

Generalized Linear Models

CS-585

Natural Language Processing

Derrick Higgins

Statistical learning for classification

- Text categorization is a classification task
 - We want to associate every document with a class (label) using some statistical model
- We have seen one way to do this (naïve Bayes), but let's be more explicit about the problem setting

Statistical learning for classification

- A document is represented as a vector \vec{x} of features (typically, words or other lexical representations)
 - We use X to represent a random variable ranging over values of \vec{x}
- The label for each document is a categorical value $y \in Y$
 - We also use Y to represent a random variable ranging over values of y
- We are given a training set of labeled documents $(\vec{x}_1, y_1), (\vec{x}_2, y_2), \dots, (\vec{x}_N, y_N)$
- The task is to identify some scoring function $\Psi(\vec{x}, y; \Theta)$
 - Ψ tells us how compatible \vec{x} is with the label y
 - Θ is the set of parameters we can adaptively modify to change the function's behavior

Linear models

- Classification rule is

$$\hat{y} = \operatorname{argmax}_{y \in Y} \Psi(\vec{x}, y; \Theta)$$

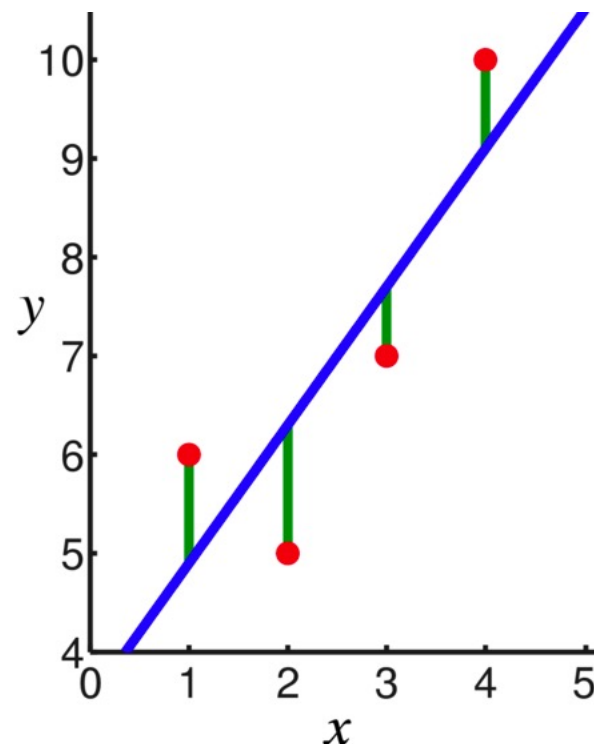
- In the case of a linear model,

$$\Psi(\vec{x}, y; \Theta) = \Theta^T f(\vec{x}, y)$$

- Typically, the feature vector $f(\vec{x}, y)$ represents each value of \vec{x} in combination with each class y .

Generalized linear models

- A linear model $\Psi(\vec{x}, y; \Theta) = \Theta^T f(\vec{x}, y)$ gives us a score for a given document, which is a real number. But in order to learn a good function, we need to turn this into a meaningful number that we can use for optimization
- In linear regression, the output of our linear function is itself our \hat{y} , and we can optimize based on the reconstruction error $\|\hat{y} - y\|_2$



https://en.wikipedia.org/wiki/Linear_regression

Generalized linear models

- A framework for associating linear functions with distributions for use in optimization/estimation
- Idea: define a *link function* g that determines the relationship between the score function Ψ (linear model) and the expected value of the distribution we are interested in

$$g(E[Y = y|X = \vec{x}]) = \Theta^T f(\vec{x}, y)$$

- For linear regression, g is the identity function, so the linear model provides our estimate directly

Generalized linear models

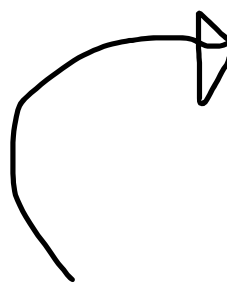
GLM Type	Link function	Uses
Linear regression	$g(E[\cdot]) = E[\cdot]$	Estimate real-valued quantity
Poisson regression	$g(E[\cdot]) = \ln(E[\cdot])$	Estimate number of events in fixed time window
Logistic regression	$g(E[\cdot]) = \ln\left(\frac{E[\cdot]}{1 - E[\cdot]}\right)$	Estimate the probability of a binary outcome variable (Bernoulli distribution)

Logistic regression

For logistic regression,

$$g(E[\cdot]) = \ln \left(\frac{E[\cdot]}{1 - E[\cdot]} \right) = \Theta^T f(\vec{x}, y)$$

...and the expectation is the estimate of a probability distribution over two outcomes, so we can write


$$\ln \left(\frac{P(Y = 1|X = \vec{x})}{1 - P(Y = 1|X = \vec{x})} \right) = \Theta^T f(\vec{x}, y)$$

$$1 - P(Y = 1|X = \vec{x}) = P(Y = 0|X = \vec{x})$$

"log odds"

Logistic regression

$$\ln \left(\frac{P(Y = 1|X = \vec{x})}{1 - P(Y = 1|X = \vec{x})} \right) = \Theta^T f(\vec{x}, y)$$

So the logit function is the link that transforms our probabilities into the linear output of our model.

What if we want to get from the linear outputs to the probability space?

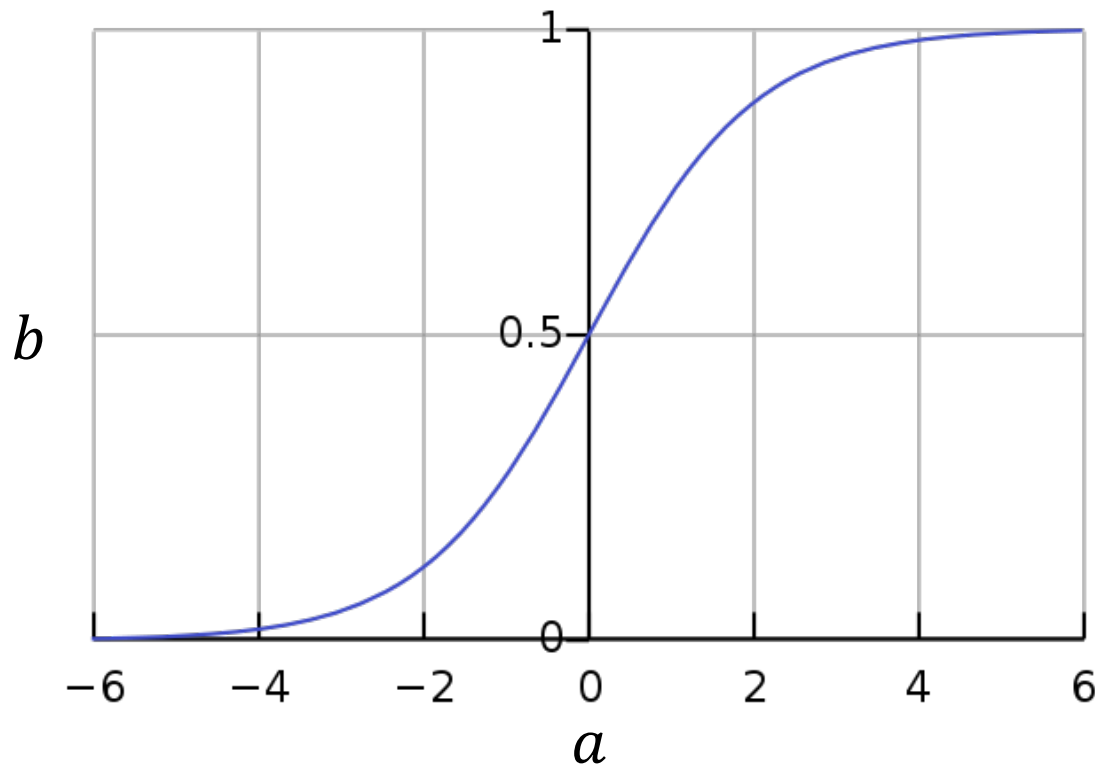
$$a = \ln \left(\frac{b}{1 - b} \right)$$

Inverse logit (or *sigmoid squashing function*)

$$P(Y = 1|X = \vec{x}) = \frac{e^{\Theta^T f(\vec{x}, y)}}{1 + e^{\Theta^T f(\vec{x}, y)}}$$

$$b = \frac{e^a}{1 + e^a}$$

Inverse logit / sigmoid



$$b = \frac{e^a}{1 + e^a}$$

$$P(Y = 1|X = \vec{x}) = \frac{e^{\Theta^T f(\vec{x}, y)}}{1 + e^{\Theta^T f(\vec{x}, y)}}$$

Inverse logit and softmax

Inverse logit maps from linear outputs to normalized probabilities for binary classification

$$P(Y = 1|X = \vec{x}) = \frac{e^{\Theta^T f(\vec{x}, y)}}{1 + e^{\Theta^T f(\vec{x}, y)}}$$

What if we have more than 2 classes? (Categorical LR)

Softmax:

$$P(Y = y|X = \vec{x}) = \frac{e^{\Theta^T f(\vec{x}, y)}}{\sum_{y' \in Y} e^{\Theta^T f(\vec{x}, y')}}, Y = \{y_1, y_2, \dots, y_N\}$$

... remember this from word2vec?

Logistic regression parameters

 Θ

computer
print
drive
sale
buy
cash
find
load

2.1	-1.1	0.1	-1.8
1.3	-1.2	-1.1	-2.0
0.5	0.0	2.3	1.3
0.1	1.2	-0.9	1.1
0.2	1.3	0.3	0.8
-0.2	4.2	-1.3	0.7
0.2	1.1	0.5	-0.1
-0.3	-1.2	0.2	2.0

Every feature in $Y \times V$ has its own element in Θ . For convenience, we refer to it as a vector rather than a matrix

 $f(\vec{x}, y)$

computers	for sale	automobiles	shopping
0	0	0	0
0	1	0	0
0	1	0	0
0	0	0	0
0	1	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Vectorized representation of document; zeros for $y \neq y^*$

Likelihood

- So for categorical logistic regression, the probability of an observed label y given the features/words x is

$$P(Y = y|X = \vec{x}) = \frac{e^{\Theta^T f(\vec{x}, y)}}{\sum_{y' \in Y} e^{\Theta^T f(\vec{x}, y')}}$$

- If we aggregate this over all of our training examples, we get the *likelihood*:

$$L(\Theta) = \prod_{i=1}^N P(y_i | \vec{x}_i; \Theta)$$

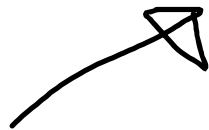
- One way of choosing parameters Θ is trying to find the value of Θ that maximizes $L(\Theta)$: the *maximum likelihood estimate*

Maximum likelihood estimation

Find

$$\begin{aligned}\operatorname{argmax}_{\Theta} L(\Theta) &= \operatorname{argmax}_{\Theta} \prod_{i=1}^N P(y_i | \vec{x}_i; \Theta) \\ &= \operatorname{argmax}_{\Theta} \prod_{i=1}^N \frac{e^{\Theta^T f(\vec{x}_i, y_i)}}{\sum_{y' \in Y} e^{\Theta^T f(\vec{x}_i, y')}} \\ &= \operatorname{argmax}_{\Theta} \sum_{i=1}^N \left(\Theta^T f(\vec{x}_i, y_i) - \log \sum_{y' \in Y} e^{\Theta^T f(\vec{x}_i, y')} \right)\end{aligned}$$

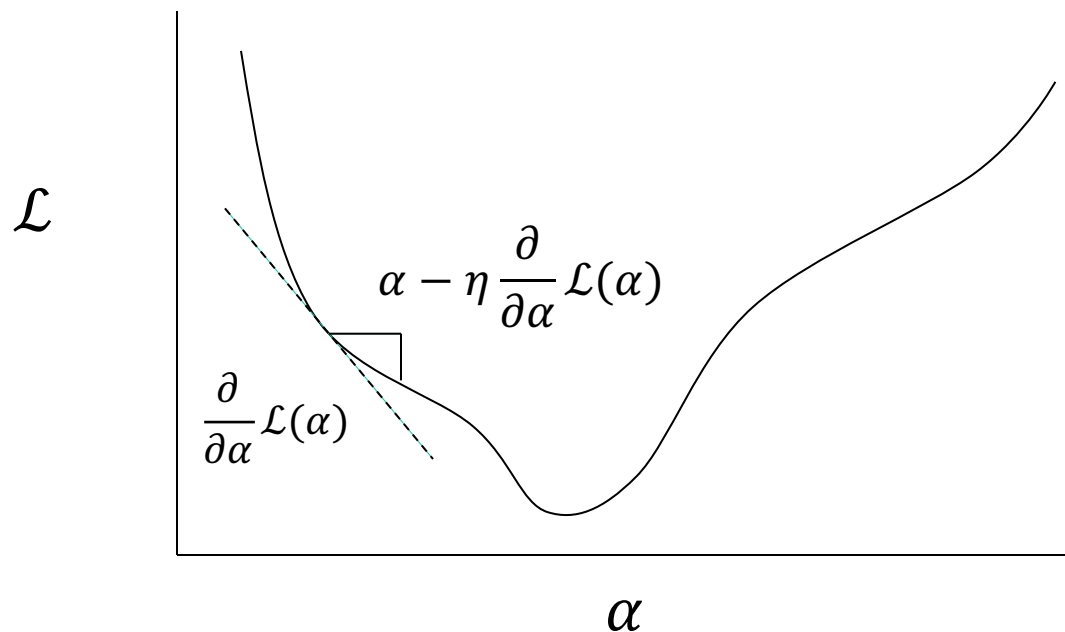
"log likelihood"



Optimization

- How do we actually find the parameter values Θ that maximize

$$\sum_{i=1}^N \left(\Theta^T f(\vec{x}_i, y_i) - \log \sum_{y' \in Y} e^{\Theta^T f(\vec{x}_i, y')} \right)?$$



Remember
gradient descent?

Gradient descent

1. Define a loss function \mathcal{L} to be minimized

- Log likelihood: $\log L(\Theta)$

2. Calculate the derivatives of the loss with respect to its inputs

$$\frac{\partial}{\partial \Theta} \left(\Theta^T f(\vec{x}_i, y_i) - \log \sum_{y' \in Y} e^{\Theta^T f(\vec{x}_i, y')} \right) = f(\vec{x}_i, y_i) \sum_{y \in Y} P(y|\vec{x}_i) f(\vec{x}_i, y)$$

3. Use the chain rule of differentiation to get the derivatives of the loss for all parameters in the model

- $\frac{\partial}{\partial \Theta} \log L(\Theta)$ is all we need

4. Update all parameters in the direction of the negative gradient

Gradient descent

- Batch gradient descent: accumulate updates across entire dataset before applying

$$\Theta_{t+1} \leftarrow \Theta_t - \eta \sum_{i=1}^{|D|} \nabla \log L(\Theta; \vec{x}_i; y_i) \quad \text{slow!}$$

- Stochastic gradient descent: update parameters after gradient calculated for each training exemplar

$$\Theta_{t+1} \leftarrow \Theta_t - \eta \nabla \log L(\Theta; \vec{x}_i; y_i) \quad \text{fast!}$$

but less stable

- Minibatch: Process updates for smaller samples of dataset (16-1024)

$$\Theta_{t+1} \leftarrow \Theta_t - \eta \sum_{i=1}^n \nabla \log L(\Theta; \vec{x}_i; y_i) \quad \text{faster, more stable}$$

More complex optimization

- Gradient descent:

$$\Theta_{t+1} \leftarrow \Theta_t - \eta \nabla \log L(\Theta)$$

- Momentum:

- Preferentially continue parameter search in direction of previous gradients; accelerate in directions of consistent updates

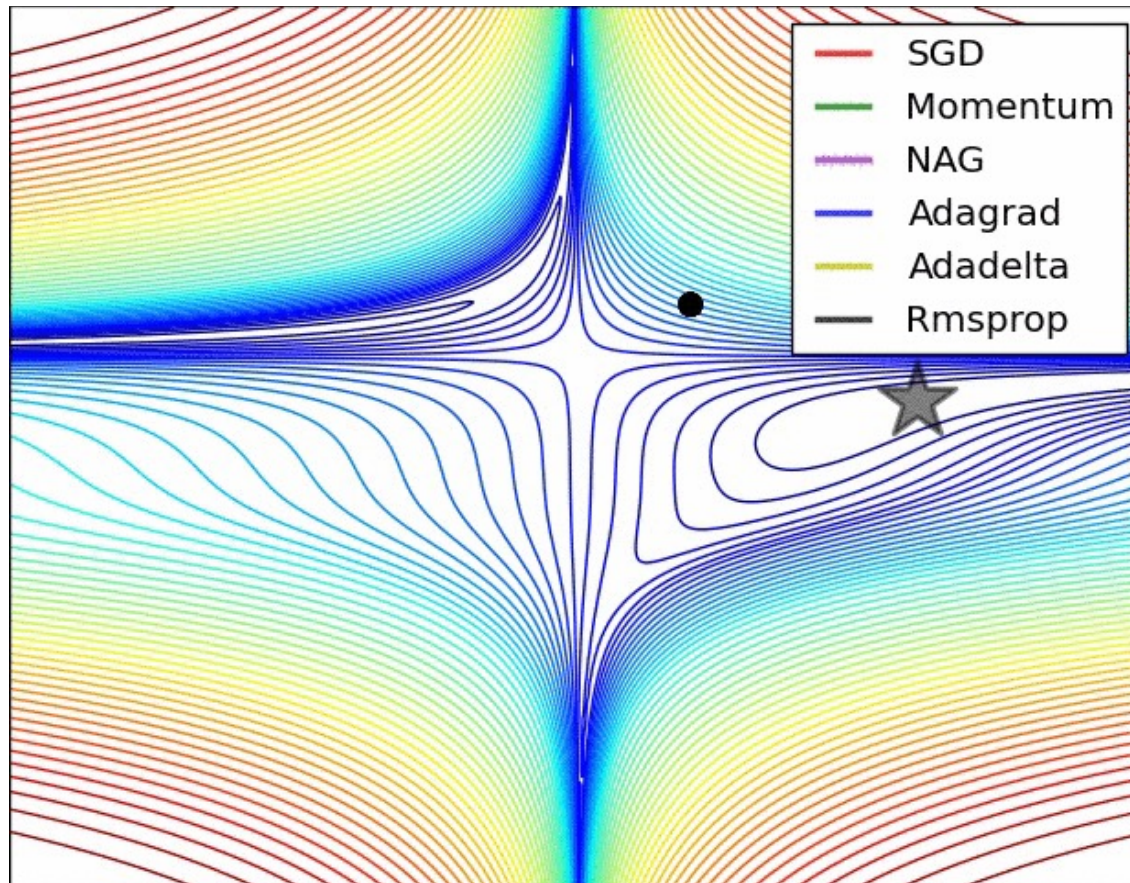
$$v_{t+1} = \gamma v_t + \eta \nabla \log L(\Theta)$$

$$\Theta_{t+1} \leftarrow \Theta_t - v_{t+1}$$

- Adaptive gradient:

- Adagrad and Adadelat: move faster in selected directions of the gradient, for parameters (words) that occur infrequently
- Cf. E-NLP 2.6.2

More complex optimization

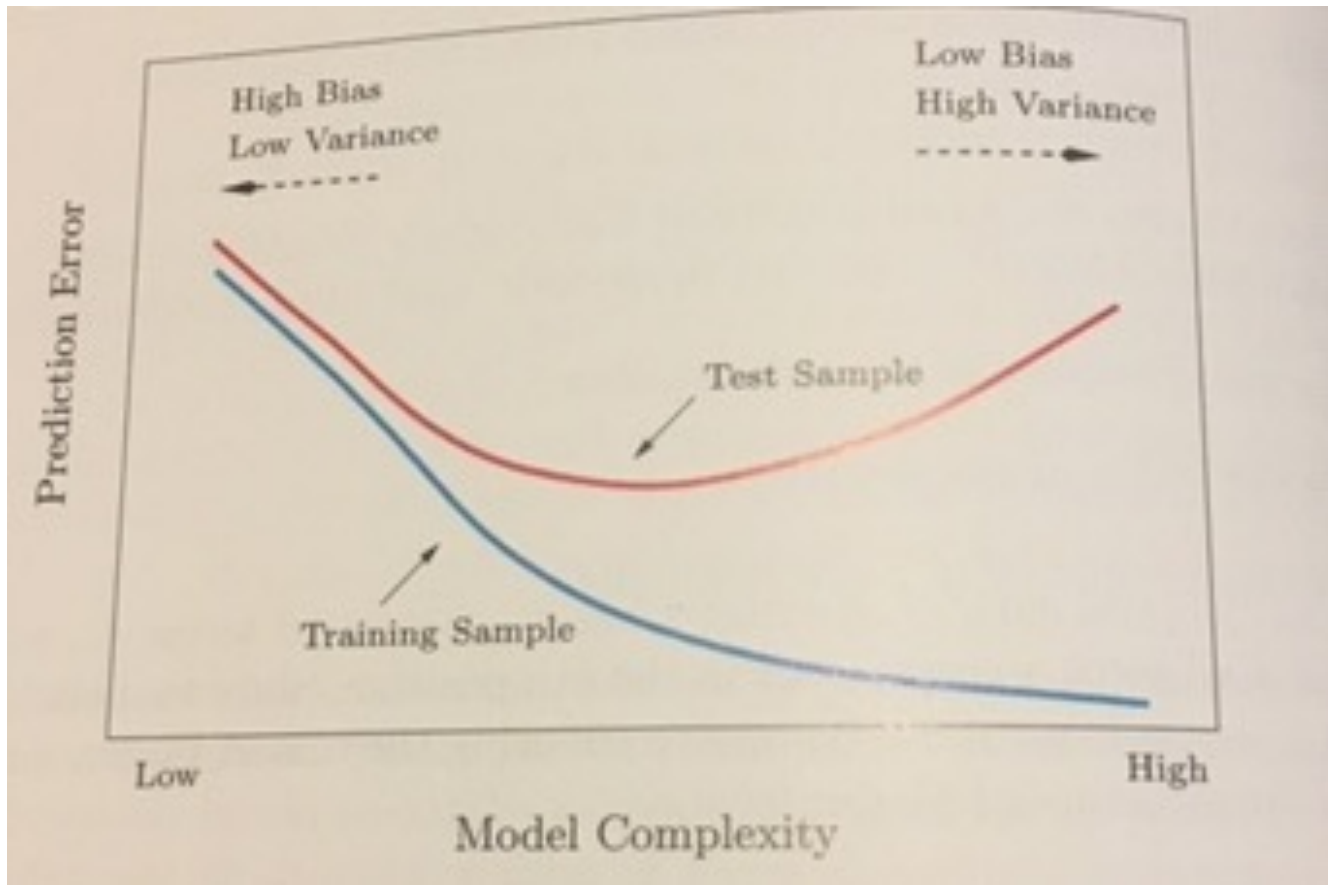


Alec Radford: <https://imgur.com/a/Hqolp>

Overfitting

- In NLP problems, we typically have a very large feature set – if every word is a feature, we will have $|V|$ features (or $|V| \times |Y|$ for multiclass classification)
- In many applications, $|V| \gg |D|$, the number of documents, so the model may be able to learn the characteristics of the training documents very exactly (“memorize” it)
- When this happens, we see that performance on the training set increases to near-perfection, while performance on test (“unseen”) data degrades
- The model fails to “generalize”

Overfitting



Overfitting

- Ways to deal with overfitting
 1. Get more data (increase $|D|$)
 2. Simplify the model (decrease $|V|$)
 3. Use *regularization* (constrain Θ)

Regularization

- Bias-Variance tradeoff
 - Bias: parameters should stay within “reasonable” bounds – not too many of them should be too large
 - Variance: parameters should be allowed to vary to capture the observed structure of the training data
- Regularization is a mechanism for increasing bias at the expense of variance

L₂ regularization for logistic regression

- Discourage model weights from getting too large by adding a penalty on the norm of the parameter vector Θ
- Instead of maximizing the log likelihood, minimize the penalized negative log likelihood

$$\log L(\Theta) = - \sum_{i=1}^N \log P(y_i | \vec{x}_i; \Theta) + \underbrace{\frac{\lambda}{2} \|\Theta\|_2^2}_{\text{L}_2 / \text{ridge penalty}}$$

L₁ penalty also possible;
encourages sparsity

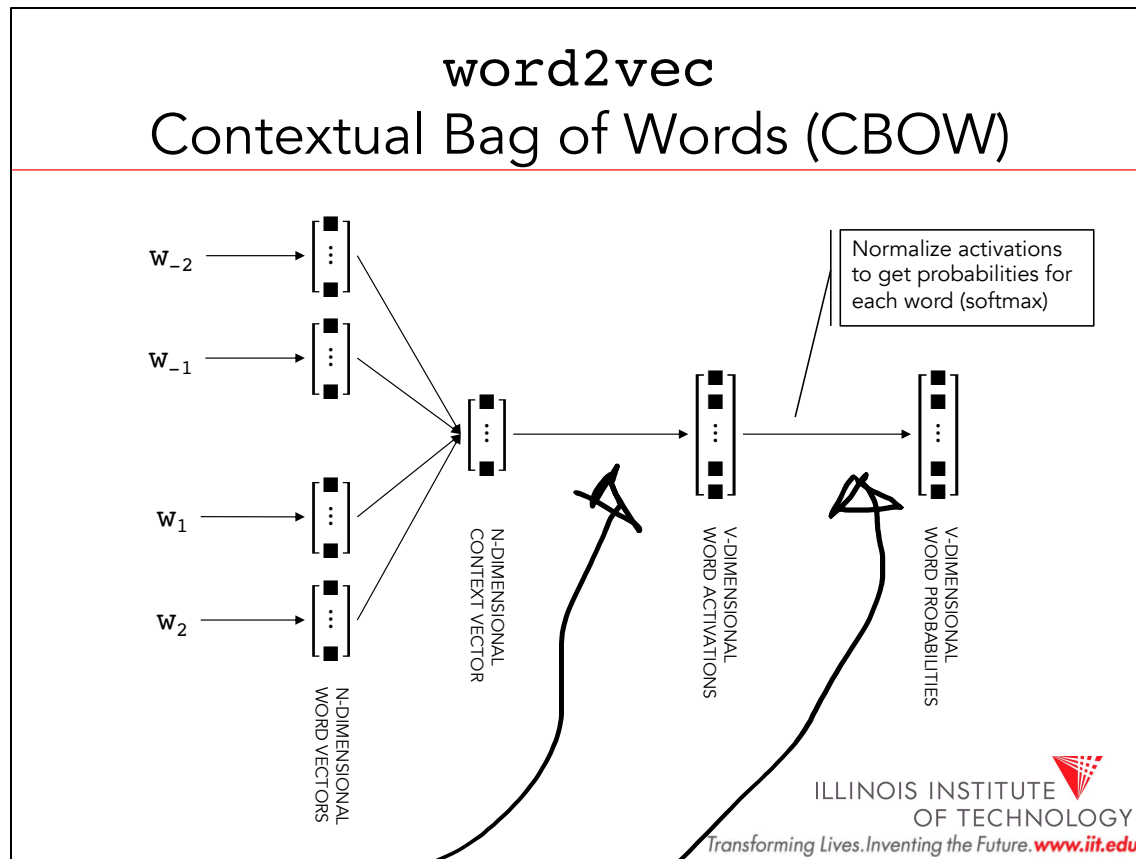
L₂ / ridge penalty

Generative vs. discriminative models

- Generative models
 - Model the joint probability distribution $P(Y, X)$
 - If we want to get the conditional probability $P(Y|X)$, we can normalize across classes: $P(Y = y|X) = \frac{P(Y=y, X)}{\sum_{y' \in Y} P(Y=y', X)}$
 - Can be understood in terms of a “generative story” – e.g., in a naïve Bayes framework, documents are the result of a process according to which
 - First, we choose a category according to distribution $P(Y)$
 - Then, we choose a bunch of words to fill the document up, according to $P(W|Y)$
- Discriminative models
 - Model $P(Y|X)$ directly
 - Most supervised classification methods are discriminative

Connection to neural networks

- Remember this?



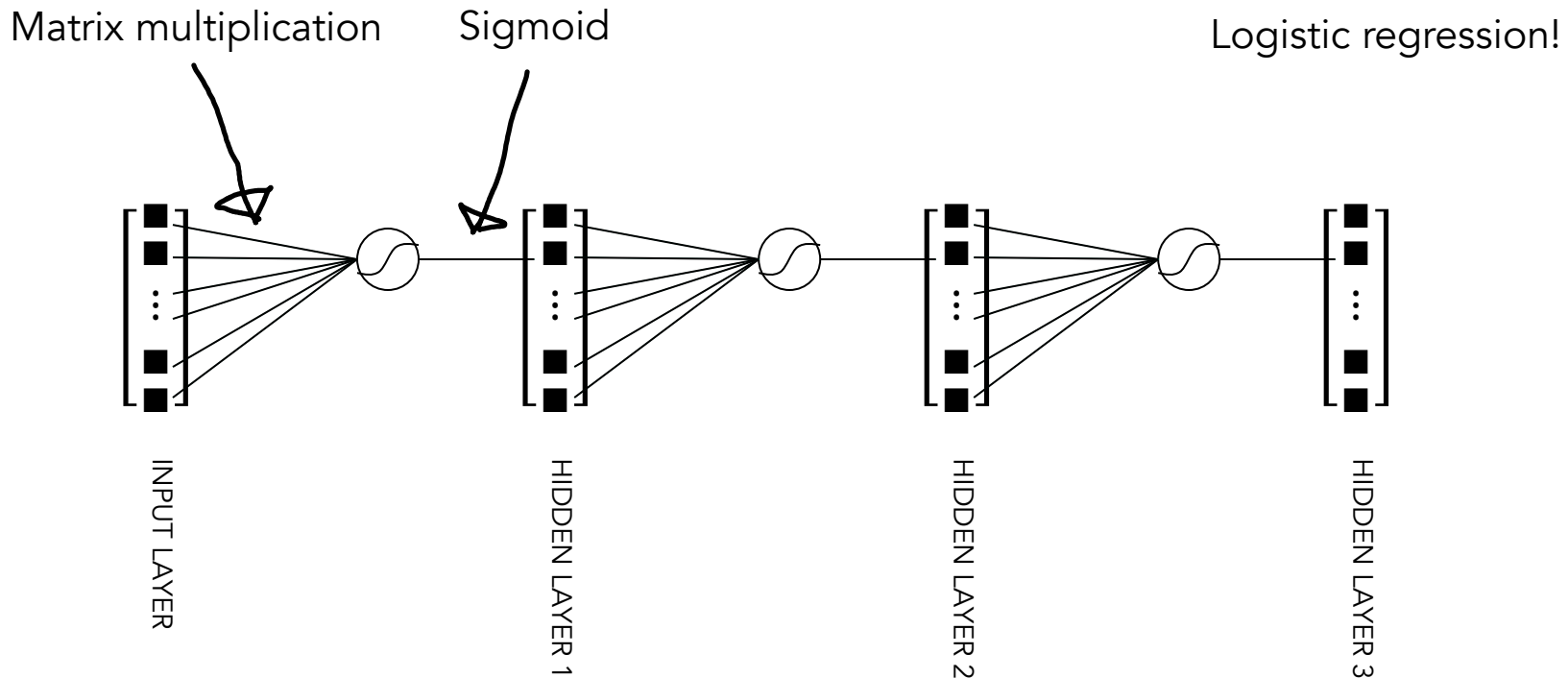
Matrix multiplication

Softmax

Connection to neural networks

- Generally, neural networks are sequences of matrix multiplications. Each layer of activations (a vector) is produced by multiplying the previous layer by a matrix
- But as we learned, a sequence of affine (linear) transformations, is itself an affine transform. So in order to allow neural networks to learn more complex functions, we have to introduce nonlinearities
- Softmax is one nonlinear transform. The sigmoid squashing function is another.

Connection to neural networks



$$\vec{h}_1 = \frac{e^{W\vec{x}}}{1 + e^{W\vec{x}}} \quad \vec{h}_2 = \frac{e^{W\vec{h}_1}}{1 + e^{W\vec{h}_1}} \quad \vec{h}_3 = \frac{e^{W\vec{h}_2}}{1 + e^{W\vec{h}_2}}$$

Example: naïve Bayes vs. logistic regression

[Notebook]