CS 6120/CS 4120: Natural Language Processing

Instructor: Prof. Lu Wang

College of Computer and Information Science

Northeastern University

Webpage: www.ccs.neu.edu/home/luwang

Outline

- Maximum Entropy
- Feedforward Neural Networks
- Recurrent Neural Networks

Introduction

- So far we’ve looked at “joint (or generative) models”
  - Language models, Naive Bayes, HMM
- But there is now much use of conditional or discriminative probabilistic models in NLP, Speech, information retrieval (and machine learning generally)
- Because:
  - They give high accuracy performance
  - They make it easy to incorporate lots of linguistically important features

Joint vs. Conditional Models

- We have some data $\{(d, c)\}$ of paired observations d and hidden classes c.
- Joint (generative) models place probabilities over both observed data and the hidden stuff (generate the observed data from hidden stuff):

$$p(c|d)=p(c,d)/p(d)$$

  - All the classic statistic NLP models:
    - *n*-gram models, Naive Bayes classifiers, hidden Markov models, probabilistic context-free grammars, IBM machine translation alignment models

Joint vs. Conditional Models

- Discriminative (conditional) models take the data as given, and put a probability over hidden structure given the data:

$$P(c|d)$$

  - Logistic regression/maximum entropy models (this lecture), conditional random fields
  - Also, SVMs, (averaged) perceptron, etc. are discriminative classifiers (but not directly probabilistic)

Conditional vs. Joint Likelihood

- A *joint* model gives probabilities $P(d,c)$ and tries to maximize this joint likelihood.
- A *conditional* model gives probabilities $P(c|d)$. It takes the data as given and models only the conditional probability of the class.
  - We seek to maximize conditional likelihood.
  - More closely related to classification error.

Maximum Entropy (MaxEnt)

- Or logistic regression

Features

- In these slides and most MaxEnt work: *features (or feature functions)* f are elementary pieces of evidence that link aspects of what we observe *d* with a category *c* that we want to predict
- A feature is a function with a **bounded** real value: $f\colon C \times D \to \mathbb{R}$

Example Task: Named Entity Type

LOCATION	LOCATION	DRUG	PERSON
in Arcadia	*in Québec*	*taking Zantac*	*saw Sue*

Example features

- $f_1(c, d) \equiv [c = \text{LOCATION} \wedge w_{-1} = \text{“in”} \wedge \text{isCapitalized}(w)]$
- $f_2(c, d) \equiv [c = \text{LOCATION} \wedge \text{hasAccentedLatinChar}(w)]$
- $f_3(c, d) \equiv [c = \text{DRUG} \wedge \text{ends}(w, \text{“c”})]$

LOCATION	LOCATION	DRUG	PERSON
in Arcadia	*in Québec*	*taking Zantac*	*saw Sue*

- Models will assign to each feature a *weight:*
  - A positive weight votes that this configuration is likely correct
  - A negative weight votes that this configuration is likely incorrect

Example features

- $f_1(c, d) \equiv [c = \text{LOCATION} \wedge w_{-1} = \text{“in”} \wedge \text{isCapitalized}(w)]$ -> weight 1.8
- $f_2(c, d) \equiv [c = \text{LOCATION} \wedge \text{hasAccentedLatinChar}(w)]$ -> weight -0.6
- $f_3(c, d) \equiv [c = \text{DRUG} \wedge \text{ends}(w, \text{“c”})]$ -> weight 0.3

- Weights will be learned by training on a labeled dataset

More about feature functions:

an indicator function – a yes/no boolean matching function – of properties of the input and a particular class

$$f_i(c, d) \equiv [\Phi(d) \wedge c = c_j]$$

[Value is 0 or 1]

Feature-Based Models

- The decision about a data point is based only on the features active at that point.

Text Classification	Word Sense Disambiguation	POS Tagging
Data BUSINESS: Stocks hit a yearly low …	Data … to restructure bank:MONEY debt.	Data DT JJ NN … The previous fall …
Label: BUSINESS Features {…, stocks, hit, a, yearly, low, …}	Label: MONEY Features {…, w_{-1}=restructure, w_{+1}=debt, L=12, …}	Label: NN Features {w=fall, t_{-1}=JJ w_{-1}=previous}

Feature-Based Linear Classifiers

- Linear classifiers at classification time:
  - Linear function from feature sets $\{f_i\}$ to classes $\{c\}$.
  - Assign a weight λ_i to each feature f_i.
  - We consider each class for sample d
  - For a pair (c,d), features vote with their weights:
    - $\text{vote(c)} = \sum_i \lambda_i f_i(c,d)$

PERSON | LOCATION | DRUG
in Québec | *in Québec* | *in Québec*

- Choose the class c which maximizes $\sum_i \lambda_i f_i(c,d)$

- Maximum Entropy:
  - Make a probabilistic model from the linear combination $\Sigma\lambda_i f_i(c,d)$

$$P(c \mid d, \lambda) = \frac{\exp \sum_i \lambda_i f_i(c,d)}{\sum_{c'} \exp \sum_i \lambda_i f_i(c',d)}$$

Makes votes positive

Normalizes votes

Feature-Based Linear Classifiers

- $f_1(c, d) \equiv [c = \text{LOCATION} \wedge w_{-1} = \text{“in”} \wedge \text{isCapitalized}(w)]$ -> weight 1.8
- $f_2(c, d) \equiv [c = \text{LOCATION} \wedge \text{hasAccentedLatinChar}(w)]$ -> weight -0.6
- $f_3(c, d) \equiv [c = \text{DRUG} \wedge \text{ends}(w, \text{“c”})]$ -> weight 0.3

$f_1(c, d) \equiv [c = \text{LOCATION} \wedge w_{-1} = \text{“in”} \wedge \text{isCapitalized}(w)]$ -> weight 1.8

$f_2(c, d) \equiv [c = \text{LOCATION} \wedge \text{hasAccentedLatinChar}(w)]$ -> weight -0.6

$f_3(c, d) \equiv [c = \text{DRUG} \wedge \text{ends}(w, \text{“c”})]$ -> weight 0.3

- Maximum Entropy:
  - Make a probabilistic model from the linear combination $\Sigma\lambda_i f_i(c,d)$

$$P(c \mid d, \lambda) = \frac{\exp \sum_i \lambda_i f_i(c, d)}{\sum_{c'} \exp \sum_i \lambda_i f_i(c', d)}$$

Makes votes positive

Normalizes votes

$f_1(c, d) \equiv [c = \text{LOCATION} \wedge w_{-1} = \text{"in"} \wedge \text{isCapitalized}(w)]$ -> weight 1.8
$f_2(c, d) \equiv [c = \text{LOCATION} \wedge \text{hasAccentedLatinChar}(w)]$ -> weight -0.6
$f_3(c, d) \equiv [c = \text{DRUG} \wedge \text{ends}(w, \text{"c"})]$ -> weight 0.3

- Maximum Entropy:
  - Make a probabilistic model from the linear combination $\Sigma\lambda_i f_i(c,d)$

$$P(c \mid d, \lambda) = \frac{\exp \sum_i \lambda_i f_i(c,d)}{\sum_{c'} \exp \sum_i \lambda_i f_i(c',d)}$$

Makes votes positive

Normalizes votes

- P(LOCATION|*in Québec*) = $e^{1.8}e^{-0.6}/(e^{1.8}e^{-0.6} + e^{0.3} + e^{0}) = 0.586$
- P(DRUG|*in Québec*) = $e^{0.3}/(e^{1.8}e^{-0.6} + e^{0.3} + e^{0}) = 0.238$
- P(PERSON|*in Québec*) = $e^{0}/(e^{1.8}e^{-0.6} + e^{0.3} + e^{0}) = 0.176$
- The weights are the parameters of the probability model, combined via a "soft max" function

Feature-Based Linear Classifiers

- Given this model form, we will choose parameters $\{\lambda_i\}$ that *maximize the conditional likelihood* of the data according to this model.
- Parameter learning is omitted and not required for this course, but is often discussed in a machine learning class.
  - E.g. gradient descent for parameter learning
- We construct not only classifications, but probability distributions over classifications.
  - There are other (good!) ways of discriminating classes – SVMs, boosting, even perceptrons – but these methods are not as trivial to interpret as distributions over classes.

Other MaxEnt Classifier Examples

- You can use a MaxEnt classifier whenever you want to assign data points to one of a number of classes:
  - Sentence boundary detection (Mikheev 2000)
    - Is a period end of sentence or abbreviation?
  - Sentiment analysis (Pang and Lee 2002)
    - Word unigrams, bigrams, POS counts, …
  - Prepositional phrase attachment (Ratnaparkhi 1998)
    - Attach to verb or noun? Features of head noun, preposition, etc.
  - Parsing decisions (Ratnaparkhi 1997; Johnson et al. 1999, etc.)

Outline

- Maximum Entropy
- Feedforward Neural Networks
- Recurrent Neural Networks

Neural Network Learning

- Learning approach based on modeling adaptation in biological neural systems.
- Perceptron: Initial algorithm for learning simple neural networks (single layer) developed in the 1950's.
- Backpropagation: More complex algorithm for learning multi-layer neural networks developed in the 1980's. (not required for this class)

ARTIFICIAL NEURON

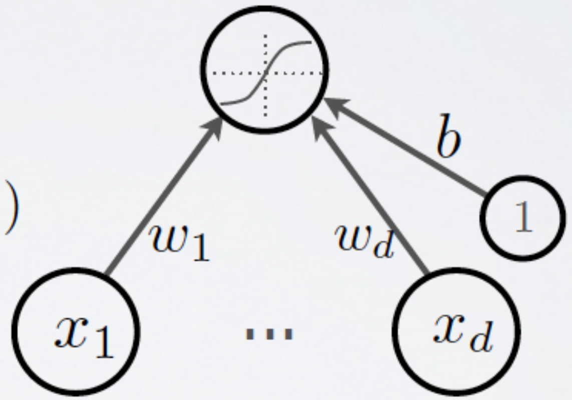
Topics: connection weights, bias, activation function

- Neuron pre-activation (or input activation):

$$a(\mathbf{x}) = b + \sum_i w_i x_i = b + \mathbf{w}^\top \mathbf{x}$$

- Neuron (output) activation

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

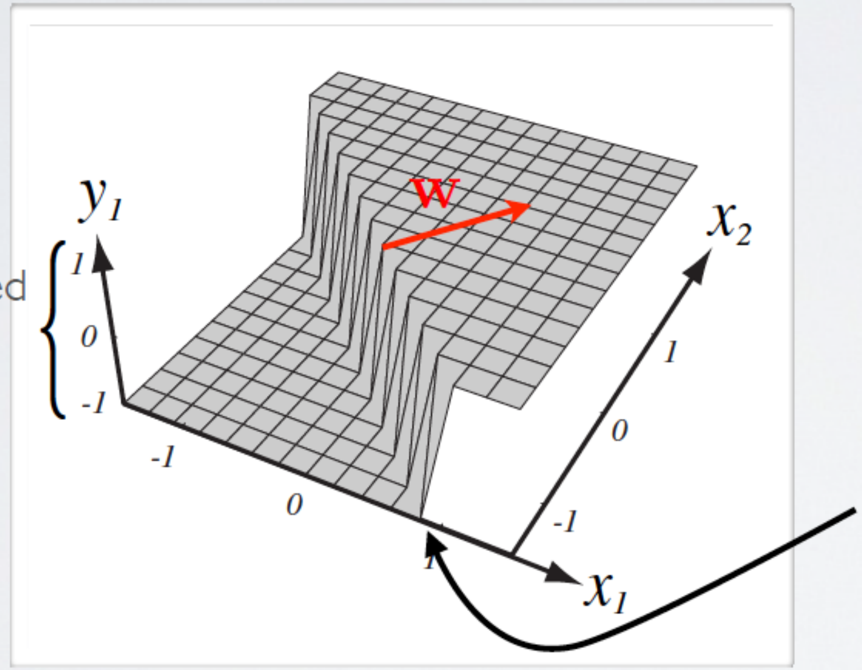


- $\mathbf{w}$ are the connection weights
- b is the neuron bias
- $g(\cdot)$ is called the activation function

ARTIFICIAL NEURON

Topics: connection weights, bias, activation function

range determined by $g(\cdot)$



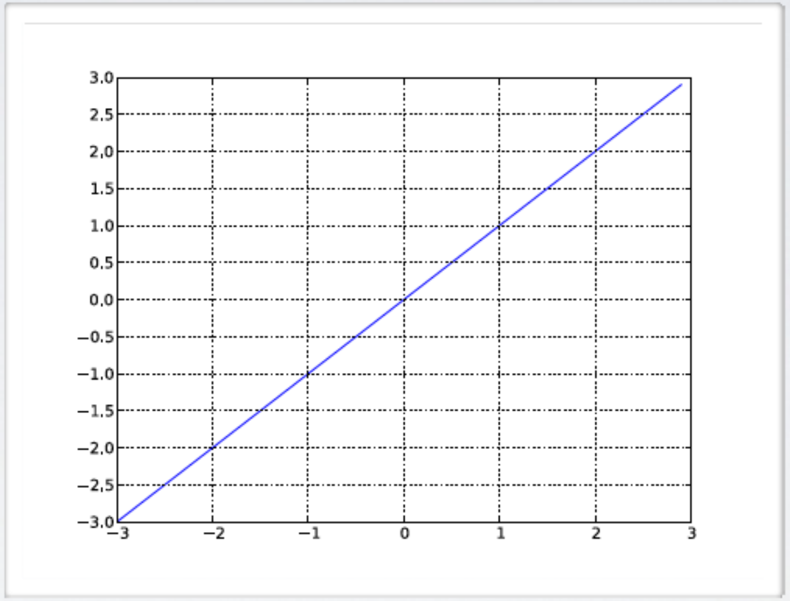
bias b only changes the position of the riff

(from Pascal Vincent's slides)

ACTIVATION FUNCTION

Topics: linear activation function

- Performs no input squashing
- Not very interesting...

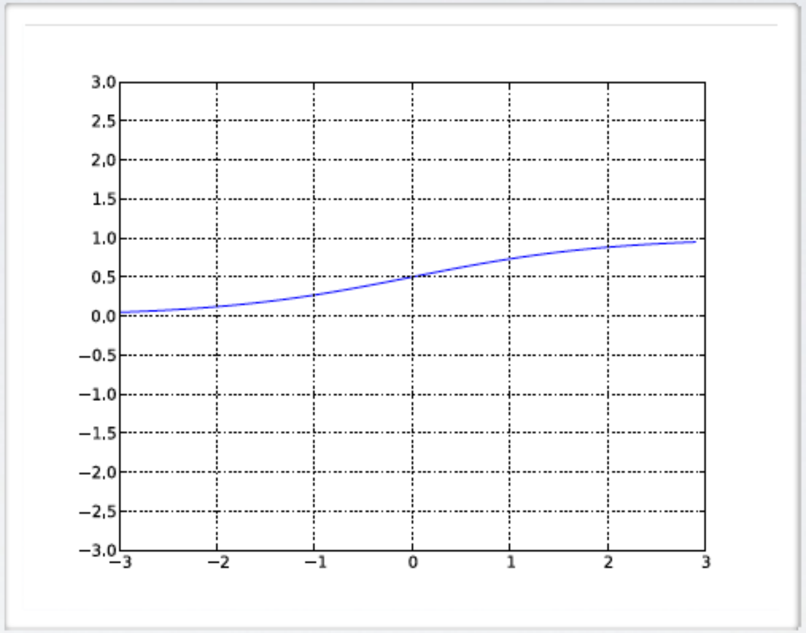


$$g(a) = a$$

ACTIVATION FUNCTION

Topics: sigmoid activation function

- Squashes the neuron's pre-activation between 0 and 1
- Always positive
- Bounded
- Strictly increasing

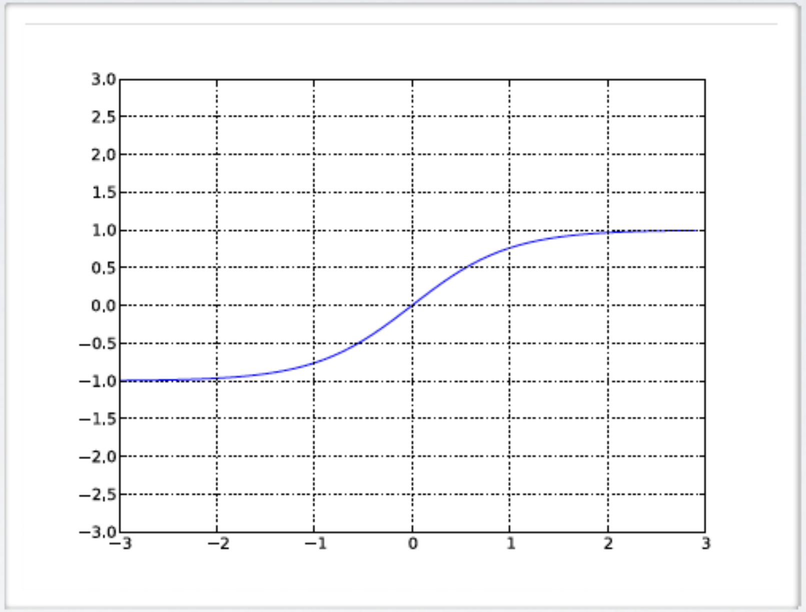


$$g(a) = \text{sigm}(a) = \frac{1}{1+\exp(-a)}$$

ACTIVATION FUNCTION

Topics: hyperbolic tangent ("tanh") activation function

- Squashes the neuron's pre-activation between -1 and 1
- Can be positive or negative
- Bounded
- Strictly increasing

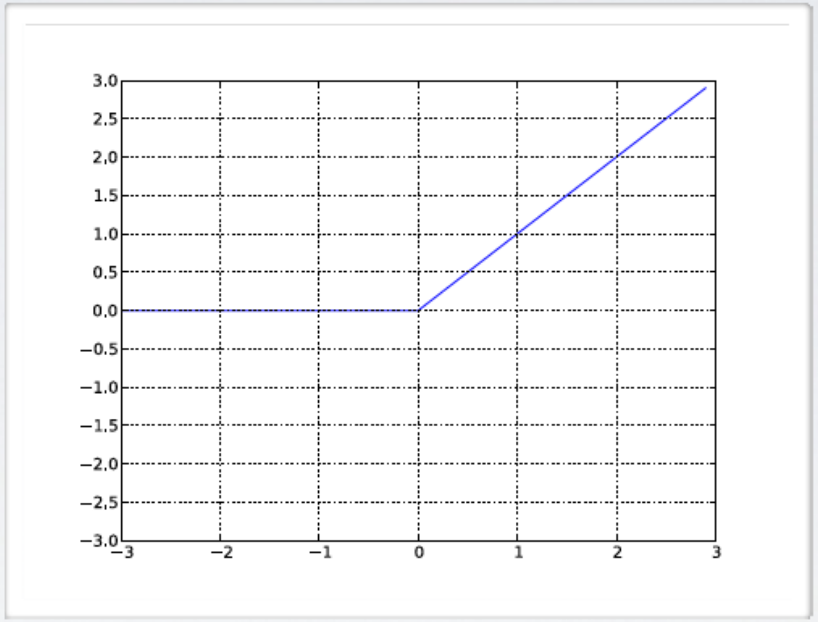


$$g(a) = \tanh(a) = \frac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)} = \frac{\exp(2a)-1}{\exp(2a)+1}$$

ACTIVATION FUNCTION

Topics: rectified linear activation function

- Bounded below by 0 (always non-negative)
- Not upper bounded
- Strictly increasing
- Tends to give neurons with sparse activities

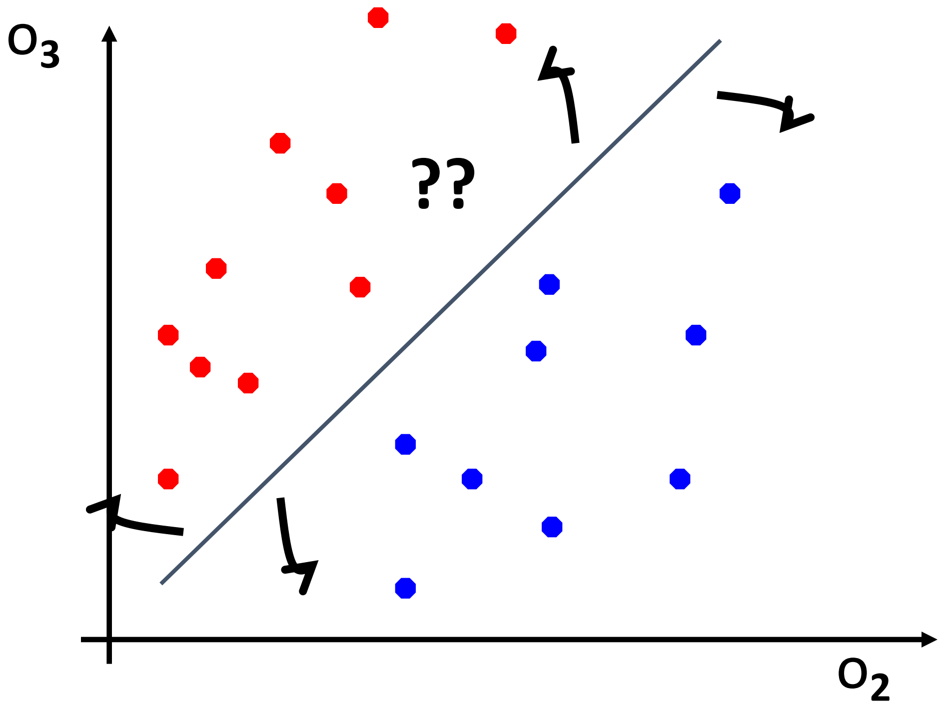


$$g(a) = \text{reclin}(a) = \max(0, a)$$

```python
class Neuron(object):
  # ...
  def forward(inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number """
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation function
    return firing_rate
```

Linear Separator

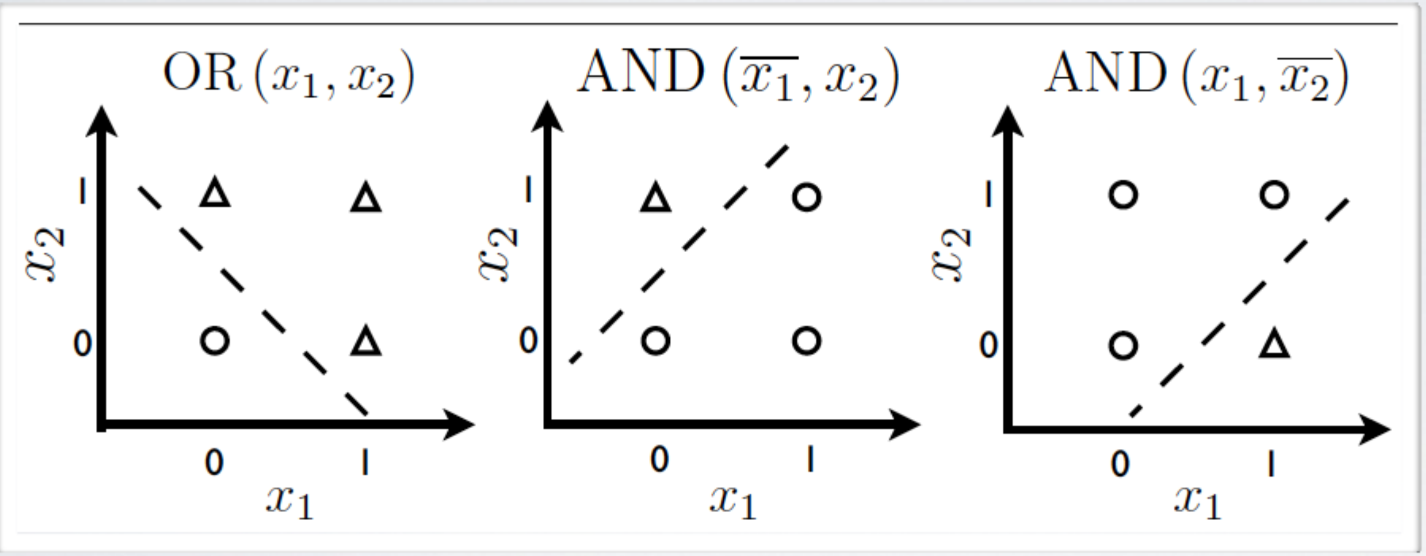
- Since one-layer neuron (aka perceptron) uses linear threshold function, it is searching for a linear separator that discriminates the classes.



ARTIFICIAL NEURON

Topics: capacity of single neuron

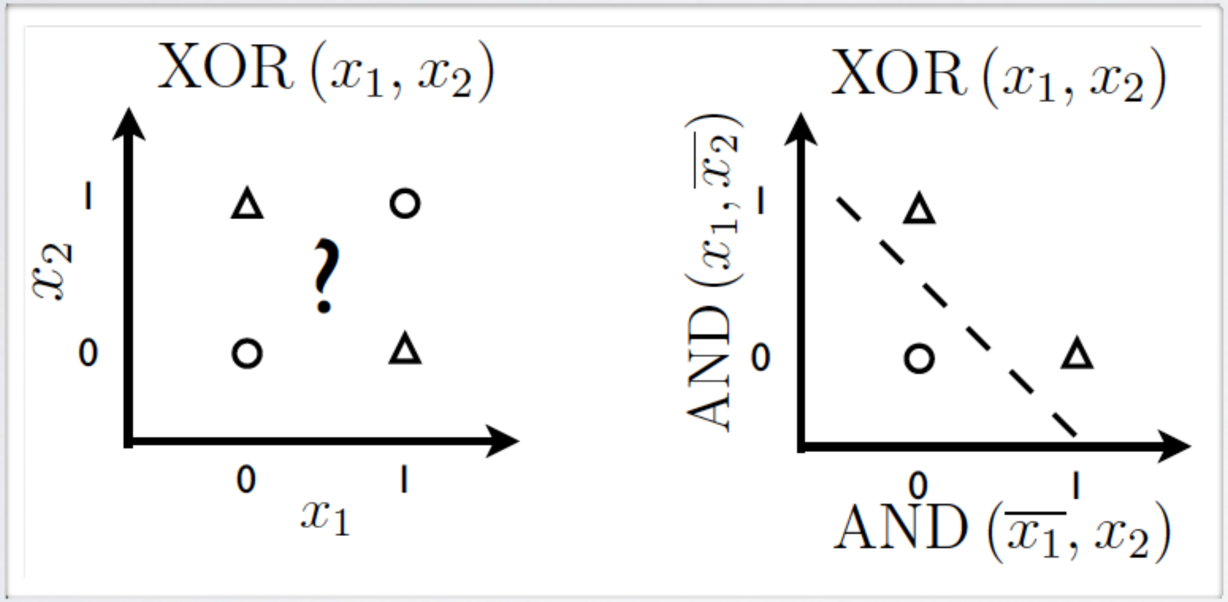
- Can solve linearly separable problems



ARTIFICIAL NEURON

Topics: capacity of single neuron

- Can't solve non linearly separable problems...



- ... unless the input is transformed in a better representation

NEURAL NETWORK

Topics: single hidden layer neural network

- Hidden layer pre-activation:

$$\mathbf{a}(\mathbf{x}) = \mathbf{b}^{(1)} + \mathbf{W}^{(1)}\mathbf{x}$$

$$\left(a(\mathbf{x})_i = b_i^{(1)} + \sum_j W_{i,j}^{(1)} x_j\right)$$

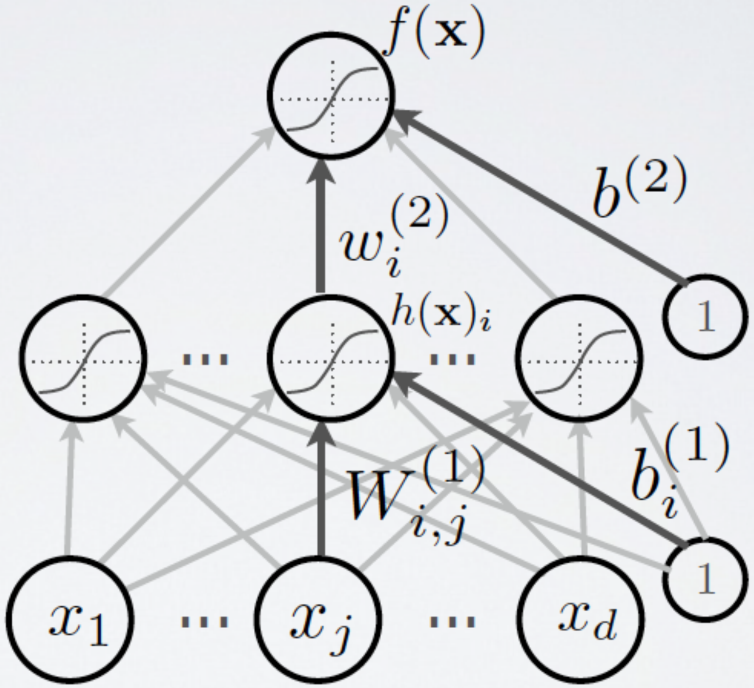
- Hidden layer activation:

$$\mathbf{h}(\mathbf{x}) = \mathbf{g}(\mathbf{a}(\mathbf{x}))$$

- Output layer activation:

$$f(\mathbf{x}) = o\left(b^{(2)} + \mathbf{w}^{(2)\top}\mathbf{h}^{(1)}\mathbf{x}\right)$$

output activation function



NEURAL NETWORK

Topics: softmax activation function

- For multi-class classification:
  - we need multiple outputs (1 output per class)
  - we would like to estimate the conditional probability $p(y = c|\mathbf{x})$
- We use the softmax activation function at the output:

$$\mathbf{o}(\mathbf{a}) = \text{softmax}(\mathbf{a}) = \left[\frac{\exp(a_1)}{\sum_c \exp(a_c)} \dots \frac{\exp(a_C)}{\sum_c \exp(a_c)} \right]^\top$$

  - strictly positive
  - sums to one
- Predicted class is the one with highest estimated probability

NEURAL NETWORK

Topics: multilayer neural network

- Could have L hidden layers:
  - layer pre-activation for $k>0$ $(\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x})$

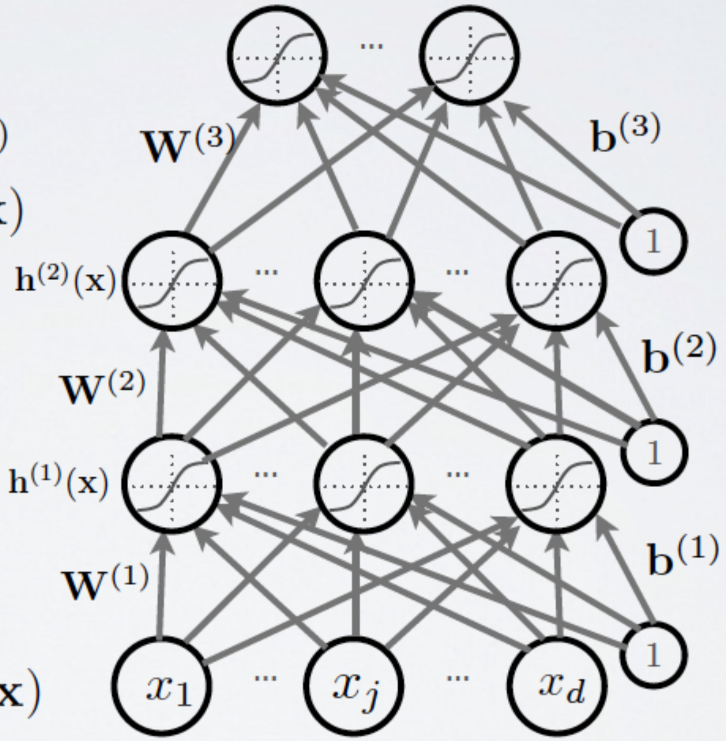
    $$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$

  - hidden layer activation (k from 1 to L):

    $$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x}))$$

  - output layer activation ($k=L+1$):

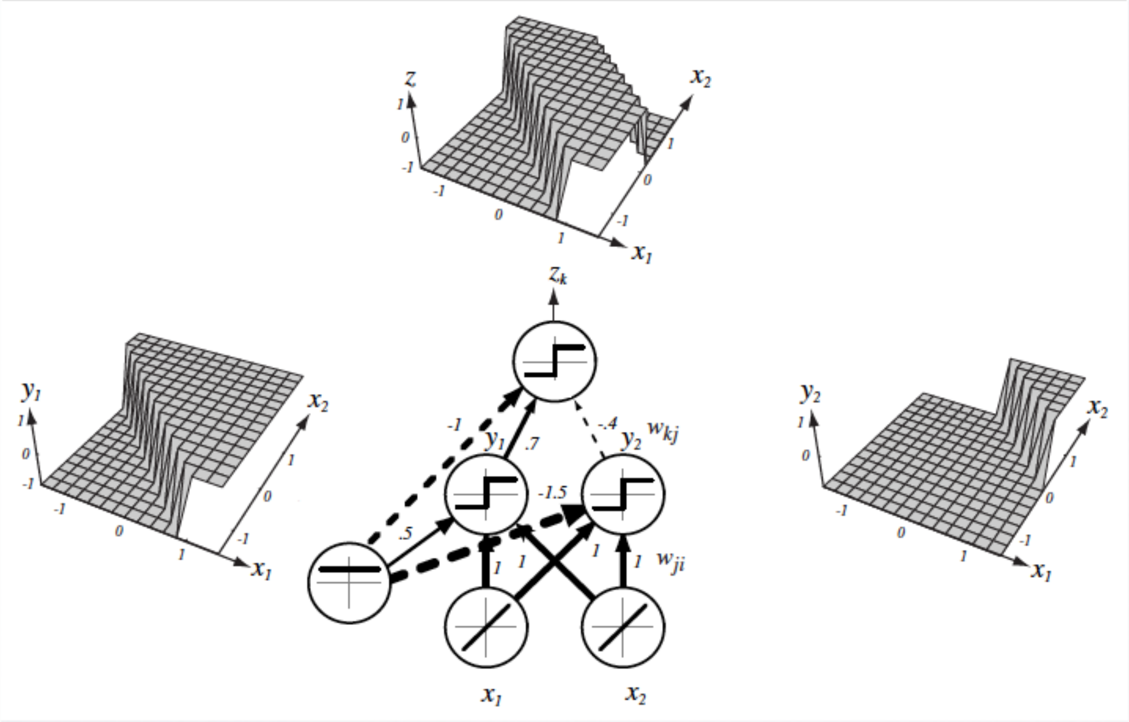
    $$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$



```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

CAPACITY OF NEURAL NETWORK

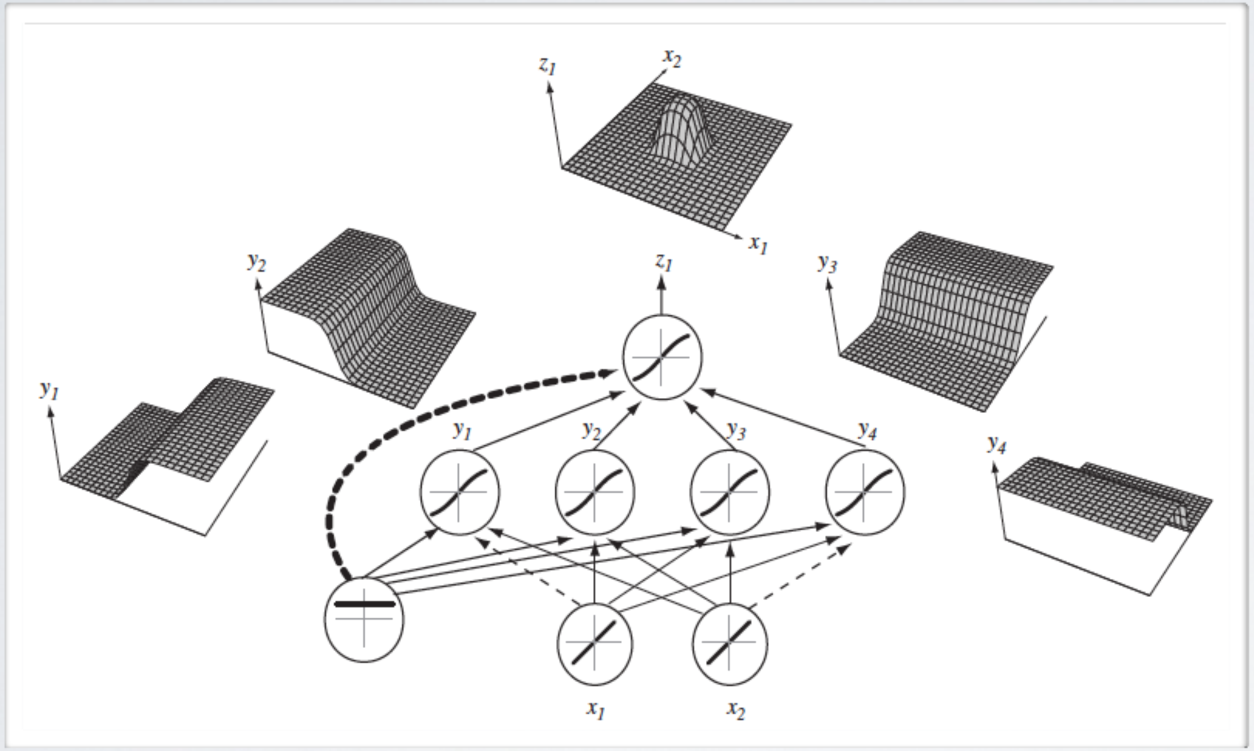
Topics: single hidden layer neural network



(from Pascal Vincent's slides)

CAPACITY OF NEURAL NETWORK

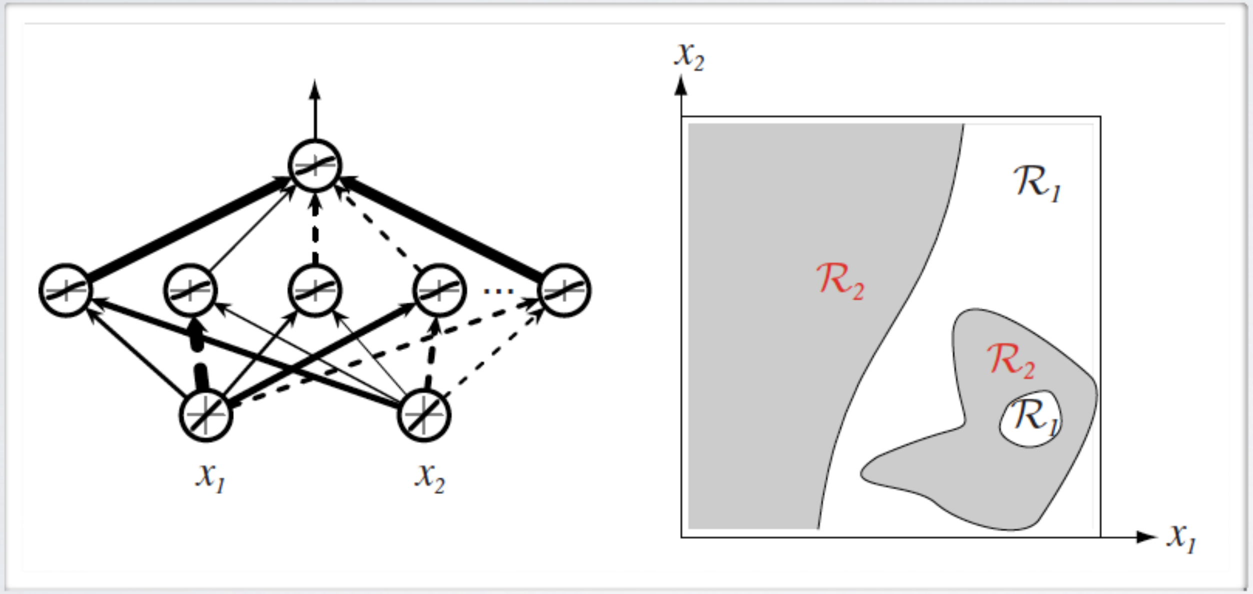
Topics: single hidden layer neural network



(from Pascal Vincent's slides)

CAPACITY OF NEURAL NETWORK

Topics: single hidden layer neural network

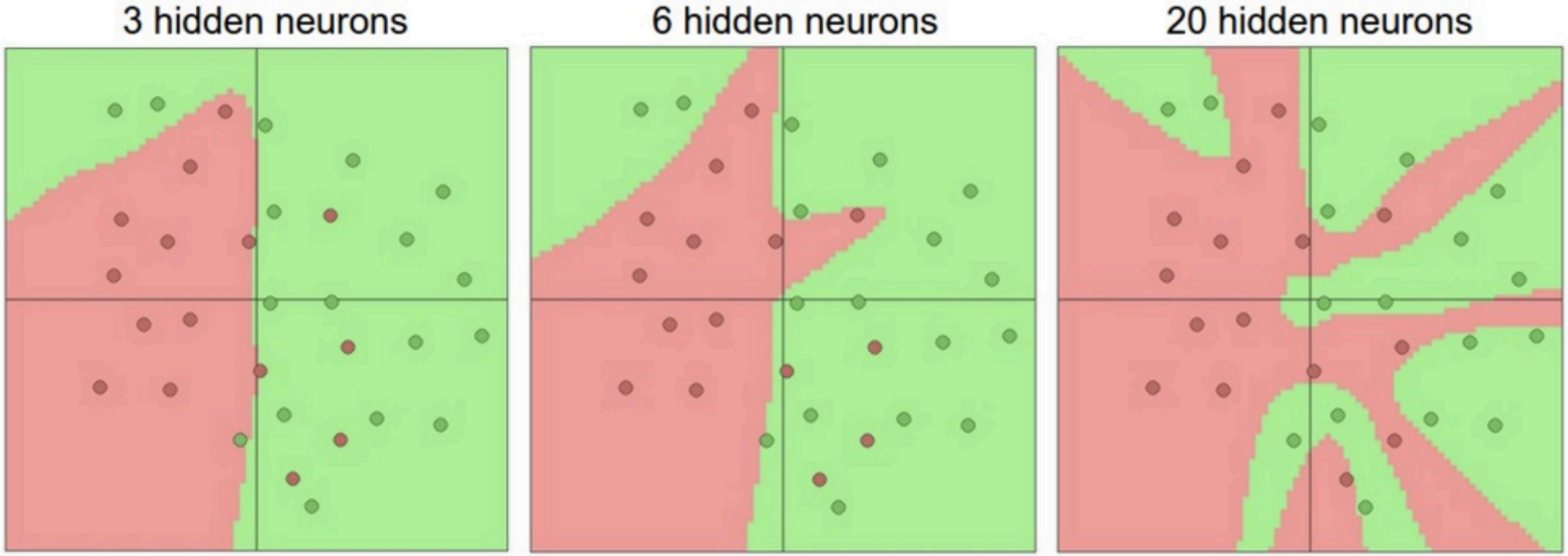


(from Pascal Vincent's slides)

CAPACITY OF NEURAL NETWORK

Topics: universal approximation

- Universal approximation theorem (Hornik, 1991):
  - "a single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units"
- The result applies for sigmoid, tanh and many other hidden layer activation functions
- This is a good result, but it doesn't mean there is a learning algorithm that can find the necessary parameter values!


3 hidden neurons
6 hidden neurons
20 hidden neurons

How to train a neural network?

Topics: multilayer neural network

- Could have L hidden layers:
  - layer input activation for $k>0$ $\quad (\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x})$

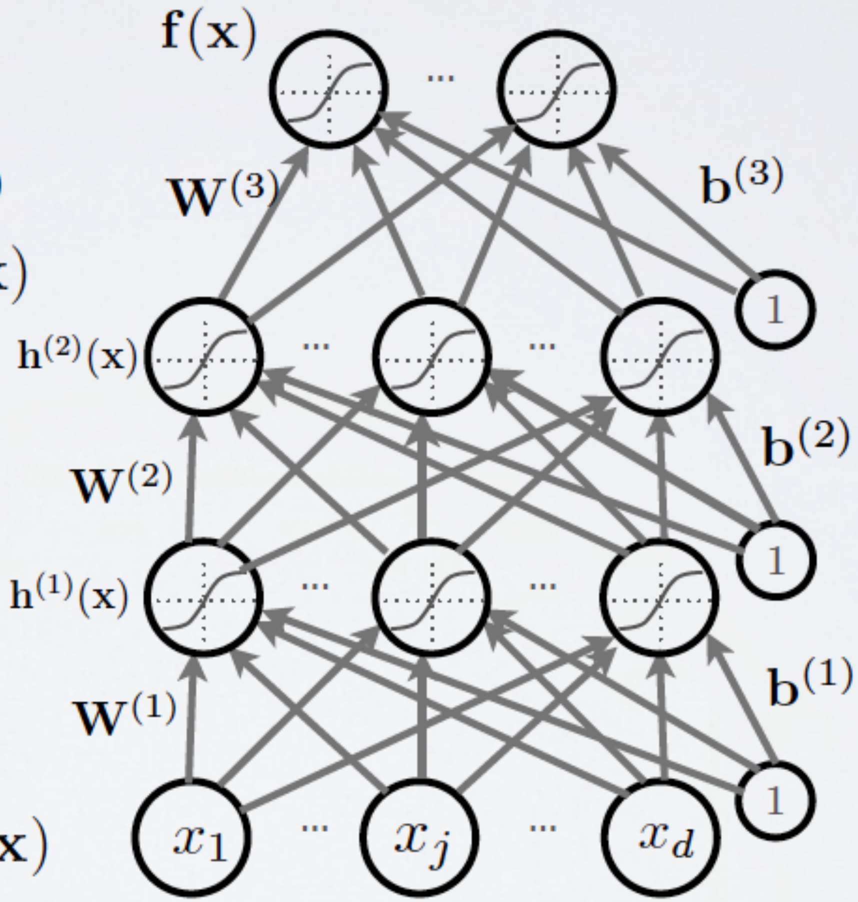
$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$

  - hidden layer activation (k from 1 to L):

$$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x}))$$

  - output layer activation ($k=L+1$):

$$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$



Empirical Risk Minimization

Topics: empirical risk minimization, regularization

- Empirical risk minimization
  - framework to design learning algorithms

$$\arg\min_{\boldsymbol{\theta}} \frac{1}{T} \sum_t l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) + \lambda \Omega(\boldsymbol{\theta})$$

  - $l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$ is a loss function
  - $\Omega(\boldsymbol{\theta})$ is a regularizer (penalizes certain values of $\boldsymbol{\theta}$)
- Learning is cast as optimization
  - ideally, we'd optimize classification error, but it's not smooth
  - loss function is a surrogate for what we truly should optimize (e.g. upper bound)

LOSS FUNCTION

Topics: loss function for classification

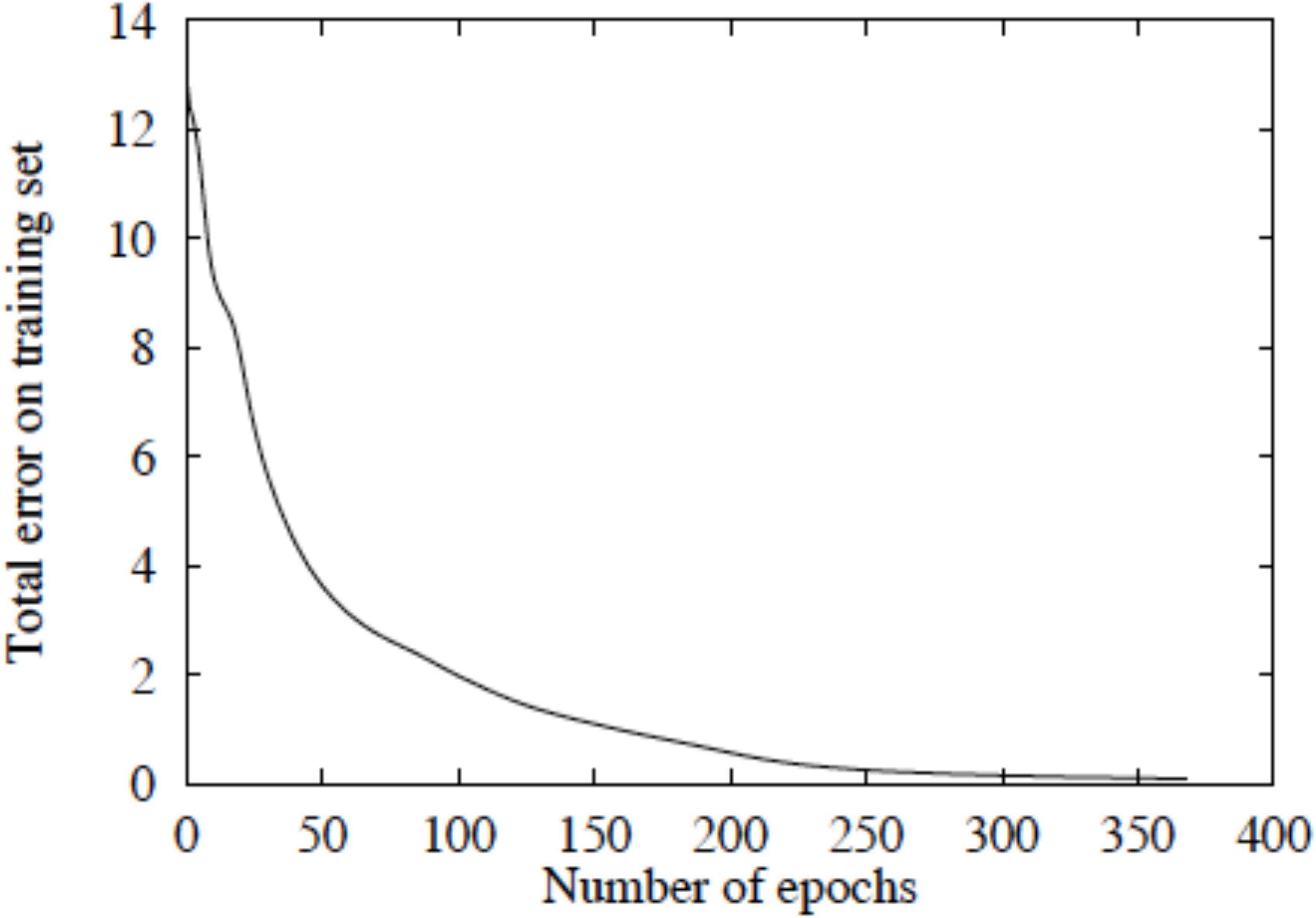
- Neural network estimates $f(\mathbf{x})_c = p(y = c|\mathbf{x})$
  - we could maximize the probabilities of $y^{(t)}$ given $\mathbf{x}^{(t)}$ in the training set

- To frame as minimization, we minimize the negative log-likelihood

natural log (ln)

$$l(\mathbf{f}(\mathbf{x}), y) = -\sum_c 1_{(y=c)} \log f(\mathbf{x})_c = -\log f(\mathbf{x})_y$$

  - we take the log to simplify for numerical stability and math simplicity
  - sometimes referred to as cross-entropy



[figure from Greg Mori's slides]

REGULARIZATION

Topics: L2 regularization

$$\Omega(\boldsymbol{\theta}) = \sum_k \sum_i \sum_j \left(W_{i,j}^{(k)}\right)^2 = \sum_k ||\mathbf{W}^{(k)}||_F^2$$

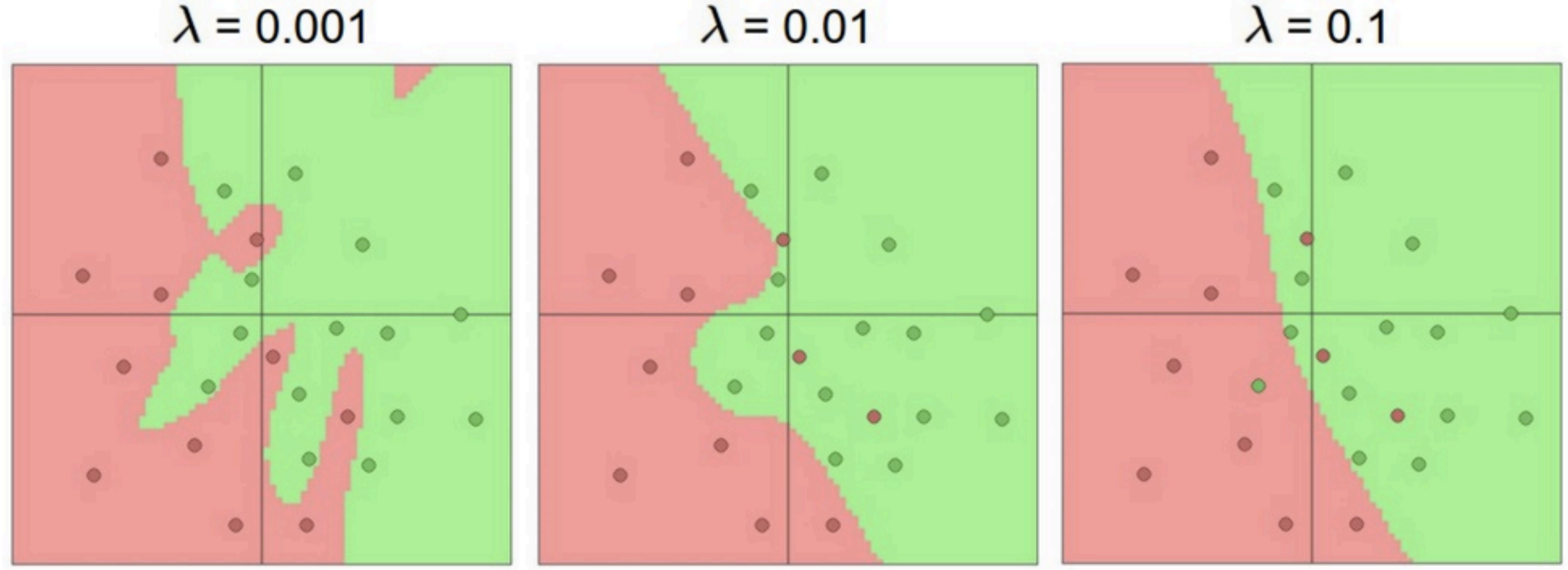
Empirical Risk Minimization

Topics: empirical risk minimization, regularization

- Empirical risk minimization
  - framework to design learning algorithms

$$\arg\min_{\boldsymbol{\theta}} \frac{1}{T} \sum_{t} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) + \lambda \Omega(\boldsymbol{\theta})$$

  - $l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$ is a loss function
  - $\Omega(\boldsymbol{\theta})$ is a regularizer (penalizes certain values of $\boldsymbol{\theta}$)
- Learning is cast as optimization
  - ideally, we'd optimize classification error, but it's not smooth
  - loss function is a surrogate for what we truly should optimize (e.g. upper bound)



[http://cs231n.github.io/neural-networks-1/]

INITIALIZATION

Topics: initialization

- For biases
  - initialize all to 0
- For weights
  - Can't initialize weights to 0 with tanh activation
    - we can show that all gradients would then be 0 (saddle point)
  - Can't initialize all weights to the same value
    - we can show that all hidden units in a layer will always behave the same
    - need to break symmetry
  - Recipe: sample $\mathbf{W}_{i,j}^{(k)}$ from $U\left[-b, b\right]$ where $b = \frac{\sqrt{6}}{\sqrt{H_k + H_{k-1}}}$

    size of $\mathbf{h}^{(k)}(\mathbf{x})$
    - the idea is to sample around 0 but break symmetry
    - other values of b could work well (not an exact science) (see Glorot & Bengio, 2010)

Model Learning

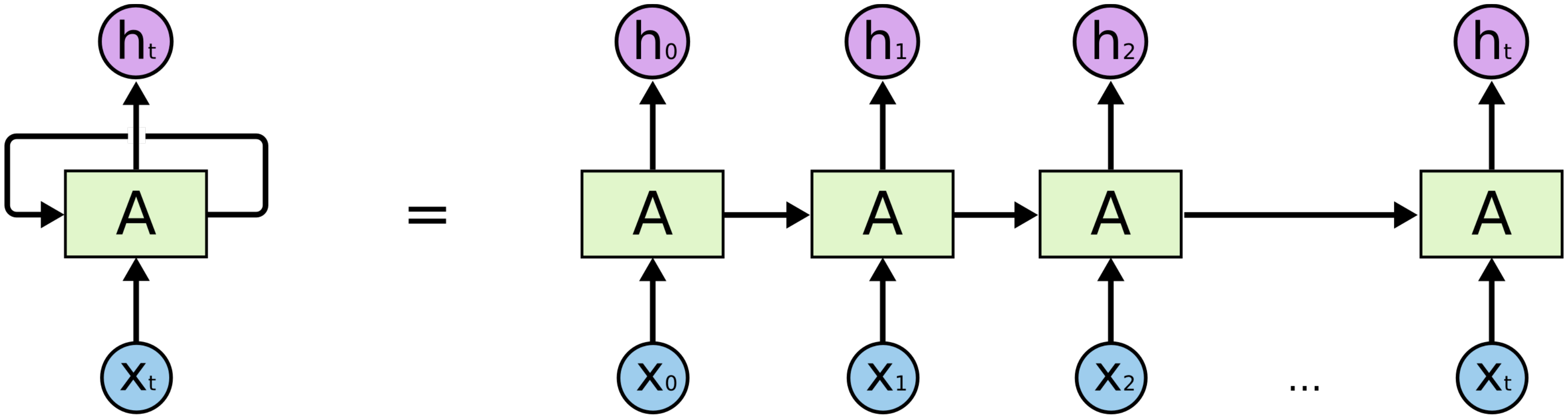
- Backpropagation (BP) algorithm (not required for this course)
- Further reading on BP:
  - https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd
  - https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Outline

- Maximum Entropy
- Feedforward Neural Networks
- Recurrent Neural Networks

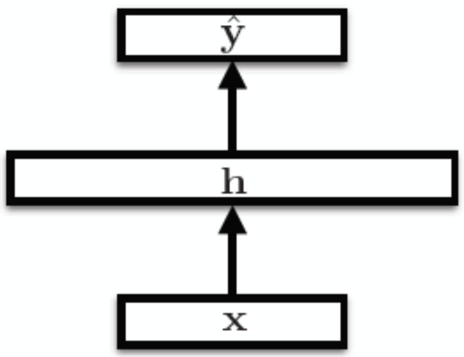
Long Distance Dependencies

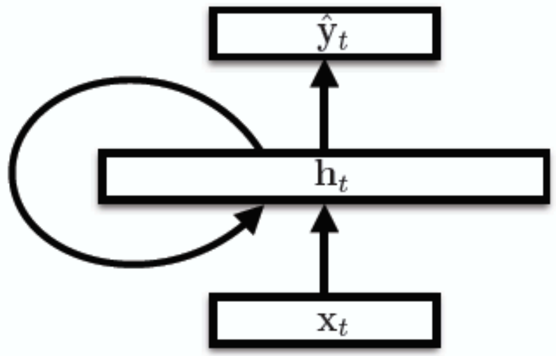
- It is very difficult to train NNs to retain information over many time steps
- This makes it very difficult to handle long-distance dependencies, such as subject-verb agreement.
- E.g. Jane walked into the room. John walked in too. It was late in the day. Jane said hi to _?_



Recurrent Neural Networks

Feed-forward NN	Recurrent NN
$\mathbf{h} = g(\mathbf{V}\mathbf{x} + \mathbf{c})$	$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$
$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h} + \mathbf{b}$	$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$



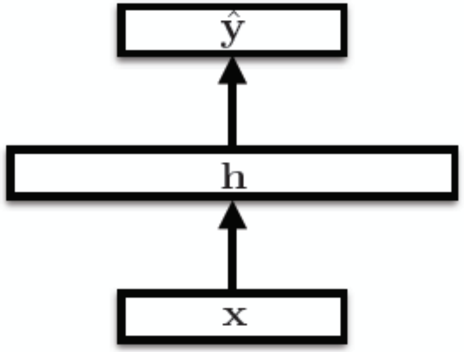


Recurrent Neural Networks

Feed-forward NN

$$\mathbf{h} = g(\mathbf{V}\mathbf{x} + \mathbf{c})$$

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h} + \mathbf{b}$$

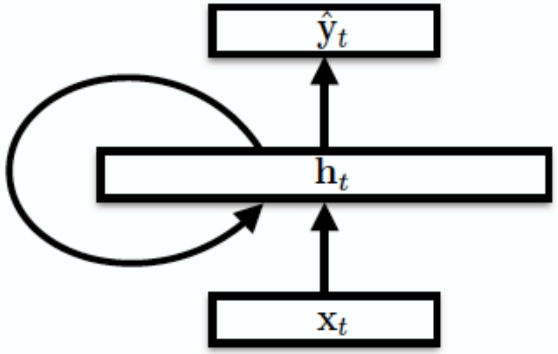


Recurrent NN

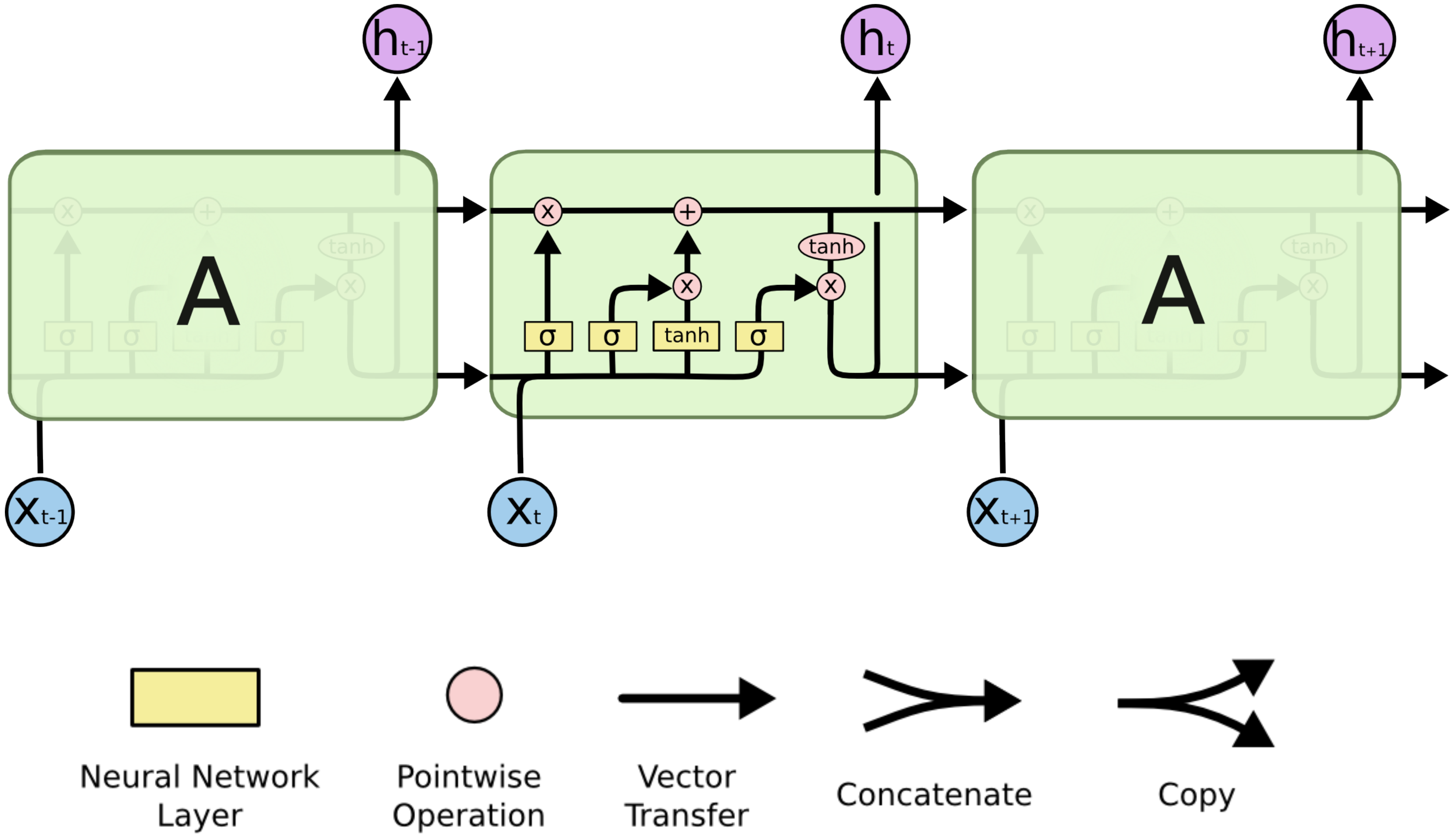
$$\cancel{\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})}$$

$$\mathbf{h}_t = g(\mathbf{V}[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{c})$$

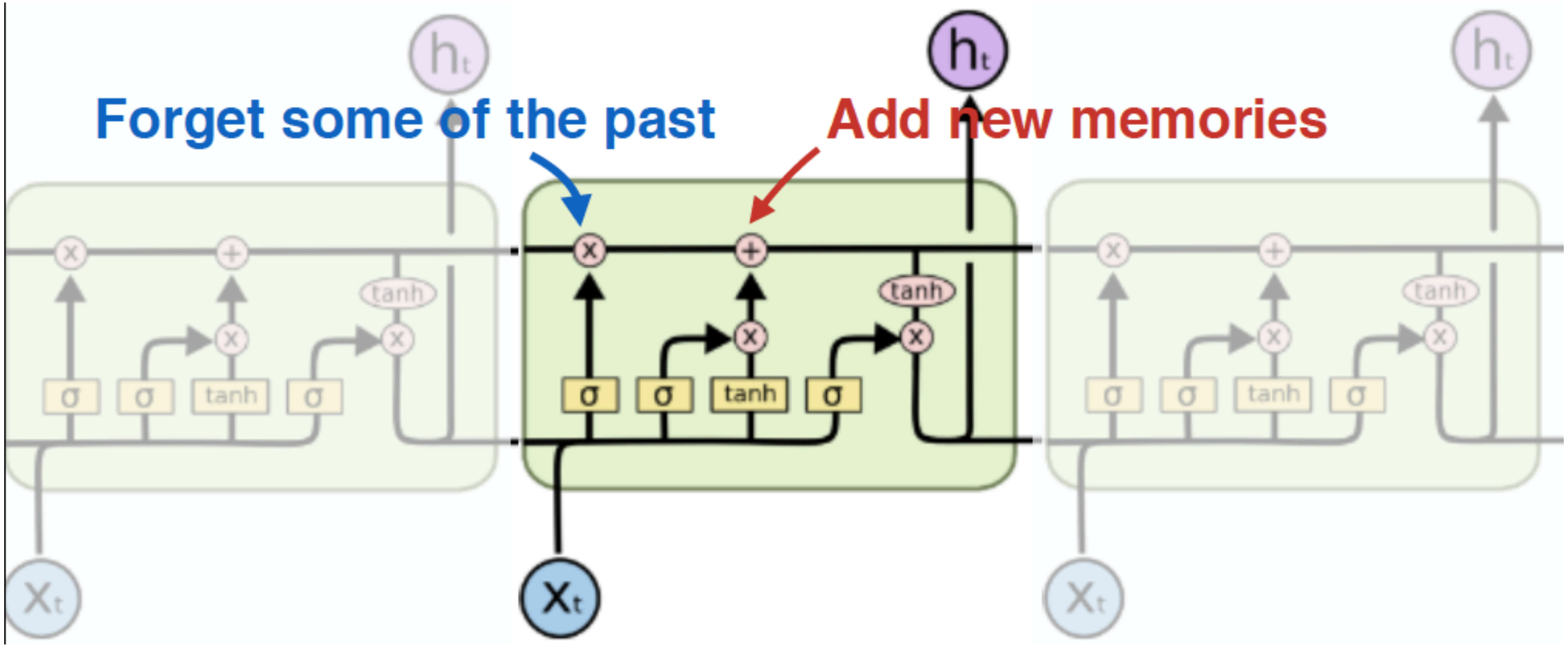
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



Long-Short Term Memory Networks (LSTMs)



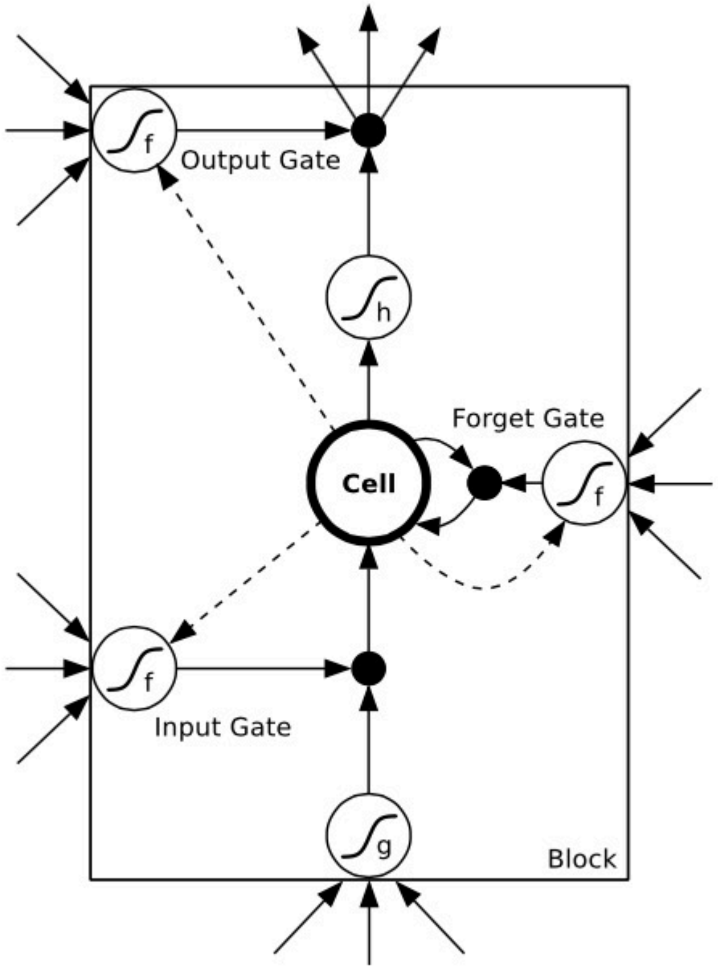
Another Visualization



Capable of modeling long-distant dependencies between states.

Figure: Christopher Olah

Long-Short Term Memory Networks (LSTMs)



$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma(W_i[x_t, h_t] + b_i) \\ \sigma(W_f[x_t, h_t] + b_f) \\ \sigma(W_o[x_t, h_t] + b_o) \\ f(W_g[x_t, h_t] + b_g) \end{pmatrix}$$

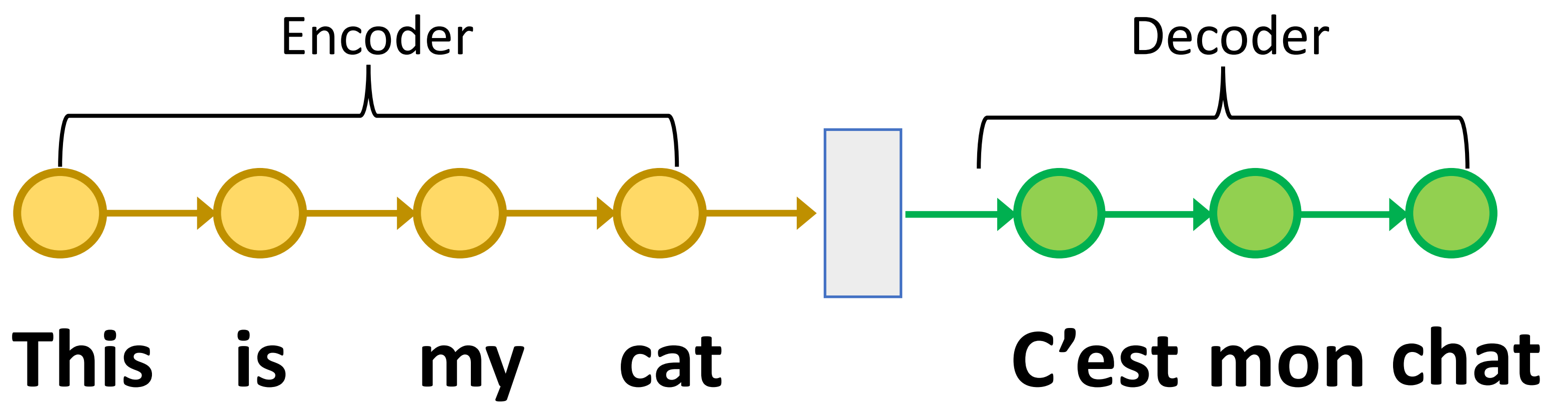
$$c_t = f_t * c_{t-1} + i_t * g_t$$

$$h_t = o_t * f(c_t)$$

Use gates to control the information to be added from the input, forgot from the previous memories, and outputted.
σ and f are *sigmoid* and *tanh* function respectively, to map the value to [-1, 1]

Sequence to Sequence

- Encoder/Decoder framework maps one sequence to a "deep vector" then another LSTM maps this vector to an output sequence.



Summary of LSTM Application Architectures

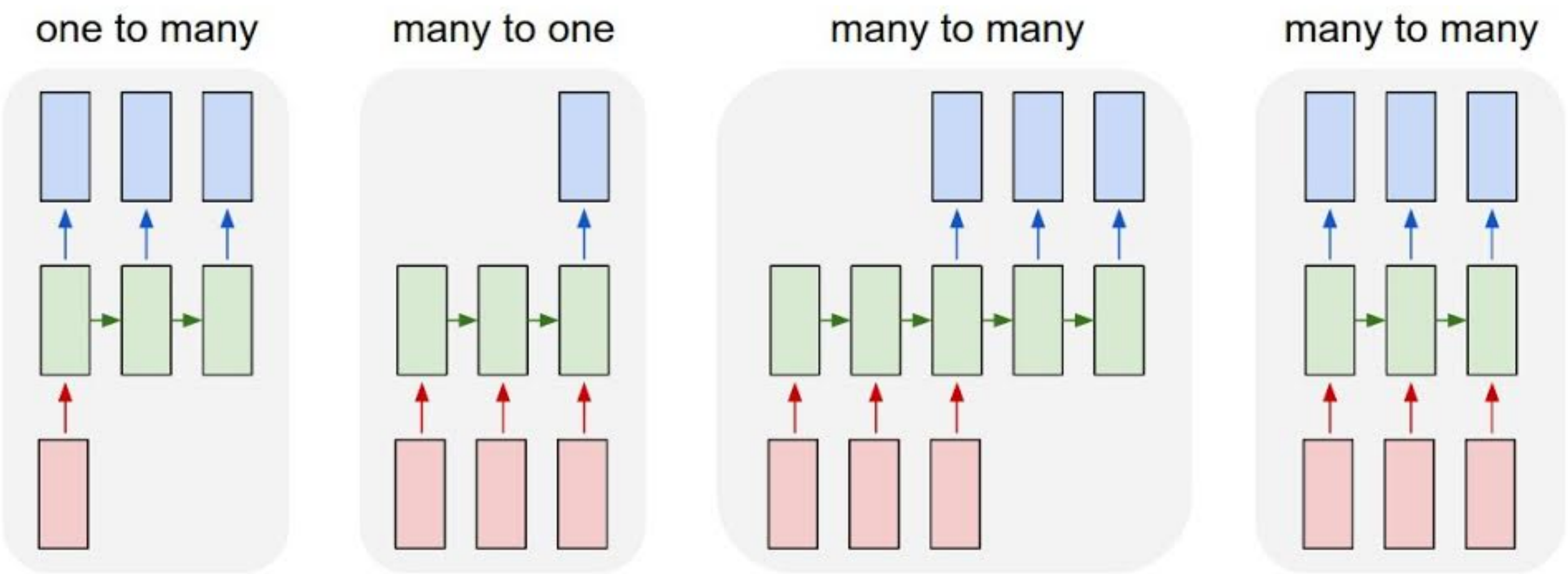


Image Captioning

Video Activity Recog
Text Classification

Video Captioning
Machine Translation

POS Tagging
Language Modeling

Successful Applications of LSTMs

- Speech recognition: Language and acoustic modeling
- Sequence labeling
  - POS Tagging
  - NER
  - Phrase Chunking
- Neural syntactic and semantic parsing
- Image captioning
- Sequence to Sequence
  - Machine Translation (Sustkever, Vinyals, & Le, 2014)
  - Video Captioning (input sequence of CNN frame outputs)