# IN4080 – 2021 FALL
## NATURAL LANGUAGE PROCESSING

Jan Tore Lønning

# Lecture 8, 4 Oct.

# Today



- POS-tagging
- HMM-tagging
- Discriminative tagging
- Logistic Regression (once more)
- Skip-gram with negative sampling

# Tagged text and tagging



```
[('They', 'PRP'), ('saw', 'VBD'), ('a', 'DT'), ('saw', 'NN'), ('.', '.')]
[('They', 'PRP'), ('like', 'VBP'), ('to', 'TO'), ('saw', 'VB'), ('.', '.')]
[('They', 'PRP'), ('saw', 'VBD'), ('a', 'DT'), ('log', 'NN')]
```

- In tagged text each token is assigned a "part of speech" (POS) tag
- A tagger is a program which automatically ascribes tags to words in text
- From the context we are (most often) able to determine the tag.
  - But some sentences are genuinely ambiguous and hence so are the tags.

# Various POS tag sets



- A tagged text is tagged according to a fixed small set of tags.
- There are various such tag sets.
- Brown tagset:
  - Original: 87 tags
  - Versions with extended tags <original>-<more>
    - Comes with the Brown corpus in NLTK
- Penn treebank tags: 35+9 punctuation tags
- Universal POS Tagset, 12 tags,

# Brown vs. Penn: Nouns



| | | |
|---|---|---|
| NN | Noun, sing. or mass | *llama* |
| NNS | Noun, plural | *llamas* |
| NNP | Proper noun, singular | *IBM* |
| NNPS | Proper noun, plural | *Carolinas* |

Penn treebank

| | | |
|---|---|---|
| NN | (common) singular or mass noun | time, world, work, school, family, door |
| NN$ | possessive singular common noun | father's, year's, city's, earth's |
| NNS | plural common noun | years, people, things, children, problems |
| NNS$ | possessive plural noun | children's, artist's parent's years' |
| NP | singular proper noun | Kennedy, England, Rachel, Congress |
| NP$ | possessive singular proper noun | Plato's Faulkner's Viola's |
| NPS | plural proper noun | Americans Democrats Belgians Chinese Sox |
| NPS$ | possessive plural proper noun | Yankees', Gershwins' Earthmen's |
| NR | adverbial noun | home, west, tomorrow, Friday, North, |
| NR$ | possessive adverbial noun | today's, yesterday's, Sunday's, South's |
| NRS | plural adverbial noun | Sundays Fridays |

Brown, original

# Different tagsets - example



| | | | Brown | Penn treebank ('wsj') | Universal |
|---|---|---|---|---|---|
| | he | she | PPS | PRP | PRON |
| I | | | PPSS | PRP | PRON |
| me | him | her | PPO | PRP | PRON |
| my | his | her | PP$ | PRP$ | DET |
| mine | his | hers | PP$$ | ? | PRON |

# Ambiguity rate



| | | WSJ | Brown |
|---|---|---|---|
| **Types:** | | | |
| **Unambiguous** | (1 tag) | 44,432 (**86%**) | 45,799 (**85%**) |
| **Ambiguous** | (2+ tags) | 7,025 (**14%**) | 8,050 (**15%**) |
| **Tokens:** | | | |
| **Unambiguous** | (1 tag) | 577,421 (**45%**) | 384,349 (**33%**) |
| **Ambiguous** | (2+ tags) | 711,780 (**55%**) | 786,646 (**67%**) |

**Figure 8.2** Tag ambiguity for word types in Brown and WSJ, using Treebank-3 (45-tag) tagging. Punctuation were treated as words, and words were kept in their original case.

# How ambiguous are tags (J&M, 2.ed)



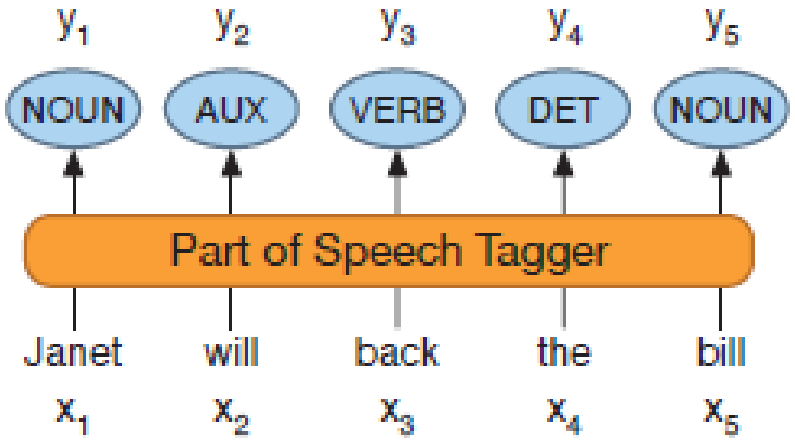
| | | 87-tag Original Brown | | 45-tag Treebank Brown | |
|---|---|---|---|---|---|
| **Unambiguous (1 tag)** | | **44,019** | | **38,857** | |
| **Ambiguous (2–7 tags)** | | **5,490** | | **8844** | |
| Details: | 2 tags | 4,967 | | 6,731 | |
| | 3 tags | 411 | | 1621 | |
| | 4 tags | 91 | | 357 | |
| | 5 tags | 17 | | 90 | |
| | 6 tags | 2 | (*well*, *beat*) | 32 | |
| | 7 tags | 2 | (*still*, *down*) | 6 | (*well*, *set*, *round*, *open*, *fit*, *down*) |
| | 8 tags | | | 4 | (*'s*, *half*, *back*, *a*) |
| | 9 tags | | | 3 | (*that*, *more*, *in*) |

BUT: Not directly comparable because of different tokenization

# Tagging as Sequence Classification



- Classification (earlier):
  - a well-defined set of observations, O
  - a given set of classes, $S=\{s_1, s_2, \ldots, s_k\}$
  - Goal: a classifier, $\gamma$, a mapping from O to S




- Sequence classification:
  - Goal: a classifier, $\gamma$, a mapping from sequences of elements from O to sequences of elements from S:
  - $\gamma(o_1, o_2, \ldots o_n) = (s_{k1}, s_{k2}, \ldots s_{kn})$

# Baseline tagger



- In all classification tasks establish a baseline classifier.
- Compare the performance of other classifiers you make to the baseline.
- For tagging, a natural baseline is the Most Frequent Class Baseline:
  - Assign each word the tag to which is occurred most frequent in the training set
  - For words unseen in the training set, assign the most frequent tag in the training set.

# Today



- POS-tagging
- HMM-tagging
- Discriminative tagging
- Logistic Regression (once more)
- Skip-gram with negative sampling

# Hidden Markov Model (HMM) tagger



## Extension of language model

- Two layers:
  - Observed: the sequence of words
  - Hidden: the tags/classes where each word is assigned a class

## Extension of Naive Bayes

- NB assigns a class to each observation
- An HMM is a sequence classifier:
  It assigns a sequence of classes to a sequence of words

# HMM is a probabilistic tagger



Notation:
$t_1^n = t_1, t_2, \ldots t_n$

- The goal is to decide: $\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n | w_1^n)$
- Using Bayes theorem: $\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} \frac{P(w_1^n | t_1^n) P(t_1^n)}{P(w_1^n)}$
- This simplifies to: $\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(w_1^n | t_1^n) P(t_1^n)$

  because the denominator is the same for all tag sequences

# Simplifying assumption 1



- For the tag sequence, we apply the chain rule
- $P(t_1^n) = P(t_1)P(t_2|t_1)P(t_3|t_1t_2) \dots P\left(t_i|t_1^{i-1}\right) \dots P(t_n|t_1^{n-1})$
- We then assume the Markov (chain) assumption
- $P(t_1^n) = P(t_1)P(t_2|t_1)P(t_3|t_2) \dots P(t_i|t_{i-1}) \dots P(t_n|t_{n-1})$
-

$$P(t_1^n) \approx P(t_1) \prod_{i=2}^{n} P(t_i|t_{i-1}) = \prod_{i=1}^{n} P(t_i|t_{i-1})$$

  - Assuming a special start tag $t_0$ and $P(t_1) = P(t_1|t_0)$

# Simplifying assumption 2



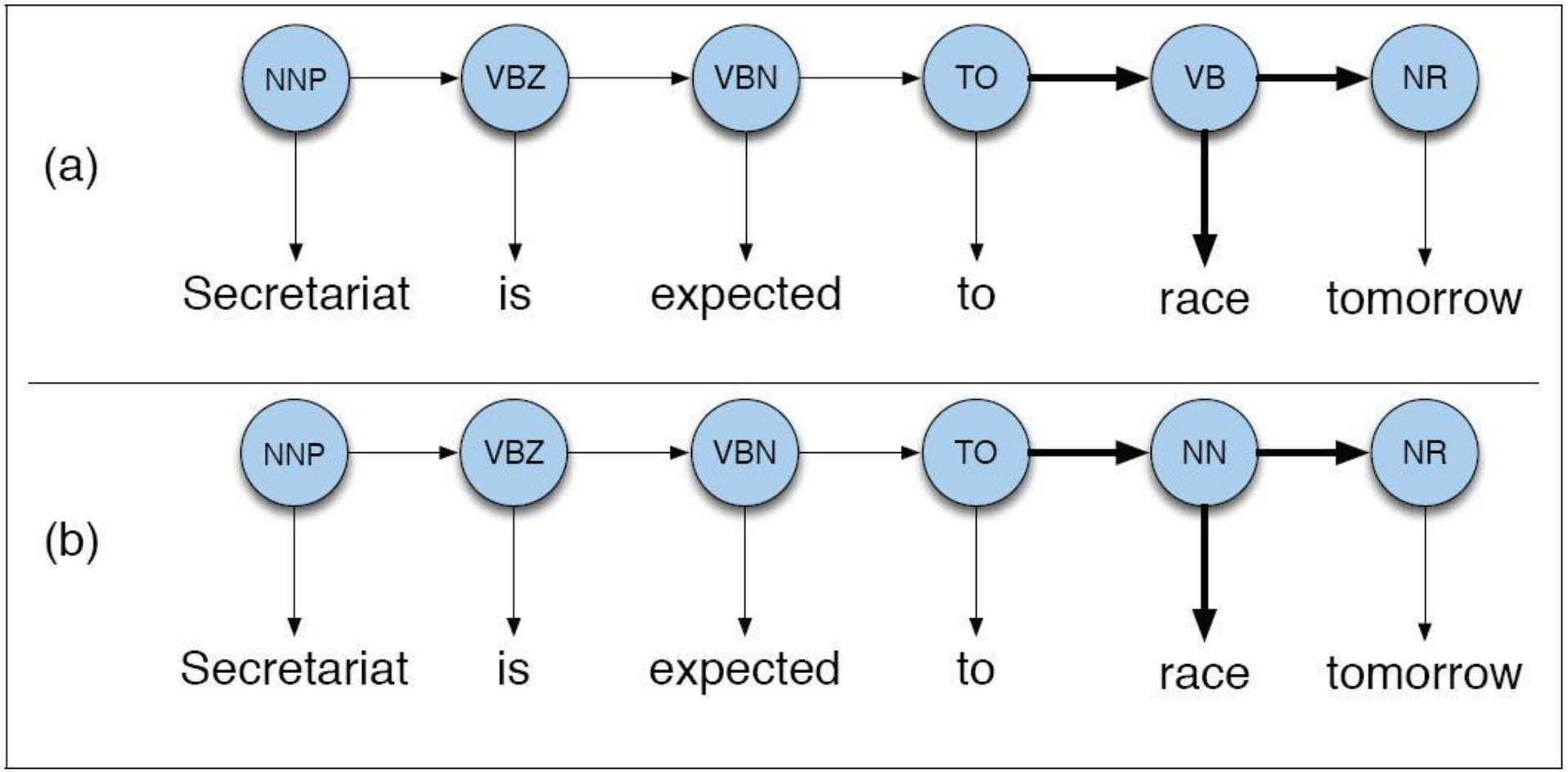
- Applying the chain rule

$$P(w_1^n|t_1^n) = \prod_{i=1}^{n} P\left(w_i|w_1^{i-1}t_1^n\right)$$

  i.e., a word depends on all the tags and on all the preceding words

- We make the simplifying assumption: $P\left(w_i|w_1^{i-1}t_1^n\right) \approx P(w_i|t_i)$
- i.e., a word depends only on the immediate tag, and hence

$$P(w_1^n|t_1^n) = \prod_{i=1}^{n} P(w_i|t_i)$$


(a)
NNP
VBZ
VBN
TO
VB
NR
Secretariat
is
expected
to
race
tomorrow
(b)
NNP
VBZ
VBN
TO
NN
NR
Secretariat
is
expected
to
race
tomorrow

# Training



- From a tagged training corpus, we can estimate the probabilities with Maximum Likelihood (as in Language Models and Naïve Bayes:)
- $\hat{P}(t_i|t_{i-1}) = \frac{C(t_{i-1},t_i)}{C(t_{i-1})}$
- $\hat{P}(w_i|t_i) = \frac{C(w_i,t_i)}{C(t_i)}$

# Putting it all together



- From a trained model, it is straightforward to calculate the probability of a sentence with a tag sequence

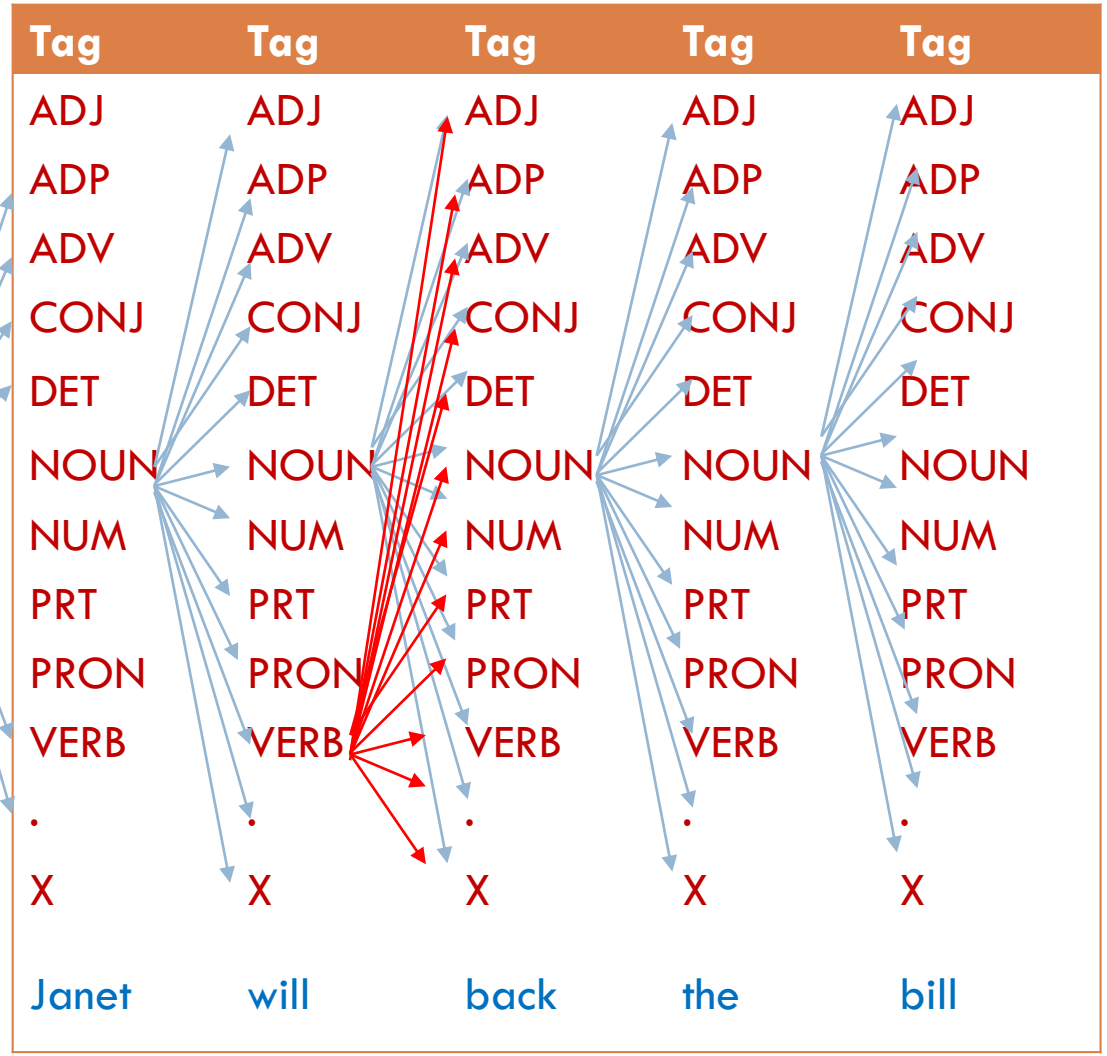
$$P(w_1^n, t_1^n) = P(t_1^n)P(w_1^n|t_1^n) \approx \prod_{i=1}^{n} P(t_i|t_{i-1}) \prod_{i=1}^{n} P(w_i|t_i)$$

$$= \prod_{i=1}^{n} P(t_i|t_{i-1})P(w_i|t_i)$$

- To find the best tag sequence, we could – in principle – calculate this for all possible tag sequences and choose the one with highest score
- $\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}}\, P(w_1^n|t_1^n)P(t_1^n)$
- Impossible in practice – There are too many

# Possible tag sequences

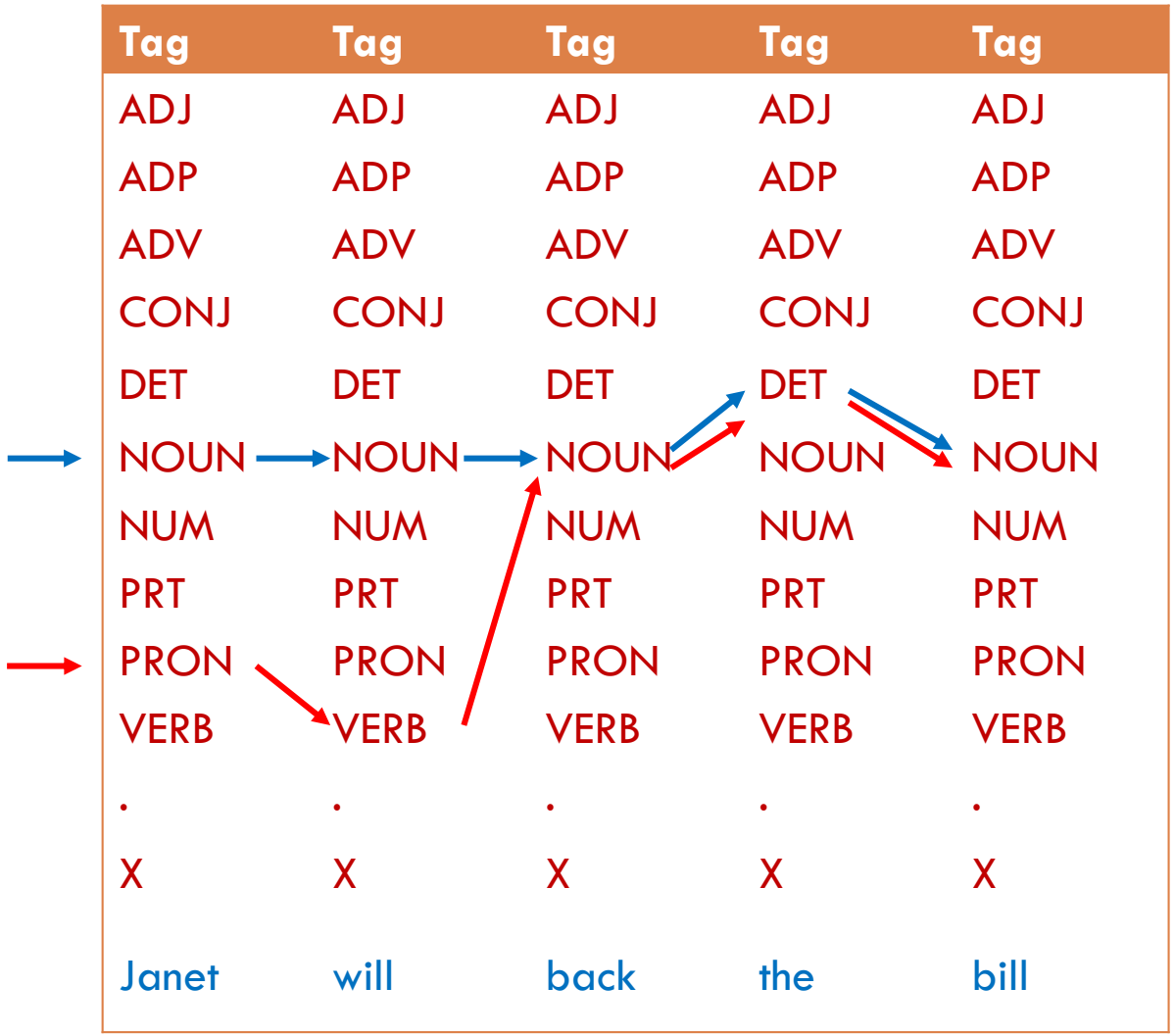





- The number of possible tag sequences =
- The number of paths through the trellis =
- $m^n$
  - $m$ is the number of tags in the set
  - $n$ is the number of tokens in the sentence
  - Here: $12^5 \approx 250{,}000$.

# Viterbi algorithm (dynamic programming)






- Walk through the word sequence
- For each word keep track of
  - all the possible tag sequences up to this word and the probability of each sequence
- If two paths are equal from a point on, then
- The one scoring best at this point will also score best at the end
- Discard the other one

# Viterbi algorithm



- A nice example of dynamic programming
- Skip the details:
  - Viterbi is covered in IN2110
  - We will use preprogrammed tools in this course – not implement ourselves
  - HMM is not state of the art taggers

# HMM trigram tagger



- Take two preceding tags into consideration
- $P(t_1^n) \approx \prod_{i=1}^{n} P(t_i|t_{i-1}, t_{i-2})$
-

$$P(w_1^n, t_1^n) = \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1}, t_{i-2})$$

- Add two initial special states and one special end state

# Challenges for the HMM-tagger



- Even with Viterbi, it is expensive:
  - For the trigram, the size of the trellis: $(n+2) \times m^3$
    - $n$ words in the sequence
    - $m$ tags in the model
  - Example, 6 words
    - 12 tags: 15,552
    - With 45 tags: 820,125
    - With 87 tags: 5,926,527
- We have probably not seen all tag trigrams during training:
  - We must use back-off or interpolation to lower n-grams
- Words not observed during training:
  - How can we include e.g. morphological features?
    - e.g., *-ly* ➔ Adv

# Today



- POS-tagging
- HMM-tagging
- Discriminative tagging
- Logistic Regression (once more)
- Skip-gram with negative sampling

# Discriminative tagging



Notation:
$t_1^n = t_1, t_2, \ldots t_n$

- The goal of tagging is to decide: $\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n | w_1^n)$
- HMM is generative.
  - It estimates $P(w_1^n | t_1^n) P(t_1^n) = P(w_1^n, t_1^n)$
- As for text classification, we could instead use a discriminative procedure and try to estimate the tag sequence directly
- $P(t_1^n | w_1^n) = P(t_1 | w_1^n) P(t_2 | t_1, w_1^n) \ldots P\left(t_i | t_1^{i-1}, w_1^n\right) \ldots = \prod_{i=1}^n P\left(t_i | t_1^{i-1}, w_1^n\right)$

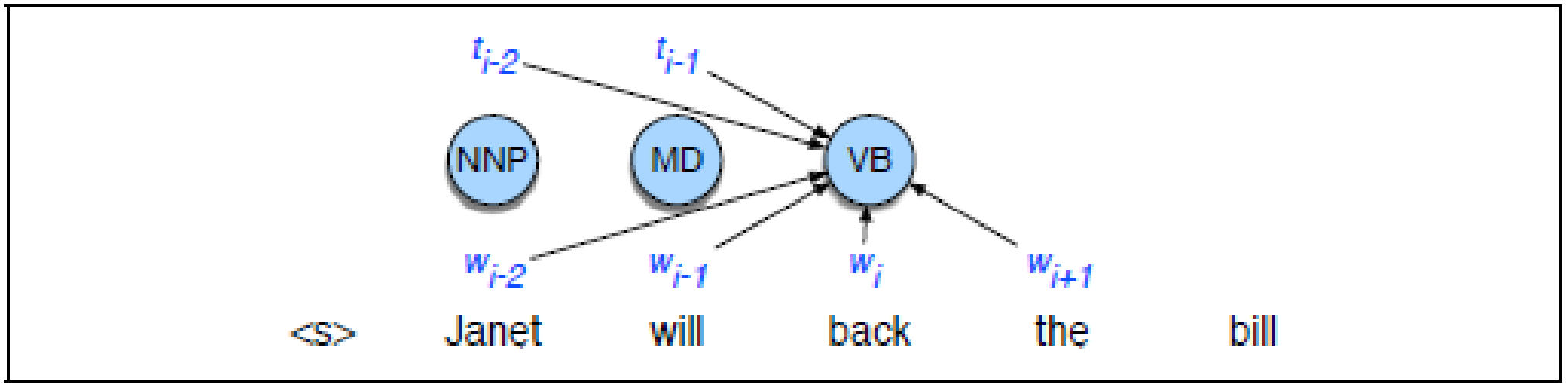



**Figure 8.13** An MEMM for part-of-speech tagging showing the ability to condition on more features.

- Apply the chain rules for probabilities
- $P(t_1^n|w_1^n) = P(t_1|w_1^n)P(t_2|t_1, w_1^n) \dots P\left(t_i|t_1^{i-1}, w_1^n\right) \dots = \prod_{i=1}^{n} P\left(t_i|t_1^{i-1}, w_1^n\right)$
- $\underset{t_1^n}{\operatorname{argmax}}\, P(t_1^n|w_1^n) = \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P\left(t_i|t_1^{i-1}, w_1^n\right)$
- Simplifying assumptions:
- $\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}}\, P(t_1^n|w_1^n) \approx \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P\left(t_i|\, t_{i-k}^{i-1} w_{i-m}^{i+m}\right)$

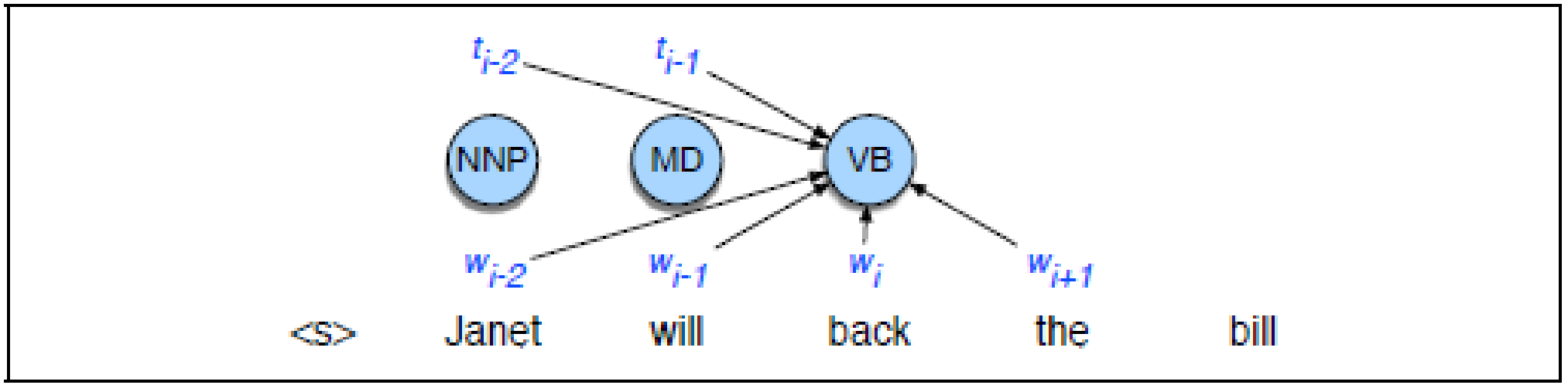



**Figure 8.13** An MEMM for part-of-speech tagging showing the ability to condition on more features.

- Simplifying assumptions:
- $\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n | w_1^n) \approx \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P\left(t_i | t_{i-k}^{i-1} w_{i-m}^{i+m}\right)$
- The tag only depends on *k* preceding tags, typically k=1 or k=2
- Tag $t_i$ only depends on the words in a window around $w_i$
  - of size *m*

# Feature extraction



- $\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n | w_1^n) \approx \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P\left(t_i | \, t_{i-k}^{i-1} w_{i-m}^{i+m}\right)$

- We use a template to extract features from preceding tag(s) and neighboring words.
- The actual number of features may be large.
- Observe that properties may be combined into one feature

$t_i = \text{VB}$ and $w_{i-2} = \text{Janet}$
$t_i = \text{VB}$ and $w_{i-1} = \text{will}$
$t_i = \text{VB}$ and $w_i = \text{back}$
$t_i = \text{VB}$ and $w_{i+1} = \text{the}$
$t_i = \text{VB}$ and $w_{i+2} = \text{bill}$
$t_i = \text{VB}$ and $t_{i-1} = \text{MD}$
$t_i = \text{VB}$ and $t_{i-1} = \text{MD}$ and $t_{i-2} = \text{NNP}$
$t_i = \text{VB}$ and $w_i = \text{back}$ and $w_{i+1} = \text{the}$

# Remarks



- $\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n | w_1^n) \approx \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P\left(t_i | \, t_{i-k}^{i-1} w_{i-m}^{i+m}\right)$

- The extracted features corresponds to J&M's "small features", $f_k(y_{i-1}, y_i, X, i)$
- J&M includes the tag into the feature:
  - There are alternative ways of presenting this
  - We do not have to include the class

$t_i = \text{VB and } w_{i-2} = \text{Janet}$
$t_i = \text{VB and } w_{i-1} = \text{will}$
$t_i = \text{VB and } w_i = \text{back}$
$t_i = \text{VB and } w_{i+1} = \text{the}$
$t_i = \text{VB and } w_{i+2} = \text{bill}$
$t_i = \text{VB and } t_{i-1} = \text{MD}$
$t_i = \text{VB and } t_{i-1} = \text{MD and } t_{i-2} = \text{NNP}$
$t_i = \text{VB and } w_i = \text{back and } w_{i+1} = \text{the}$

# Features (for unknown words)



- We may include features which inspect properties of the word

$w_i$ contains a particular prefix (from all prefixes of length $\leq 4$)
$w_i$ contains a particular suffix (from all suffixes of length $\leq 4$)
$w_i$ contains a number
$w_i$ contains an upper-case letter
$w_i$ contains a hyphen
$w_i$ is all upper case
$w_i$'s word shape
$w_i$'s short word shape
$w_i$ is upper case and has a digit and a dash (like *CFC-12*)
$w_i$ is upper case and followed within 3 words by Co., Inc., etc.

# Decoding



- Goal: $\underset{t_1^n}{\operatorname{argmax}} P(t_1^n|w_1^n) = \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P\left(t_i|t_1^{i-1}, w_1^n\right)$
- Simplest alternative: Greedy sequence decoding:
  - Choose the best tag for the first word in the sentence $\underset{t_1}{\operatorname{argmax}} P(t_1|w_1^n)$
  - Then choose the best tag for the second word in the sentence, given the choice for the first word, $\underset{t_2}{\operatorname{argmax}} P(t_2|t_1, w_1^n)$
  - And so on, tagging one word at a time, $\underset{t_i}{\operatorname{argmax}} P(t_i|t_{i-1}, w_1^n)$

  until we have finished the sentence.

# Training a greedy classifier



- The training examples are extracted from a tagged corpus
- For each word occurrence in the corpus, one training example:
  - The class is the correct tag
  - The features are extracted from the context
- One can then
  - train any multi-class ML-algorithm, e.g., multinomial logistic regression
  - apply greedy tagging on untagged text.

# Shortcomings of greedy decoding



- Early decisions, considers only one tag at a time
  - If $\hat{t}_1 = \underset{t_1}{\operatorname{argmax}}\, P(t_1|w_1^n)$ and $\hat{t}_2 = \underset{t_1}{\operatorname{argmax}}\, P(\hat{t}_1|w_1^n)$ then

    $\underset{t_1 t_2}{\operatorname{argmax}}\, P(t_1 t_2|w_1^n)$ does have to equal $\hat{t}_1 \hat{t}_2$

- Compare to HMM which considers whole tag sequences and choose the most probable sequence.

# Maximum Entropy Markov Models (MEMM)



- If the model uses a limited history,
- $\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n | w_1^n) \approx \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P\left(t_i | \, t_{i-k}^{i-1} w_{i-m}^{i+m}\right)$

  one may use a form of Viterbi decoding
- We may then find $\underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P\left(t_i | \, t_{i-k}^{i-1} w_{i-m}^{i+m}\right)$
- This should make better result

# However



- The greedy sequence decoding does surprisingly well
- And equally surprising: using preceding tags as features does not improve the tagger that much compared to not including them.
- See mandatory assignment 2B

# Conditional Random Fields



- Even if we use Viterbi decoding and find the most probable overall tag sequence, we so far trained or model on the greedy task.
- What we ideally should have done was to also train or model on the task of predicting the optimal whole sequence
- Conditional Random Fields (CRFs) is a generalization compared to MEMM:
  - Makes it possible to optimize training for whole tag sequences
  - Slow in training
  - Considered the best tool for sequence labelling until a few years ago
- Currently, neural networks ("deep learning") are considered the best tool

| | Generative | Discrimantive |
|---|---|---|
| Classification | Naive Bayes | Logistic regression |
| Sequence labeling | HMM | CRF |

# Summary



- Tags
- Tagging: the process of disambiguation and choosing tags
- HMM is a generative model for tagging
- Reasonably efficient for the bigram tagger
- In need of smoothing and module for unseen words
- A discriminative tagger may exploit more properties of the words

# Today



- POS-tagging
- HMM-tagging
- Discriminative tagging
- Logistic Regression (once more)
- Skip-gram with negative sampling

# SoftMax – Multinomial Logistic Regression



- Partly repetition – partly some properties/details we haven't considered so far
- Serve as background for better understanding
  - Learning skip-gram with negative sampling (ch. 6)
  - CRF (ch. 8)
  - Neural networks (ch. 7, 9, etc.)

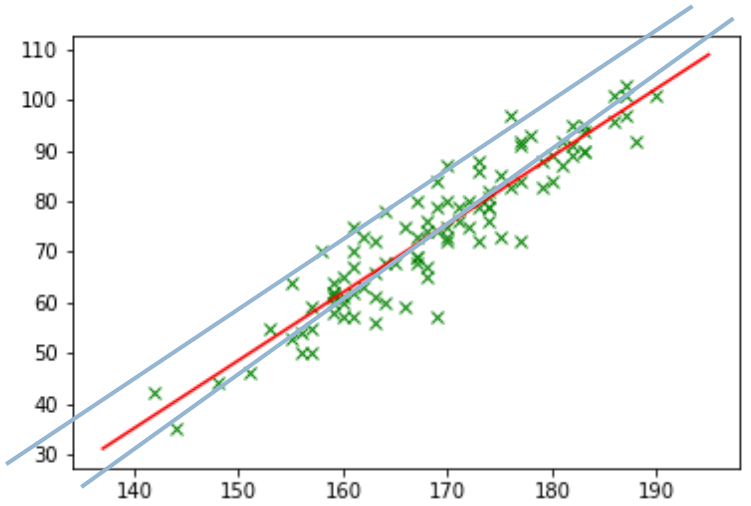
# Today



- POS-tagging
- HMM-tagging
- Discriminative tagging
- Logistic Regression (once more)
  - Linear regression
  - Logistic regression
  - Multinomial logistic regression (soft-max)
- Skip-gram with negative sampling

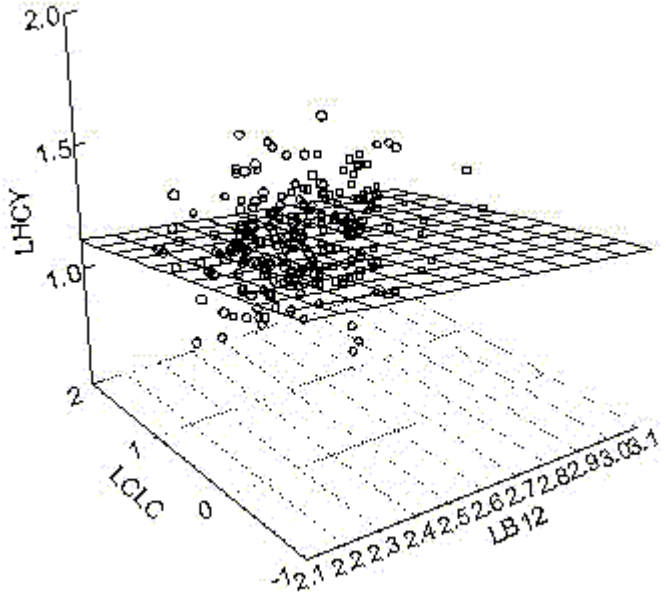
# Linear regression






- Try to fit a straight line to the observed data
- General case: try to fit the best (hyper-)plane

$$\hat{y} = f(x_0, x_1, \ldots, x_n) = \sum_{i=0}^{n} w_i x_i = \vec{w} \cdot \vec{x}$$




- Loss function: Mean Square Error:

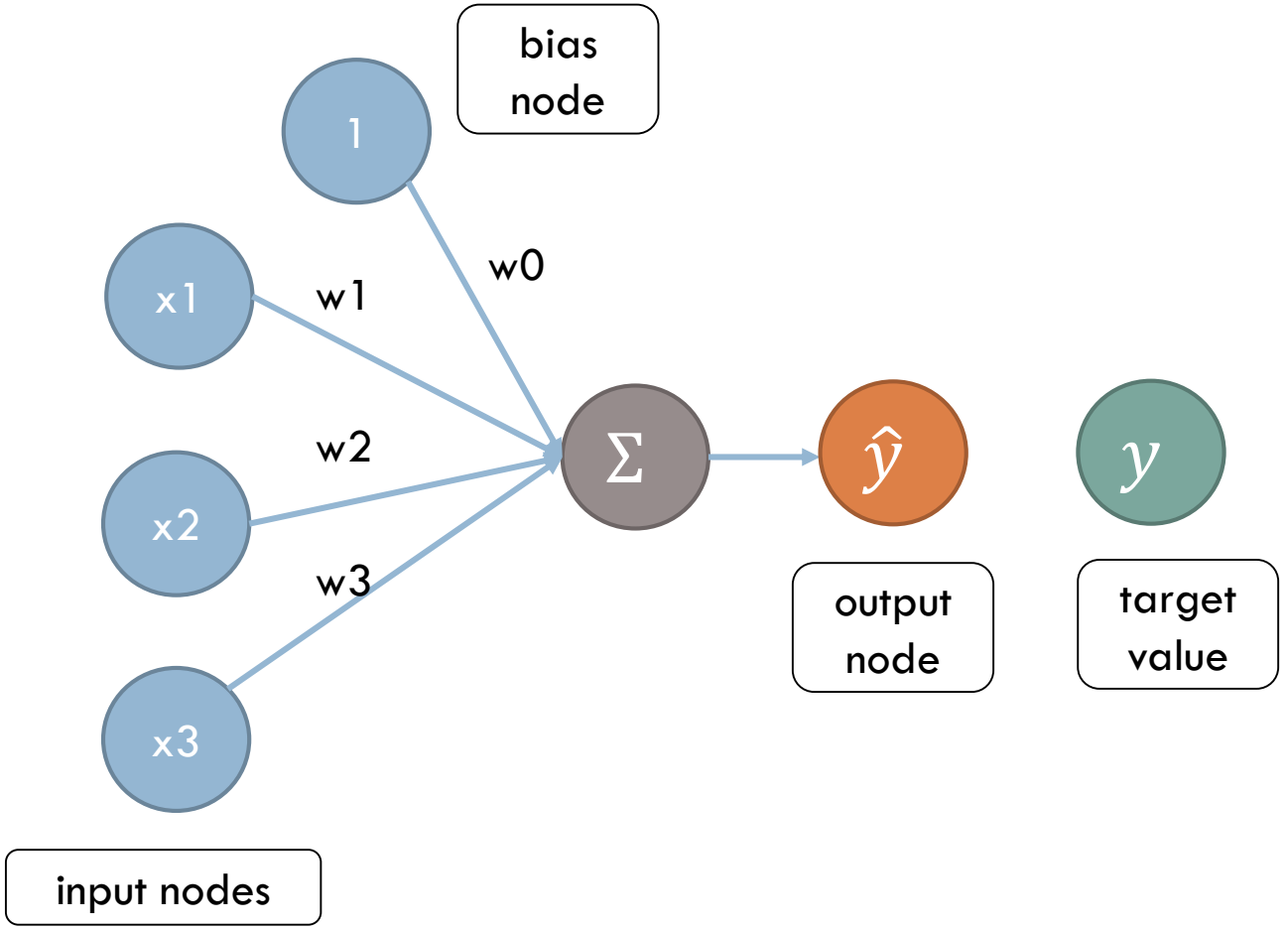
$$\frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

  - (average over all the *m* many observations)

# Linear regression as a network



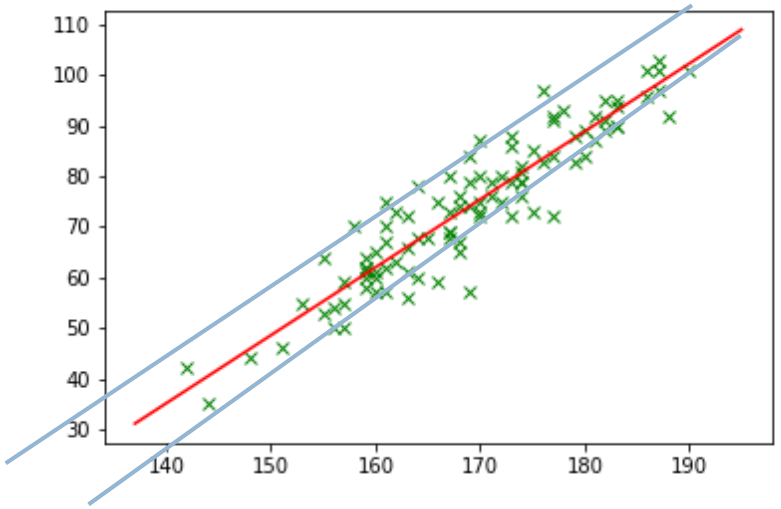
- Each feature, $x_i$, of the input vector is an input node
- An additional bias node $x_0 = 1$ for the intercept
- A weight at each edge,
- Multiply the input values with the respective weights: $w_i x_i$
- Sum them
- $\hat{y} = \sum_{i=0}^{m} w_i x_i = \boldsymbol{w} \cdot \boldsymbol{x}$

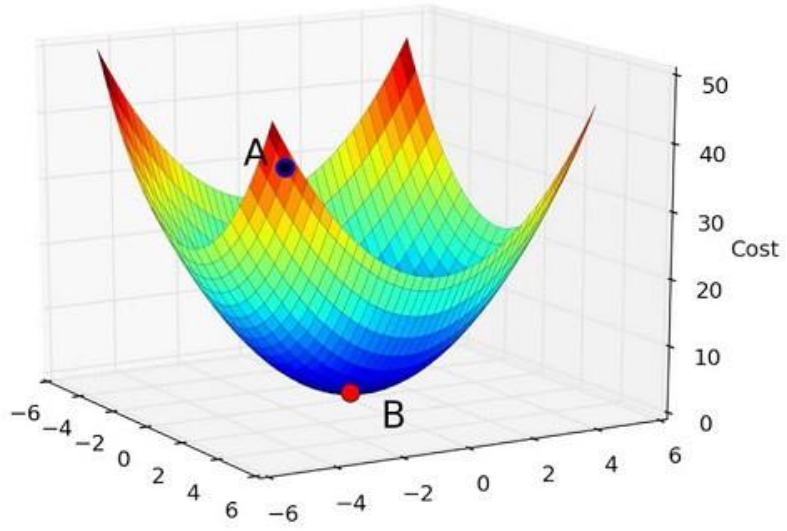


# Gradient descent



- Start with one candidate line, i.e., an initial set of weights
- Consider training examples
- Adjust the weights to reduce the loss

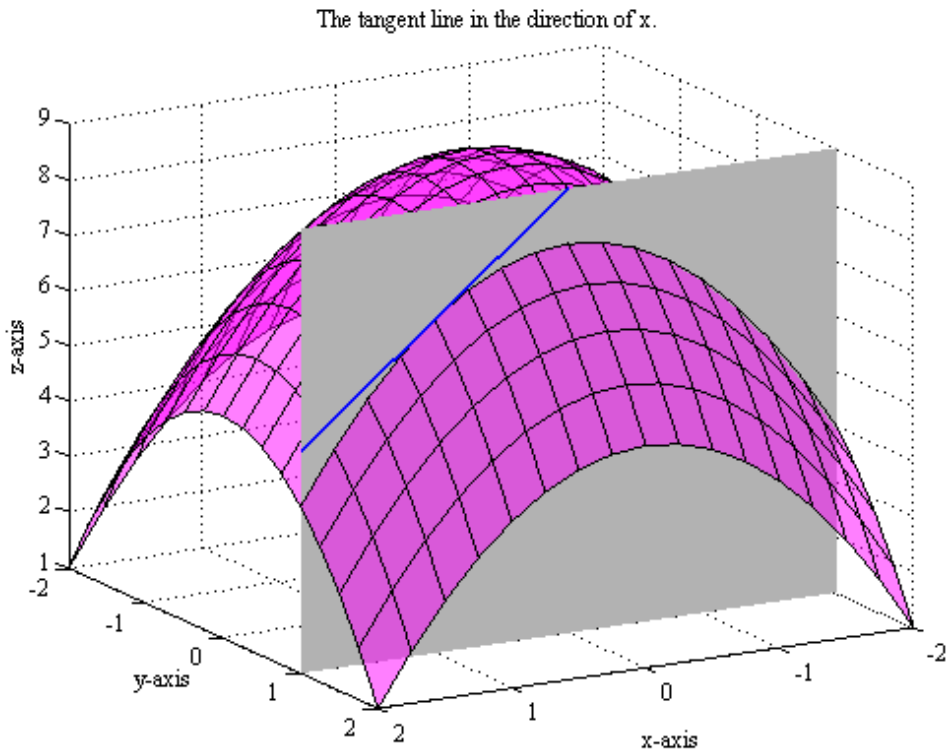


# Partial derivatives



- A function of more than one variable, e.g. $f(x, y)$
- The partial derivative, e.g. $\frac{\partial f}{\partial x}$ is the derivative one gets by keeping the other variables constant
- E.g. $f(x, y) = ax^2 + by^2 + c$, $\frac{\partial f}{\partial x} = 2ax$ and $\frac{\partial f}{\partial y} = 2by$

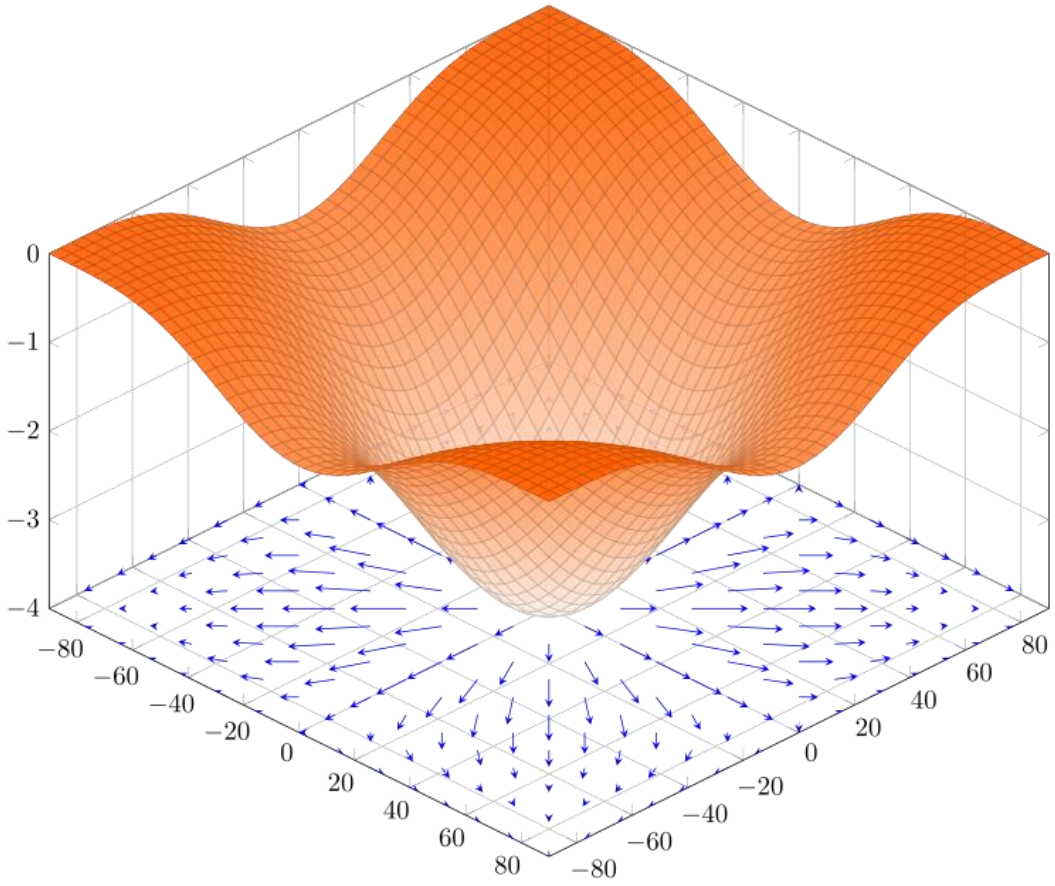



https://www.wikihow.com/Image:OyXsh.png

# Gradient descent



- We move in the opposite direction of where the gradient is pointing.
- Intuitively:
  - Take small steps in all direction parallel to the (feature) axes
  - The length of the steps are proportional to the steepness in each direction



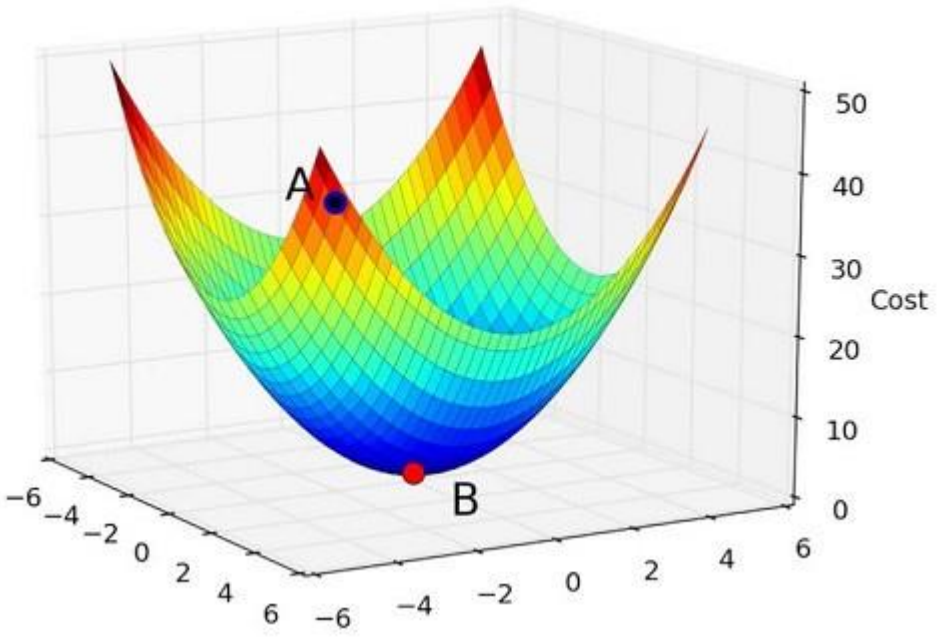
# Gradient descent (for linear regression)



- Loss: Mean squared error :
  - $L(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{n}\sum_{j=1}^{n}\left(\hat{y}_j - y_j\right)^2$
  - $\hat{y}_j = \sum_{i=0}^{m} w_i x_{j,i} = \boldsymbol{w} \cdot \boldsymbol{x}_j$

- $\frac{\partial}{\partial w_i} L(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{n}\sum_{j=1}^{n} 2\left(\hat{y}_j - y_j\right) x_{j,i}$

- Update $w_i$: $w_i = w_i - \eta \frac{\partial}{\partial w_i} L(\hat{\boldsymbol{y}}, \boldsymbol{y})$




$n$ is the number of observations,
$0 \le j \le n$ and
$m$ is the number of features for each observation,
$0 \le i \le m$

# Inspecting the update



$$w_i = w_i - \eta \frac{\partial}{\partial w_i} L(\hat{\boldsymbol{y}}, \boldsymbol{y})$$

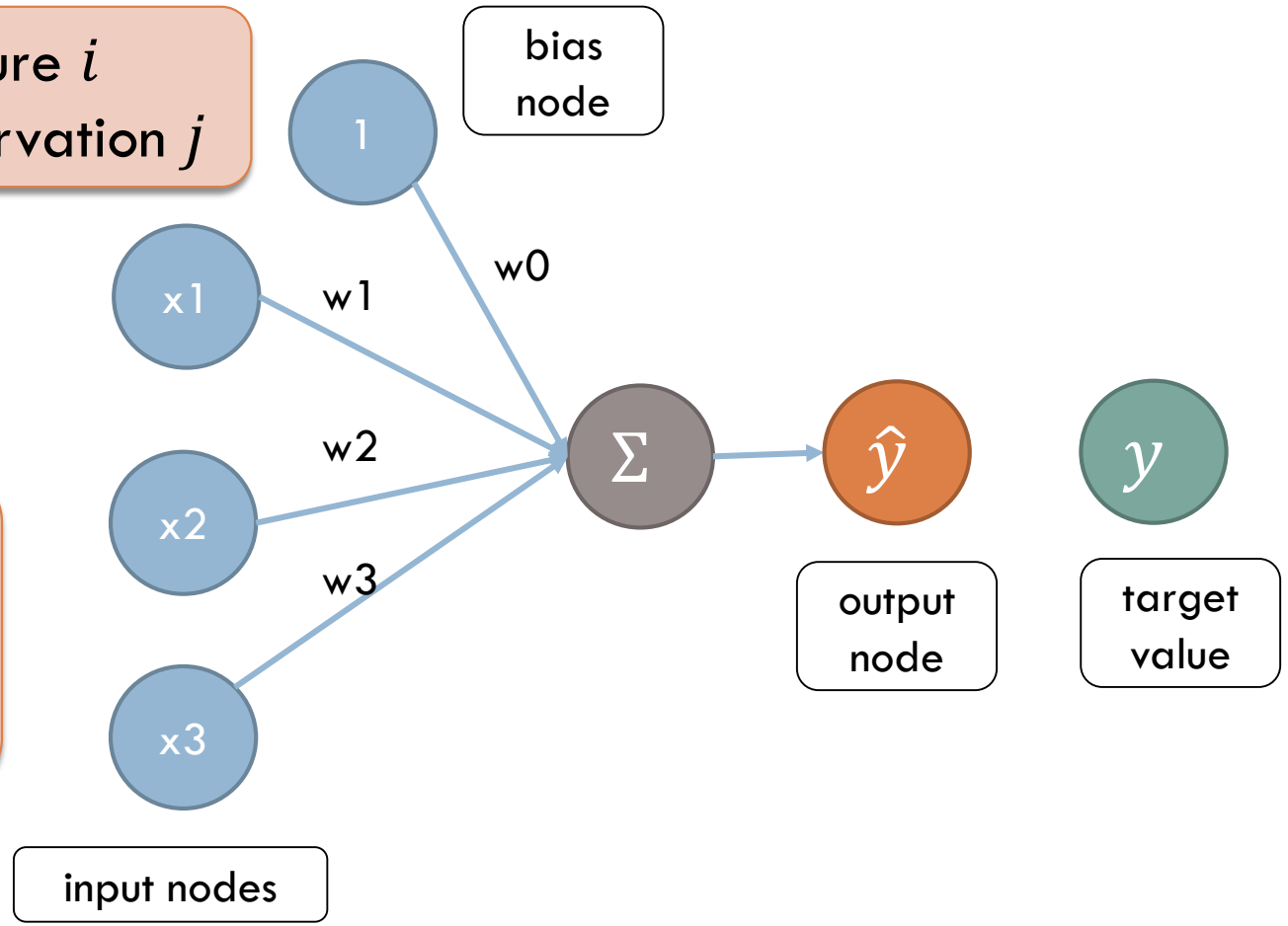
$$w_i = w_i - \eta \frac{1}{n} \sum_{j=1}^{n} 2\left(\hat{y}_j - y_j\right) x_{j,i}$$

Feature $i$ for observation $j$

The error term (delta term) of this prediction, from the loss function

The contribution to the error from this weight

$\eta$ is the learning rate



# Today

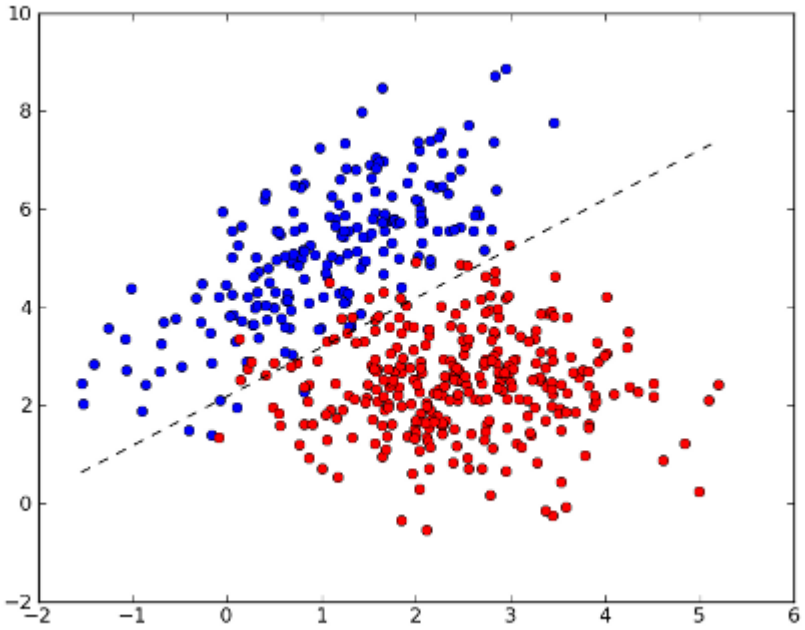


- POS-tagging
- HMM-tagging
- Discriminative tagging
- Logistic Regression (once more)
  - Linear regression
  - Logistic regression
  - Multinomial logistic regression (soft-max)
- Skip-gram with negative sampling

# Linear classifiers

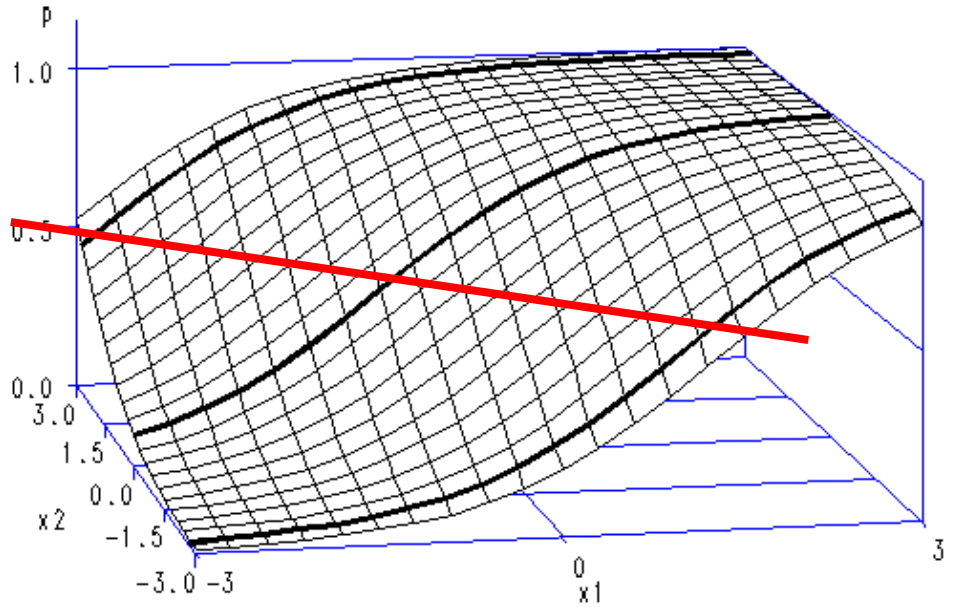


- Linear classifiers try to find a straight line that separates the two classes (in 2-dim)



# Logistic regression






From IDRE, UCLA

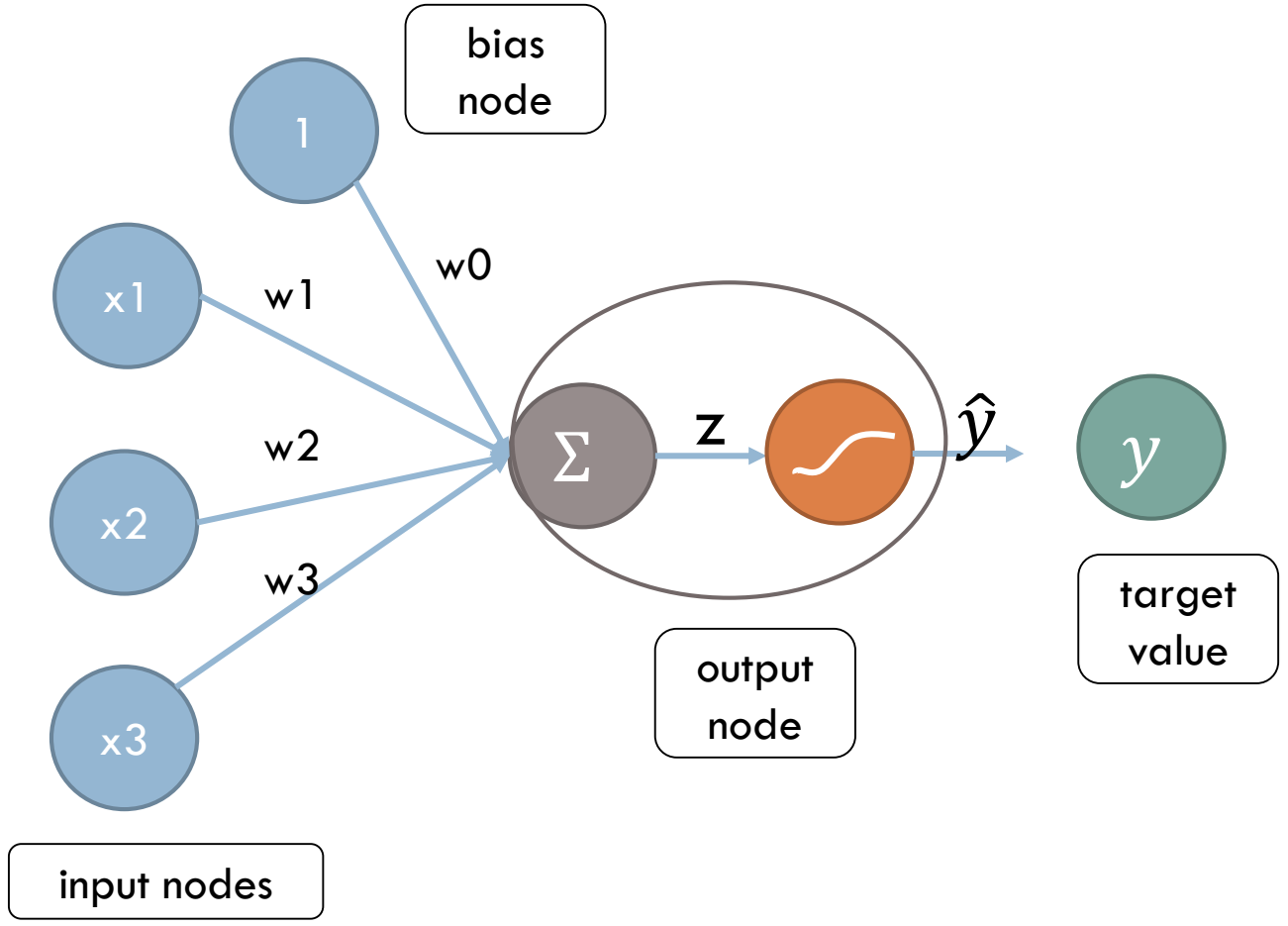
- Example with two features: $x_1, x_2$
- Apply weights: $w_0, w_1, w_2$
- Let $z = w_0 + w_1 x_1 + w_2 x_2$
- Apply the logistic function, $\sigma$, and check whether
  - $\hat{y} = \sigma(z) = \frac{1}{1+e^{-z}} > 0.5$

Geometrically:
Folding a plane along a sigmoid
The decision boundary is the intersection of this surface and the plane 0.5: a straight line

# Logistic regression as a network



- $z = \sum_{i=0}^{m} w_i x_i = \boldsymbol{w} \cdot \boldsymbol{x}$
- $\hat{y} = \sigma(z) = \frac{1}{1+e^{-z}}$
- The target value $y$ is 0 or 1
- The predicted value: $0 \leq \hat{y} \leq 1$



# Cross-entropy loss



- The underlying idea is that we want to maximize the joint probability of all the predictions we make
  - $\prod_{i=1}^{m} P\left(y^{(i)} \mid \vec{x}^{(i)}\right)$, over all the training data i = 1, 2, …m
- This is the same as maximizing
  - $\log \prod_{i=1}^{m} P\left(y^{(i)} \middle| \vec{x}^{(i)}\right) = \sum_{i=1}^{m} \log P\left(y^{(i)} \middle| \vec{x}^{(i)}\right)$
- This is the same as minimizing
  - $L_{CE}(\vec{w}) = -\log \prod_{i=1}^{m} P(y^{(i)} | \vec{x}^{(i)}) = \sum_{i=1}^{m} -\log P(y^{(i)} | \vec{x}^{(i)})$
  - Which is an instance of what is called the cross-entropy loss

# Warning



- You don't have to understand the next slide
- I have included it in case your are interested in how we find the gradient and the update
- What is important is the update on the next slide

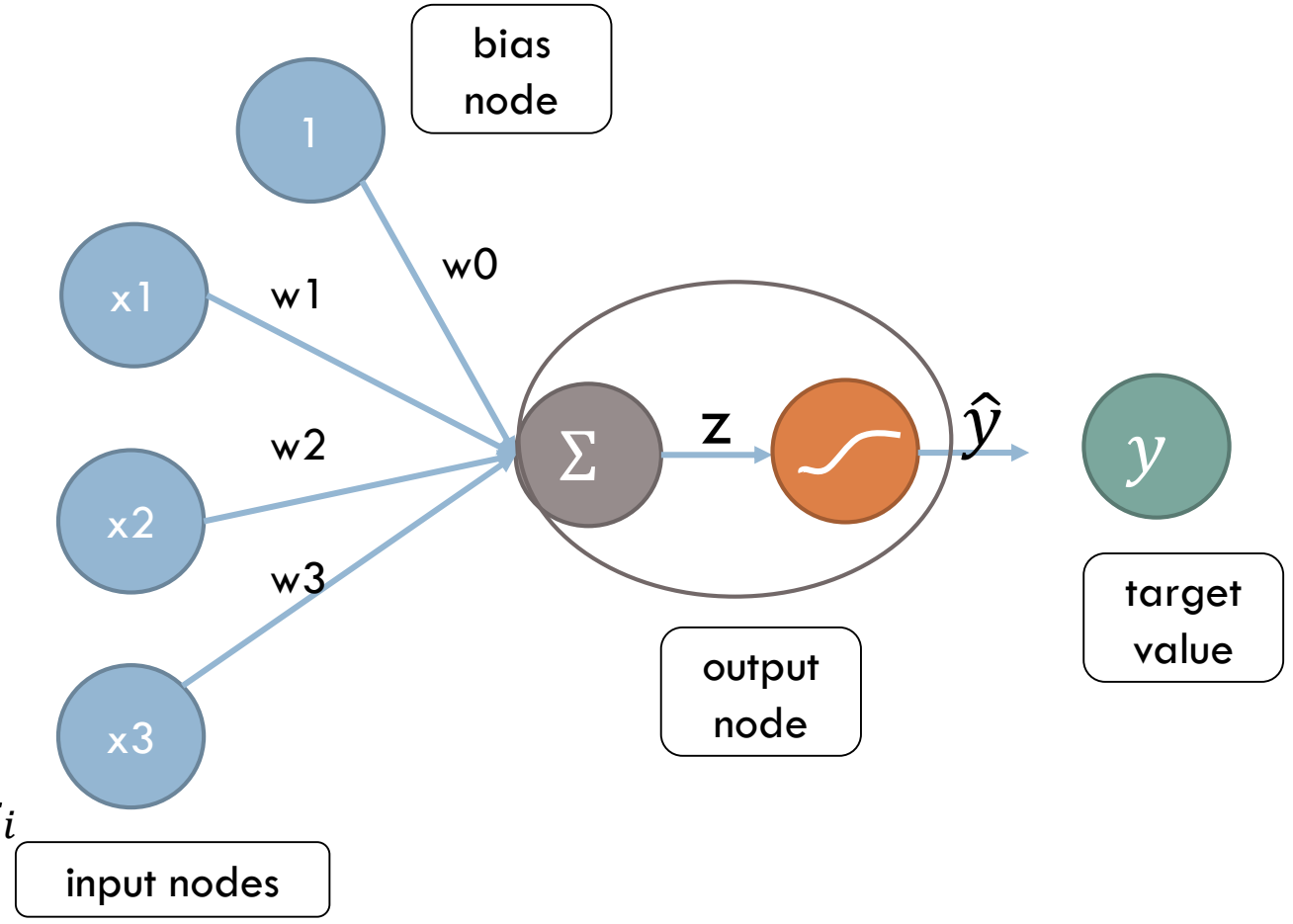


# The gradient



- $z = \sum_{i=0}^{m} w_i x_i = \boldsymbol{w} \cdot \boldsymbol{x}$
- $\hat{y} = \sigma(z) = \frac{1}{1+e^{-z}}$
- Loss: $L_{CE} = -\sum_{j=1}^{n} \log\left[\hat{y}_j^{\,y_j}\left(1-\hat{y}_j\right)^{(1-y_j)}\right]$
- $\frac{\partial}{\partial w_i} L_{CE} = \frac{\partial}{\partial \hat{y}} L_{CE} \times \frac{\partial \hat{y}}{\partial z} \times \frac{\partial z}{\partial w_i}$
- $\frac{\partial}{\partial \hat{y}} L_{CE} = -\frac{(y-\hat{y})}{\hat{y}(1-\hat{y})}$
- $\frac{\partial \hat{y}}{\partial z} = \hat{y}(1-\hat{y})$
- $\frac{\partial z}{\partial w_i} = x_i$
- $\frac{\partial}{\partial w_i} L_{CE} = -\frac{(y-\hat{y})}{\hat{y}(1-\hat{y})}\hat{y}(1-\hat{y})x_i = -(y-\hat{y})x_i$

To simplify, consider only one observation, $y_j$

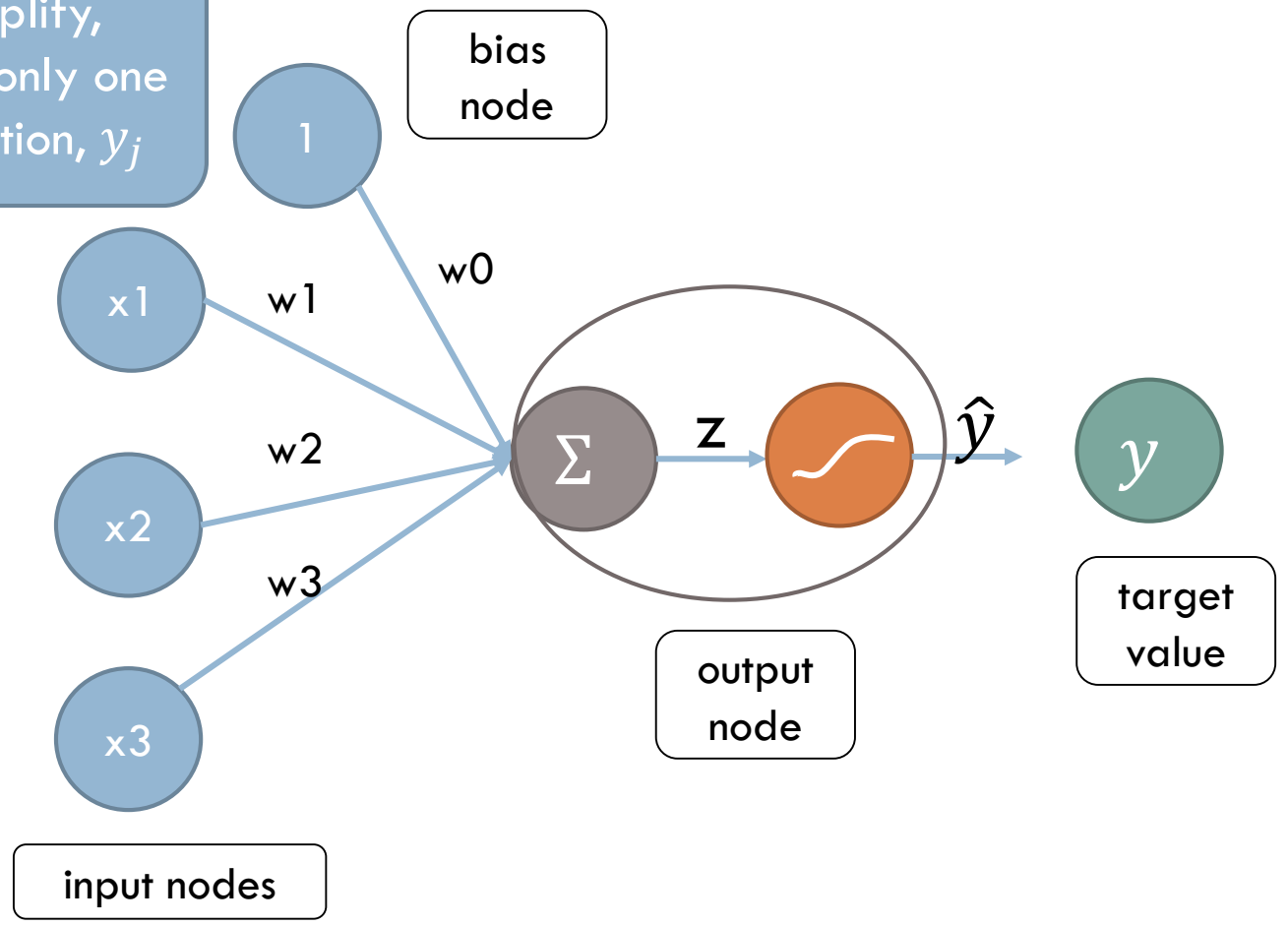


# The update



- $\frac{\partial}{\partial w_i} L_{CE} = -(y - \hat{y}) x_i$
- $w_i = w_i - \eta \frac{\partial}{\partial w_i} L_{CE} =$
  $w_i + \eta(y - \hat{y}) x_i$
- where
  - $y = 1$ or $y = 0$
  - $\hat{y} = \sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-\boldsymbol{w}\cdot\boldsymbol{x}}}$
  - $\eta$ is the learning rate

To simplify, consider only one observation, $y_j$



# Inspection



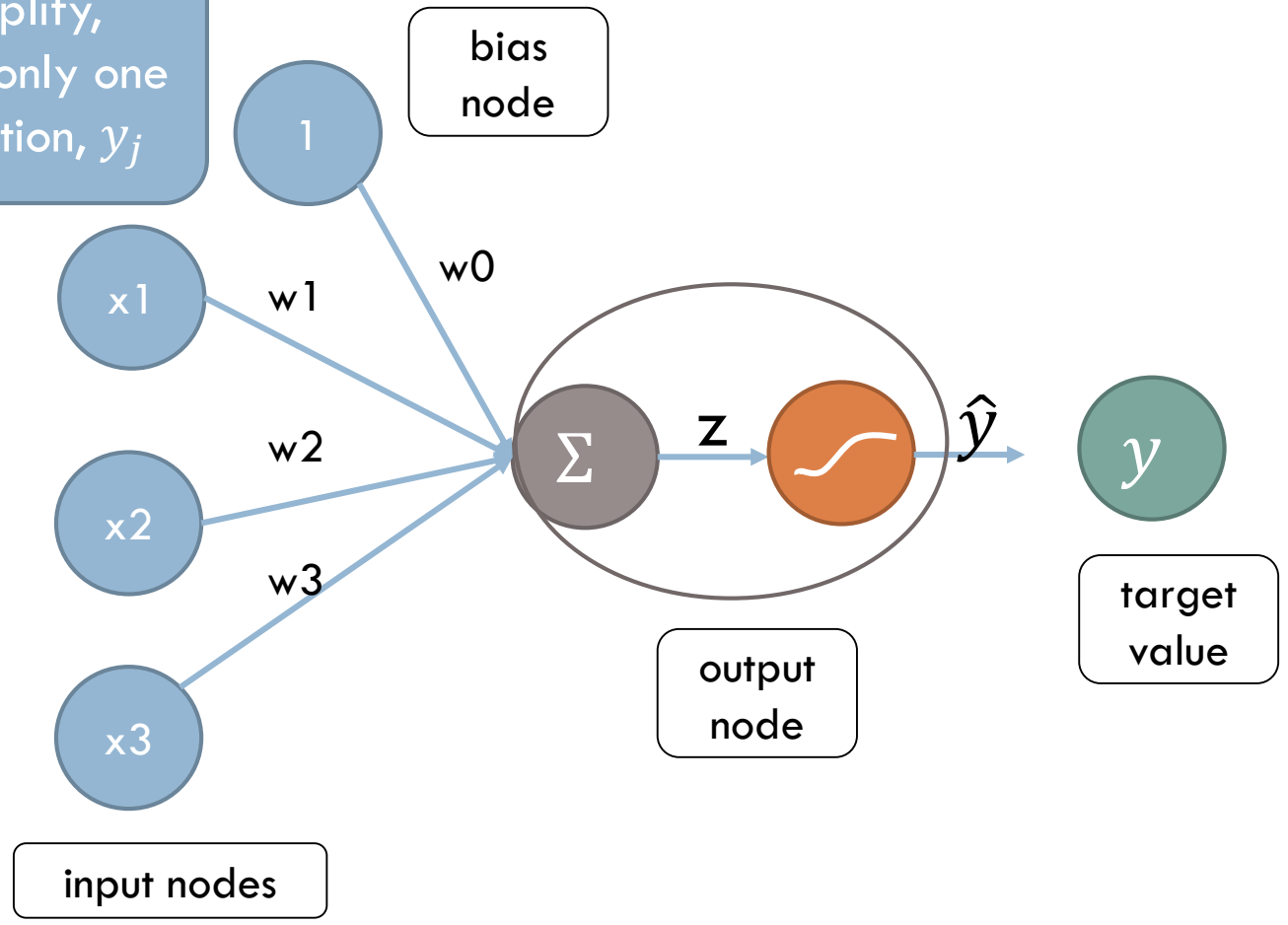
- $\frac{\partial}{\partial w_i} L_{CE} = -(y - \hat{y}) x_i$
- $w_i = w_i - \eta \frac{\partial}{\partial w_i} L_{CE} =$

$$w_i + \eta(y - \hat{y}) x_i$$

To simplify, consider only one observation, $y_j$

The error term (delta term) of this prediction, from the loss function

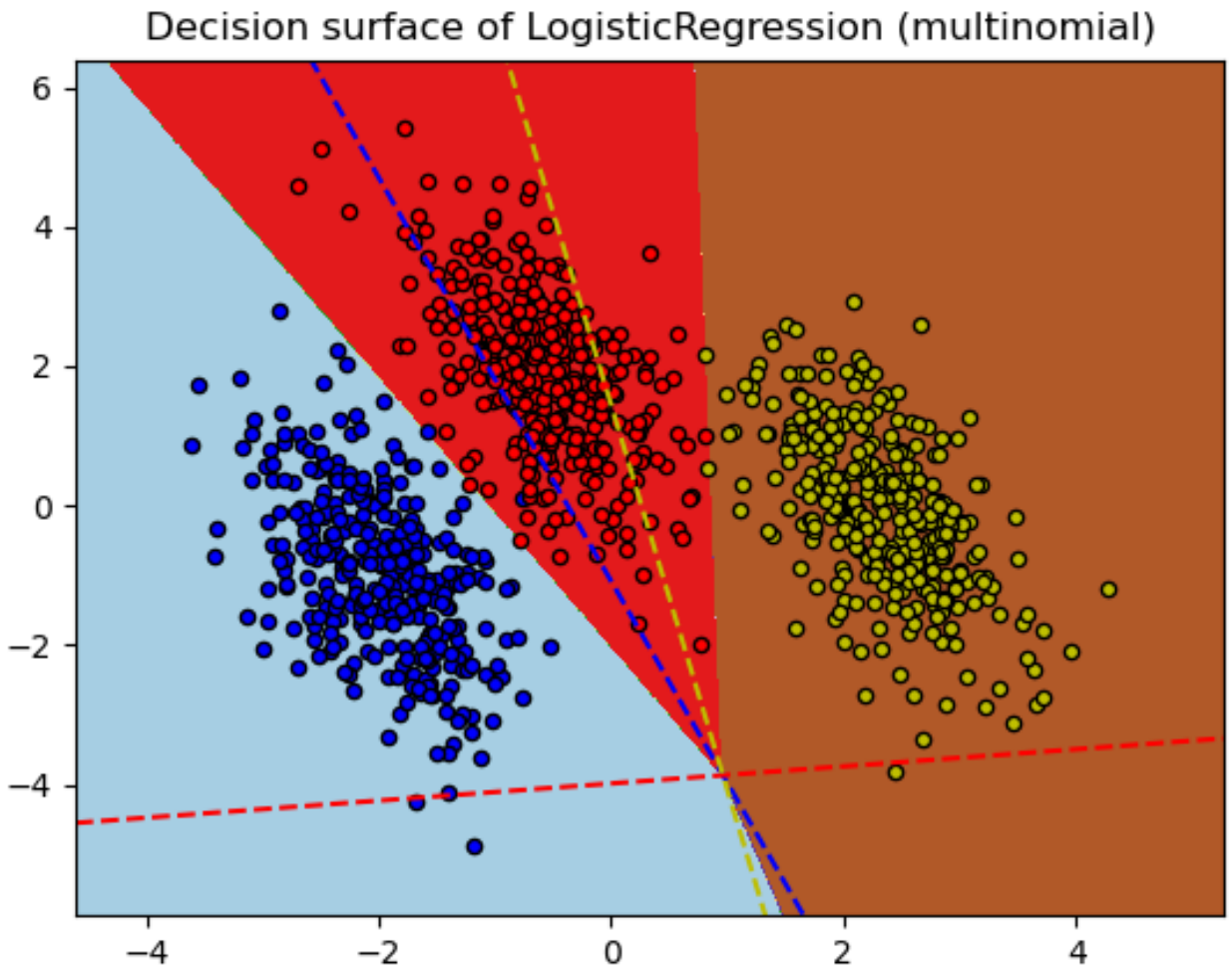
The contribution to the error from this weight



# Today



- POS-tagging
- HMM-tagging
- Discriminative tagging
- Logistic Regression (once more)
  - Linear regression
  - Logistic regression
  - Multinomial logistic regression (soft-max)
- Skip-gram with negative sampling




The decision boundaries turn out to be straight lines

https://scikit-learn.org/stable/auto_examples/linear_model/plot_logistic_multinomial.html

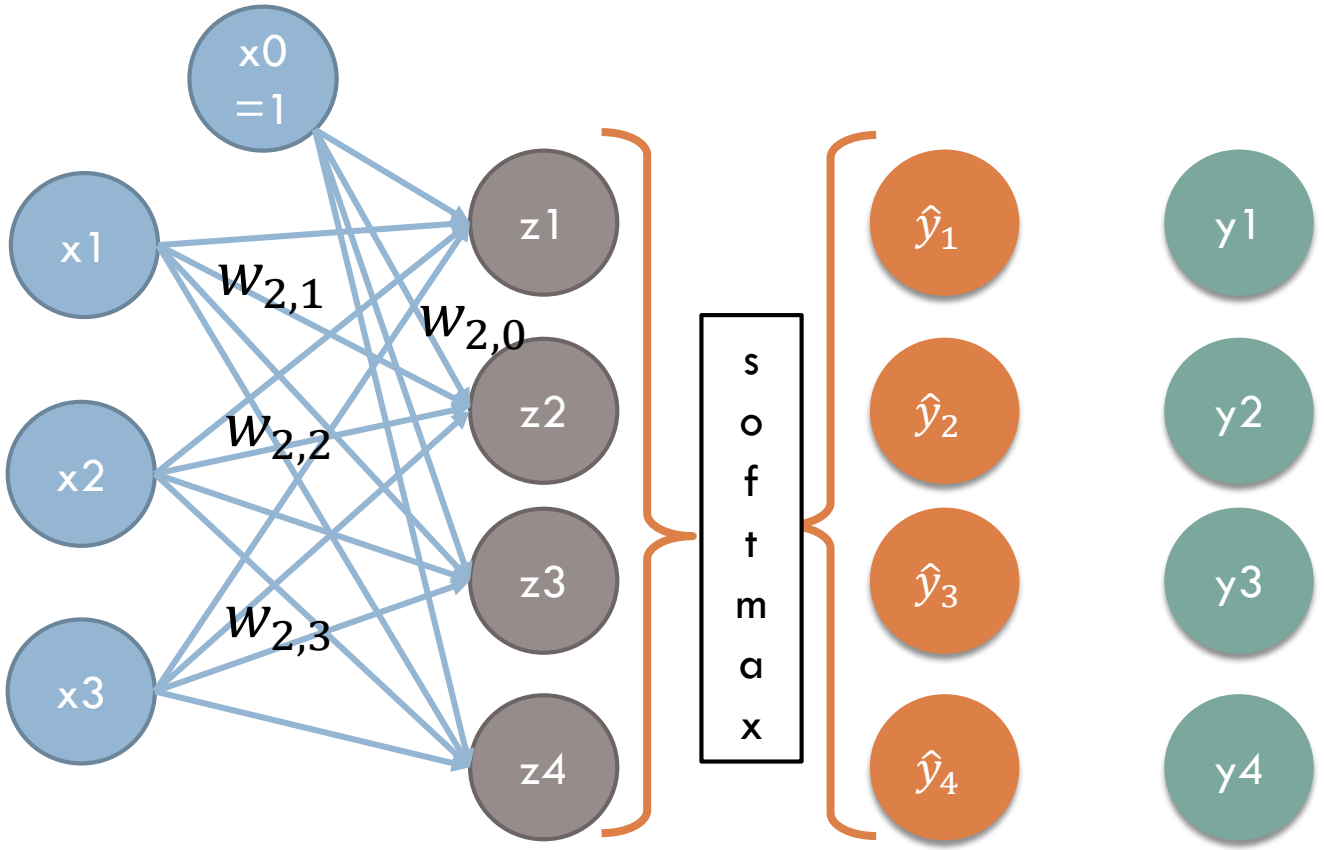
# Multinomial LogReg as a network



- $z_i = \sum_{j=0}^{n} w_{i,j} x_j$
- $w_{i,j}$ is the weight into node $i$ from node j
- $\hat{y}_i = (sm(z_1, \ldots, z_m))_i$

$$= \frac{e^{z_i}}{\sum_{k=1}^{m} e^{z_k}}$$

- Observe that $\hat{y}_i$ is a function of all the $z_k$-s




Beware: Jurafsky and Martin uses $w_{i,j}$ where Marsland, IN3050, uses $w_{j,i}$

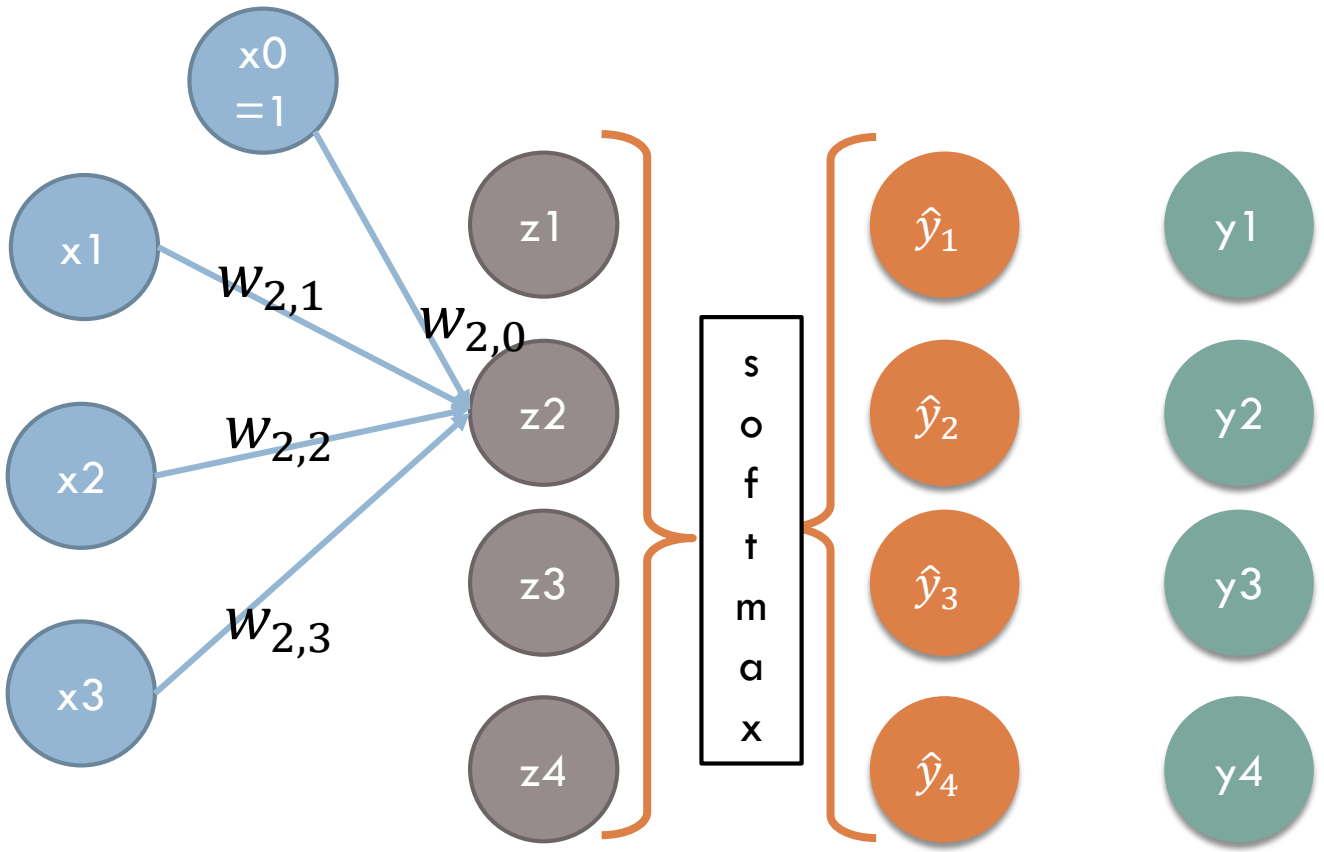
# Connections going into a node



- $z_i = \sum_{j=0}^{n} w_{i,j} x_j$
- $w_{i,j}$ is the weight into node $i$ from node j
- $\hat{y}_i = (sm(z_1, \ldots, z_m))_i$

$$= \frac{e^{z_i}}{\sum_{k=1}^{m} e^{z_k}}$$

- Observe that $\hat{y}_i$ is a function of all the $z_k$-s




Beware: Jurafsky and Martin uses $w_{i,j}$ where Marsland, IN3050, uses $w_{j,i}$

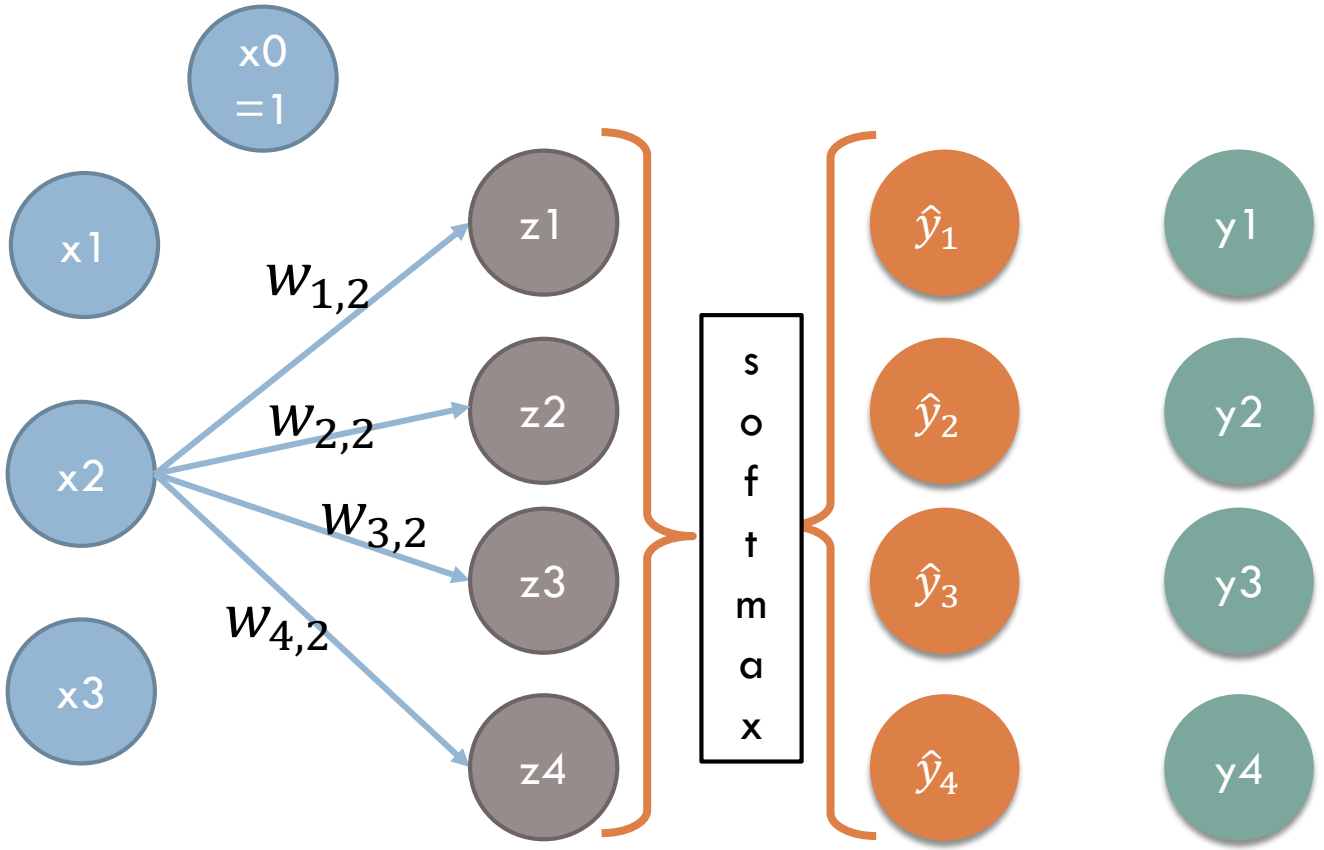
# Connections going out of a node



- $z_i = \sum_{j=0}^{n} w_{i,j} x_j$
- $w_{i,j}$ is the weight into node $i$ from node j
- $\hat{y}_i = (sm(z_1, \ldots, z_m))_i$

$$= \frac{e^{z_i}}{\sum_{k=1}^{m} e^{z_k}}$$

- Observe that $\hat{y}_i$ is a function of all the $z_k$-s



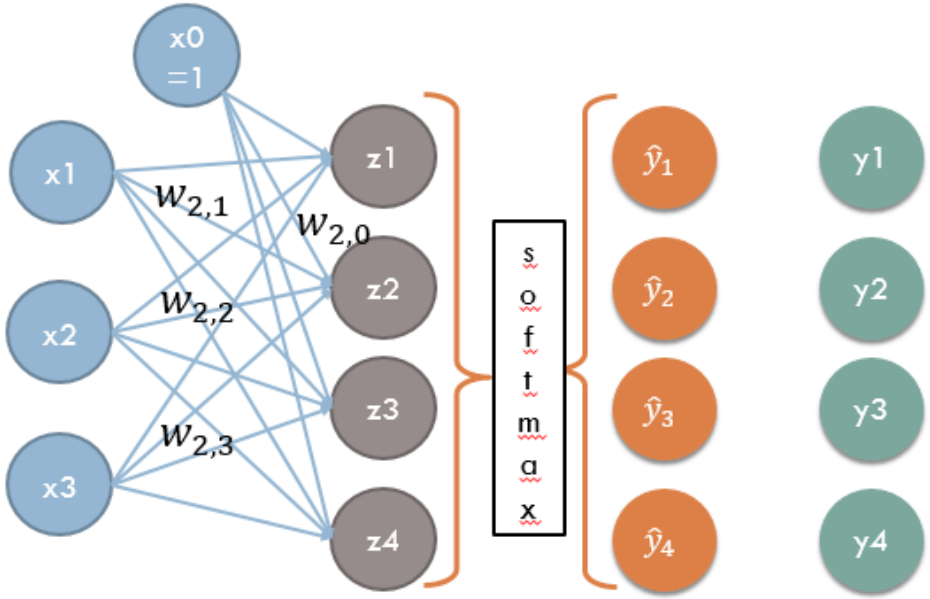

Beware: Jurafsky and Martin uses $w_{i,j}$ where Marsland, IN3050, uses $w_{j,i}$

# Matrix form



$$W\mathbf{x} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{m,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix} = \mathbf{z}$$

- For those of you who know matrices:
  - The connections between the layers: a matrix
  - Running it through the connections: matrix multiplication

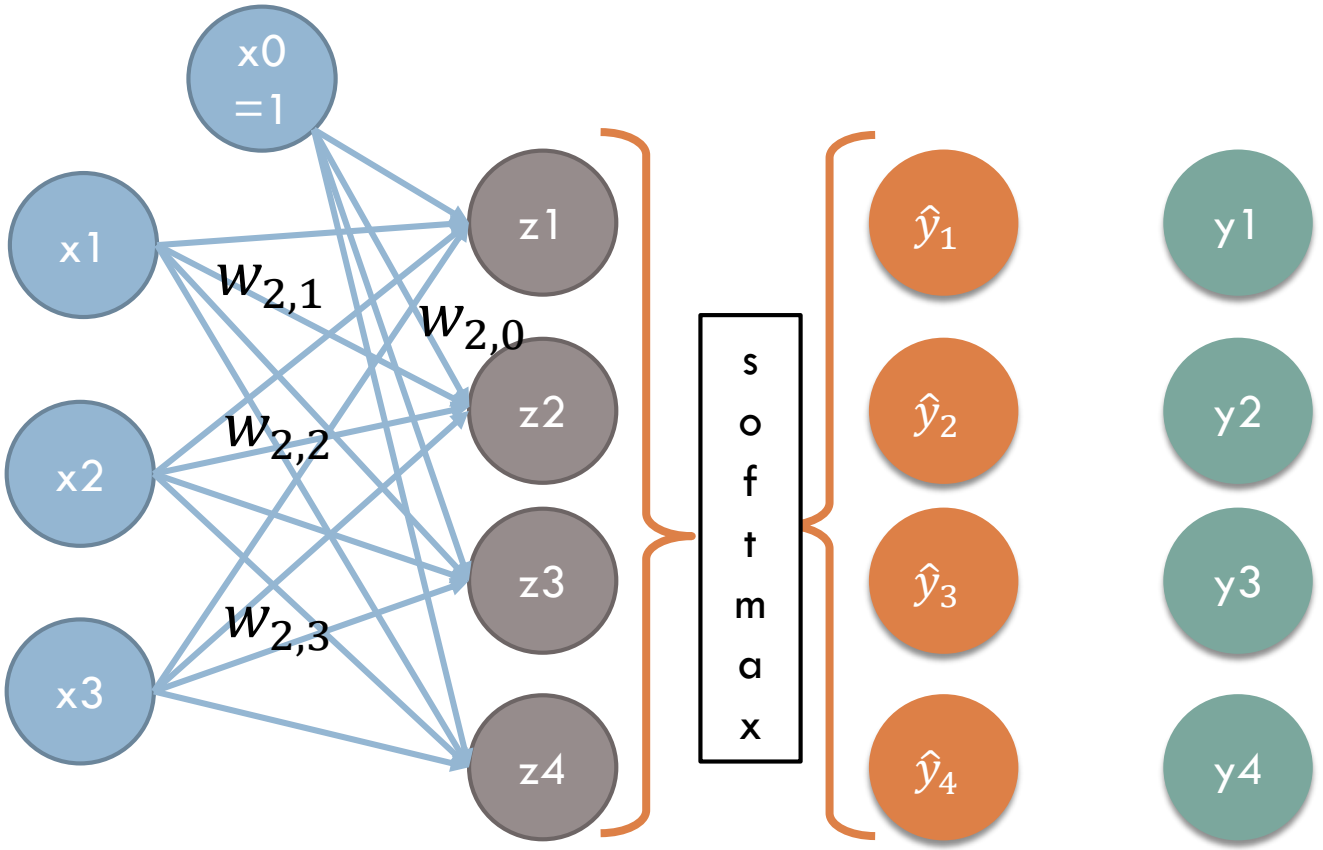



Beware: Jurafsky and Martin uses $w_{i,j}$ where Marsland, IN3050, uses $w_{j,i}$

# Training Multinomial LogReg



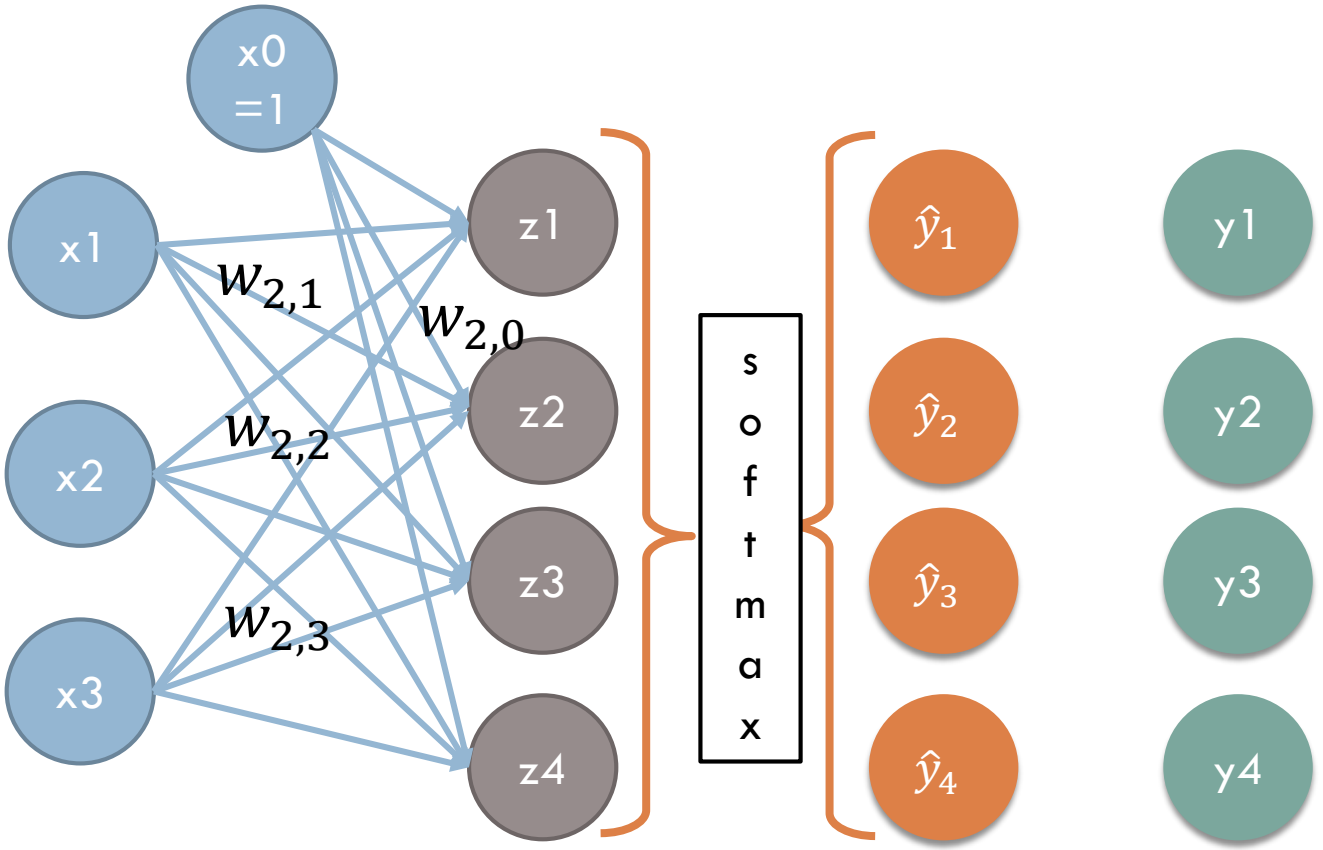
- The target has the form $(0,0,\ldots,0,1,0,\ldots,0)$,
  - say $\hat{y}_s = 1$ and $\hat{y}_j = 0$ for $j \neq s$
- We compare
  - $\hat{\mathbf{y}} = (\hat{y}_1, \hat{y}_2, .. \hat{y}_m)$
- to the target labels
  - $\mathbf{y} = (y_1, y_2, \ldots y_m)$
- using cross-entropy loss
  - $L_{CE}(\hat{\mathbf{y}}, \mathbf{y})$



# Training Multinomial LogReg



- $w_{i,j} = w_{i,j} + \eta(y_i - \hat{y}_i)x_j$
- which means
  - $w_{s,j} = w_{s,j} + \eta(1 - \hat{y}_s)x_j$
  - $w_{i,j} = w_{i,j} - \eta(\hat{y}_i)x_j$
    - for $j \neq s$
- where
  - $z_i = \sum_{j=0}^{n} w_{i,j}x_j$
  - $\hat{y}_i = \frac{e^{z_i}}{\sum_{k=1}^{m} e^{z_k}}$



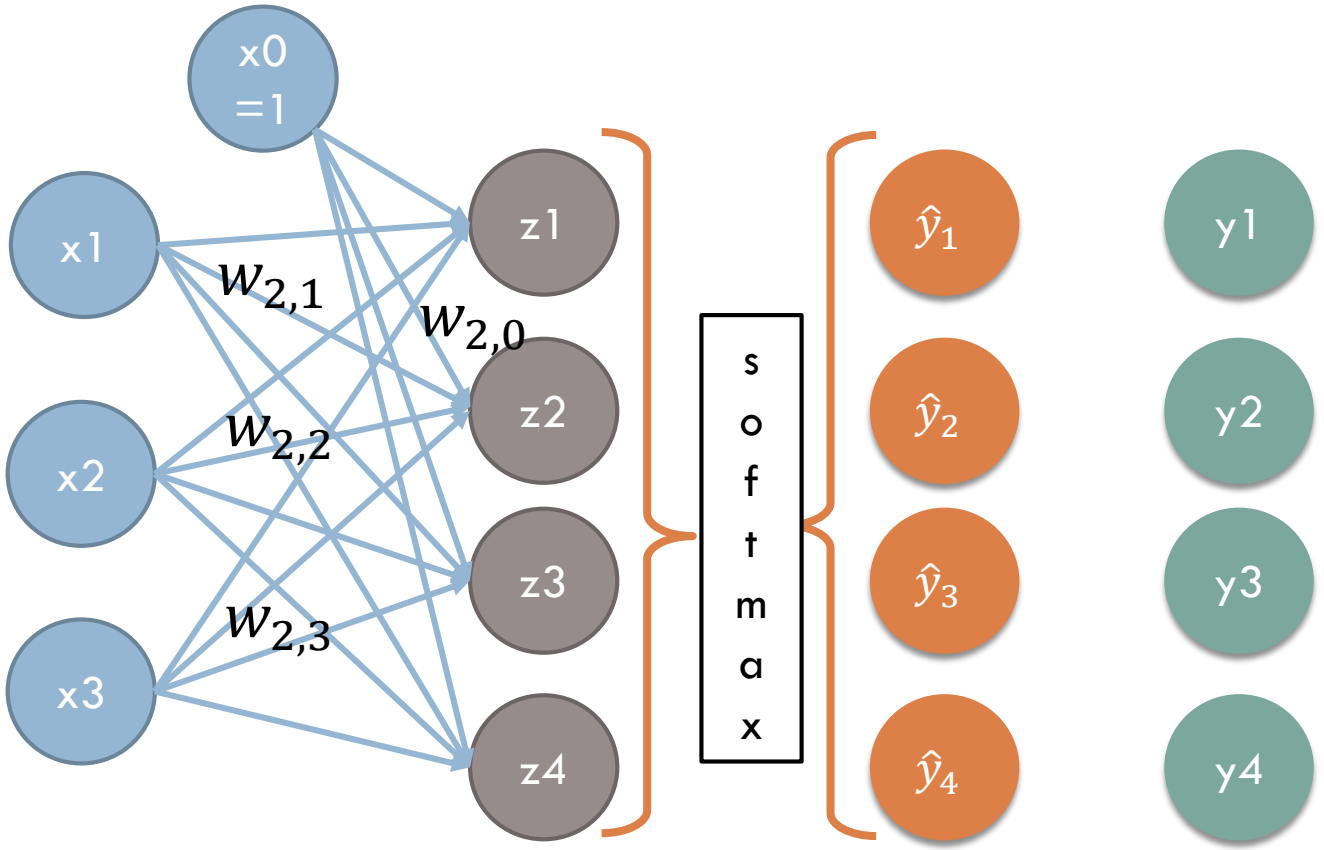
# Training Multinomial LogReg



- $w_{i,j} = w_{i,j} + \eta(y_i - \hat{y}_i)x_j$

The error term (delta term) of this prediction, at node $i$

The contribution to the error from this weight




- Observe that one observation will update all the weights

# Today

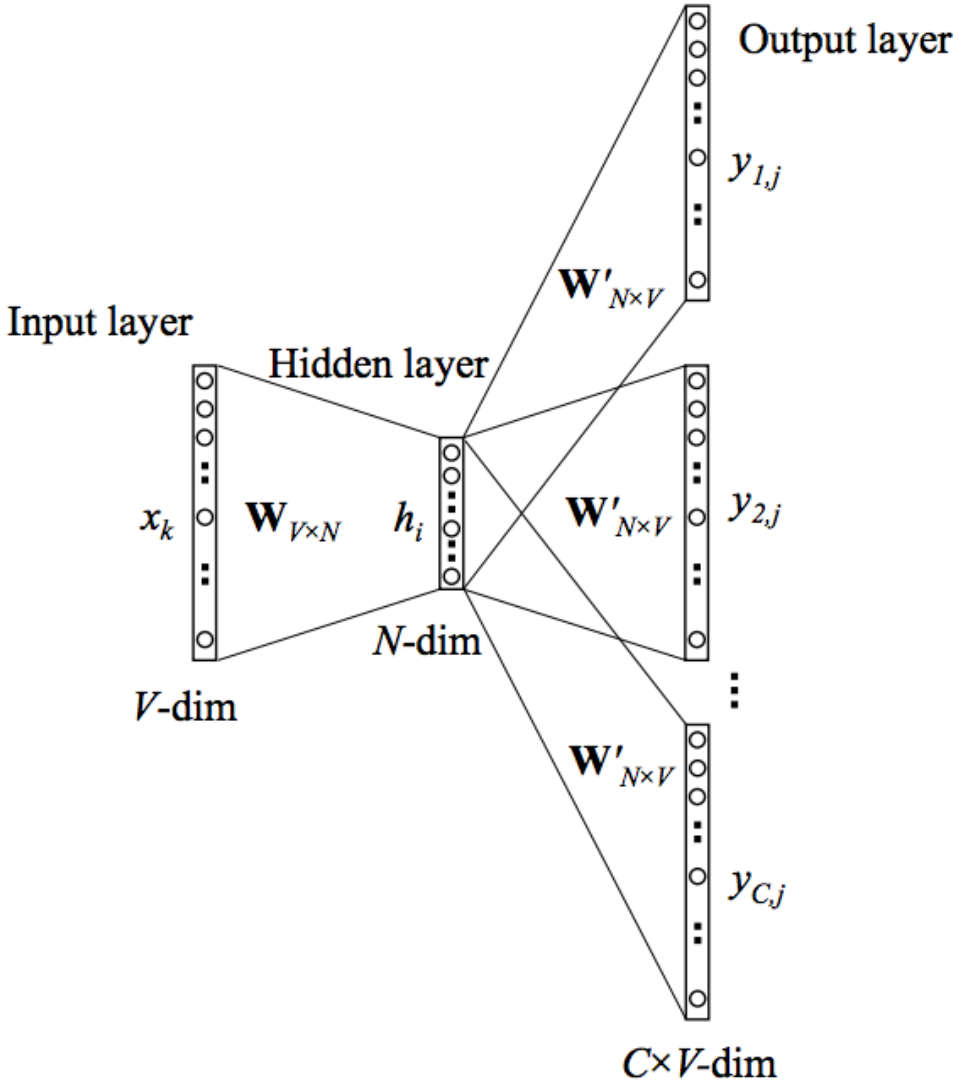


- POS-tagging
- HMM-tagging
- Discriminative tagging
- Logistic Regression (once more)
- Skip-gram with negative sampling

# Skip-gram



- From $w_t$ predict all the words in a window $(w_{t-k}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+k})$
- Assume independence of the context words, i.e. from $w_t$ predict each of the words w in $\{w_{t-k}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+k}\}$




https://commons.wikimedia.org/wiki/File:Skip-gram.png

# Skip-gram model



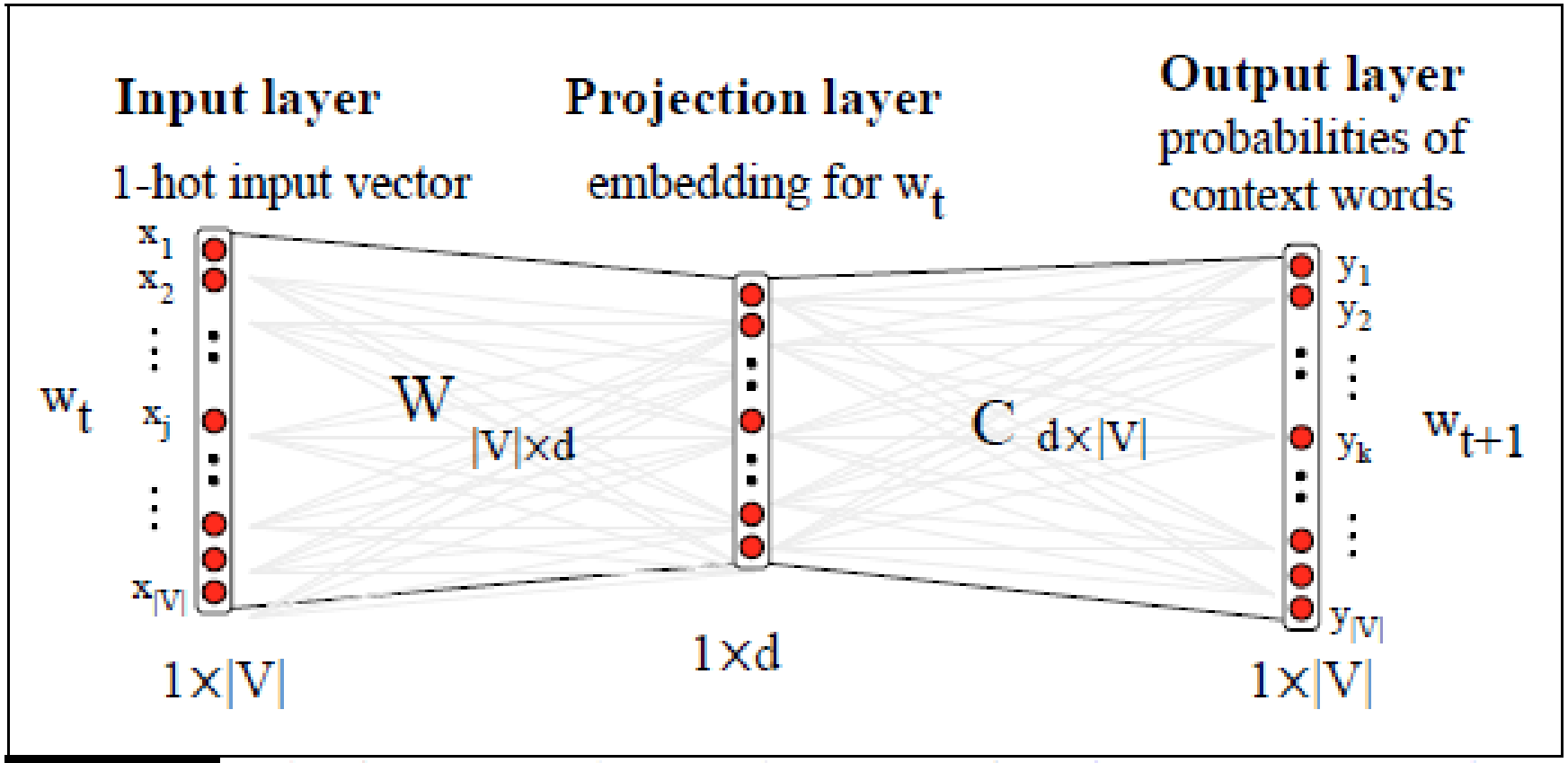



**Figure 16.5** The skip-gram model viewed as a network (Mikolov et al. 2013, Mikolov et al. 2013a).

From J&M 3.ed. 2018 Ch. 16

# Skip-gram with negative sampling



- To train a skip gram model is expensive
- Soft-max $P(C_j|\vec{x}) = \frac{e^{\overrightarrow{w_j}\cdot\vec{x}}}{\sum_{i=1}^{k} e^{\overrightarrow{w_i}\cdot\vec{x}}}$
  - where the classes corresponds to the next word
  - i.e. in making an update for a pair $(w_t, w_s)$ one has to calculate the weighted expression $e^{\overrightarrow{w_i}\cdot\vec{x}}$ for each word in the vocabulary
- Looking for cheaper training methods

# Skip-gram with negative sampling



1. Treat the target word and a neighboring context word as a positive example.
2. Randomly sample other words in the lexicon to get negative samples
3. Use logistic regression to train a classifier to distinguish those two cases
4. Use the weights as the embeddings

# Skip-Gram Training Data

- Training sentence:
- ... lemon, a tablespoon of apricot preserves or a ...
- c1 c2 t c3 c4

- Training data: input/output pairs centering on *apricot*
- Asssume a +/- 2 word window

# Skip-Gram Training Data



- ... lemon, a tablespoon of apricot preserves or a ...
- c1 c2 t c3 c4
- For each positive example, we'll create *k* negative examples.
  - Using *noise* words: Any random word that isn't *t*

| positive examples + | |
|---|---|
| t | c |
| apricot | tablespoon |
| apricot | of |
| apricot | preserves |
| apricot | or |

| negative examples - | | | |
|---|---|---|---|
| t | c | t | c |
| apricot | aardvark | apricot | twelve |
| apricot | puddle | apricot | hello |
| apricot | where | apricot | dear |
| apricot | coaxial | apricot | forever |

# How to compute p(+|t,c)?

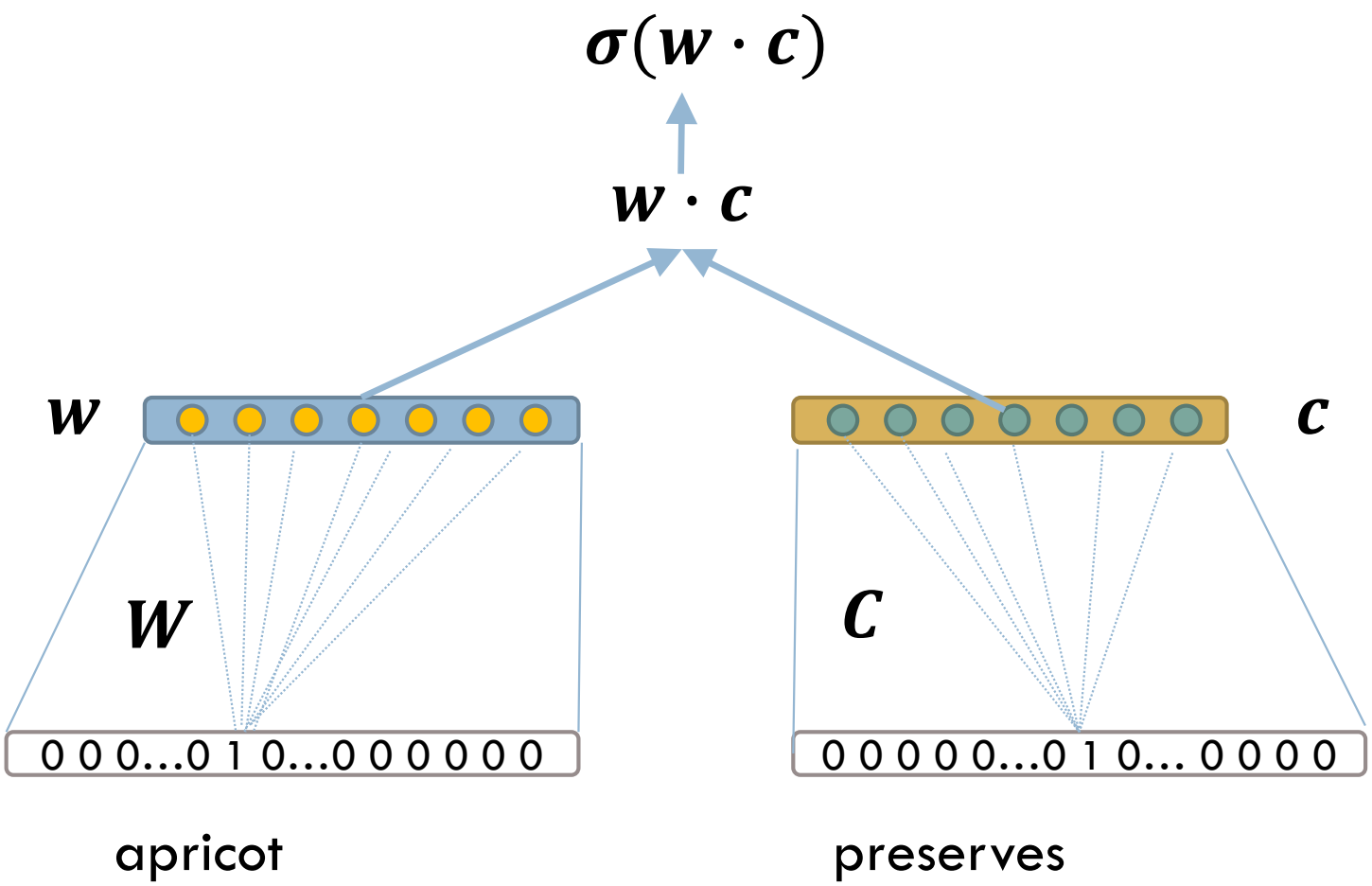
Word2vec

- One of various ways to train the classifier to distinguish pos and neg words

- Intuition:
  - Words are likely to appear near similar words
  - Model similarity with dot-product!
  - Similarity $(t, c) \sim t \cdot c$
- *Problem:*
  - *Dot product is not a probability!*
    - *(Neither is cosine)*

# Another view






- We feed a pair of words $(w, c)$ to distinct hidden embedding layers
- Compare to target (1 or 0)

# Turning dot products into probabilities

- Sim(w,c) $\approx w \cdot c$
- To turn this into a probability
- We'll use the sigmoid from logistic regression:

$$P(+|w,c) = \sigma(c \cdot w) = \frac{1}{1+\exp(-c \cdot w)}$$

$$P(-|w,c) = 1 - P(+|w,c)$$

$$= \sigma(-c \cdot w) = \frac{1}{1+\exp(c \cdot w)}$$

Slide from Jurafsky & Martin

# Goal

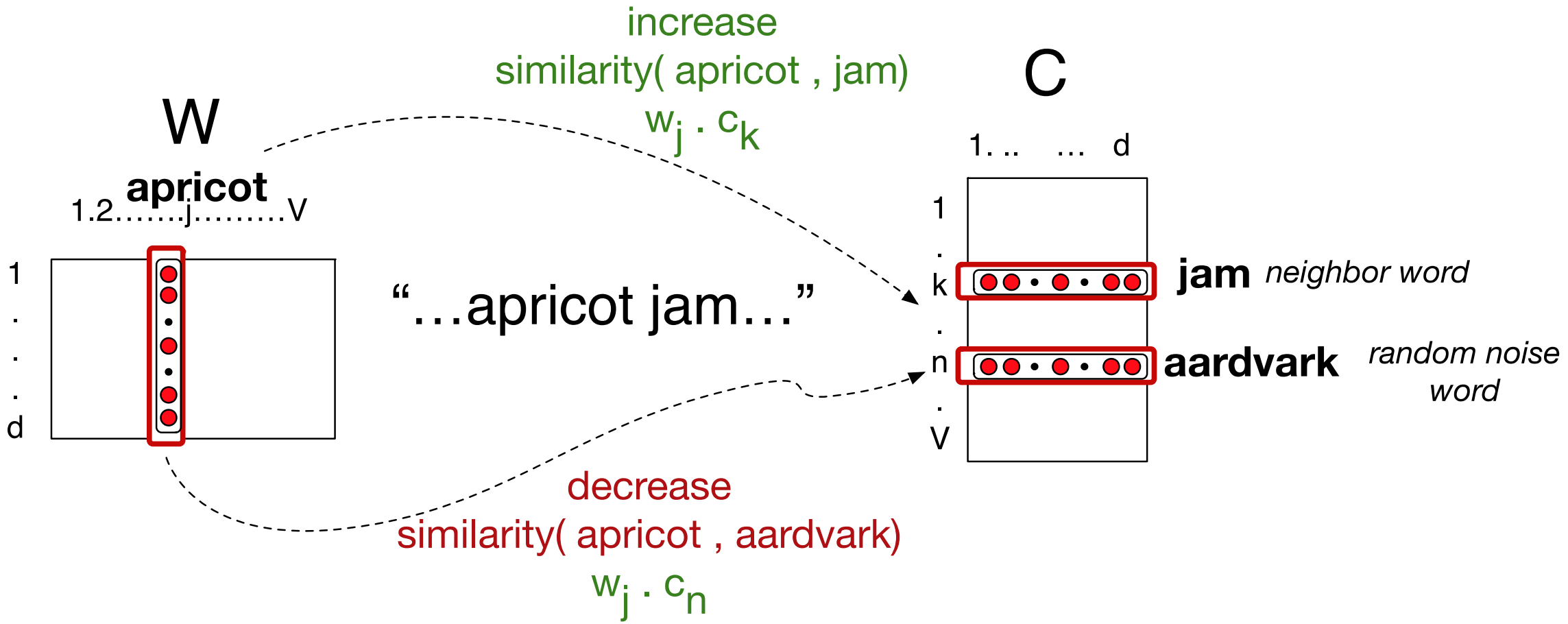


- Given a tuple (target, context)
  - (*apricot*, *jam*)
  - (*apricot*, *aardvark*)
  - (*apricot*, *twelve*)
- Calculate the probabilities
  - $P(+|t,c)$
  - $P(-|t,c) = 1 - P(+|t,c)$
- Maximize

$$\sum_{(t,c)\in+} logP(+|t,c) + \sum_{(t,c)\in-} logP(-|t,c)$$

  - where

$$P(+|t,c) = \frac{1}{1+e^{-t\cdot c}}$$


increase
similarity( apricot , jam)
$w_j \cdot c_k$
W
apricot
1.2.....j.....V
1
.
.
.
d
"…apricot jam…"
C
1. .. ... d
1
.
k
.
n
.
V
jam
neighbor word
aardvark
random noise word
decrease
similarity( apricot , aardvark)
$w_j \cdot c_n$

# Learning



- Like Multinomial Logistic Regression
- Start with randomly initialized weights for W and C
- For the training items (w, c), calculate $\hat{y} = \sigma(\boldsymbol{c} \cdot \boldsymbol{w}) = \frac{1}{1+e^{-\boldsymbol{c}\cdot\boldsymbol{w}}}$
- Compare to the gold labels using cross-entropy loss
  - The gold label is 1 if c is a context word and 0 if c is a negative example
  - This is like Multinomial Logistic regression
- Use the derivative of the loss with respect to **c:** $\frac{\partial}{\partial c} Lce$ to update **c**
- and the derivative of the loss with respect to **w** to update **w**

# Update equation in SGD



- We skip the derivation, but these are the resulting update equations

$$c_{pos}^{t+1} = c_{pos}^{t} - \eta[\sigma(c_{pos}^{t} \cdot w^{t}) - 1]w^{t}$$

$$c_{neg}^{t+1} = c_{neg}^{t} - \eta[\sigma(c_{neg}^{t} \cdot w^{t})]w^{t}$$

$$w^{t+1} = w^{t} - \eta\left[[\sigma(c_{pos} \cdot w^{t}) - 1]c_{pos} + \sum_{i=1}^{k}[\sigma(c_{neg_i} \cdot w^{t})]c_{neg_i}\right]$$

- Similar to the multinomial logistic regression, where we update weights
- Her we update both the $w$-s and the $c$-s.

# Result



- We learn two separate embedding matrices W and C
- We can use W as representations for the words
  - (or combine with C in some ways)

- What have we learned:
  - If two words *w1* and *w2* occur in similar contexts
    - = with the same (or similar) context words, e.g. c,
  - then both *w1* and *w2* should have a large cosine with c,
    - hence have similar vectors.

# Use of embeddings



- Embeddings are used as representations for words as input in all kinds of NLP tasks using deep learning:
  - Text classification
  - Language models
  - Named-entity recognition
  - Machine translation
  - etc.
- IN5550 Spring 2020

# Resources



- gensim
  - Easy-to-use tool for training own models
- Word2wec
  - https://code.google.com/archive/p/word2vec/
- https://fasttext.cc/
- https://nlp.stanford.edu/projects/glove/
- http://vectors.nlpl.eu/repository/
  - Pretrained embeddings, also for Norwegian