Machine Learning

Intro to NN

Dan Goldwasser

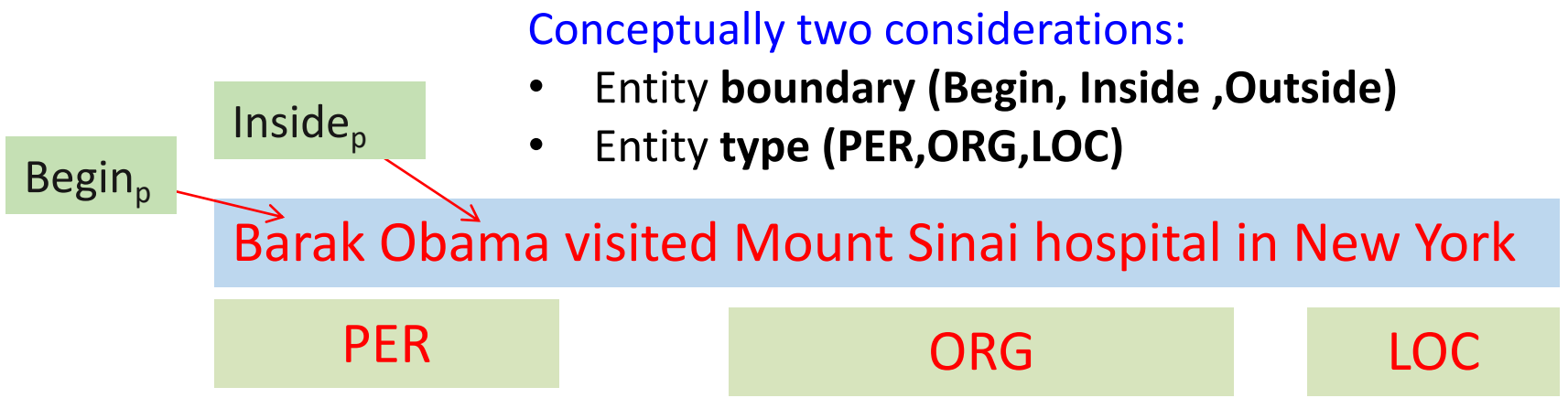
dgoldwas@purdue.edu

Goals for today

- *Neural networks introduction*
  - *What can and can't be learned by linear models?*
  - *NN intuition: engineering vs.* ***learning*** *a new representation for the data*
- *We will start with a basic intro to NN*
  - *Representation, training*
- *..and we'll move to NLP specific architectures*
- ***Big Questions:***
  - *How/whether to account for linguistic structure*
  - *How to model the interactions between words/sentences*
  - *How to do "neural reasoning" in complex problems?*

Dealing with Structures

- Structured prediction – dealing with multiple decisions at the same time. Modeling the interactions between decisions is the key challenge.



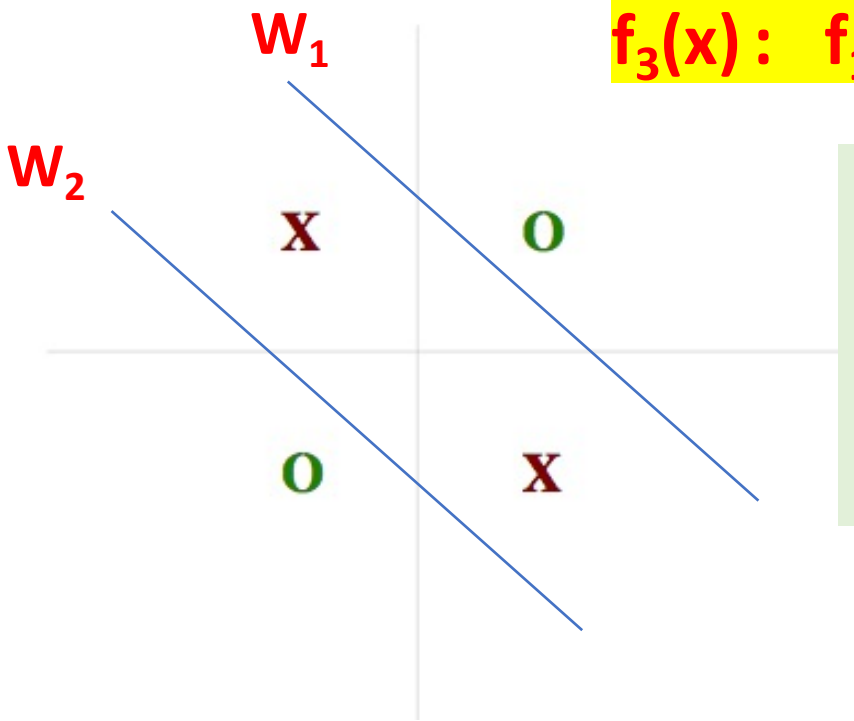
Many of the predictions are context dependent: e.g., Mount Sinai is a LOC, while Mount Sinai hospital is an ORG.

How can you capture it using the machinery we currently have? At what cost?

Linear vs. Non Linear Classification

- Up to this point we focus on linear classifiers.
- They depend on *manually finding expressive* features which define simple learning problems.
- Simple solutions often break.
  - BoW is hard to beat, and works great for simple problems
    - Harder cases: nuanced problems, domain differences
    - Tends to blowup the feature space.
- Non-Linear classifiers – complex decision boundaries
  - Decision trees, **neural-nets**,..
  - NN dynamically learn a feature representation

Limitations of Linear Models



$f_3(x)$: $f_1(x)$ OR $f_2(x)$

Can you find a way to combine linear models to solve this problem?
(i.e., chain their predictions)

Learn *a model for* ***representing*** *the data, that would simplify the problem,* **now build a simple classifier over it.**

Big idea behind neural net – *jointly learn the representation to make classification easier with the classification problem.*

Neural Network

- Simply put, NN's are functions **f: X→Y**
  - *f is a **non-linear** function*
  - *X is a **vector** of continuous or discrete variables*
  - *Y is a **vector** of continuous or discrete variables*
- **Very expressive classifier**
  - In fact, NN can be used to represent any function
- The function f is represented using a network of logistic units

Multi Layer Neural Networks

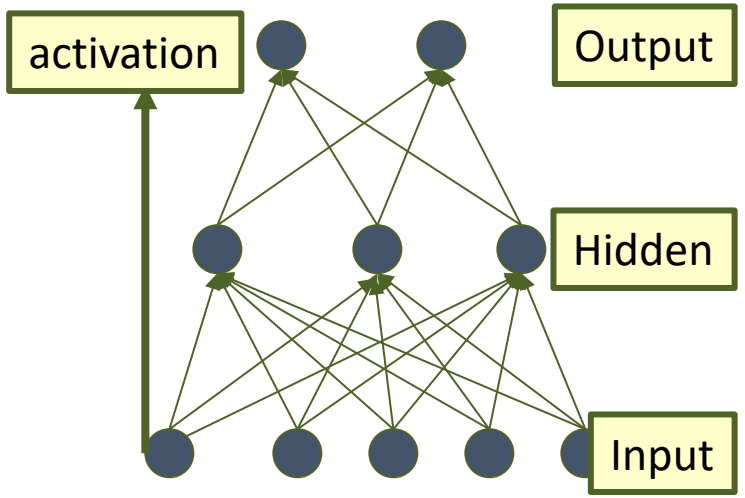
- Multi-layer network were designed to overcome the computational (**expressivity**) limitation of a single threshold element.

- The idea is to **stack** several

  layers of threshold elements,

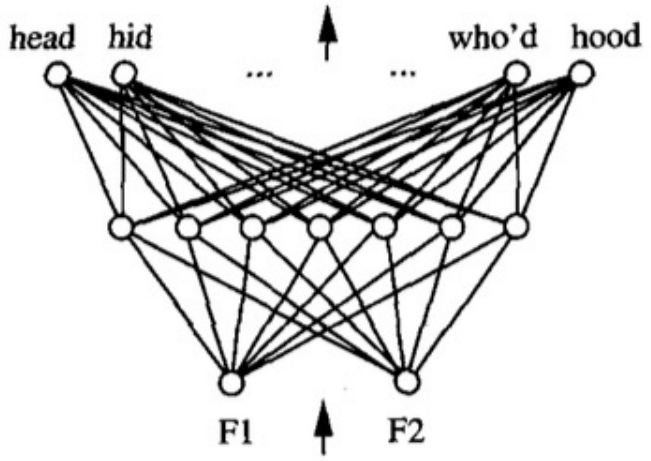
  each layer using the output of

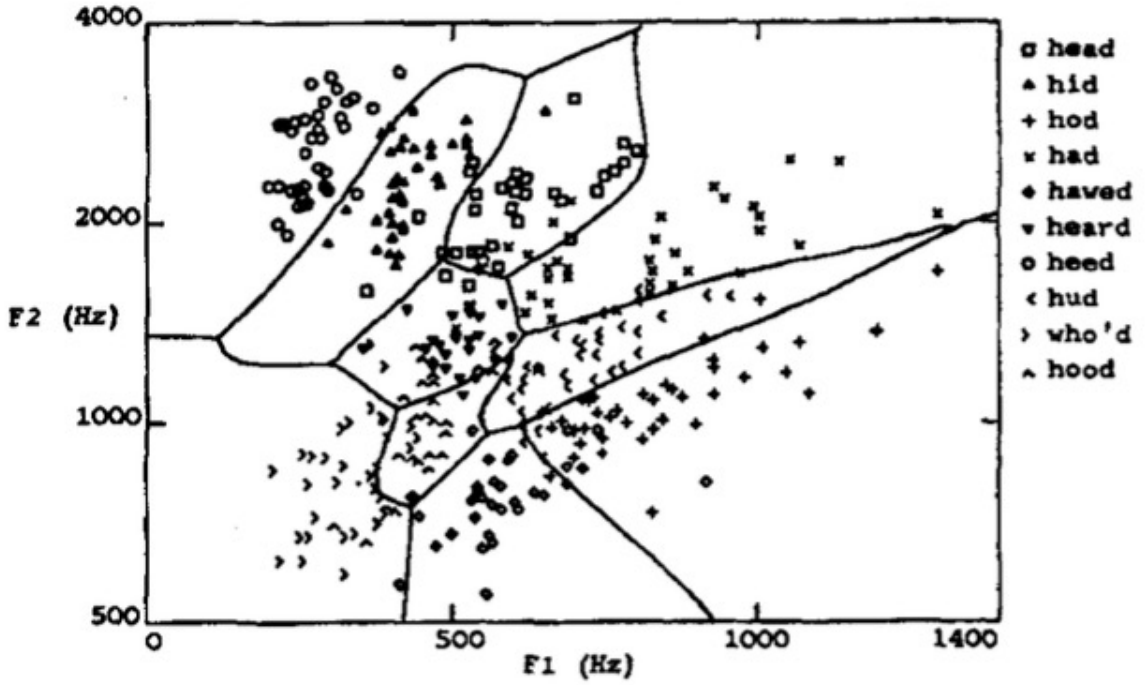
  the previous layer as input



Multi-layer networks **can represent arbitrary functions**, but building effective learning methods for such network was/is difficult.

Example: NN for speech vowel recognition





ALVINN: autonomous land vehicle in a NN

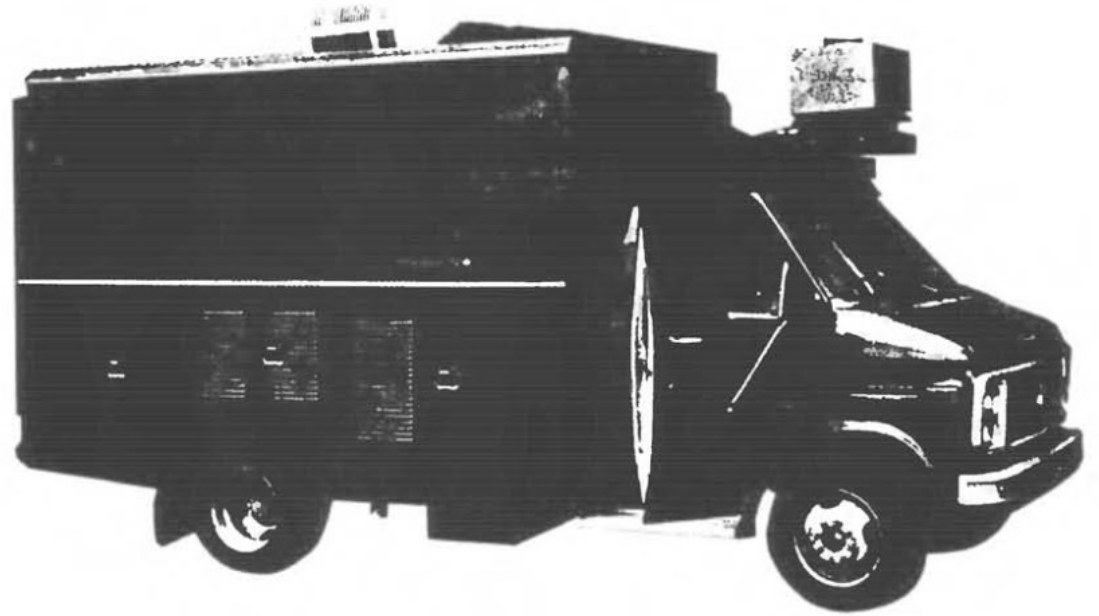
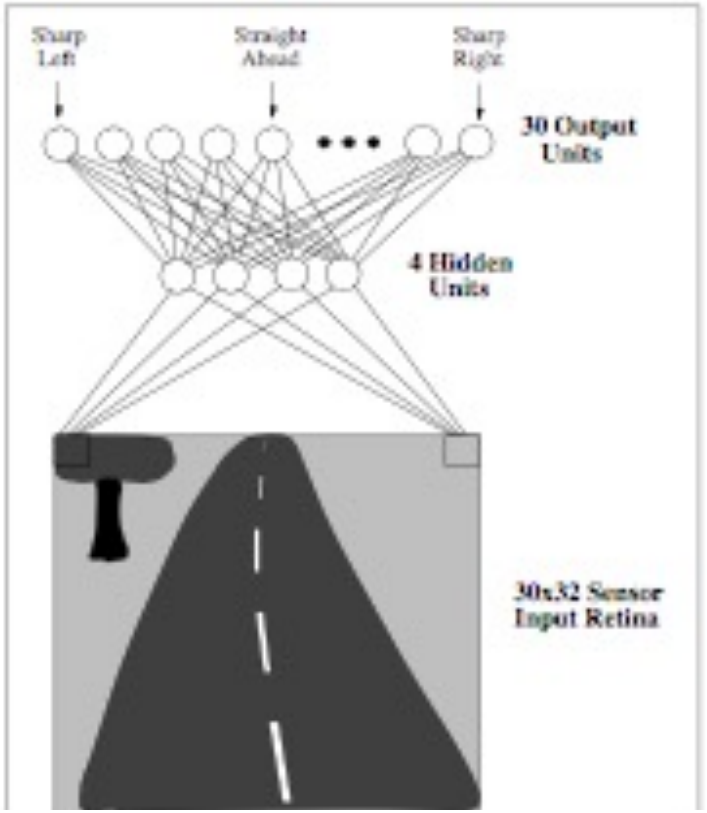
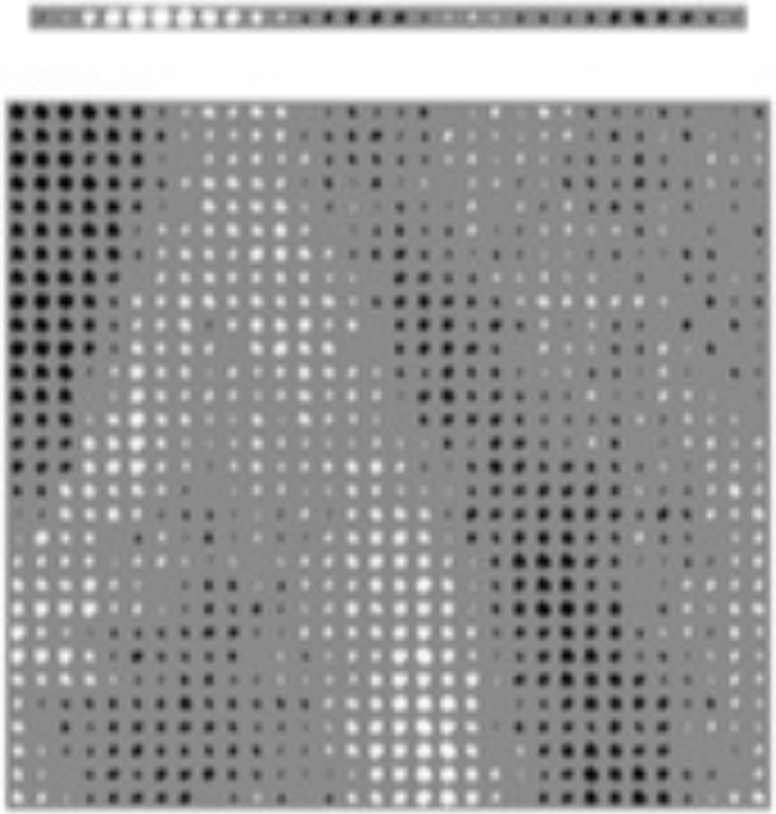


Figure 3: NAVLAB, the CMU autonomous navigation test vehicle.

Pomerleau ‘89

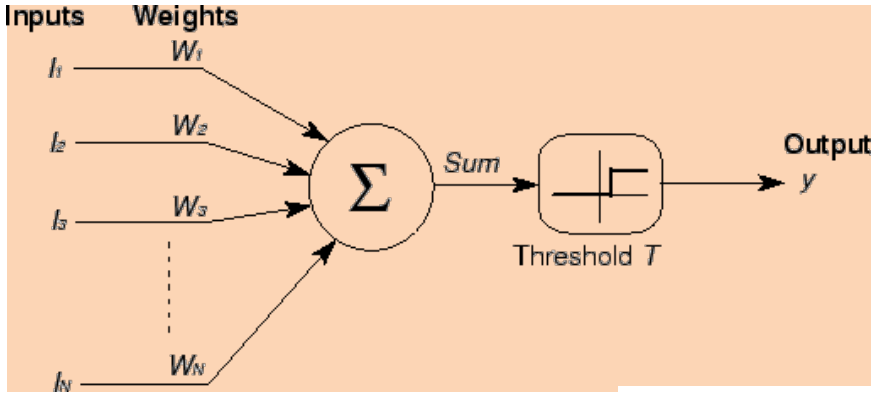
ALVINN: autonomous land vehicle in a NN

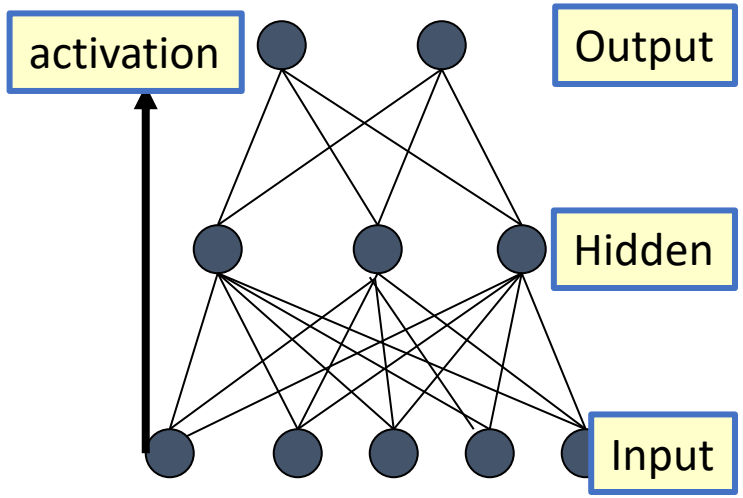




Basic Units in Multi-Layer NN

- Basic element: **linear unit**
  - But, we would like to represent nonlinear functions
    - Multiple layers of linear functions are still linear functions
  - Threshold units are not smooth (we would like to use gradient-based algorithms)



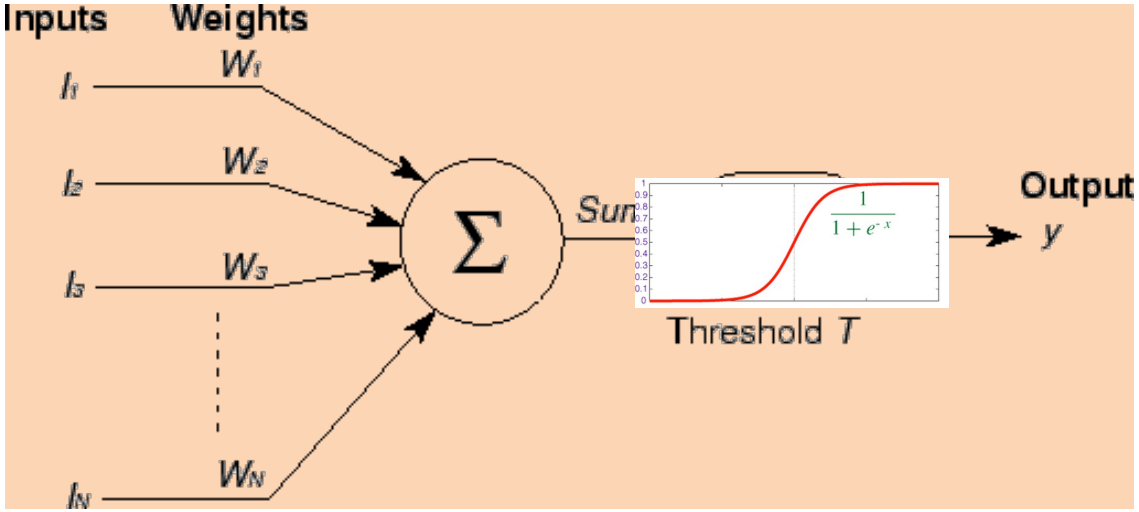


Basic Units in Multi-Layer NN

- Basic element: **sigmoid unit**
  - Input to a unit *j* is defined as: $\Sigma w_{ij} x_i$
  - Output is defined as : $\sigma (\Sigma w_{ij} x_i)$
    - σ is simply the logistic function:

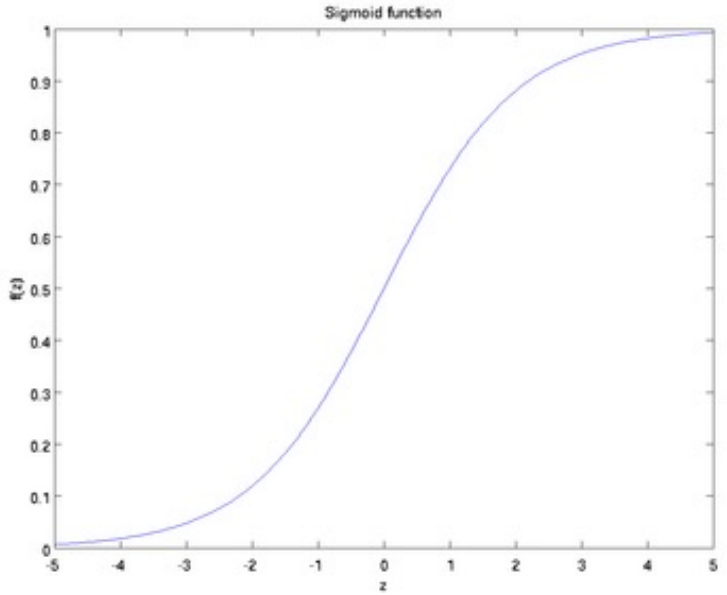
$$\frac{1}{1+e^{-x}}$$

Note: similar to previous algorithms, We encode the bias/threshold, as a "fake" Feature that is always active

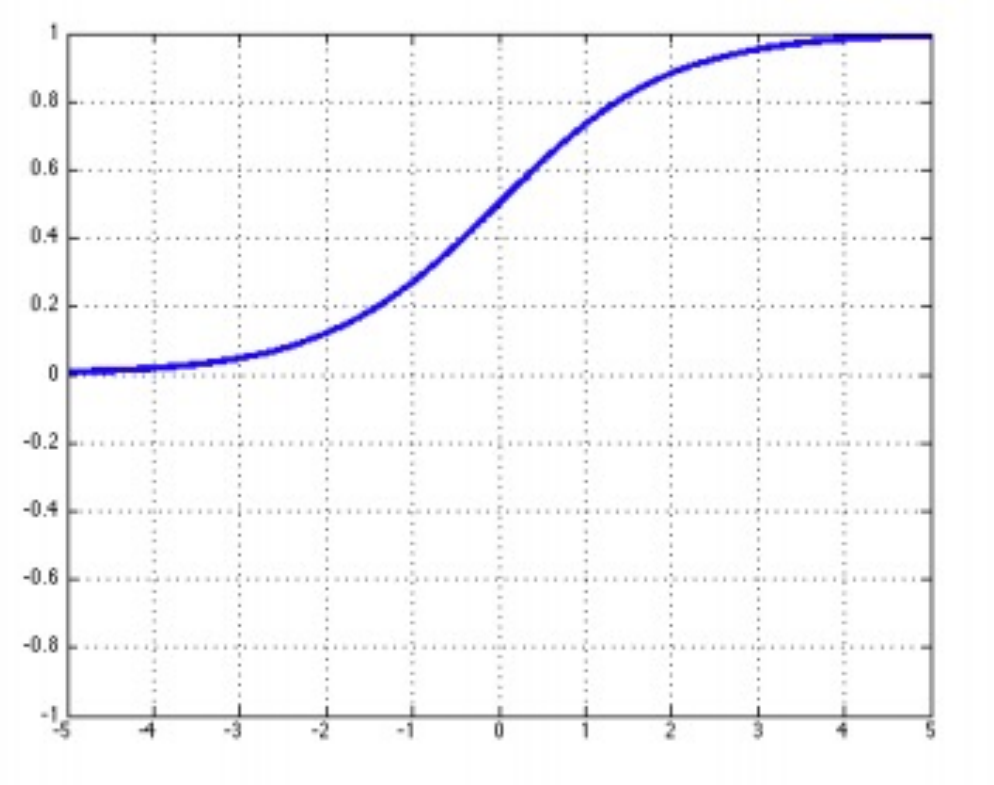


Basic Units in Multi-Layer NN

- Basic element: **sigmoid unit**
  - You can also replace the logistic function with other smooth activation functions



Sigmoid

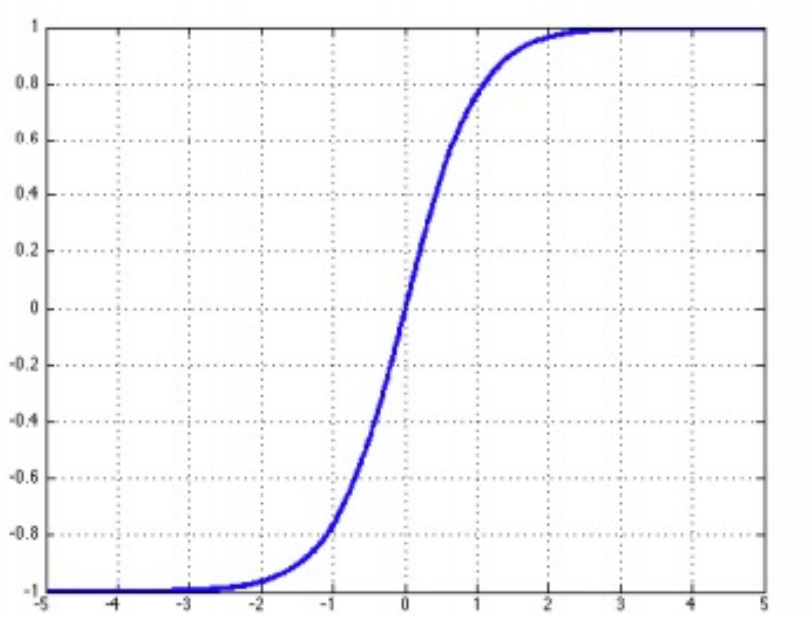


$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

Images taken from web.stanford.edu/class/cs224n

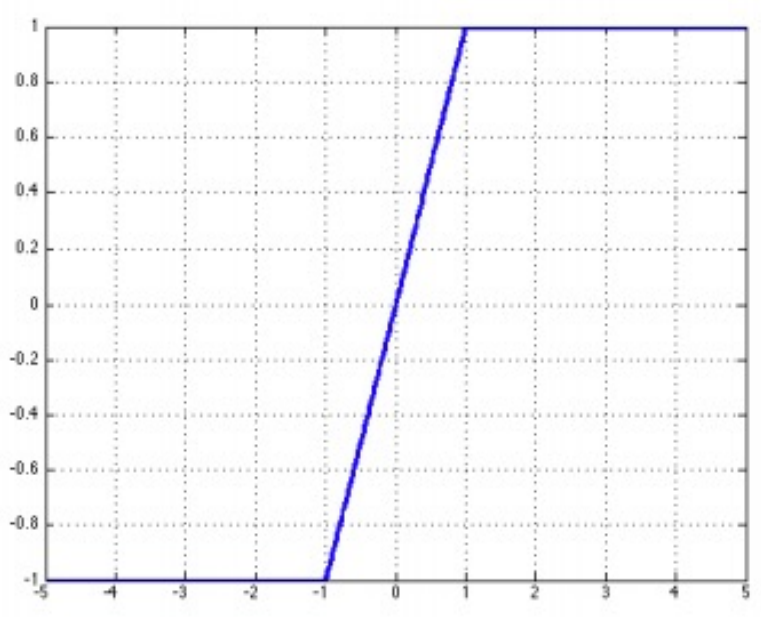
Tanh

- sd



$$\tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)} = 2\sigma(2z) - 1$$

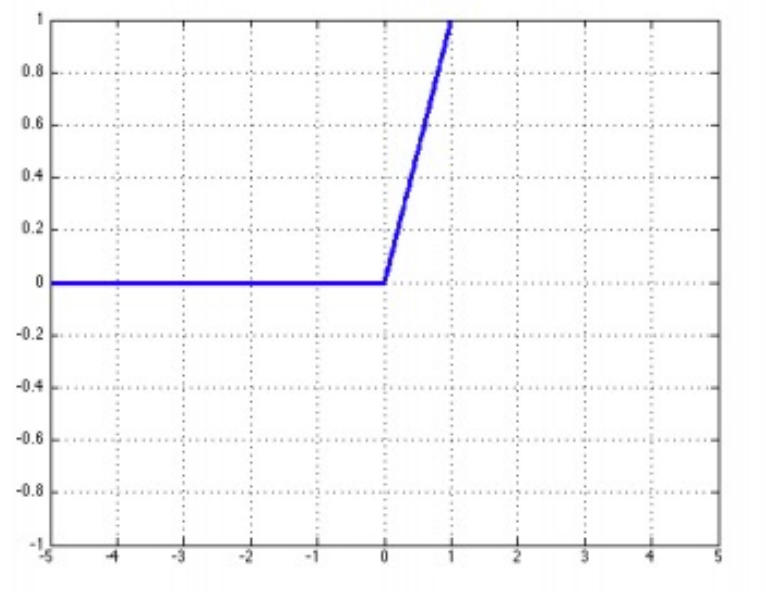
where $\tanh(z) \in (-1, 1)$



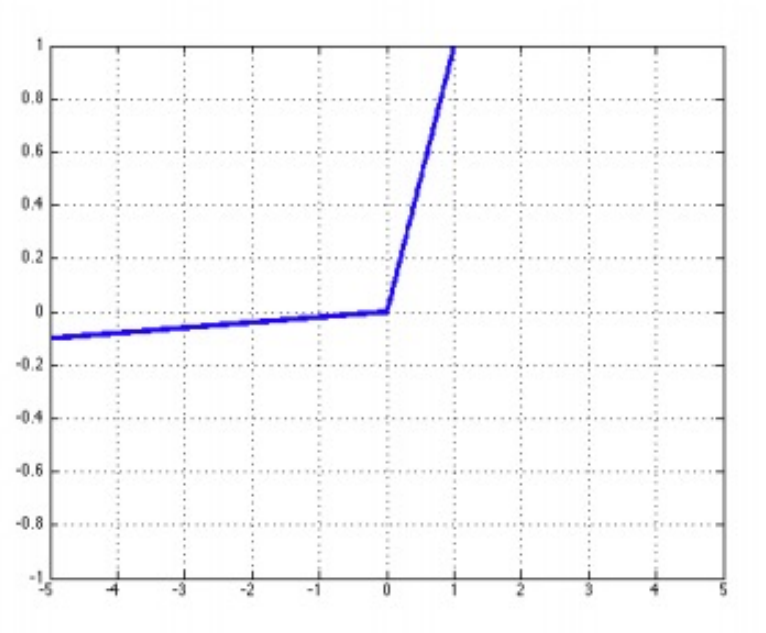
$$\text{hardtanh}(z) = \begin{cases} -1 & : z < -1 \\ z & : -1 \leq z \leq 1 \\ 1 & : z > 1 \end{cases}$$

Images taken from web.stanford.edu/class/cs224n

Rectified Linear Units (RelU)



$$\text{rect}(z) = \max(z, 0)$$



$$\text{leaky}(z) = \max(z, k \cdot z)$$

$$\text{where } 0 < k < 1$$

Images taken from web.stanford.edu/class/cs224n

Multi Layer NN

- Another approach for increasing expressivity:

  Stacking multiple units to form a network

- Compute the output of the network using a ‘feed-forward’ computation
- Learn the parameters of the network using the backpropagation algorithm
- Any Boolean function can be represented using a two layer network
- Any bounded continuous function can be approximated using a two layer network

Multi Layer NN: forward computation

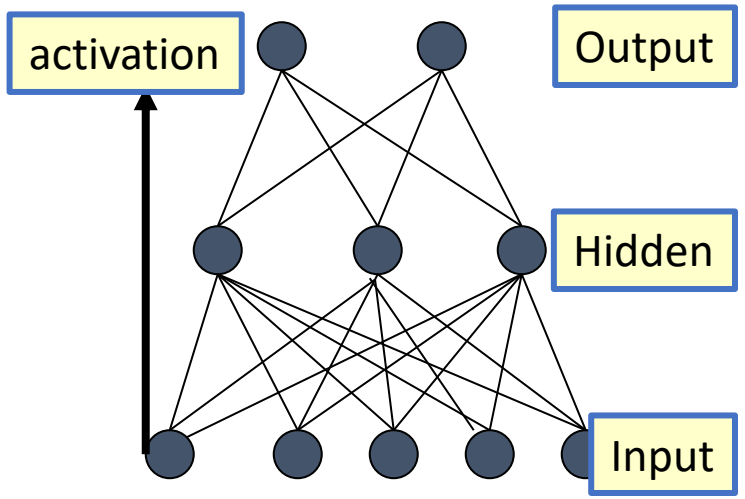
- Observe an input vector x
- *Push x through the network:*
  - For each **hidden unit** compute the activation value

$$h_j = \sigma(t_j) = \sigma(\sum_i w_{ji}^1 \, x_i)$$

  - For each **output value**, compute the activation value coming from the hidden units
  - **Prediction:**

$$\hat{y}_k = \sigma(s_k) = \sigma(\sum_j w_{kj}^2 h_j)$$

    - **Categories**: winner take all
    - **Vector**: take all output values
    - **Binary outputs**: Round to nearest 0-1 value



Let’s build our own NN!

- For a given problem, we have to decide:
  - How many input units?
  - How many hidden layers?
  - How many output units?

- What are the considerations we should have when deciding the number of hidden units?
  - Is there a “right” number?

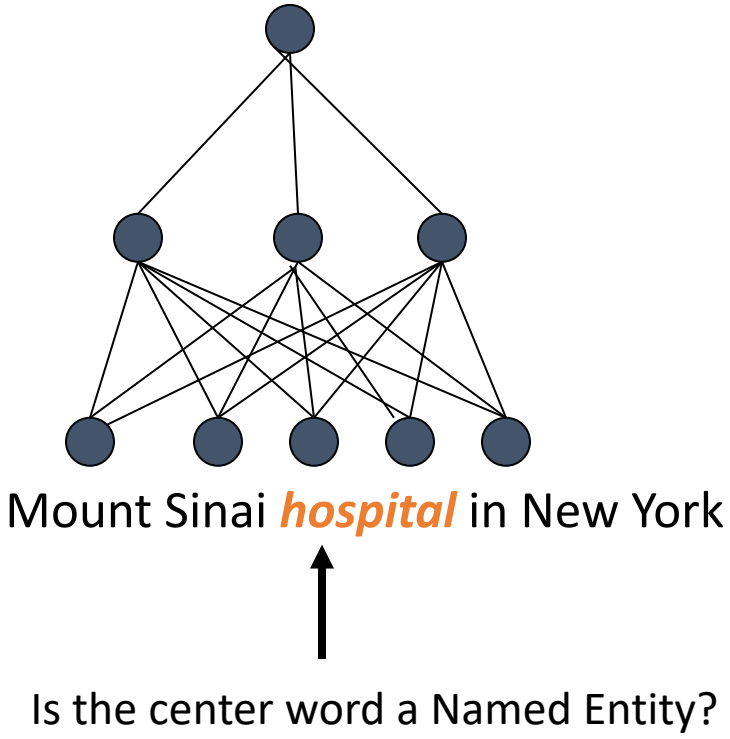
Let’s build our own NN!

- Let’s revisit a familiar problem, and design a NN
  - **Named Entity Recognition**
    - **Binary case**: given a sentence, decide which words are NE and which are not
    - **Multi-class case**: decide if a word is a Loc, Per, Org, None

- **What is the right architecture?**

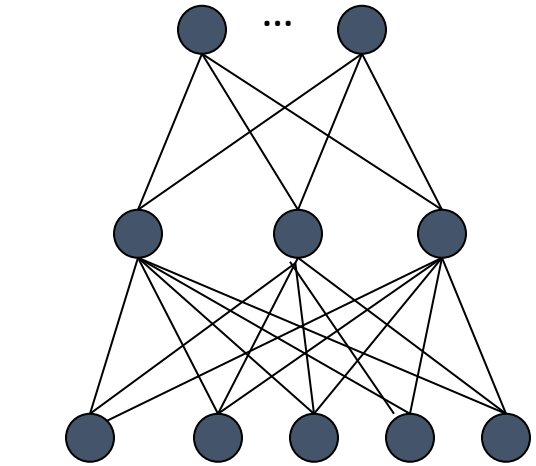
NN for NER

Binary case: Single output unit, can be interpreted as a threshold function



NN for NER

Multiclass case: winner-takes-all

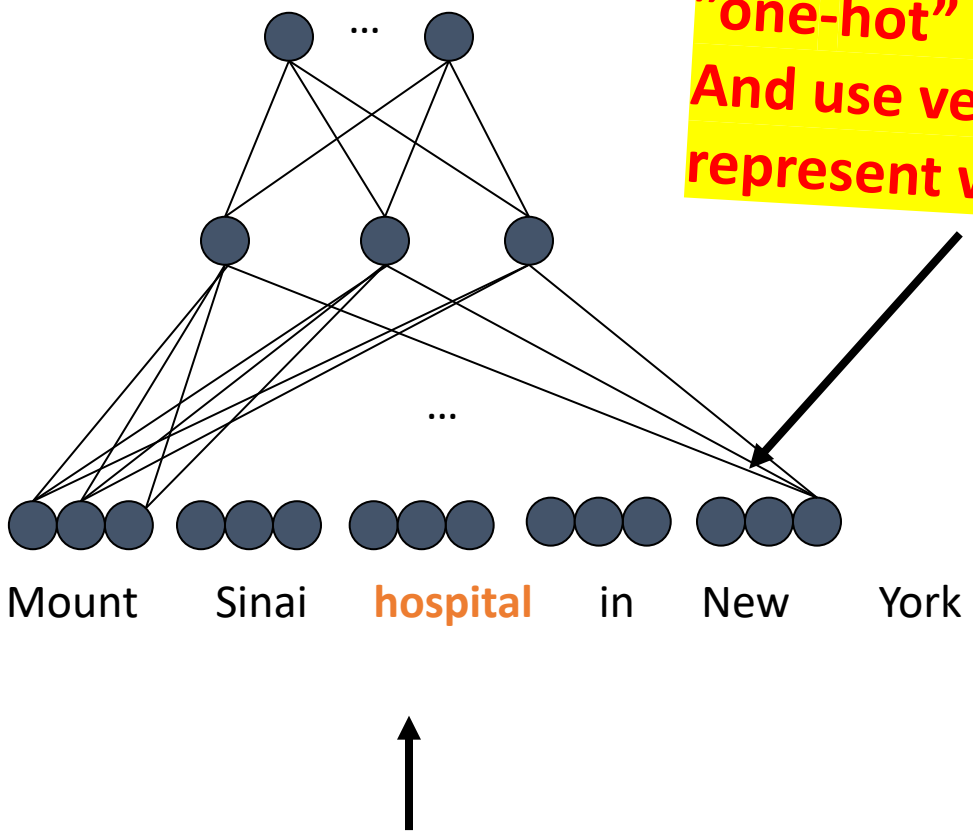


Mount Sinai ***hospital*** in New York

Is the center word a Named Entity?

NN for NER

Multiclass case: winner-takes-all



We can also replace the "one-hot" representation, And use vectors to represent words!

Word Vectors (short version)

- **BoW representation is also known as 1-hot**
  - Points to the fact that it is an incredibly sparse representation.
- **Dense vectors are an alternative** – each word is represented in a continuous space, using dense (i.e., not sparse) vectors
  - Can help reduce the feature space
- **Where do these vectors come from?**

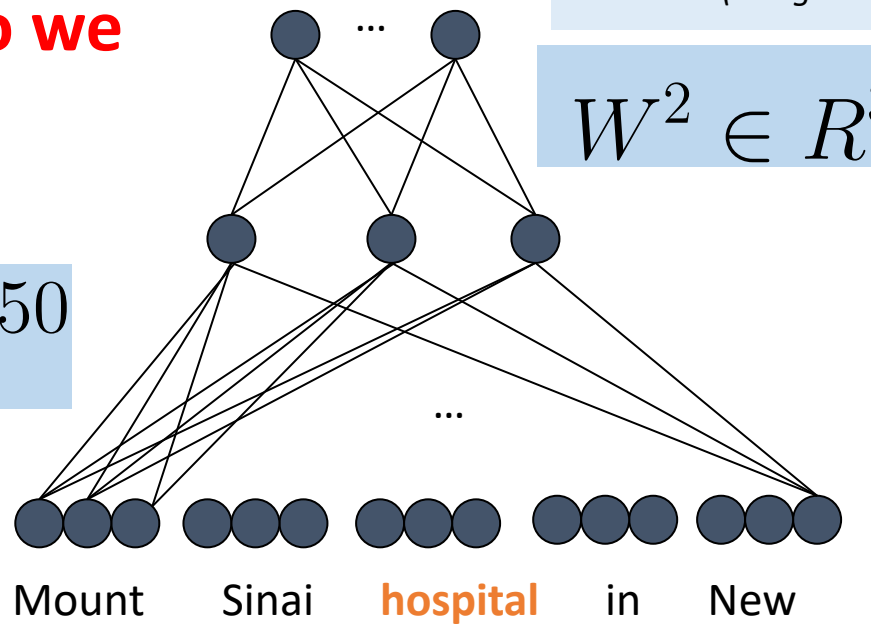
NN for NER

How many parameters do we have to train?

$R^{3 X 7}$ *(+ segmentation)*

$W^2 \in R^{3\times 4}$

$W^1 \in R^{3\times 50}$



Let's assume our word embedding are in R^{50}

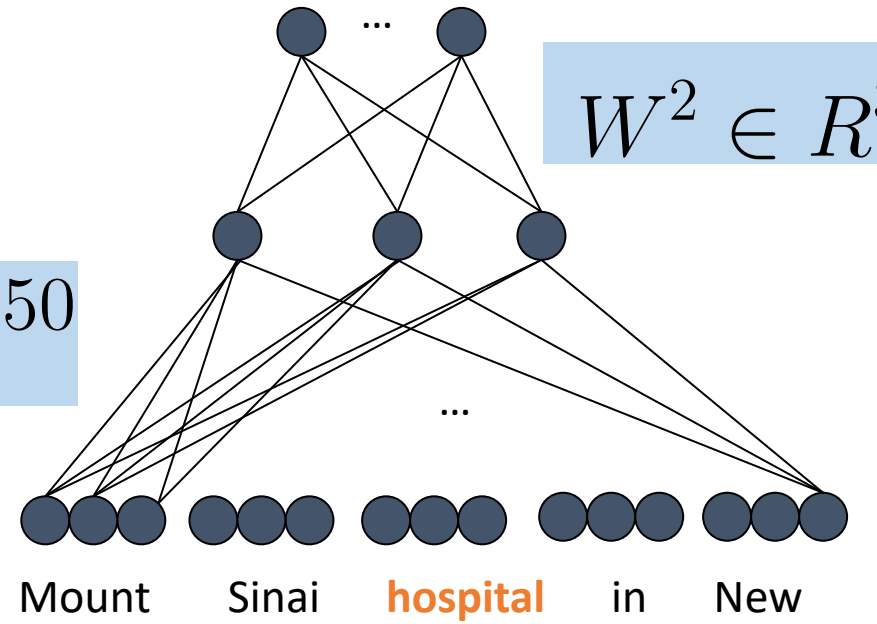
$x \in R^{50}$

Question: *How would this architecture change if we wanted to add NE segmentation as well?*

NN for NER

$W^2 \in R^{3\times 4}$

$W^1 \in R^{3\times 50}$



$x \in R^{50}$

Forward computation:

$$h_j = \sigma(t_j) = \sigma(\sum_i w^1_{ji}\, x_i)$$

Hidden layer

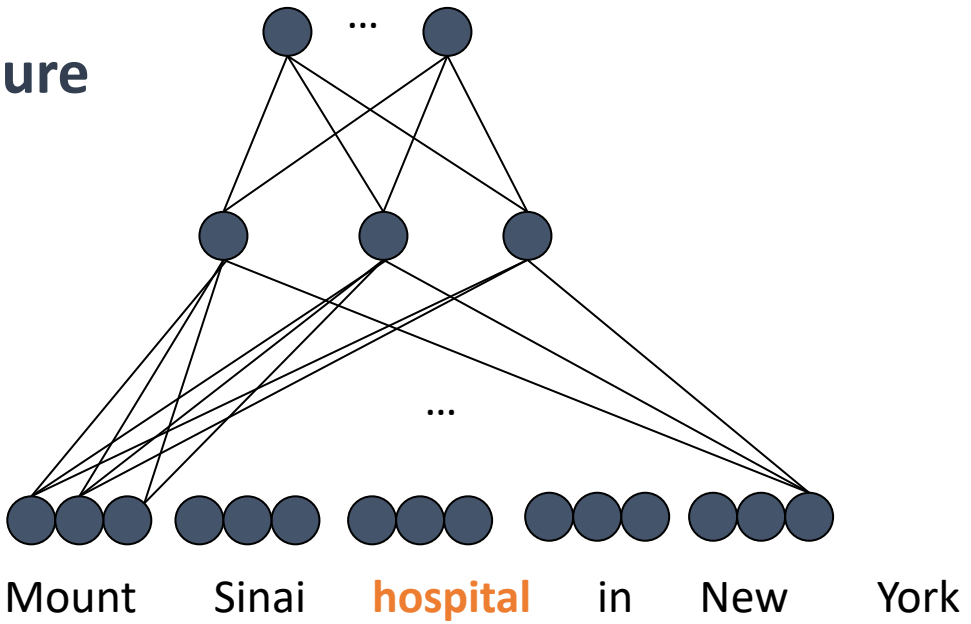
$$\hat{y}_k = \sigma(s_k) = \sigma(\sum_j w^2_{kj} h_j)$$

output layer

NN for NER

Can we simplify this architecture and *not* use a hidden layer?

Why not just use the word vectors directly?



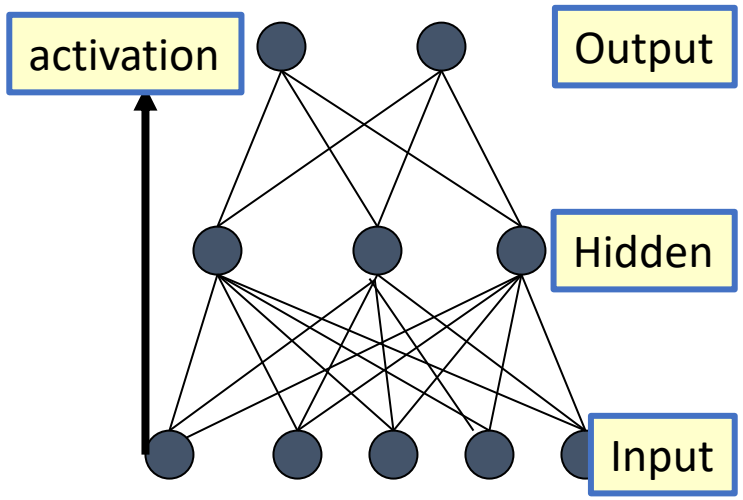
What do we get by adding an *extra* layer?

Training Neural Nets

- Learning so far – derive the gradient of the loss function, and use GD/SGD to optimize.
- Neural nets learn in the same way, however the process has to account for structure of the network.
- Mathematically – NN correspond to multiple steps of **function composition.**

$$\hat{y}_k = \sigma(s_k) = \sigma(\sum_j w^2_{kj} h_j)$$

$$h_j = \sigma(t_j) = \sigma(\sum_i w^1_{ji}\ x_i)$$



Training Neural Nets

- We can use the **chain rule** to get the gradient:

Chain rule:

$$\frac{\partial}{\partial v_{w_i}} f(z(v_{wi})) = \frac{\partial f}{\partial z} \cdot \frac{\partial z}{\partial v_{w_i}}$$

Simple Example

$$z = 3y$$

$$y = x^2$$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} = (3)(2x) = 6x$$

Back-Prop Comments

- **No guarantee of convergence;** may oscillate or reach a local minima.
  - *In practice, many large networks can be trained on large amounts of data for realistic problems.*
  - **To avoid local minima**: several trials with different random initial weights with majority or voting techniques
- Many epochs (tens of thousands) may be needed for adequate training.
  - Large data sets may require many hours (days) of CPU
- **Termination criteria**: Number of epochs; Threshold on training set error; No decrease in error; Increased error on a validation set.

Multi Layer NN: forward computation

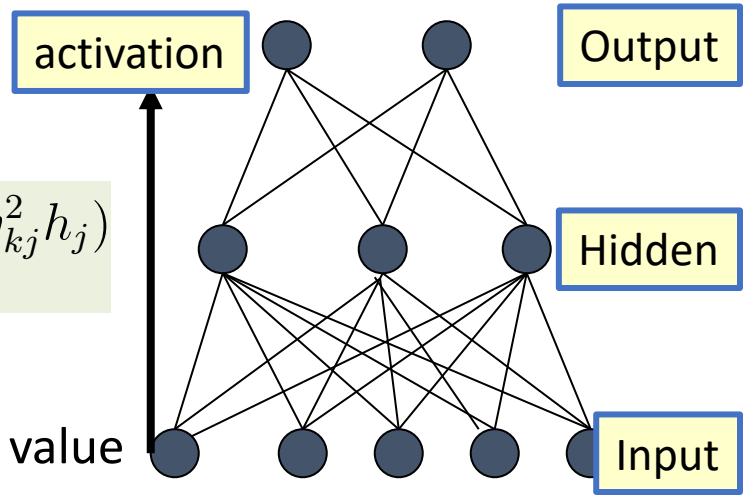
- Observe an input vector x
- *Push x through the network:*
  - For each **hidden unit** compute the activation value

$$h_j = \sigma(t_j) = \sigma(\sum_i w^1_{ji}\ x_i)$$

  - For each **output value**, compute the activation value coming from the hidden units
  - **Prediction:**

$$\hat{y}_k = \sigma(s_k) = \sigma(\sum_j w^2_{kj} h_j)$$

    - **Categories**: winner take all
    - **Vector**: take all output values
    - **Binary outputs**: Round to nearest 0-1 value



Back-Prop Comments

- **No guarantee of convergence;** may oscillate or reach a local minima.
  - *In practice, many large networks can be trained on large amounts of data for realistic problems.*
  - **To avoid local minima**: several trials with different random initial weights with majority or voting techniques
- Many epochs (tens of thousands) may be needed for adequate training.
  - Large data sets may require many hours (days) of CPU
- **Termination criteria**: Number of epochs; Threshold on training set error; No decrease in error; Increased error on a validation set.

Over-training Prevention

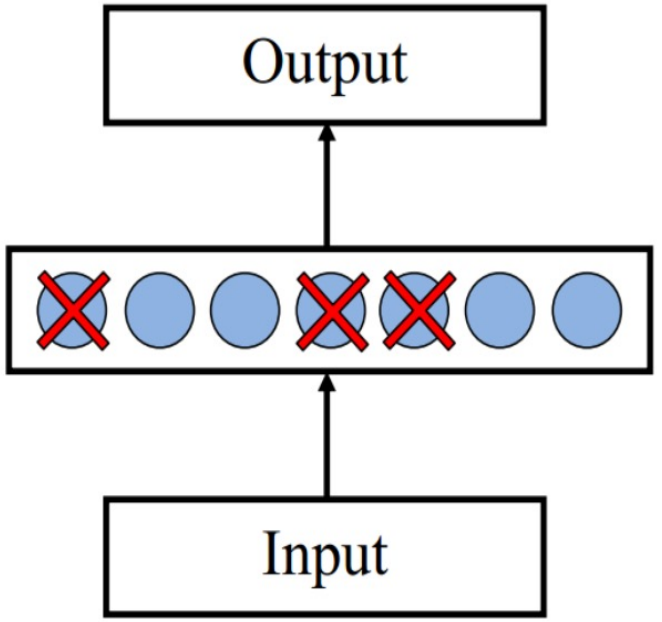
- Running too many epochs may over-train the network and result in over-fitting
  - Keep a hold-out validation set and test accuracy after every epoch
  - Maintain weights for best performing network on the validation set and return it when performance decreases significantly beyond that.
  - ***Why not just stop once validation error starts increasing?***
- To avoid losing training data to validation:
  - Use 10-fold cross-validation to determine the average number of epochs that optimizes validation performance
  - Train on the full data set using this many epochs to produce the final results

Over-training Prevention

- Too few hidden units prevent the system from adequately fitting the data and learning the concept.
- Using too many hidden units leads to over-fitting.
- **Similar cross-validation method can be used to determine an appropriate number of hidden units.**
- Another approach to prevent over-fitting is **weight-decay**: all weights are multiplied by some fraction in (0,1) after every epoch.
  - Encourages smaller weights and less complex hypothesis
- Equivalently: **use a regularizer**

Dropout Training

- Proposed by (Hinton et-al 2012)



Prevent feature co-adaptation

Encourage "independent contributions"

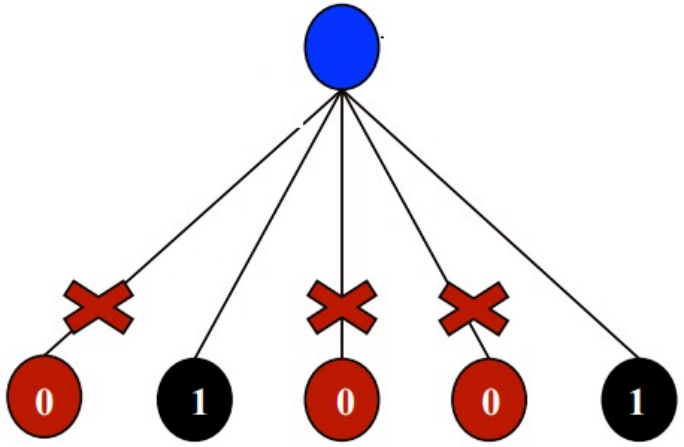
From different features

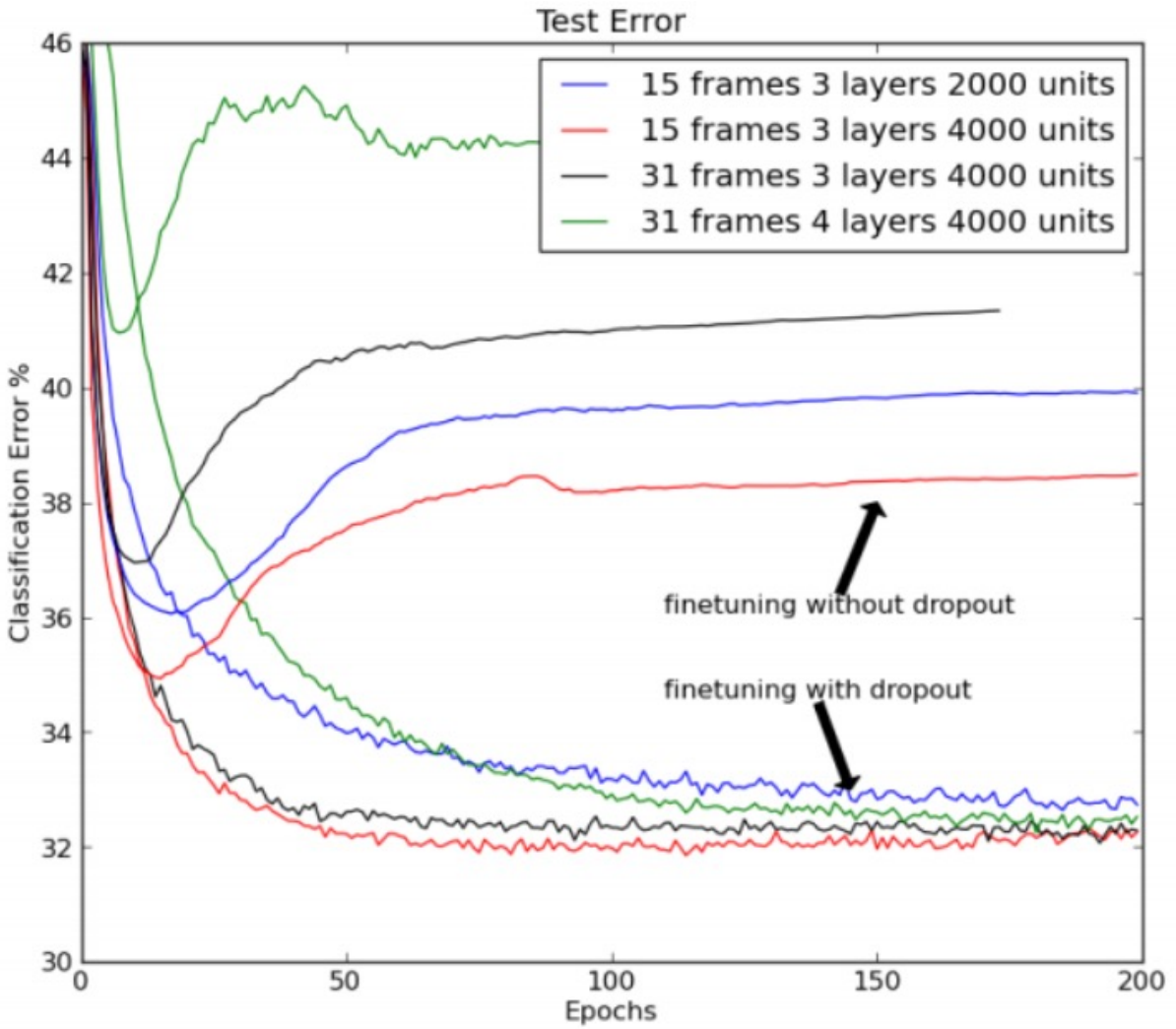
- At each training step, decide whether to delete one hidden unit with some probability p

Dropout training

- **Model averaging effect**
  - Average the results of multiple NN
  - Each NN has a different initialization point, resulting in a different model
  - Extremely computationally intensive for NNs!
- Much stronger than the known regularizer

- **What about the input space?**
  - Do the same thing!





- Dropout of 50% of the hidden units and 20% of the input units (Hinton et al, 2012)

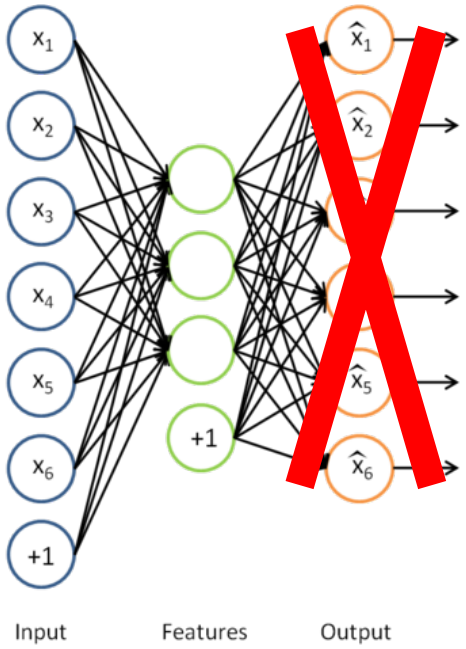
Learning Hidden Layer Representation

- **NN can be seen as a way to learn a feature representation**
  - Weight-tuning sets weights that define hidden units representation most effective at minimizing the error
- Backpropagation can define new hidden layer features that are not explicit in the input representation, but which capture properties of the input instances that are most relevant to learning the target function.
- *Trained hidden units can be seen as newly constructed features that re-represent the examples so that they are linearly separable*

Sparse Auto-Encoder

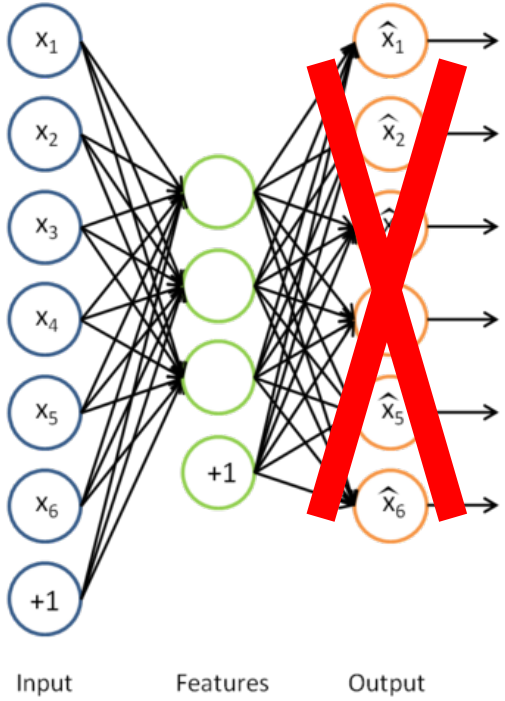
Goal: perfect reconstruction of the input vector x, by the output x'

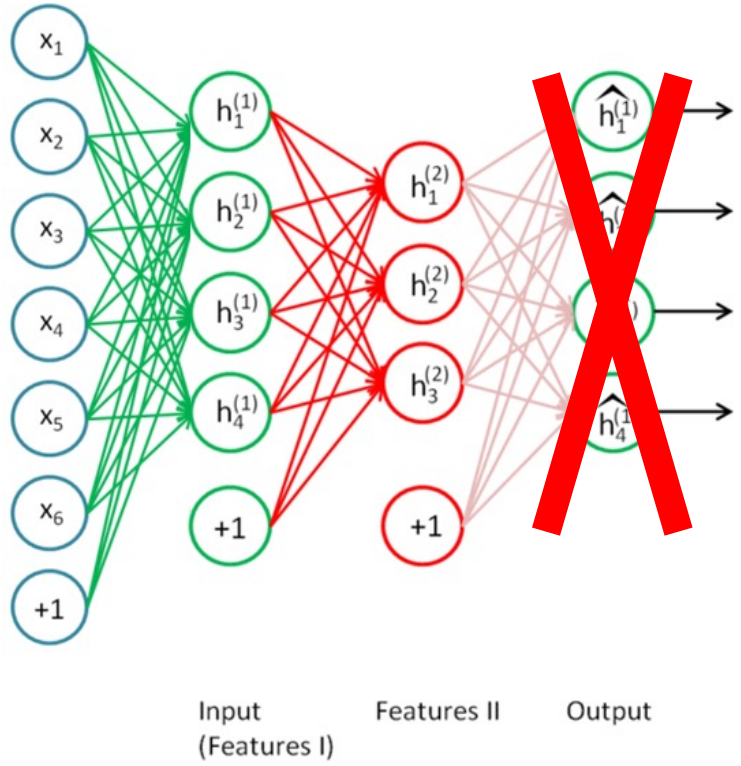
- **Simple approach**:
  - Minimize the error function l(h(x),x)
  - After optimization:
    - Drop the reconstruction layer



Stacking Auto Encoder

- Add a new layer, and a reconstruction layer for it.
- Repeat.





Representation Learning

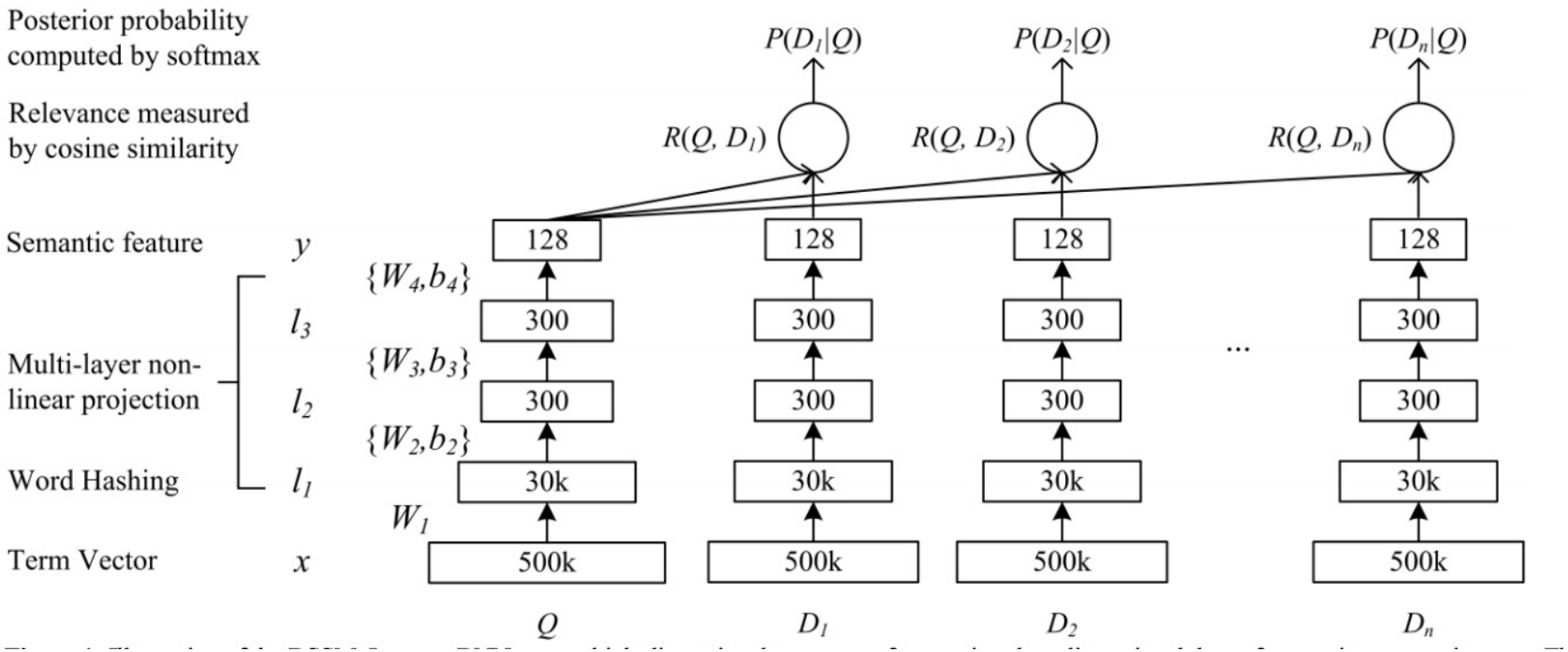
- Learning Deep Structured Semantic Models for Web Search using Clickthrough Data. Huang et-al. CIKM'13
- **IR goal:** *find relevant documents given a short query*
  - **Challenge:** *how can a short query capture the intent and information need of the user?*
- **Settings**: *Massive clickthrough log data, consisting of documents and search queries.*

Paper Review

- Our goal is to learn a representation for queries such that they match documents selected by users.
- **What is the simplest way to do that given the data?**
- Given a document and query, are the two a good match or not?
- How should we encode each element?

Paper Review

- Learning Deep Structured Semantic Models for Web Search using Clickthrough Data. Huang et-al. CIKM'13



Paper Review

#	Models	NDCG@1	NDCG@3	NDCG@10
1	TF-IDF	0.319	0.382	0.462
2	BM25	0.308	0.373	0.455
3	WTM	0.332	0.400	0.478
4	LSA	0.298	0.372	0.455
5	PLSA	0.295	0.371	0.456
6	DAE	0.310	0.377	0.459
7	BLTM-PR	0.337	0.403	0.480
8	DPM	0.329	0.401	0.479
9	DNN	0.342	0.410	0.486
10	L-WH linear	0.357	0.422	0.495
11	L-WH non-linear	0.357	0.421	0.494
12	**L-WH DNN**	**0.362**	**0.425**	**0.498**

Table 2: Comparative results with the previous state of the art approaches and various settings of DSSM.

10,000 feet view

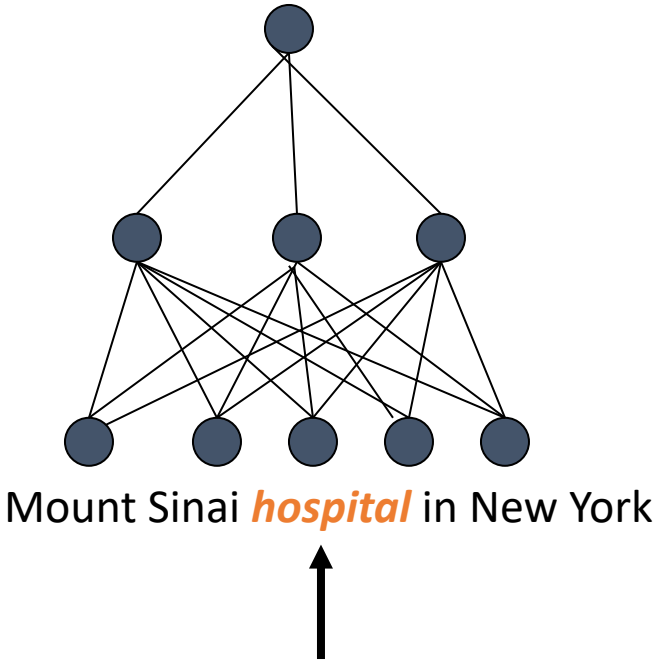
- Neural networks are an extremely flexible way to define complex prediction models.
  - **Simple update rule:** propagate the error on the architecture of the network (essentially DAG).
  - All deep learning models share this property, **just different DAGs**!
- **Key issues**:
  - Preventing over fitting
  - Representation learning, pre-training with minimal supervision

10,000 feet view

- So far, we looked at simple classification problems.
  - Assume a word window, that provides fixed sized inputs.
    - In case you “run out of input” – zero padding.
- What can you do if the size of the input is not fixed?
  - Some notion of compositionality is needed
  - Simplest approach: sum up word vectors
    - Document structure is lost.. **Can we do better?**

Open Question

Recall the NER problem and the architecture associated with it.



How can we define a NN for other NLP problems?
E.g., Sentiment Analysis defined over complete sentences?
(**hint**: how do you deal with different sized inputs?)

Distributional Similarity

A bottle of tesgüino is on the table
Everybody likes tesgüino
Tesgüino makes you drunk
We make tesgüino out of corn.

Question:

What is tesgüino?

Tesgüino = (Bottle = 123, Table =54, drunk = 141, Corn = 91, …)

Bourbon = (Bottle = 231, Table =41, drunk =231, corn =121, …)

Vodka = (Bottle = 311, Table =82, drunk =321, corn =0, …)

Distributional Similarity

Given the vector based representation of words
we can compute their similarity easily -

$$\cos(v, w) = \frac{v \cdot w}{|v|\ |w|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

Using Positive PMI, ensure that Cosine similarity
will have non-negative values

A machine learning perspective

- So far we looked at words as **discrete objects**.
  - For example, when building a sentiment classifier, each word was represented as a different coordinate

- "Great" = [0,0,0,0,0,0,0,1,0,0,… 0]
- "Awesome" = [0,0,0,1,0,0,…,0]
  - *This is known as "**one hot**" representation.*

A machine learning perspective

- Using “one-hot” representation, the connections between words are lost.

- We typically designed **complex feature functions** to get over that:
  - Maintain a dictionary of related words according to
    - Meaning (identify synonyms)
    - Word group (slang, function words, positive, negative,..)

A machine learning perspective

- **Can we use vector based methods to represent words other decision tasks?**
  - We can potentially overcome lexical sparseness problems
  - Reduce the dimensionality of the learning problem
- How should we evaluate the learned semantic representation?

Word Embedding

- Basic idea: represent words in a continuous vector space.
  - Similar idea as using PMI
- **Key difference:**
  - Find low dimensional **dense** representation
  - **Instead of counting co-occurrence, use discriminative learning methods**
    - Predict surrounding words

Word2Vec

“ AI fields such as NLP, machine learning , vision, have increased in popularity in recent years”

- For each word, predict other words in window C
- **Training Objective**: maximize the probability of context word, given the current word.

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j<c} \log p(w_{t+j}|w_t)$$

Word2Vec

- We want to evaluate $p(w_o|w_i)$
- For each word maintain **two** vectors
  - **Inside** word and **outside** word (context)
  - V represent inside, V' outside.

$$p(w_o|w_i) = \frac{\exp(v'_{w_o} v_{w_i})}{\sum_{w=1}^{W} \exp(v'_w v_{w_i})}$$

Efficient Implementation

- For non-trivial vocabulary, the normalization factor is too costly to compute accurately.

$$p(w_o|w_i) = \frac{\exp(v'_{w_o} v_{w_i})}{\sum_{w=1}^{W} \exp(v'_w v_{w_i})}$$

- **Skip-gram with negative sampling**
  - Binary logistic regression for a small subset:
    - True pair, small subset of negative examples.

Skip-gram with Negative Sampling

- New objective function:

$$\log \sigma(v'_{w_o} v_{w_I}) + \sum_{i=1}^{k} \mathop{\mathbb{E}}_{w_i \sim P_n(w)} \left[\log \sigma(-v'_{w_i}{}^T v_{w_I})\right]$$

Maximize the probability of center + context words

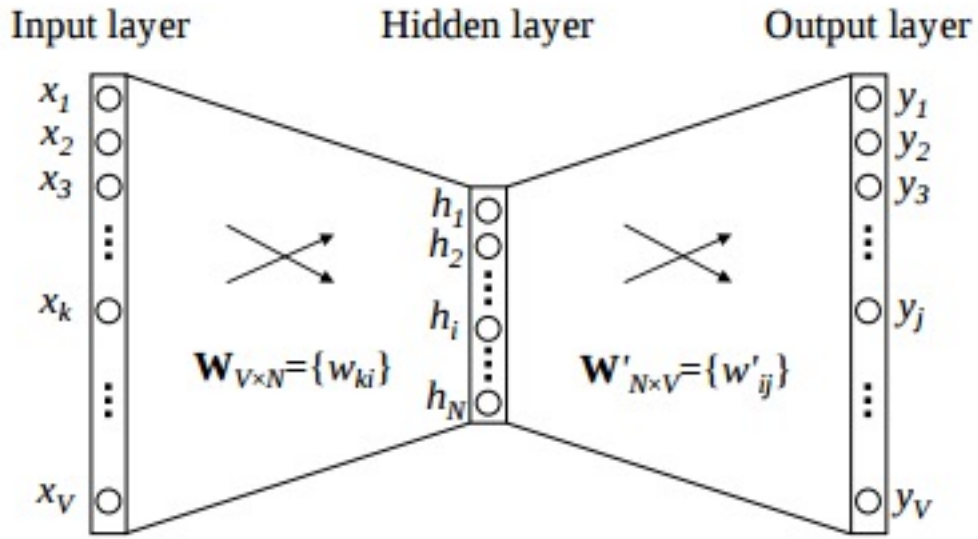
Minimize the probability of random words

Note:

- Only pick a **small subset** of negative examples
- samples are drawn from a distribution: $P_n(w)$
- $P_n(w)$ captures unigram statistics, modified to increase the probability of sampling low frequency words.

Word Embedding vs. AE

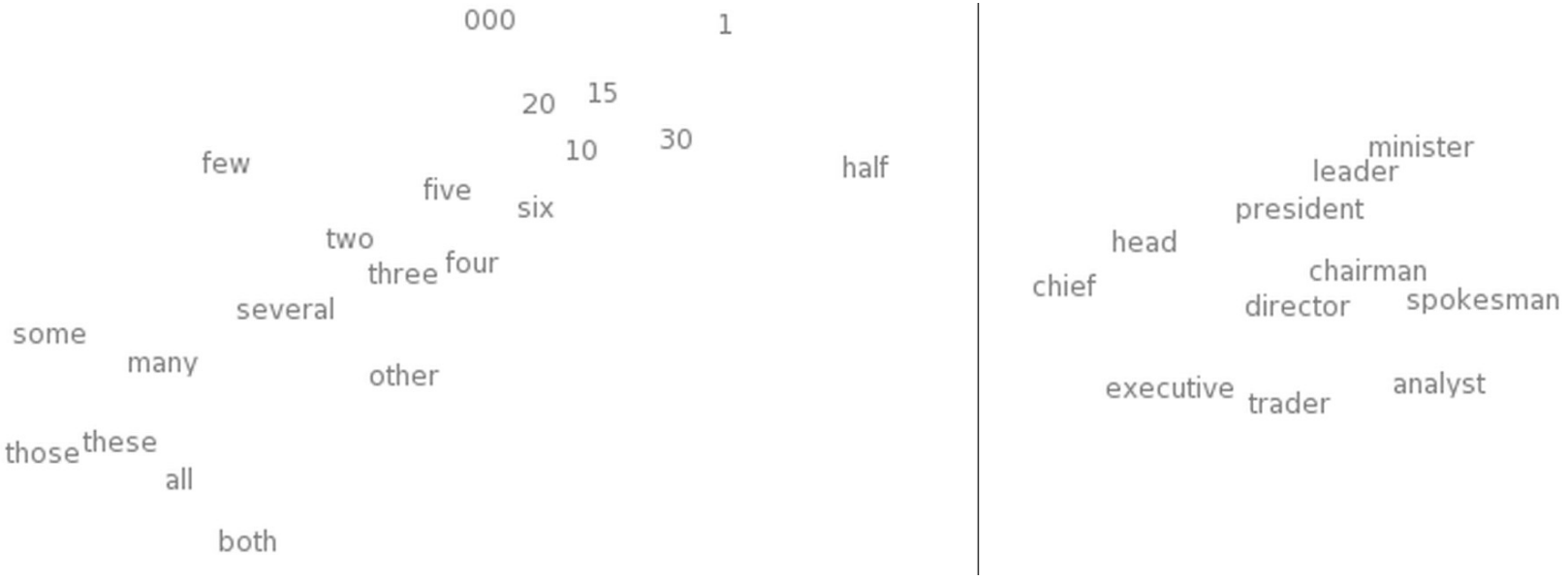
- Is this the same as an Auto-Encoder?



word2vec model architecture

Word Embedding

- **Word embedding**: *move to a low dimensional, real valued dense representation of the input*
  - Key idea: similar words should have similar vectors



Turian et-al 2010

Word2Vec

```
Enter word or sentence (EXIT to break): Chinese river

                  Word       Cosine distance
-------------------------------------------
       Yangtze_River                0.667376
             Yangtze                0.644091
      Qiantang_River                0.632979
    Yangtze_tributary               0.623527
     Xiangjiang_River               0.615482
        Huangpu_River               0.604726
       Hanjiang_River               0.598110
        Yangtze_river               0.597621
          Hongze_Lake               0.594108
              Yangtse               0.593442
```

Mikolov et-al 2013