COL333/671: Introduction to AI
Semester I, 2021

Learning – II

Rohan Paul

Outline

- Last Class
  - Basics of machine learning
- This Class
  - Neural Networks
- Reference Material
  - Please follow the notes as the primary reference on this topic. Additional reading from AIMA book Ch. 18 (18.2, 18.6 and 18.7) and DL book Ch 6 sections 6.1 – 6.5 (except 6.4).

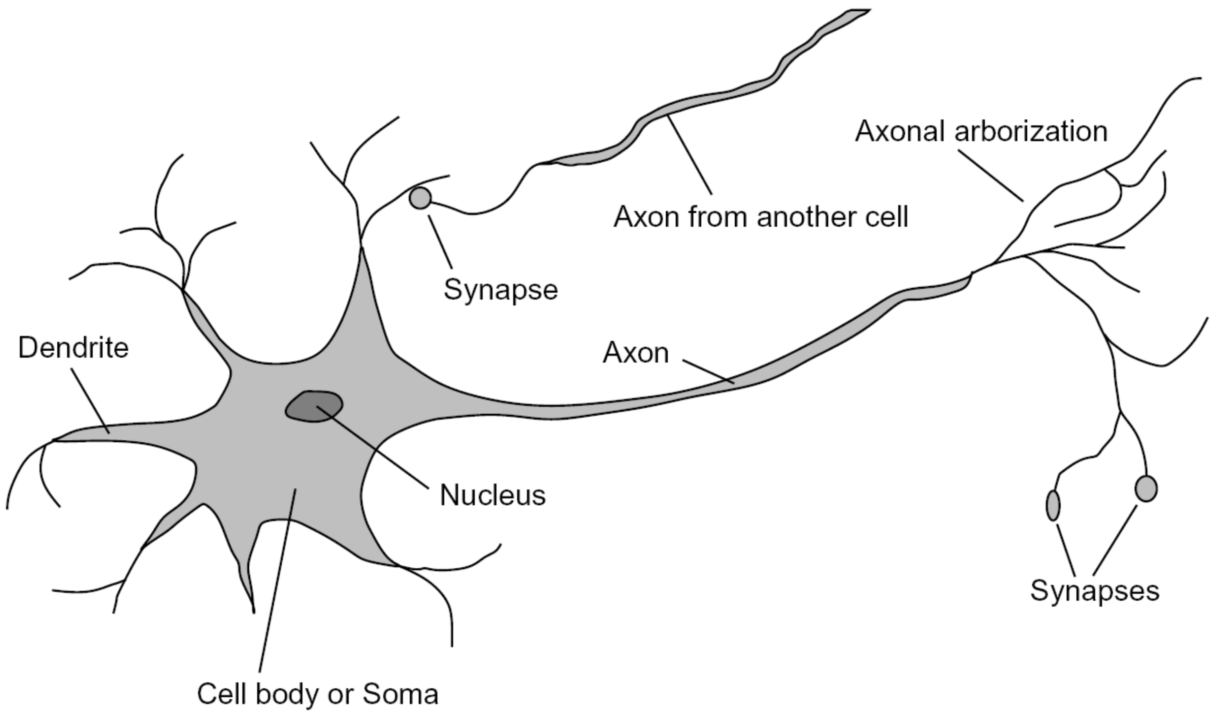
Acknowledgement

These slides are intended for teaching purposes only. Some material has been used/adapted from web sources and from slides by Doina Precup, Dorsa Sadigh, Percy Liang, Mausam, Parag, Emma Brunskill, Alexander Amini, Dan Klein, Anca Dragan, Nicholas Roy and others.

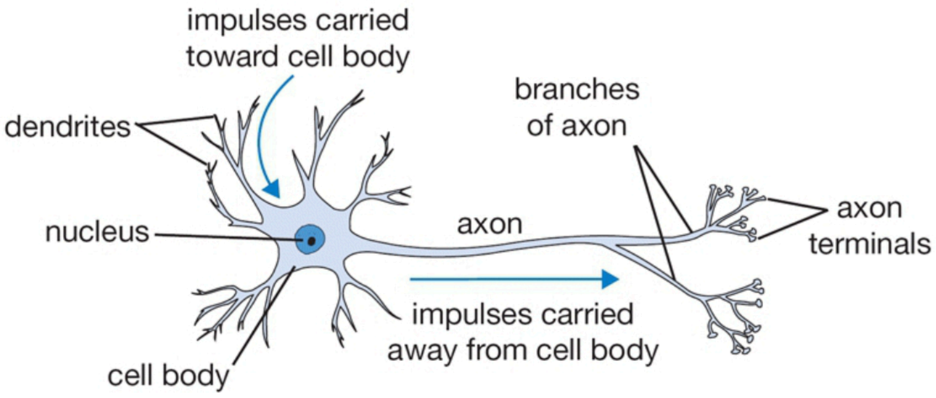
Neuron

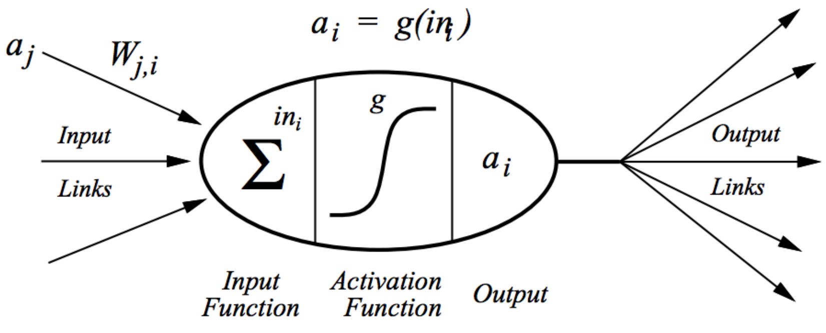
A simplified view

- Activations and inhibitions
- Parallelism
- Connected networks



Modeling a Neuron





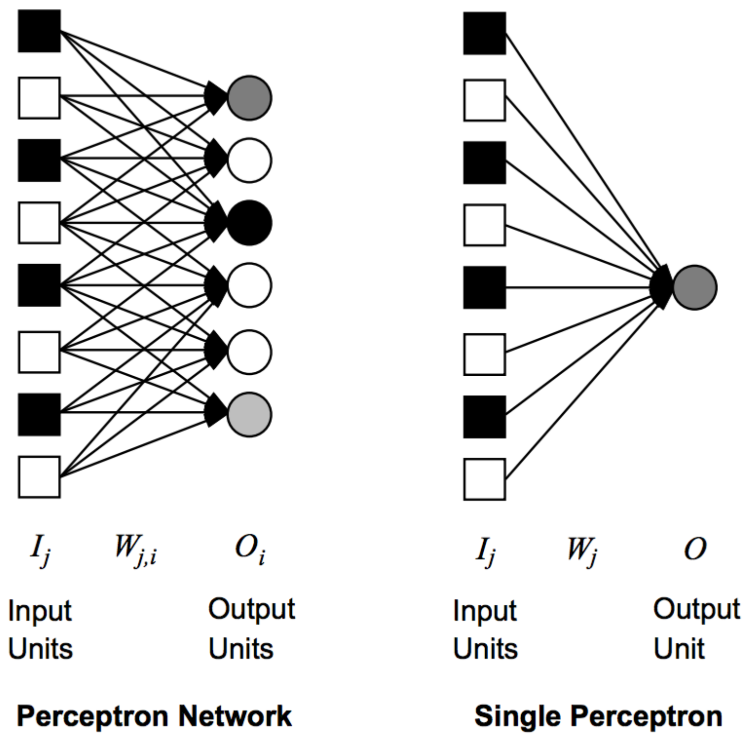
$$a_i = g(\sum_j W_{j,i} a_j)$$

Main processing unit

- Setup where there is a function that connects the inputs to the output.
- Problem: learning this function that maps the input to the output.

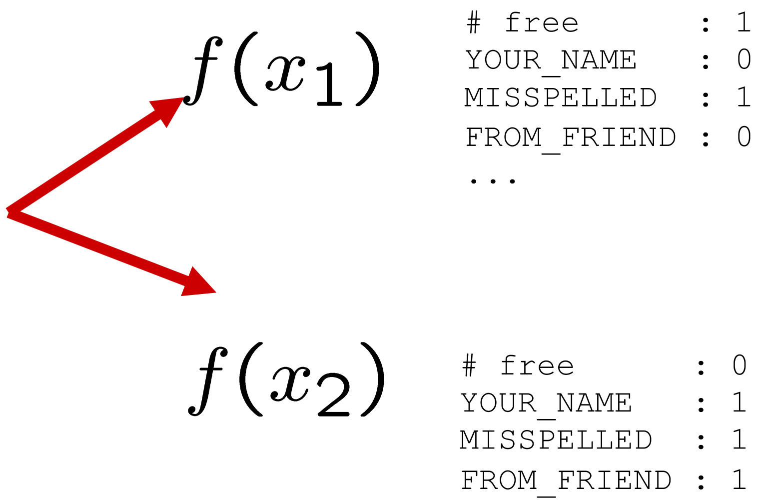
Perceptron

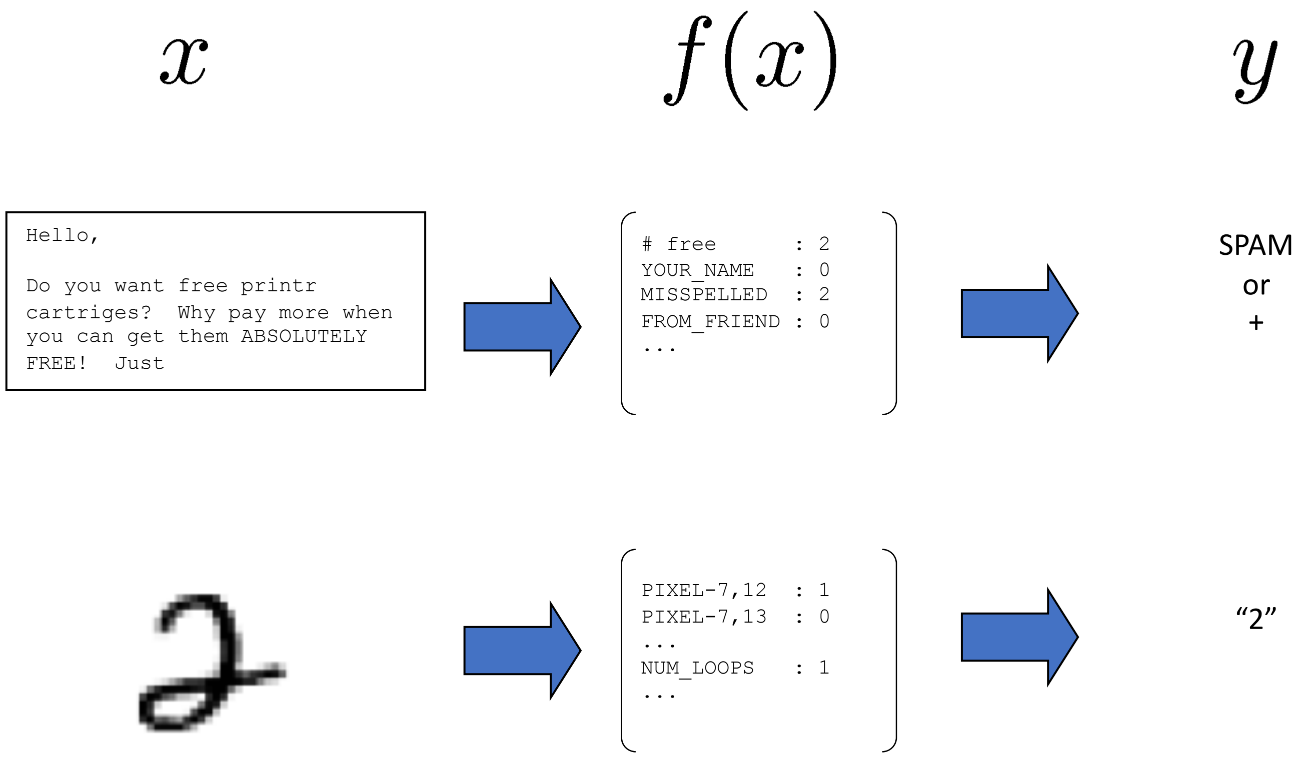
- Introduced in the late 50s
  - Minsky and Papert.
- Perceptron convergence theorem Rosenblatt 1962:
  - Perceptron will learn to classify any linearly separable set of inputs
- Note: the earlier class talked about model-based classification. Here, we do not build a model. Operate directly on feature weights.



Feature Space

- Extract Features from data
- Learn a model with these features
- Data can be viewed as a point in the feature space.

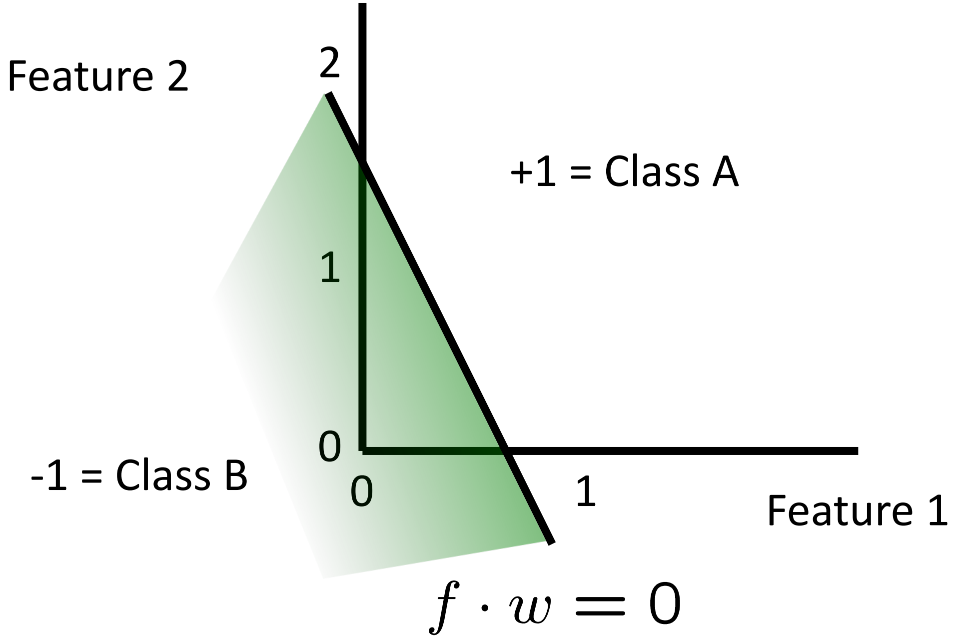


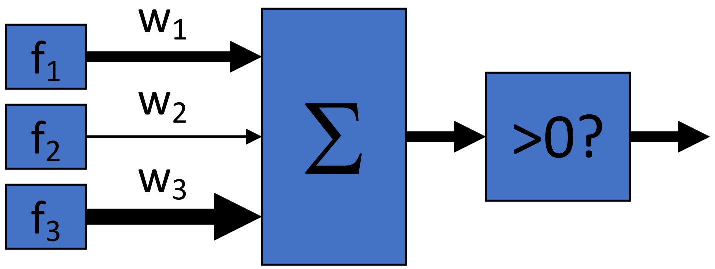


Linear Classification

- A decision boundary is a hyperplane orthogonal too the weight vector.

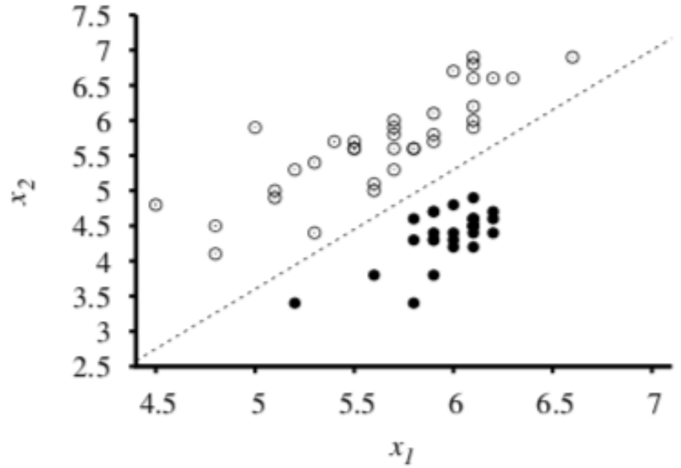
$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$





Perceptron

Decision rule (Binary case)



Binary classification task

$$h_{\mathbf{w}}(\mathbf{x}) = 1 \text{ if } \mathbf{w} \cdot \mathbf{x} \geq 0 \text{ and } 0 \text{ otherwise.}$$

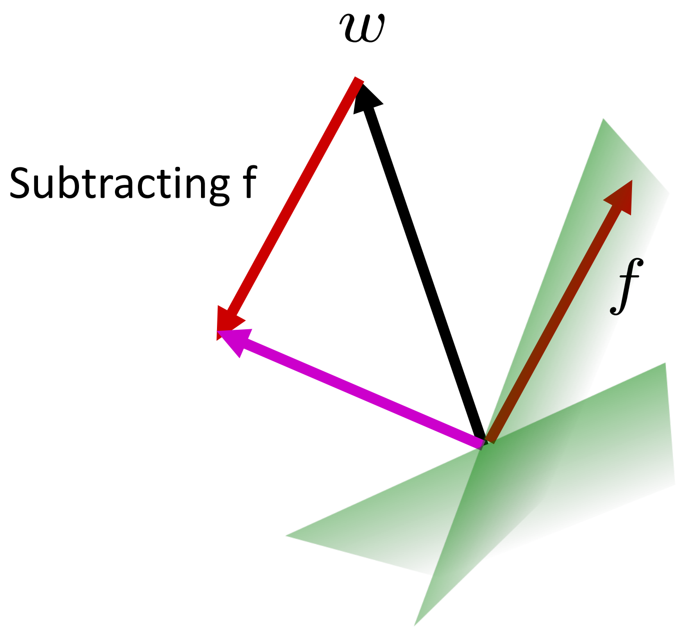
$$h_{\mathbf{w}}(\mathbf{x}) = \mathit{Threshold}(\mathbf{w} \cdot \mathbf{x}) \text{ where } \mathit{Threshold}(z) = 1 \text{ if } z \geq 0 \text{ and } 0 \text{ otherwise.}$$

One side of the decision boundary is class. A and the other is class B. A threshold is introduced.

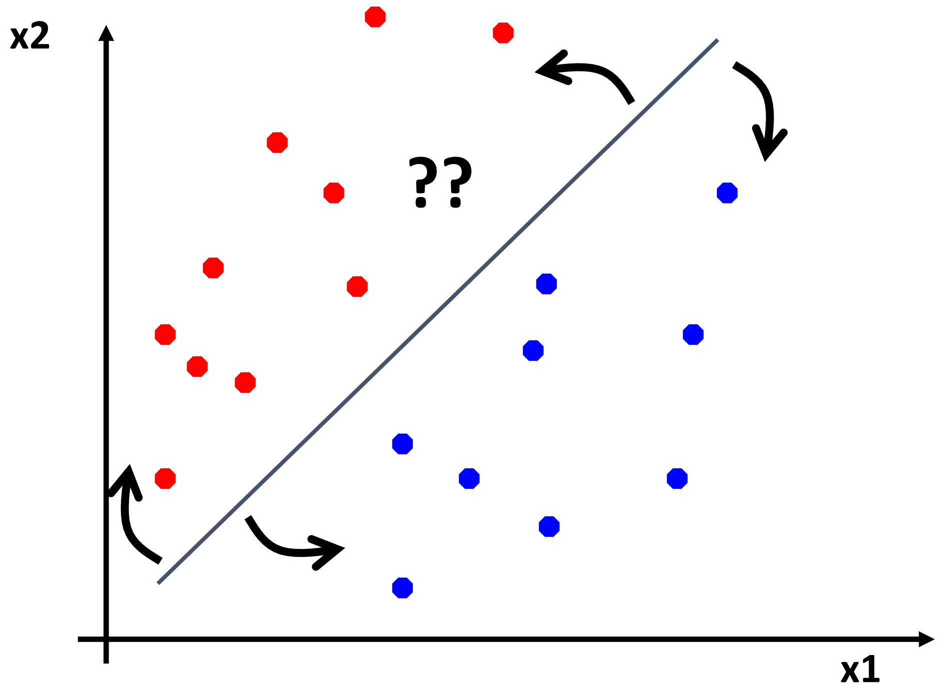
Perceptron

Learning rule

- Classify with the current weights
- If correct no change.
- If the classification is wrong: *adjust* the weight vector by adding or subtracting the feature vector.



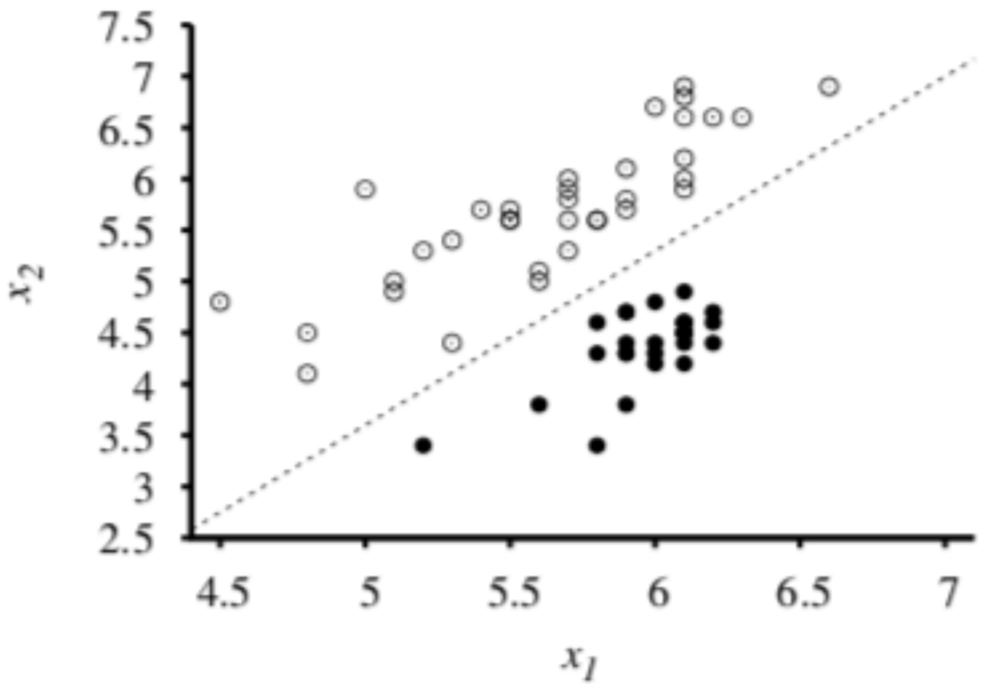
$$w_i \leftarrow w_i + \alpha\,(y - h_{\mathbf{w}}(\mathbf{x})) \times x_i$$



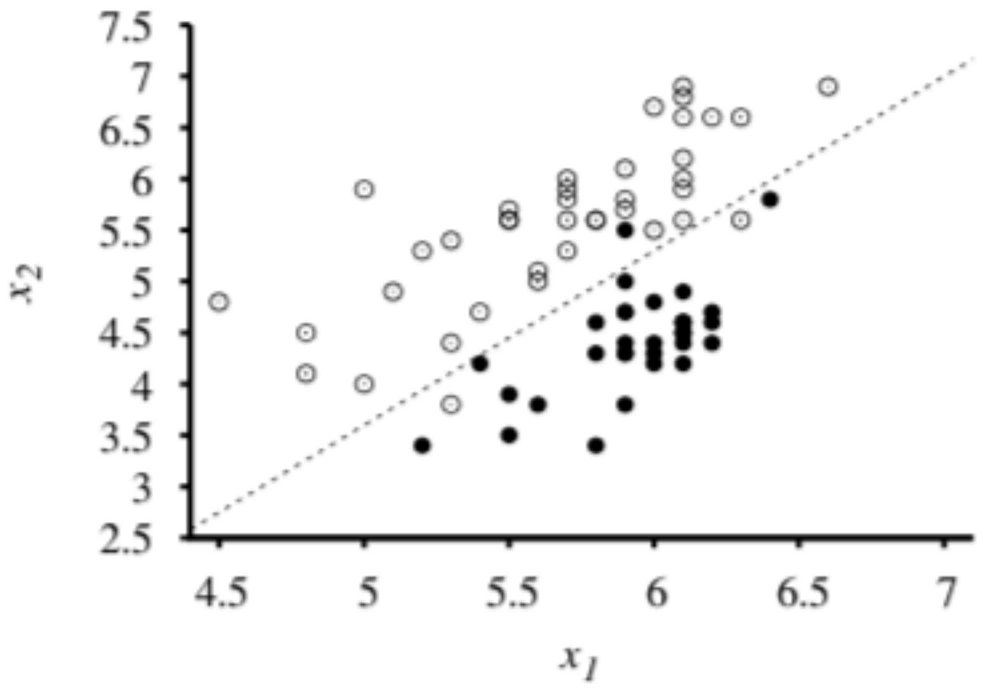
Binary classification task

Perceptron

Case: Linearly-separable

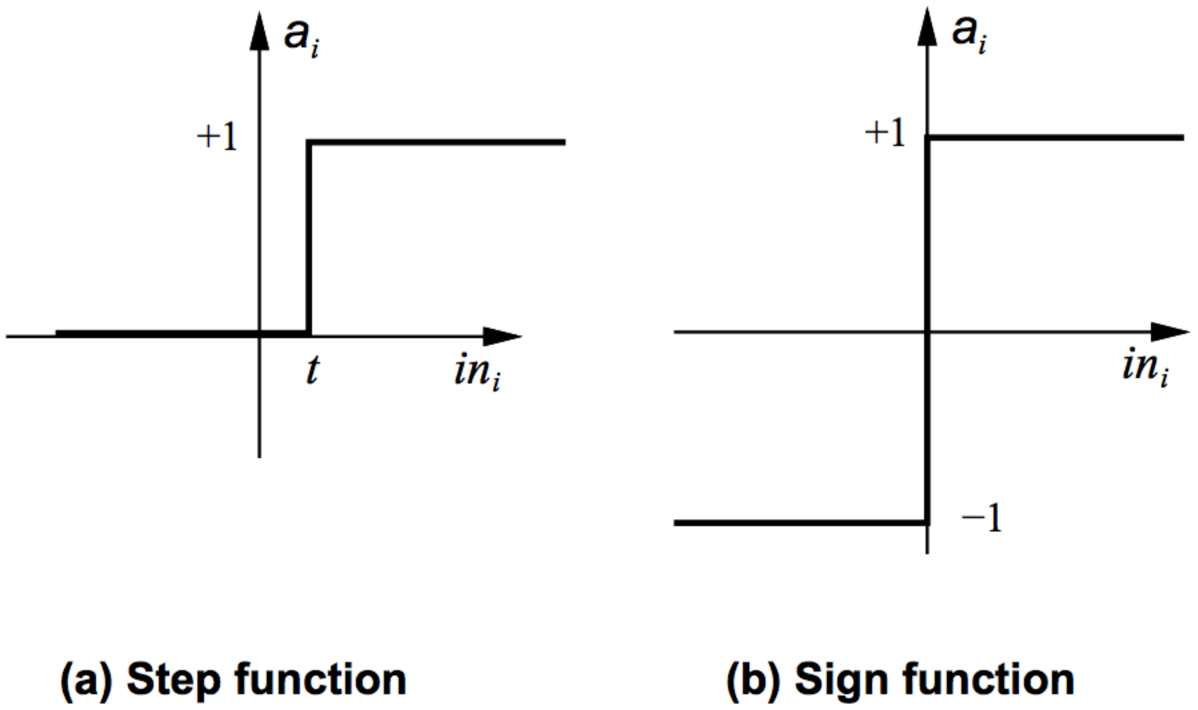


Case: Not Linearly-separable



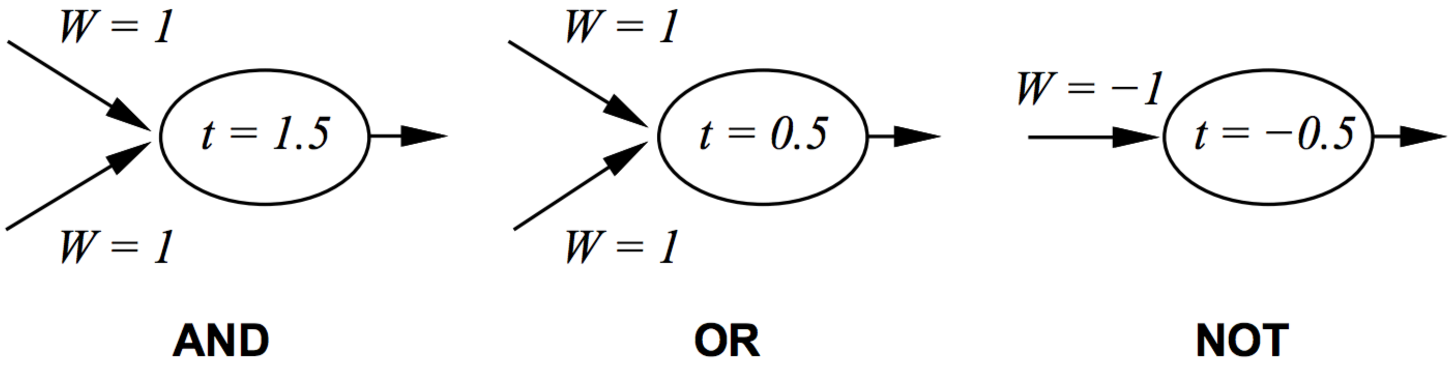
Perceptron learning rule converges to a perfect linear separator when the data points are linearly separable. Problem when there is non-separable data.

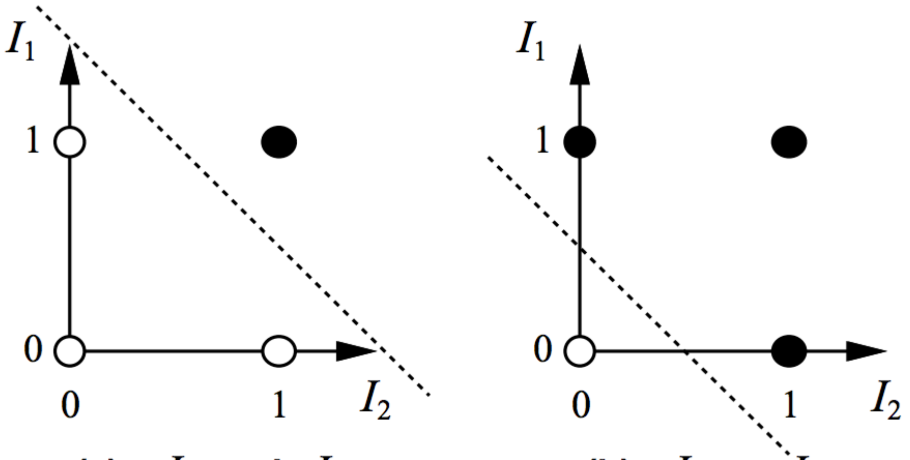
Threshold Functions



- Till now, threshold functions were linear.
- Can we modify the threshold function to handle the non-separable case?
- Can we ”soften” the outputs?

Boolean Functions and Perceptron

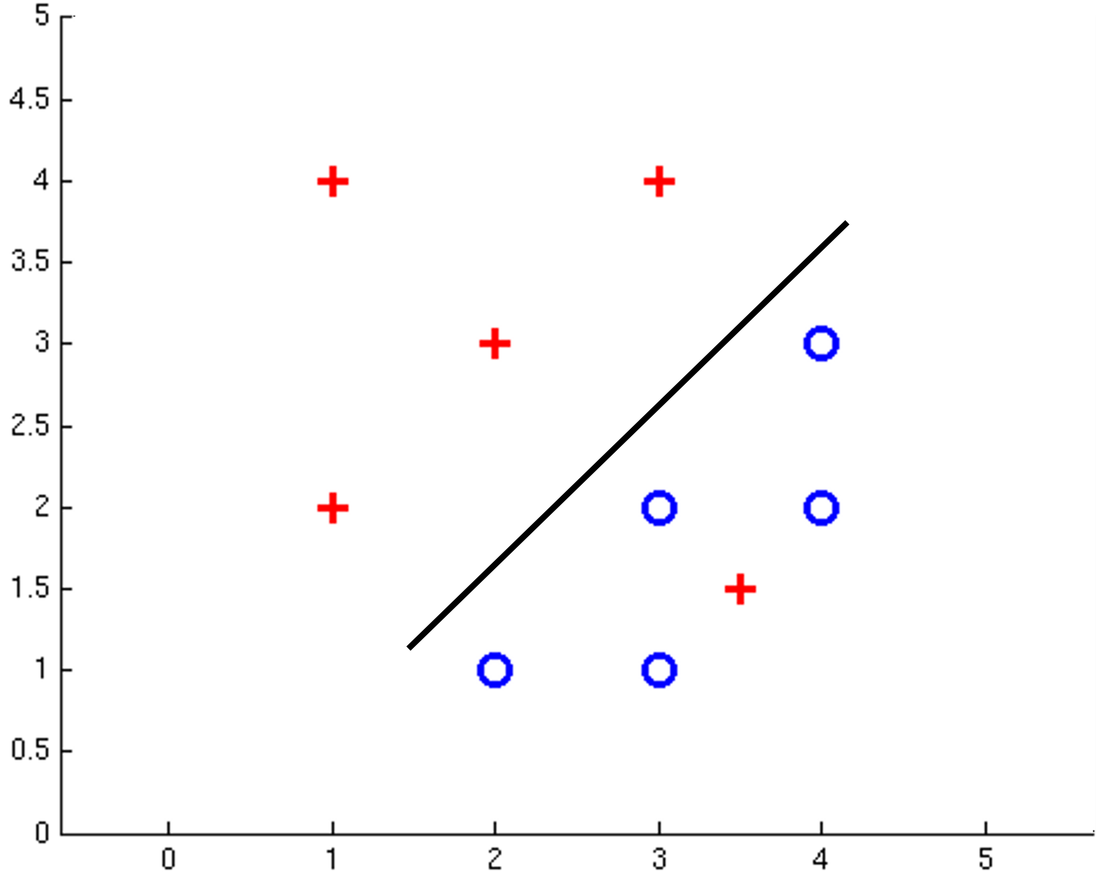




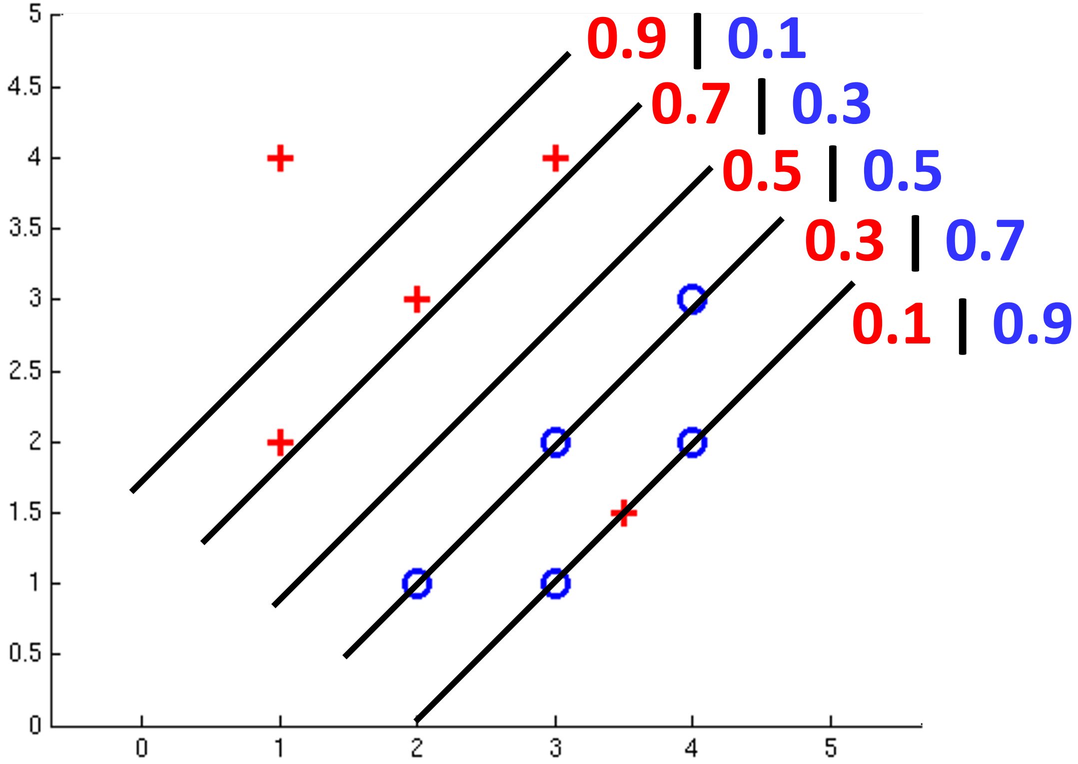
(a) I_1 **and** I_2 **(b)** I_1 **or** I_2

Non-separable case

Deterministic Decisions



Probabilistic Decisions

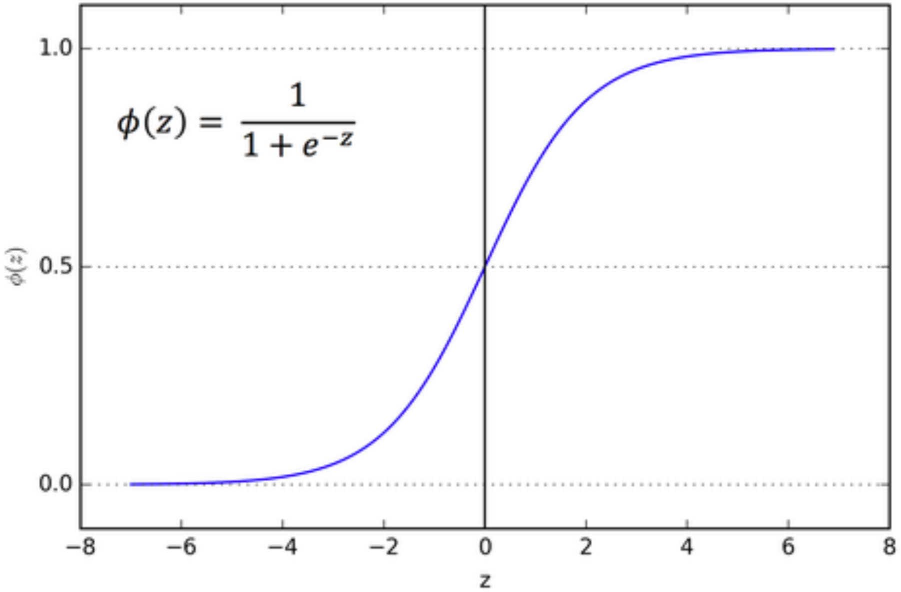


Logistic Output

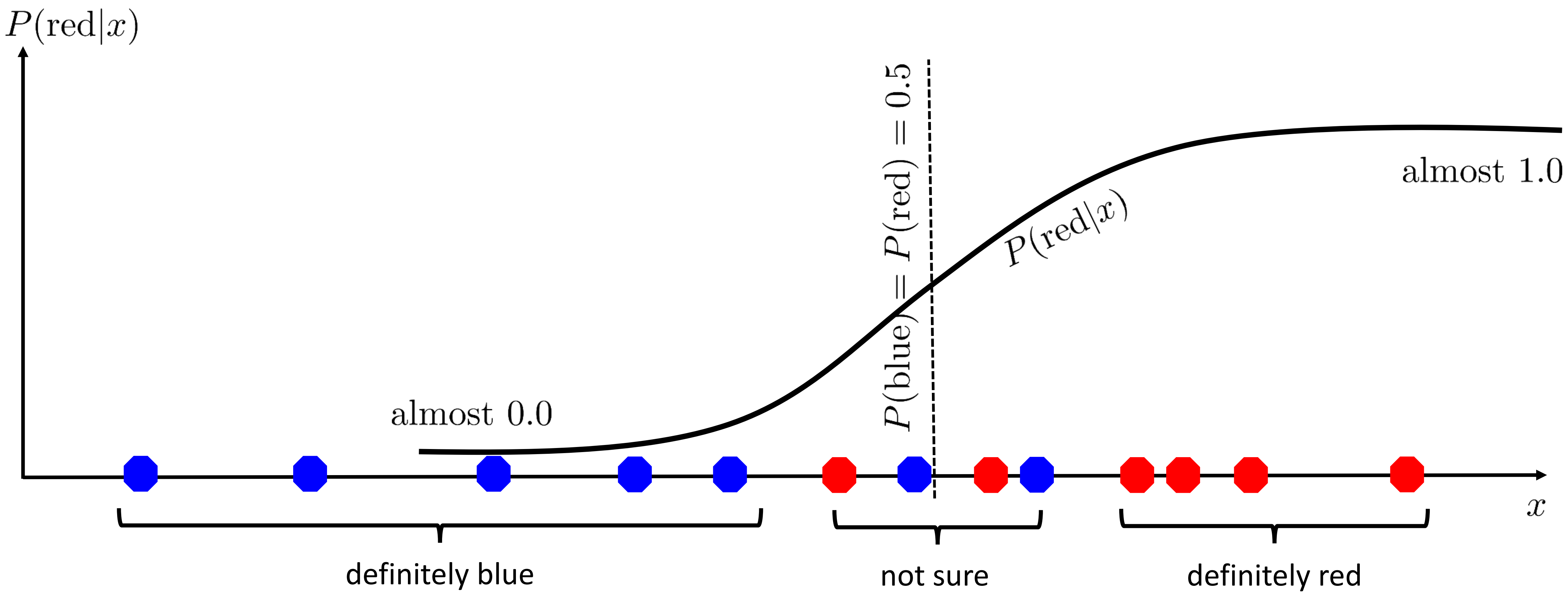
- **Logistic Function**
  - Very positive values. Probability -> 1
  - Very negative values. Probability -> 0.
  - Makes the prediction. Converts to a probability
  - Softens the decision boundary.
- **Logistic Regression**
  - Fitting the weights of this model to minimize loss on a data set is called logistic regression.

$$Logistic(z) = \frac{1}{1+e^{-z}}$$

$$h_{\mathbf{w}}(\mathbf{x}) = Logistic(\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1+e^{-\mathbf{w} \cdot \mathbf{x}}}$$



Example (red or blue classes)



$$P(\text{red}|x) = \frac{e^{w_{\text{red}} \cdot x}}{e^{w_{\text{red}} \cdot x} + e^{w_{\text{blue}} \cdot x}}$$

probability increases exponentially as we move away from boundary

Normalizer

Estimating weights using MLE

Logistic Regression
Maximize the log-likelihood

$$\max_w \; ll(w) = \max_w \; \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

$$P(y^{(i)} = +1|x^{(i)}; w) = \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$

$$P(y^{(i)} = -1|x^{(i)}; w) = 1 - \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$

Softmax Output

- Multi-class setting
  - A probability distribution over a discrete variable with n possible values.
  - Generalization of the sigmoid function to multiple outputs.
- Output of a classifier
  - Distribution over n different classes. The individual outputs must sum to one.

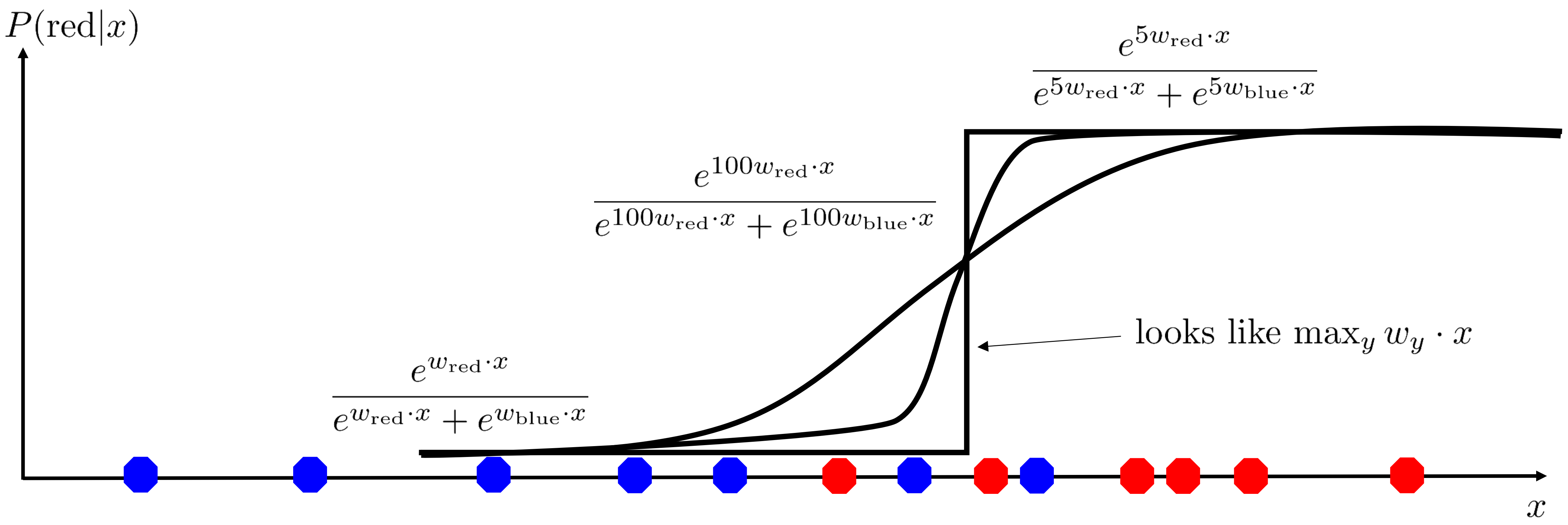
Prediction of the unnormalized probabilities.

$$z_i = \log \tilde{P}(y = i \mid \boldsymbol{x})$$

Exponentiate and normalize the values.

$$\text{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}.$$

Softmax Example



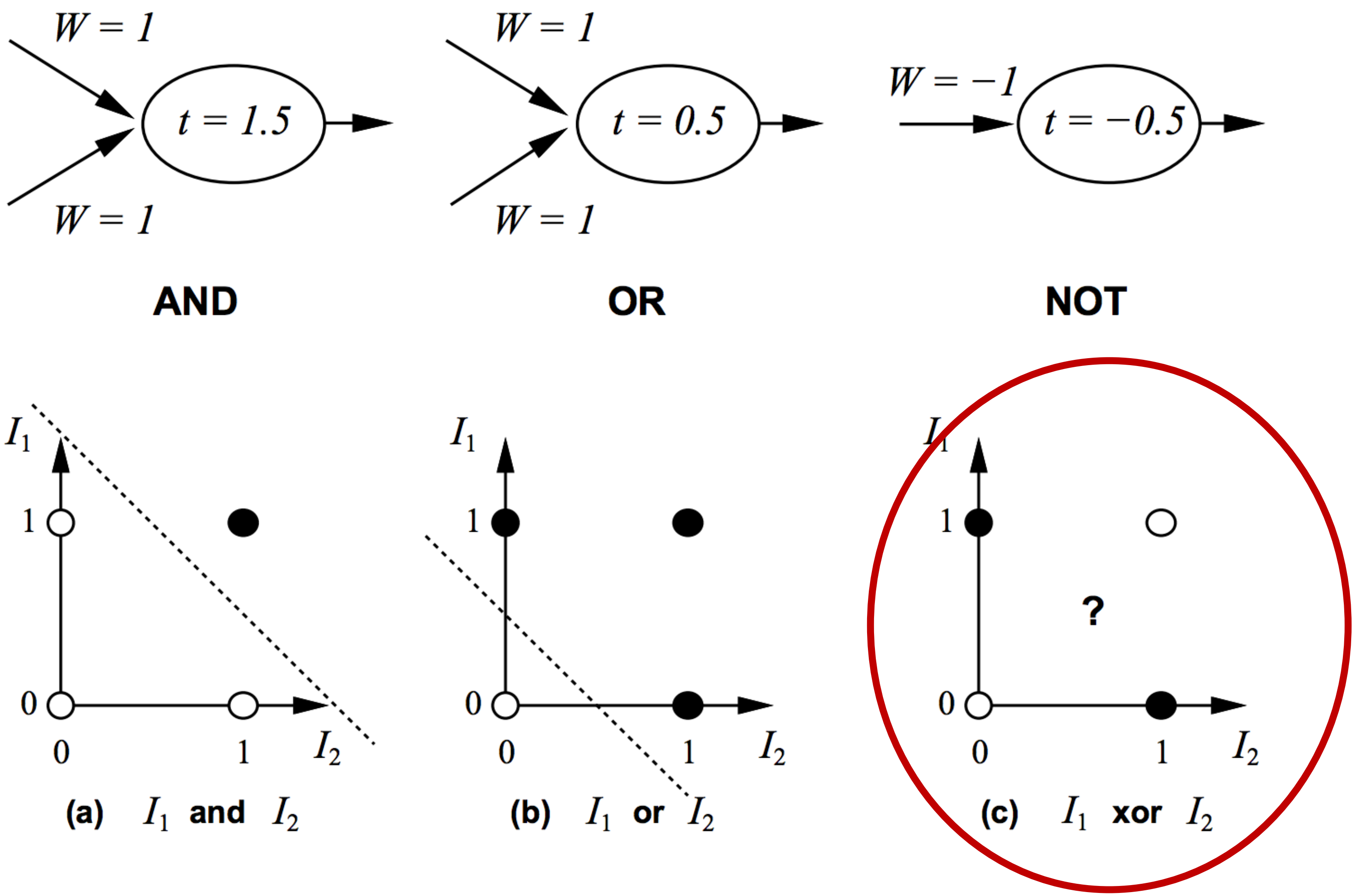
$$P(\text{red}|x) = \frac{e^{w_{\text{red}}\cdot x}}{e^{w_{\text{red}}\cdot x} + e^{w_{\text{blue}}\cdot x}}$$

Multi-class Logistic Regression

$$\max_{w} \quad ll(w) = \max_{w} \quad \sum_{i} \log P(y^{(i)}|x^{(i)}; w)$$

$$P(y^{(i)}|x^{(i)}; w) = \frac{e^{w_{y^{(i)}} \cdot f(x^{(i)})}}{\sum_{y} e^{w_y \cdot f(x^{(i)})}}$$

Can a perceptron learn XOR?



Non-separability and Non-linear Functions

- The original feature space is mapped to some higher-dimensional feature space wher the training set is separable.
- Need a non-linear function to describe the features.
- Applying a non-linear kernel map. Affine transformation.

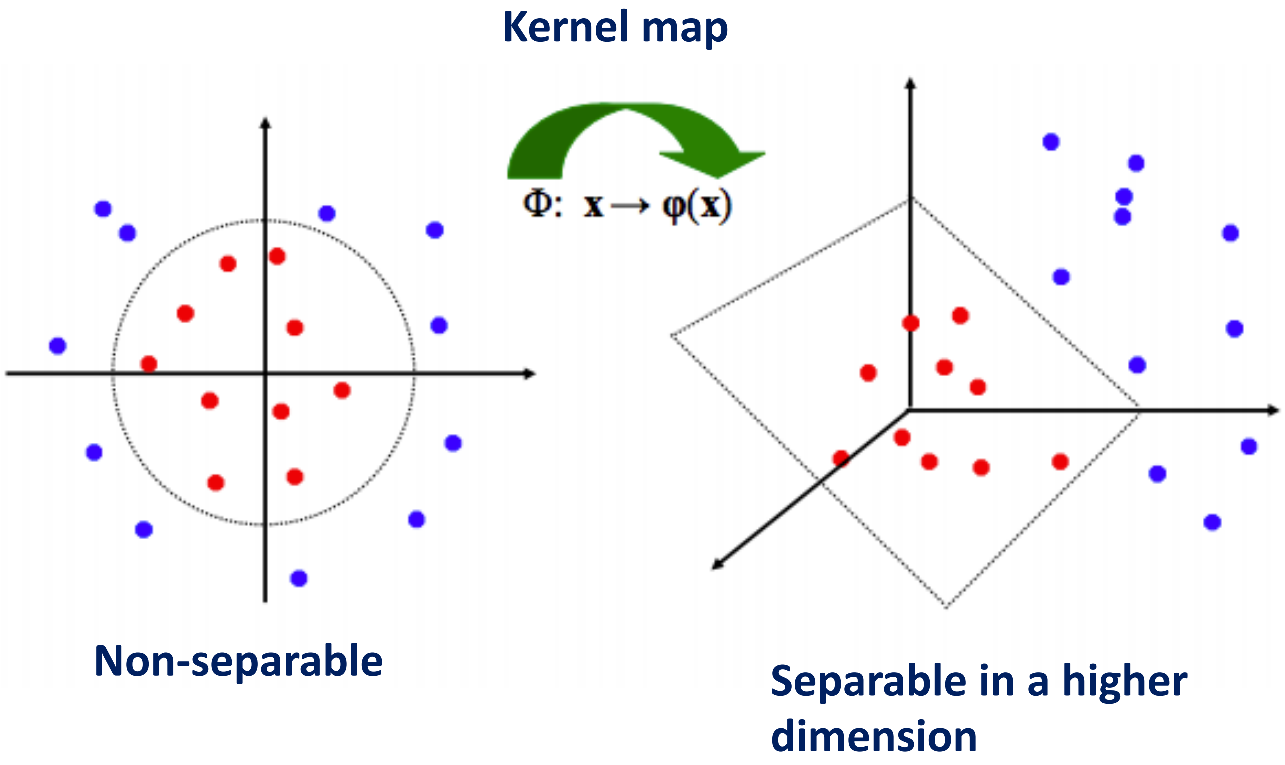
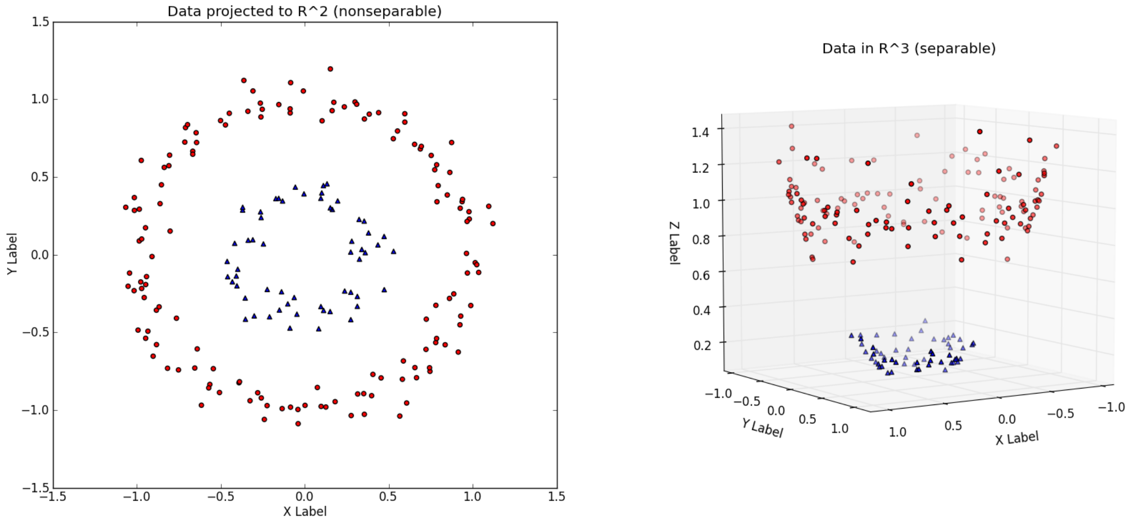


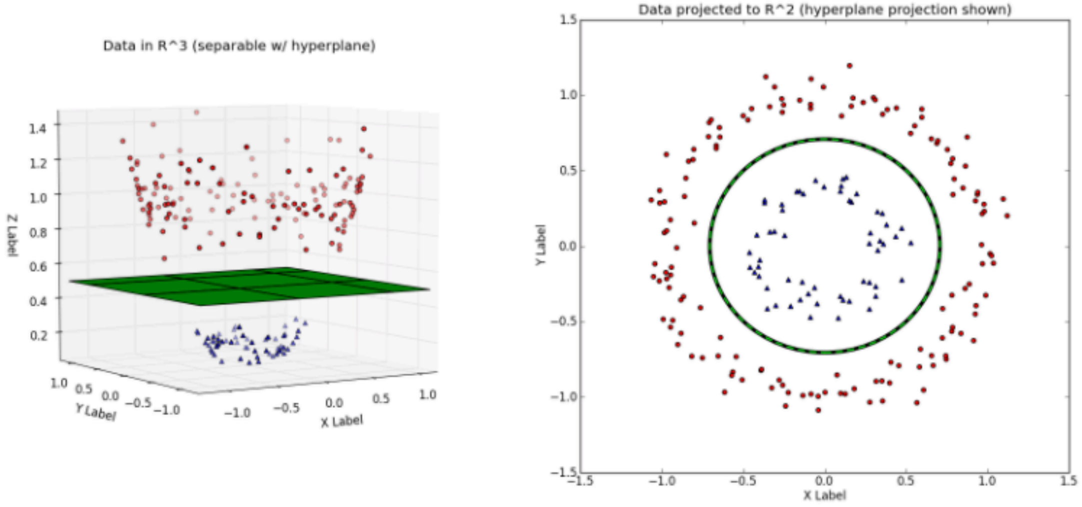
Figure from Ray Mooney

Example: Kernel Map

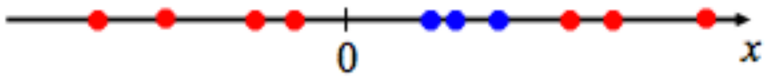


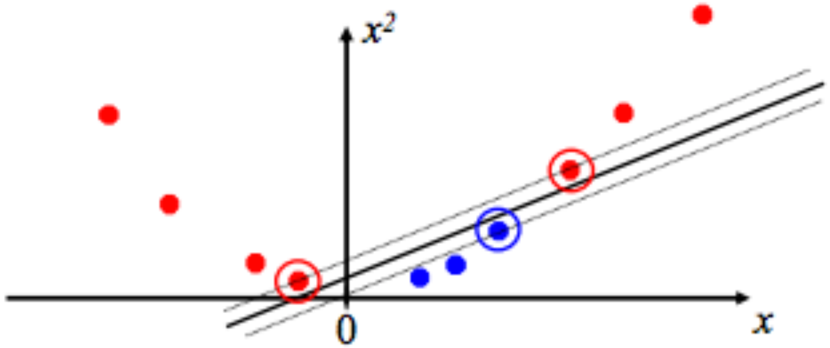
Left) A dataset in $\mathbb{R}^2$, not linearly separable. (Right) The same dataset transformed by the transformation:

$$[x_1, x_2] = [x_1, x_2, {x_1}^2 + {x_2}^2] .$$

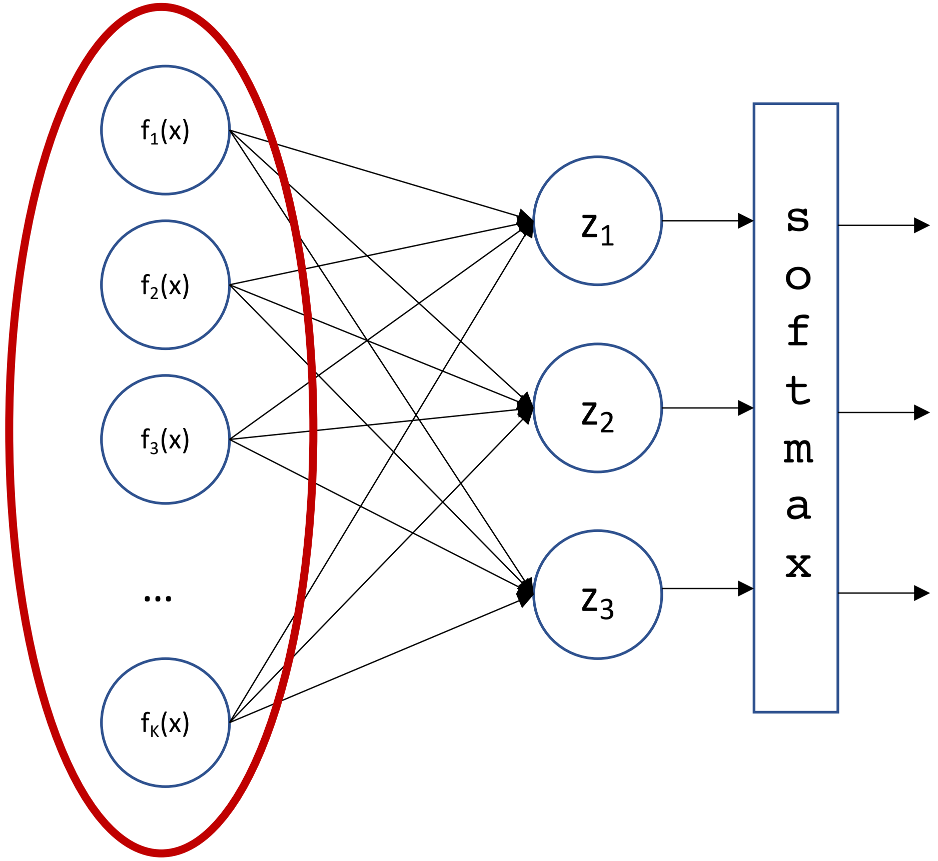


6: (Left) The decision boundary $\vec{w}$ shown to be linear in $\mathbb{R}^3$. (Right) The decision boundary $\vec{w}$, when transformed back to $\mathbb{R}^2$, is nonlinear.





Till now, the features were hand-crafted



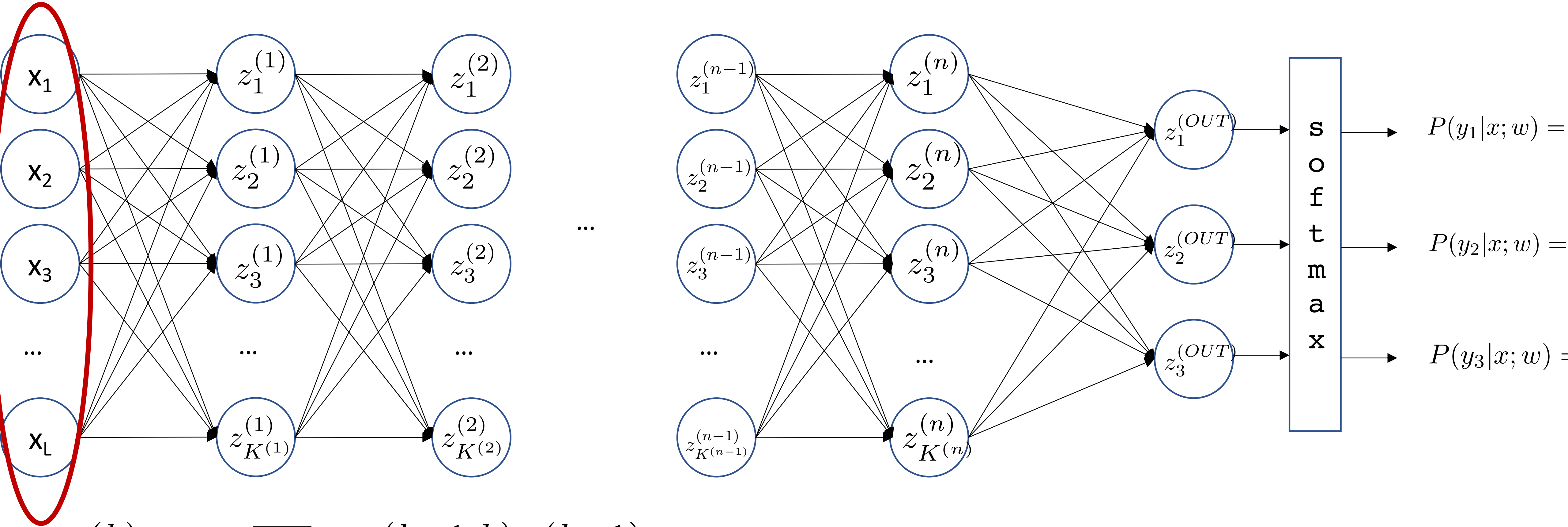
$$P(y_1|x;w) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$P(y_2|x;w) = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$P(y_3|x;w) = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

Still, we are designing these features. Can these be acquired in a data-driven manner? Can the parameters controlling these non-linear functions be learned?

Neural networks: learning the features

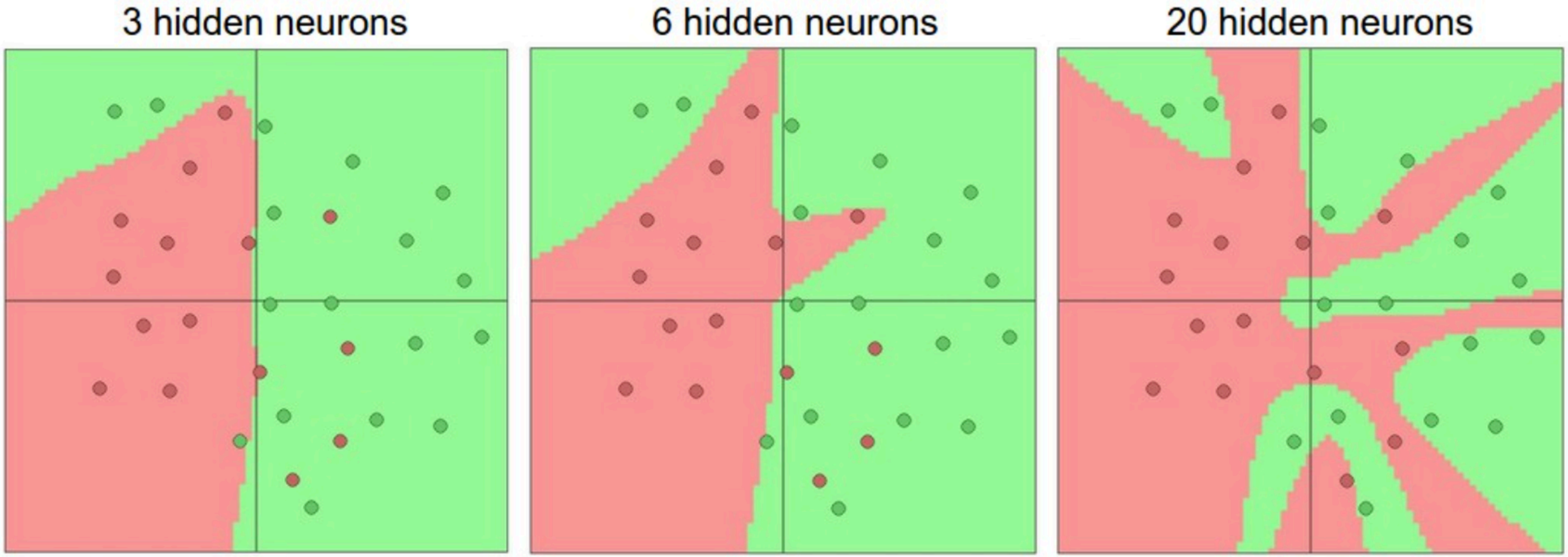


$$z_i^{(k)} = g(\sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)})$$

g = nonlinear activation function

Structure these models by composing many units.
Paradigm is called *deep learning.*

Representation of complex functions

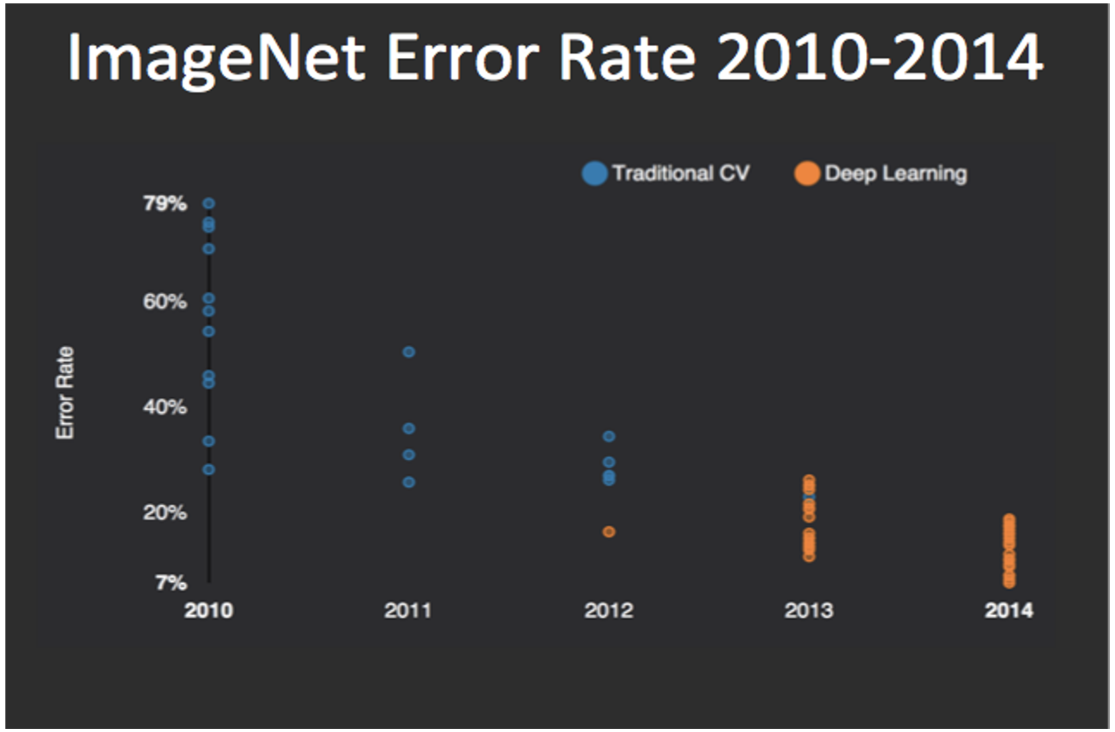


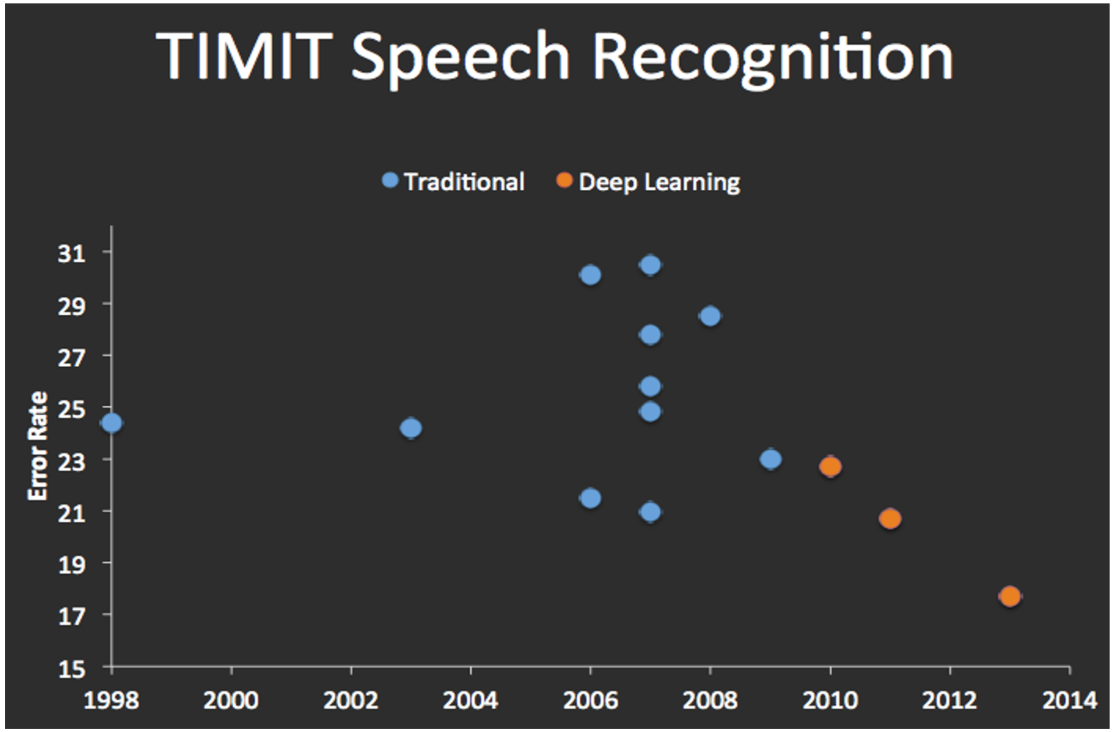
Larger Neural Networks can represent more complicated functions. The data are shown as circles colored by their class, and the decision regions by a trained neural network are shown underneath. You can play with these examples in this ConvNetsJS demo.

Deep Neural Networks

- Last layer
  - Logistic regression
- Several Hidden Layers
  - Computing the features. The features are learned rather than hand-designed.
- Universal function approximation theorem
  - `If` neural net is large enough
  - `Then` neural net can represent any continuous mapping from input to output with arbitrary accuracy
  - Note: overfitting is a challenge.
  - In essence, hyper-parametric function approximation.

Neural Networks: Successes





Learning XOR

XOR is not linearly separable.

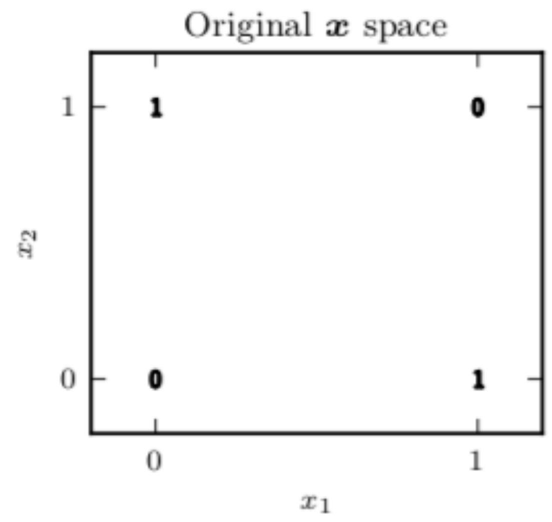


Figure 6.1, left

Rectified Linear Activation

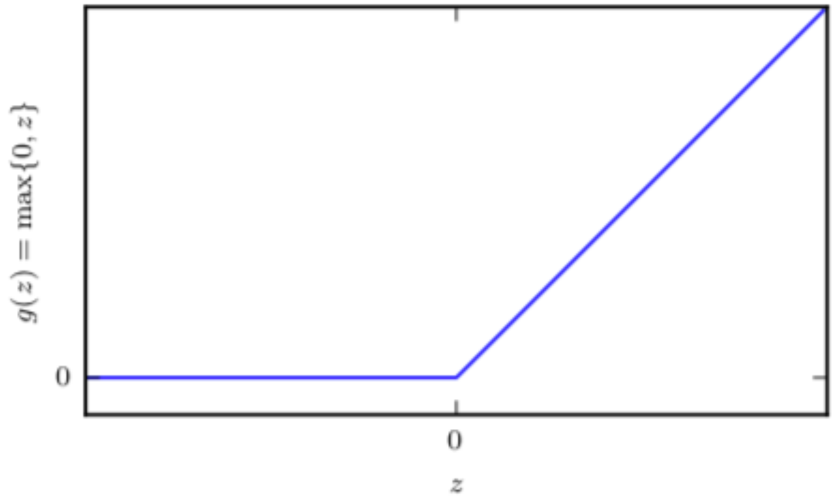


Figure 6.3

Network Diagram

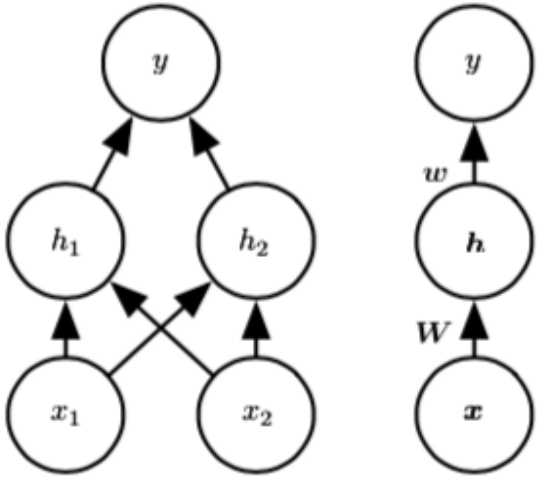


Figure 6.2

More compact representation

Example from Ch 6, DL Book

Learning XOR

Network Diagram

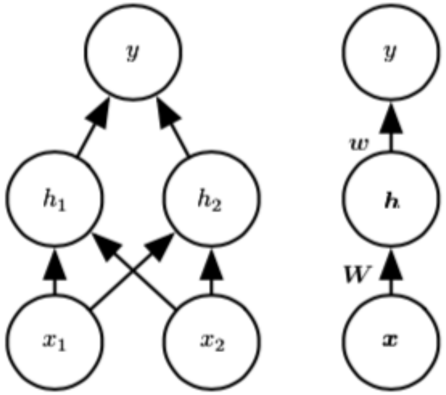


Figure 6.2

Model

$$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^\top \max\{0, \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}\} + b.$$

$$\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix},$$

$$\boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix},$$

$$\boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix},$$

XOR is separable in the transformed space

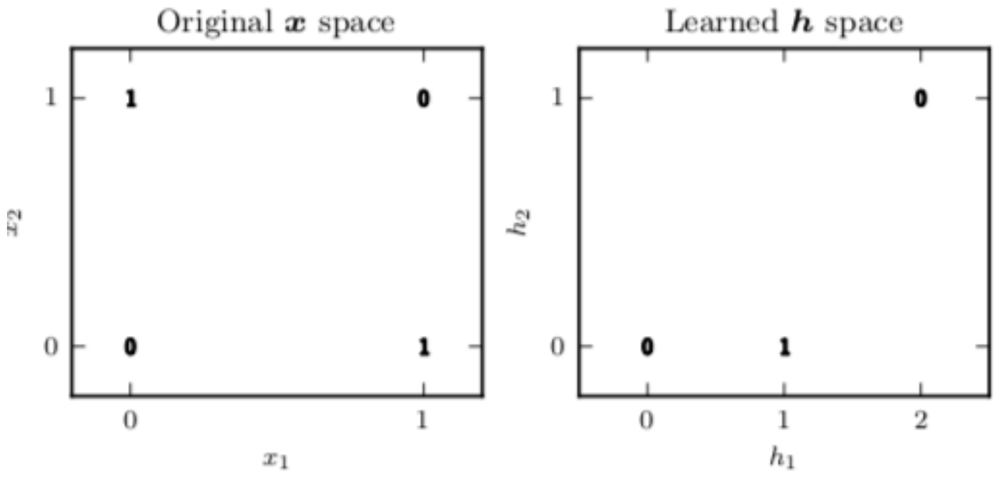


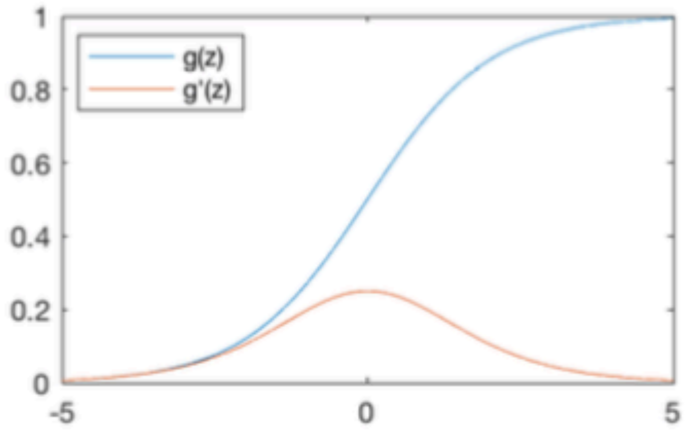
Figure 6.1

Takeaway: Applying ReLU to the output of a linear transformation yields a non-linear transformation. The problem can be solved in the transformed space.

Example from Ch 6, DL Book

Common Activation Functions

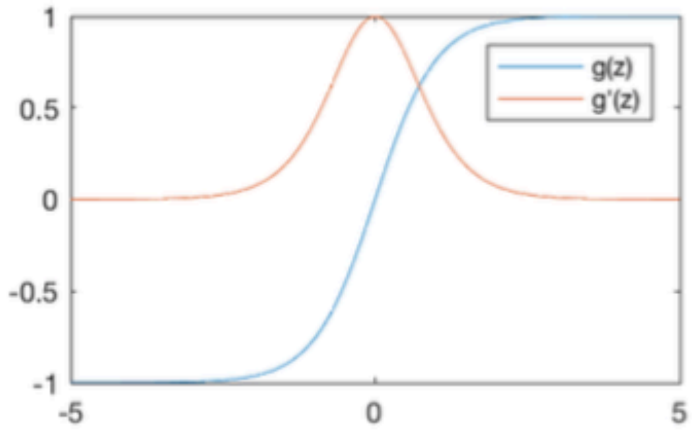
Sigmoid Function



$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

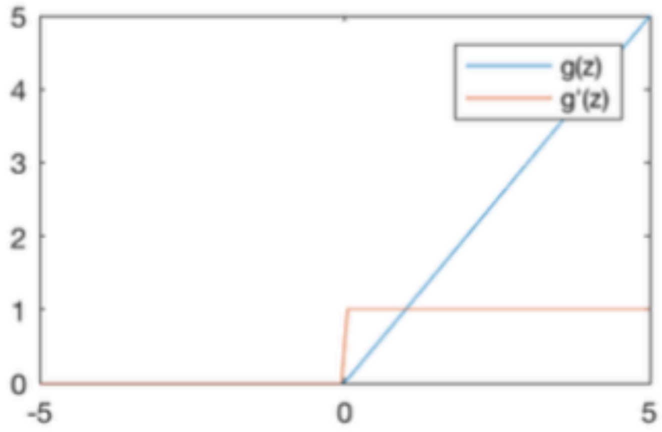
Hyperbolic Tangent



$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

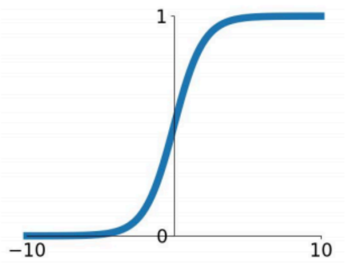
$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

[source: MIT 6.S191 introtodeeplearning.com]

Activation Functions

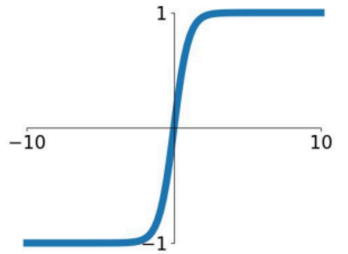
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



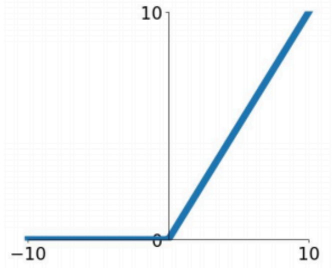
tanh

$\tanh(x)$



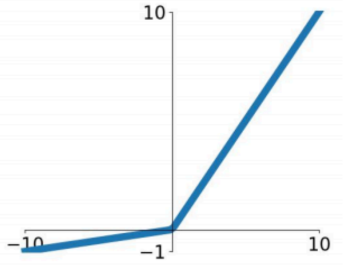
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

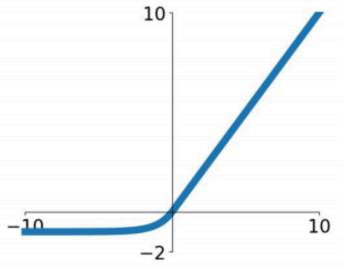


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

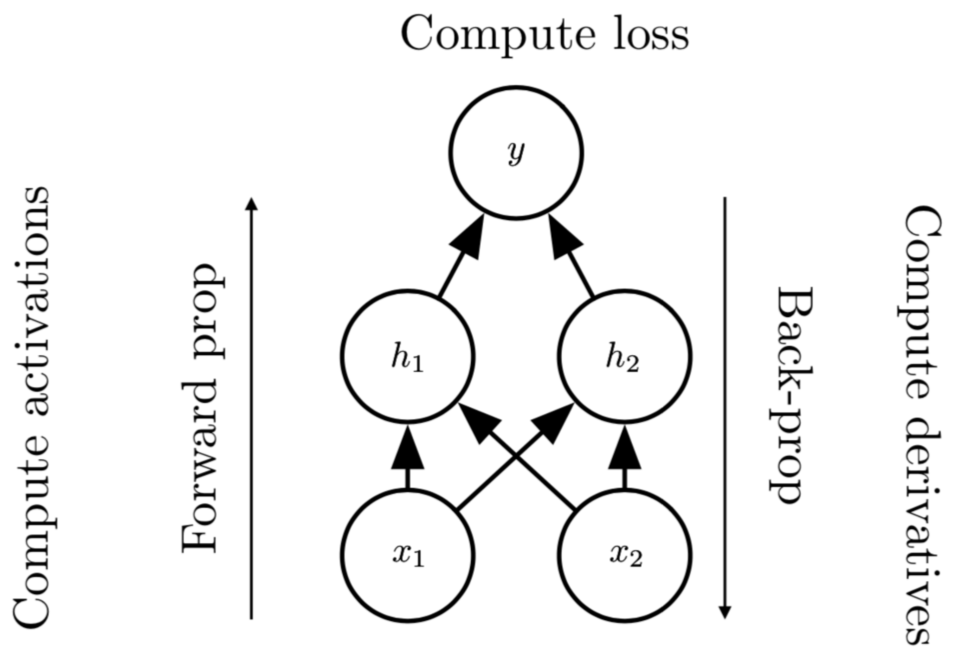
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Backpropagation and Computation Graphs

- Backpropagation
  - In a NN, need a way to optimize the output loss with respect to the inputs.
  - Apply the chain rule to obtain the gradient.



$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}.$$

$$\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^{\top} \nabla_{\boldsymbol{y}} z,$$

Material from Ch 6, DL Book

Backpropagation and Computation Graphs

- Computation Graphs
  - A way to organize the computation in a neural network.
  - Also enables identification and caching of repeated sub-expressions.

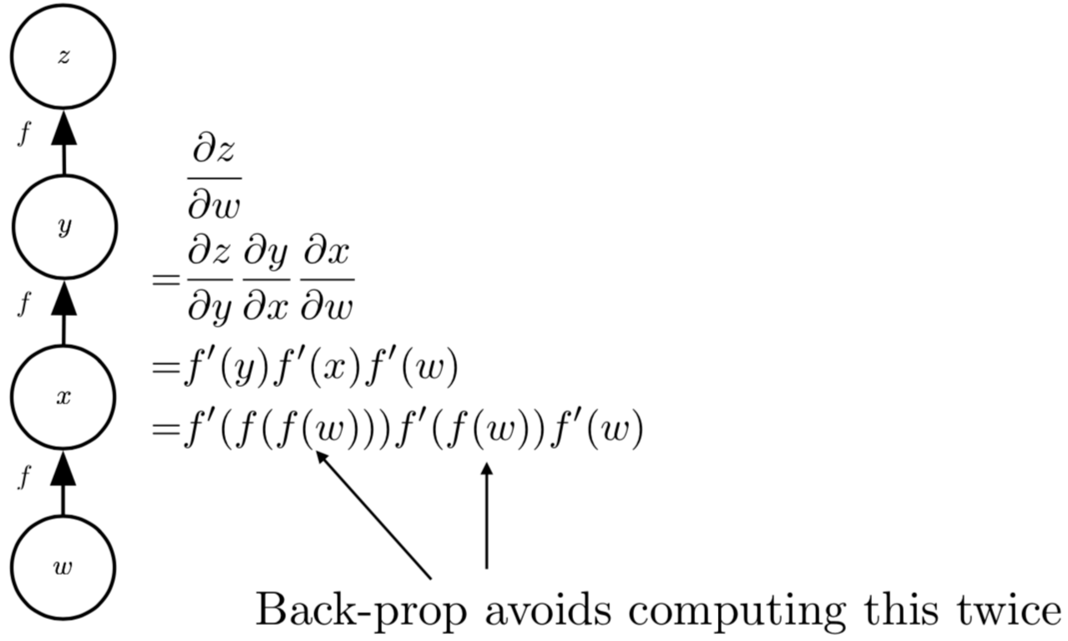


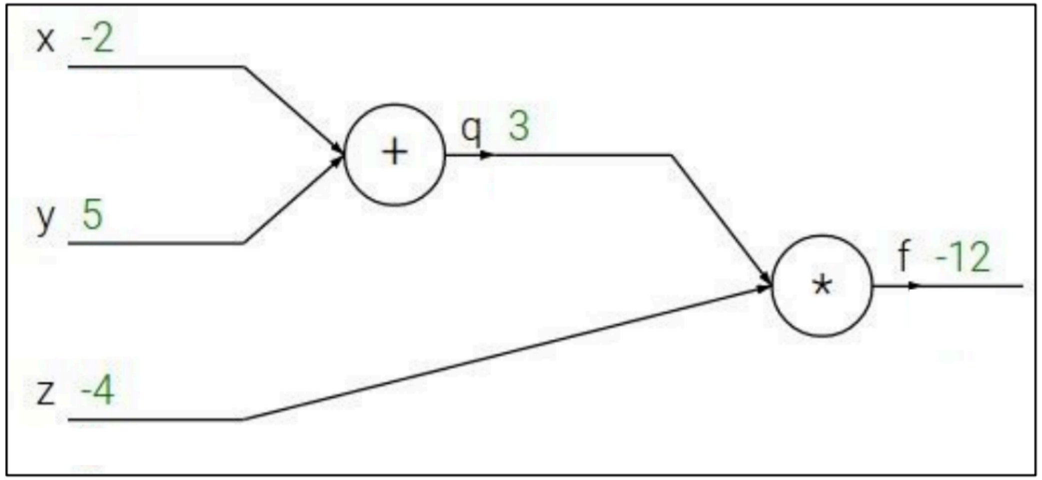
Figure 6.9

Backpropagation: Toy Example

Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



Example from: http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture04.pdf

Backpropagation: Toy Example

Backpropagation: a simple example

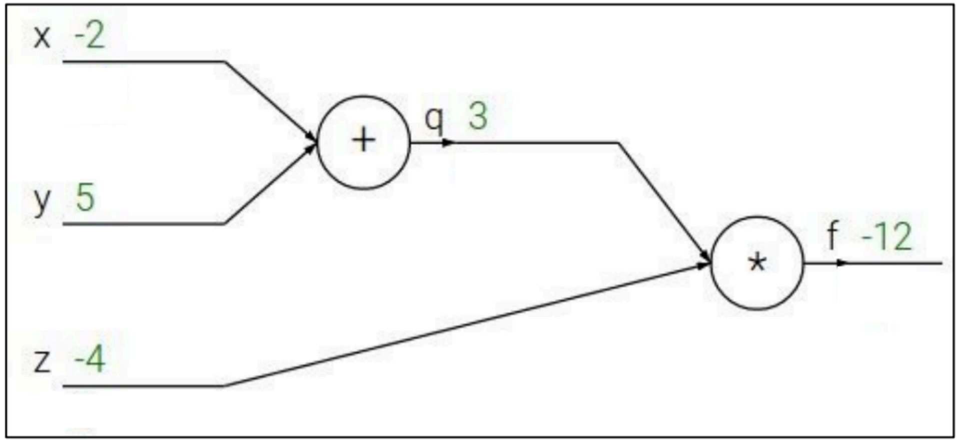
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: Toy Example

Backpropagation: a simple example

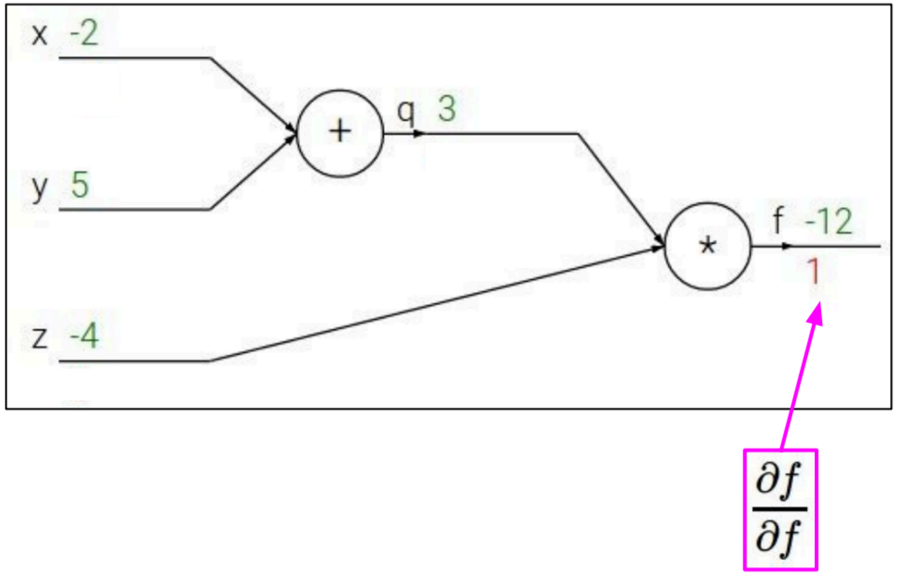
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: Toy Example

Backpropagation: a simple example

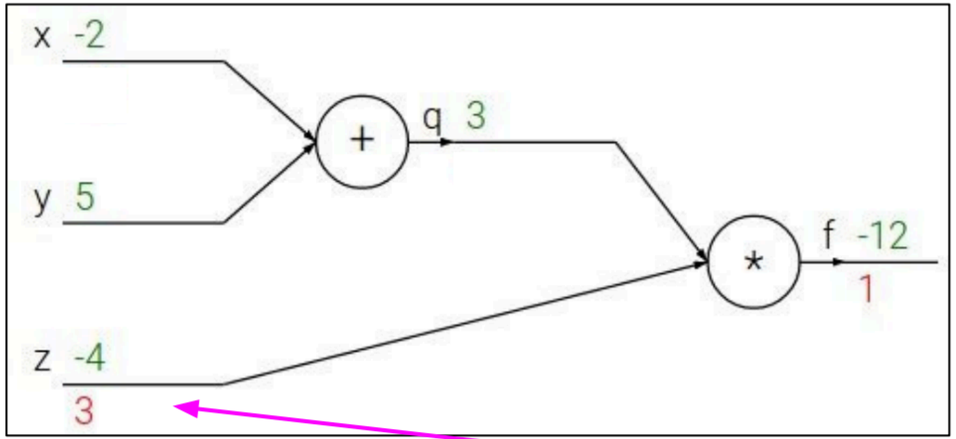
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$\frac{\partial f}{\partial z}$

Backpropagation: Toy Example

Backpropagation: a simple example

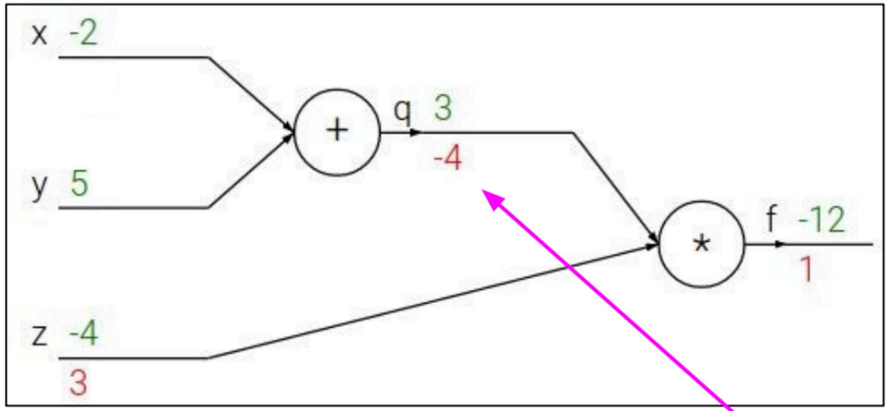
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial q}$$

Backpropagation: Toy Example

Backpropagation: a simple example

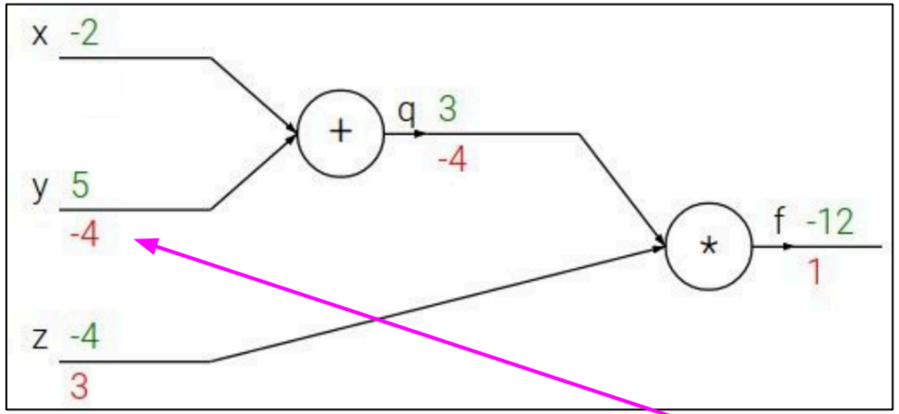
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial y}$$

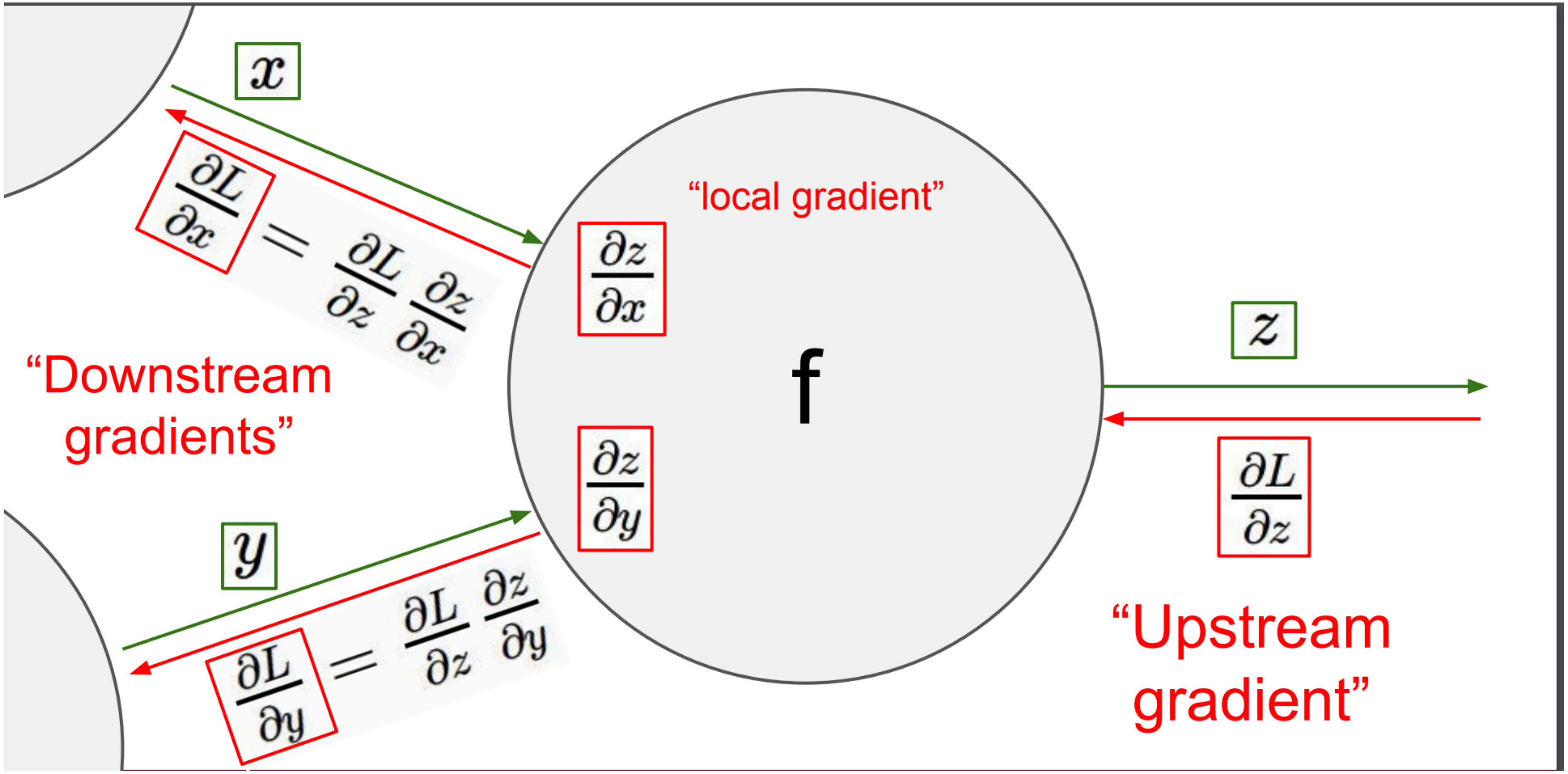
Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

Upstream gradient

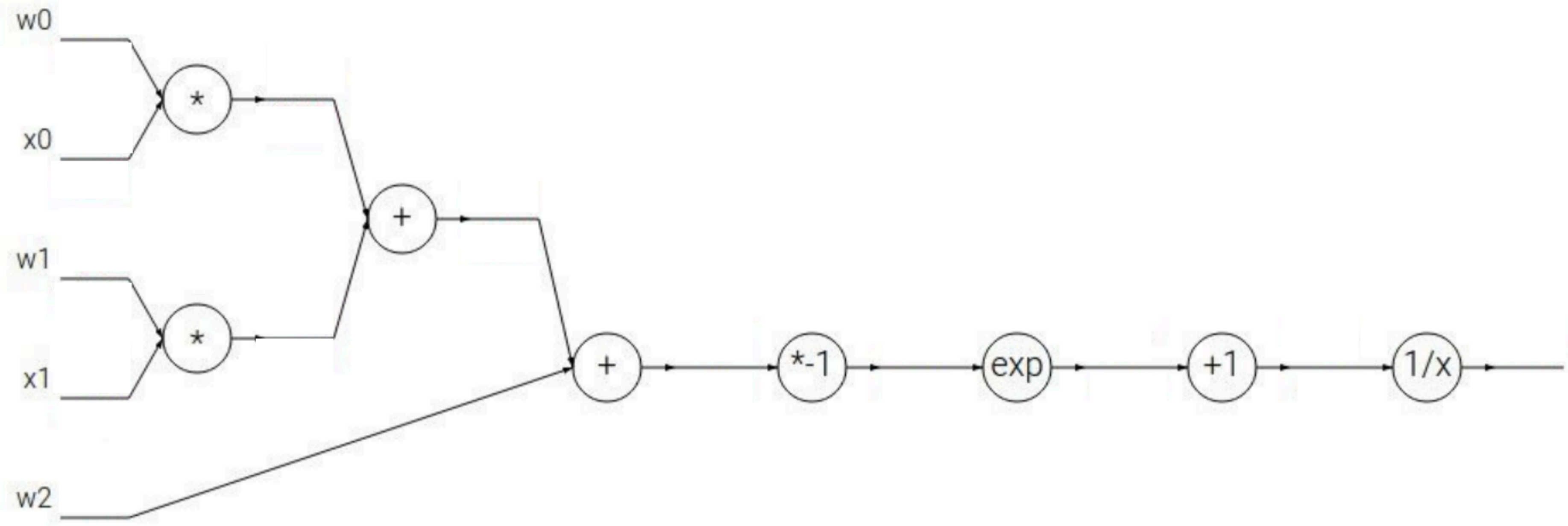
Local gradient

Backpropagation



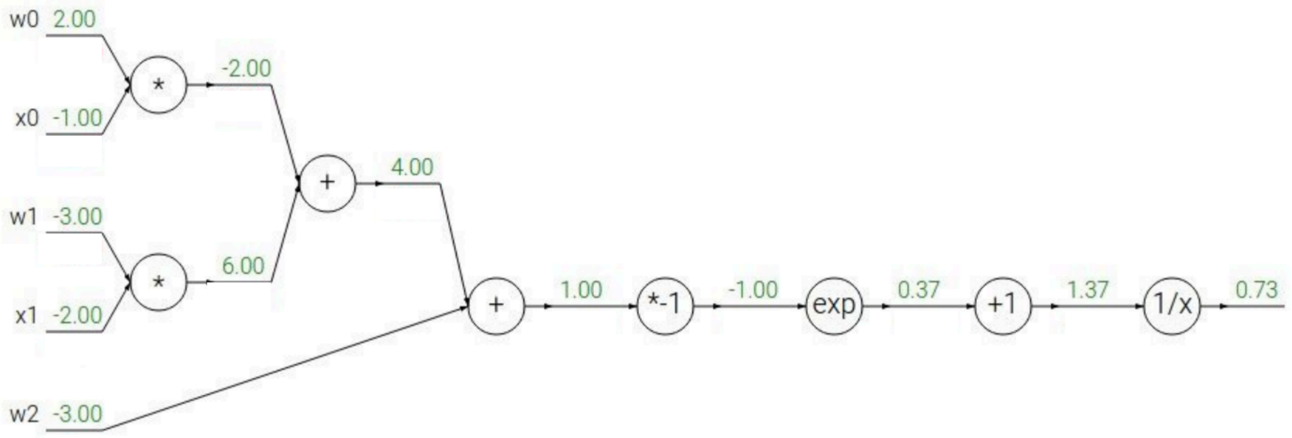
Backpropagation: Example

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



Backpropagation: Example

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$

$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$

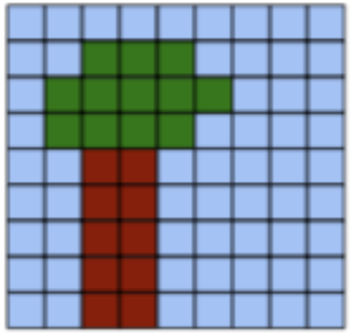
$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$

Other Links

- Visualization
  - http://playground.tensorflow.org
- Libraries
  - https://pytorch.org/
  - https://www.tensorflow.org/
  - https://pypi.org/project/Theano/
  - http://pyro.ai/

Locality and Translational Invariance

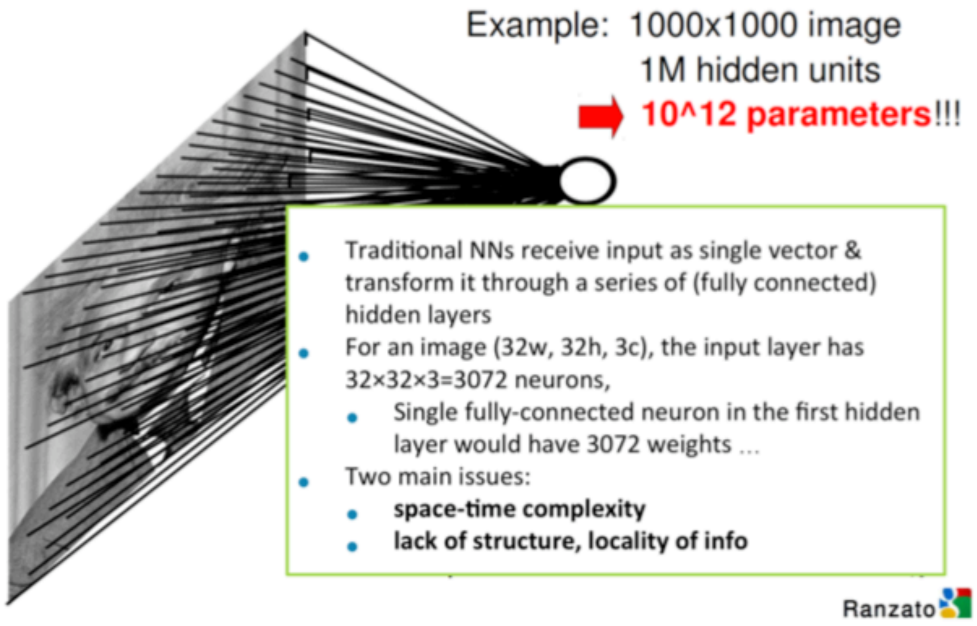


A digital image is a 2D **grid of pixels**.

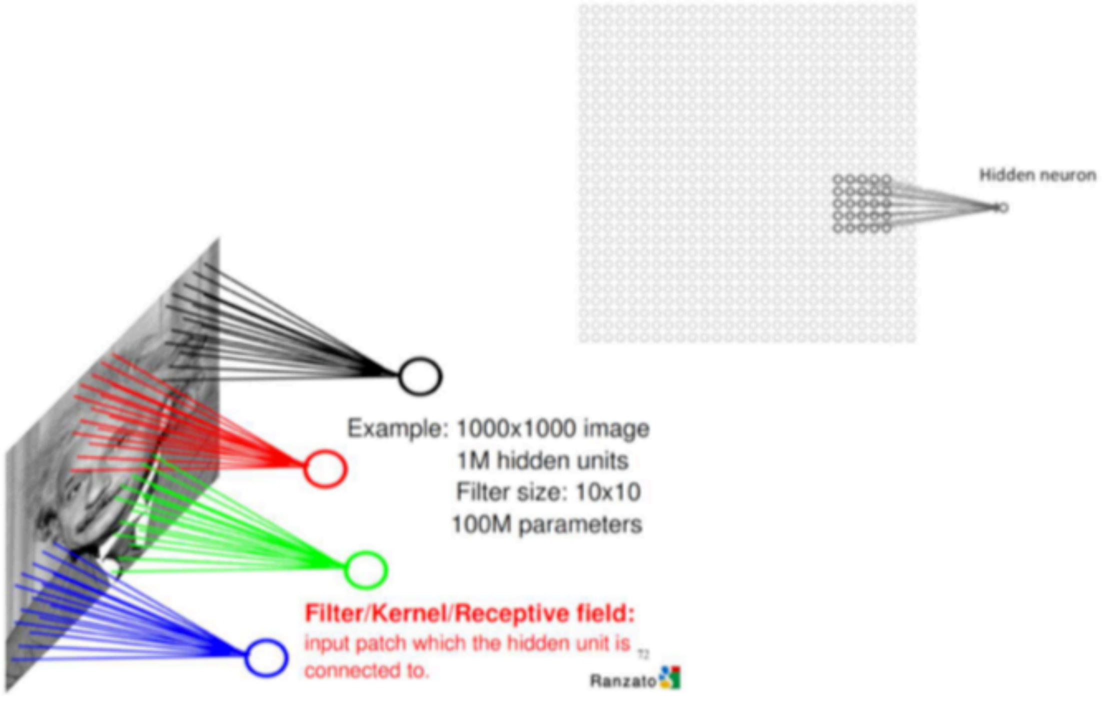
A neural network expects a **vector of numbers** as input.

Locality of Information: Receptive Fields

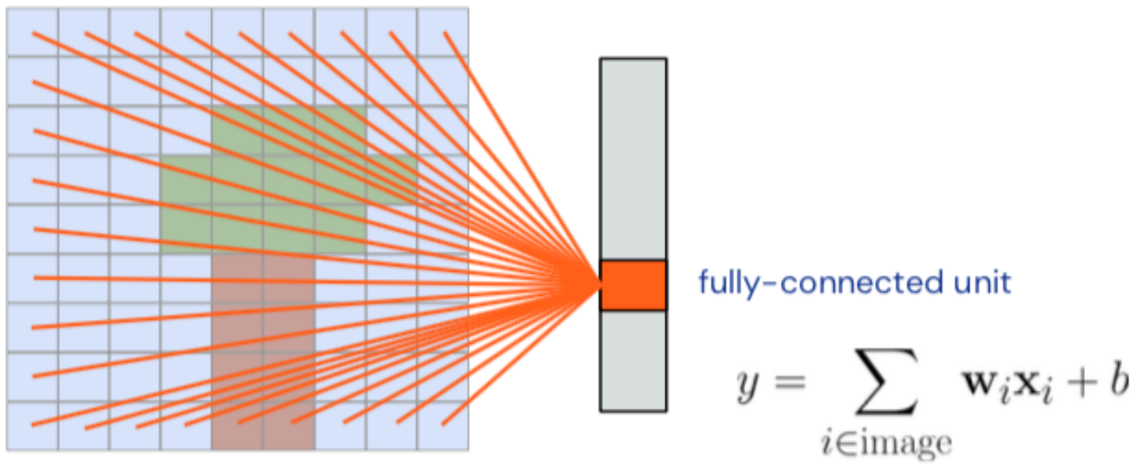
Fully connected network.

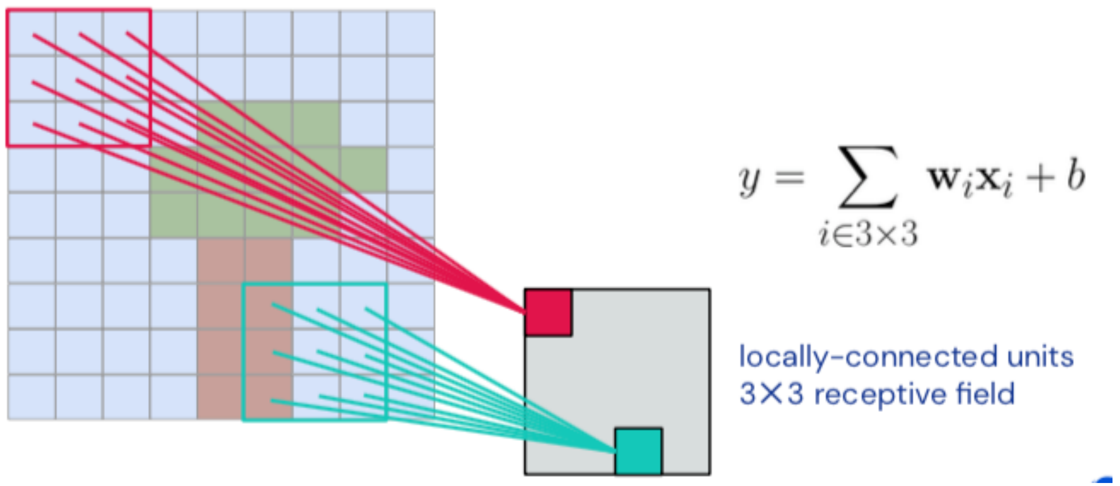


Convolutional NN



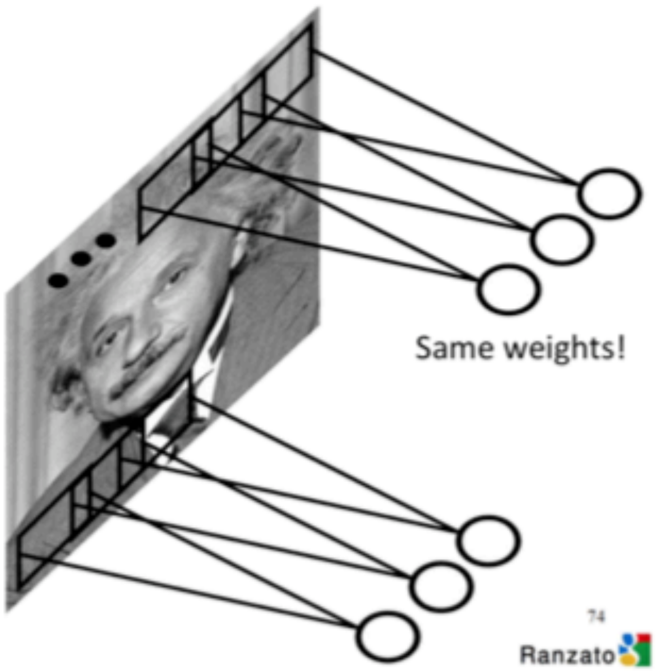
From Globally to Locally Connected



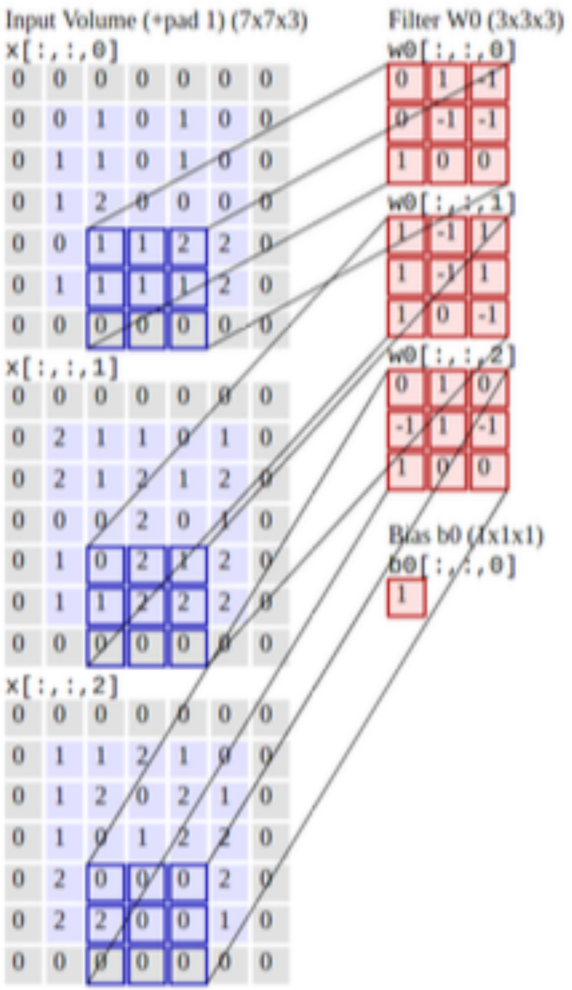


Convolutional NN

Feature Maps



- The map from the input layer to the hidden layer is therefore a feature map: all nodes detect the same feature in different parts
- The map is defined by the shared weights and bias
- The shared map is the result of the application of a convolutional filter (defined by weights and bias), also known as convolution with learned kernels

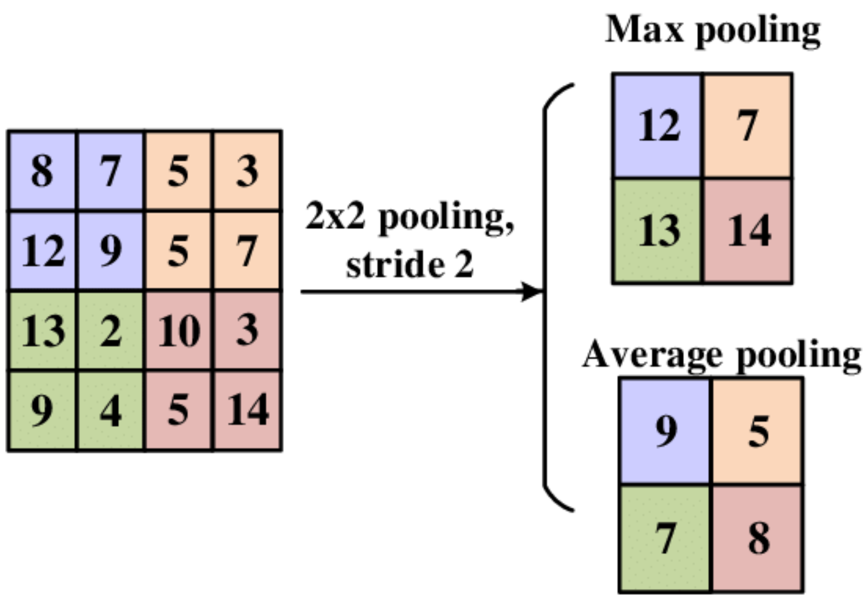


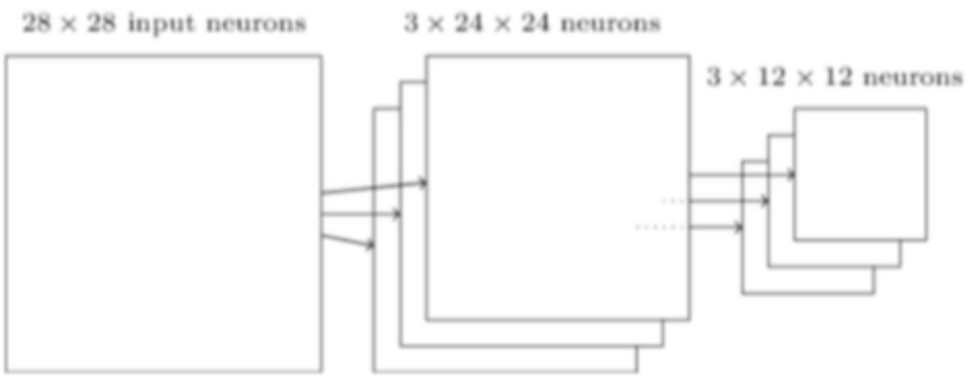
Pooling layers

Pooling layers are usually used immediately after convolutional layers.

Pooling layers simplify / subsample / compress the information in the output from convolutional layer

A pooling layer takes each feature map output from the convolutional layer and prepares a condensed feature map





Convolution NN

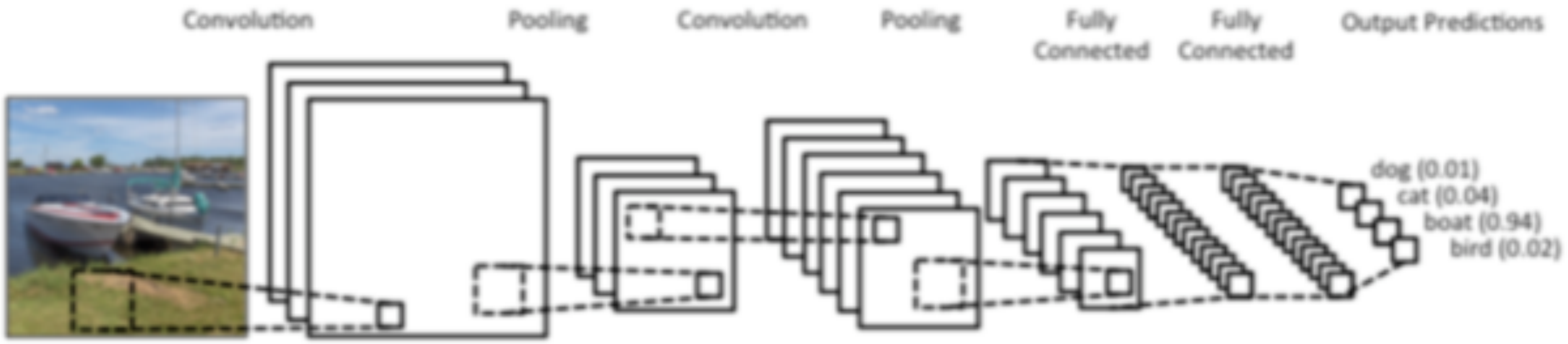
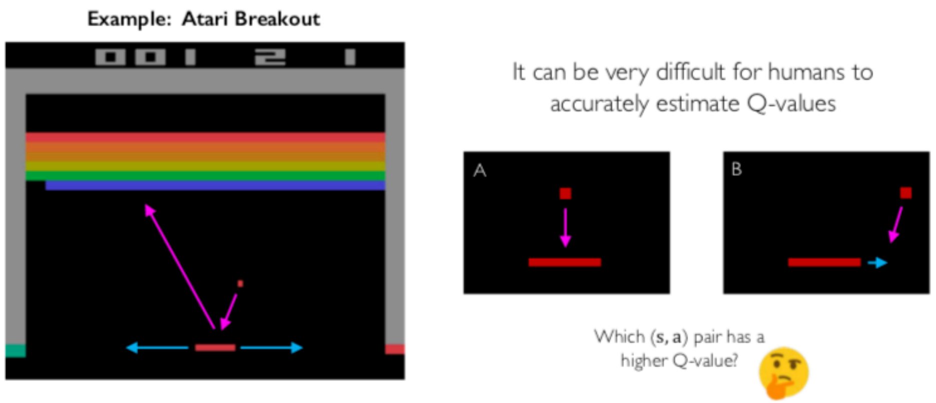
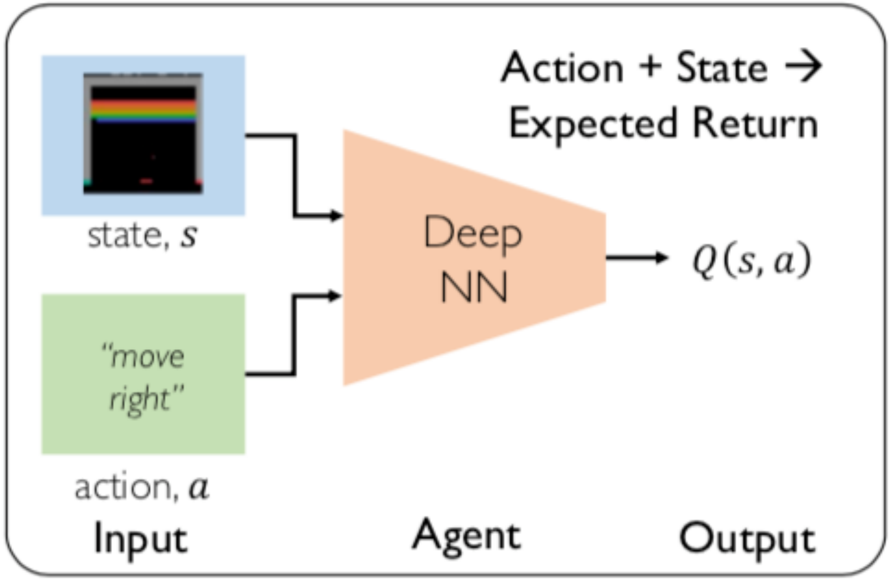


Image: http://d3kbpzbmcynnmx.cloudfront.net/wp-content/uploads/2015/11/Screen-Shot-2015-11-07-at-7.26.20-AM.png

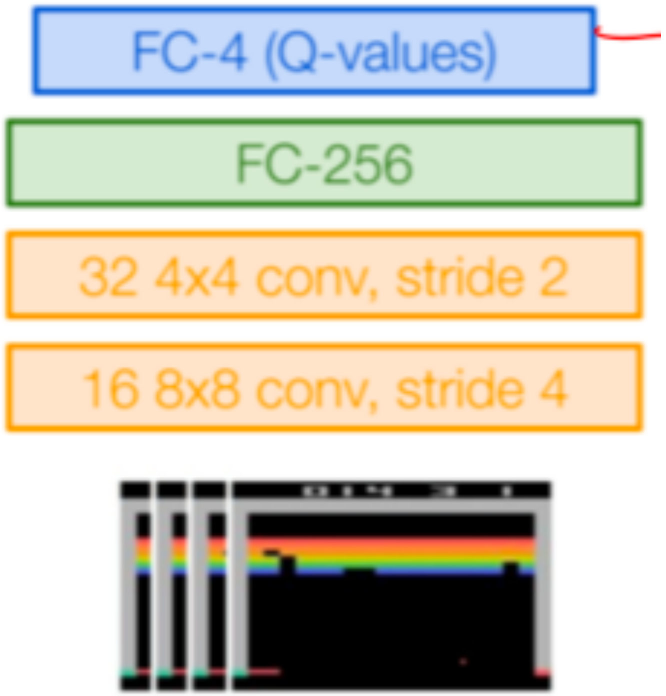
- Consider local structure and common extraction of features
- Not fully connected. Locality of processing
- Weight sharing for parameter reduction
- Learn the parameters of multiple convolutional filter banks
- Compress to extract salient features & favor generalization

Application



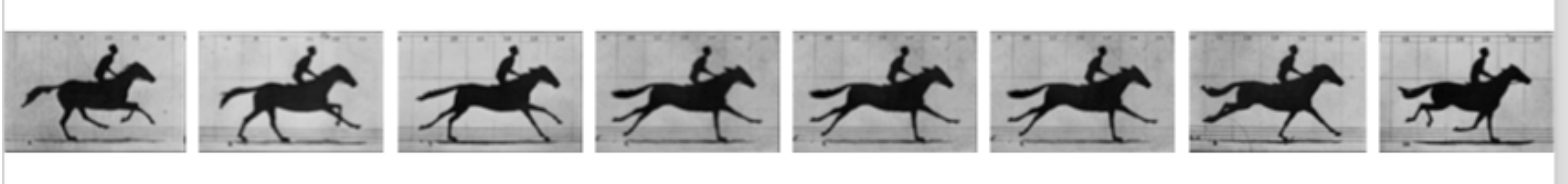


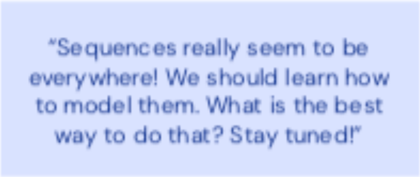
Output: Q(s, left), Q(s, right), …..



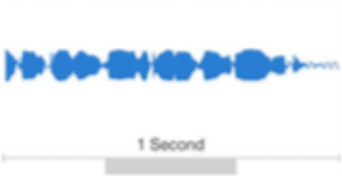
Current state s_t: 84*84*4 stack of last four frames. After RGB-> grayscale conversion, downsampling and cropping.

Sequences



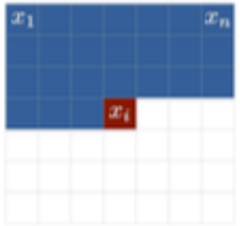


Words, letters



Speech

Videos



Images

Programs

Decision making

Collection of elements where elements can be repeated, order matters and can be of variable or infinite length.

Sequences

	Supervised learning	Sequence modelling
Data	$\{x, y\}_i$	$\{x\}_i$
Model	$y \approx f_\theta(x)$	$p(x) \approx f_\theta(x)$
Loss	$\mathcal{L}(\theta) = \sum_{i=1}^{N} l(f_\theta(x_i), y_i)$	$\mathcal{L}(\theta) = \sum_{i=1}^{N} \log p(f_\theta(x_i))$
Optimisation	$\theta^* = \arg\min_\theta \mathcal{L}(\theta)$	$\theta^* = \arg\max_\theta \mathcal{L}(\theta)$

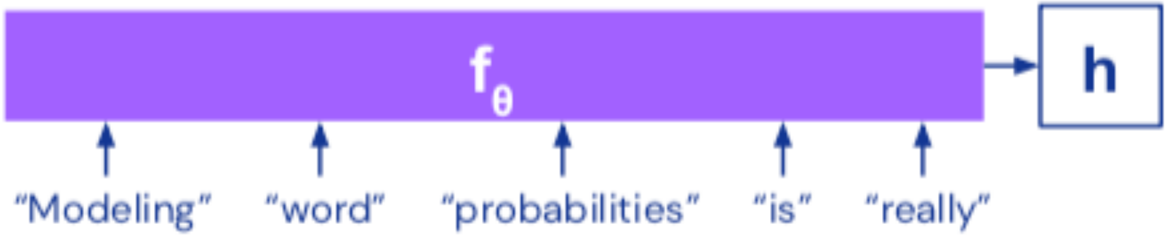
Modeling the conditional distribution

The chain rule
Computing the joint p(**x**) from conditionals

$$p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t | x_1, ..., x_{t-1})$$

Modeling	$p(x_1)$
Modeling **word**	$p(x_2 \| x_1)$
Modeling word **probabilities**	$p(x_3 \| x_2, x_1)$
Modeling word probabilities **is**	$p(x_4 \| x_3, x_2, x_1)$
Modeling word probabilities is **really**	$p(x_5 \| x_4, x_3, x_2, x_1)$
Modeling word probabilities is really **difficult**	$p(x_6 \| x_5, x_4, x_3, x_2, x_1)$

Vectorizing the conditional likelihood

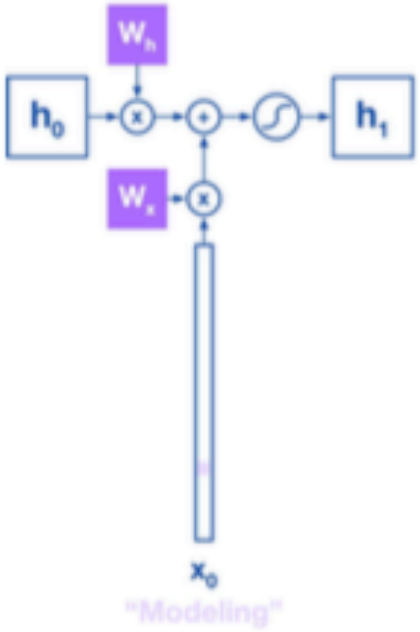


Desirable properties for f_θ:

- Order matters
- Variable length
- Learnable (differentiable)
- Individual changes can have large effects (non-linear/deep)

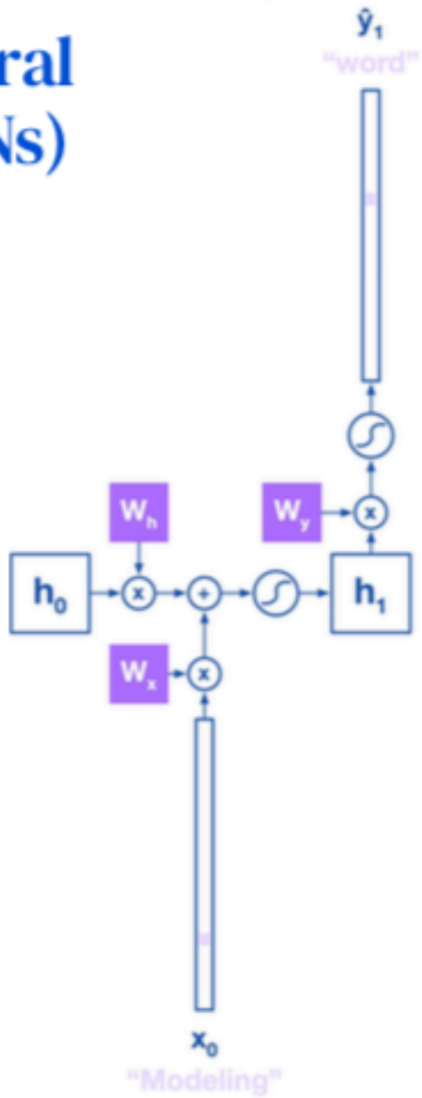
Recurrent Neural Networks (RNNs)

Persistent state variable **h** stores information from the context observed so far.



$$\mathbf{h}_t = \tanh(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t)$$

Recurrent Neural Networks (RNNs)

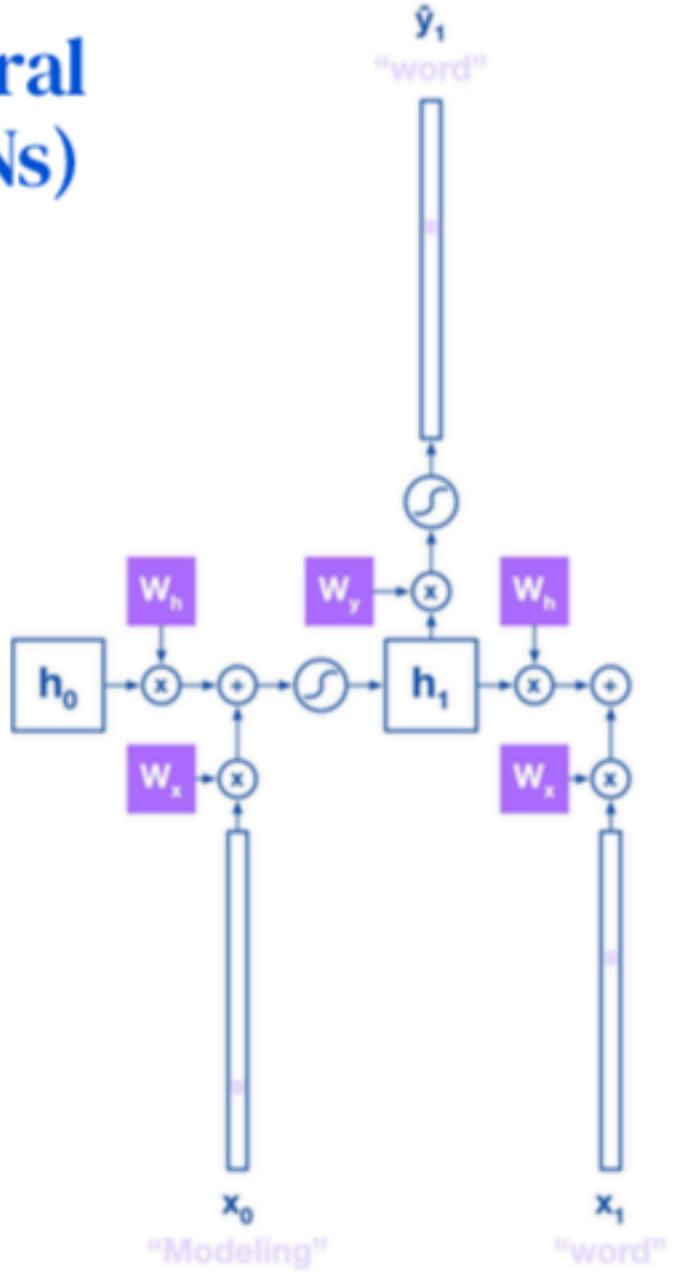


RNNs predict the target **y** (the next word) from the state **h.**

$$p(\mathbf{y_{t+1}}) = softmax(\mathbf{W}_y \mathbf{h}_t)$$

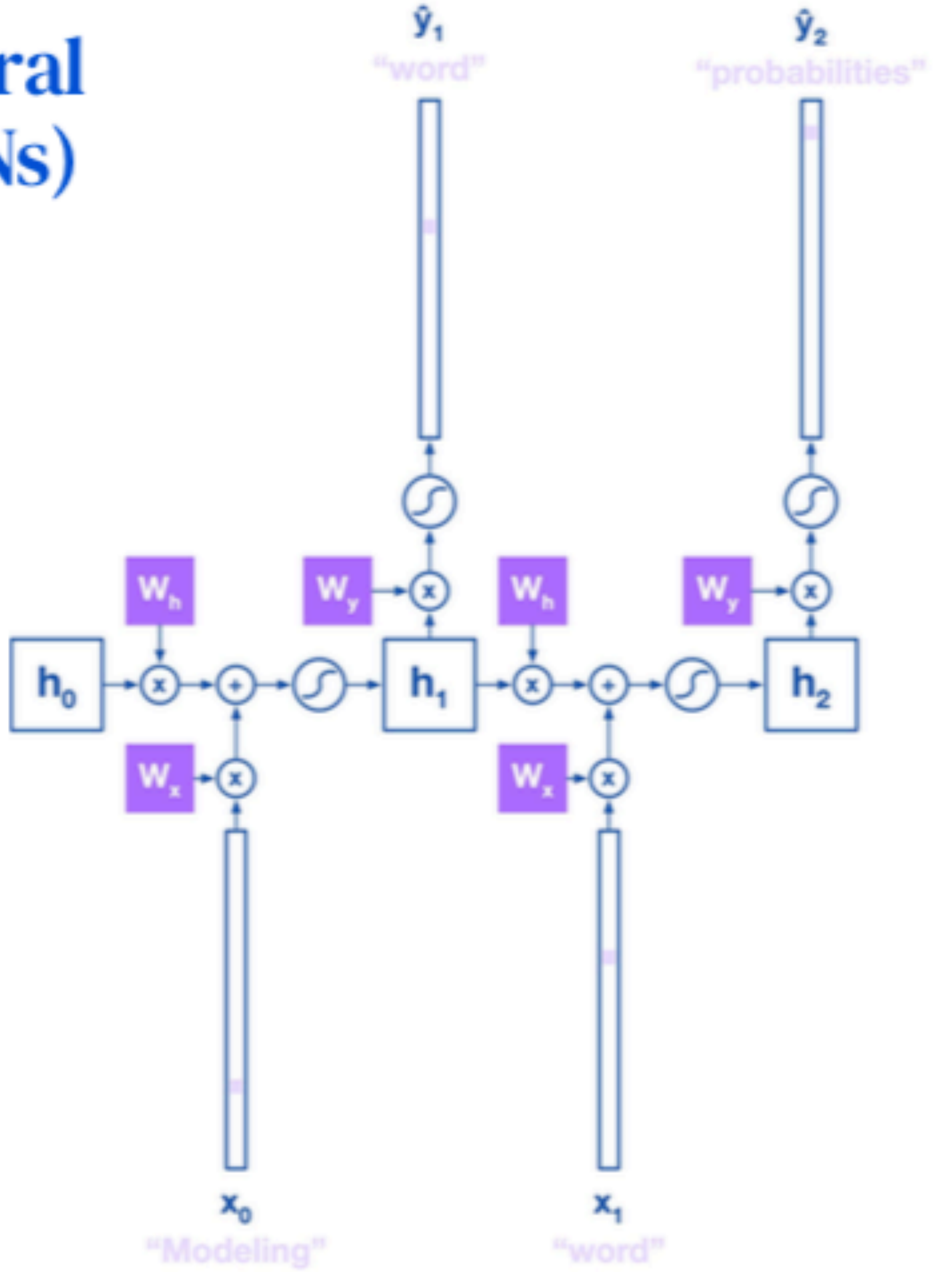
Softmax ensures we obtain a distribution over all possible words.

Recurrent Neural Networks (RNNs)



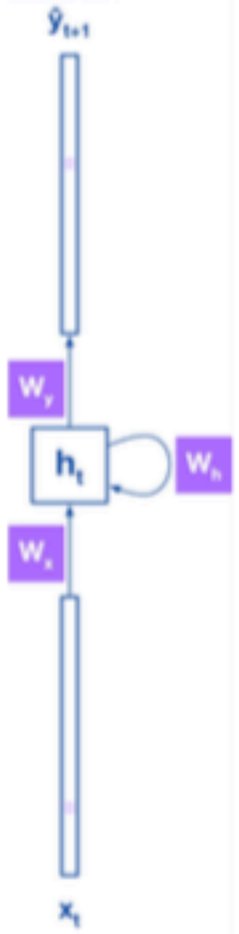
Input next word in sentence x_1

Recurrent Neural Networks (RNNs)

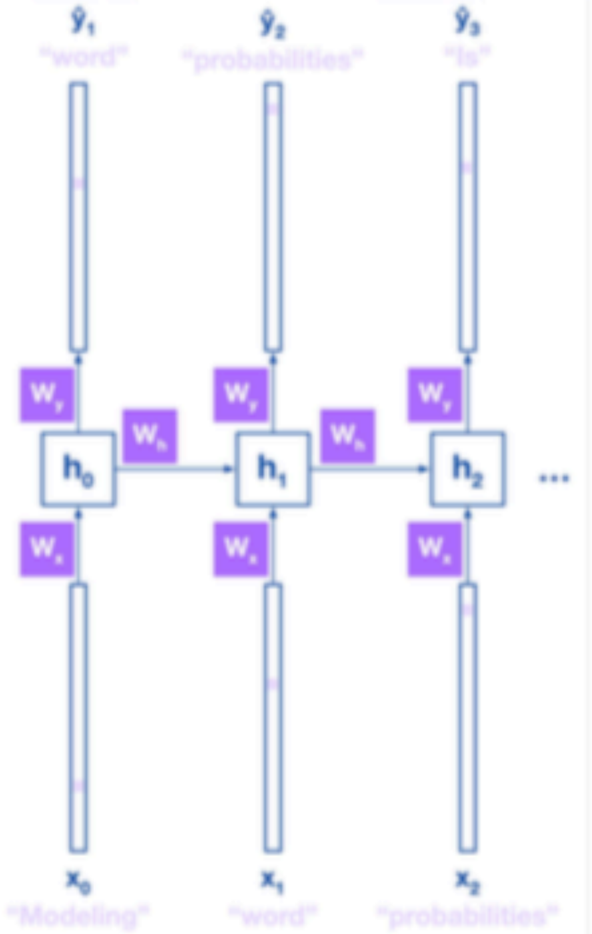


Recurrent Neural Networks (RNNs)

Weights are shared over time steps



RNN



RNN rolled out over time

Loss: Cross Entropy

Next word prediction is essentially a classification task where the number of classes is the size of the vocabulary.

As such we use the cross-entropy loss:

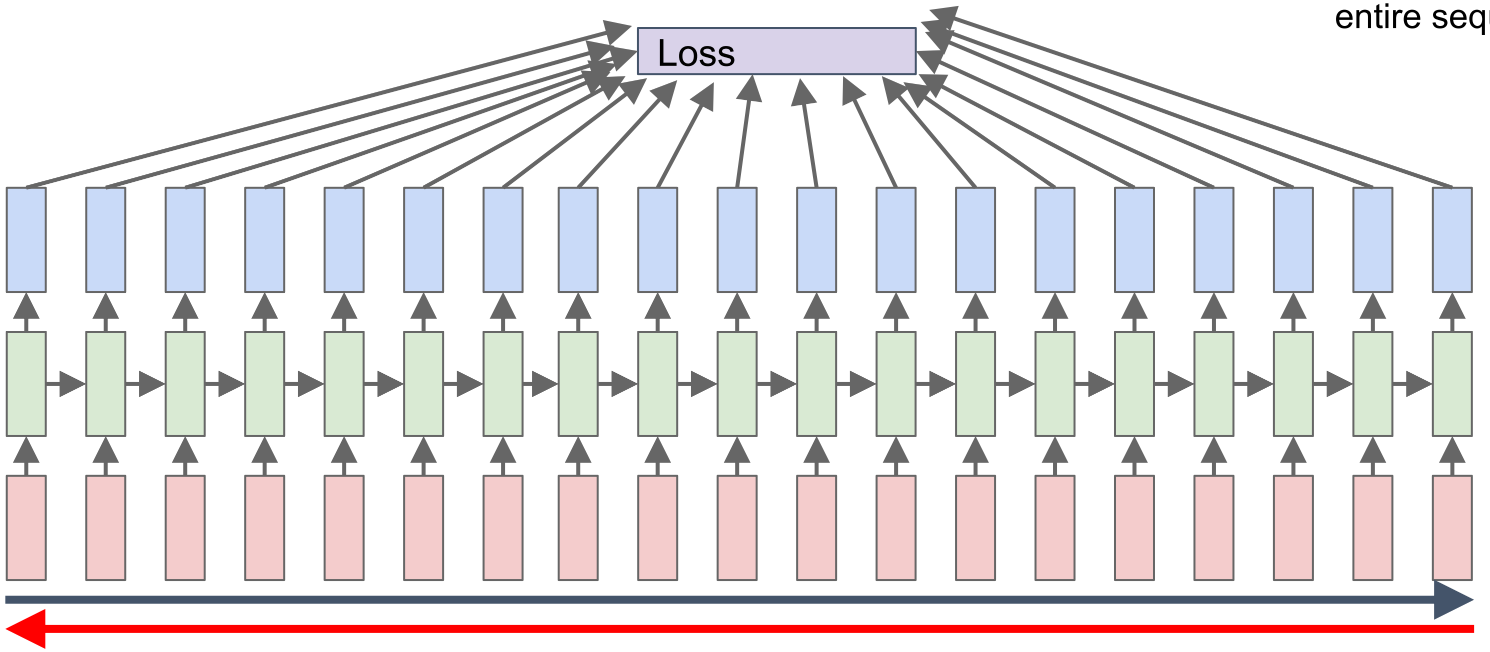
For one word: $$\mathcal{L}_\theta(\mathbf{y}, \hat{\mathbf{y}})_t = -\mathbf{y}_t \log \hat{\mathbf{y}}_t$$

For the sentence: $$\mathcal{L}_\theta(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{t=1}^{T} \mathbf{y}_t \log \hat{\mathbf{y}}_t$$

With parameters $\theta = \{\mathbf{W}_y, \mathbf{W}_x, \mathbf{W}_h\}$

Backprop through Time

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient

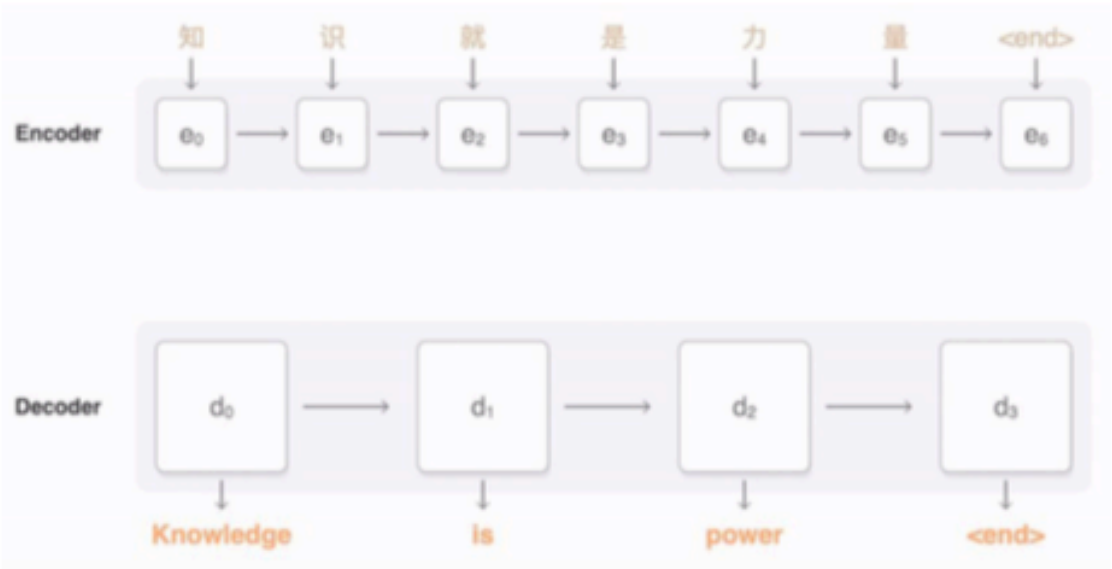


RNNs can have long or short dependencies. When there are long dependencies, gradients have trouble back-propagating through.

Other models such as LSTMs and beyond address that problem.

Applications

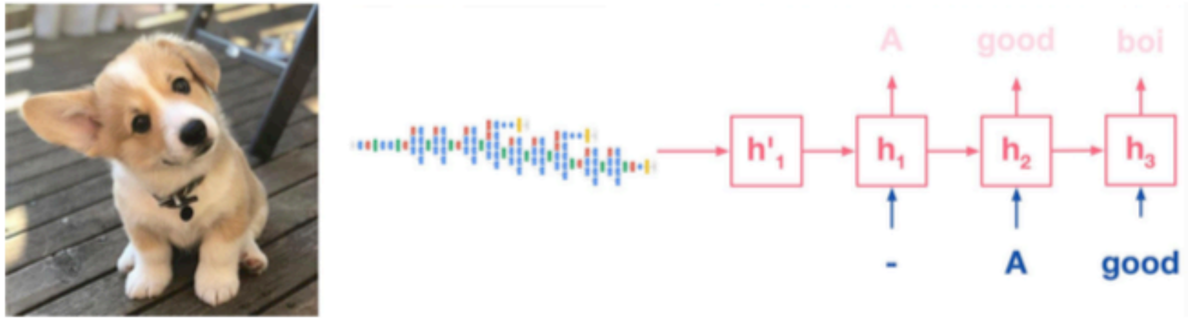
Google Neural Machine Translation

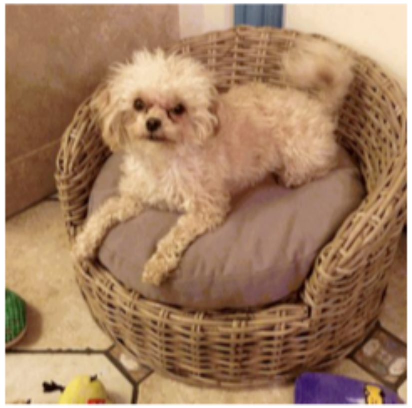


Wu et al, 2016
(Kalchbrenner et al, 2013; Sutskever et al, 2014; Cho et al, 2014; Bhadanau et al, 2014; ...)

Applications

$p(\text{language}_1 \mid \text{language}_2) \rightarrow p(\text{language}_1 \mid \text{image})$



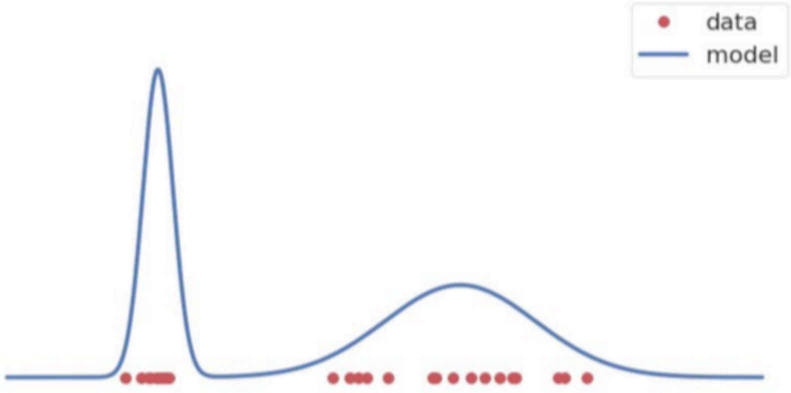


Human: A brown dog laying in a red wicker bed.

Best Model: A small dog is sitting on a chair.

Initial Model: A large brown dog laying on top of a couch.

Generative Models



Goal of generative modeling is to kearn a model of the true (unknown) underlying data distribution from samples.

Goodfellow, et al. **Generative adversarial networks.** NIPS (2014)

Denton, et al. **Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks.** NIPS (2015)

Radford et al. **Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks** ICLR (2015)

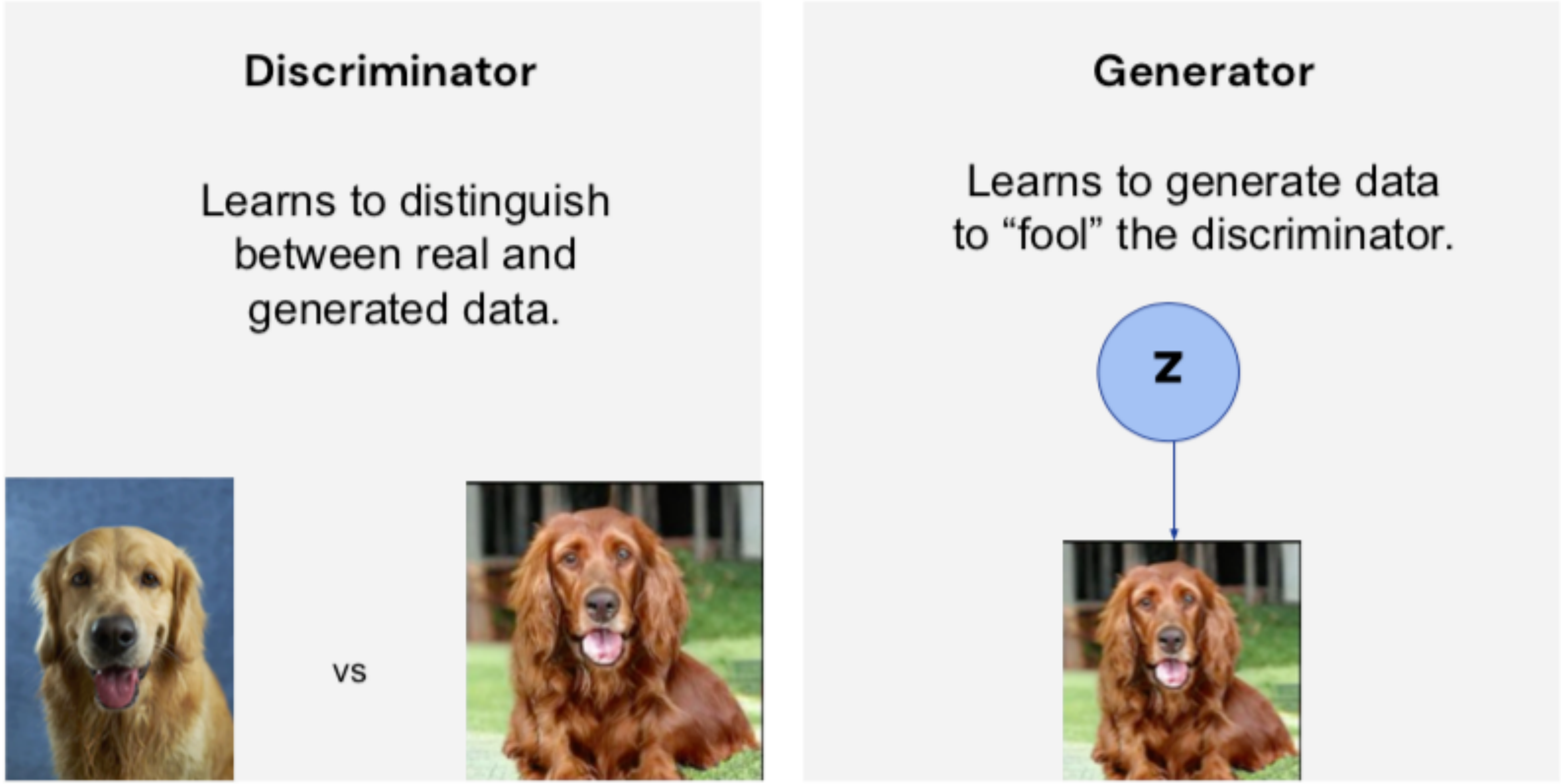
Miyato et al. **Spectral normalization for Generative Adversarial Networks** ICLR (2018)

Karras et al. **Large Scale GAN Training for High Fidelity Natural Image Synthesis** ICLR (2018)

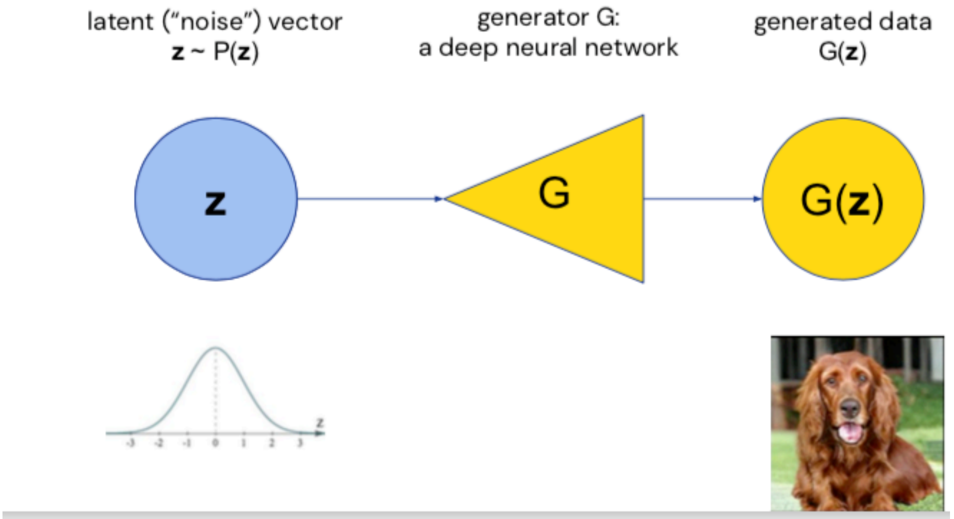
Brock et al. **Large Scale GAN Training for High Fidelity Natural Image Synthesis** ICLR (2019)

Karras et al. **A Style-Based Generator Architecture for Generative Adversarial Networks** CVPR (2019)

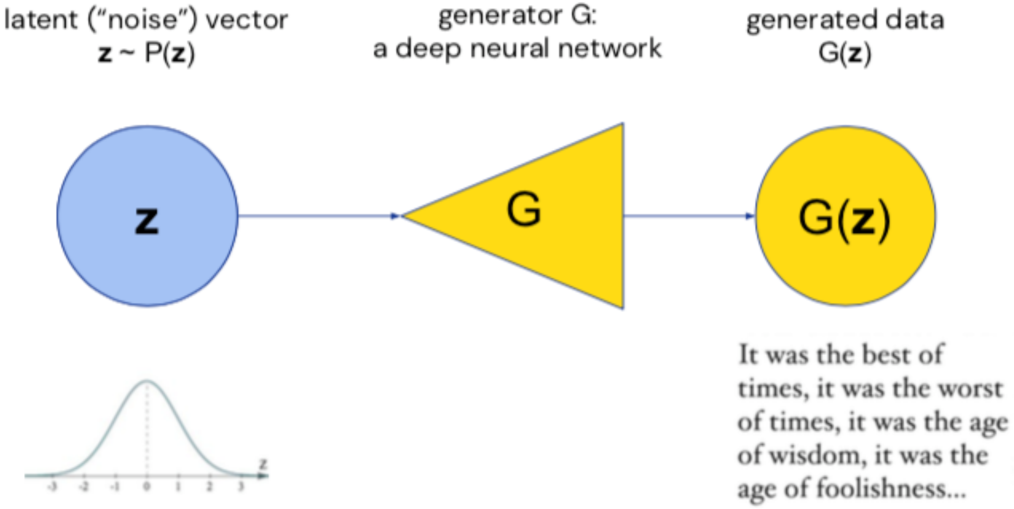
Generative Adversarial Networks



Generative Adversarial Networks

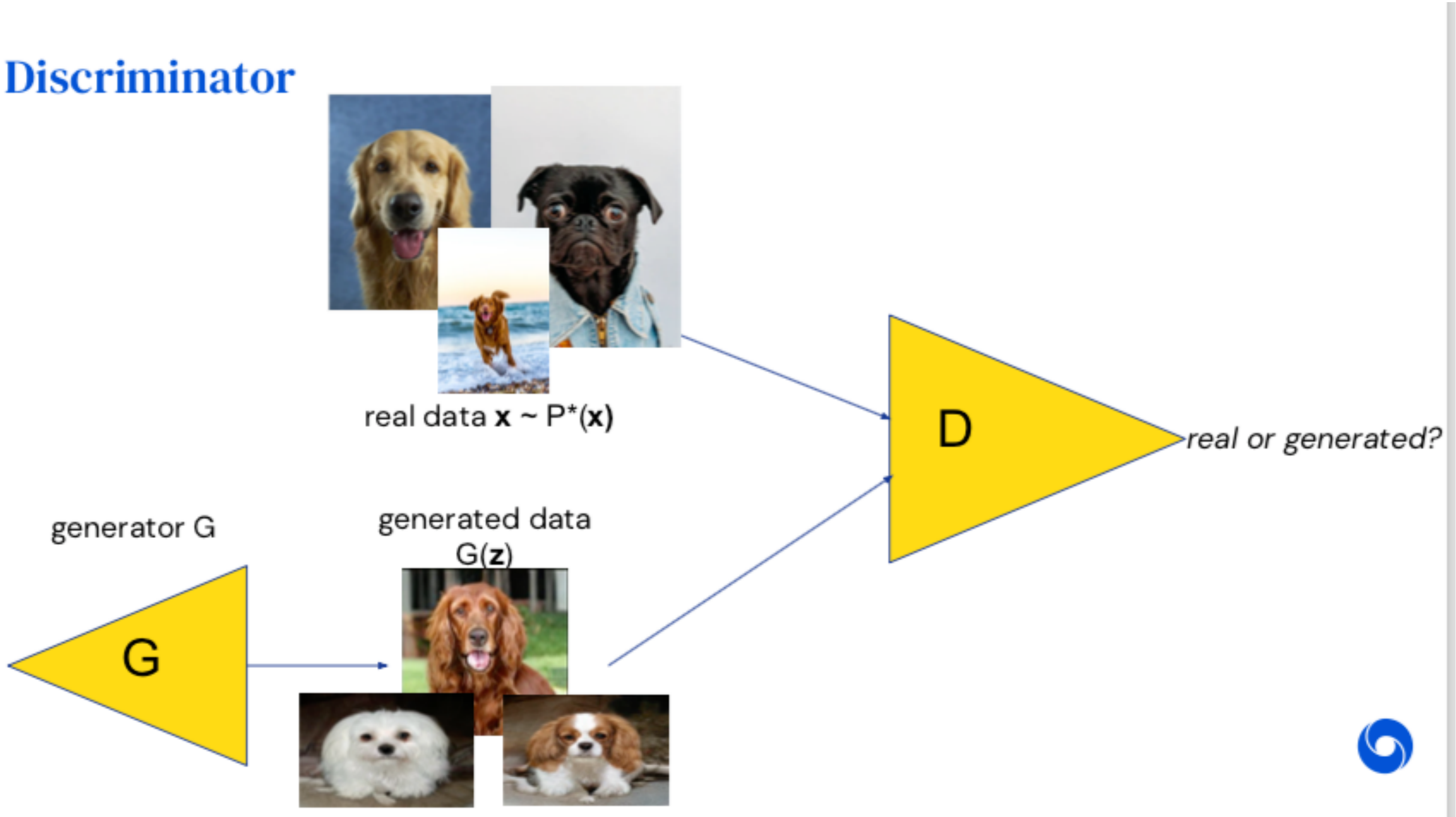


Generation of image data



Generation of text data

Generative Adversarial Networks



Generative Adversarial Networks

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

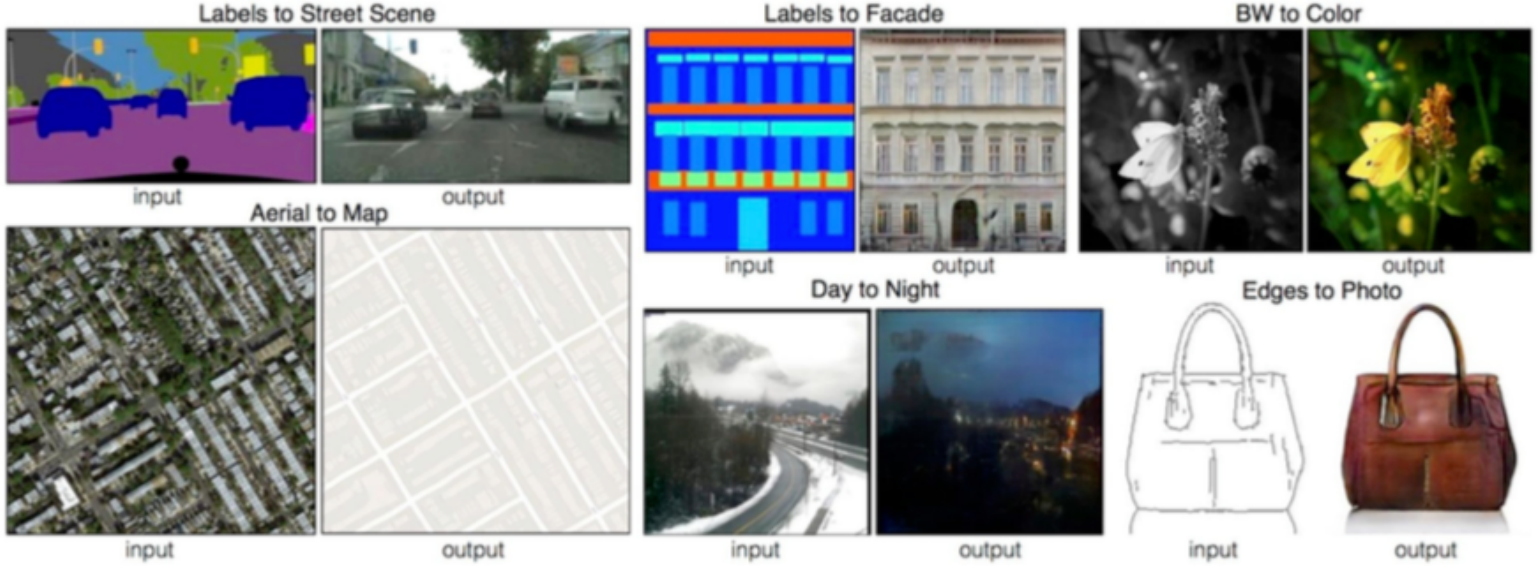
log-probability that D correctly predicts real data **x** are real

log-probability that D correctly predicts generated data G(**z**) are generated

discriminator's (D) goal: **maximize** prediction accuracy

generator's (G) goal: **minimize** D's prediction accuracy, by **fooling** D into believing its outputs G(**z**) are real as often as possible

Applications



Example results on several image-to-image translation problems. In each case we use the same architecture and objective, simply training on different data.

- Train a generator to **translate** between images of two different domains
- Standard GAN objective combined with reconstruction error

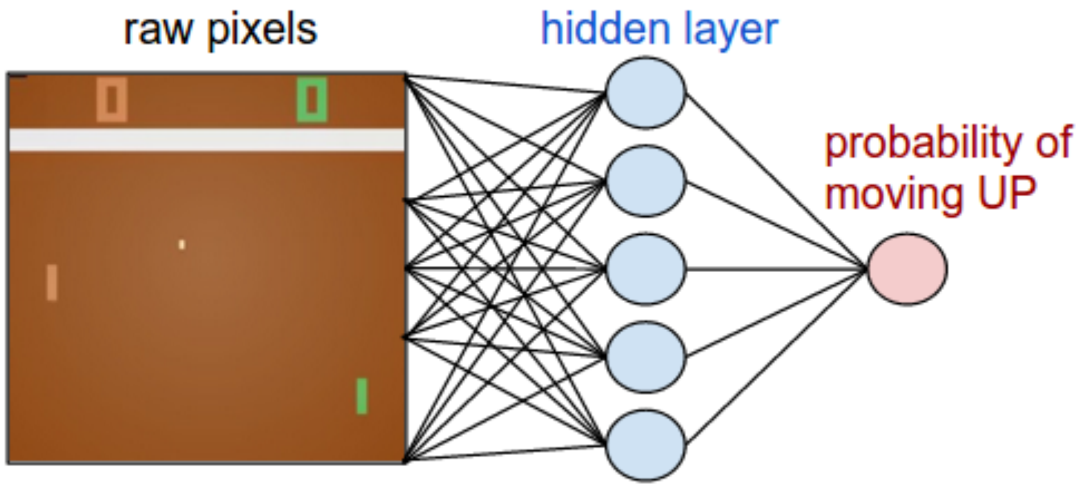
$$\mathcal{L}_{GAN}(G, D) = \mathbb{E}_y[\log D(y)] + \mathbb{E}_{x,z}[\log(1 - D(G(x, z))].$$

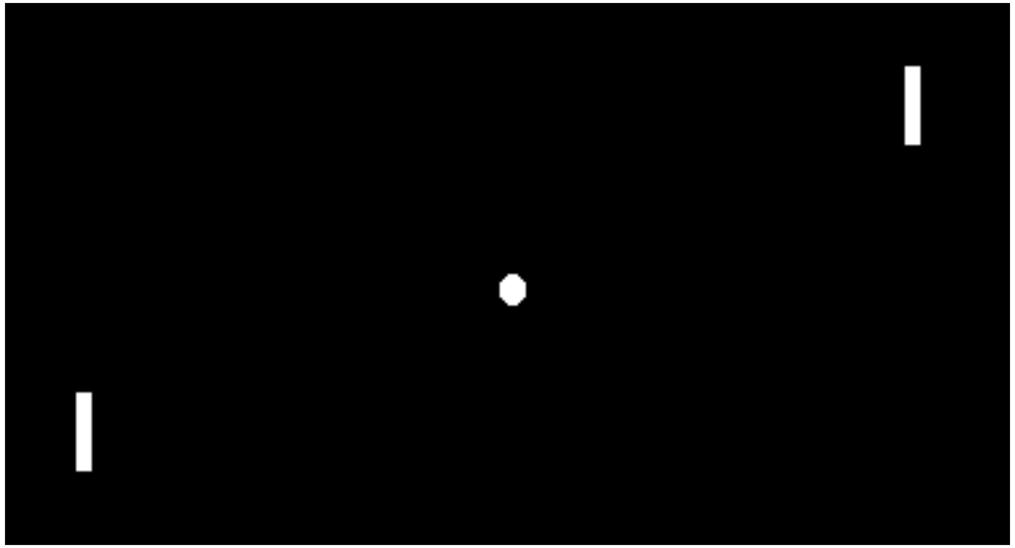
$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}[\|y - G(x, z)\|_1].$$

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G).$$

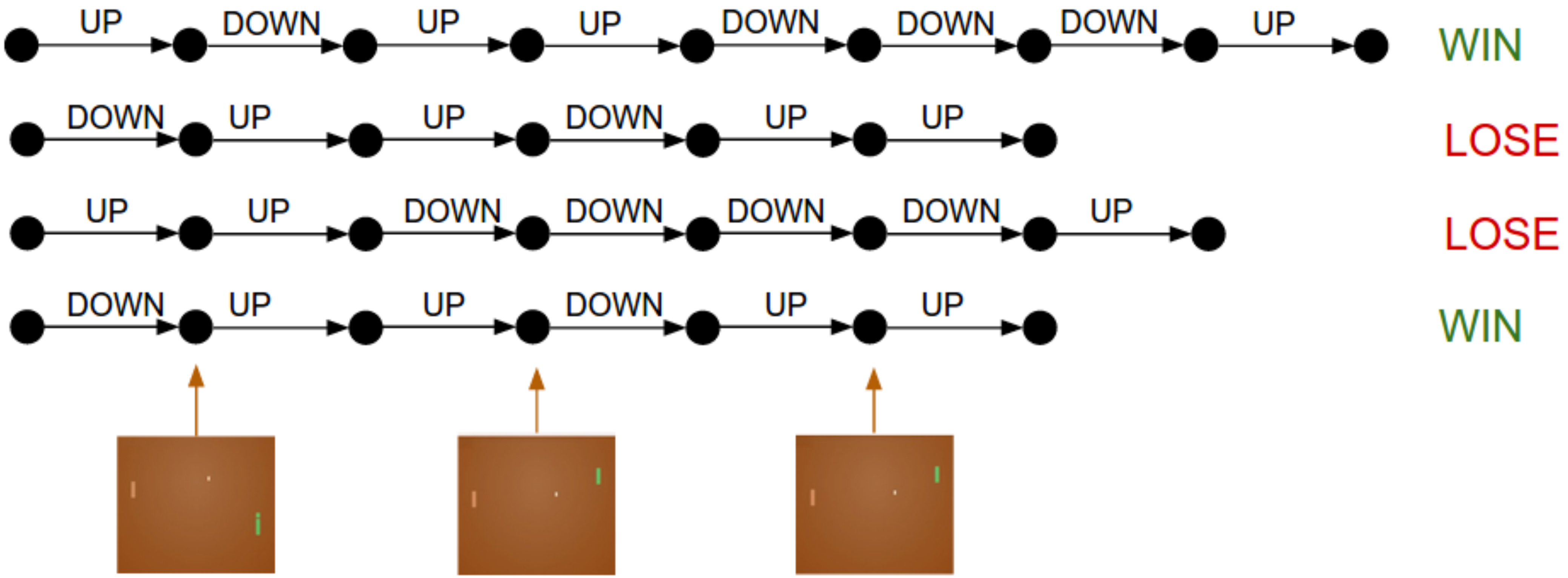
Pix2Pix (Isola et al.)

Neural Networks in RL





Sequential Task



Parameterized Policy

- Class of policies defined by parameters θ

$$\boxed{\pi_\theta(a|s) : \mathcal{S} \rightarrow \mathcal{A}}$$

  - Eg: θ can be parameters of linear transformation, deep network, etc.

- Want to maximize:

$$J(\pi) = \mathbb{E}\left[\sum_{t=1}^{T} \mathcal{R}(s_t, a_t)\right]$$

- In other words,

$$\pi^* = \arg \max_{\pi:\mathcal{S}\rightarrow\mathcal{A}} \mathbb{E}\left[\sum_{t=1}^{T} \mathcal{R}(s_t, a_t)\right] \Longrightarrow \theta^* = \arg\max_{\theta} \mathbb{E}\left[\sum_{t=1}^{T} \mathcal{R}(s_t, a_t)\right]$$

Training signal comes from reward

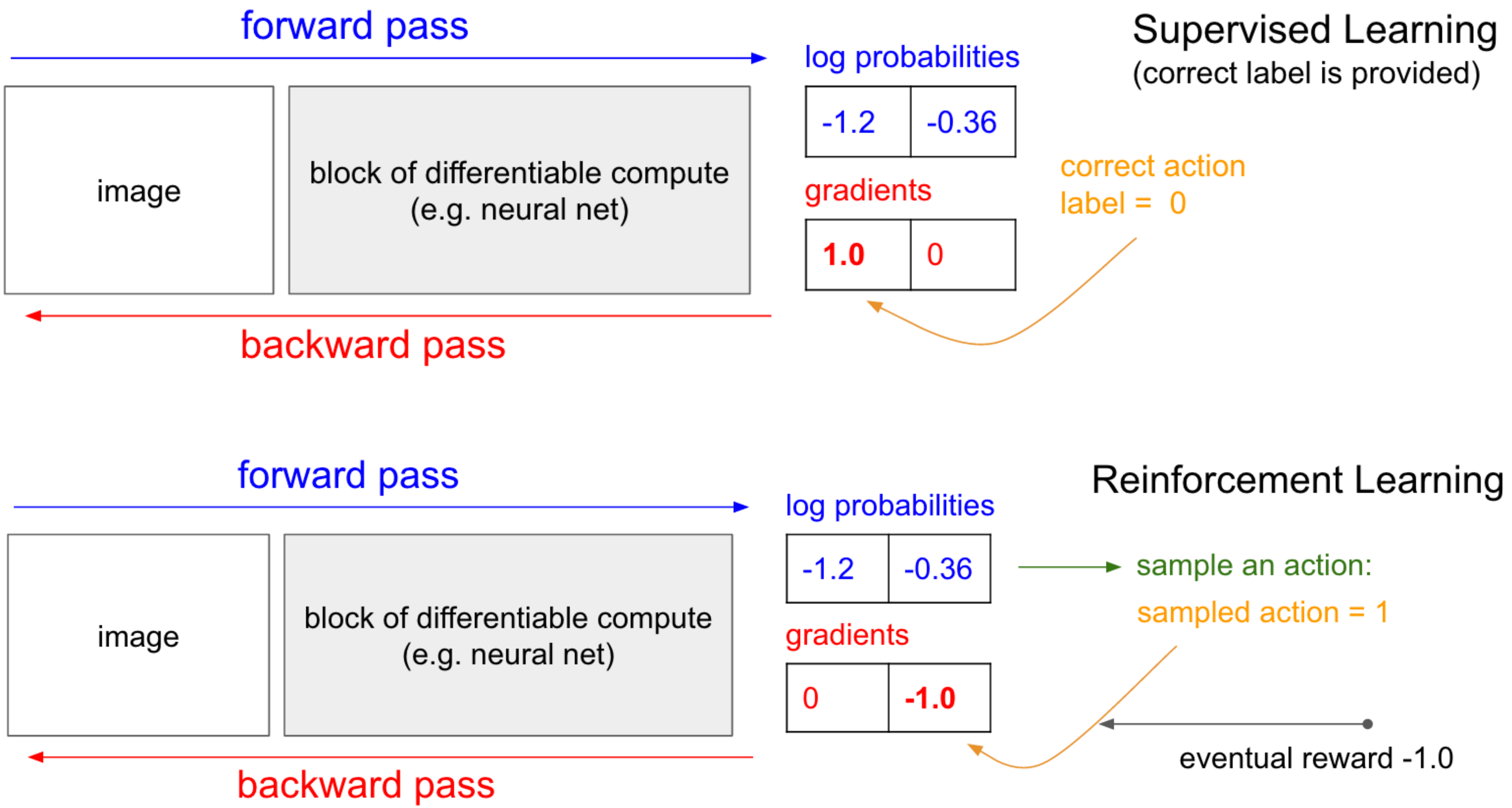


Image Source: http://karpathy.github.io/2016/05/31/rl/

Gathering Experience

- Slightly re-writing the notation

Let $\tau = (s_0, a_0, \ldots s_T, a_T)$ denote a trajectory

$$\pi_\theta(\tau) = p_\theta(\tau) = p_\theta\left(s_0, a_0, \ldots s_T, a_T\right)$$
$$= p(s_0) \prod_{t=0}^{T} p_\theta\left(a_t \mid s_t\right) \cdot p\left(s_{t+1} \mid s_t, a_t\right)$$

$$\arg\max_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)}\left[\mathcal{R}(\tau)\right]$$

Gathering Experience

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}\left[\mathcal{R}(\tau)\right]$$

$$= \mathbb{E}_{a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)}\left[\sum_{t=0}^{T} \mathcal{R}(s_t, a_t)\right]$$

- How to gather data?
  - We already have a policy: π_θ
  - Sample N trajectories $\{\tau_i\}_{i=1}^N$ by acting according to π_θ

$$\approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} r(s_t^i, a_t^i)$$

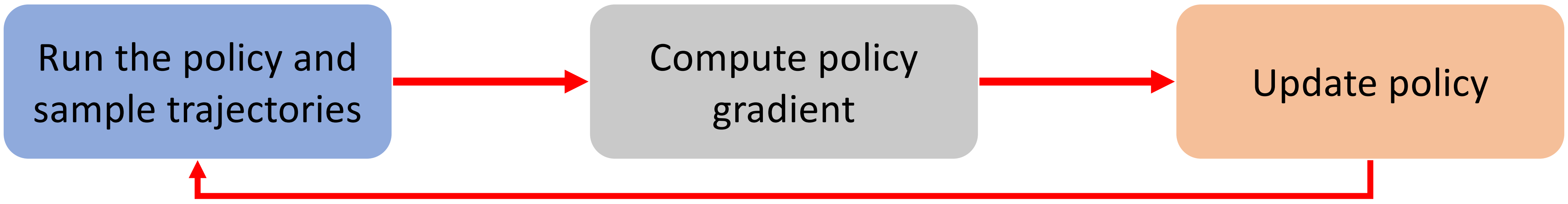
Reinforce Algorithm

- Sample trajectories $\tau_i = \{s_1, a_1, \ldots s_T, a_T\}_i$ by acting according to π_θ

- Compute policy gradient as

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i}^{N} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta \left(a_t^i \mid s_t^i \right) \cdot \sum_{t=1}^{T} \mathcal{R} \left(s_t^i \mid a_t^i \right) \right]$$

- Update policy parameters: $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$



Slide credit: Sergey Levine