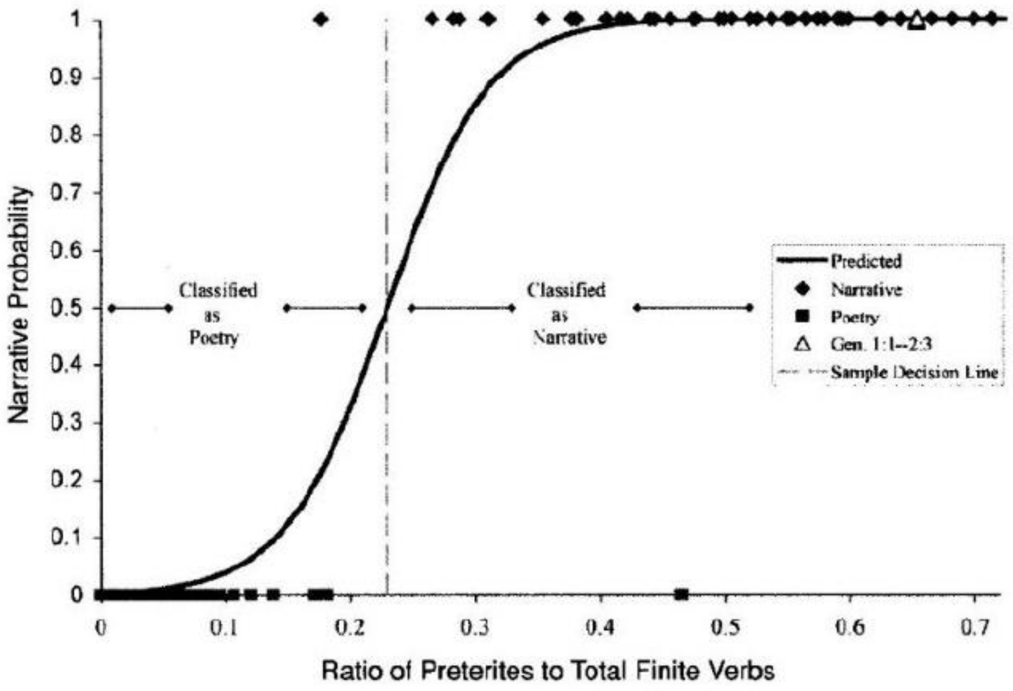


Figure 10-3. The solid curved line is called a logistic regression curve. The vertical axis measures the probability that an Old Testament passage is narrative, based on the use of preterite verbs. The probability is zero for poetry and unity or one for narrative. Passages with high preterite verb counts, falling to the right of the vertical dotted line, are likely narrative. The triangle on the upper right represents Genesis 1:1–2:3, which is clearly literal, narrative history.

Class probabilities for multiple dimensions

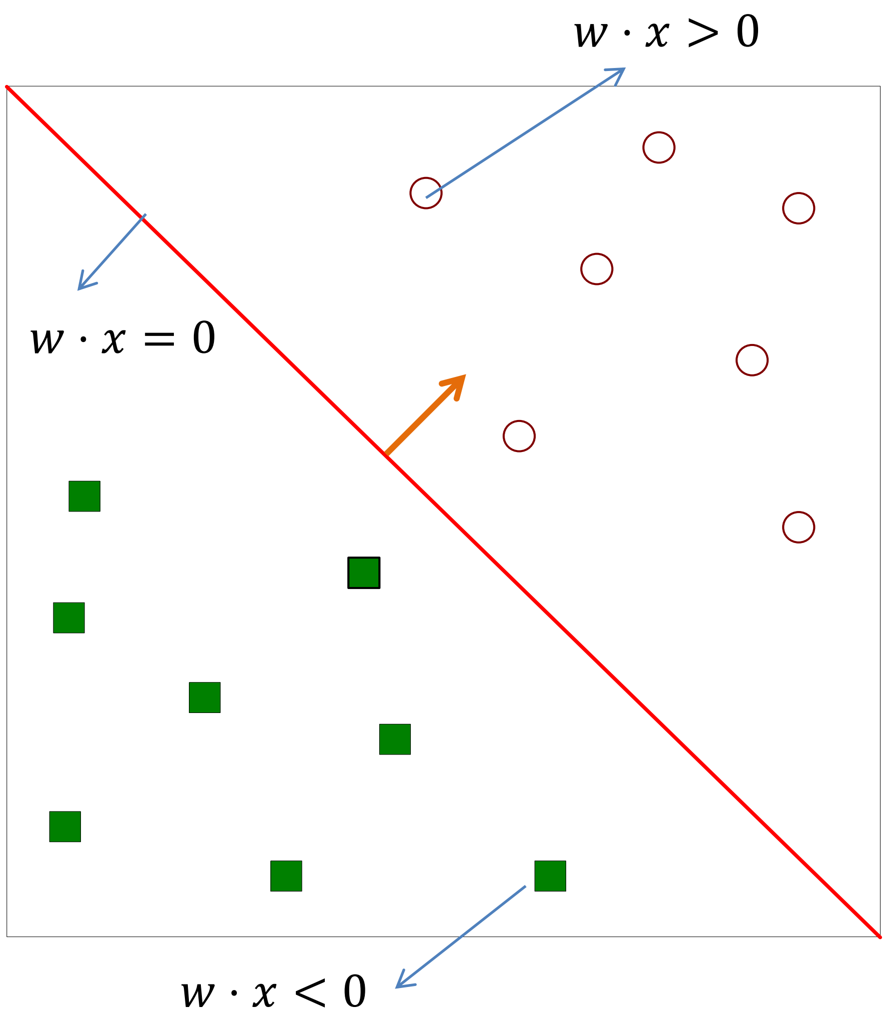
- Assume a linear classification boundary

For the positive class the bigger the value of $w \cdot x$, the further the point is from the classification boundary, the higher our certainty for the membership to the positive class

- Define $P(C_+|x)$ as an increasing function of $w \cdot x$

For the negative class the smaller the value of $w \cdot x$, the further the point is from the classification boundary, the higher our certainty for the membership to the negative class

- Define $P(C_-|x)$ as a decreasing function of $w \cdot x$



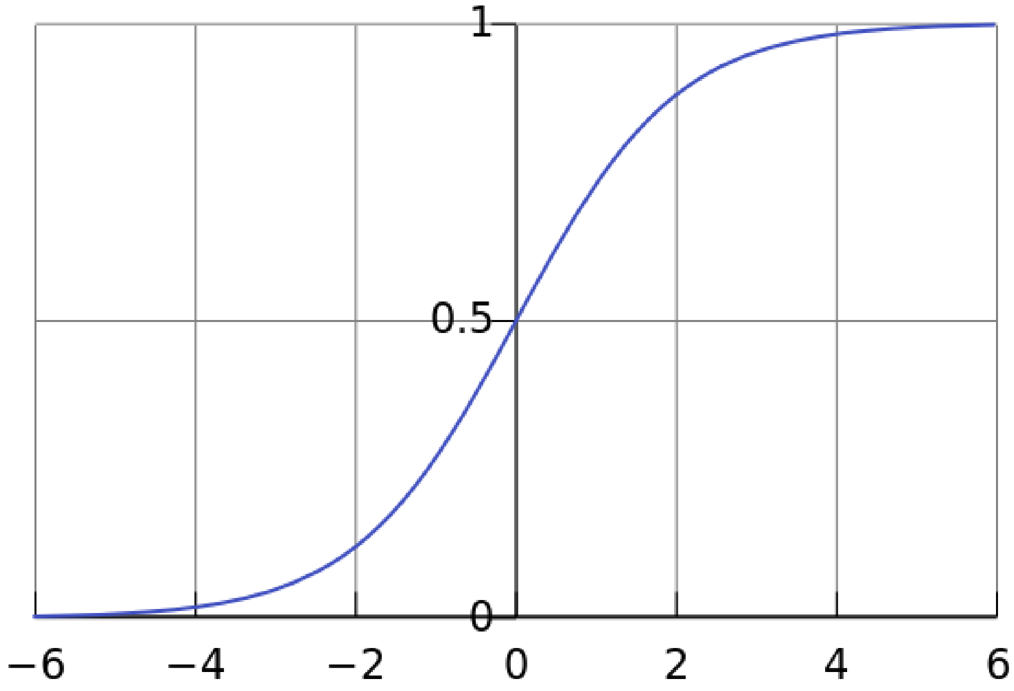
Logistic Regression

$$f(t) = \frac{1}{1+e^{-t}}$$

Class probabilities

$$P(C_+|x) = \frac{1}{1+e^{-w\cdot x-a}}$$

$$P(C_-|x) = \frac{e^{-w\cdot x-a}}{1+e^{-w\cdot x-a}}$$



Logistic Regression: Find the vector w, a that maximizes the probability of the observed data

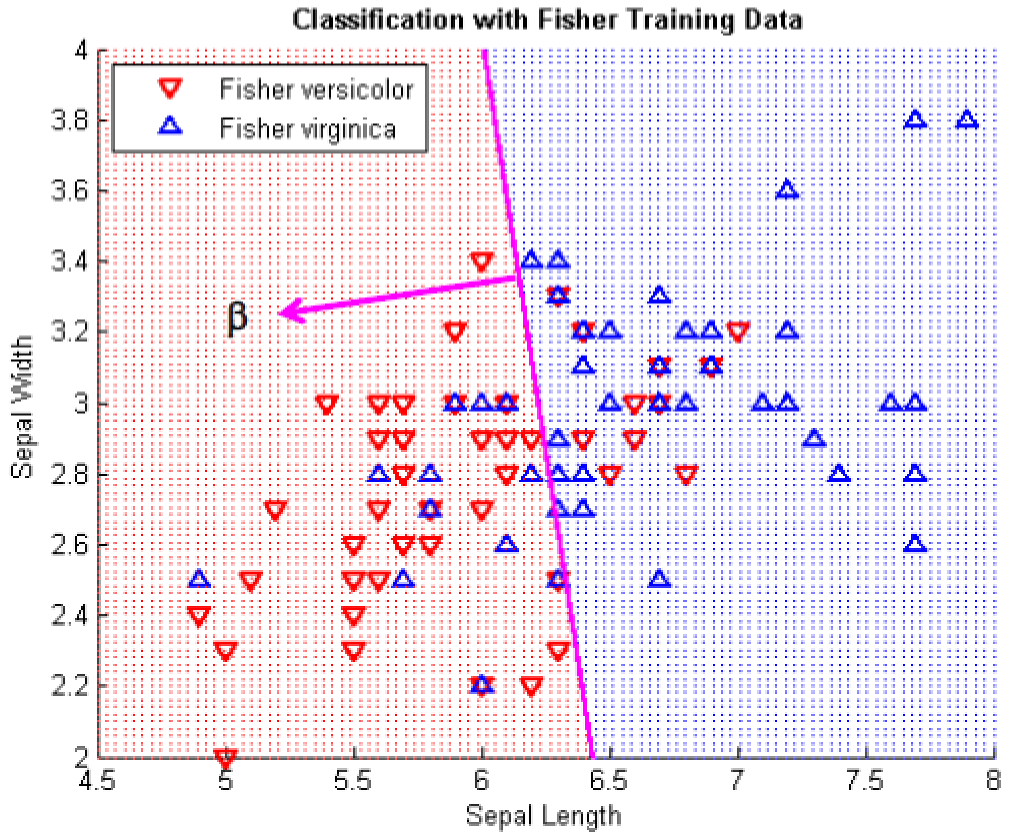
$$\log \frac{P(C_+|x)}{P(C_-|x)} = w \cdot x + a$$

Linear regression on the log-odds ratio

Logistic regression in 2-d

Coefficients

$$\beta_1 = -1.9$$
$$\beta_2 = -0.4$$
$$\alpha = 13.04$$



Estimating the coefficients

- Maximum Likelihood Estimation:
  - We have pairs of the form (x_i, y_i)
- Log Likelihood function

$$L(w) = \sum_i [y_i \log P(y_i | x_i, w) + (1 - y_i) \log(1 - P(y_i | x_i, w))]$$

- Unfortunately, it does not have a closed form solution
  - Use gradient descend to find local minimum

Logistic Regression

- Produces a probability estimate for the class membership which is often very useful.
- The weights can be useful for understanding the feature importance.
- Works for relatively large datasets
- Fast to apply.

NEURAL NETWORKS

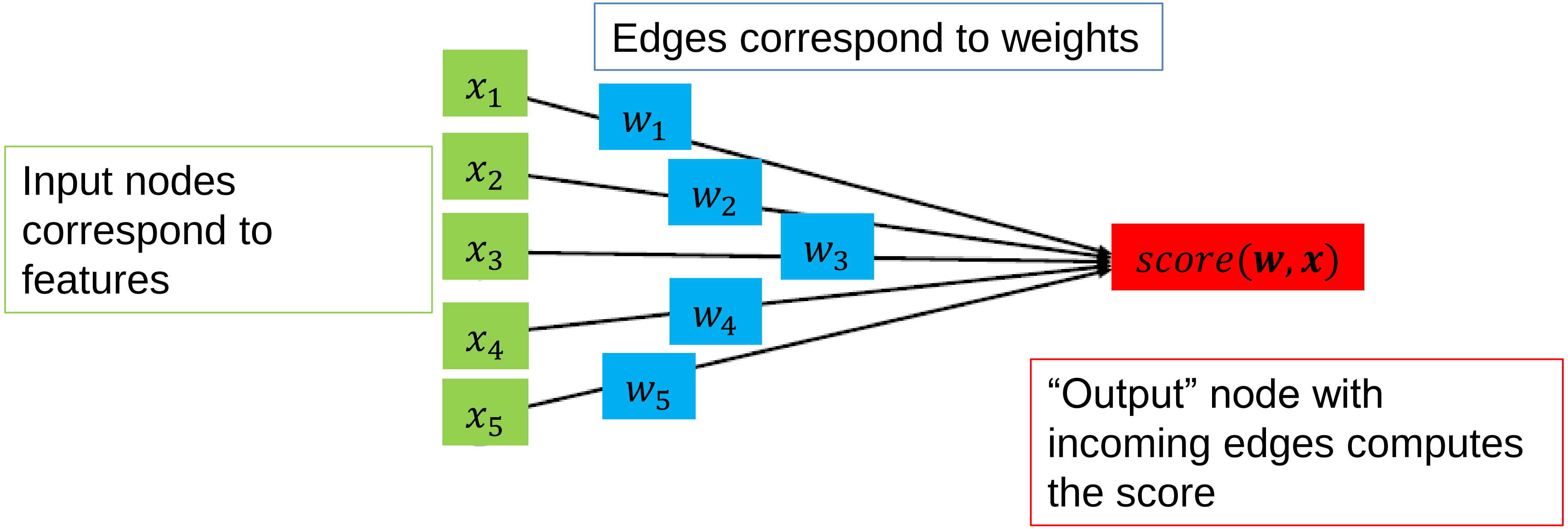
(Thanks to Philipp Koehn for the material borrowed from his slides)

Linear Classification

- A simple model for classification is to take a linear combination of the feature values and compute a score.
- Input: Feature vector $\boldsymbol{x} = (x_1, \dots, x_n)$
- Model: Weights $\boldsymbol{w} = (w_1, \dots, w_n)$
- Output: $score(\boldsymbol{w}, \boldsymbol{x}) = \sum_i w_i x_i$
- Make a decision depending on the output score.
  - E.g.: Decide “Yes” if $score(\boldsymbol{w}, \boldsymbol{x}) > 0$ and “No” if $score(\boldsymbol{w}, \boldsymbol{x}) < 0$
- The perceptron classification algorithm

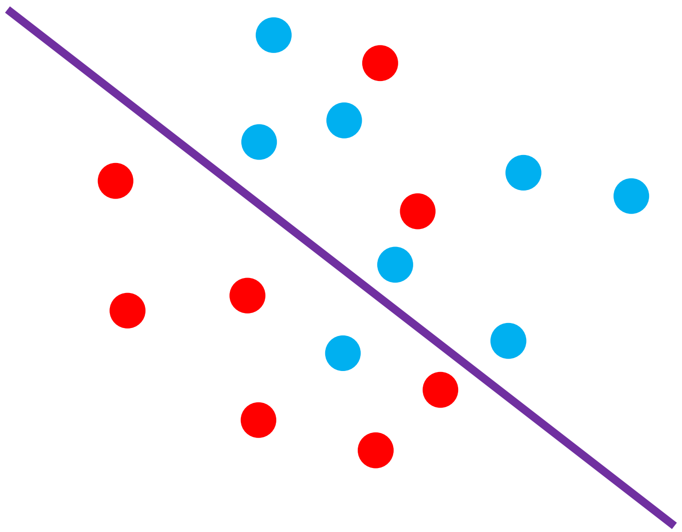
Linear Classification

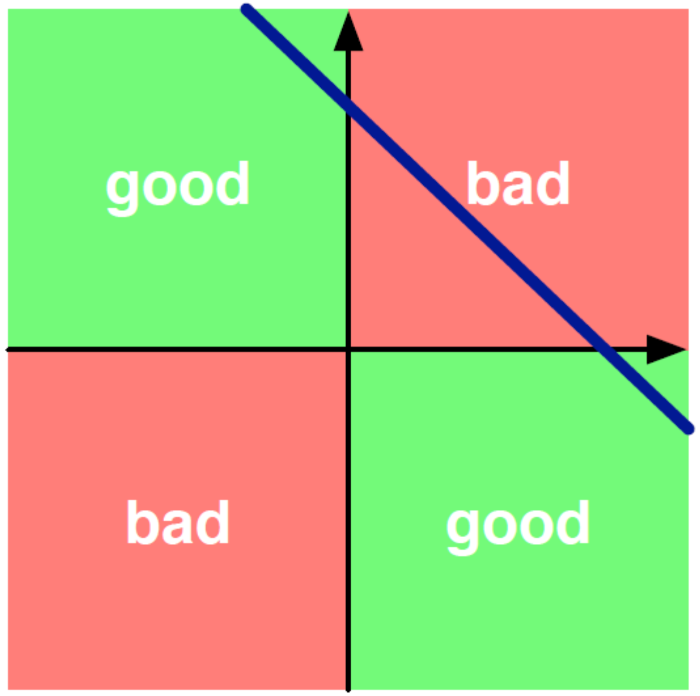
- We can represent this as a network



Linear models

- Linear models partition the space according to a hyperplane

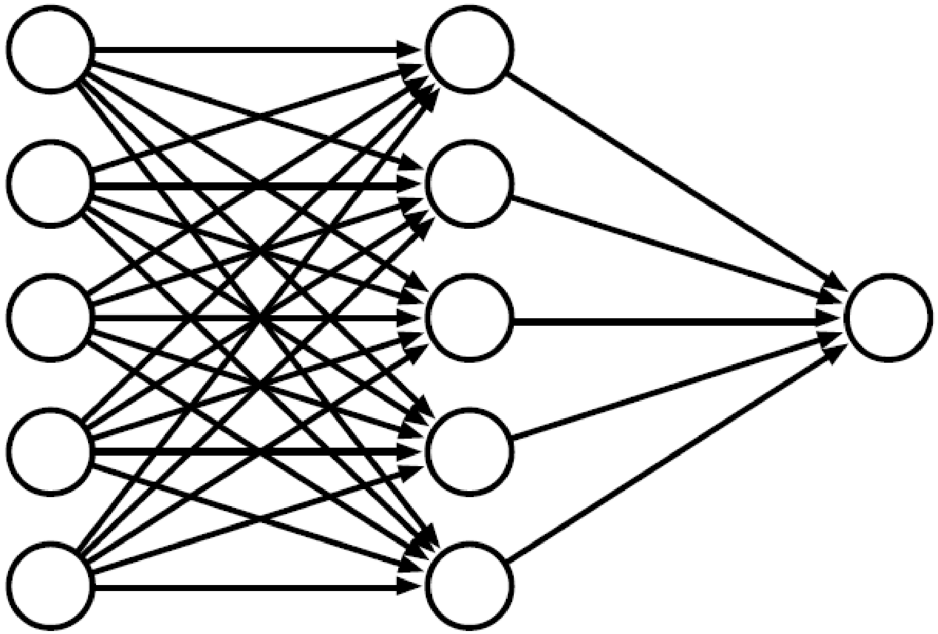




- But they cannot model everything

Multiple layers

- We can add more layers:
  - Each arrow has a weight
  - Nodes compute scores from incoming edges and give input to outgoing edges



Did we gain anything?

Non-linearity

- Instead of computing a linear combination

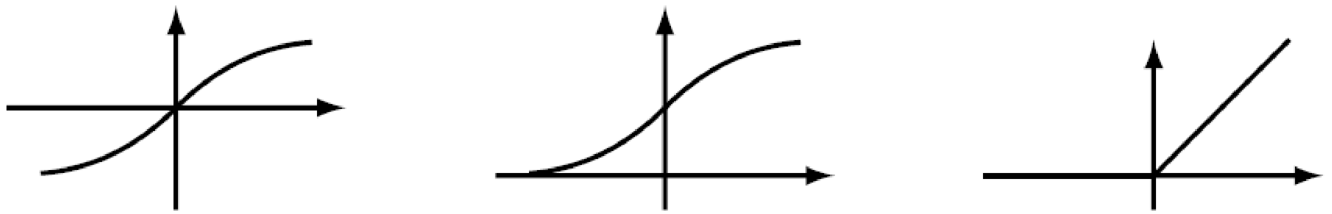
$$score(\boldsymbol{w}, \boldsymbol{x}) = \sum_i w_i x_i$$

- Apply a non-linear function on top:

$$score(\boldsymbol{w}, \boldsymbol{x}) = g\left(\sum_i w_i x_i\right)$$

- Popular functions:

tanh(x) sigmoid(x) = $\frac{1}{1+e^{-x}}$ relu(x) = max(0,x)

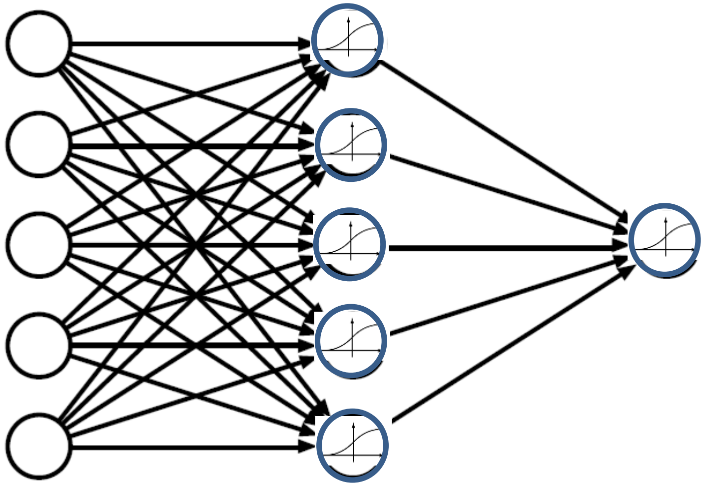


(sigmoid is also called the "logistic function")

These functions play the role of a soft "switch" (threshold function)

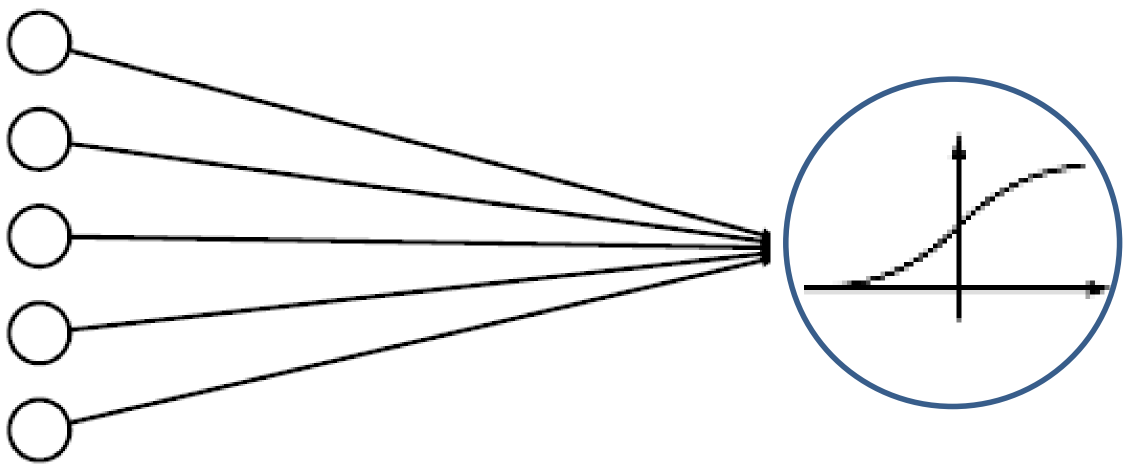
Multiple layers

- Each layer applies a logistic function to the input linear combination:
  - The result is a more complex function



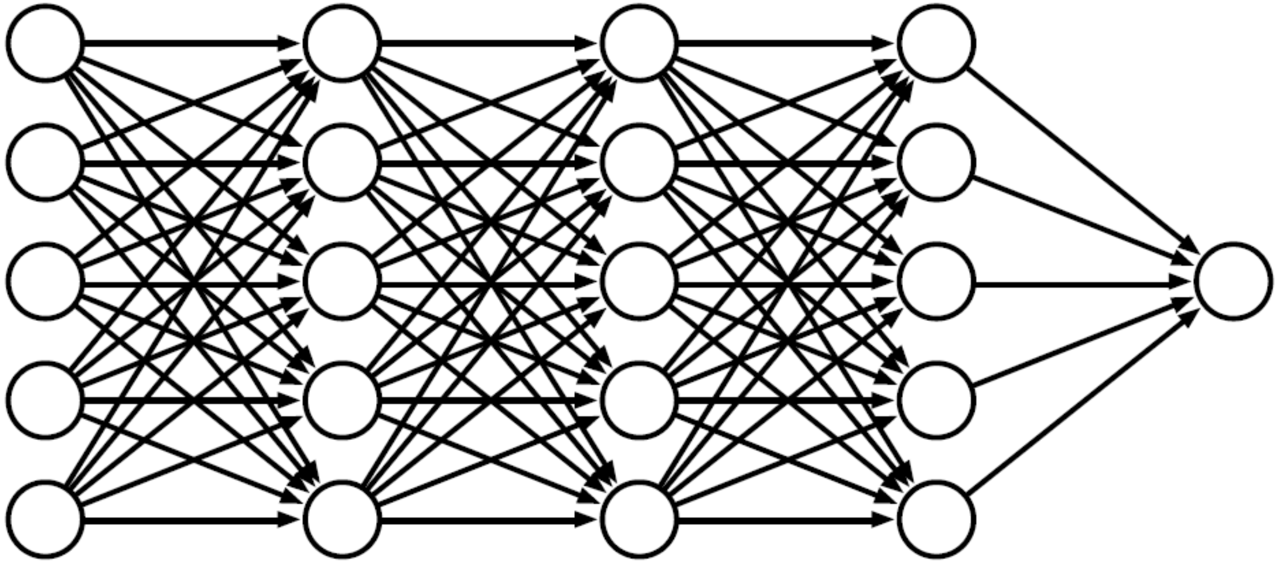
Side note

- Logistic regression classifier:
  - Single layer with a logistic function



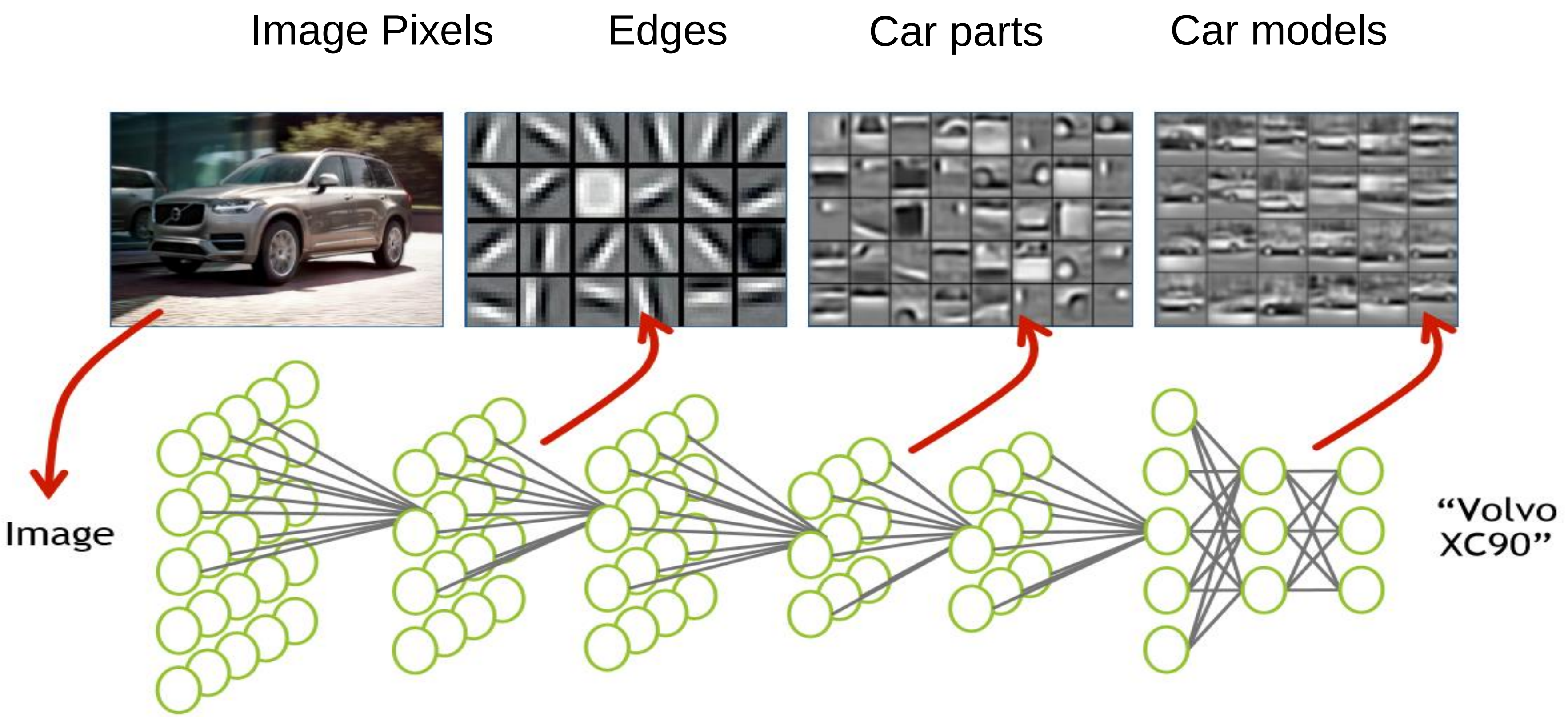
Deep learning

- Networks with multiple layers



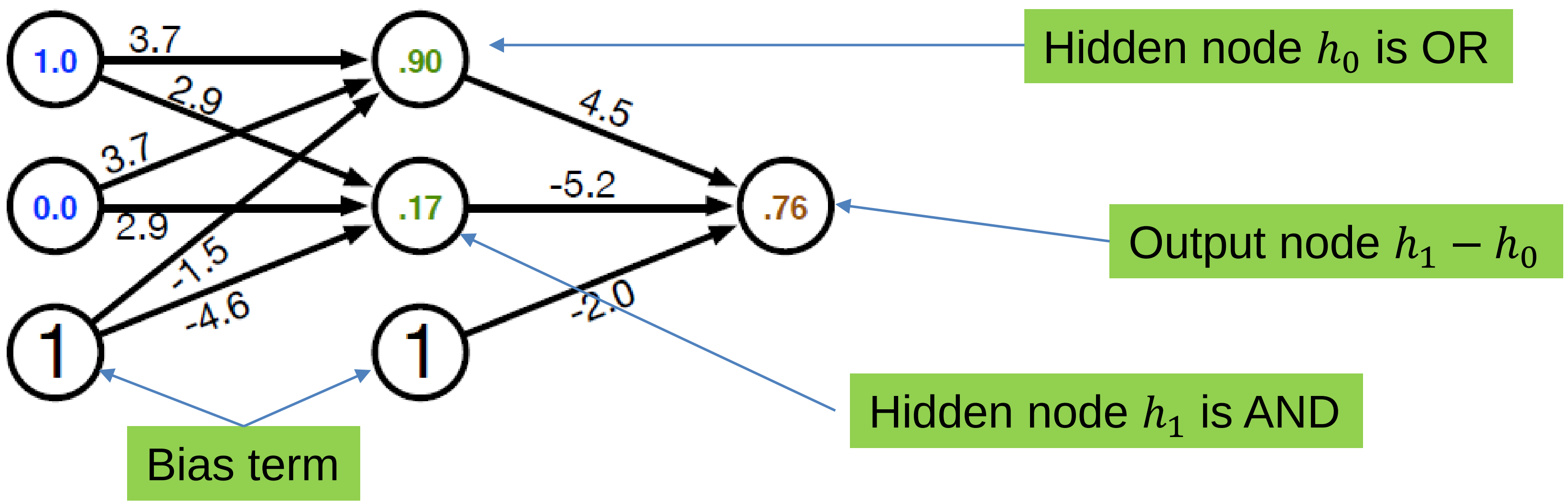
- Each layer can be thought of as a processing step
- Multiple layers allow for the computation of more complex functions

Deep Learning



Example

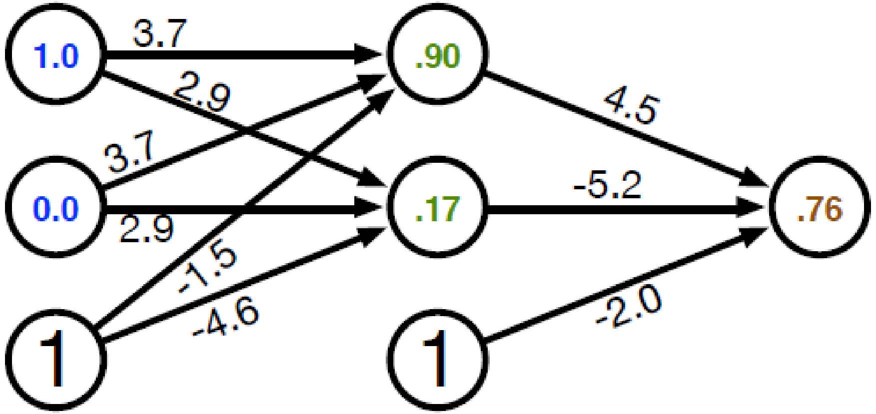
- A network that implements XOR



Input x_0	Input x_1	Hidden h_0	Hidden h_1	Output y_0
0	0	0.12	0.02	$0.18 \to 0$
0	1	0.88	0.27	$0.74 \to 1$
1	0	0.73	0.12	$0.74 \to 1$
1	1	0.99	0.73	$0.33 \to 0$

Error

- The computed value is 0.76 but the correct value is 1
  - There is an error in the computation



  - How do we set the weights so as to minimize this error?

Gradient Descent

- The error is a function of the weights:

$$E = f(\boldsymbol{w}) = f(w_1, w_2, \dots, w_N)$$

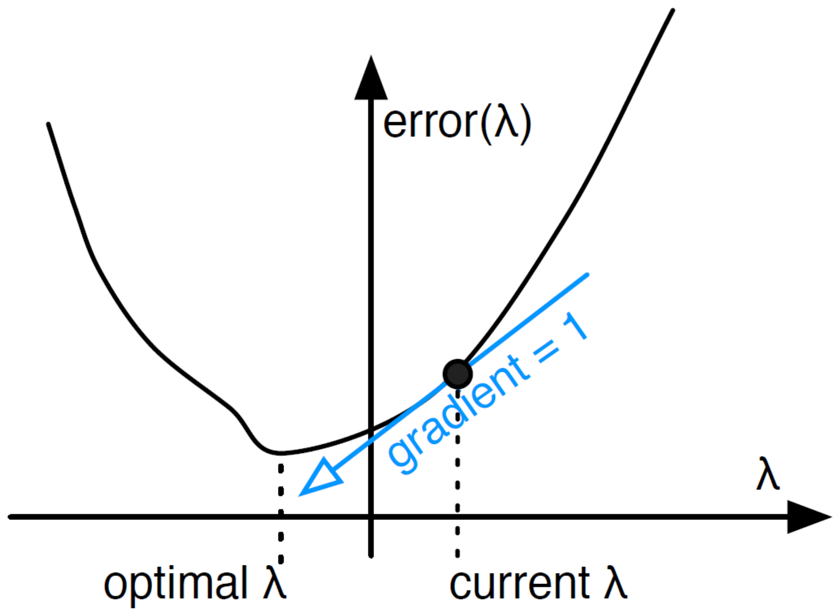
- We want to find the weights that minimize the error
- Compute gradient: gives the direction to the minimum

$$\boldsymbol{\partial f} = \left(\frac{\partial f(w_1)}{\partial w_1}, \dots, \frac{\partial f(w_N)}{\partial w_N}\right)$$

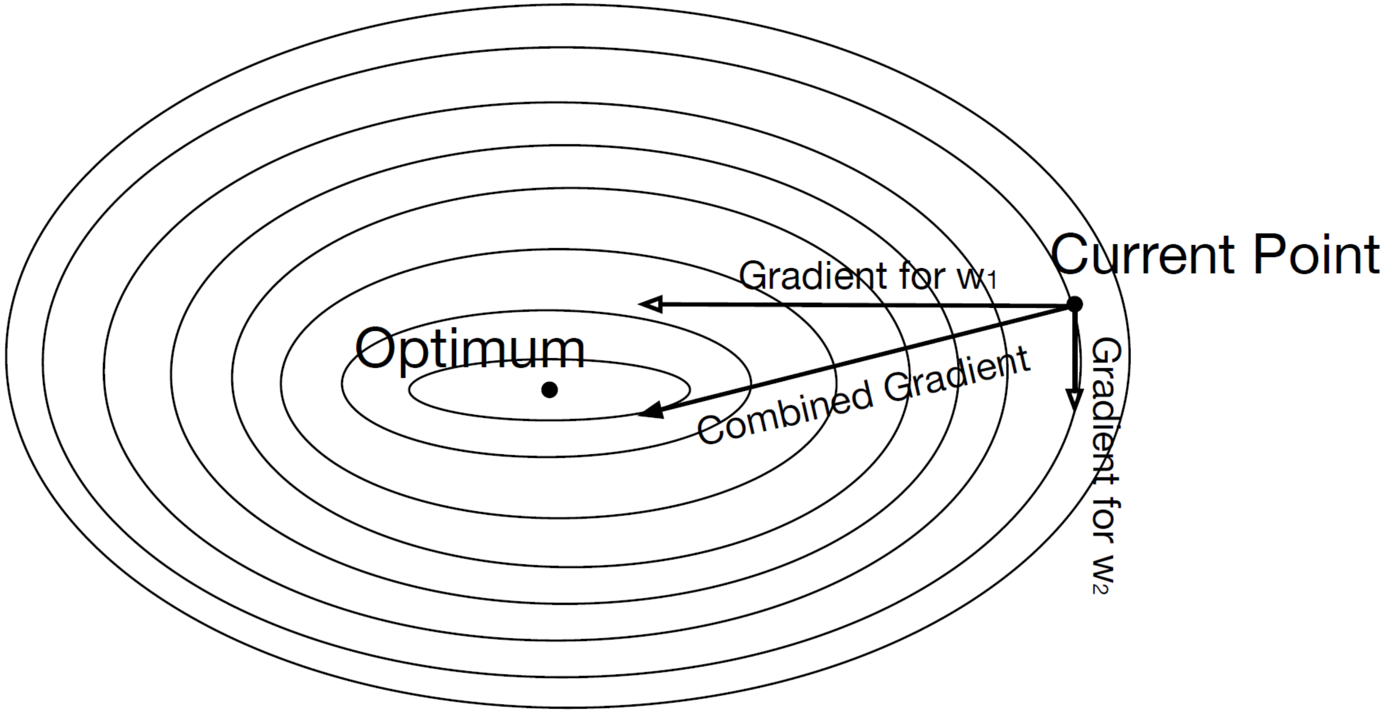
- Adjust weights, moving at the direction of the gradient.

$$\boldsymbol{w} = \boldsymbol{w} - \boldsymbol{\eta \partial f}$$

Gradient Descent



Gradient Descent



Backpropagation

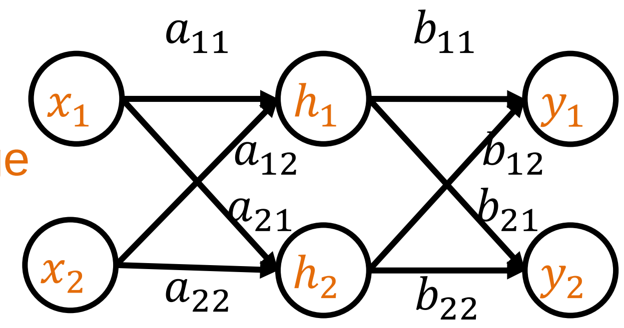
- How can we compute the gradients? Backpropagation!
- Main idea:
  - Start from the final layer: compute the gradients for the weights of the final layer.
  - Use these gradients to compute the gradients of previous layers using the chain rule
  - Propagate the error backwards
- Backpropagation essentially is an application of the chain rule for differentiation.
  - Chain rule:

$$\frac{\partial g\big(f(x)\big)}{\partial x} = \frac{\partial g\big(f(x)\big)}{\partial f}\frac{\partial f(x)}{\partial x}$$

Forward and backward passes

- The training process works as follows:
  - Start with some initial weights
  - Forward pass: Compute the outputs of all internal nodes
  - Backward pass: Perform backpropagation to estimate the gradients
  - Change the weights to move towards the direction of the gradient
  - Repeat

All terms in orange are computed in the forward pass



Error: $E = \|y - t\|^2 = (y_1 - t_1)^2 + (y_2 - t_2)^2$

$$\frac{\partial E}{\partial b_{11}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial s_{y_1}}\frac{\partial s_{y_1}}{\partial b_{11}} = \delta_{y_1} h_1$$

$$\boxed{\delta_{y_1} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial s_{y_1}} = 2(y_1 - t_1)g'(s_{y_1}) = \frac{\partial E}{\partial s_{y_1}}}$$

Notation:
Activation function: g

$s_{y_1} = b_{11}h_1 + b_{12}h_2$, $y_1 = g(s_{y_1})$
$s_{y_2} = b_{21}h_1 + b_{22}h_2$, $y_2 = g(s_{y_2})$
$s_{h_1} = a_{11}x_1 + a_{12}x_2$, $h_1 = g(s_{h_1})$
$s_{h_2} = a_{21}x_1 + a_{22}x_2$, $h_2 = g(s_{h_2})$

$$\frac{\partial E}{\partial b_{21}} = \delta_{y_2} h_1 \qquad \boxed{\delta_{y_2} = \frac{\partial E}{\partial s_{y_2}} = 2(y_2 - t_2)g'(s_{y_2})}$$

$$\frac{\partial E}{\partial b_{12}} = \delta_{y_1} h_2 \qquad \frac{\partial E}{\partial b_{22}} = \delta_{y_2} h_2$$

We have already computed h_1, h_2

$$\frac{\partial E}{\partial a_{11}} = \frac{\partial E}{\partial s_{h_1}}\frac{\partial s_{h_1}}{\partial a_{11}} = \delta_{h_1} x_1 \qquad \frac{\partial E}{\partial a_{22}} = \frac{\partial E}{\partial s_{h_2}}\frac{\partial s_{h_2}}{\partial a_{22}} = \delta_{h_2} x_2 \qquad \frac{\partial E}{\partial a_{21}} = \delta_{h_1} x_2 \qquad \frac{\partial E}{\partial a_{12}} = \delta_{h_2} x_1$$

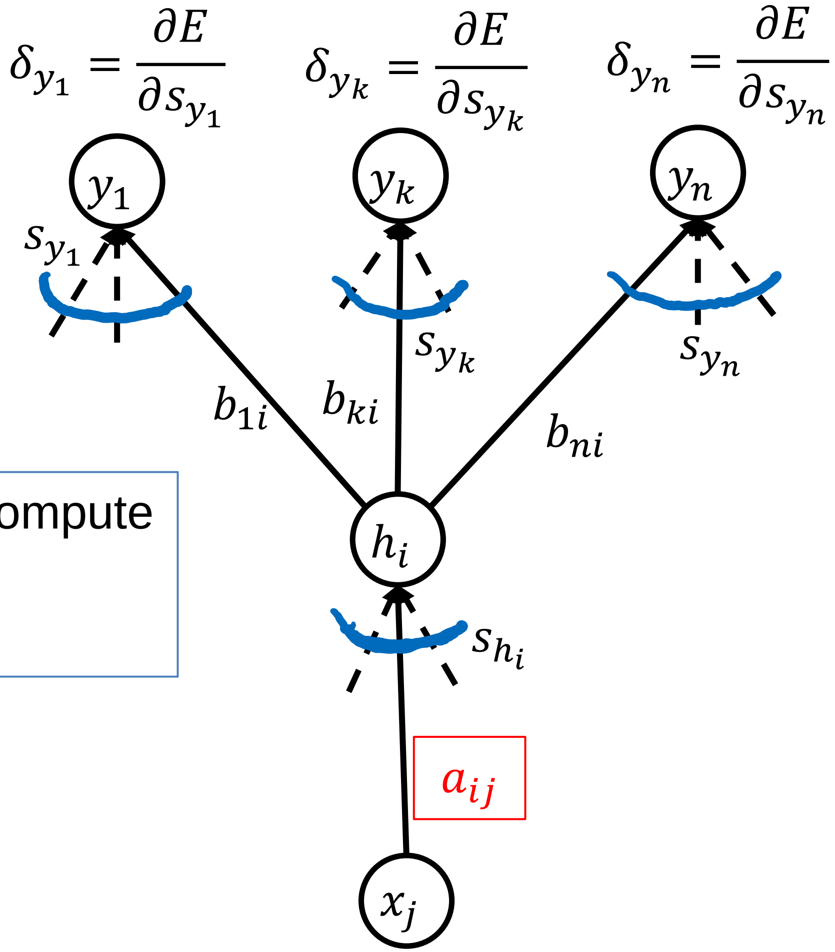
$$\delta_{h_1} = \frac{\partial E}{\partial s_{h_1}} = \frac{\partial E}{\partial h_1}\frac{\partial h_1}{\partial s_{h_1}} = \left(\frac{\partial E}{\partial s_{y_1}}\frac{\partial s_{y_1}}{\partial h_1} + \frac{\partial E}{\partial s_{y_2}}\frac{\partial s_{y_2}}{\partial h_1}\right)g'(s_{h_1}) = (\delta_{y_1} b_{11} + \delta_{y_2} b_{21})g'(s_{h_1})$$

$$\delta_{h_2} = (\delta_{y_1} b_{12} + \delta_{y_2} b_{22})g'(s_{h_2})$$

Backpropagation

We have already computed the δ_{y_k}'s at the previous step of the back propagation

We have already computed the h_i and x_j's at the forward pass



We want to compute $\frac{\partial E}{\partial a_{ij}}$

$$\frac{\partial E}{\partial a_{ij}} = \sum_{k=1}^{n} \delta_{y_k} b_{ki}\, g'(s_{h_i}) x_j$$

For the sigmoid activation function:

$$g(t) = \frac{1}{1+e^{-t}}$$

The derivative is:

$$g'(t) = g(t)(1 - g(t))$$

This makes it easy to compute it. We have:

$$g'(s_{h_i}) = h_i(1 - h_i)$$

Therefore

$$\frac{\partial E}{\partial a_{ij}} = \sum_{k=1}^{n} \delta_{y_k} b_{ki}\, h_i(1 - h_i) x_j$$

Stochastic gradient descent

- Ideally the loss should be the average loss over all training data.
- We would need to compute the loss for all training data every time we update the gradients.
  - However, this is expensive.
- Stochastic gradient descent: Consider one input point at the time. Each point is considered only once.
- Intermediate solution: Use mini-batches of data points.

WORD EMBEDDINGS

Thanks to Chris Manning for the slides

Basic Idea

- You can get a lot of value by representing a word by means of its neighbors
- "You shall know a word by the company it keeps"
    - (J. R. Firth 1957: 11)
- One of the most successful ideas of modern statistical NLP

government debt problems turning into	banking	crises as has happened in
saying that Europe needs unified	banking	regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

Basic idea

Define a model that aims to predict between a center word w_c and context words in some window of length m in terms of word vectors

$$\ldots \underbrace{\overset{w_{c-2}}{\text{turning}}\ \overset{w_{c-1}}{\text{into}}}_{\text{context words}}\ \underbrace{\overset{w_c}{\text{banking}}}_{\text{Center word}}\ \underbrace{\overset{w_{c+1}}{\text{crises}}\ \overset{w_{c+2}}{\text{as}}}_{\text{context words}} \ldots$$

window of size 2 each side

Word2Vec

Predict between every word and its context words

Two **algorithms**

1. Skip-grams (SG)

   Predict context words given the center word

   $P(w_{c-1}|w_c), P(w_{c-2}|w_c), P(w_{c+1}|w_c), P(w_{c+2}|w_c)$

2. Continuous Bag of Words (CBOW)

   Predict center word from a bag-of-words context

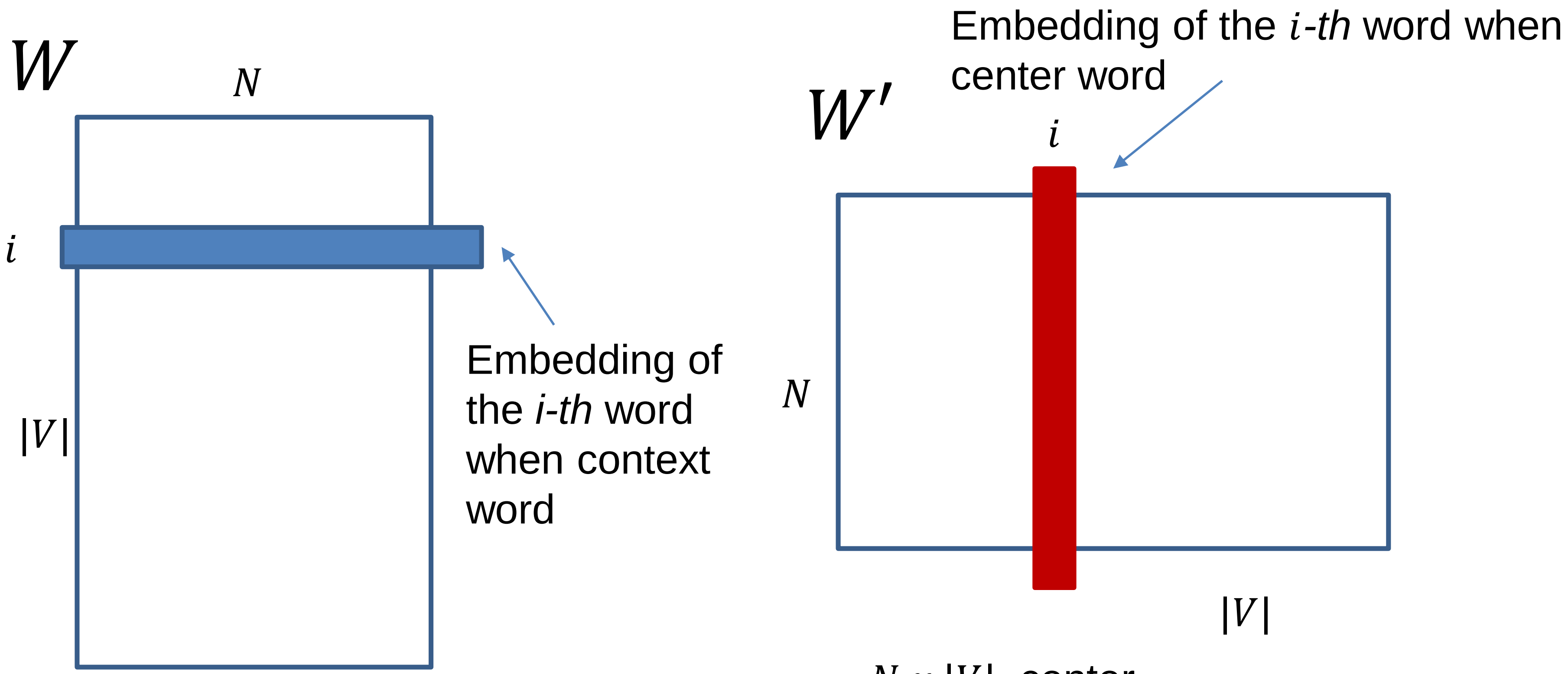
$$P\left(w_c \middle| w_{c-2}, w_{c-1}, w_{c+1}, w_{c+2}\right)$$

Position independent (do not account for distance from center)

CBOW

Use a window of context words to predict the center word

Learn two matrices (N size of embedding, $|V|$ number of words)



$|V|\ x\ N$ context embeddings when *input*

$N\ x\ |V|$ center embeddings when *output*

CBOW

Given window size *m*, $x^{(c)}$ one hot vector for context words, *y* one hot vector for the center word

1. Input: the *one hot vectors* for the *2m* context words
$x^{(c-m)}, \ \ldots, \ x^{(c-1)}, x^{(c+1)}, \ldots, x^{(c+m)}$

2. Compute the *embeddings of the context words*
$v_{c-m} = Wx^{(c-m)}, \ \ldots, v_{c-1} = Wx^{(c-1)}, v_{c+1} = Wx^{(c+1)}, \ldots, v_{c+m} = Wx^{(c+m)}$

3. *Average* these vectors: $\hat{v} = \frac{v_{c-m}+v_{c-m+1}+\cdots v_{c+m}}{2m}, \ \hat{v} \in R^N$

4. Generate a *score vector:* $z = W' \hat{v}$

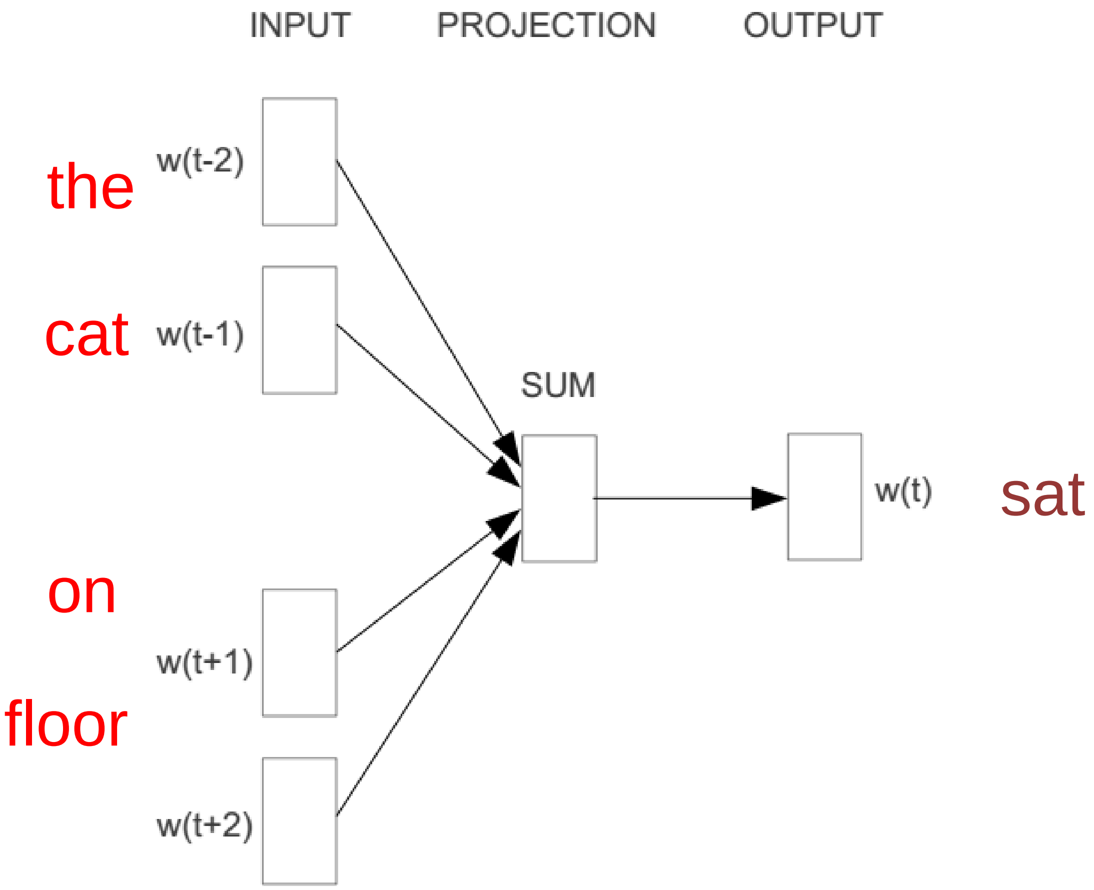
5. Turn the *score vector to probabilities:* $\hat{y}$ = *softmax(z)*

Softmax

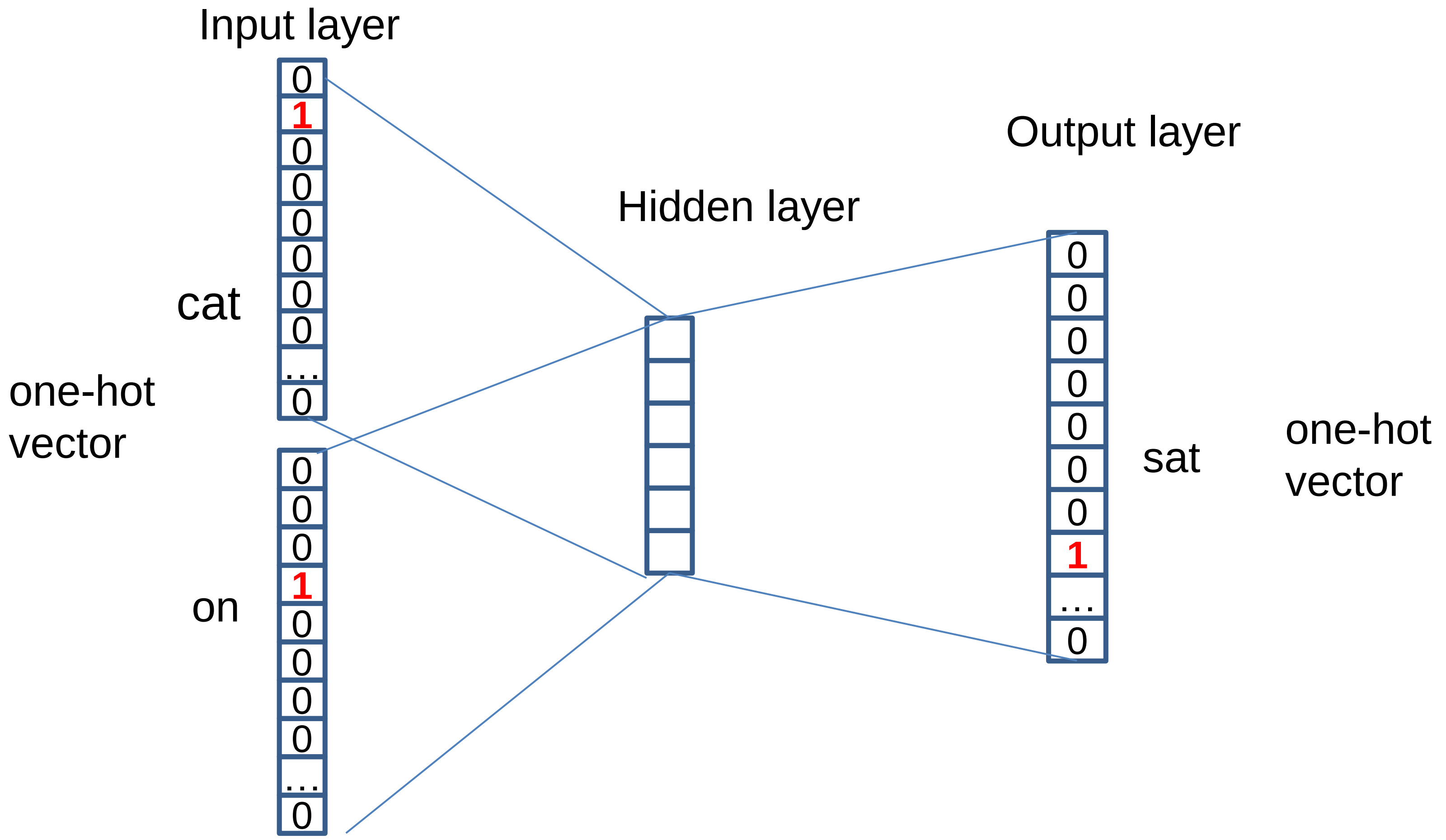
$$p_i = \frac{e^{v_i}}{\sum_j e^{v_j}}$$

We want this to be close to 1 for the center word

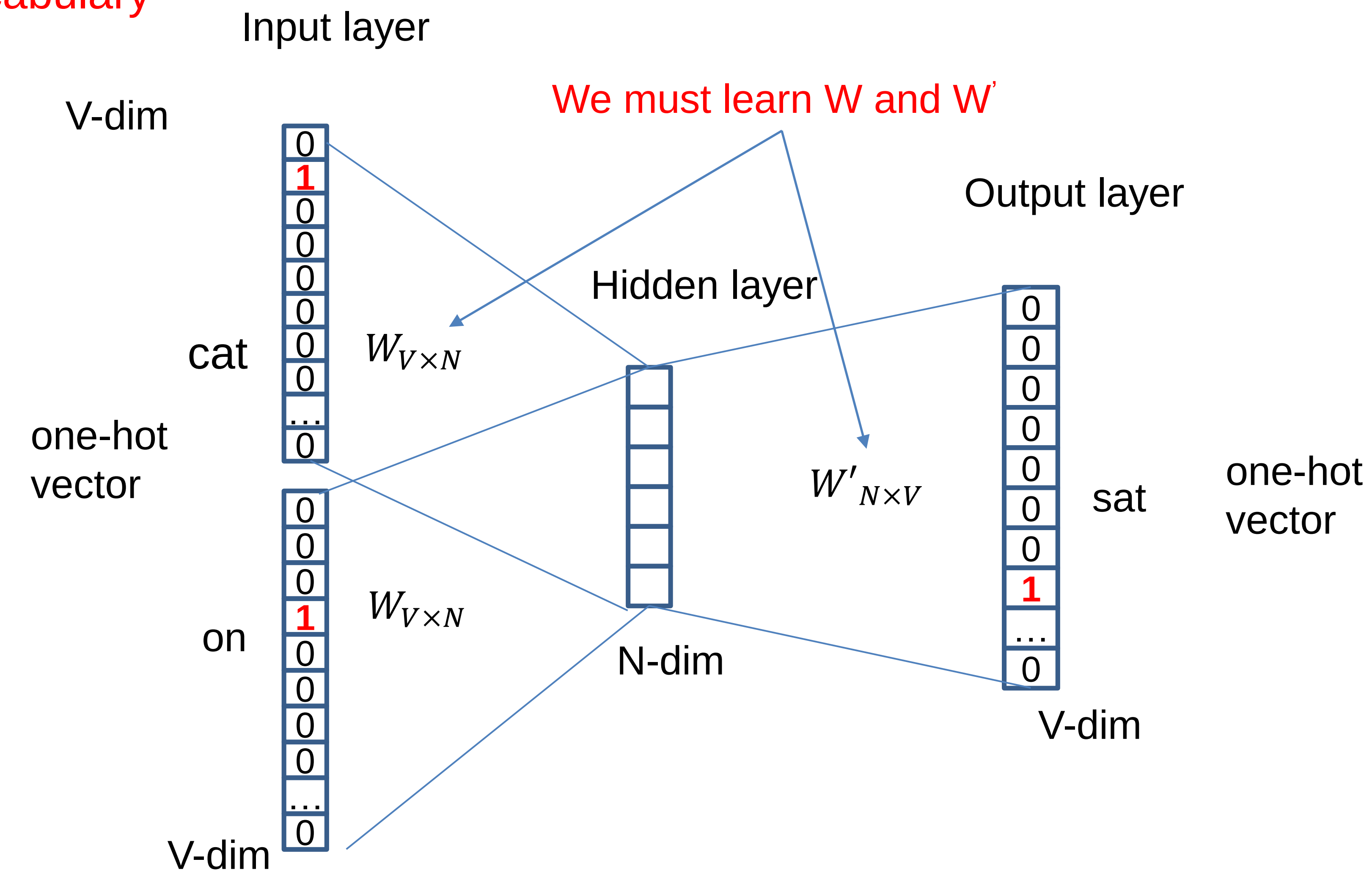
- E.g. "The cat sat on floor"
  - Window size = 2



Index of cat in vocabulary

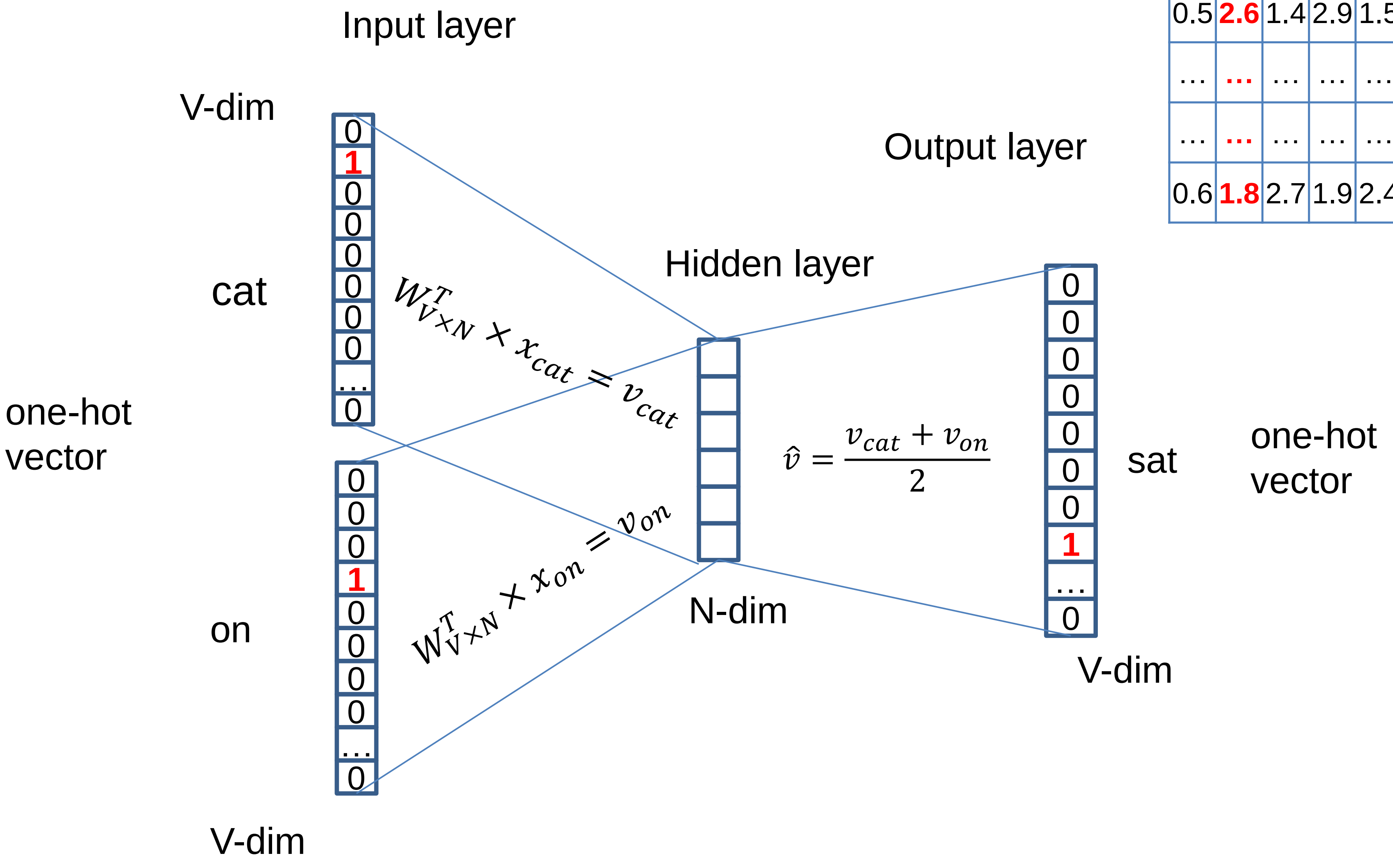


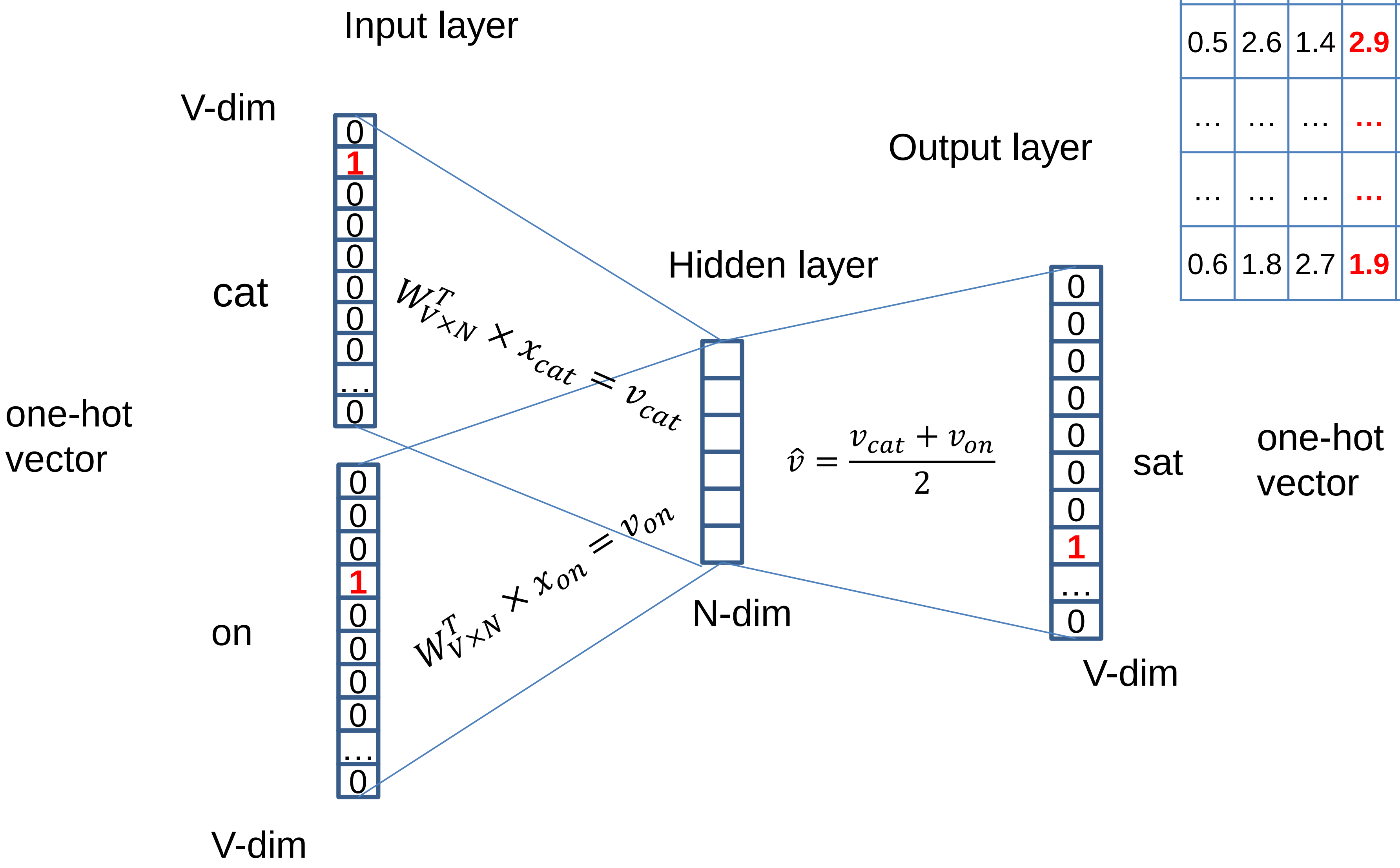
Index of cat in vocabulary



$$W_{V\times N}^{T} \times x_{cat} = v_{cat}$$

$$\begin{bmatrix} 0.1 & \color{red}{2.4} & 1.6 & 1.8 & 0.5 & 0.9 & \dots & \dots & \dots & 3.2 \\ 0.5 & \color{red}{2.6} & 1.4 & 2.9 & 1.5 & 3.6 & \dots & \dots & \dots & 6.1 \\ \dots & \color{red}{\dots} & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \color{red}{\dots} & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0.6 & \color{red}{1.8} & 2.7 & 1.9 & 2.4 & 2.0 & \dots & \dots & \dots & 1.2 \end{bmatrix} \times \begin{bmatrix} 0 \\ \color{red}{1} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \dots \\ 0 \end{bmatrix} = \begin{bmatrix} \color{red}{2.4} \\ \color{red}{2.6} \\ \color{red}{\dots} \\ \color{red}{\dots} \\ \color{red}{1.8} \end{bmatrix}$$



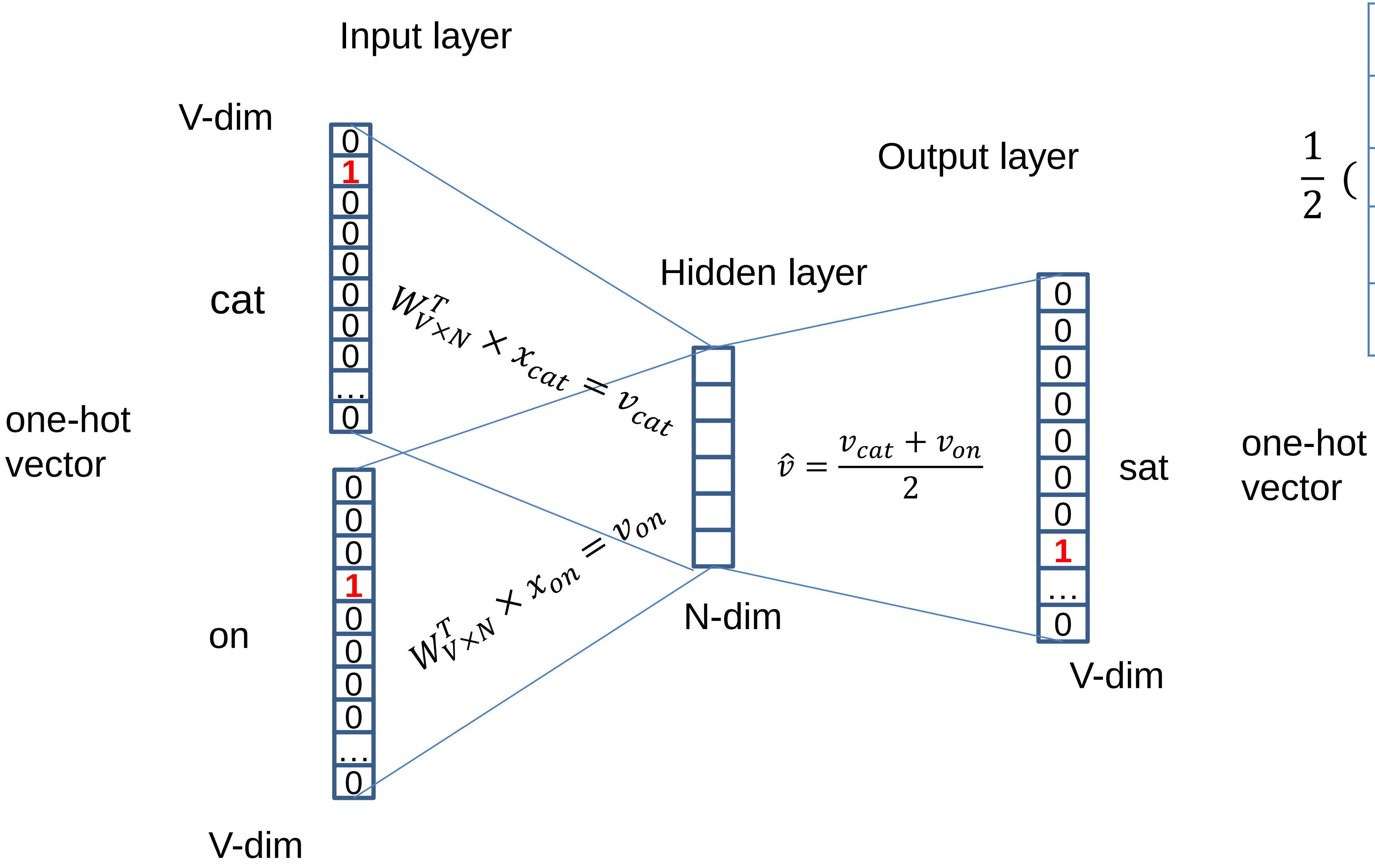


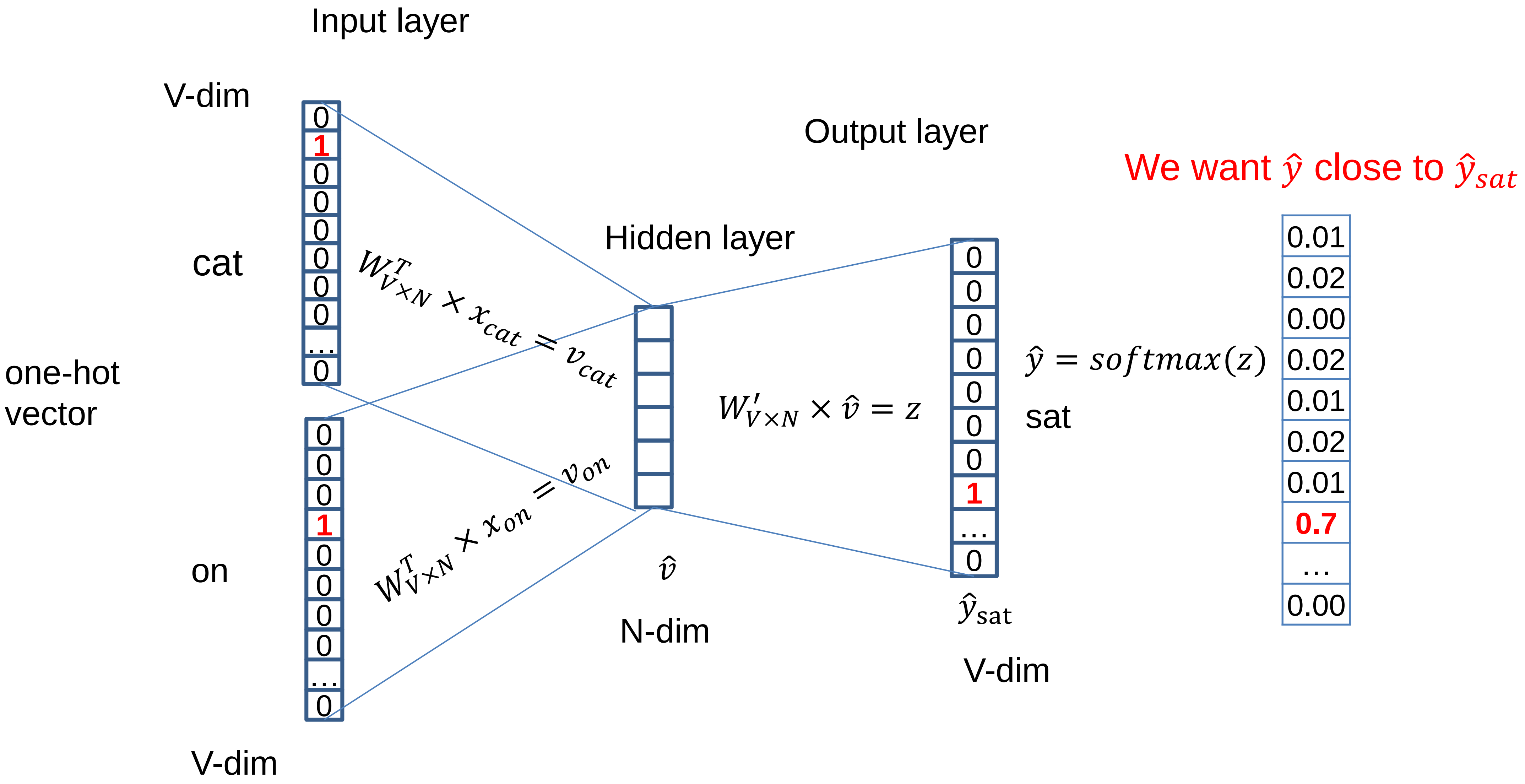
$$W_{V\times N}^{T} \times x_{on} = v_{on}$$

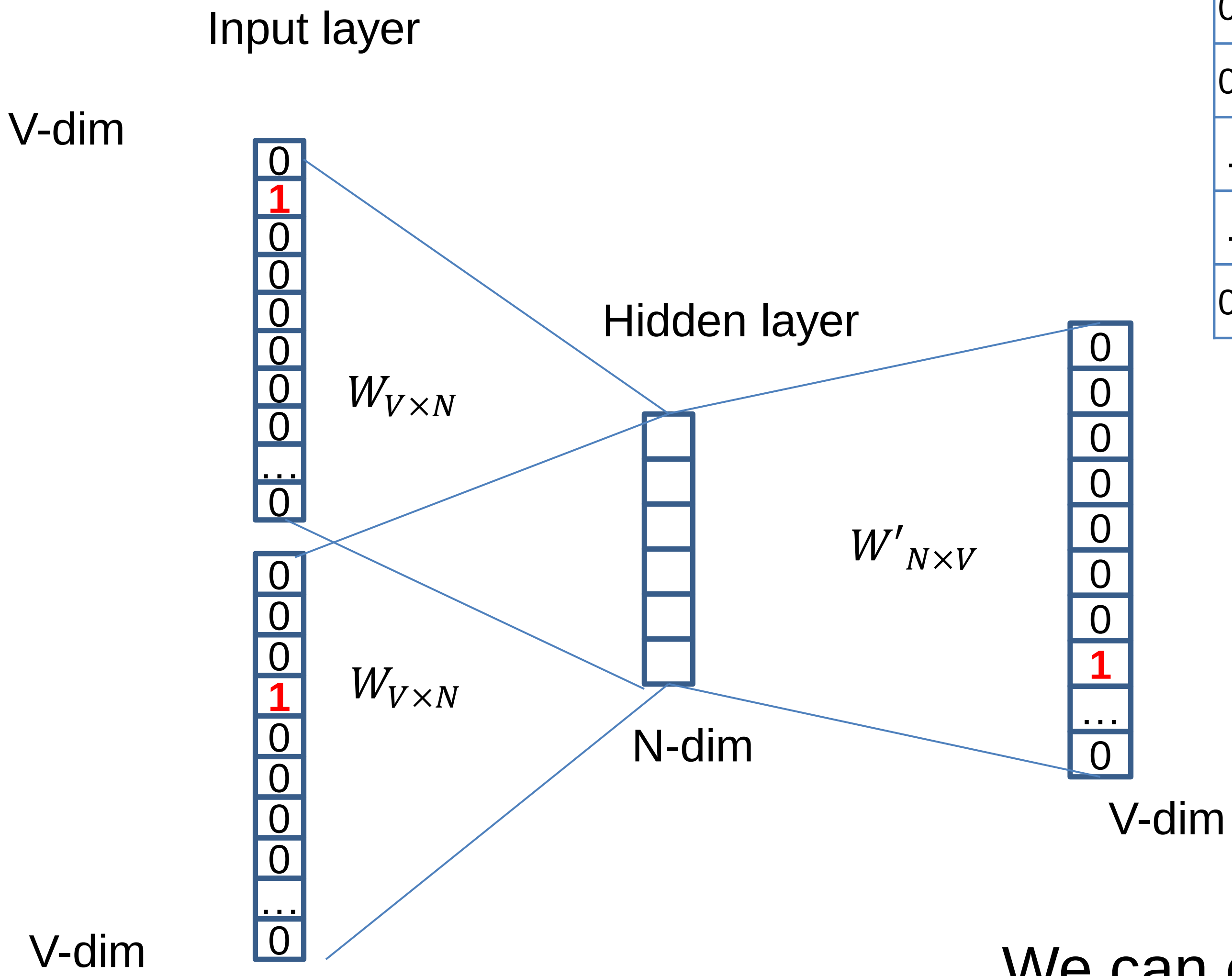
$$\begin{bmatrix} 0.1 & 2.4 & 1.6 & 1.8 & 0.5 & 0.9 & \dots & \dots & \dots & 3.2 \\ 0.5 & 2.6 & 1.4 & 2.9 & 1.5 & 3.6 & \dots & \dots & \dots & 6.1 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0.6 & 1.8 & 2.7 & 1.9 & 2.4 & 2.0 & \dots & \dots & \dots & 1.2 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ \dots \\ 0 \end{bmatrix} = \begin{bmatrix} 1.8 \\ 2.9 \\ \dots \\ \dots \\ 1.9 \end{bmatrix}$$

$$\frac{1}{2}(v_{cat} + v_{on}) = \hat{v}$$

$$\frac{1}{2}\left(\begin{bmatrix}2.4\\2.6\\\dots\\\dots\\1.8\end{bmatrix} + \begin{bmatrix}1.8\\2.9\\\dots\\\dots\\1.9\end{bmatrix}\right) = \begin{bmatrix}2.1\\4.05\\\dots\\\dots\\1.85\end{bmatrix}$$




Input layer
V-dim
cat
one-hot vector
on
V-dim
$W^T_{V\times N} \times x_{cat} = v_{cat}$
$W^T_{V\times N} \times x_{on} = v_{on}$
Hidden layer
$\hat{v}$
N-dim
$W'_{V\times N} \times \hat{v} = z$
Output layer
sat
$\hat{y}_{sat}$
V-dim
$\hat{y} = softmax(z)$
We want $\hat{y}$ close to $\hat{y}_{sat}$
0.01
0.02
0.00
0.02
0.01
0.02
0.01
0.7
…
0.00



$W_{V\times N}$

0.1	**2.4**	1.6	1.8	0.5	0.9	…	…	…	3.2
0.5	**2.6**	1.4	2.9	1.5	3.6	…	…	…	6.1
…	**…**	…	…	…	…	…	…	…	…
…	**…**	…	…	…	…	…	…	…	…
0.6	**1.8**	2.7	1.9	2.4	2.0	…	…	…	1.2

The word embeddings

We can consider either W (context) or W' (center) as the word's representation.
Or even take the average.

Skipgram

Given the center word, predict (or, generate) the context words

W: $N \times |V|$, input matrix, word representation as center word
W': $|V| \times N$, output matrix, word representation as context word

$y^{(j)}$ one hot vector for context words

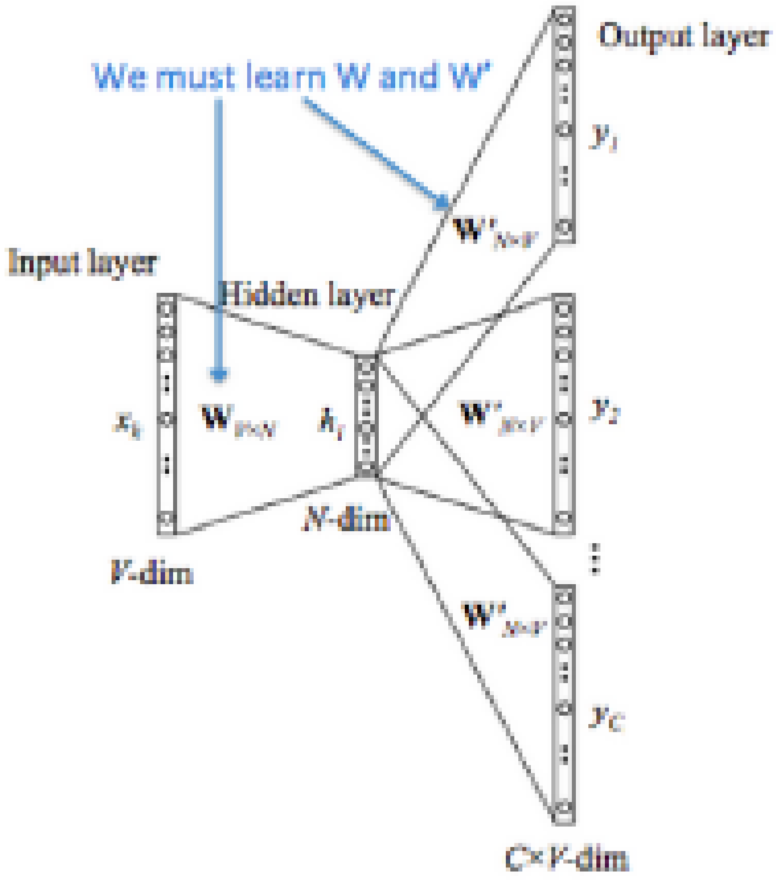
1. Get *one hot vector* of the center word x^c

2. Get the *embedding of the center word:* $v_c = W\, x^c$

3. Get the *embedding of all context words:* $z = W'\, v_c$

5. Turn the *score vector into probabilities: $\hat{y}$ = softmax(z)*

We want this to be close to 1 for the context words



Skipgram

- For each word $t = 1 \ldots T$, predict surrounding words in a window of "radius" m of every word.
- Objective function: Maximize the probability of any context word given the current center word:

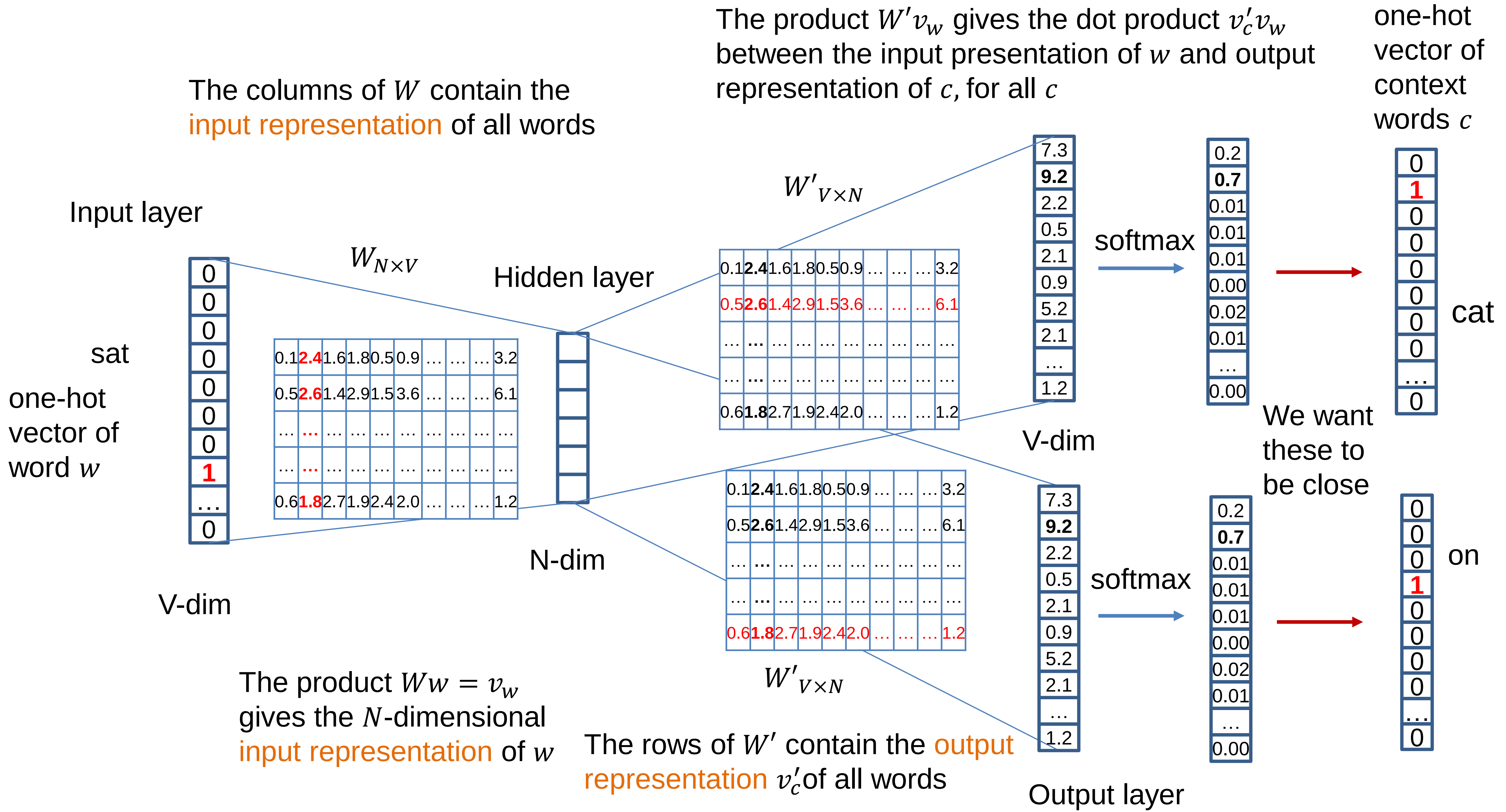
likelihood

$$J'(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \ne 0}} p(w_{t+j} | w_t; \theta)$$

Negative Log Likelihood

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \ne 0}} \log p(w_{t+j} | w_t; \theta)$$

where θ represents all variables we will optimize



The columns of W contain the input representation of all words
The product $W'v_w$ gives the dot product $v_c'v_w$ between the input presentation of w and output representation of c, for all c
one-hot vector of context words c
Input layer
sat
one-hot vector of word w
V-dim
$W_{N \times V}$
Hidden layer
N-dim
$W'_{V \times N}$
softmax
cat
on
V-dim
We want these to be close
Output layer
The product $Ww = v_w$ gives the N-dimensional input representation of w
The rows of W' contain the output representation v_c' of all words

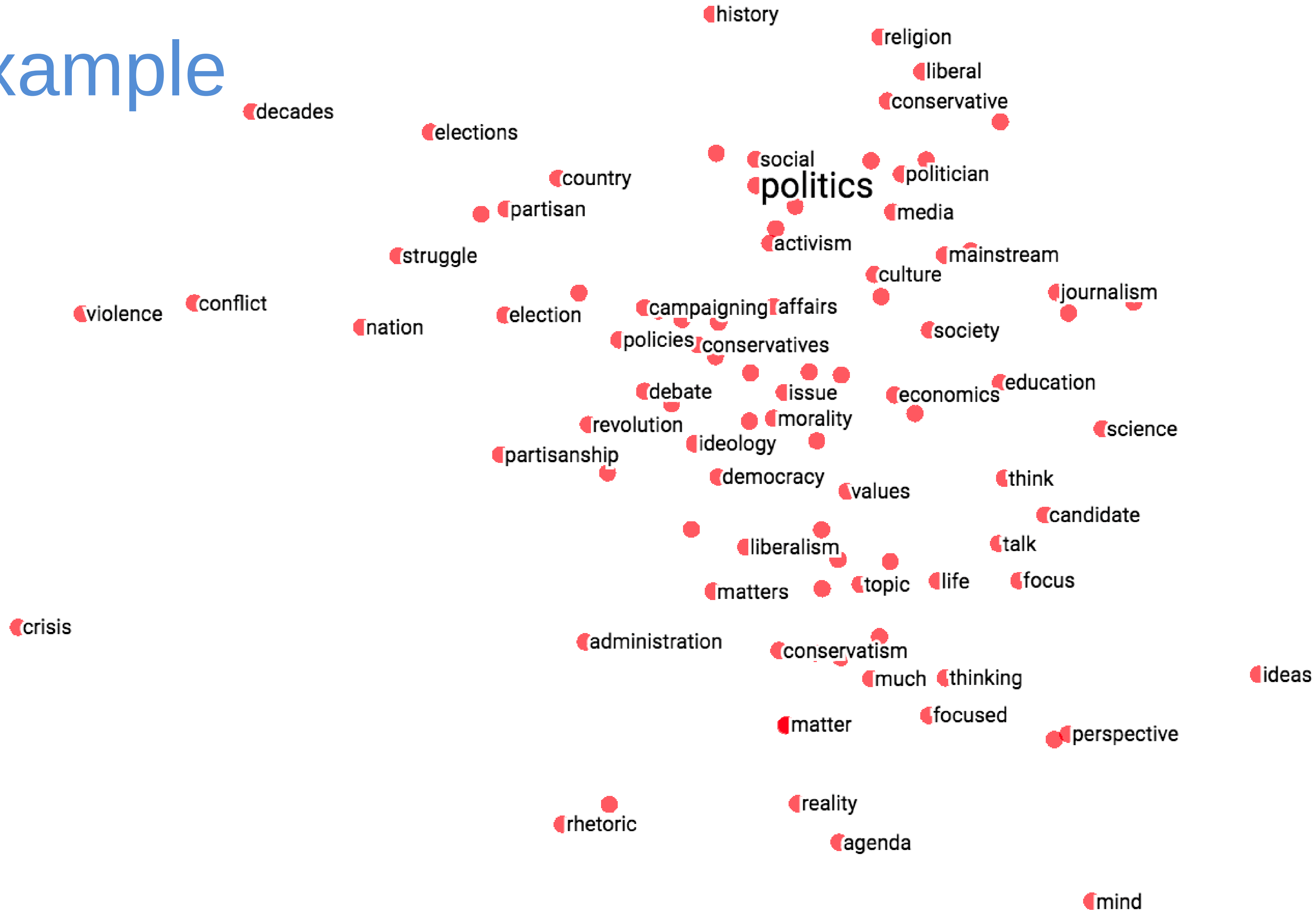
- The basic skipgram utilizes the softmax function:

$$p(c|w) = \frac{\exp\left(v_c'^T v_w\right)}{\sum_{i=1}^{T} \exp(v_i'^T v_w)}$$

- Where:
  - T – # of words in the corpus.
  - v_w - input vector of *w*.
  - v'_w - output vector of *w*.

Word	Input	Output
King	[0.2,0.9,0.1]	[0.5,0.4,0.5]
Queen	[0.2,0.8,0.2]	[0.4,0.5,0.5]
Apple	[0.9,0.5,0.8]	[0.3,0.9,0.1]
Orange	[0.9,0.4,0.9]	[0.1,0.7,0.2]

An example


history
religion
liberal
conservative
decades
elections
social
politician
country
politics
partisan
media
activism
struggle
mainstream
culture
journalism
violence
conflict
election
campaigning
affairs
nation
society
policies
conservatives
debate
issue
economics
education
revolution
morality
science
ideology
partisanship
democracy
values
think
candidate
liberalism
talk
matters
topic
life
focus
crisis
administration
conservatism
much
thinking
ideas
matter
focused
perspective
reality
rhetoric
agenda
mind

These representations are *very good* at encoding similarity and dimensions of similarity!

- Analogies testing dimensions of similarity can be solved quite well just by doing vector subtraction in the embedding space

  Syntactically

  - $x_{apple} - x_{apples} \approx x_{car} - x_{cars} \approx x_{family} - x_{families}$
  - Similarly for verb and adjective morphological forms

  Semantically (Semeval 2012 task 2)

  - $x_{shirt} - x_{clothing} \approx x_{chair} - x_{furniture}$
  - $x_{king} - x_{man} \approx x_{queen} - x_{woman}$

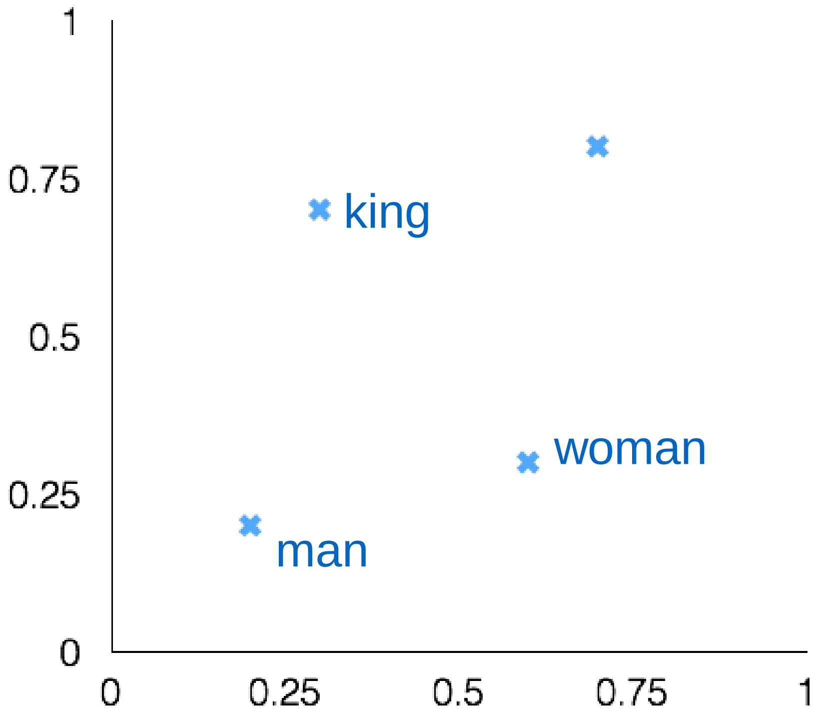
Test for linear relationships, examined by Mikolov et al.

a:b :: c:?

$$d = \arg\max_x \frac{(w_b - w_a + w_c)^T w_x}{||w_b - w_a + w_c||}$$

man:woman :: king:?

+	king	[0.30 0.70]
–	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]



SUPERVISED LEARNING

Learning

- Supervised Learning: learn a model from the data using labeled data.
  - Classification and Regression are the prototypical examples of supervised learning tasks. Other are possible (e.g., ranking)
- Unsupervised Learning: learn a model – extract structure from unlabeled data.
  - Clustering and Association Rules are prototypical examples of unsupervised learning tasks.
- Semi-supervised Learning: learn a model for the data using both labeled and unlabeled data.
- Self-supervised Learning: Use supervised learning techniques for predicting target variables that are extracted in an unsupervised way

Supervised Learning Steps

- Model the problem
  - What is you are trying to predict? What kind of optimization function do you need? Do you need classes or probabilities?
- Extract Features
  - How do you find the right features that help to discriminate between the classes?
- Obtain labeled data
  - Obtain a collection of labeled data. Make sure it is large enough, accurate and representative. Ensure that classes are well represented.
- Decide on the technique
  - What is the right technique for your problem?
- Apply in practice
  - Can the model be trained for very large data? How do you test how you do in practice? How do you improve?

Modeling the problem

- Sometimes it is not obvious. Consider the following three problems
  - Detecting if an email is spam
  - Categorizing the queries in a search engine
  - Ranking the results of a web search
  - Predicting the reply to a question.
  - Predicting the path of a moving object

Feature extraction

- Feature extraction, or feature engineering is the most tedious but also the most important step
  - How do you separate the players of the Greek national team from those of the Swedish national team?

- One line of thought: throw features to the classifier and the classifier will figure out which ones are important
  - More features, means that you need more training data
- Another line of thought: Feature Selection: Select carefully the features using various functions and techniques
  - Computationally intensive

- Deep Neural Networks
  - Use raw data for classification in a supervised or self-supervised way
  - Produce a representation of the data using intermediate weights of the DNNs
  - Use these representations as features

Training data

- An overlooked problem: How do you get labeled data for training your model?
  - E.g., how do you get training data for ranking web search results?
  - Chicken and egg problem
- Usually requires a lot of manual effort and domain expertise and carefully planned labeling
  - Results are not always of high quality (lack of expertise)
  - And they are not sufficient (low coverage of the space)
- Recent trends:
  - Find a source that generates the labeled data for you, or use the data themselves for the prediction task (self-supervised learning)
  - Crowd-sourcing techniques

Dealing with small amounts of labeled data

- Semi-supervised learning techniques have been developed for this purpose.
- Self-training: Train a classifier on the data, and then feed back the high-confidence output of the classifier as input
- Co-training: train two “independent” classifiers and feed the output of one classifier as input to the other.
- Regularization: Treat learning as an optimization problem where you define relationships between the objects you want to classify, and you exploit these relationships
  - Example: Image restoration

Technique

- The choice of technique depends on the problem requirements (do we need a probability estimate?) and the problem specifics (does independence assumption hold? do we think classes are linearly separable?)
- For many cases finding the right technique may be trial and error
- For many cases the exact technique does not matter.

Big Data Trumps Better Algorithms

- If you have enough data then the algorithms are not so important

- The web has made this possible.
  - Especially for text-related tasks
  - Search engine uses the collective human intelligence

Google lecture: [Theorizing from the Data]()

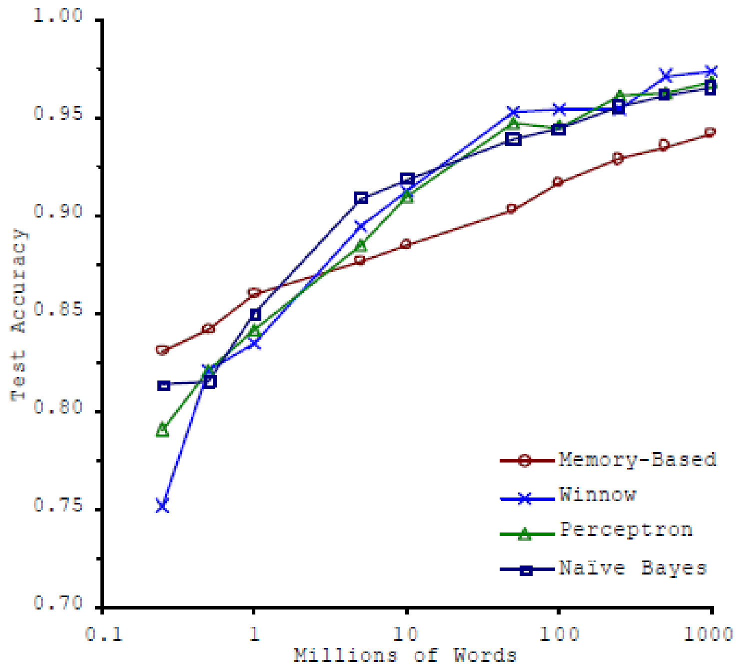


Figure 1. Learning Curves for Confusion Set Disambiguation

Apply-Test

- How do you scale to very large datasets?
  - Distributed computing – map-reduce implementations of machine learning algorithms (Mahaut, over Hadoop, Spark)

- How do you test something that is running online?
  - You cannot get labeled data in this case
  - A/B testing

- How do you deal with changes in data?
  - Active learning