Deep Learning III

CS 760: Machine Learning
Spring 2018
Mark Craven and David Page

www.biostat.wisc.edu/~craven/cs760

1

Goals for the Lecture

- You should understand the following concepts:
  - generative adversarial networks (GANs)
  - competitive training
  - Nash equilibrium

  - long short-term memory (LSTM) networks
  - Bidirectional LSTMs (BiLSTMs)
  - Attention

  - embeddings
  - word embeddings
  - Word2Vec

2

Generative Adversarial Networks

- Combine deep net such as CNN with adversarial training
- One network tries to generate data to fool a second network
- Second network tries to discriminate fake from real data
- Objective for one is (roughly) negative of the other's objective
- Often can generate realistic new data
- Some background on adversarial training…

Prisoners' Dilemma

Two suspects in a major crime are held in separate cells. There is enough evidence to convict each of them of a minor offense, but not enough evidence to convict either of them of the major crime unless one of them acts as an informer against the other (defects). If they both stay quiet, each will be convicted of the minor offense and spend one year in prison. If one and only one of them defects, he will be freed and used as a witness against the other, who will spend four years in prison. If they both defect, each will spend three years in prison.

Players: The two suspects.

Actions: Each player's set of actions is {Quiet, Defect}.

Preferences: Suspect 1's ordering of the action profiles, from best to worst, is (Defect, Quiet) (he defects and suspect 2 remains quiet, so he is freed), (Quiet, Quiet) (he gets one year in prison), (Defect, Defect) (he gets three years in prison), (Quiet, Defect) (he gets four years in prison). Suspect 2's ordering is (Quiet, Defect), (Quiet, Quiet), (Defect, Defect), (Defect, Quiet).

Suspect1/ Suspect 2	Quiet	Defect
Quiet	2,2	0,3
Defect	3,0	1,1

3 represents best outcome, 0 worst, etc.

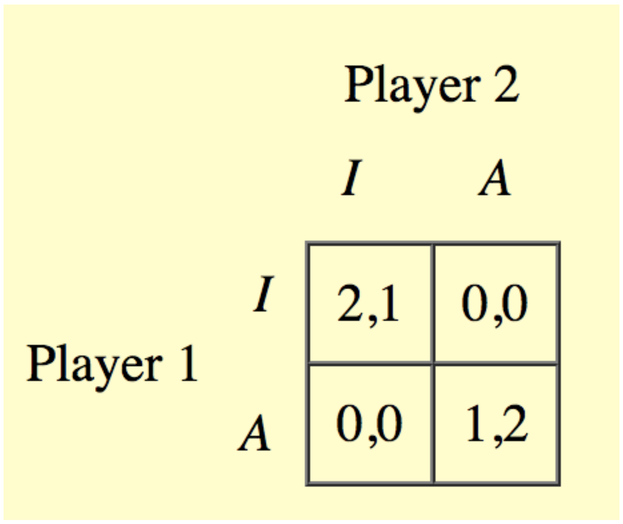
Nash Equilibrium

Let (S, f) be a game with n players, where S_i is the strategy set for player i, $S = S_1 \times S_2 \times \cdots \times S_n$ is the set of strategy profiles and $f(x) = (f_1(x), \ldots, f_n(x))$ is its payoff function evaluated at $x \in S$. Let x_i be a strategy profile of player i and x_{-i} be a strategy profile of all players except for player i. When each player $i \in \{1, \ldots, n\}$ chooses strategy x_i resulting in strategy profile $x = (x_1, \ldots, x_n)$ then player i obtains payoff $f_i(x)$. Note that the payoff depends on the strategy profile chosen, i.e., on the strategy chosen by player i as well as the strategies chosen by all the other players. A strategy profile $x^* \in S$ is a Nash equilibrium (NE) if no unilateral deviation in strategy by any single player is profitable for that player, that is

$$\forall i, x_i \in S_i : f_i(x_i^*, x_{-i}^*) \geq f_i(x_i, x_{-i}^*).$$

Thanks, Wikipedia.

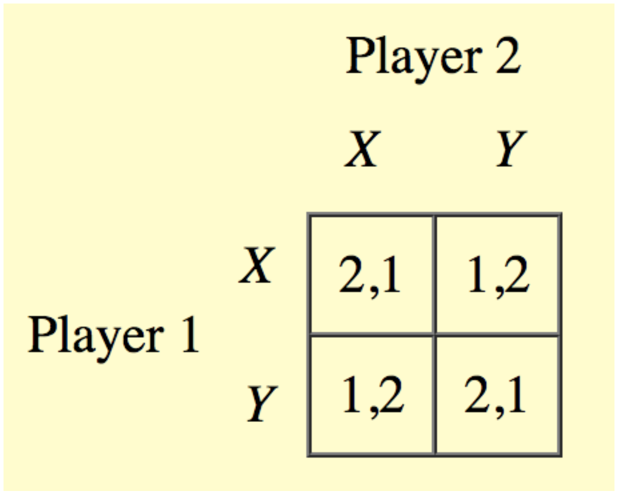
Another Example



		Player 2	
		I	*A*
Player 1	*I*	2,1	0,0
	A	0,0	1,2

Thanks, Prof. Osborne of U. Toronto, Economics

…And Another Example



		Player 2	
		X	*Y*
Player 1	*X*	2,1	1,2
	Y	1,2	2,1

Thanks again, Prof. Osborne of U. Toronto, Economics

Minimax with Simultaneous Moves

- *maximin* value: largest value player can be assured of *without* knowing other player's actions

$$\underline{v_i} = \max_{a_i} \min_{a_{-i}} v_i(a_i, a_{-i})$$

Where:

- i is the index of the player of interest.
- $-i$ denotes all other players except player i.

- *minimax* value: smallest value other players can force this player to receive *without* knowing this player's action

$$\overline{v_i} = \min_{a_{-i}} \max_{a_i} v_i(a_i, a_{-i})$$

- *minimax* is an upper bound on *maximin*

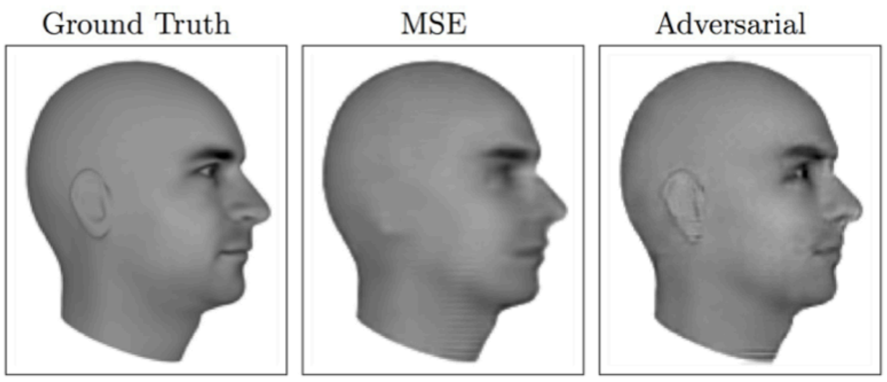
Key Result

- **Utility**: numeric reward for actions
- **Game**: 2 or more players take turns or take simultaneous actions. Moves lead to states, states have utilities.
- Game is like an optimization problem, but each player tries to maximize own objective function (utility function)
- **Zero-sum game**: each player's gain or loss in utility is exactly balanced by others'
- **In zero-sum game, *Minimax* solution is same as *Nash Equilibrium***

Generative Adversarial Networks

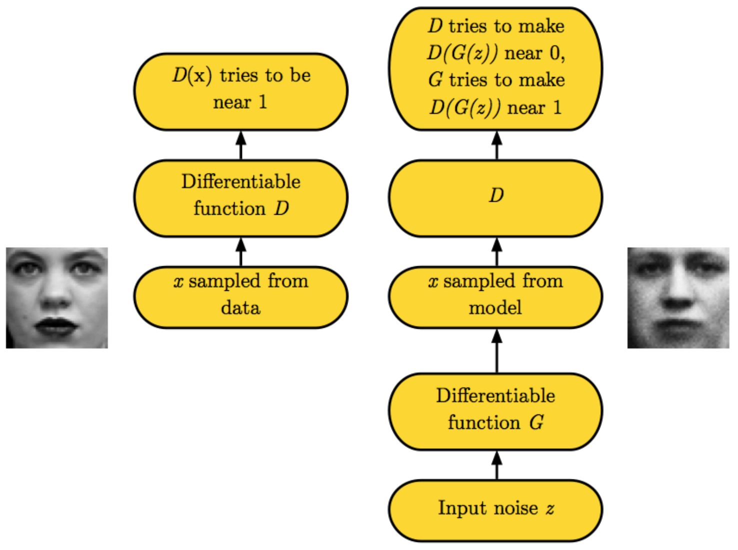
- **Approach:** Set up zero-sum game between deep nets to
  - **Generator:** Generate data that looks like training set
  - **Discriminator:** Distinguish between real and synthetic data
- **Motivation:**
  - Building accurate generative models is hard (e.g., *learning **and** sampling* from Markov net or Bayes net)
  - Want to use all our great progress on *supervised learners* to do this *unsupervised* learning task better
  - Deep nets may be our favorite supervised learner, especially for image data, if nets are convolutional (use tricks of sliding windows with parameter tying, cross-entropy transfer function, batch normalization)

Does It Work?



Thanks, Ian Goodfellow, NIPS 2016 Tutorial on GANS, for this and most of what follows…

A Bit More on GAN Algorithm


D(x) tries to be near 1
Differentiable function D
x sampled from data
D tries to make $D(G(z))$ near 0, G tries to make $D(G(z))$ near 1
D
x sampled from model
Differentiable function G
Input noise z

The Rest of the Details

- Use deep convolutional neural networks for Discriminator D and Generator G
- Let **x** denote trainset and **z** denote random, uniform input
- Set up zero-sum game by giving D the following objective, and G the negation of it:

$$-\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log\left(1 - D\left(G(\boldsymbol{z})\right)\right)$$

- Let D and G compute their gradients simultaneously, each make one step in direction of the gradient, and repeat until neither can make progress… Minimax

Not So Fast

- While preceding version is theoretically elegant, in practice the gradient for G vanishes before we reach best practical solution

- While no longer true Minimax, use same objective for D but change objective for G to:

$$-\frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log D(G(\boldsymbol{z}))$$

- Sometimes better if instead of using one minibatch at a time to compute gradient and do batch normalization, we also have a fixed subset of training set, and use combination of fixed subset and current minibatch

Comments on GANs

- Potentially can use our high-powered supervised learners to build better, faster data generators (can they replace MCMC, etc.?)

- While some nice theory based on Nash Equilibria, better results in practice if we move a bit away from the theory

- In general, many in ML community have strong concern that we don't really understand why deep learning works, including GANs

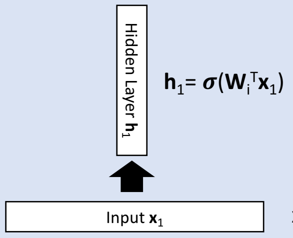
- Still much research into figuring out why this works better than other generative approaches for some types of data, how we can improve performance further, how to take these from image data to other data types where CNNs might not be the most natural deep network structure

Long Short-Term Memory Networks (LSTMs)

- Recurrent neural networks are well-suited to sequence data
  - Learning from sentences or other NLP
  - Learning from temporal sequences
- Recurrent neural networks model human short-term memory
  - Take into account recent thought processes, in addition to new input
  - Last hidden state and current input state both influence new hidden state and new output
- Sometimes start of sentence is relevant to ending output, so may want to keep early events in short-term memory a long time
  - Garden path sentences when output at each step is part-of-speech
  - "The complex houses married and single soldiers and their families."

Recall RNNs (thanks Ed Choi, Jimeng Sun)

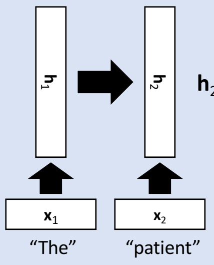
- Recurrent Neural Network (RNN)
  - Binary classification


Hidden Layer $\mathbf{h}_1$
$\mathbf{h}_1 = \sigma(\mathbf{W}_i^T \mathbf{x}_1)$
Input $\mathbf{x}_1$

$\mathbf{x}_1$ (First element in the sequence "The")

Recurrent Neural Networks

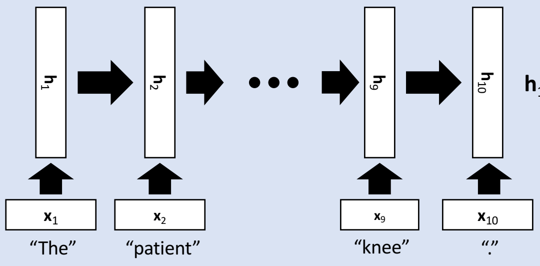
- Recurrent Neural Network (RNN)
  - Binary classification



$\mathbf{h}_2 = \sigma(\mathbf{W}_h^T\mathbf{h}_1 + \mathbf{W}_i^T\mathbf{x}_2)$

Recurrent Neural Networks

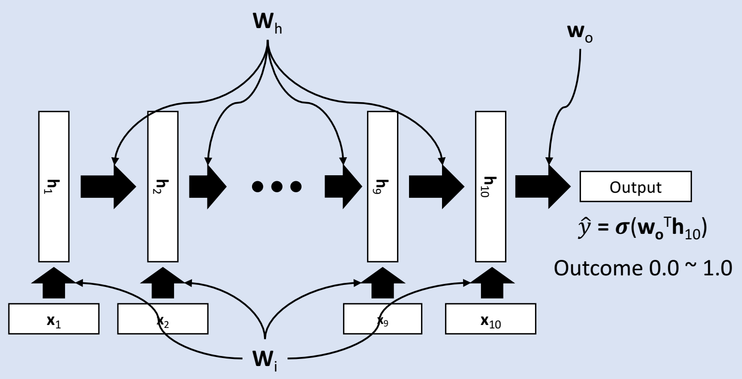
- Recurrent Neural Network (RNN)
  - Binary classification



$\mathbf{h}_{10} = \sigma(\mathbf{W}_h^T\mathbf{h}_9 + \mathbf{W}_i^T\mathbf{x}_{10})$

Recurrent Neural Networks

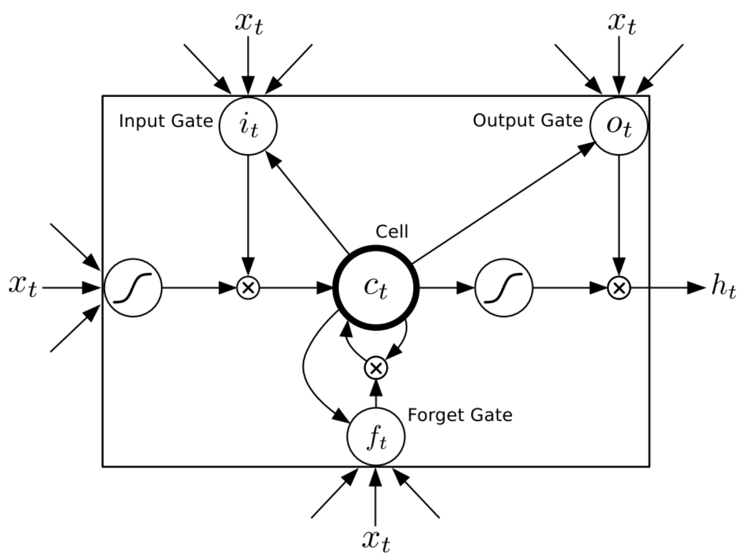
- Recurrent Neural Network (RNN)
  - Binary classification



Long Chain Equals Deep Learning

- Backpropagation must work its way back through the sequence: backpropagation through time (BTT)
- Harder to learn long-distance relationships
- LSTMs and related architectures try to make this easier by making long short-term memory the default, so effort is required to forget

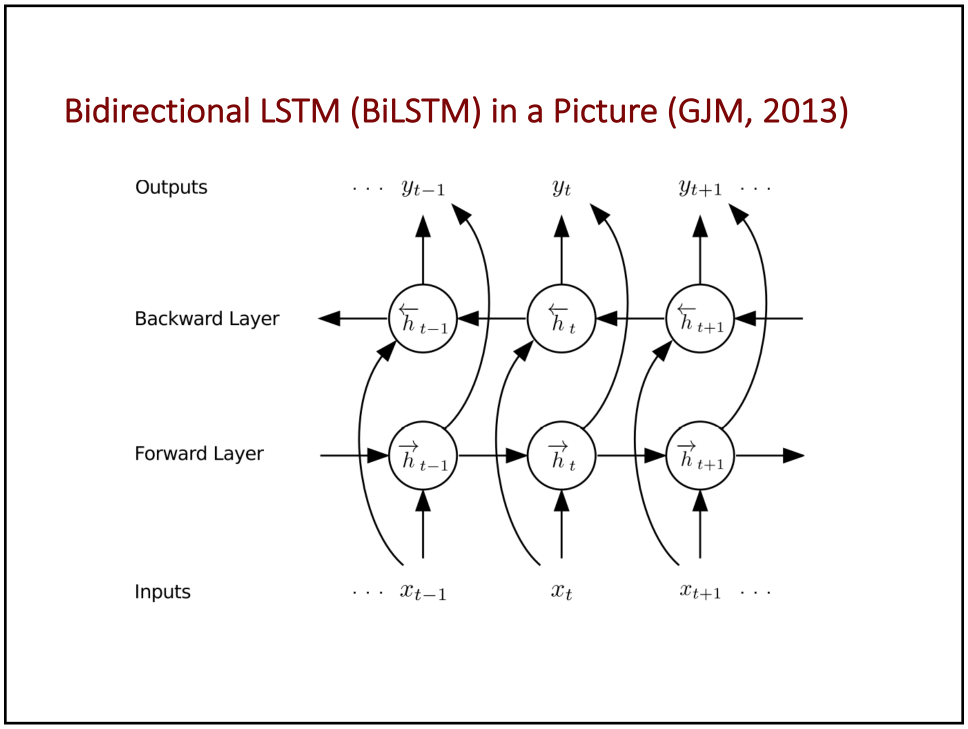
LSTM Cell in a Picture (Graves, Jaitly, Mohamed, 2013)



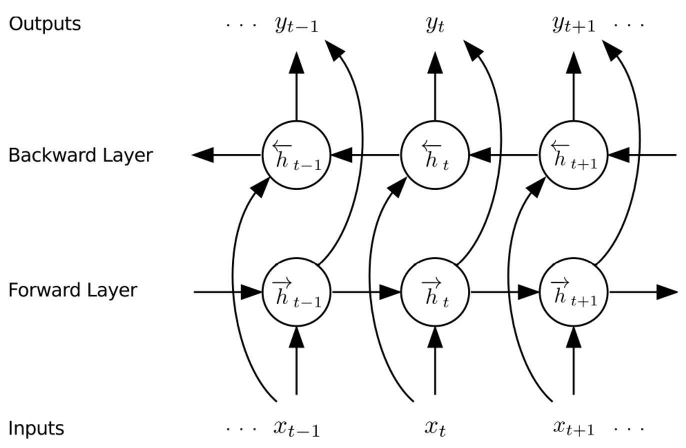
LSTM Equations (GJM, 2013)

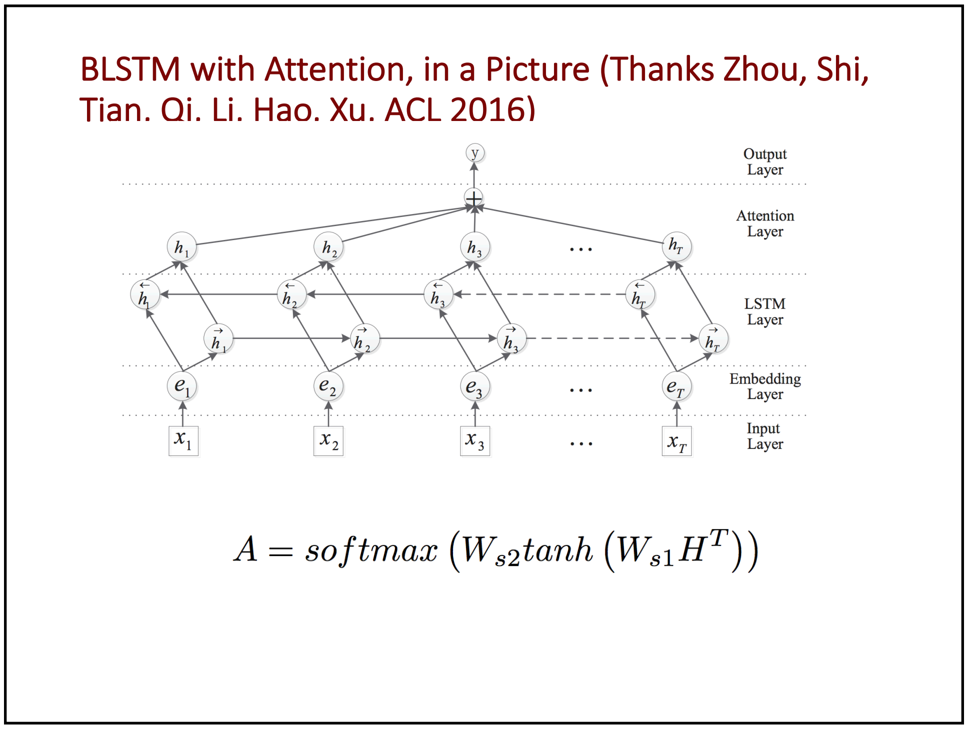
$$\begin{aligned}
i_t &= \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right)\\
f_t &= \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right)\\
c_t &= f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right)\\
o_t &= \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right)\\
h_t &= o_t \tanh(c_t)
\end{aligned}$$

Remember input, hidden, output, sigmoids (σ), tanh's are all vectors. Weight matrices from cell to gate, such as W_{ci}, are all diagonal, so element *m* of gate vector receives input only from element *m* of cell vector.

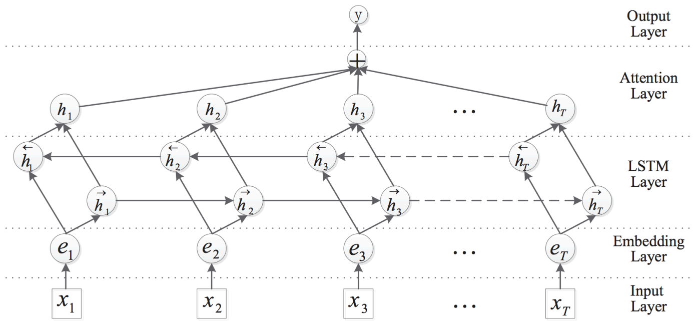


Bidirectional LSTM (BiLSTM) in a Picture (GJM, 2013)





BLSTM with Attention, in a Picture (Thanks Zhou, Shi, Tian. Qi. Li. Hao. Xu. ACL 2016)



$$A = softmax\left(W_{s2} tanh\left(W_{s1} H^T\right)\right)$$

Comments on BLSTMs with Attention

- Can have one *y* or a different output y_i at each time/location in sequence
- At each time/location in sequence, choose a (possibly different) subset of times to attend to for computing the output
- *BLSTMs with Attention* now the best approach for many NLP tasks (Manning, 2017)
- BLSTMs and other RNN models require more sequential processing than CNNs; therefore less speedup with GPUs with RNNs than with CNNs, though speedup is still substantial

Word embeddings

- standard representation for words is a one-hot (*1-of-k*) encoding

$$
\begin{array}{r}\text{aardvark}\\ \text{anteater}\\ \text{apple}\\ \\ \text{cat}\\ \\ \text{dog}\\ \\ \text{zymergy}\end{array}
\begin{bmatrix}0\\1\\0\\ \vdots\\0\\ \vdots\\0\\ \vdots\\0\end{bmatrix}
\qquad
\begin{array}{r}\text{aardvark}\\ \text{anteater}\\ \text{apple}\\ \\ \text{cat}\\ \\ \text{dog}\\ \\ \text{zymergy}\end{array}
\begin{bmatrix}0\\0\\0\\ \vdots\\1\\ \vdots\\0\\ \vdots\\0\end{bmatrix}
\qquad
\begin{array}{r}\text{aardvark}\\ \text{anteater}\\ \text{apple}\\ \\ \text{cat}\\ \\ \text{dog}\\ \\ \text{zymergy}\end{array}
\begin{bmatrix}0\\0\\0\\ \vdots\\0\\ \vdots\\1\\ \vdots\\0\end{bmatrix}
$$

- this encoding doesn't capture any information about the semantic similarity among words which may be useful for many NLP tasks

$$\boldsymbol{w}_{cat} \cdot \boldsymbol{w}_{dog} = 0$$

Word embeddings

- How can we find dense, lower-dimensional vectors that capture semantic similarity among words?

$$\begin{array}{cc} \text{cat} & \text{dog} \\ \begin{bmatrix} .26 \\ .01 \\ .37 \\ \vdots \\ .07 \end{bmatrix} & \begin{bmatrix} .24 \\ .04 \\ .38 \\ \vdots \\ .13 \end{bmatrix} \end{array}$$

- The key is to learn to model the context around words.
  Firth 1957: "*You shall know a word by the company it keeps*."

"…what if your dog has cold paws…"

The skip-gram model

- learn to predict context words in a window around given word
  - collect a large corpus of text
  - define a window size s
  - construct training instances that map from word → other word in window

"…what if your dog has cold paws…"

dog → *what*

dog → *if*

dog → *your*

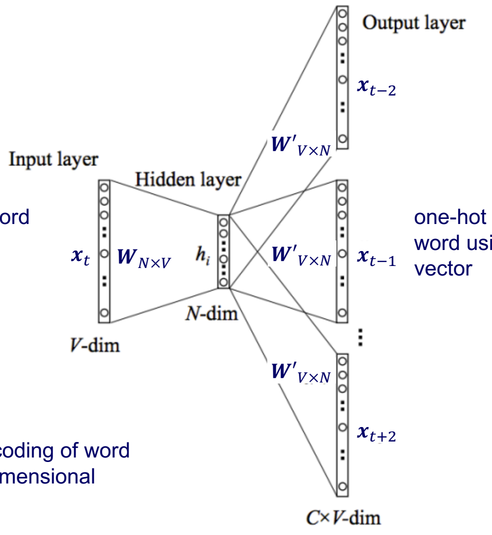
dog → *has*

dog → *cold*

dog → *paws*

The skip-gram model

one-hot encoding of word using V-dimensional vector

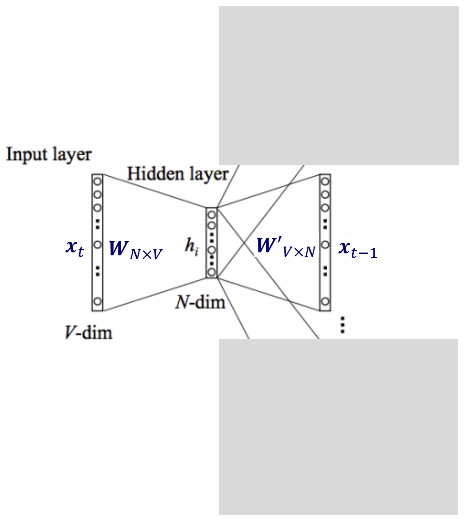


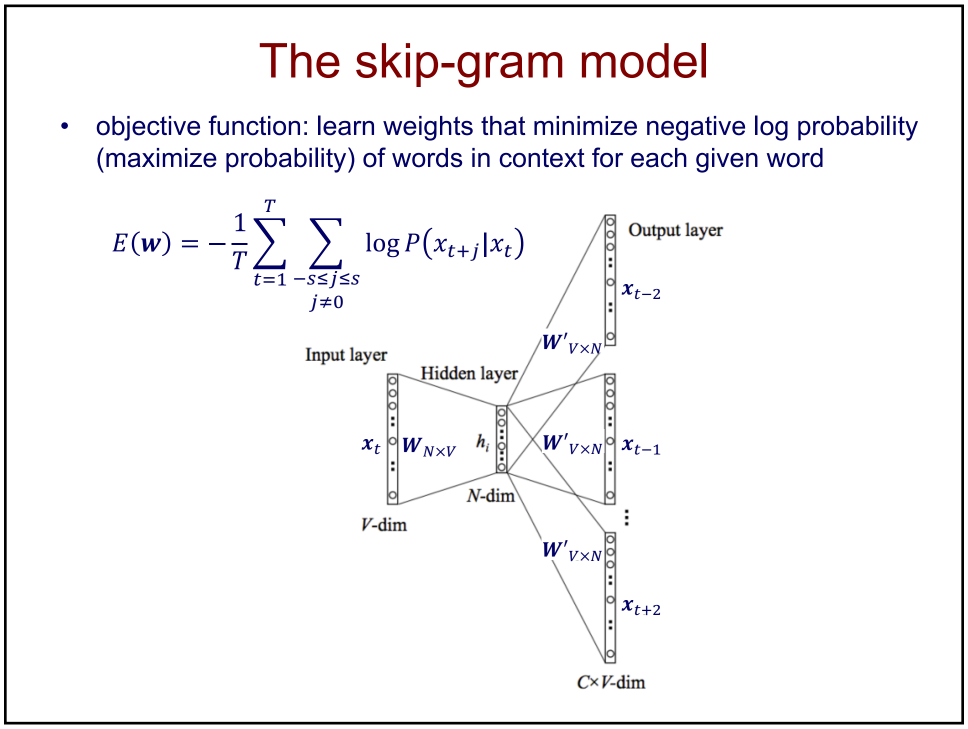
one-hot encoding of context word using V-dimensional vector

dense encoding of word using N-dimensional vector

The skip-gram model

- same weights used for all context positions, so effectively
  - network has one set of ouptut units
  - training instances are pairs (*dog* → *what*)

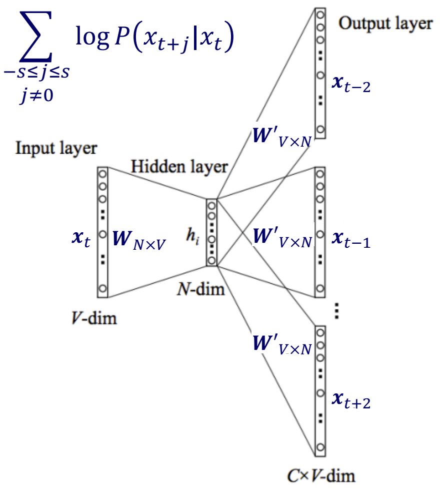


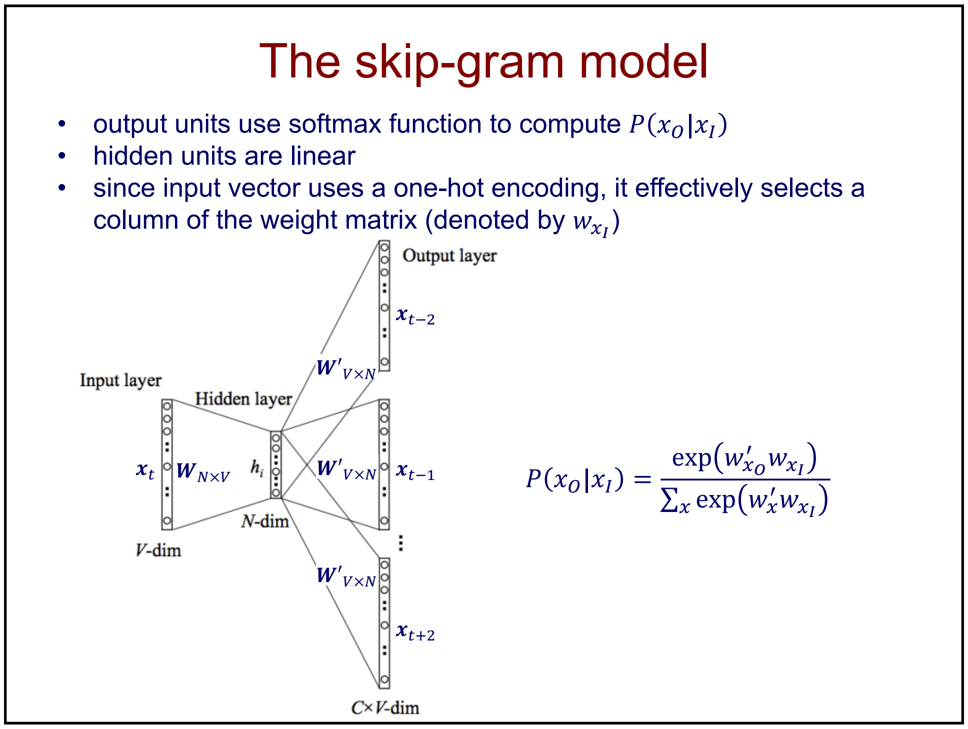


The skip-gram model

- objective function: learn weights that minimize negative log probability (maximize probability) of words in context for each given word

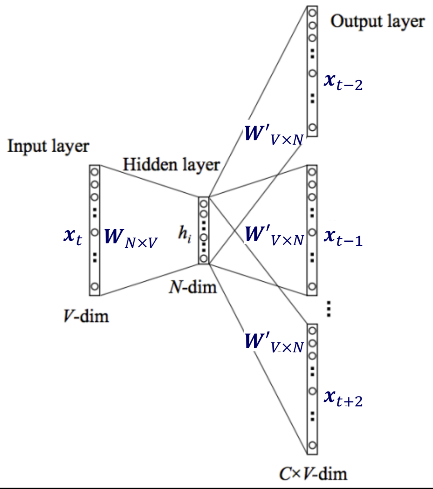
$$E(\boldsymbol{w}) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-s\le j\le s\\ j\neq 0}} \log P(x_{t+j}|x_t)$$





The skip-gram model

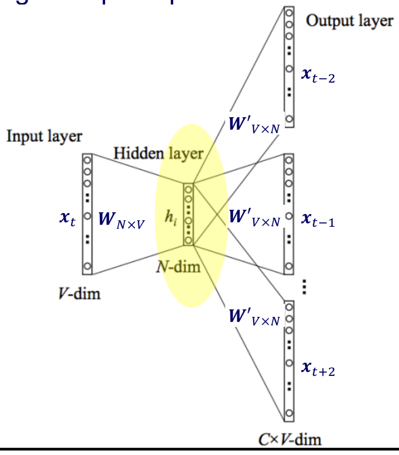
- output units use softmax function to compute $P(x_O|x_I)$
- hidden units are linear
- since input vector uses a one-hot encoding, it effectively selects a column of the weight matrix (denoted by w_{x_I})



$$P(x_O|x_I) = \frac{\exp\left(w'_{x_O} w_{x_I}\right)}{\sum_x \exp\left(w'_x w_{x_I}\right)}$$

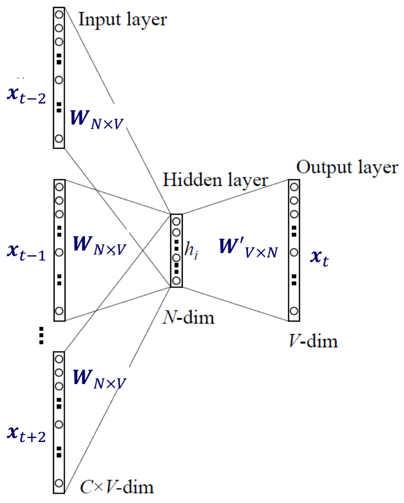
Using word embeddings

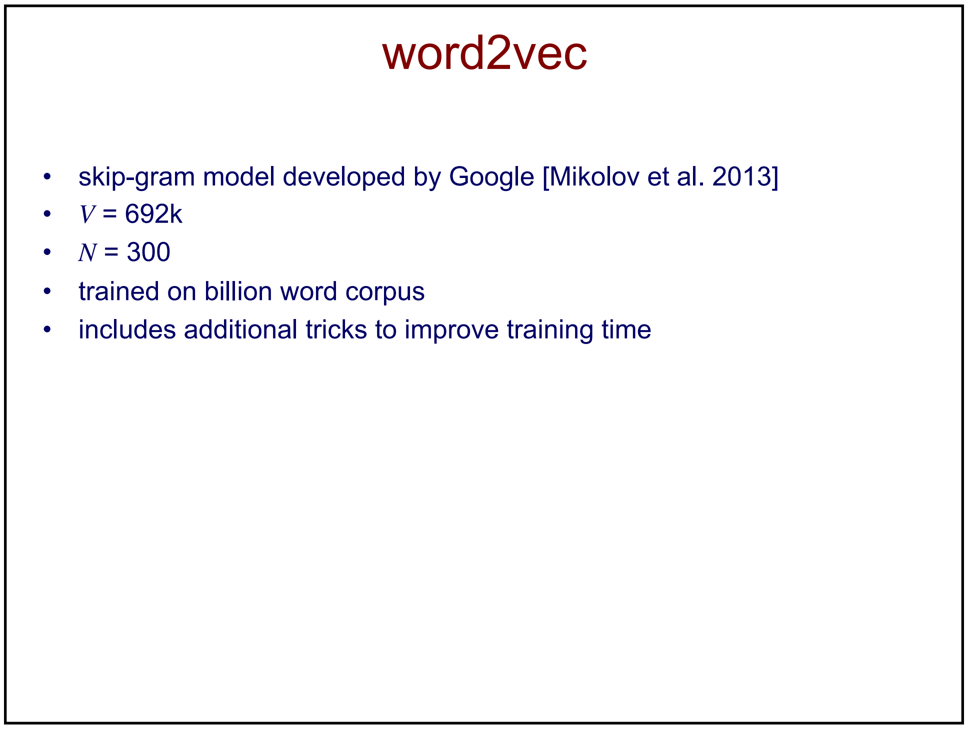
- after training, it's the hidden layer (equivalently, the first layer of weights) we care about
- the hidden units provide a dense, low-dimensional representation for each word
- we can use these low-dimensional vectors in any NLP application (e.g. as input representation for deep network)



An alternative: the Continuous Bag of Words (CBOW) model

- learn to predict a word given its context





word2vec

- skip-gram model developed by Google [Mikolov et al. 2013]
- V = 692k
- N = 300
- trained on billion word corpus
- includes additional tricks to improve training time