Word Representations

LSA and word embeddings

Word representations

- In vector space, this is a sparse vector with one 1 and a lot of zeros.

[0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]

1-hot representation

Problem (Dimensionality high:)

20K (speech) – 50K (PTB) – 500K (big vocab) – 13M (Google 1T)

book [00000010000000000]

library [00000000000001000]

book & library = 0

Distributional similarity based representations

You can get a lot of value by representing a word by means of its neighbours

"You shall know a word by the company it keeps" — (J. R. Firth 1957: 11)

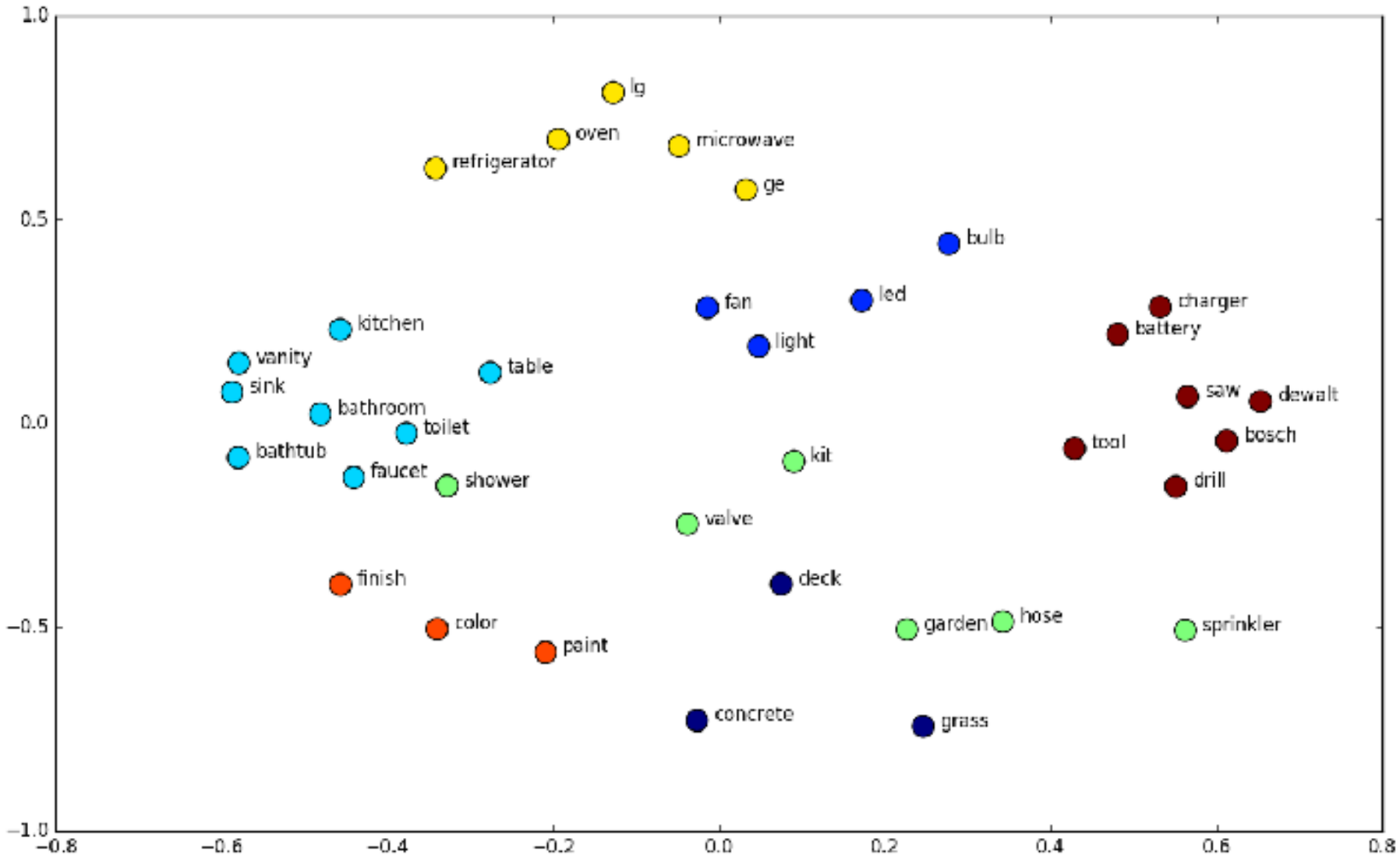
…government debt problems turning into banking crises as has happened in…

…..saying that Europe needs unified banking regulation to replace the hodgepodge…

↖ These words will represent *banking* ↗

- Idea: Exploit context for **better** and **denser** word representations
- How to reduce dimensionality and get better context-aware representation

Dense word representation



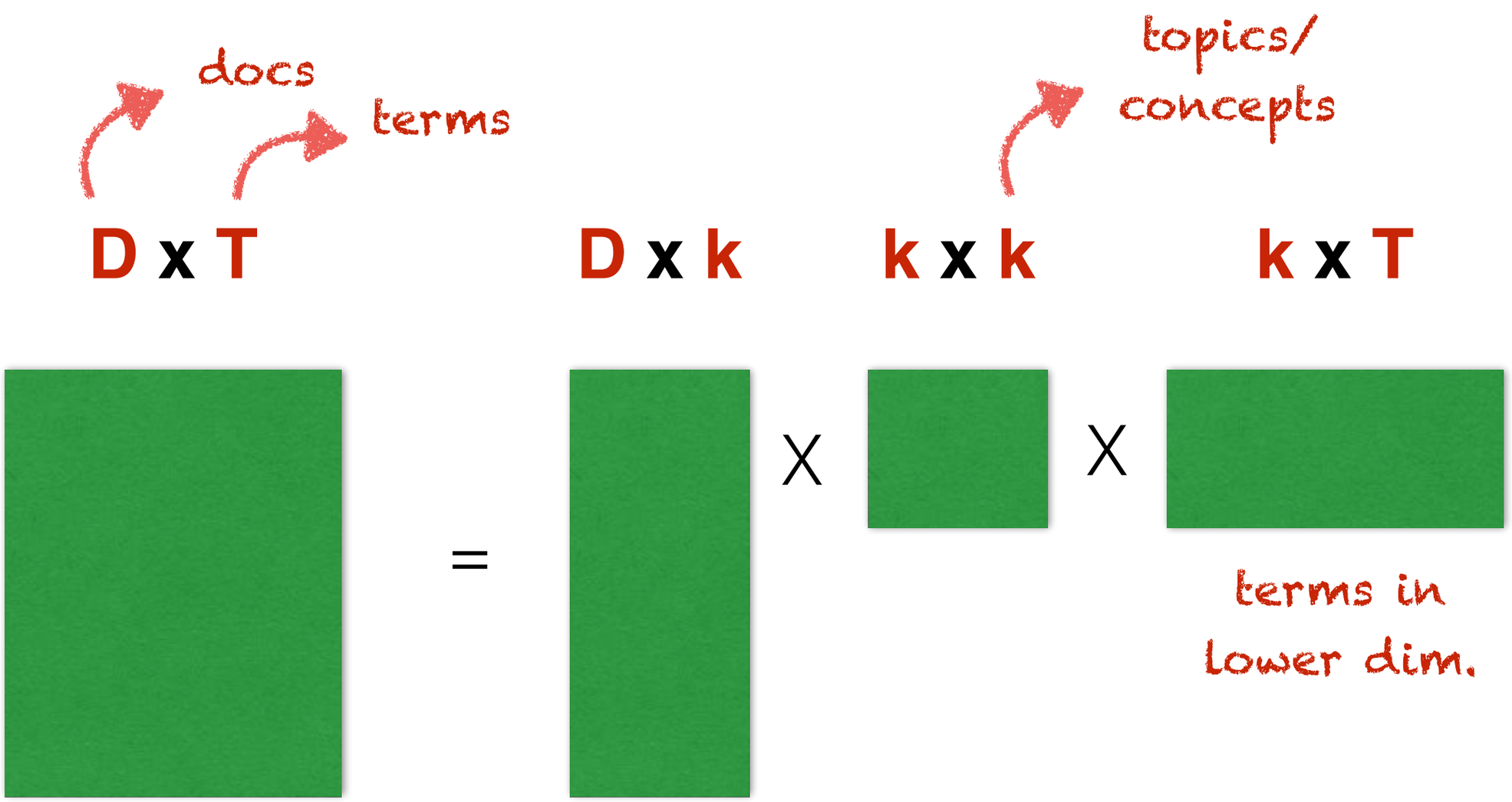
0.286
0.792
−0.177
−0.107
0.109
−0.542
0.349
0.271

garden

- Similar words are nearby in **lower dimension** space
- Cosine similarity between words to compute similarity

Latent Semantic Analysis

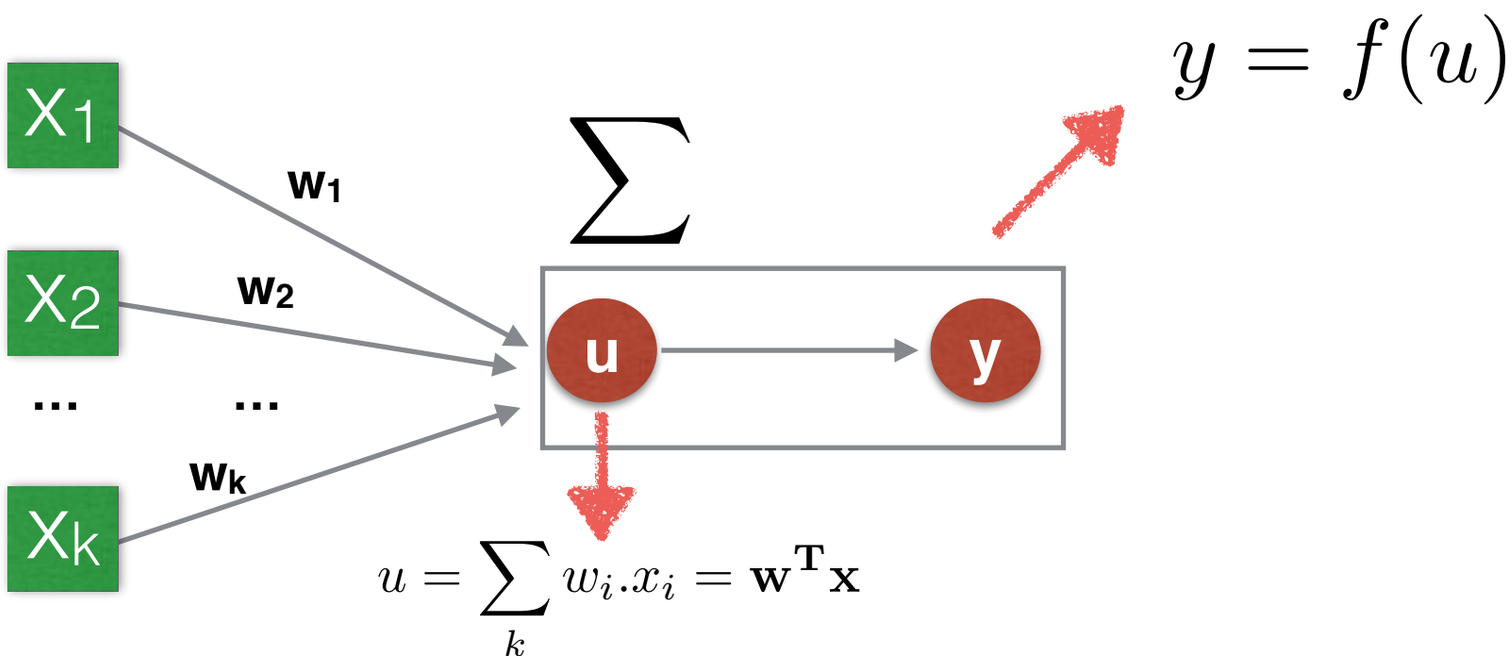
- Idea: Exploit context for **better** and **denser** word representations
- Decompose the term document matrix using *singular value decomposition*



Latent Semantic Analysis - Disadvantages

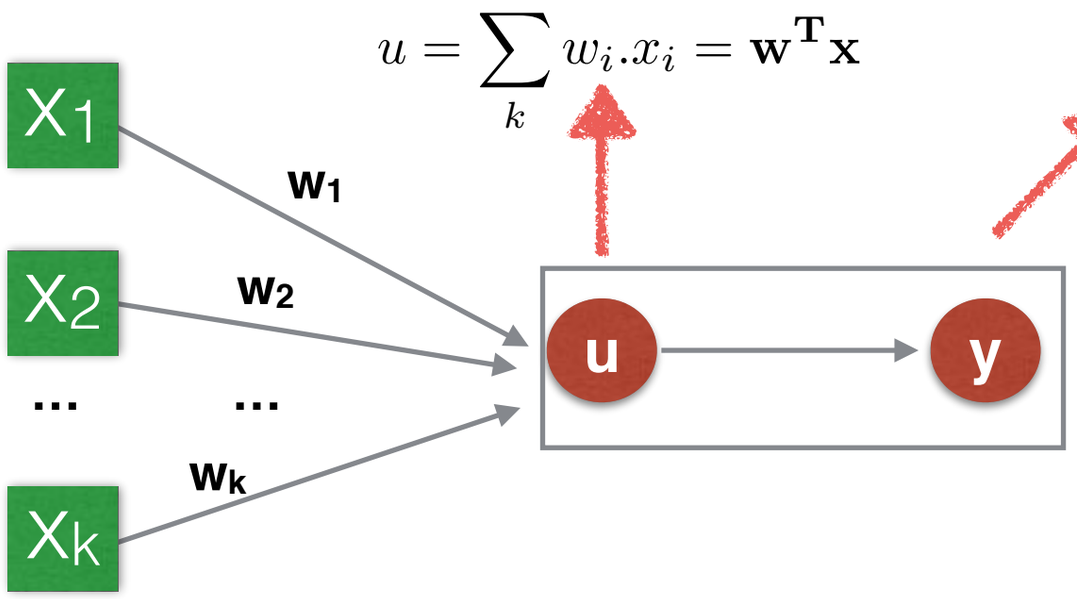
- Term document matrices are essentially vector representations of documents
- Computational cost in SVD scales quadratically for $d \times n$ matrix:
  - $O(nd^2)$ (when $d < n$)
- Not feasible for large collections
- Difficult for out-of-sample words of documents
- Probabilistic framework was promising in PLSI (covered earlier)
- We will now combine ideas from Language models and vector spaces in a neural network based architecture

Artificial Neural Networks - Perceptron



- Simplest neural network is a perceptron
- At each layer, weighted sum of the input is thresholded based on a transfer function **f**
  - Example: logistic function or a logit $\sigma(u) = \frac{1}{1+e^{-u}}$
- A neural network is a *function approximator*
- What are the parameters to be estimated ?

Perceptron Learning



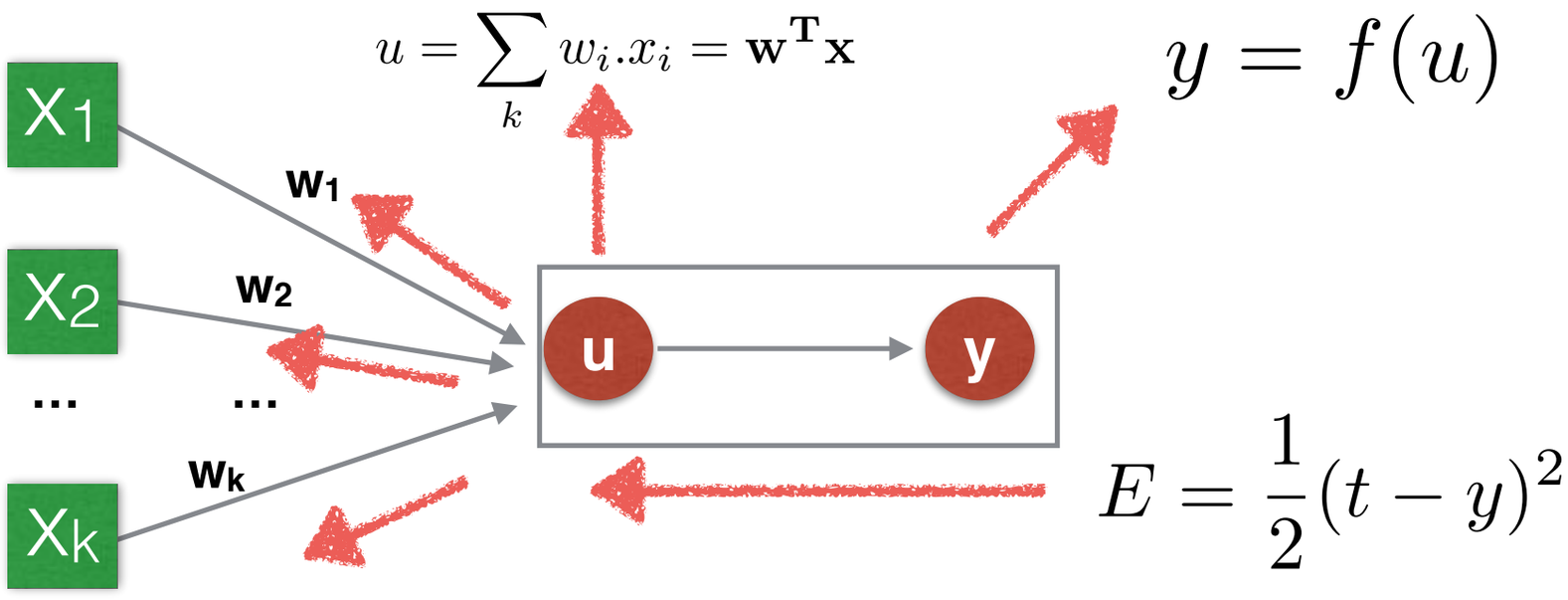
$$E = \frac{1}{2}(t - y)^2$$

- Given a set of training examples (x,t), what are the weights that minimise the training error or loss
- Loss function (E) examples : Squared loss, Cross-entropy loss, Hinge-loss
- **Gradient descent:** Move in the direction of the steepest descent, that is compute update equations for new weights

learning rate

compute $\frac{\partial E}{\partial w_i}$ → $\mathbf{w}^{(\text{new})} = \mathbf{w}^{(\text{old})} - \eta \cdot \frac{\partial E}{\partial w_i}$

Perceptron Learning

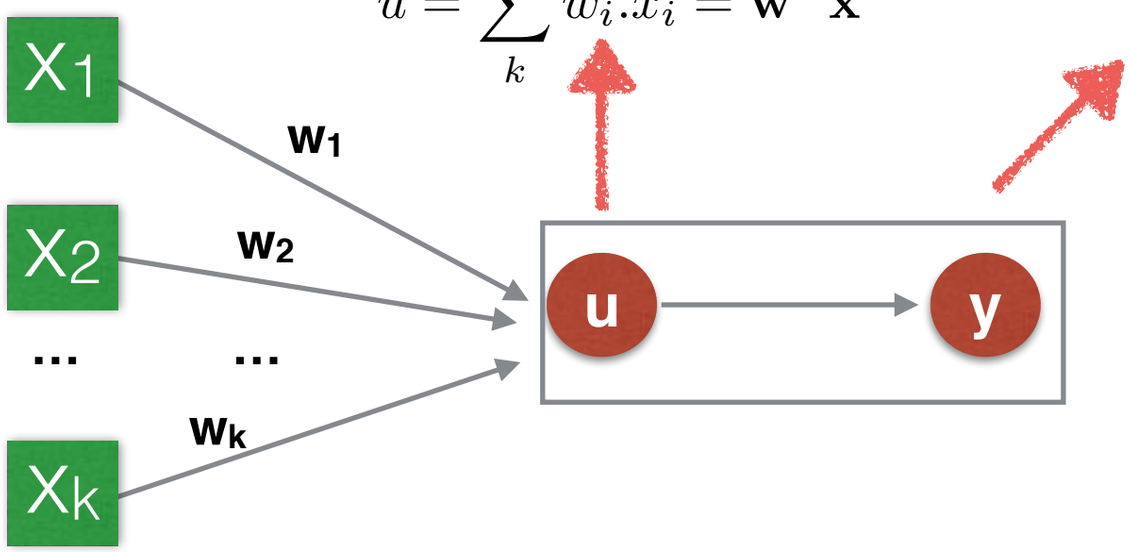


- Given a set of training examples (x,t), what are the weights that minimise the training error or loss
- Loss function (E) examples : Squared loss, Cross-entropy loss, Hinge-loss
- **Gradient descent:** Move in the direction of the steepest descent, that is compute update equations for new weights

learning rate

compute $\frac{\partial E}{\partial w_i}$ → $\mathbf{w}^{(\text{new})} = \mathbf{w}^{(\text{old})} - \eta \cdot \frac{\partial E}{\partial w_i}$

Exercise



$$u = \sum_k w_i.x_i = \mathbf{w^T x}$$

$$\sigma(u) = \frac{1}{1+e^{-u}}$$

hint

$$\Delta z = \frac{\partial z}{\partial y}\Delta y$$

$$\Delta y = \frac{\partial y}{\partial x}\Delta x$$

$$\Delta z = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\Delta x$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

Given squared loss, logistic transfer function

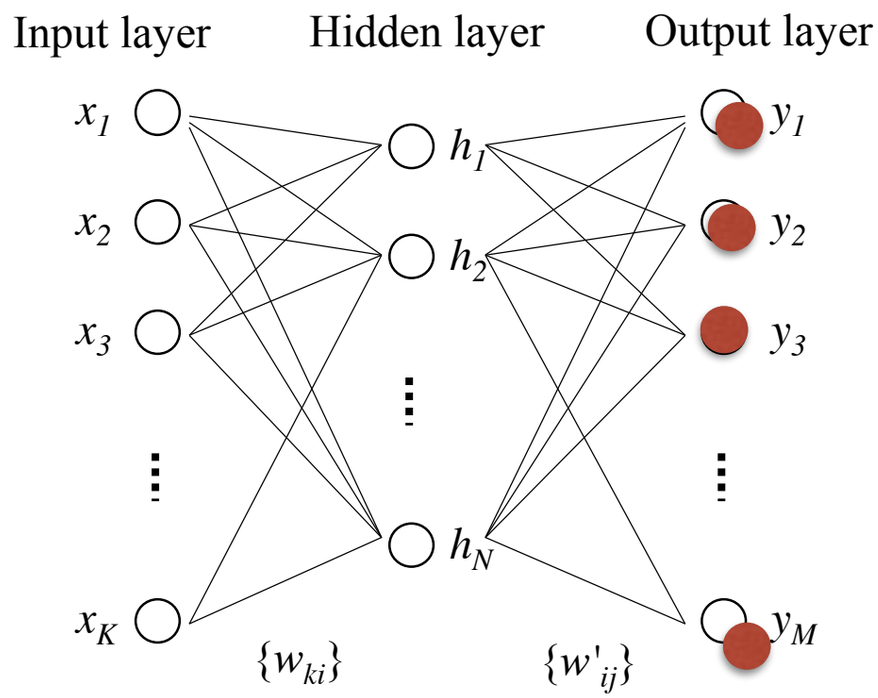
$$E = \frac{1}{2}(t-y)^2 \qquad \sigma(u) = \frac{1}{1+e^{-u}}$$

$$\mathbf{w}^{(\text{new})} = \mathbf{w}^{(\text{old})} - \eta \cdot \frac{\partial E}{\partial w_i}$$

find weight update equation of any of the weights

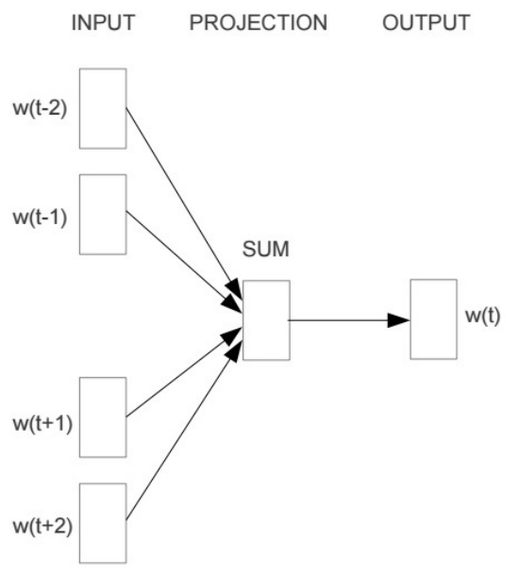
Artificial Neural Networks - Multilayer

- Typically multiple hidden layers, input gets transformed
- Multiple weight matrices
- Stochastic gradient descent used for optimization
- NN designed according to the task at hand
- Training done by a procedure called
  - **Back Propagation:**
    - Forward pass to determine current error
    - backward pass to update all weights

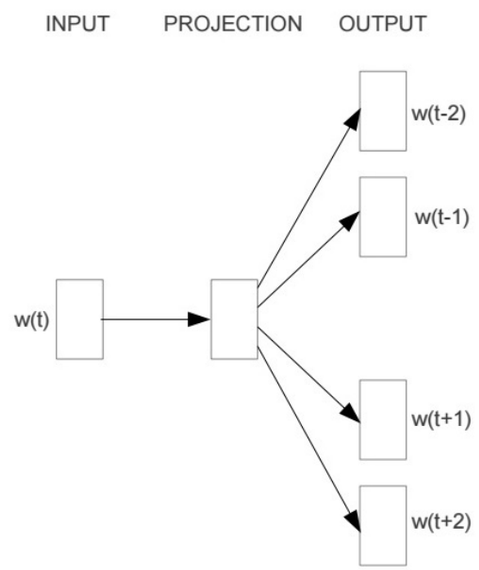


Word Embeddings

- Considers learning word embeddings as a prediction task
- Training context : “germany wins worldcup in brazil”
- **Continuous Bag of Words Model (CBOW):**
  - Guess the central word in the context
  - “germany wins X in brazil”
- **Skip-gram**
  - Guess the context given the central word
  - “X X worldcup X X”

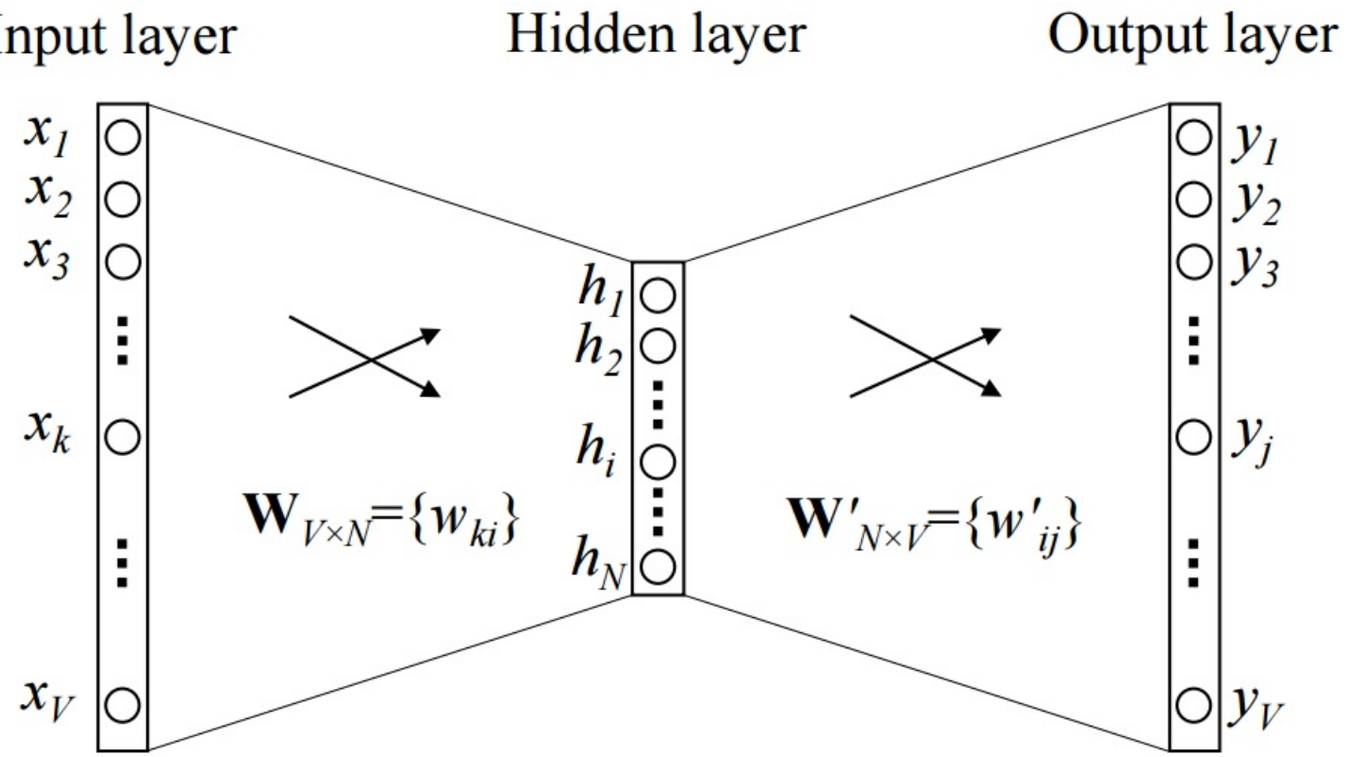


CBOW



Skip-gram

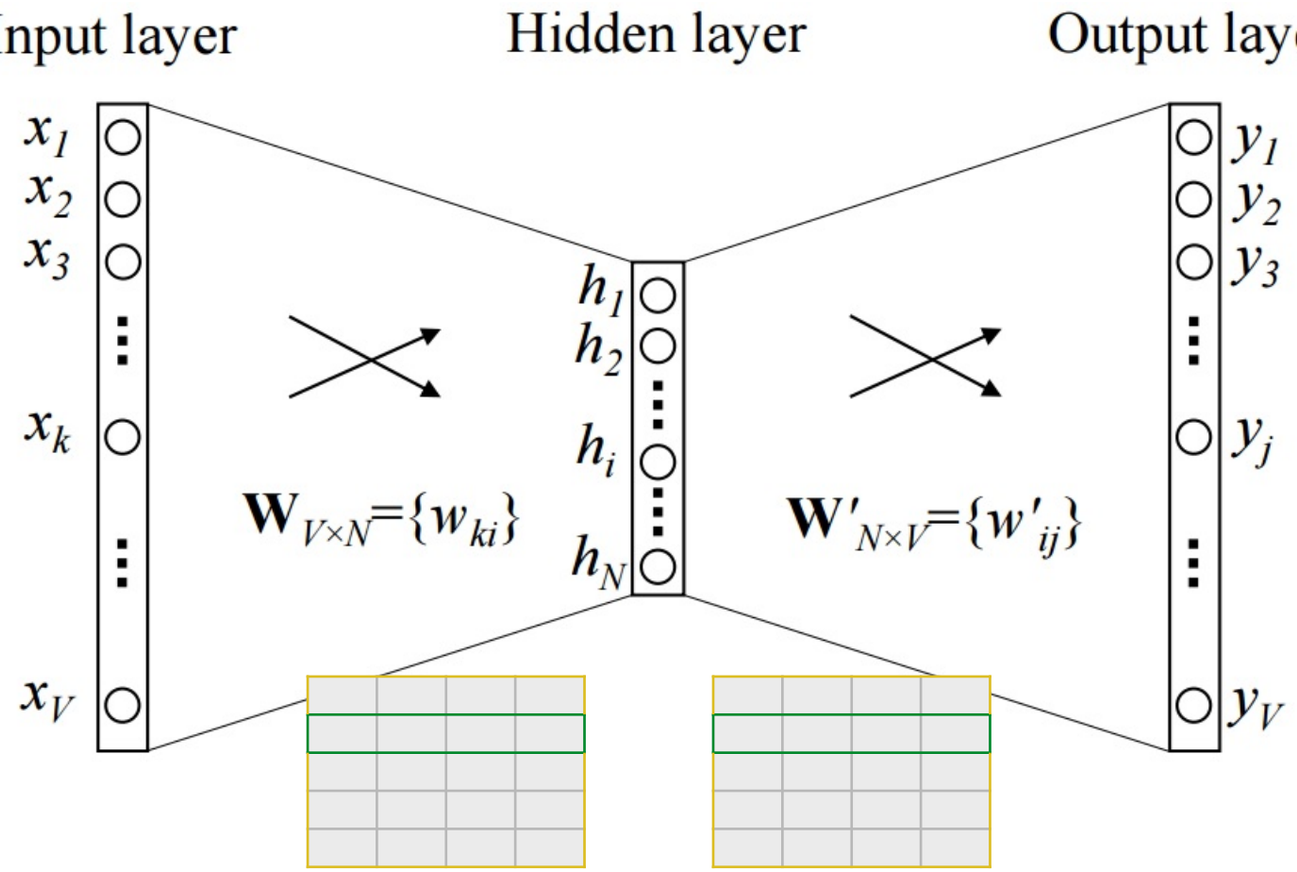
CBOW Model



Credit: Xin Rong

- Consider that we have a window length of one, or c = 1
- Input and Output dimensions are or size V (vocabulary), hidden layer N << V
- central question: how do we learn the parameters of this model (weights) given training examples from text

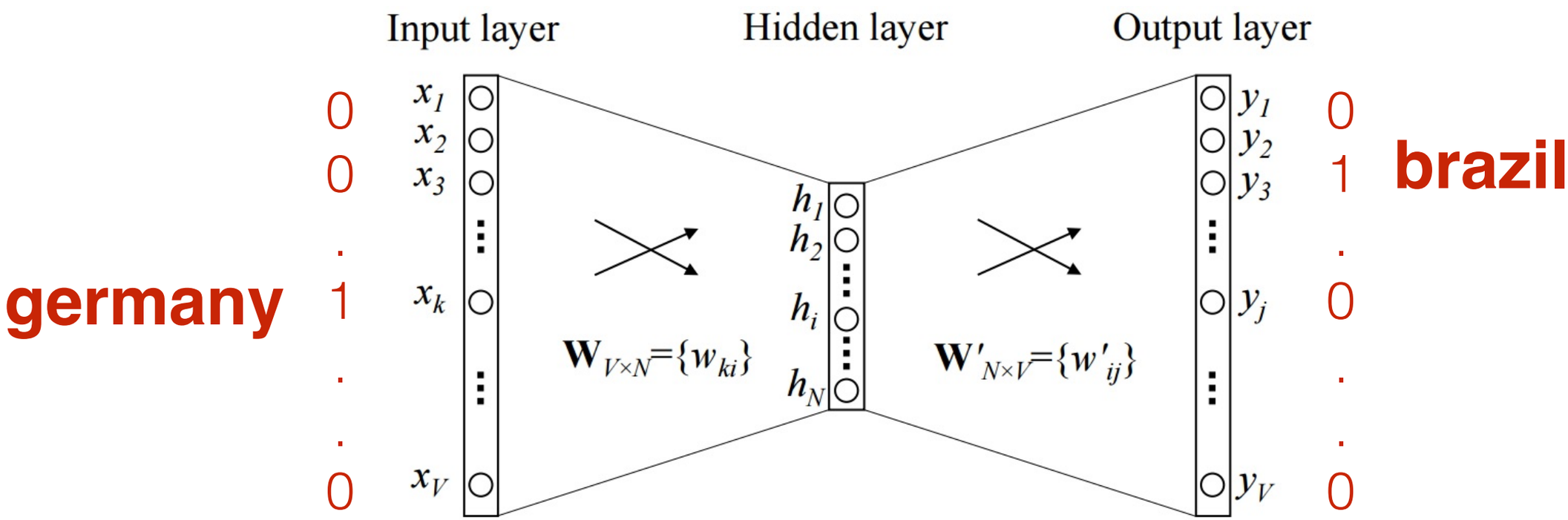
CBOW Model



Credit: Xin Rong

- Consider that we have a window length of one, or c = 1
- Input and Output dimensions are or size V (vocabulary), hidden layer N << V
- central question: how do we learn the parameters of this model (weights) given training examples from text

CBOW Model



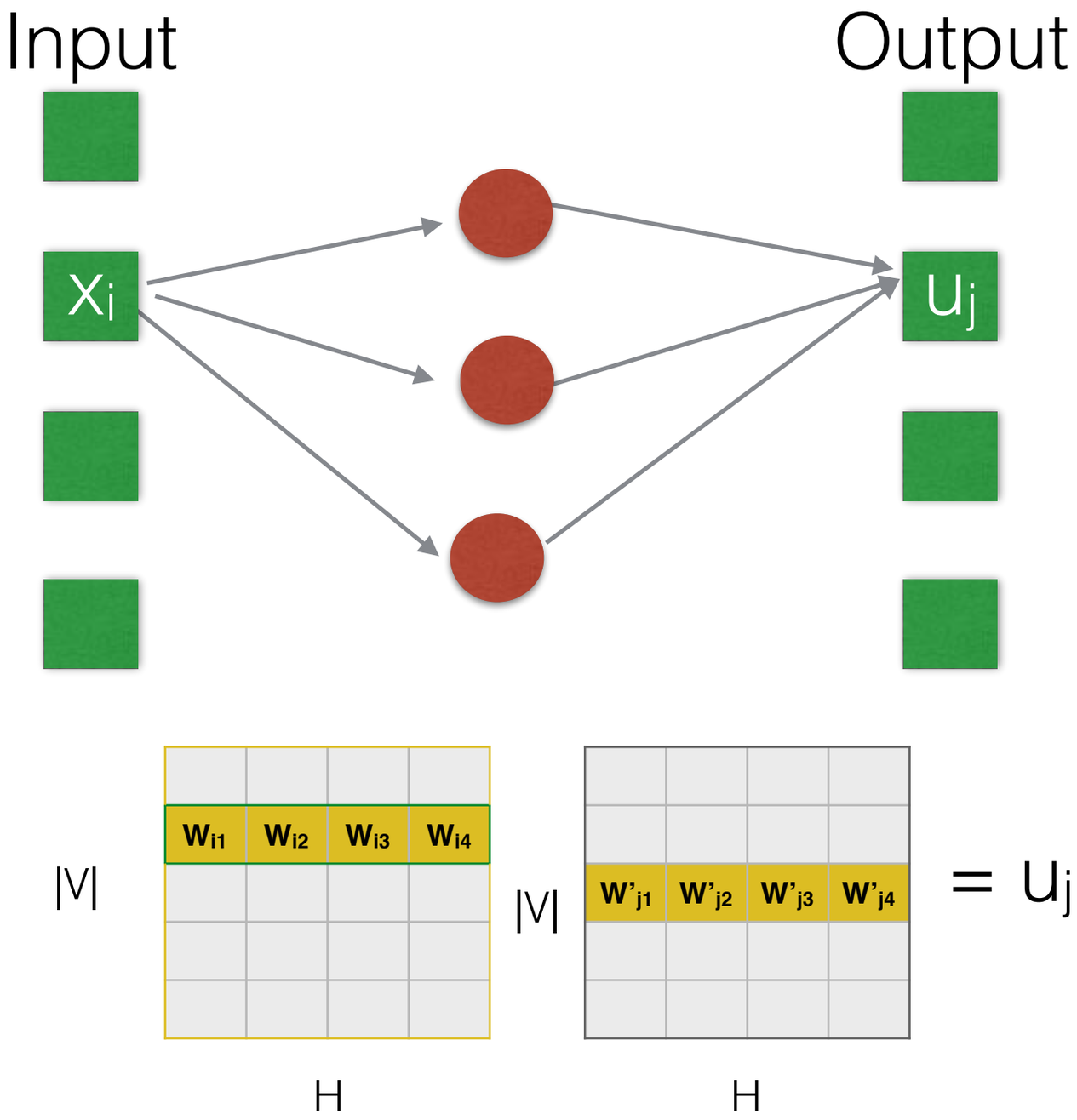
Credit: Xin Rong

$\mathbf{h} = \mathbf{x}^T W$ is the k-th row of W, $W_{k:}$.

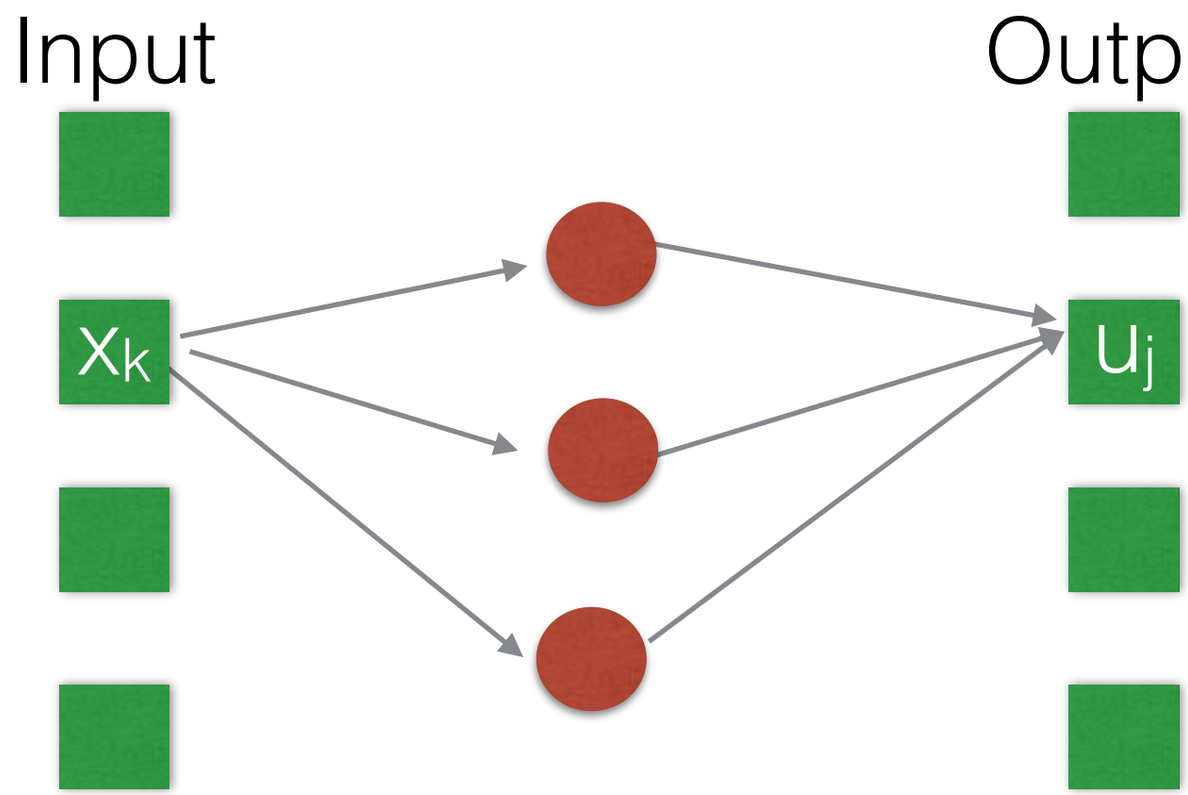
- h is the **input representation** in the hidden space
- **Output representation** : h.W'
- What is cosine distance between output and input representation ?

Word2Vec - Simple Training

- Cosine similarity at each output node is the measure of similarity w.r.t the given configuration of weights (prarams)
- Loss = comparison between the expected output and current *output distributions*
- Construct a distribution (softmax function)
- Use cross-entropy for loss



Training using back propagation

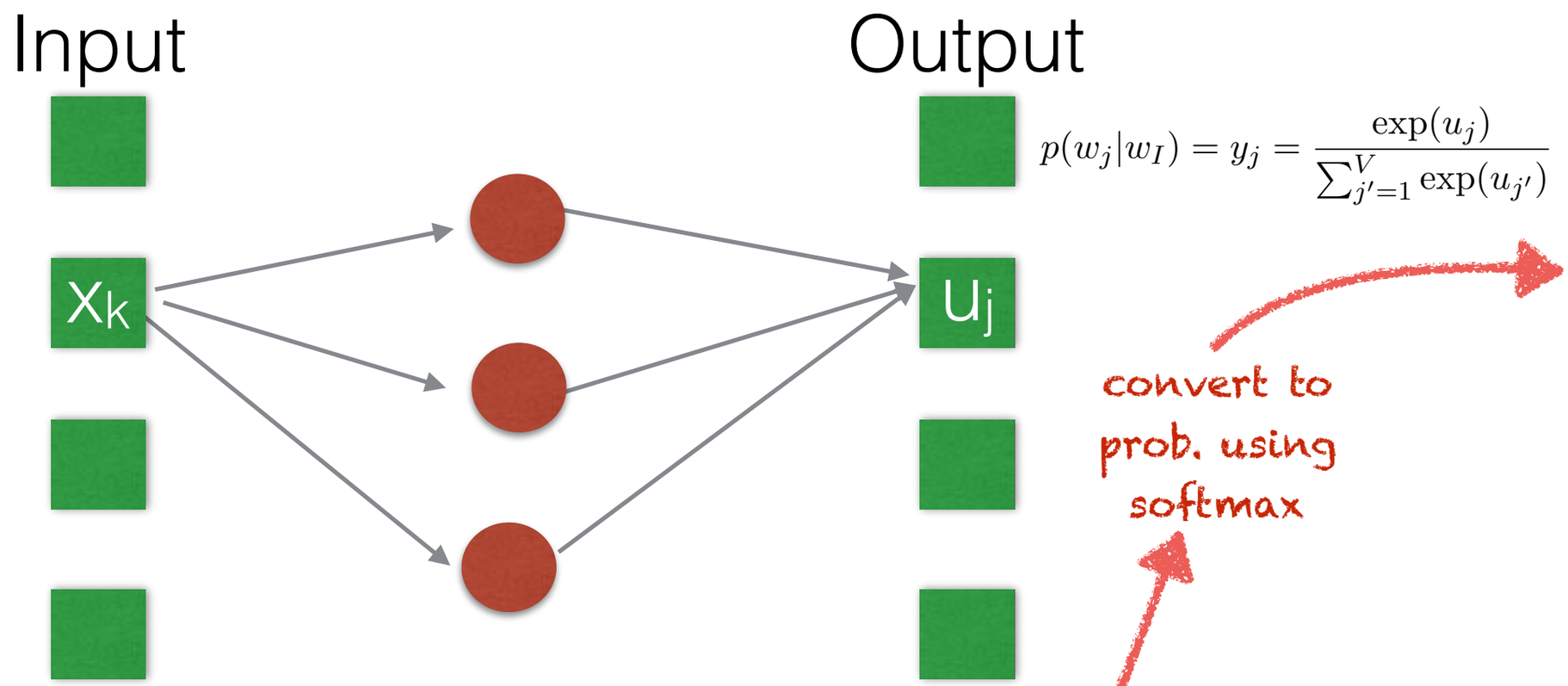


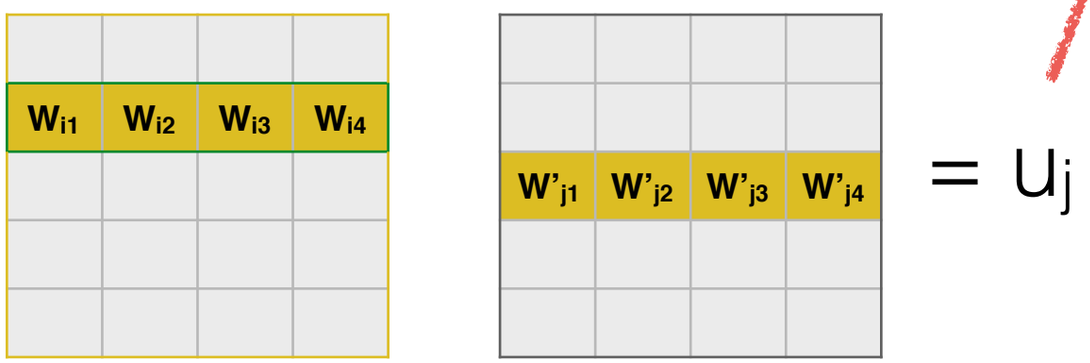
W_{i1}	W_{i2}	W_{i3}	W_{i4}

W'_{j1}	W'_{j2}	W'_{j3}	W'_{j4}

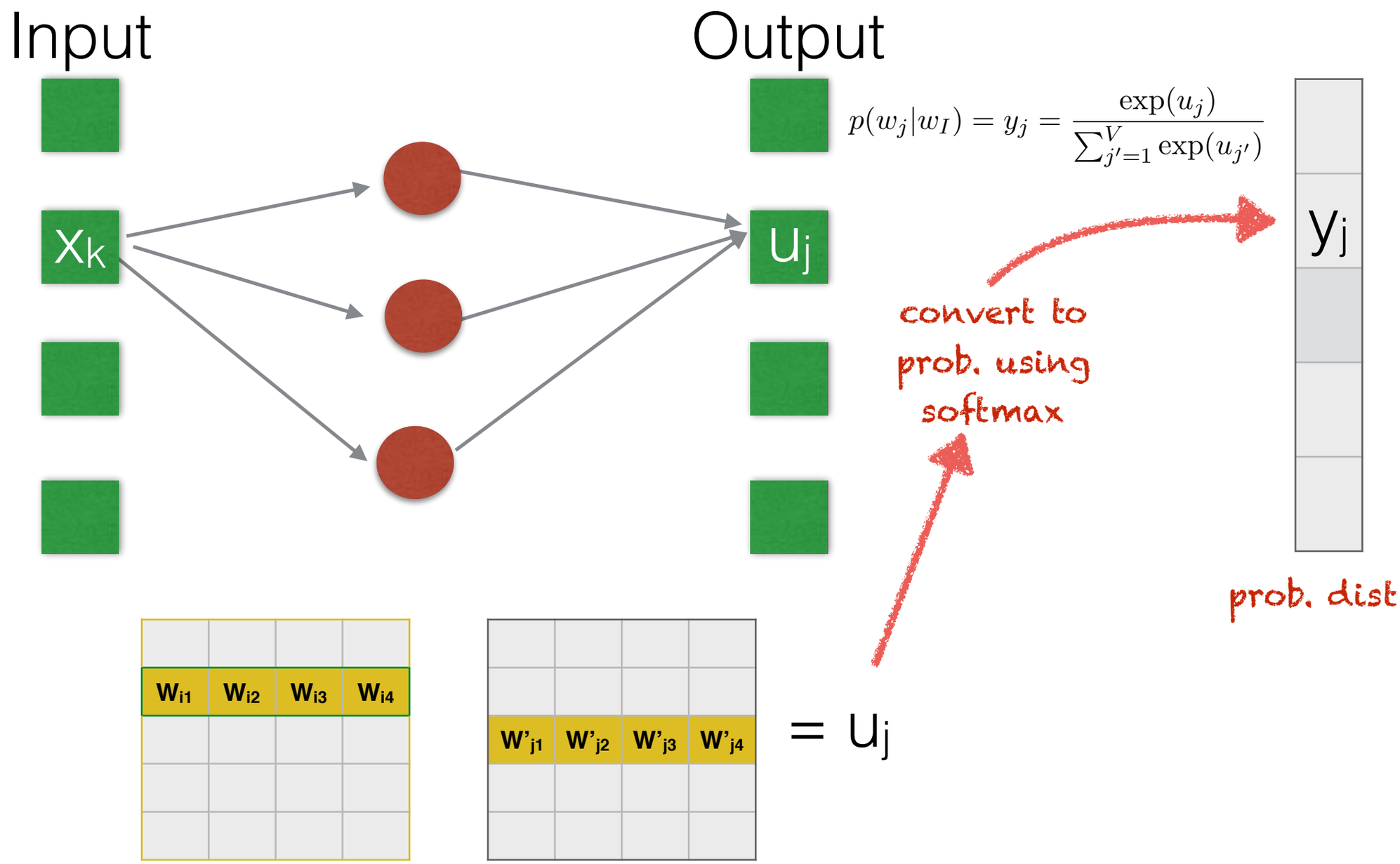
$= u_j$

Training using back propagation

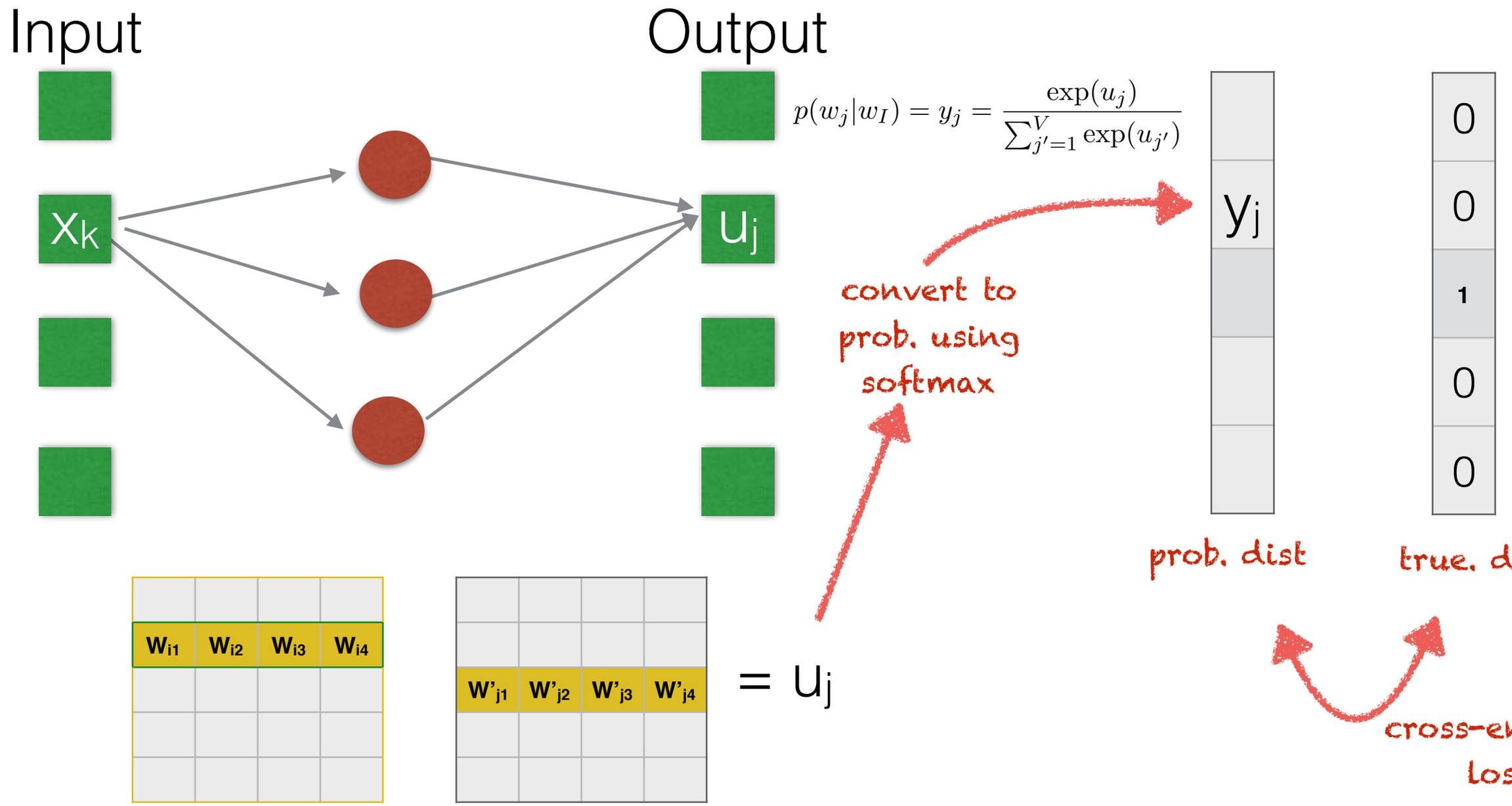




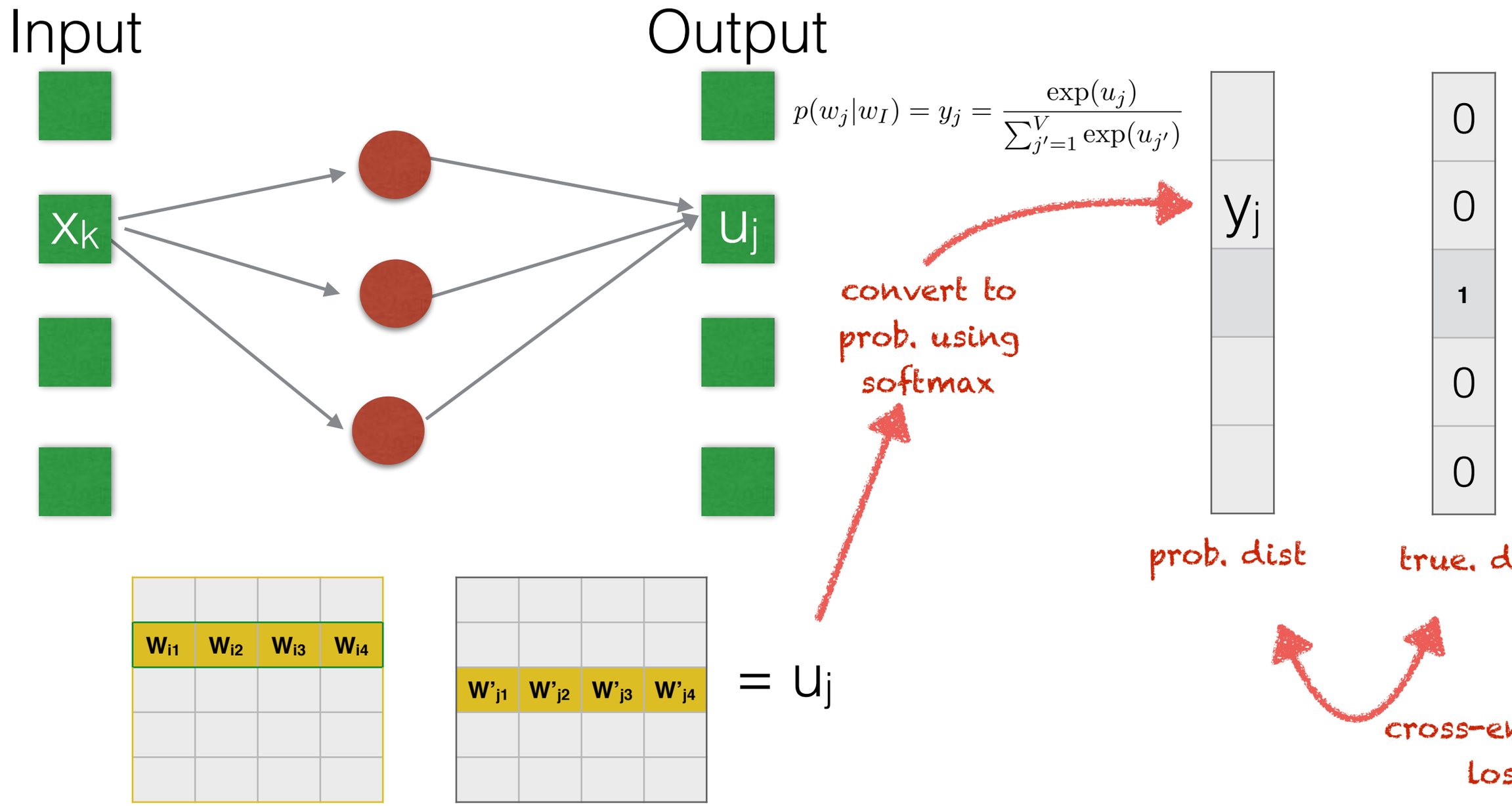
Training using back propagation



Training using back propagation



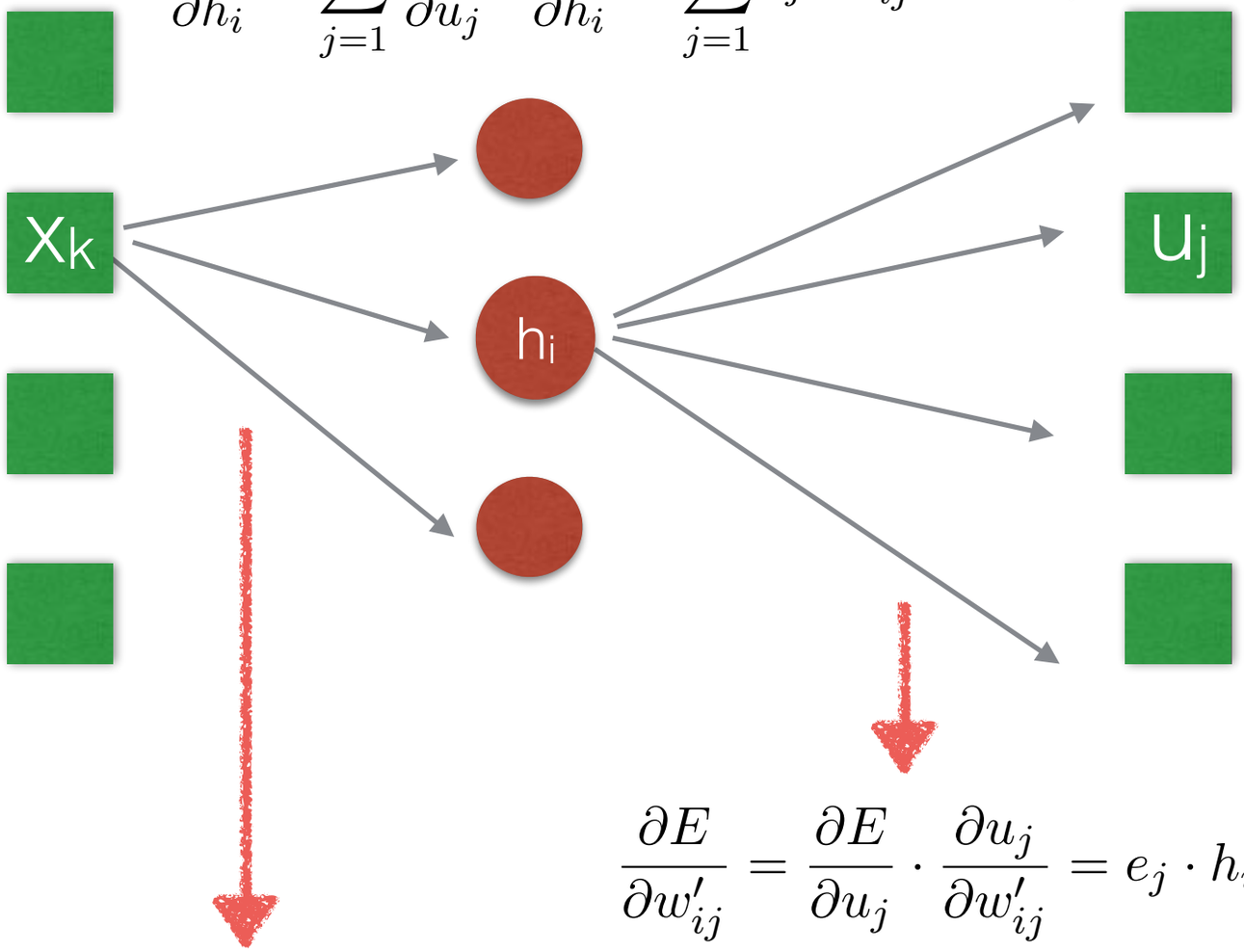
Training using back propagation



Choose weights W and W' s.t. cross entropy loss is minimised

Training using back propagation

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{V} \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial h_i} = \sum_{j=1}^{V} e_j \cdot w'_{ij} := \mathrm{EH}_i$$



$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial w'_{ij}} = e_j \cdot h_i$$

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial h_i} \cdot \frac{\partial h_i}{\partial w_{ki}} = \mathrm{EH}_i \cdot x_k$$

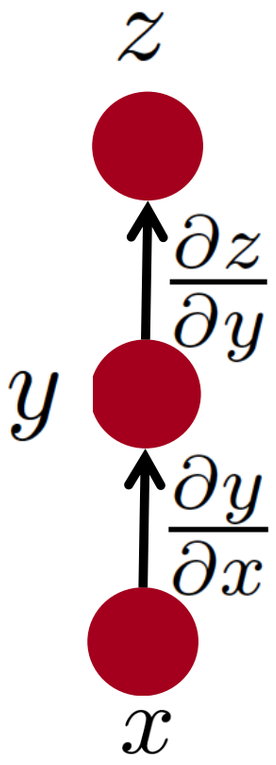
E

$$p(w_j|w_I) = y_j = \frac{\exp(u_j)}{\sum_{j'=1}^{V} \exp(u_{j'})}$$

$$\frac{\partial E}{\partial u_j} = y_j - t_j := e_j$$

Basics: Chain rule

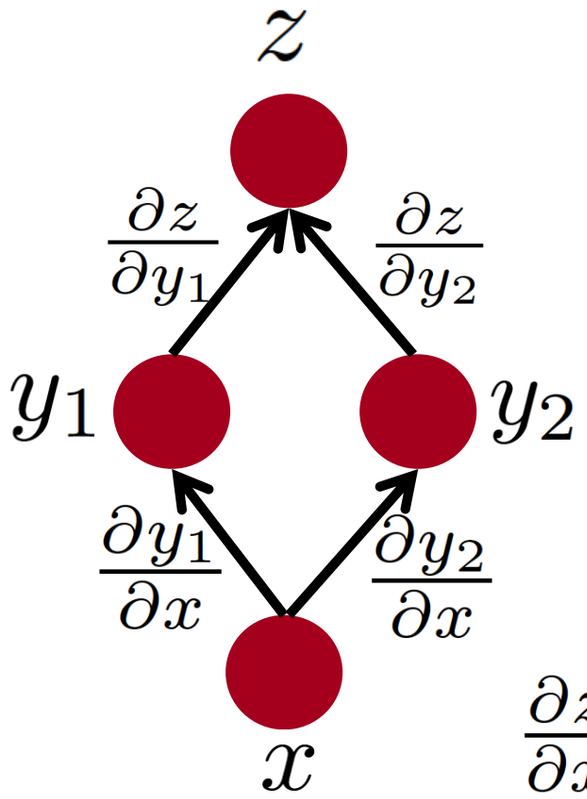
z = f(g(x))



$$\Delta z = \frac{\partial z}{\partial y}\Delta y$$
$$\Delta y = \frac{\partial y}{\partial x}\Delta x$$
$$\Delta z = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\Delta x$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

z = f(g1(x) + g2(x))



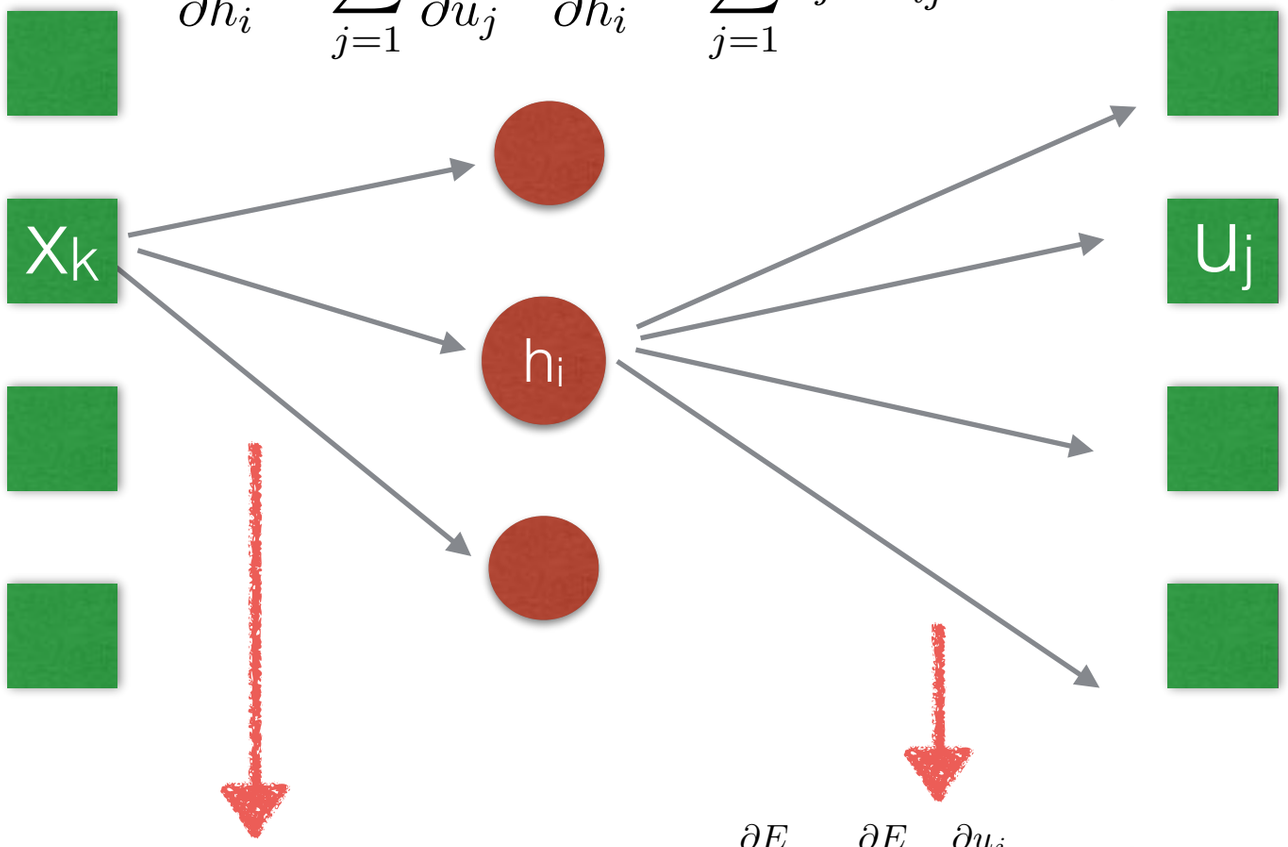
multi-path chain rule

$$\frac{\partial z}{\partial x}$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1}\frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2}\frac{\partial y_2}{\partial x}$$

Training : Update Equations

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{V} \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial h_i} = \sum_{j=1}^{V} e_j \cdot w'_{ij} := \mathrm{EH}_i$$



$$\frac{\partial E}{\partial u_j} = y_j - t_j := e_j$$

E

$$p(w_j|w_I) = y_j = \frac{\exp(u_j)}{\sum_{j'=1}^{V} \exp(u_{j'})}$$

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial h_i} \cdot \frac{\partial h_i}{\partial w_{ki}} = \mathrm{EH}_i \cdot x_k$$

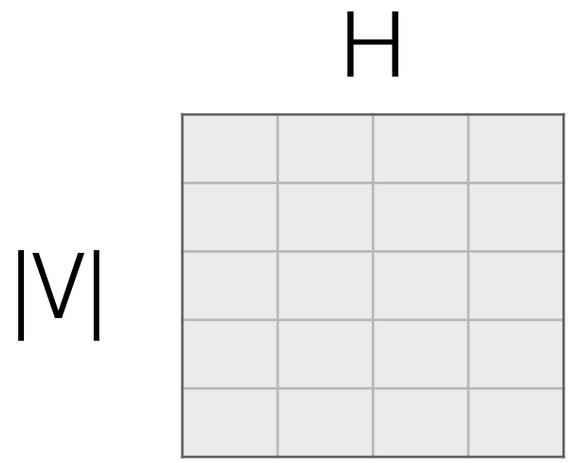
$$\mathbf{v}_{w_I}^{(\mathrm{new})} = \mathbf{v}_{w_I}^{(\mathrm{old})} - \eta \mathrm{EH}^T$$

$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial w'_{ij}} = e_j \cdot h_i$$

$$\mathbf{v}'^{(\mathrm{new})}_{w_j} = \mathbf{v}'^{(\mathrm{old})}_{w_j} - \eta \cdot e_j \cdot \mathbf{h}$$

Limitations

- The weight matrices are the final word embeddings
- Typically they are averaged for the final vector representation of words in the hidden dimensions
- Computational complexity of word2vec ?
  - Each word vector is updated
  - Millions of words to be updated for each input
  - Large training time



$$\frac{\exp\left({\mathbf{v}'_{w_j}}^T \mathbf{v}_{w_I}\right)}{\sum_{j'=1}^{V} \exp\left({\mathbf{v}'_{w_{j'}}}^T \mathbf{v}_{w_I}\right)}$$

the bottleneck

Negative Sampling

$$\frac{\exp\left({\mathbf{v}'_{w_j}}^T \mathbf{v}_{w_I}\right)}{\sum_{w_{j'} \in W_{neg}} \exp\left({\mathbf{v}'_{w_{j'}}}^T \mathbf{v}_{w_I}\right)}$$

new obj.

- For a current state of the parameters computing the denominator is expensive (V is typically large)
- Sample limited set of negative words for updating
- Update equation only word words in the context and the negative samples

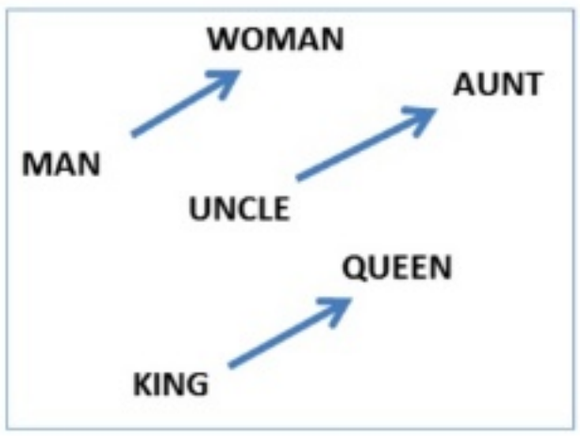
TASK - 30 points = explain the Levy, Goldberg explanation of Negative sampling

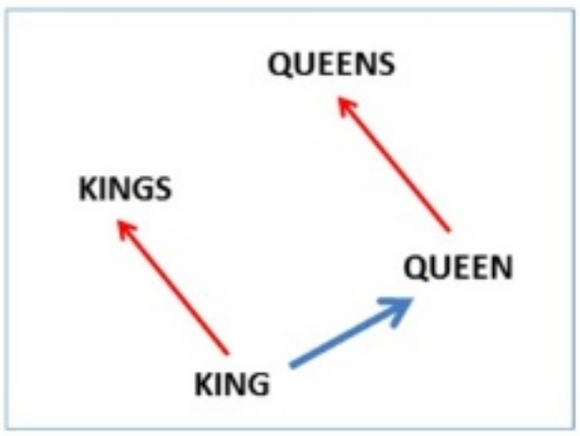
Goldberg, Yoav, and Omer Levy. "word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method."

Time for some magic

- Word embeddings somehow also capture relationships
  - Word similarity as well as analogy
- Try it out :
  - JAVA: https://deeplearning4j.org/word2vec
  - Python: https://radimrehurek.com/gensim/models/word2vec.html
- Applications
  - Query expansions
  - Query suggestions

vec("man") – vec("king") + vec("woman") = vec("queen")





References

- Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." *Advances in neural information processing systems*. 2013.
- Rong, Xin. "word2vec parameter learning explained." *arXiv preprint arXiv: 1411.2738* (2014).
- Goldberg, Yoav, and Omer Levy. "word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method." *arXiv preprint arXiv: 1402.3722* (2014).