



COL333/671: Introduction to AI

Semester I, 2024-25

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.

Learning a Best Fit Hypothesis: Linear Regression Task

Linear regression (no bias)

$$\hat{y} = \mathbf{w}^\top \mathbf{x}$$

Linear regression (with bias)

$$\hat{y} = \mathbf{w}^\top \mathbf{x} + b$$

Error term/Loss term

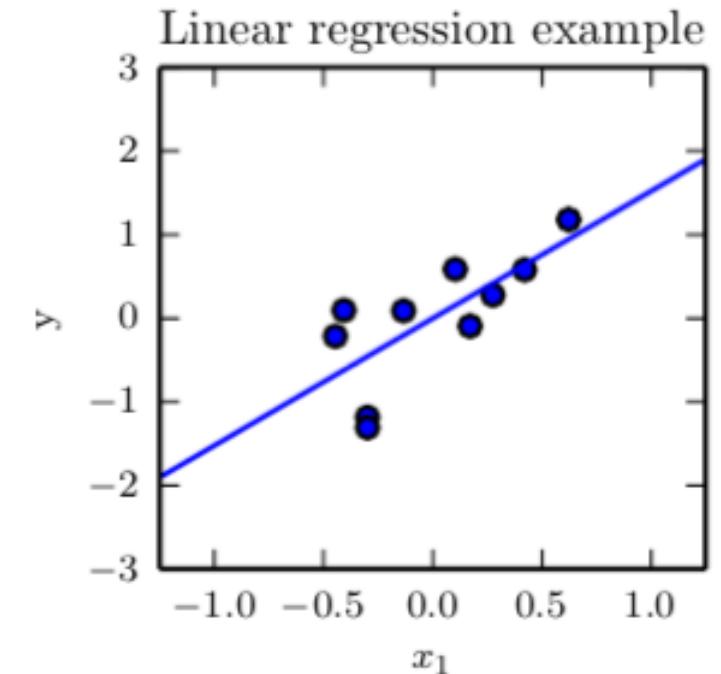
$$\text{MSE}_{\text{test}} = \frac{1}{m} \|\hat{\mathbf{y}}^{(\text{test})} - \mathbf{y}^{(\text{test})}\|_2^2,$$

Learning: Optimizing the loss will estimate the model parameters w and b .

Online: Given the trained model, we can predict a value.

Examples:

- Predicting pollution levels from visibility
- Predicting reactivity of a molecule from structural data.



Linear Regression Example

Learning/Training:

Optimize the error w.r.t. the model parameters, w

$$\nabla_w \text{MSE}_{\text{train}} = 0$$

$$\Rightarrow \nabla_w \frac{1}{m} \|\hat{\mathbf{y}}^{(\text{train})} - \mathbf{y}^{(\text{train})}\|_2^2 = 0$$

$$\Rightarrow \frac{1}{m} \nabla_w \|\mathbf{X}^{(\text{train})} \mathbf{w} - \mathbf{y}^{(\text{train})}\|_2^2 = 0$$

$$\Rightarrow \nabla_w (\mathbf{X}^{(\text{train})} \mathbf{w} - \mathbf{y}^{(\text{train})})^\top (\mathbf{X}^{(\text{train})} \mathbf{w} - \mathbf{y}^{(\text{train})}) = 0 \quad (5.9)$$

$$\Rightarrow \nabla_w (\mathbf{w}^\top \mathbf{X}^{(\text{train})\top} \mathbf{X}^{(\text{train})} \mathbf{w} - 2\mathbf{w}^\top \mathbf{X}^{(\text{train})\top} \mathbf{y}^{(\text{train})} + \mathbf{y}^{(\text{train})\top} \mathbf{y}^{(\text{train})}) = 0 \quad (5.10)$$

$$\Rightarrow 2\mathbf{X}^{(\text{train})\top} \mathbf{X}^{(\text{train})} \mathbf{w} - 2\mathbf{X}^{(\text{train})\top} \mathbf{y}^{(\text{train})} = 0 \quad (5.11)$$

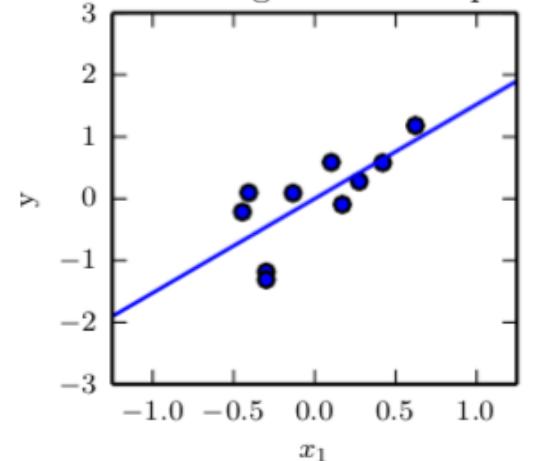
$$\Rightarrow \mathbf{w} = (\mathbf{X}^{(\text{train})\top} \mathbf{X}^{(\text{train})})^{-1} \mathbf{X}^{(\text{train})\top} \mathbf{y}^{(\text{train})} \quad (5.12)$$

Inference:

Use the trained model i.e. w , to perform predictions

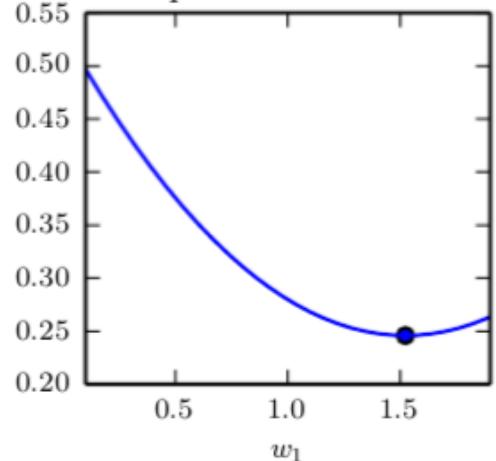
Optimal w and the implied linear model

Linear regression example



Function/model space

Optimization of w



Parameter space

Classification Task

Softmax applied at the last stage

$$\mathbf{z} = z_\theta(\mathbf{x})$$

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z})$$

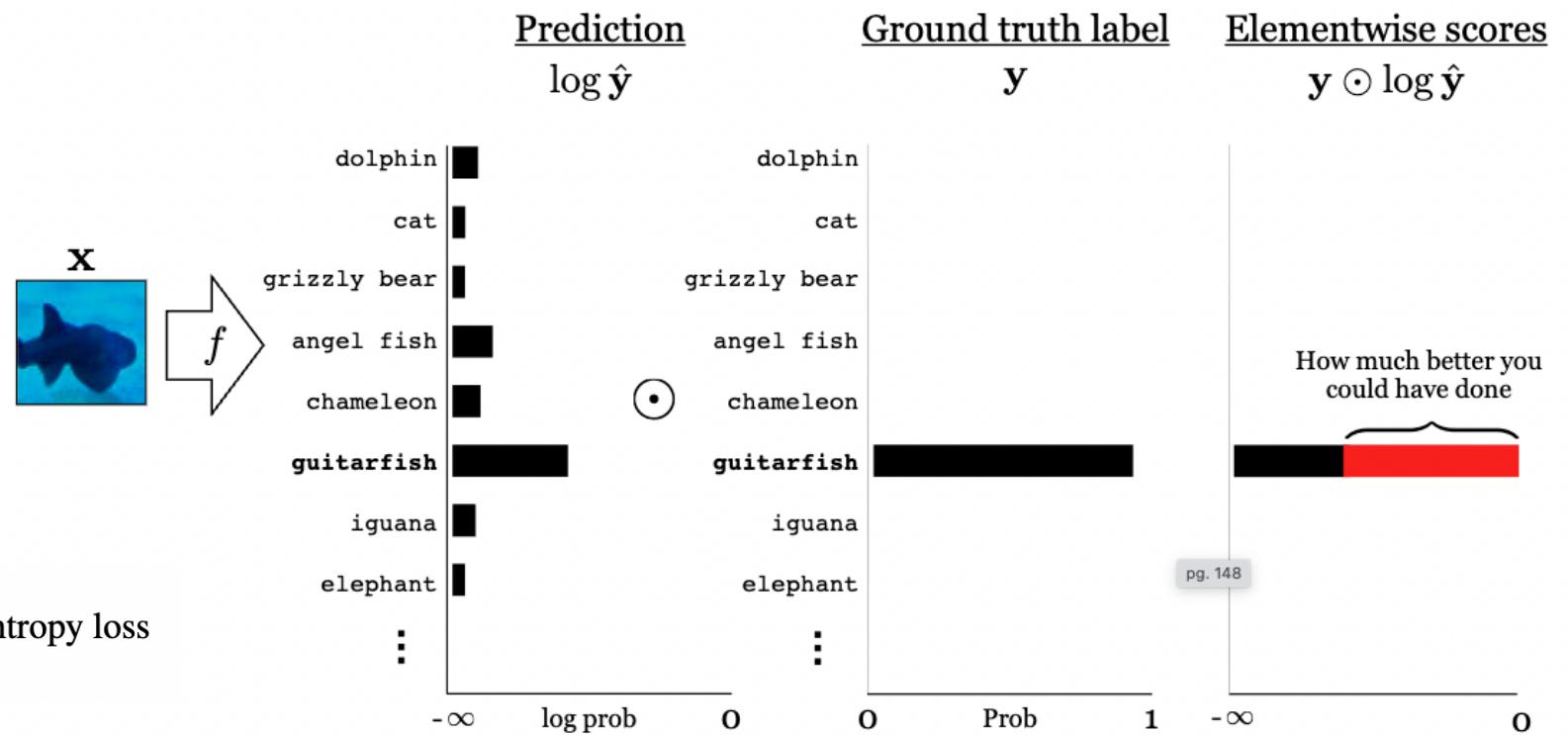
$$\hat{y}_j = \frac{e^{-z_j}}{\sum_{i=1}^K e^{-z_i}}$$

pg. 148

$$\hat{\mathbf{y}} = f_\theta(\mathbf{x}) = \text{softmax}(z_\theta(\mathbf{x}))$$

$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = H(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{k=1}^K y_k \log \hat{y}_k \quad / \quad \text{cross-entropy loss}$$

Classification of an image as containing certain species.



How much to fit the data?

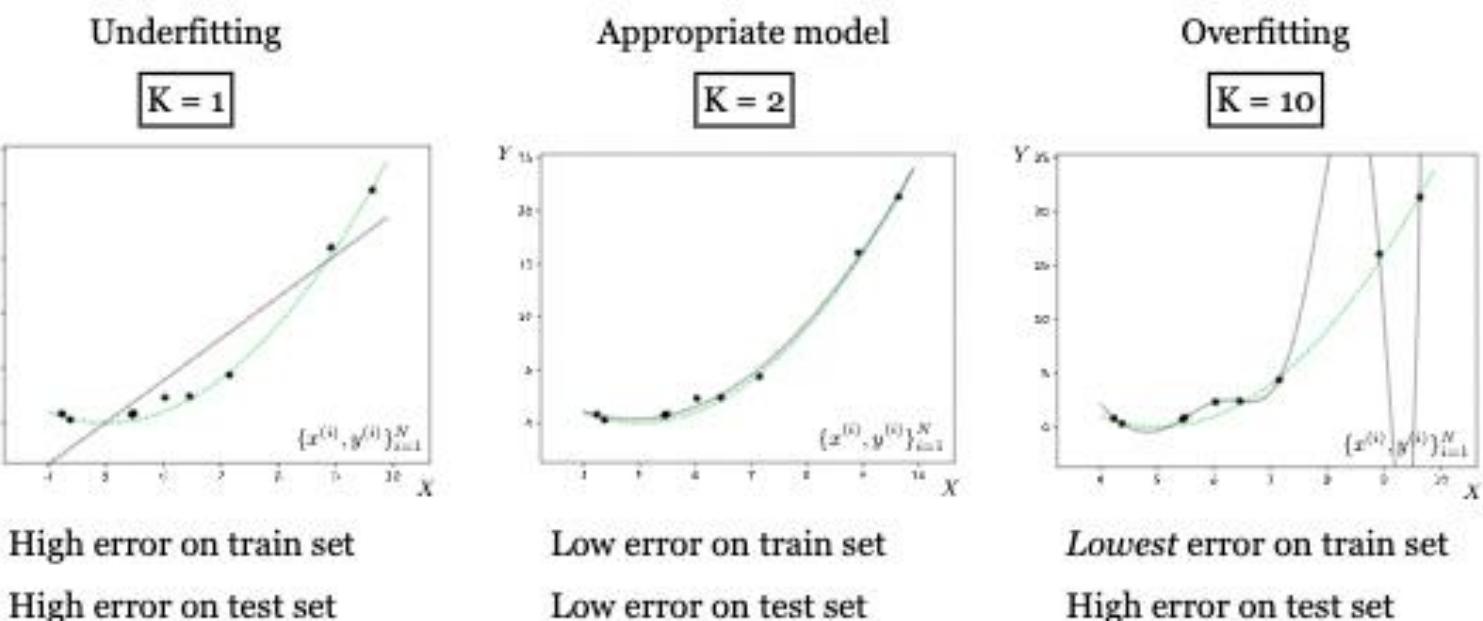
Overfitting and underfitting with polynomial functions

We seek a reasonable model that is neither underfitting nor over fitting.

That means, we prefer certain types of models.

How to “regularize” or solution, incorporate certain preferences?

Figure shows polynomial curve fitting with increasing k



Core problem in machine learning: How to fit the data just enough?

Regularization

- Adding a **preference** for one solution in its hypothesis space to another.
 - Incorporate that preference in the objective function we are optimization.

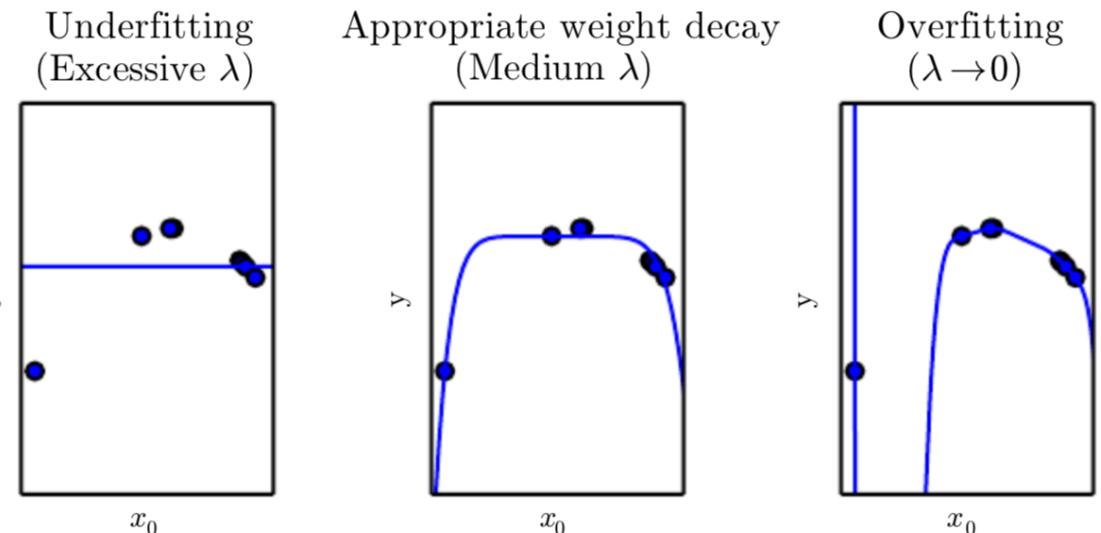
- **Weight term**

- Adding a term to the loss function that prefers smaller squared sum of weights. A prior over the parameters.
- Penalize a very complex model to explain the date.

- **Lambda parameter**

- Selected ahead of time that controls the strength of our preference for smaller weights.
- It is a “hyper”-parameter that needs tuning, expressing our trade off.

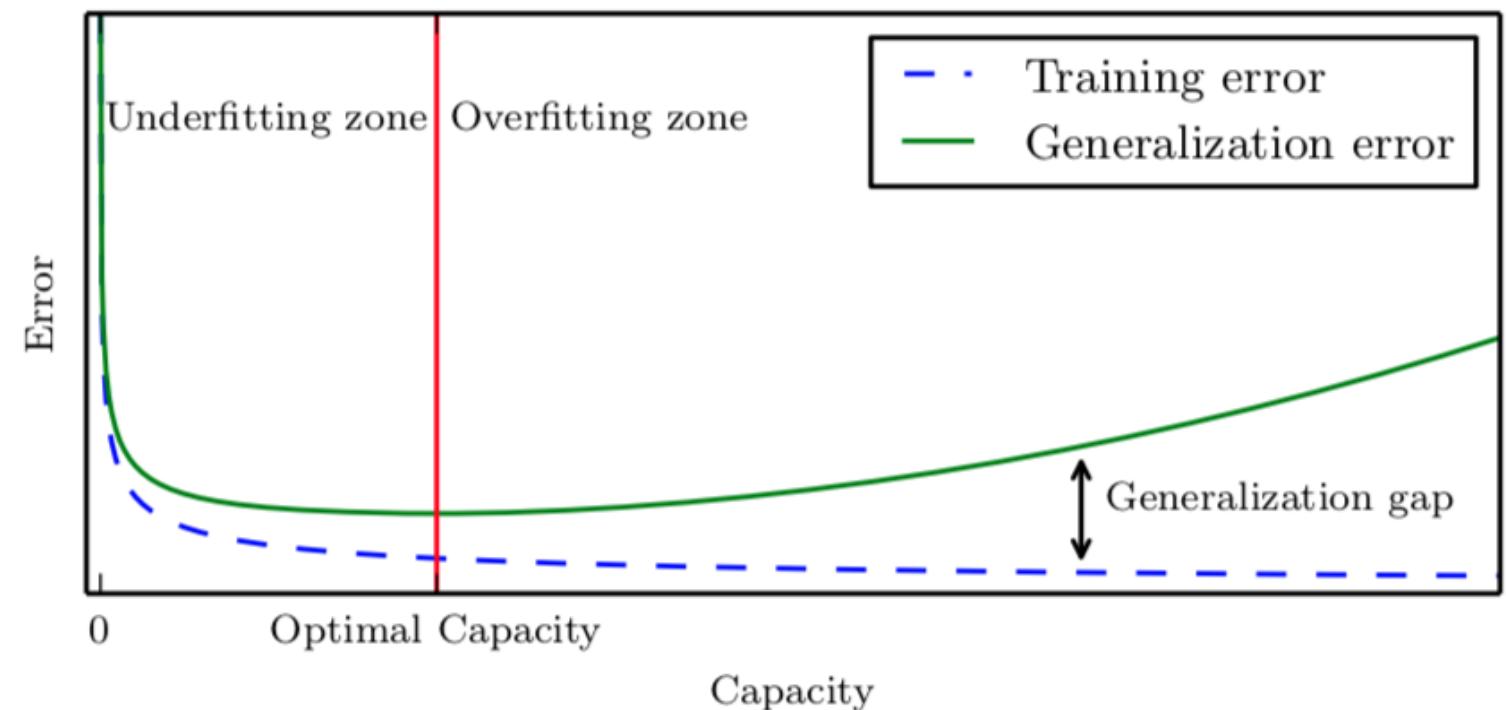
$$J(\mathbf{w}) = \text{MSE}_{\text{train}} + \lambda \mathbf{w}^\top \mathbf{w}$$



Material from
<https://www.deeplearningbook.org/>
Chapter 5

Relation between model capacity and error

- **Underfitting regime**
 - Training error and generalization error are both high.
- **Increase capacity**
 - Training error decreases
 - Gap between training and generalization error increases.
- **Overfitting**
 - The size of this gap outweighs the decrease in training error,
 - capacity is too large, above the optimal capacity.



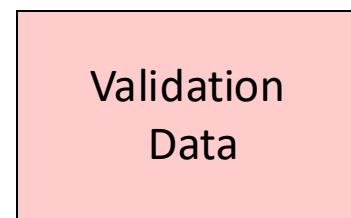
We can train a model only on the training set. The test set is not available during training.

How do we know the generalization error when we cannot use the test set?

Material from
<https://www.deeplearningbook.org/>
Chapter 5

Hyper-parameters and Validation Sets

- Parameters: $P(X)$, $P(Y|X)$
- Hyper-parameters: k, lambda etc.
- Selecting hyper-parameters
 - For each value of the hyperparameters, train and test on the held-out data or the validation data set.
 - Choose the best value and do a final test on the test data



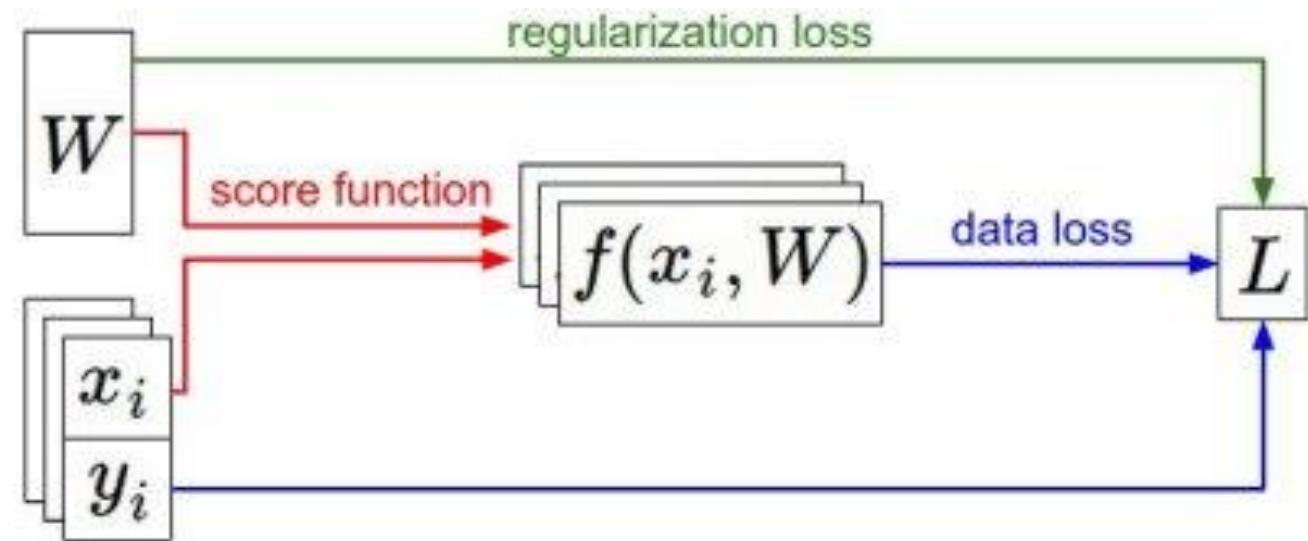
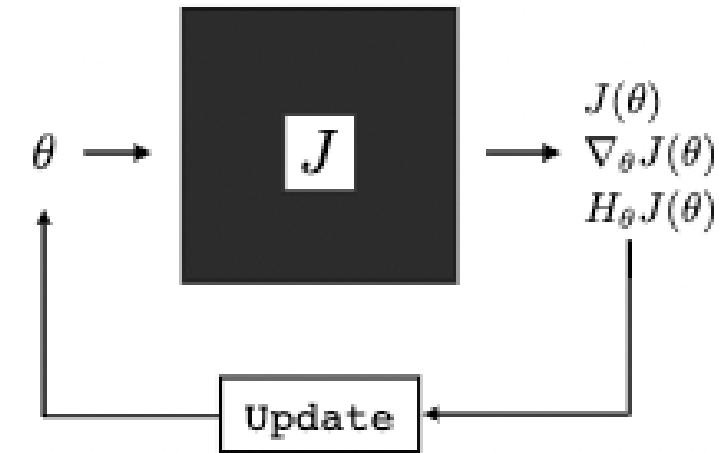
$$J_{\text{approx}} = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(f_\theta(x_{(\text{train})}^{(i)}), y_{(\text{train})}^{(i)})$$

$$J_{\text{gen}} = \mathbb{E}_{x,y \sim p_{\text{data}}} [\mathcal{L}(f_\theta(x), y)]$$

$$\approx \frac{1}{N} \sum_{i=1}^N \mathcal{L}(f_\theta(x_{(\text{val})}^{(i)}), y_{(\text{val})}^{(i)})$$

Gradient-based Learning

- The loss term is composed of the predictions under the weights and the regularization term.
- The goal is to optimize the loss function given the parameters



Gradient Descent

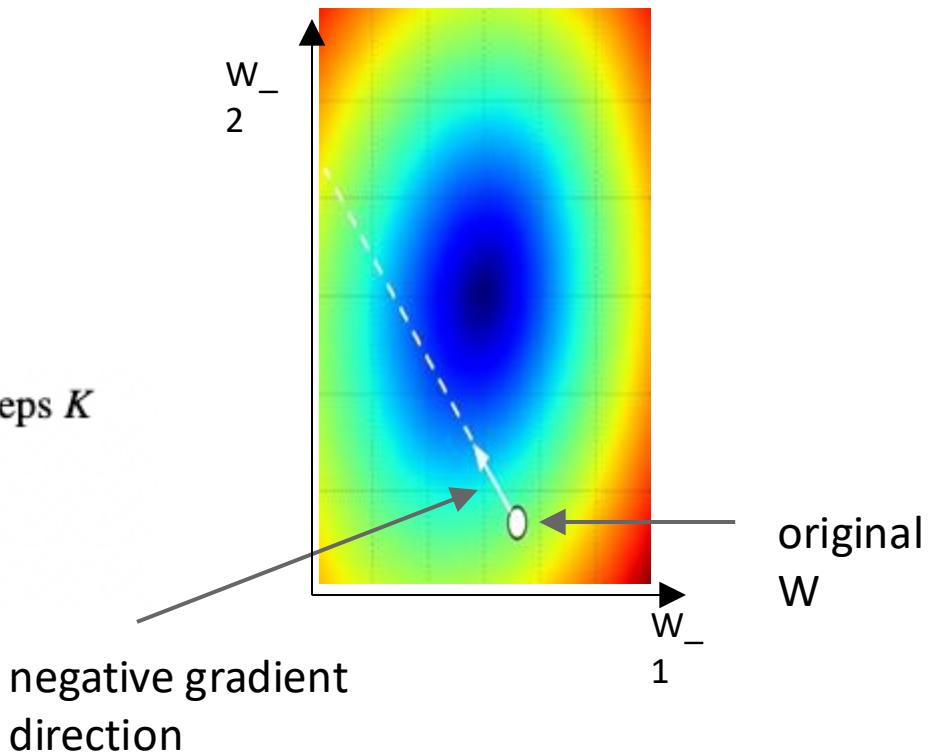
- Compute the gradient and take the step in that direction.
- Gradient computation
 - Analytic
 - Numerical

1 **Input:** objective function J , initial parameter vector θ^0 , learning rate η , number of steps K

2 **Output:** trained parameter vector $\theta^* = \theta^K$

3 **for** $k=0, \dots, K-1$ **do**

4 $\theta^{k+1} \leftarrow \theta^k - \eta \nabla_{\theta} J(\theta^k)$



Learning rate schedules

Gradient descent with adaptive learning rate.

- 1 **Input:** objective function J , initial parameter vector θ^0 , initial learning rate η^0 , learning rate function lr , number of steps K
- 2 **Output:** trained parameter vector $\theta^* = \theta^K$
- 3 **for** $k=0, \dots, K-1$ **do**
- 4 $\eta^k \leftarrow \text{lr}(\eta^0, k)$
- 5 $\theta^{k+1} \leftarrow \theta^k - \eta^k \nabla_{\theta} J(\theta^k)$

Types of learning rate schedules.

$$\text{lr}(\eta^0, k) = \beta^{-k} \eta^0 \quad / \quad \text{exponential decay}$$

$$\text{lr}(\eta^0, k) = \beta^{-\lfloor k/M \rfloor} \eta^0 \quad / \quad \text{stepwise exponential decay}$$

$$\text{lr}(\eta^0, k) = \frac{(K-k)}{K} \eta^0 \quad / \quad \text{linear decay}$$

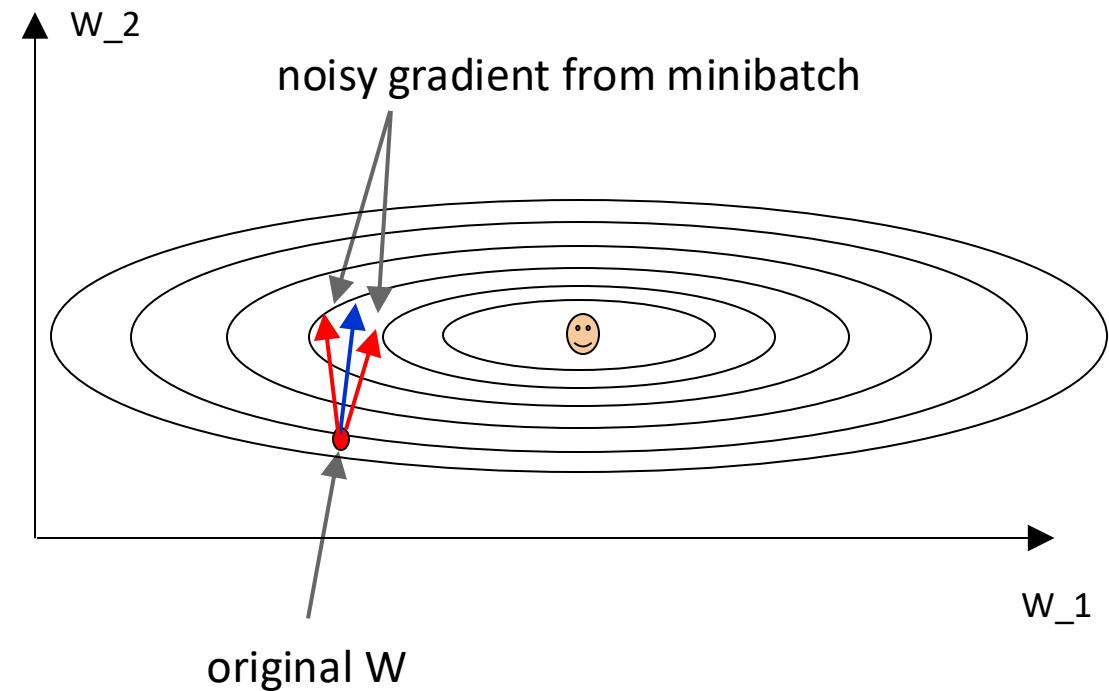
Mini-batch Gradient Descent

Problem: Large data sets. Difficult to compute the full loss function over the entire training set in order to perform only a single parameter update

Solution: Only use a small portion of the training set to compute the gradient.

```
# Vanilla Minibatch Gradient Descent

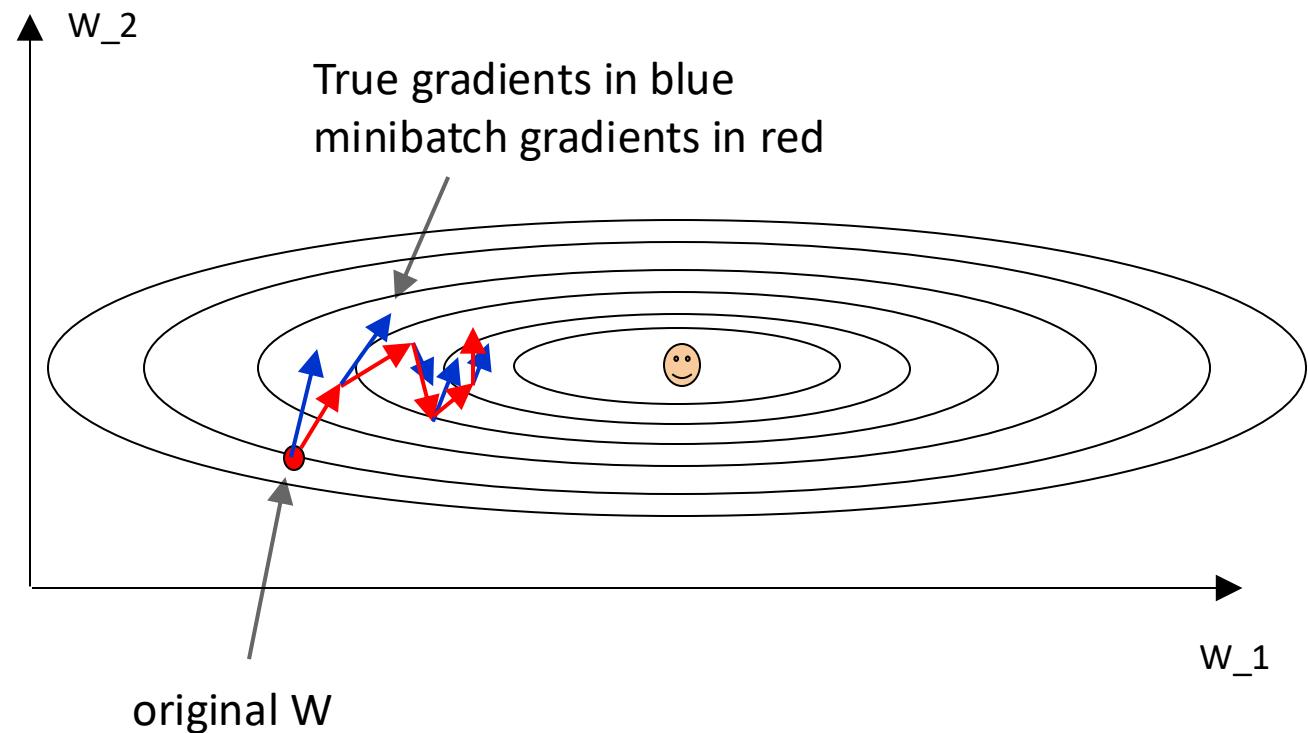
while True:
    data_batch = sample_training_data(data, 256) # sample 256 examples
    weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
    weights += - step_size * weights_grad # perform parameter update
```



Stochastic Gradient Descent

Setting the mini-batch to contain only a single example.

Gradients are noisy but still make good progress on average.



Gradient vs Mini-batch Gradient Descent

Cost function decomposes as a sum over training examples of per-example loss function

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}, y \sim \hat{p}_{\text{data}}} L(\mathbf{x}, y, \boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m L(\mathbf{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

$L(\mathbf{x}, y, \boldsymbol{\theta}) = -\log p(y \mid \mathbf{x}; \boldsymbol{\theta})$

Gradient for an additive cost function

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m \nabla_{\boldsymbol{\theta}} L(\mathbf{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

Mini-batch m' , where m' is kept constant while m is increasing.

Update the parameters with the estimated gradient from the mini-batch.

$$\mathbf{g} = \frac{1}{m'} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m'} L(\mathbf{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}) \quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \mathbf{g}$$

Stochastic Gradient Descent

$$\{\mathbf{x}^{(b)}, \mathbf{y}^{(b)}\}_{b=1}^B$$

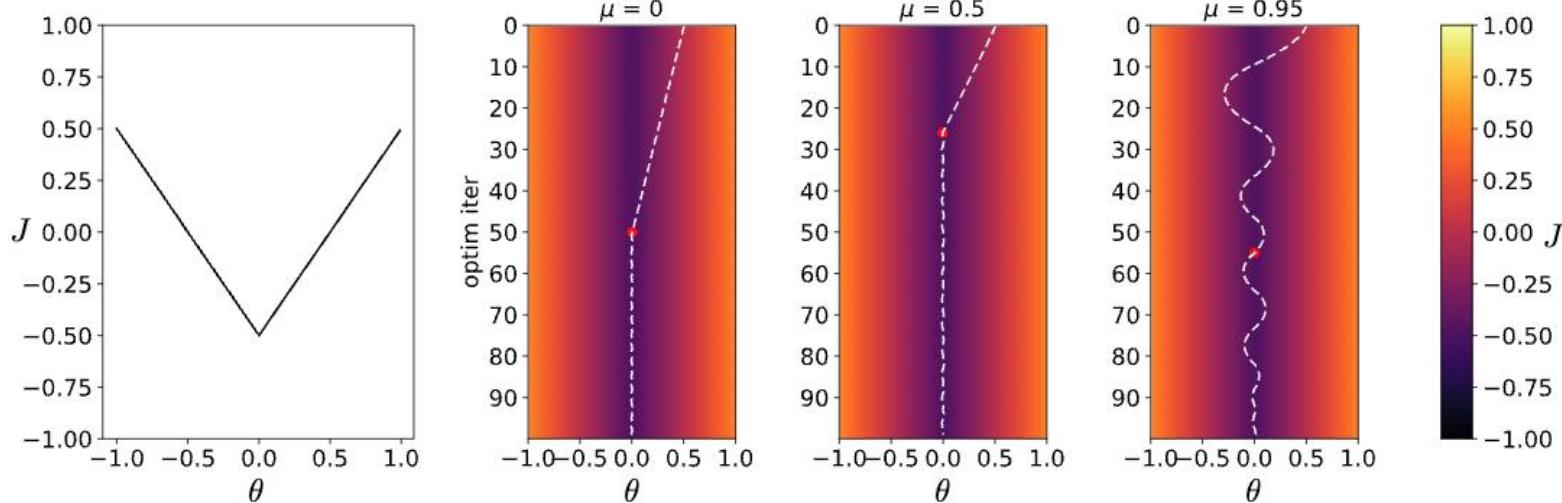
$$\tilde{\mathbf{g}} = \frac{1}{N} \sum_{b=1}^B \nabla_{\theta} \mathcal{L}(f_{\theta}(\mathbf{x}^{(b)}), \mathbf{y}^{(b)})$$

```
1 Input: initial parameter vector  $\theta^0$ , data  $\{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^N$ , learning rate  $\eta$ , batch size  $B$ , number of steps  $K$ 
2 Output: trained parameter vector  $\theta^* = \theta^K$ 
3 for  $k=0, \dots, K-1$  do
4    $\{\mathbf{x}^{(b)}, \mathbf{y}^{(b)}\}_{b=1}^B \sim \{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^N$       / sample batch of training data
5    $\tilde{\mathbf{g}} = \frac{1}{N} \sum_{b=1}^B \nabla_{\theta} \mathcal{L}(f_{\theta}(\mathbf{x}^{(b)}), \mathbf{y}^{(b)})$ 
6    $\theta^k \leftarrow \theta^{k-1} - \eta \tilde{\mathbf{g}}$ 
```

SGD – sample the batch randomly in each iteration. Random sampling adds some robustness to optimization.

Gradient Descent with Momentum

- 1 **Input:** objective function J , initial parameter vector θ^0 , learning rate η , momentum μ , number of steps K
- 2 **Output:** trained parameter vector $\theta^* = \theta^K$
- 3 $\mathbf{v}^0 = \mathbf{0}$
- 4 **for** $k=0, \dots, K-1$ **do**
- 5 $\mathbf{v}^{k+1} = \mu \mathbf{v}^k - \eta \nabla_{\theta} J(\theta^k)$
- 6 $\theta^{k+1} \leftarrow \theta^k + \mathbf{v}^{k+1}$

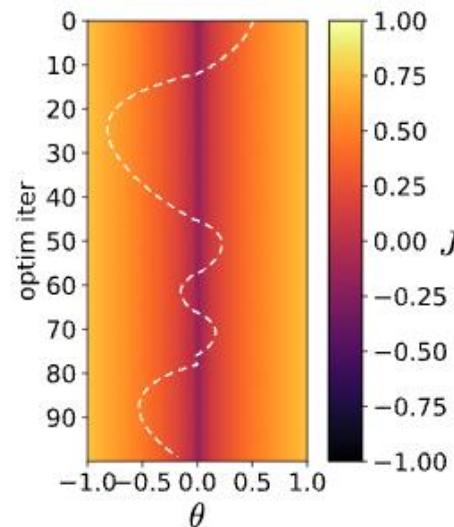
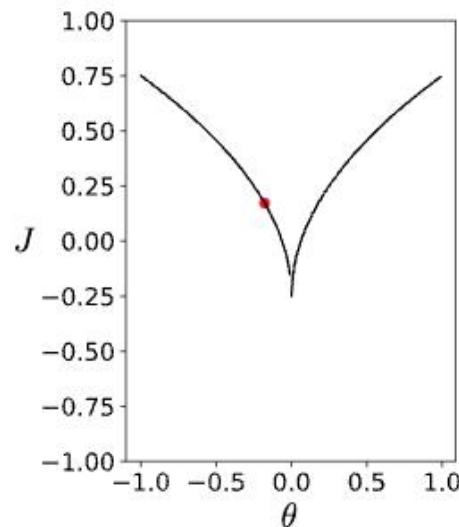


Gradient Clipping

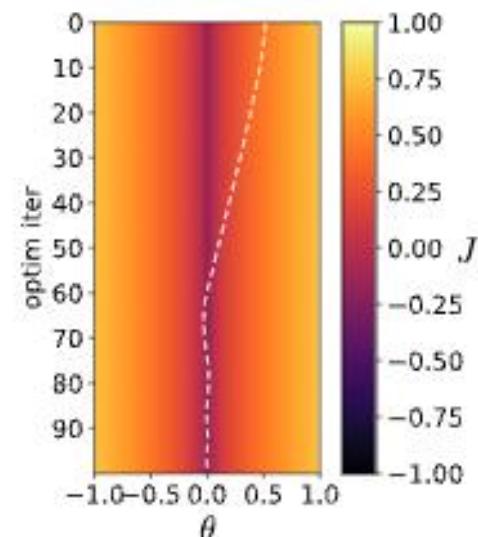
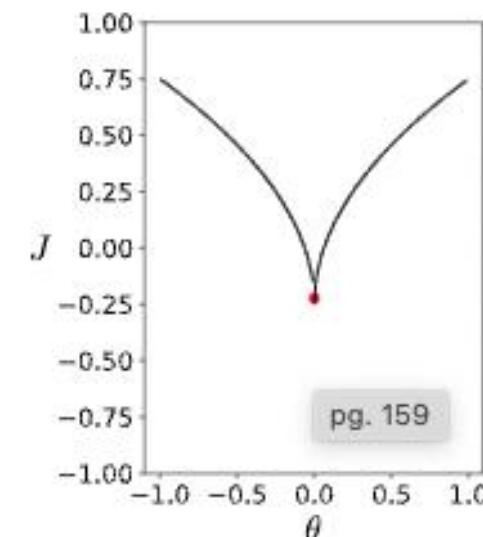
clip is the “clipping” function: $\text{clip}(v, -m, m) = \max(\min(v, m), -m)$

- 1 **Input:** objective function J , initial parameter vector θ^0 , learning rate η , number of steps K , max gradient magnitude m
- 2 **Output:** trained parameter vector $\theta^* = \theta^K$
- 3 **for** $k = 0, \dots, K-1$ **do**
- 4 $v = \nabla_{\theta} J(\theta^k)$
- 5 $\theta^{k+1} \leftarrow \theta^k - \eta [\text{clip}(v_1, -m, m), \dots, \text{clip}(v_M, -m, m)]^T$

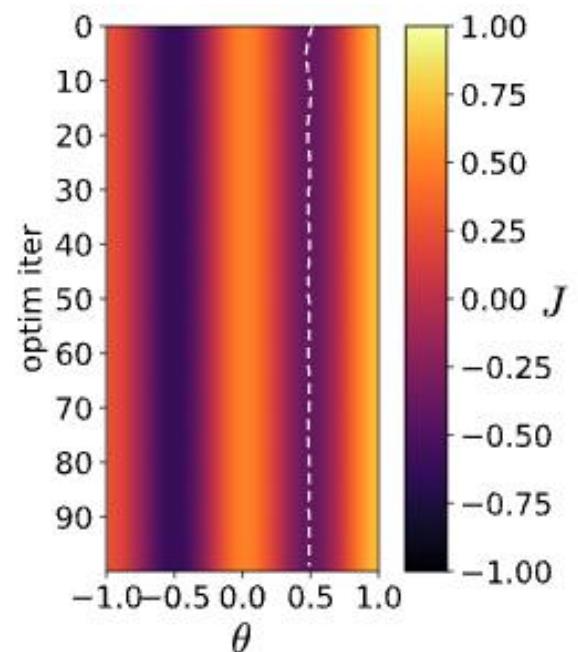
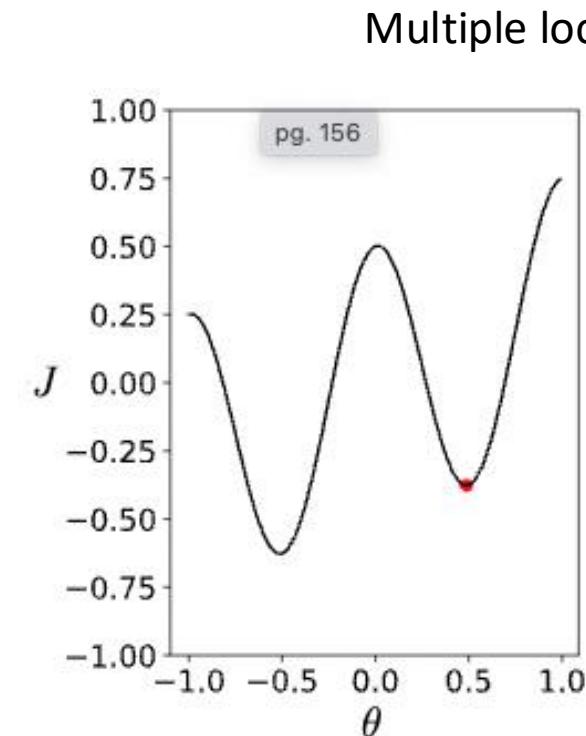
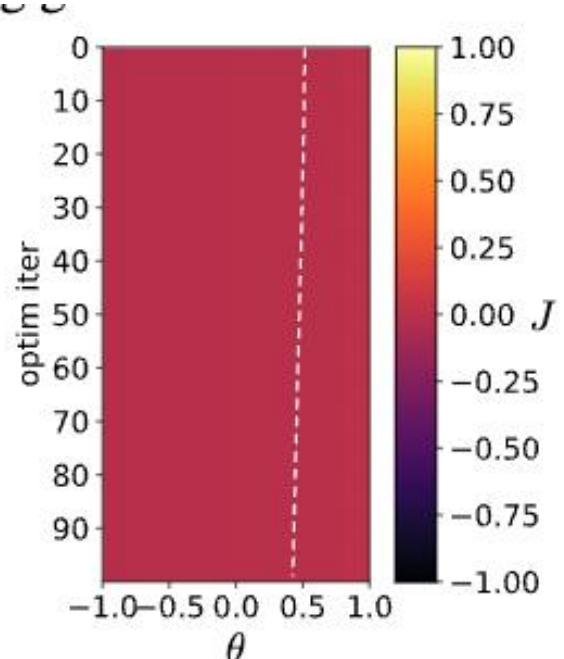
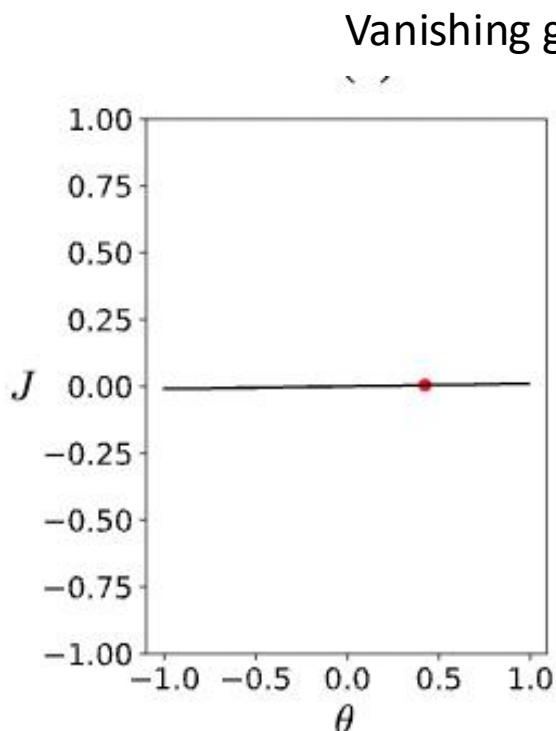
Without clipping – instabilities



With clipping – stable convergence



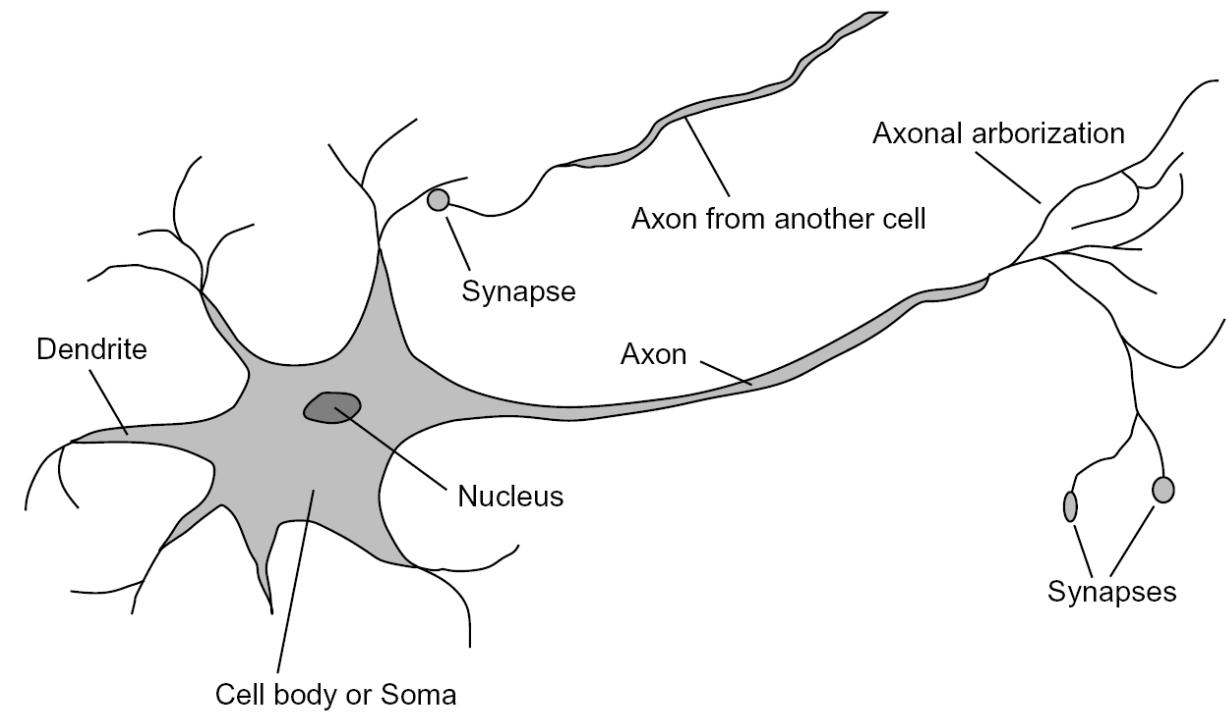
Other Aspects



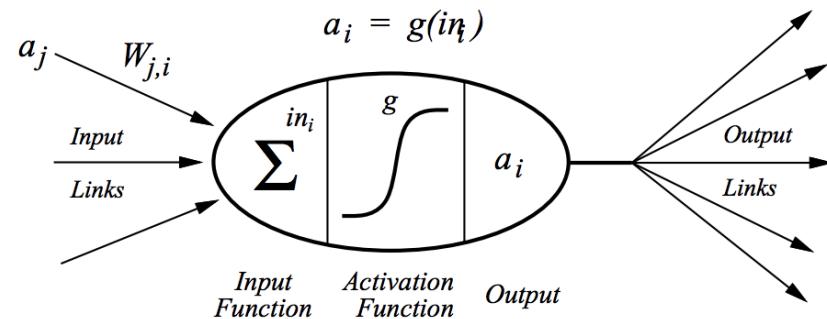
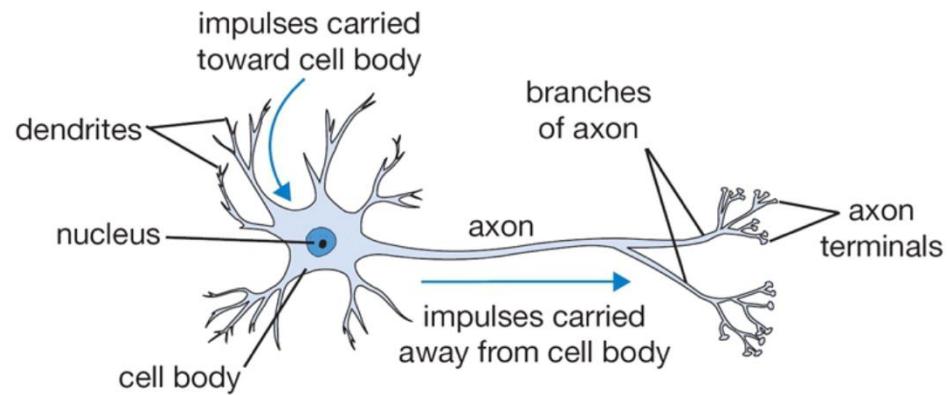
Neural Networks: *Biological*

Neurons in the brain

- Activations and inhibitions
- Parallelism
- Connected networks



Modeling a Neuron



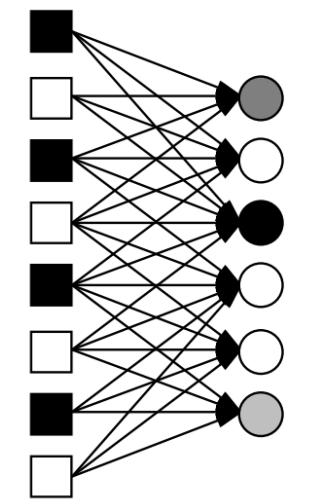
$$a_i = g\left(\sum_j W_{j,i} a_j\right)$$

Main processing unit

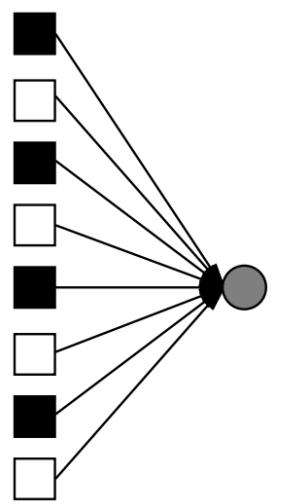
- Setup where there is a function that connects the inputs to the output.
- Problem: learning this function that maps the input to the output.

Perceptron: Model of a Single Neuron

- 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.



I_j $W_{j,i}$ O_i
Input Units Output Units
Perceptron Network



I_j W_j O
Input Units Output Unit
Single Perceptron

Feature Space

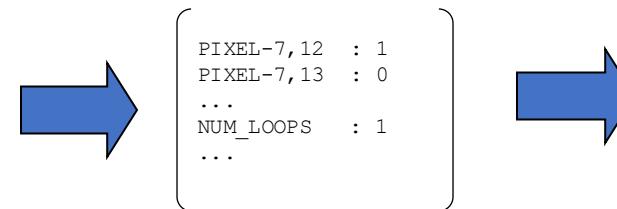
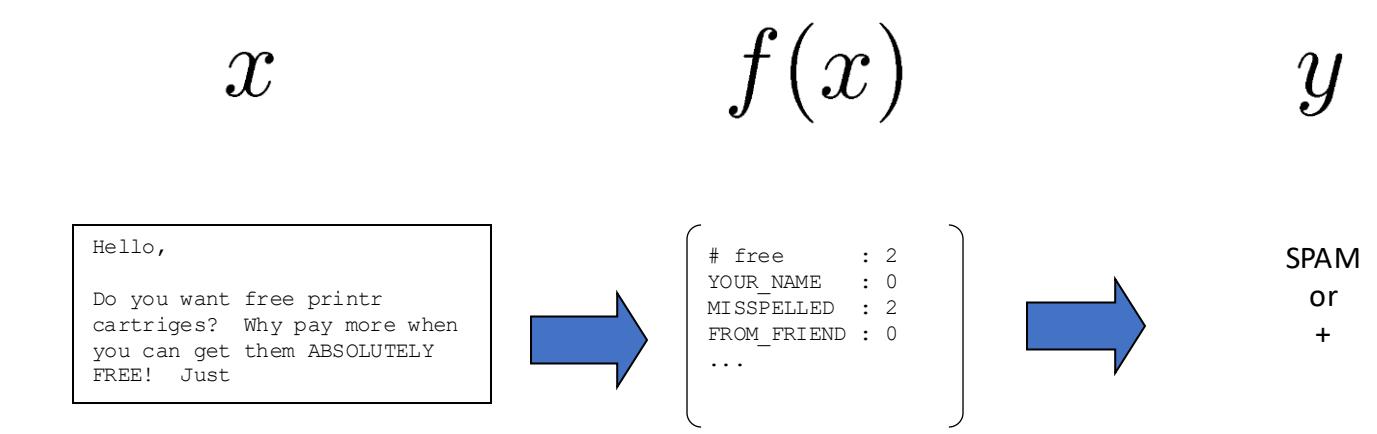
- Extract Features from data
- Learn a model with these features
- Data can be viewed as a point in the feature space.

$$f(x_1)$$

```
# free      : 1  
YOUR_NAME  : 0  
MISSPELLED : 1  
FROM_FRIEND : 0  
...
```

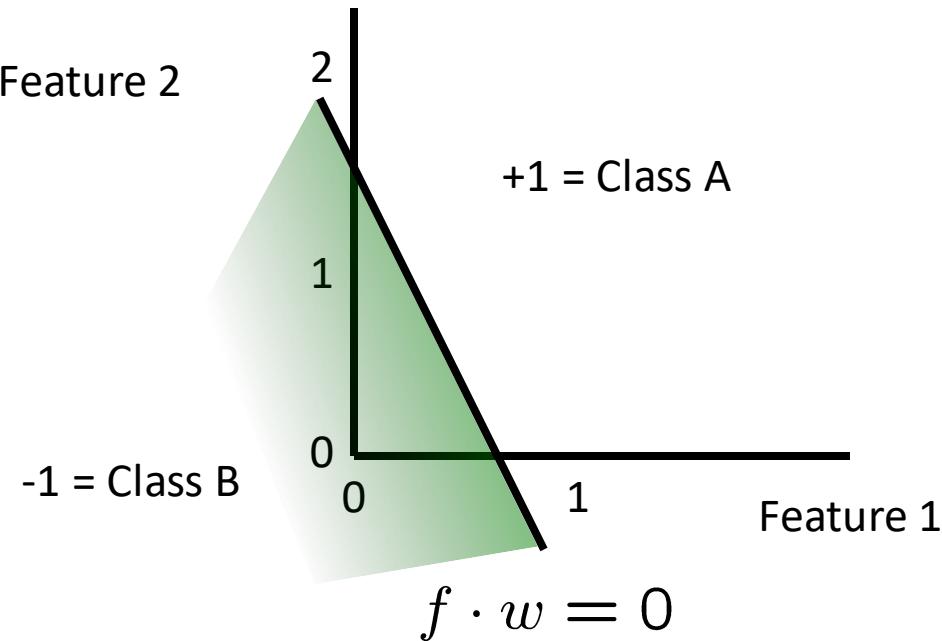
$$f(x_2)$$

```
# free      : 0  
YOUR_NAME  : 1  
MISSPELLED : 1  
FROM_FRIEND : 1
```

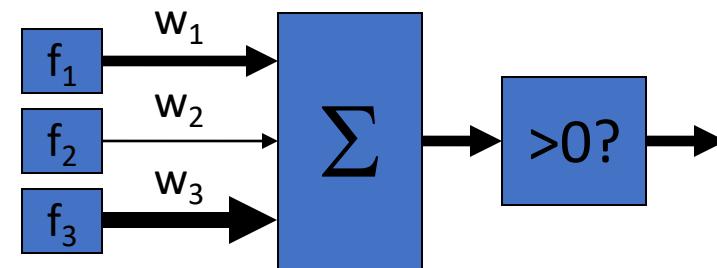


Classification with a Perceptron

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



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

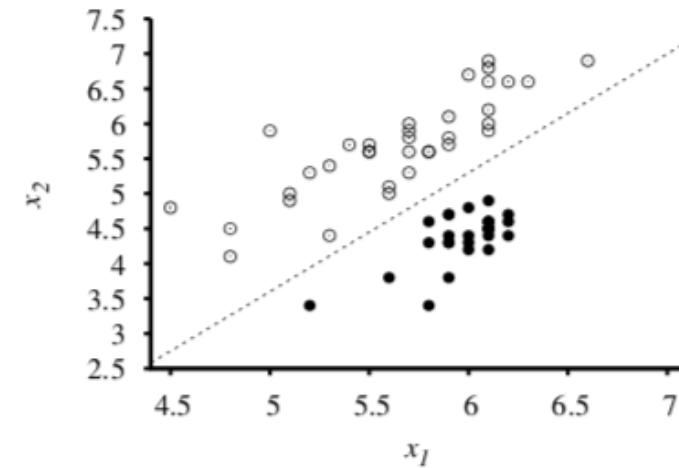


Perceptron

Decision rule (Binary case)

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

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



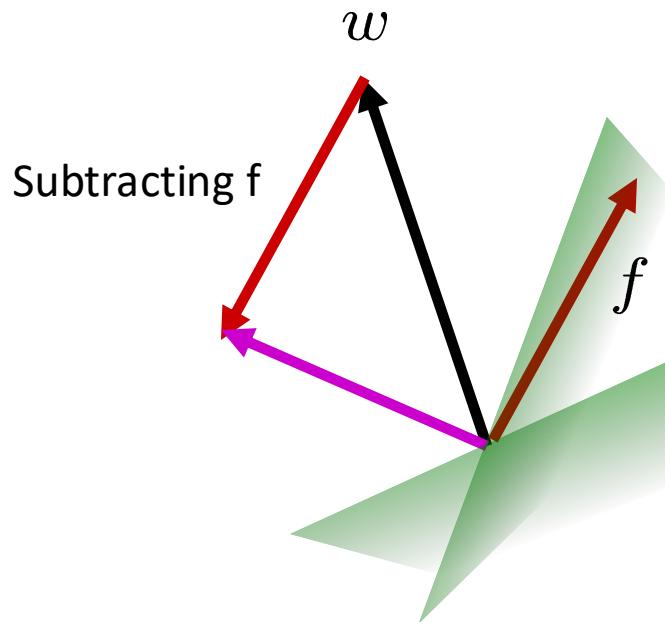
Binary classification task

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

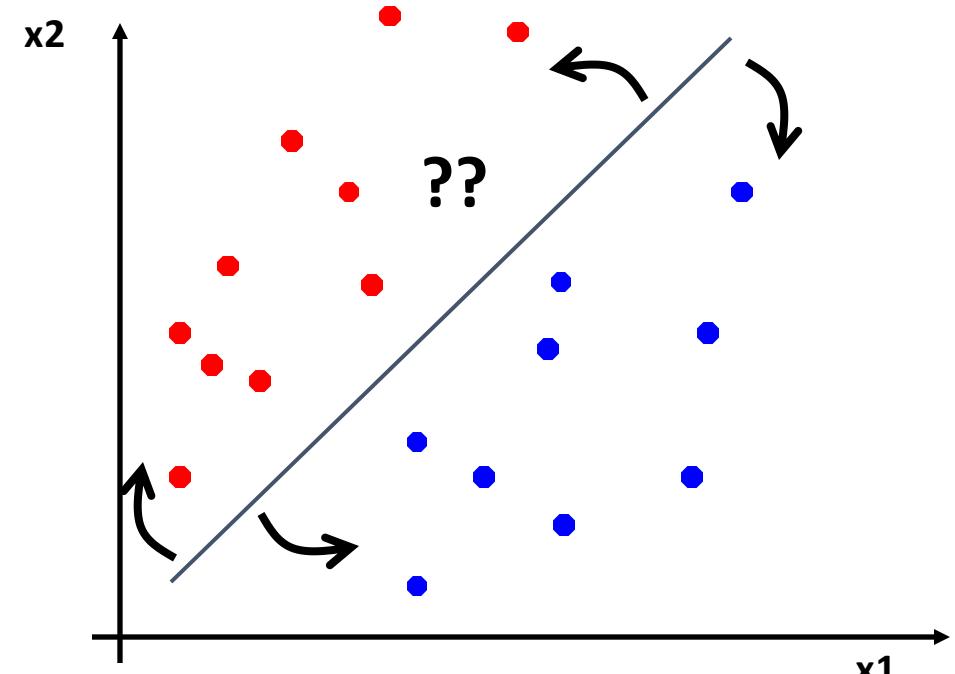
How to arrive at the separator?

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.



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

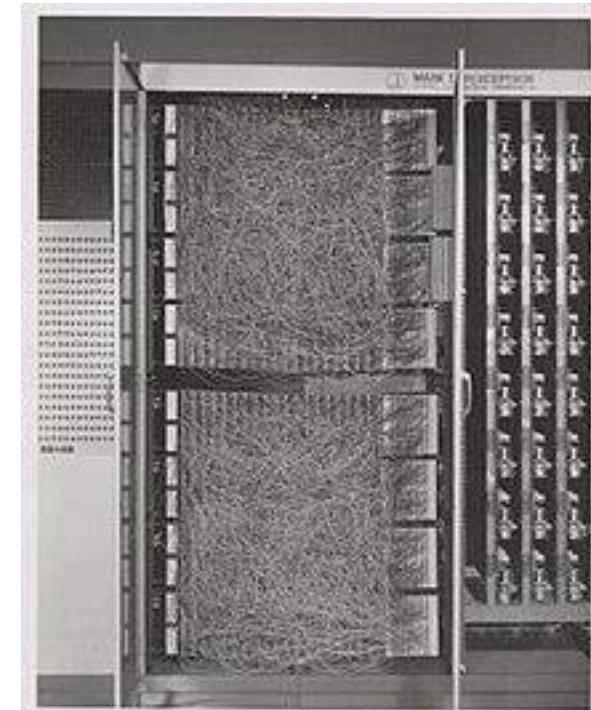
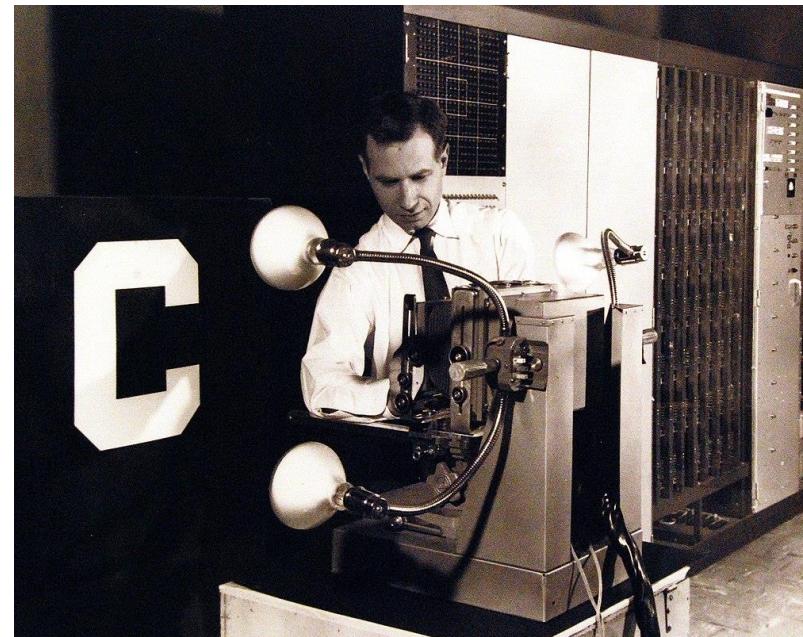
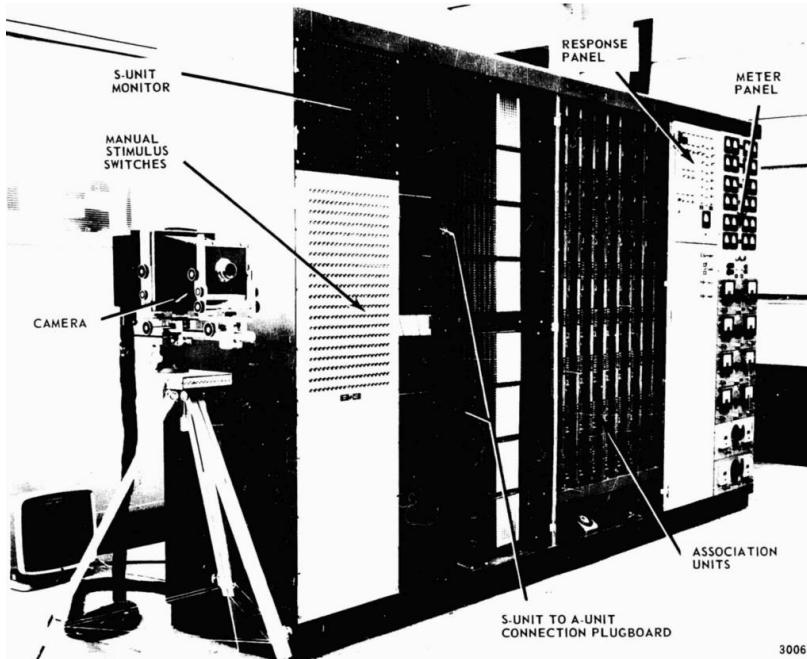


Binary classification task

Animation: <https://adrianstoll.com/post/perceptron-animation/>

Some History

A physical realization of the perceptron machine.



The perceptron: A probabilistic model for information storage and organization in the brain.

© Request Permissions

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6), 386–408. <https://doi.org/10.1037/h0042519>

To answer the questions of how information about the physical world is sensed, in what form is information remembered, and how does information retained in memory influence recognition and behavior, a theory is developed for a hypothetical nervous system called a perceptron. The theory serves as a bridge between biophysics and psychology. It is possible to predict learning curves from neurological variables and vice versa. The quantitative statistical approach is fruitful in the understanding of the organization of cognitive systems. 18 references. (APA PsycInfo Database Record (c) 2016 APA, all rights reserved)

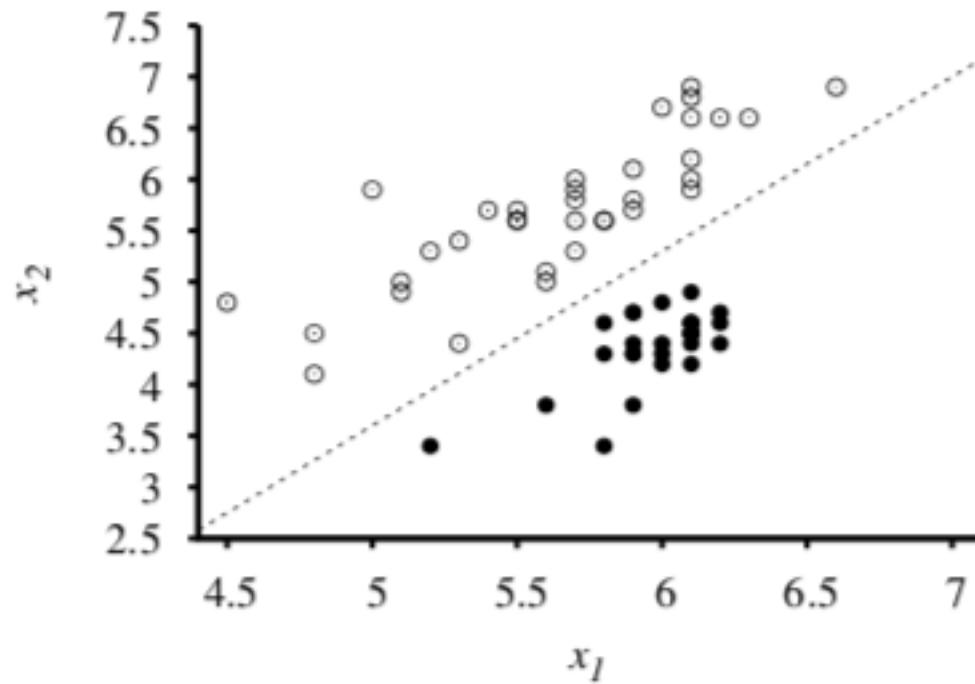
Copyright

Holder: American Psychological Association

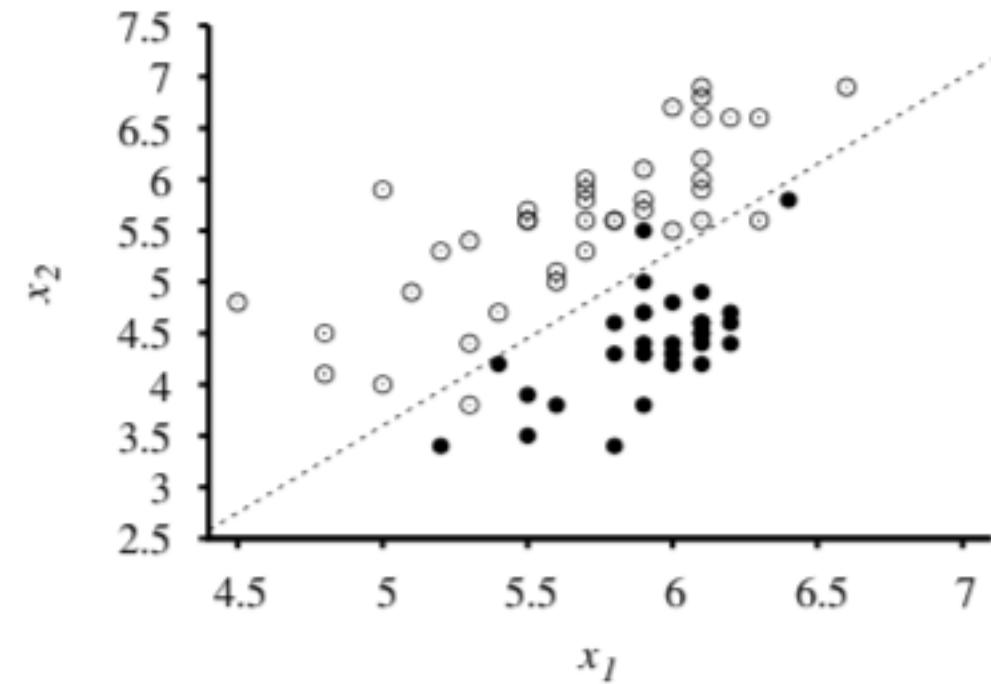
Year: 1958

Perceptron

Case: Linearly-separable

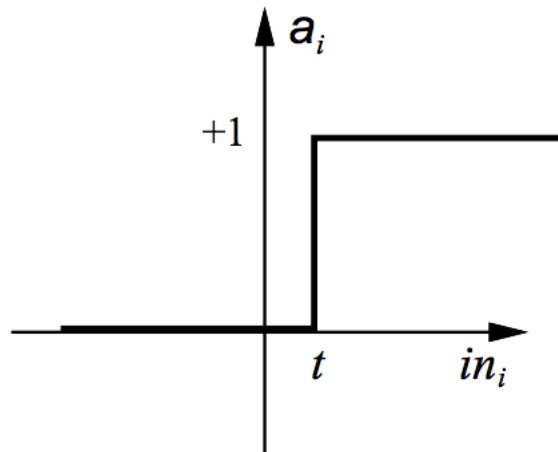


Case: Not Linearly-separable

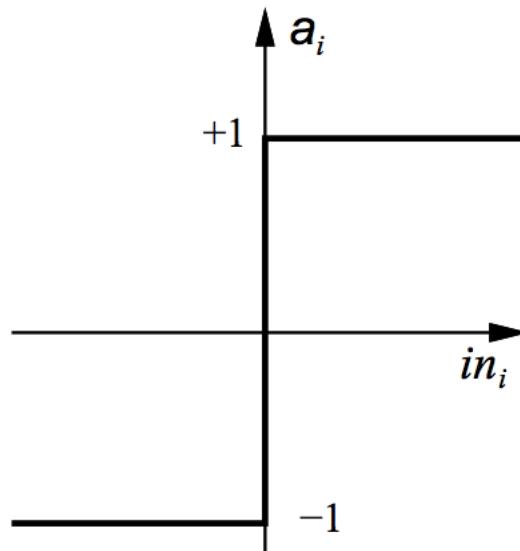


Perceptron learning rule converges to a perfect linear separator when the data points are linearly separable. Problem when there is non-separable data.

Threshold Functions



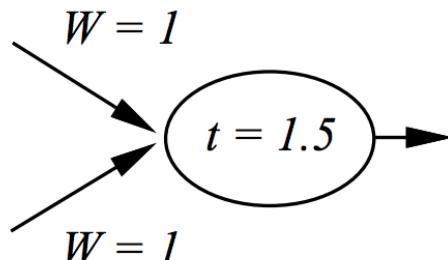
(a) Step function



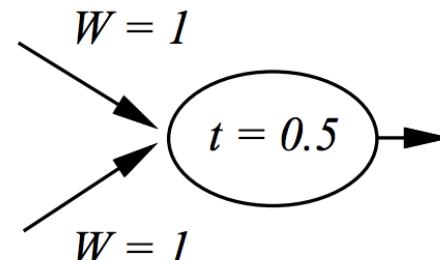
(b) Sign function

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

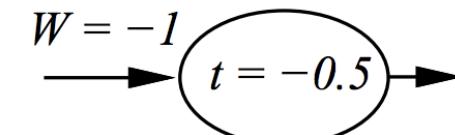
Boolean Functions and Perceptron



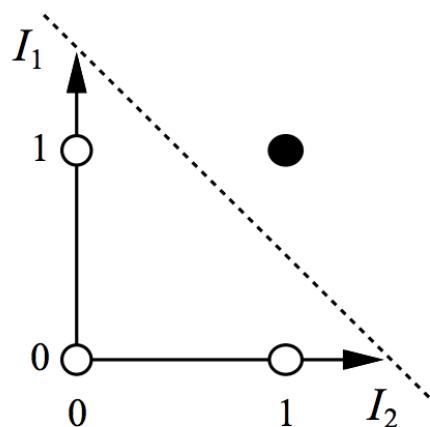
AND



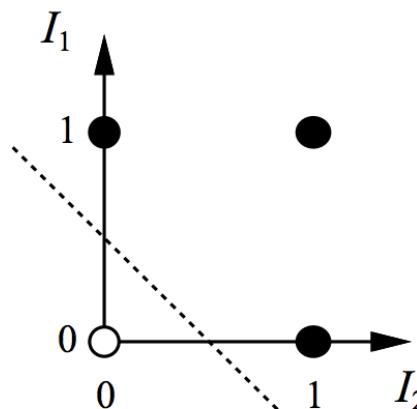
OR



NOT



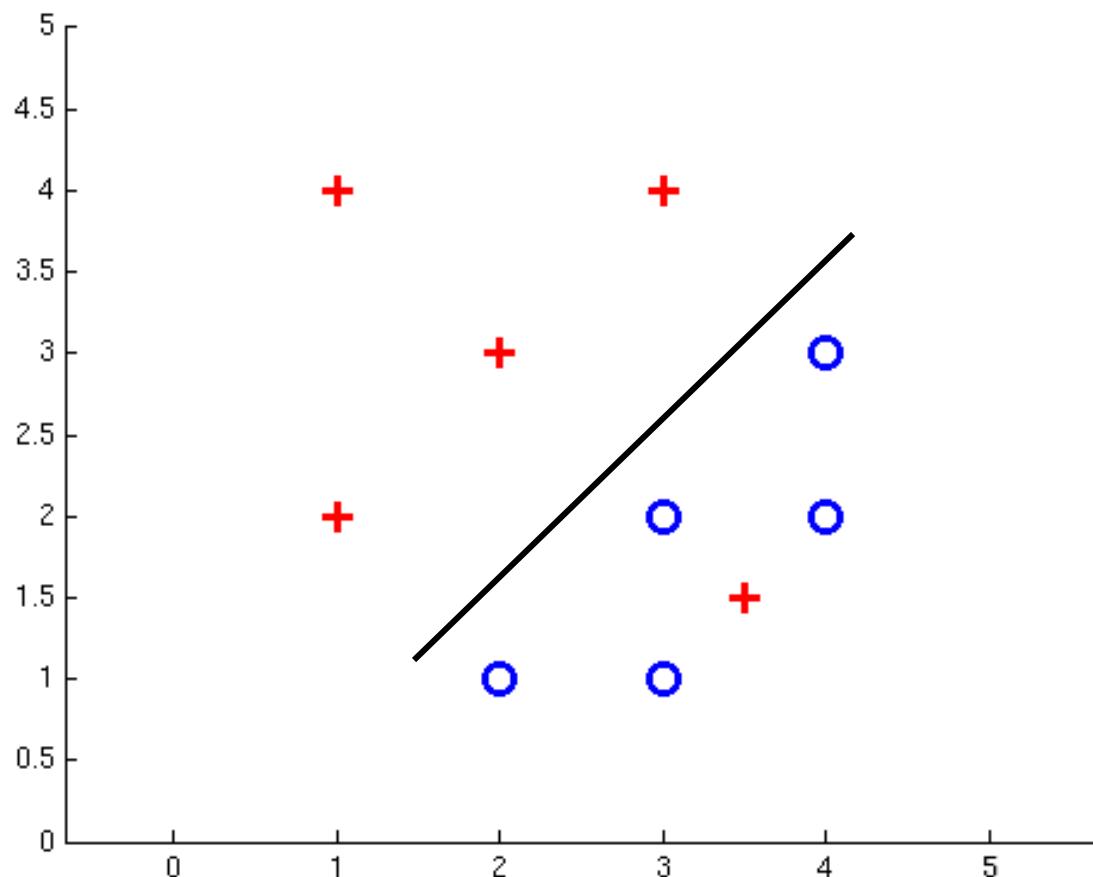
(a) I_1 and I_2



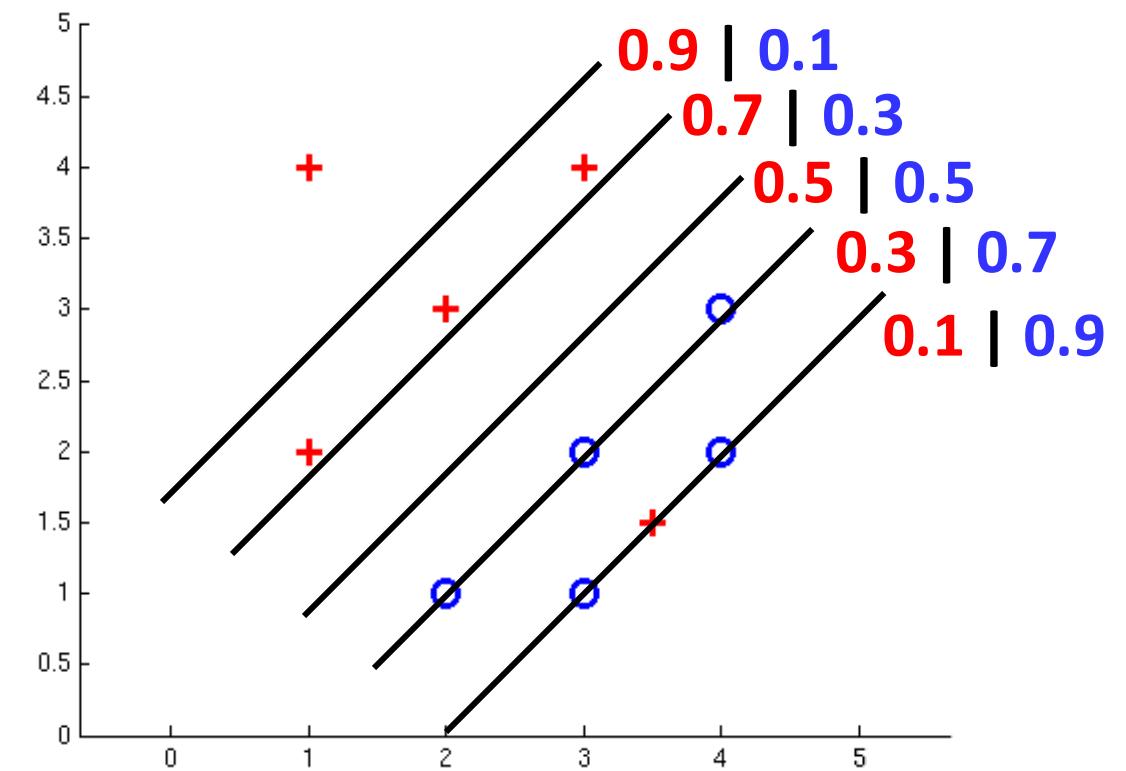
(b) I_1 or I_2

Non-separable case

Deterministic Decisions



Probabilistic Decisions



Logistic Output

- **Logistic Function**

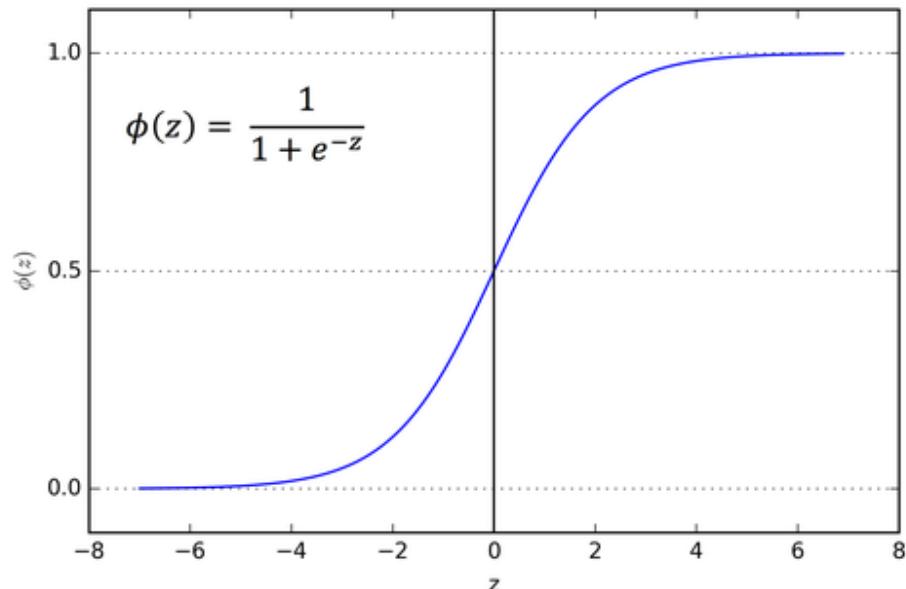
- Very positive values. Probability $\rightarrow 1$
- Very negative values. Probability $\rightarrow 0$.
- Makes the prediction. Converts to a probability
- Softens the decision boundary.

- **Logistic Regression**

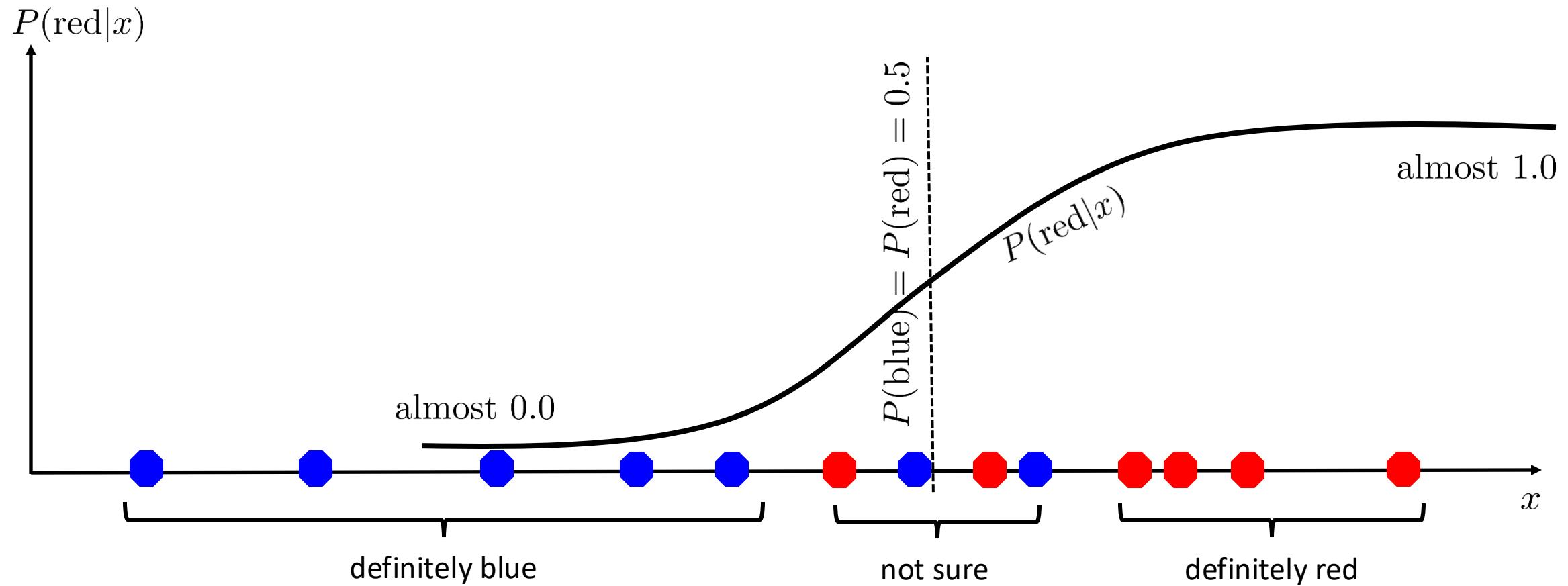
- Fitting the weights of this model to minimize loss on a data set is called logistic regression.

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

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



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

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

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

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

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.

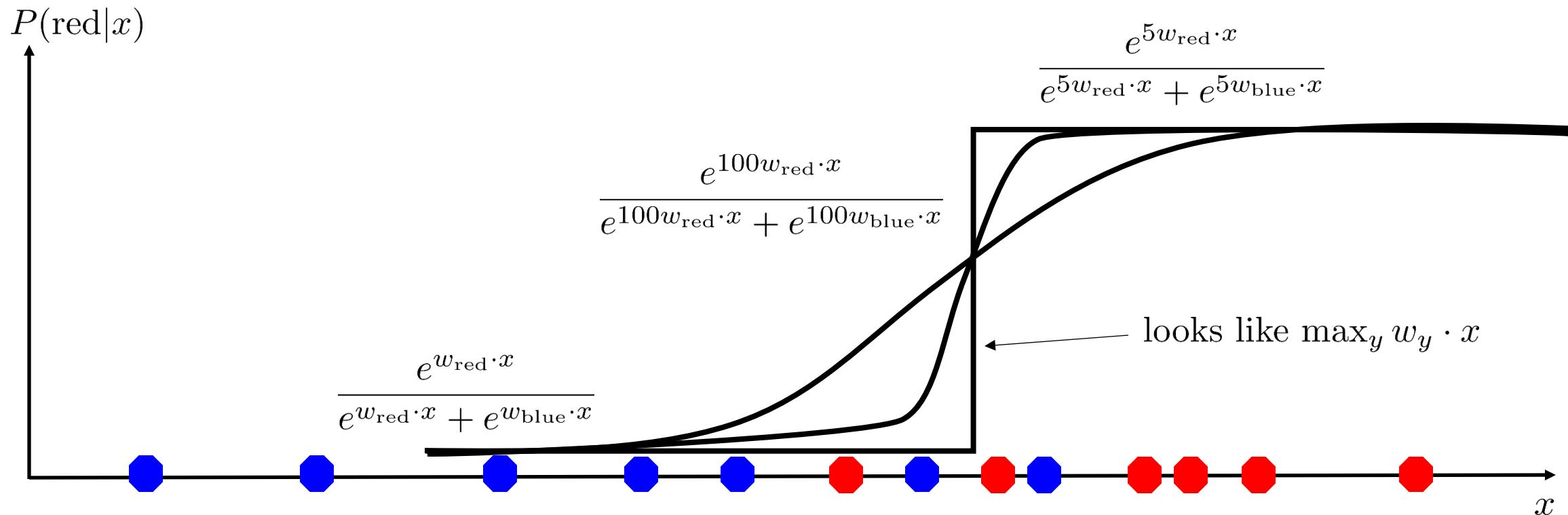
Prediction of the unnormalized probabilities.

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

Exponentiate and normalize the values.

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

Softmax Example



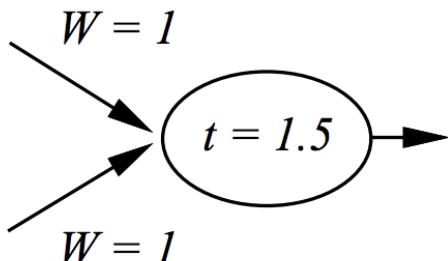
$$P(\text{red}|x) = \frac{e^{w_{\text{red}} \cdot x}}{e^{w_{\text{red}} \cdot x} + e^{w_{\text{blue}} \cdot x}}$$

Multi-class Setting

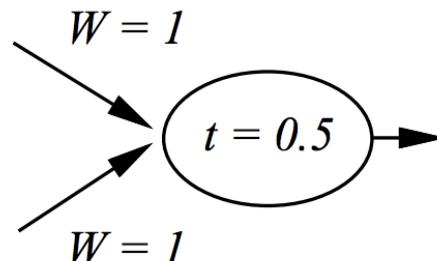
$$\max_w \text{ll}(w) = \max_w \sum_i \log P(y^{(i)} | x^{(i)}; w)$$

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

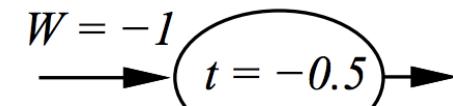
Can a perceptron learn XOR?



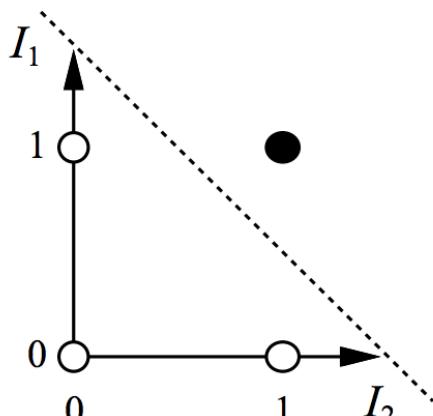
AND



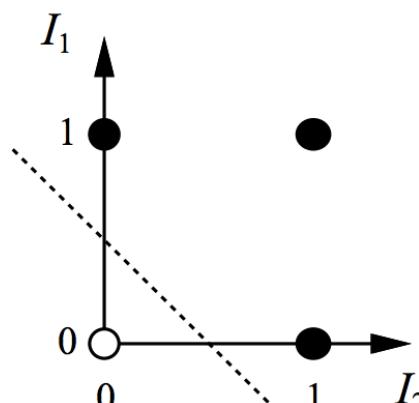
OR



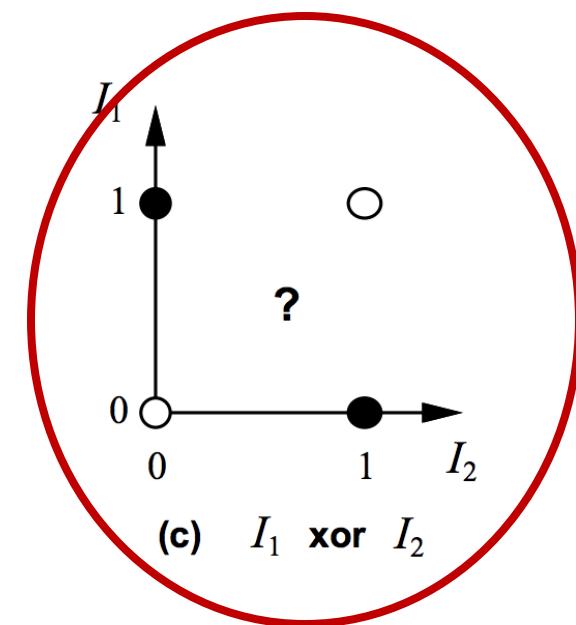
NOT



(a) I_1 and I_2



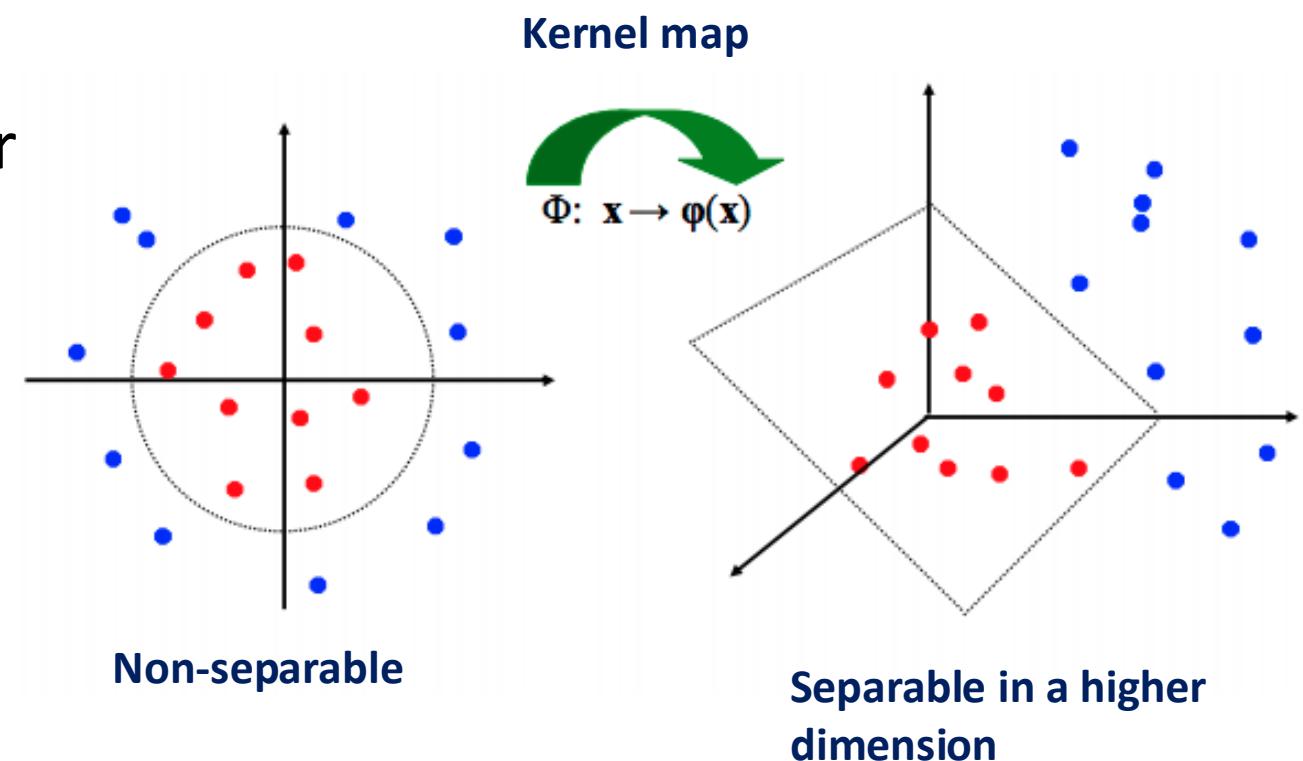
(b) I_1 or I_2



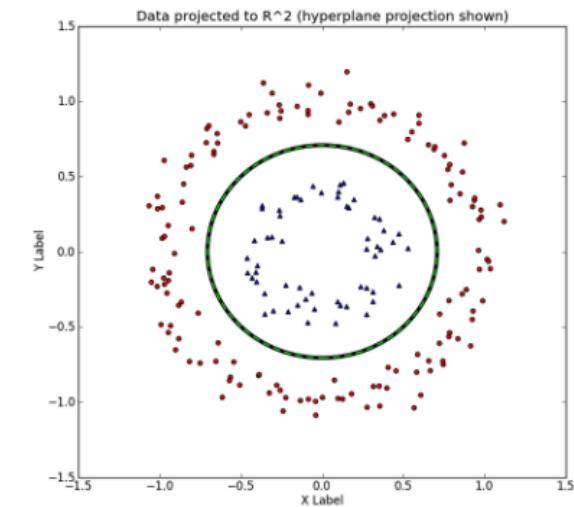
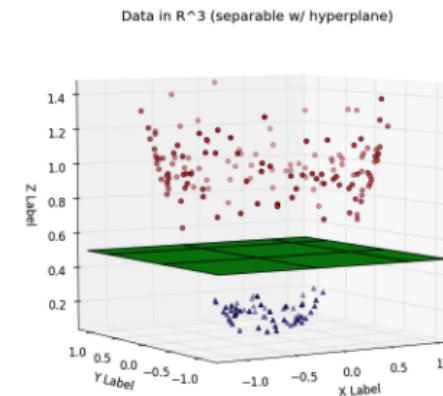
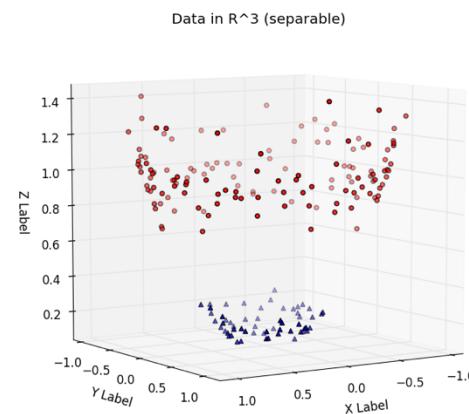
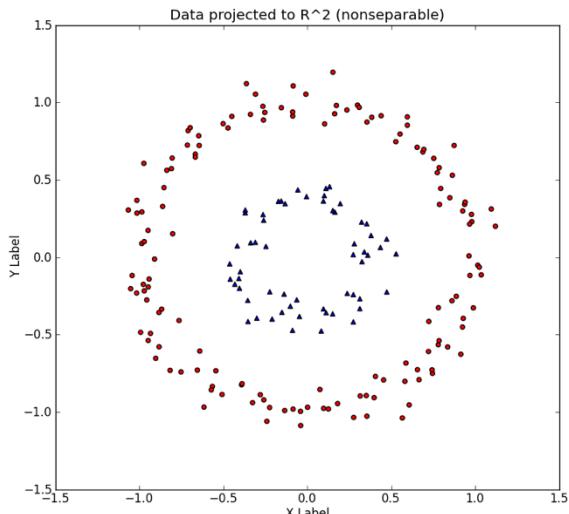
(c) I_1 xor I_2

Non-separability and Non-linear Functions

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



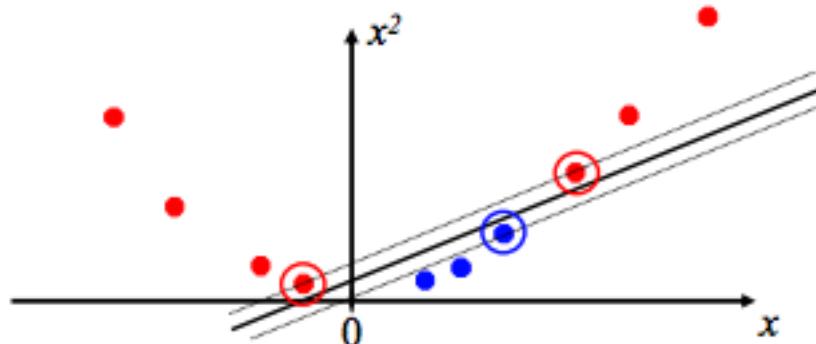
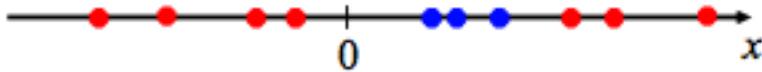
Example: Kernel Map



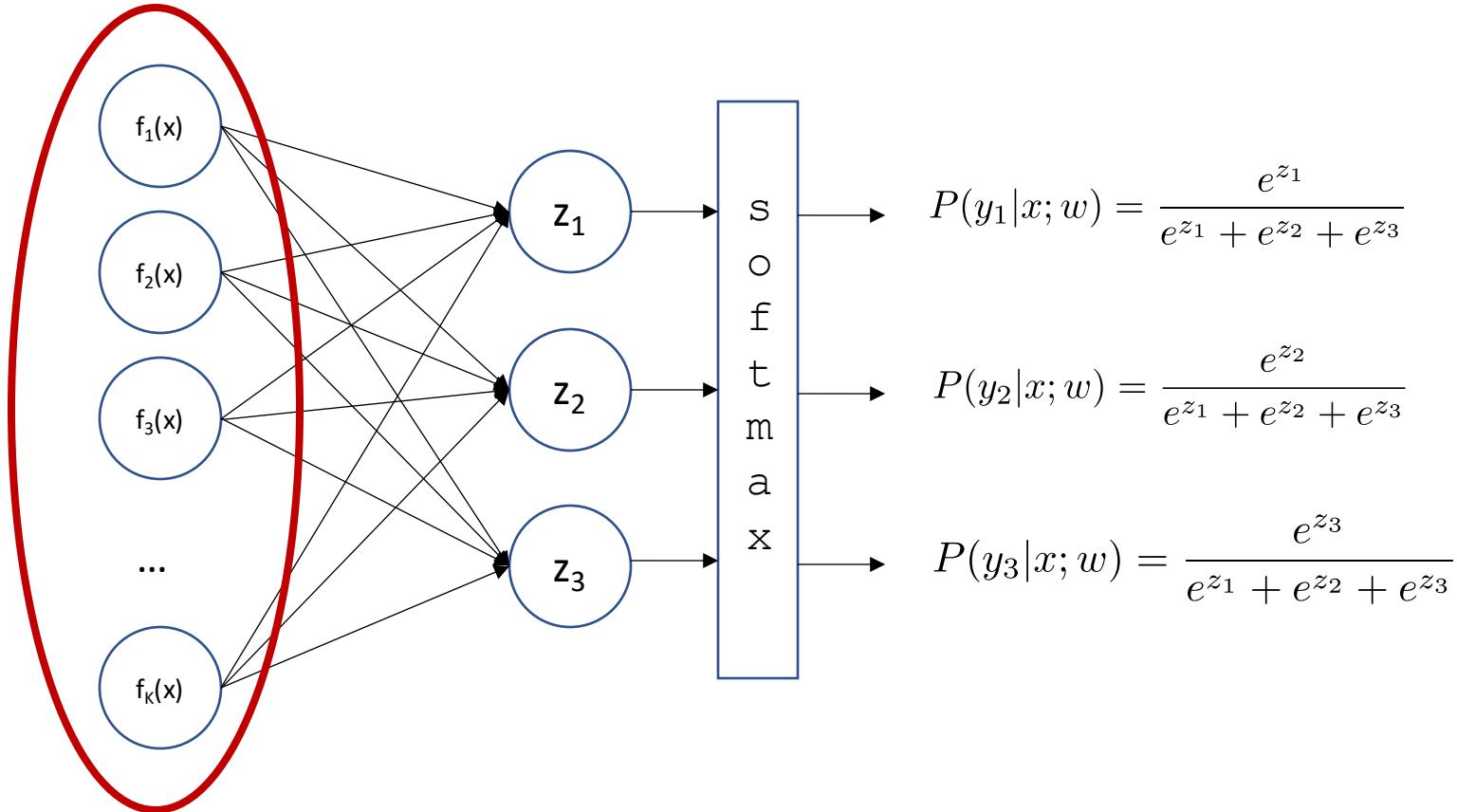
Left) A dataset in \mathbb{R}^2 , not linearly separable. (Right) The same dataset transformed by the transformation:

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

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.

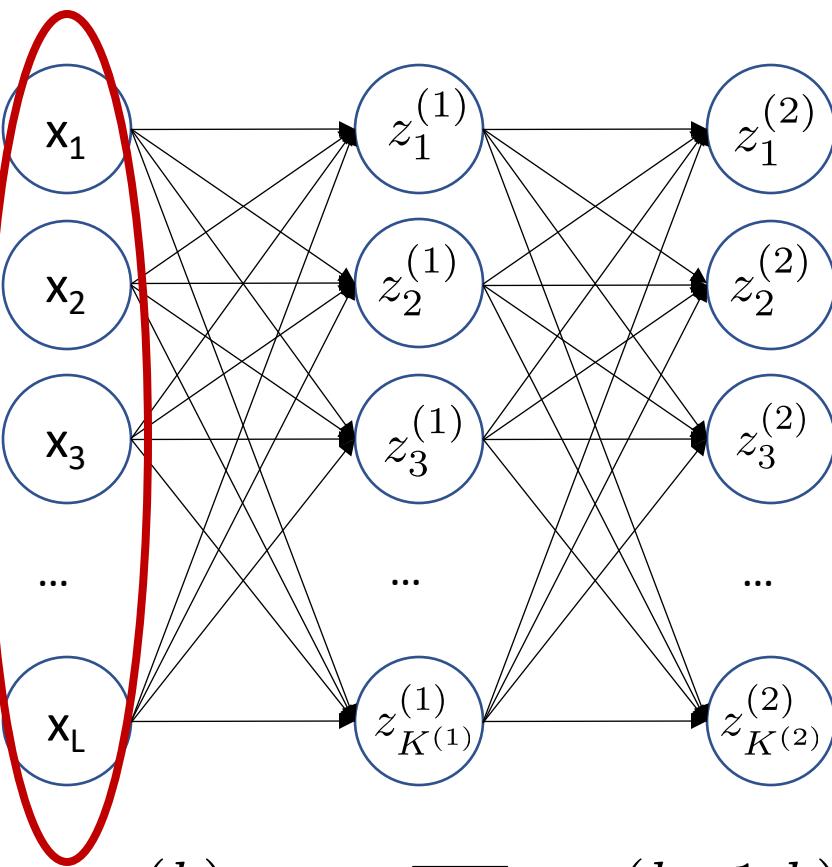


Feature extraction is inherent in neural networks



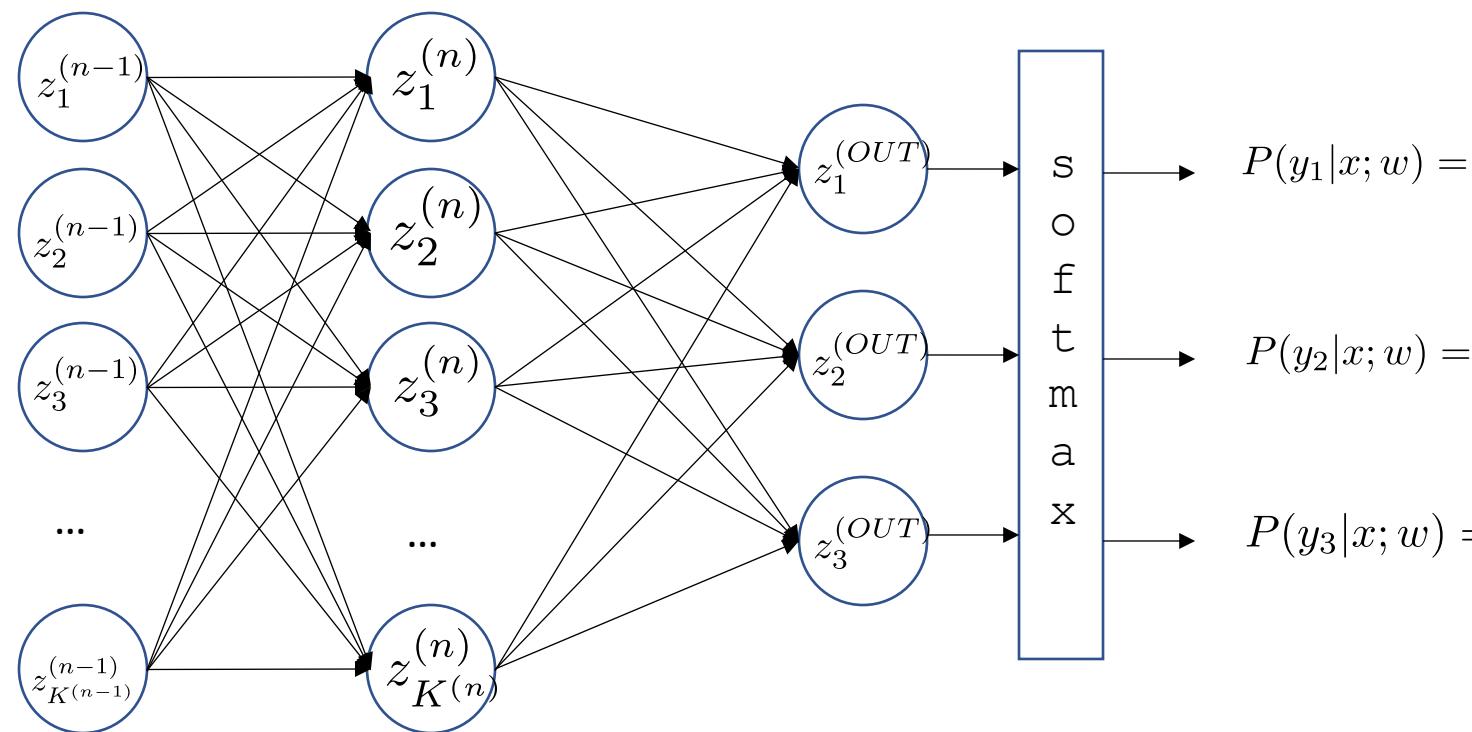
Still, we are designing these features. Can these be acquired in a data-driven manner? Can the parameters controlling these non-linear functions be learned?

Neural networks: learning the features



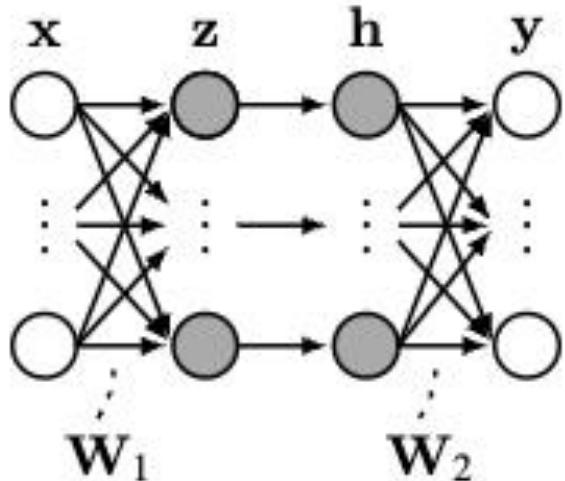
$$z_i^{(k)} = g\left(\sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)}\right)$$

g = nonlinear activation function



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

Multi-layer Perceptrons



$$\mathbf{z} = \mathbf{W}_1 \mathbf{x} + \mathbf{b}_1$$

/ linear layer

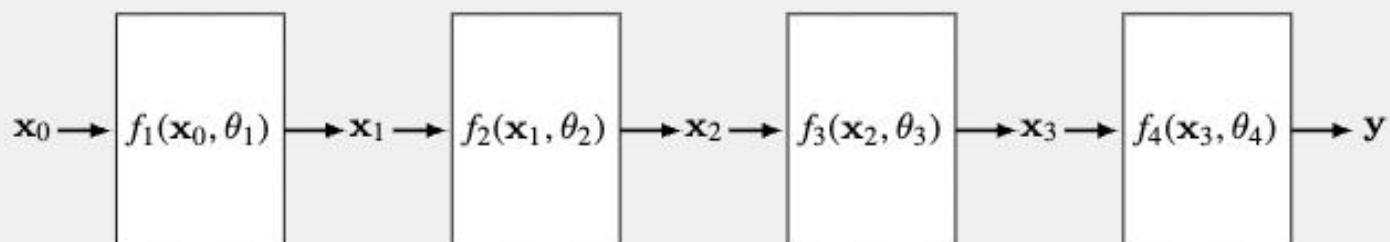
$$\mathbf{h} = g(\mathbf{z})$$

/ activation function

$$\mathbf{y} = \mathbf{W}_2 \mathbf{h} + \mathbf{b}_2$$

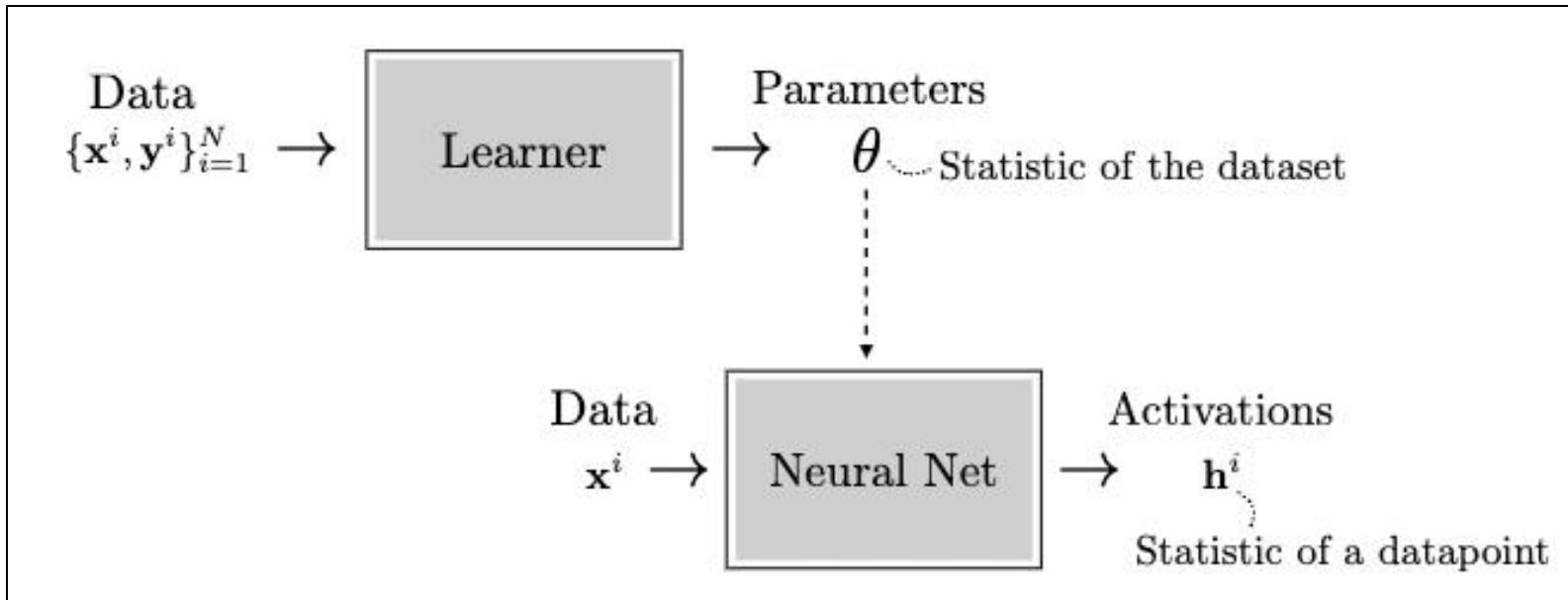
/ linear layer

A multilayer network is a sequence of transformations f_1, \dots, f_L that produce a series of activations $\mathbf{x}_1, \dots, \mathbf{x}_L$:



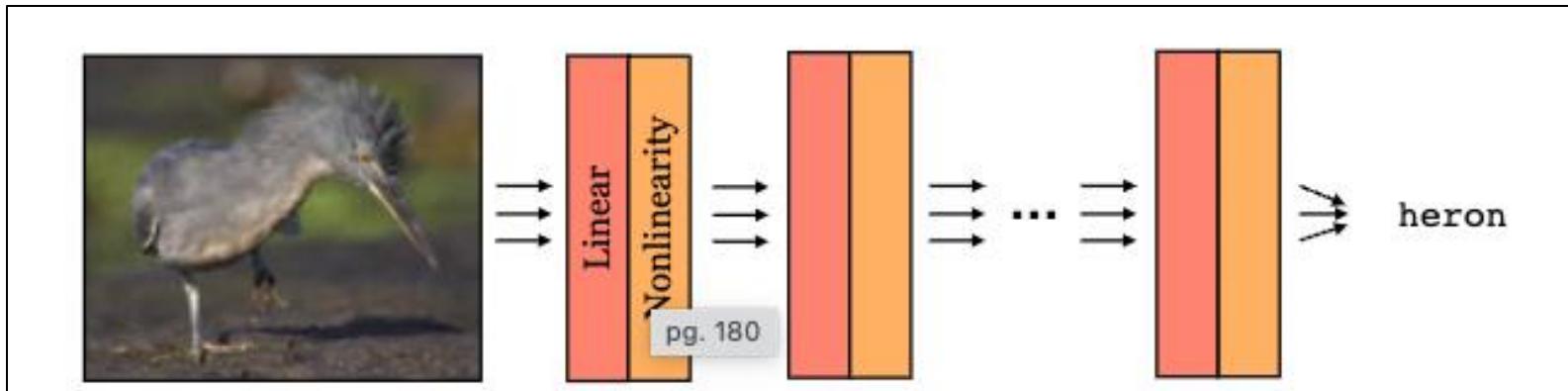
$$\mathbf{x}_{l+1} = f_{l+1}(\mathbf{x}_l, \theta_{l+1})$$

Neural Networks: *In essence*



During training one learns the parameters of the network.

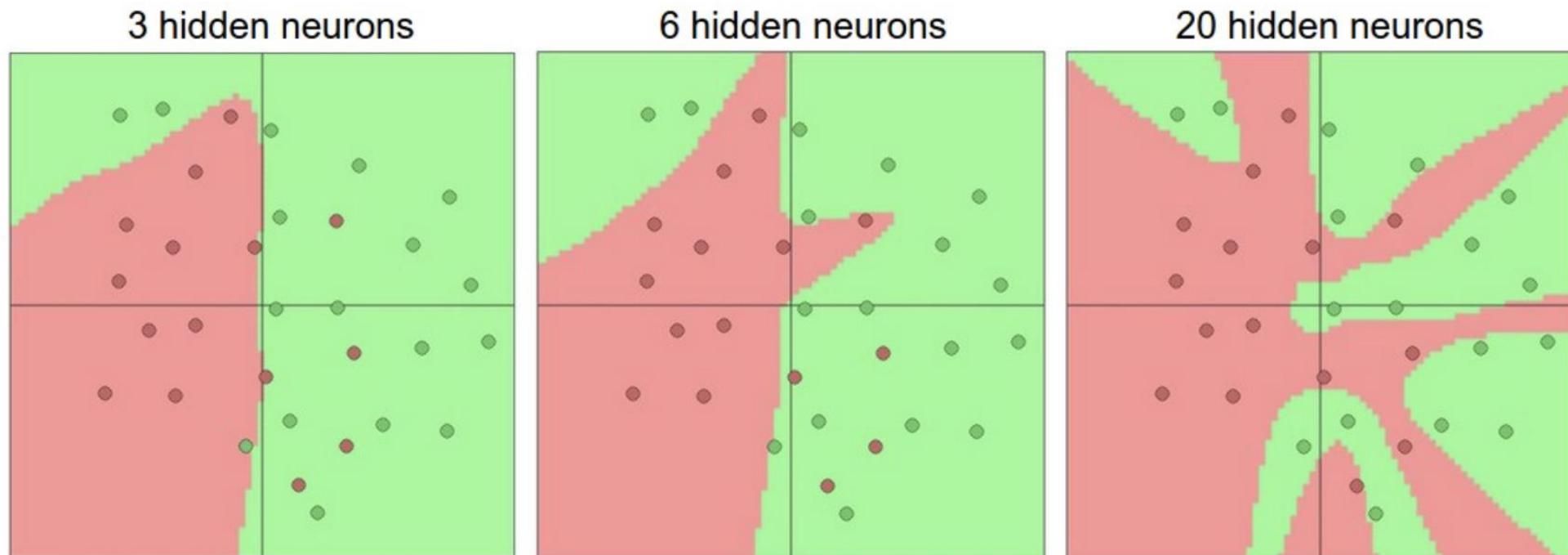
At inference time, directly apply the weights to form the predication.



Deep nets consist of linear layers interleaved with nonlinearities.

Representation of complex functions

Key success of neural networks is in learning complex functions. The ability to learn complex functions are crucial for performing complex classification, regression and other machine learning tasks.



<https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle®Dataset=reg-plane&learningRate=0.03®ularizationRate=0&noise=0&networkShape=4,2&seed=0.67214&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false>

Deep Neural Networks

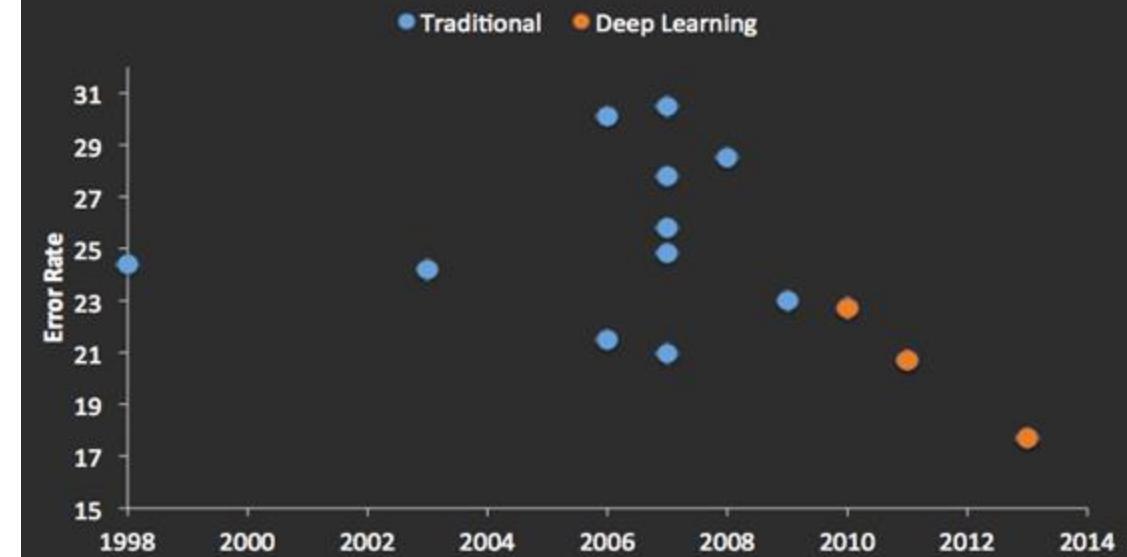
- Last layer
 - Logistic regression
- Several Hidden Layers
 - Computing the features. The features are learned rather than hand-designed.
- Universal function approximation theorem
 - If neural net is large enough
 - Then neural net can represent any continuous mapping from input to output with arbitrary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

Neural Networks: Successes

ImageNet Error Rate 2010-2014



TIMIT Speech Recognition



Learning XOR

XOR is not linearly separable.

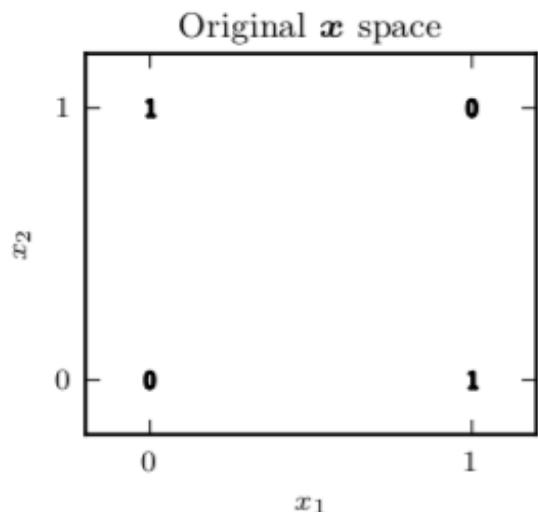


Figure 6.1, left

Rectified Linear Activation

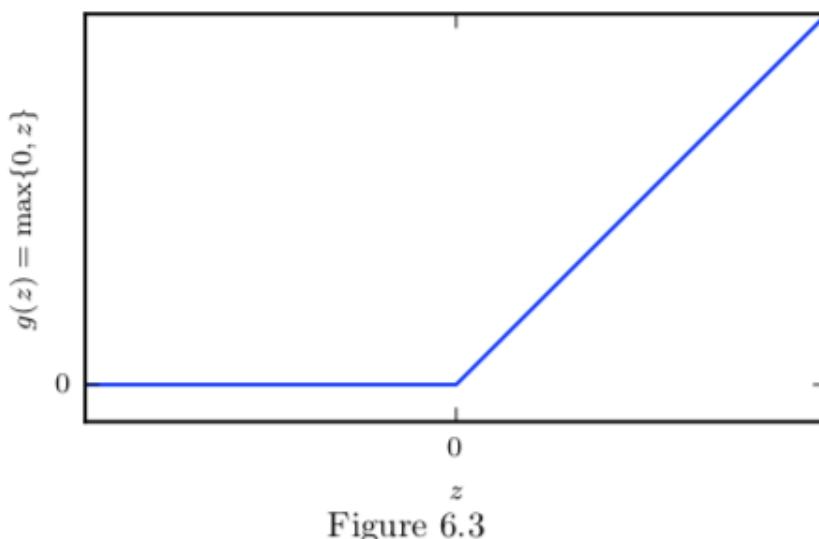


Figure 6.3

Network Diagram

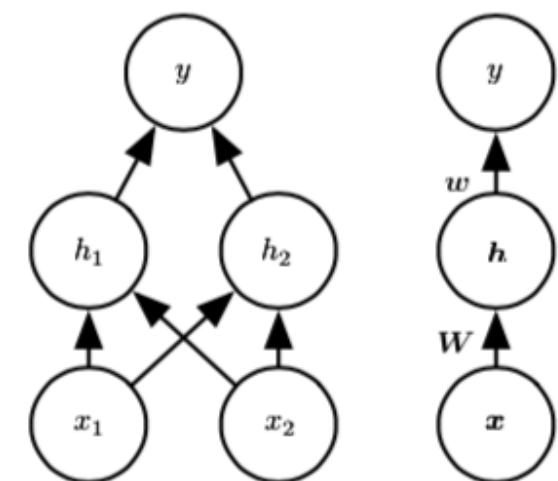


Figure 6.2

More compact representation

Example from Ch 6, DL Book

Learning XOR

Network Diagram

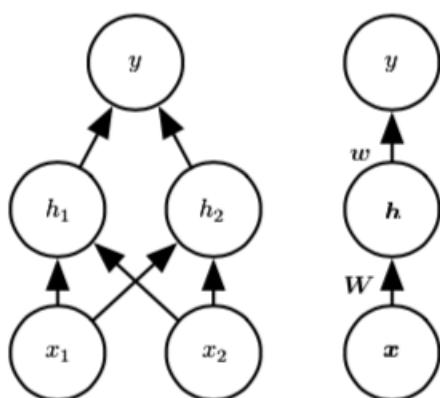


Figure 6.2

Model

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

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

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

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

XOR is separable in the transformed space

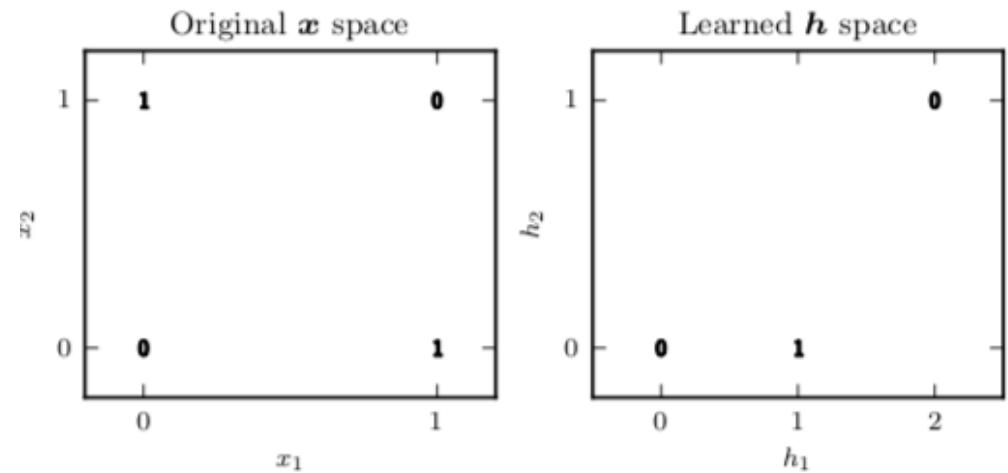


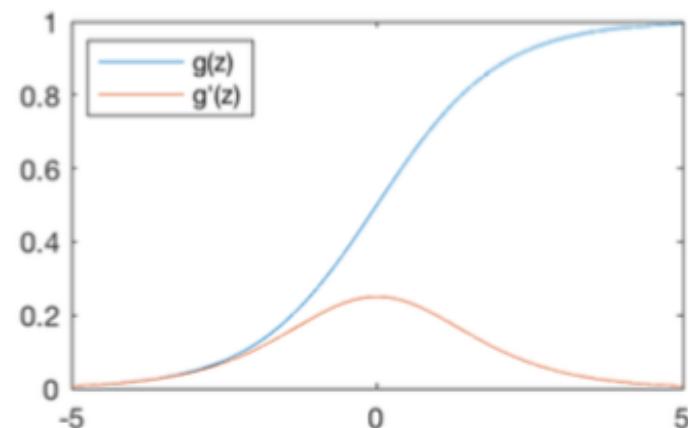
Figure 6.1

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

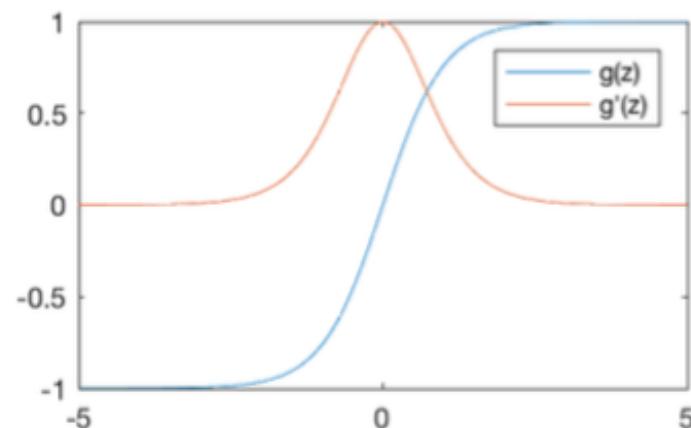
Sigmoid Function



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

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

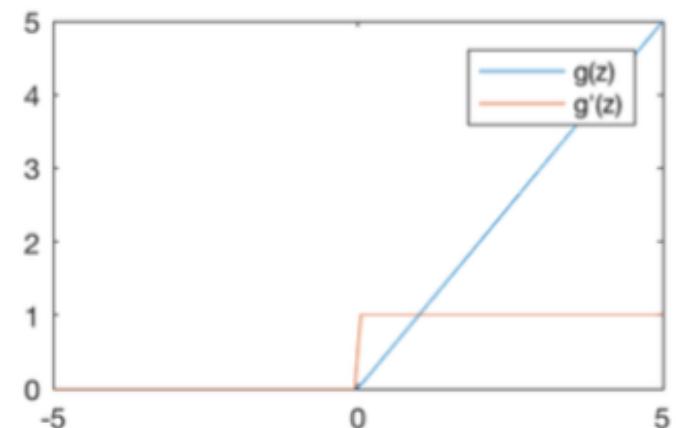
Hyperbolic Tangent



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

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

Rectified Linear Unit (ReLU)



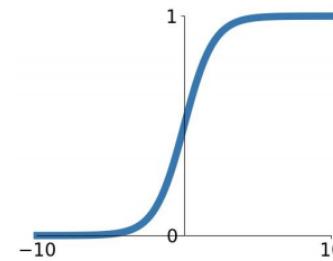
$$g(z) = \max(0, z)$$

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

Activation Functions

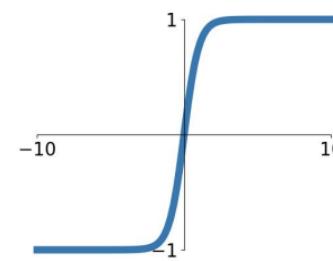
Sigmoid

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



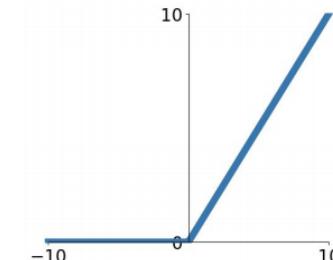
tanh

$$\tanh(x)$$



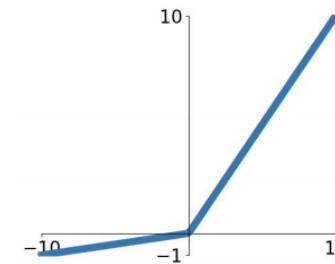
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

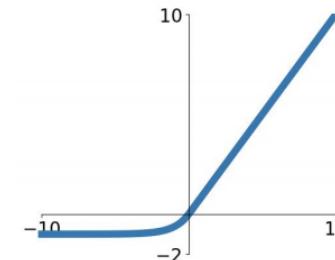


Maxout

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

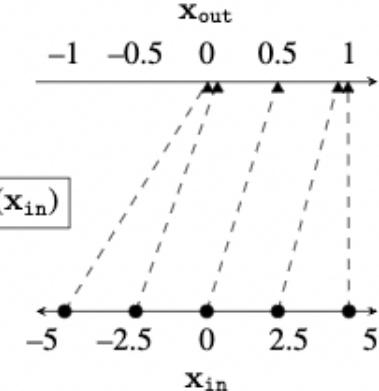
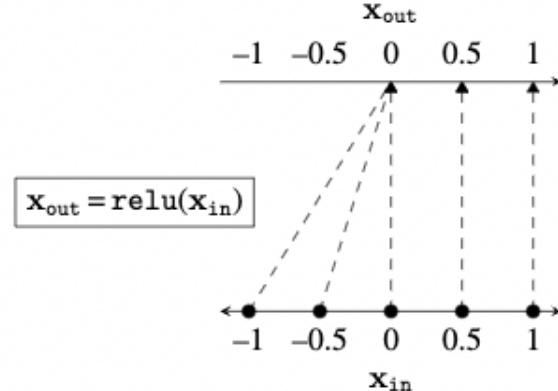
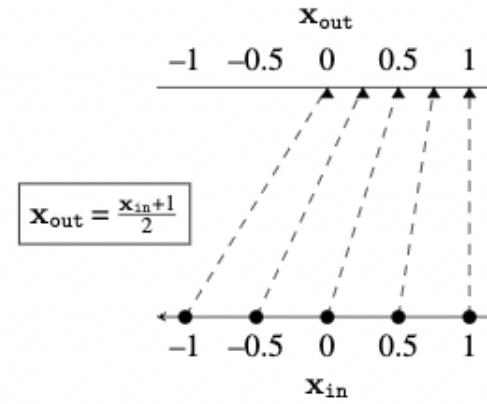
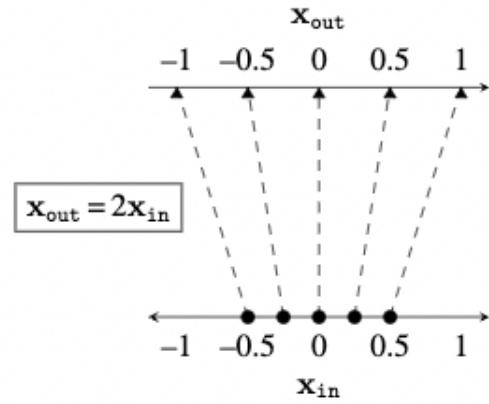
ELU

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

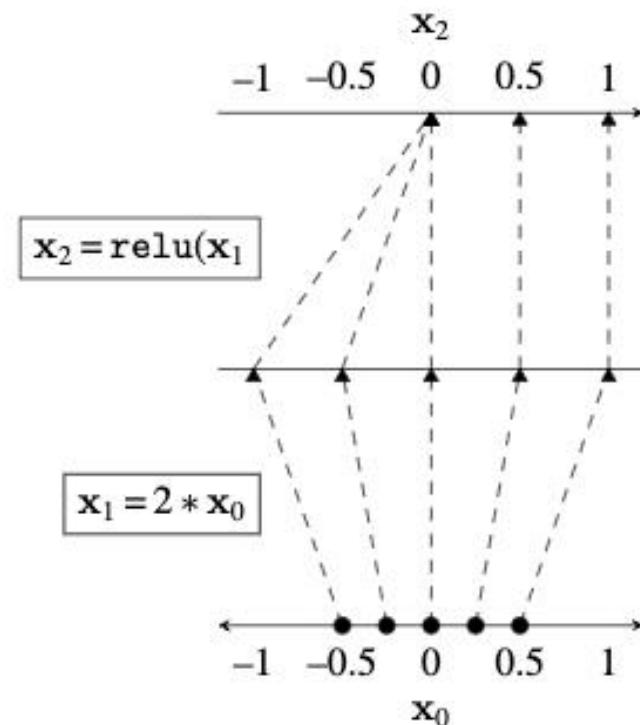


Insight: Neural Networks are Distribution Transformers

Mapping plots for simple functions that could be neural layers

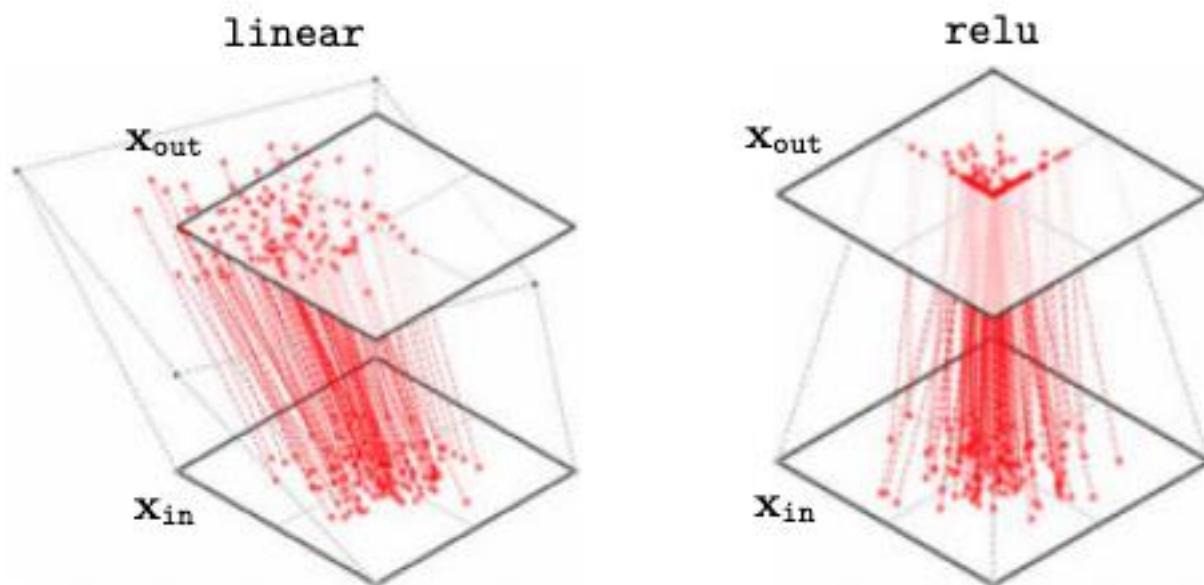


Mapping plot for a linear-relu stack



Mapping Diagrams for a data distribution

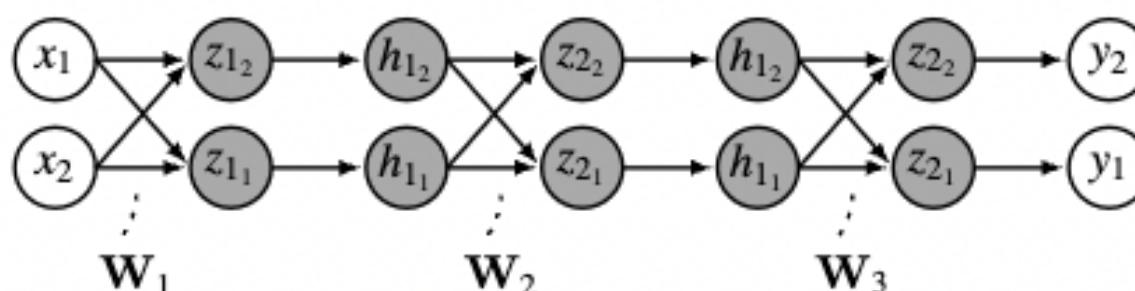
2D mapping diagram for several neural layers.



The linear layer mapping with shift, stretch and rotate depending on its weights and biases. The ReLU will map many points to the axes of the positive quadrant. Density will build up along the axes.

Binary Classification Example

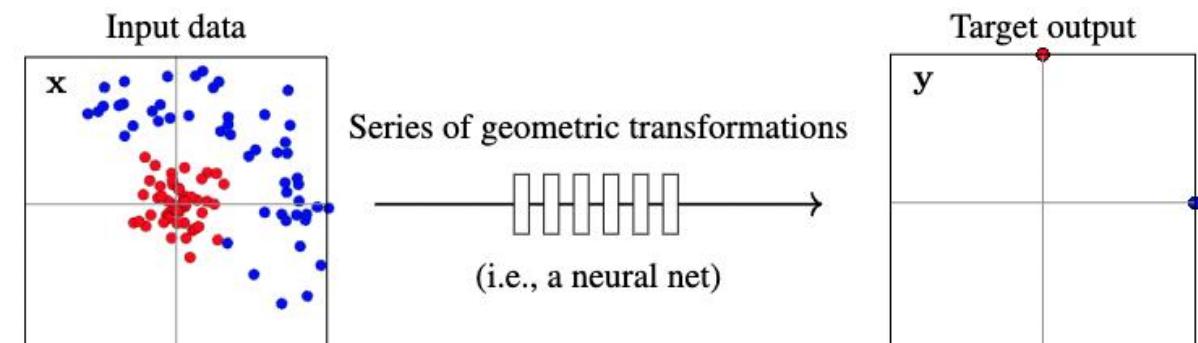
An MLP with three linear layers and two outputs, suitable for performing binary softmax regression.



Linear-relu-linear-relu-linear architecture

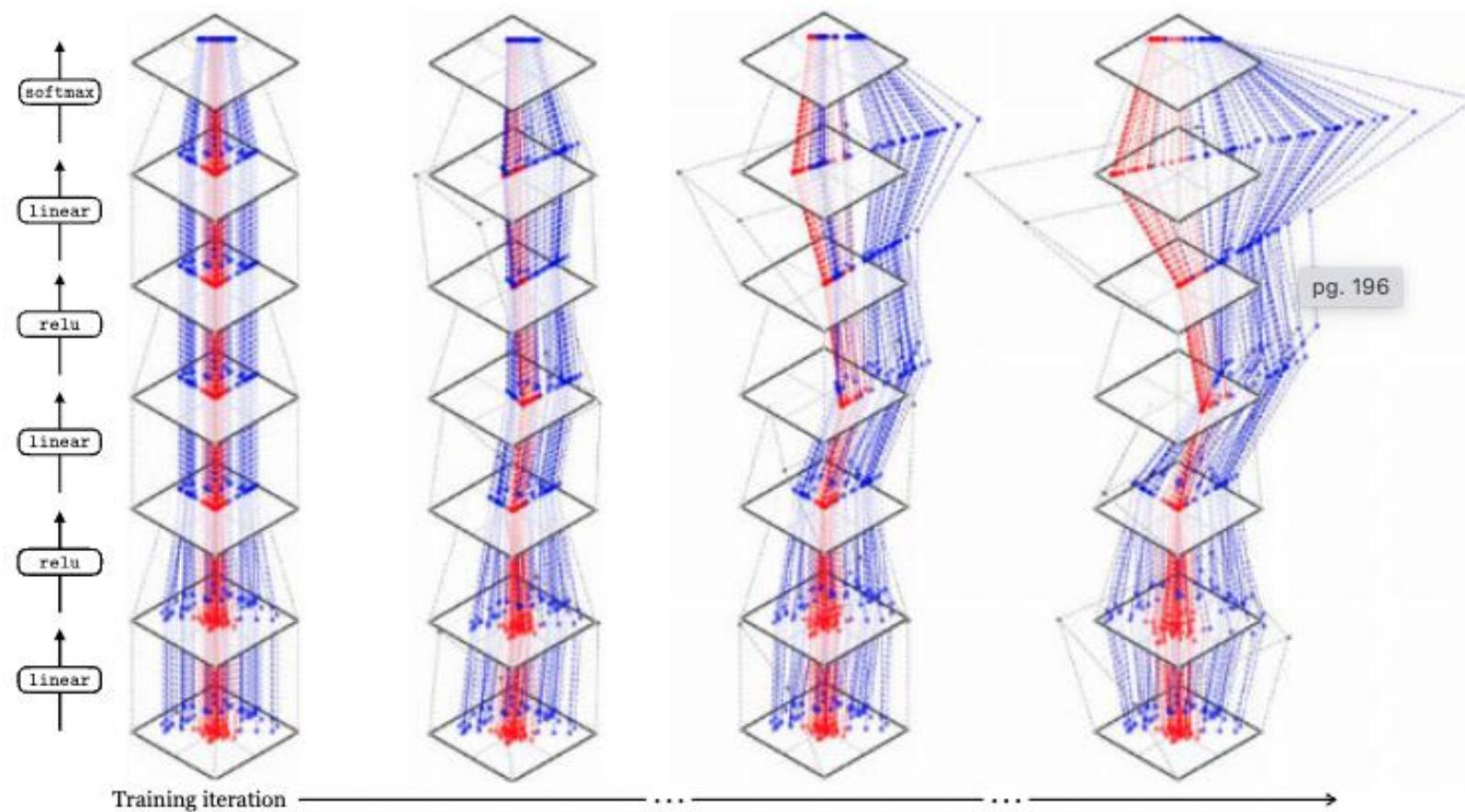
$$\begin{aligned} z_1 &= \mathbf{W}_1 \mathbf{x} + \mathbf{b}_1 && / \text{ linear} \\ h_1 &= \text{relu}(z_1) && / \text{ relu} \\ z_2 &= \mathbf{W}_2 h_1 + \mathbf{b}_2 && / \text{ linear} \\ h_2 &= \text{relu}(z_2) && / \text{ relu} \\ z_3 &= \mathbf{W}_3 h_2 + \mathbf{b}_3 && / \text{ linear} \\ y &= \text{softmax}(z_3) && / \text{ softmax} \end{aligned}$$

Goal of neural network classifier is to arrange the input data distribution to match the target label distribution.



Left shows original data distribution. Right shows the target output (one hot codes)

Remapping input data layer by layer



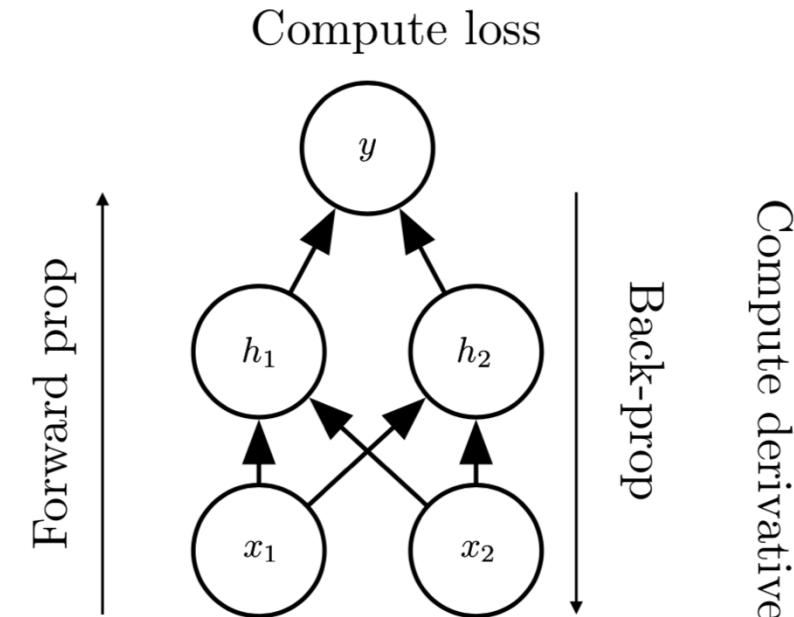
Target output moves the red and blue points closer to the ground truth data.
As training progresses the network gradually achieves this separation.

Backpropagation and Computation Graphs

- Backpropagation
 - In a NN, need a way to optimize the output loss with respect to the inputs.
 - Core question: assess how does the output change if the input changes by a certain amount.
 - Apply the chain rule to obtain the gradient.
- *How to manage this computation efficiently?*

<https://xnought.github.io/backprop-explainer/>

Compute activations



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

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

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.

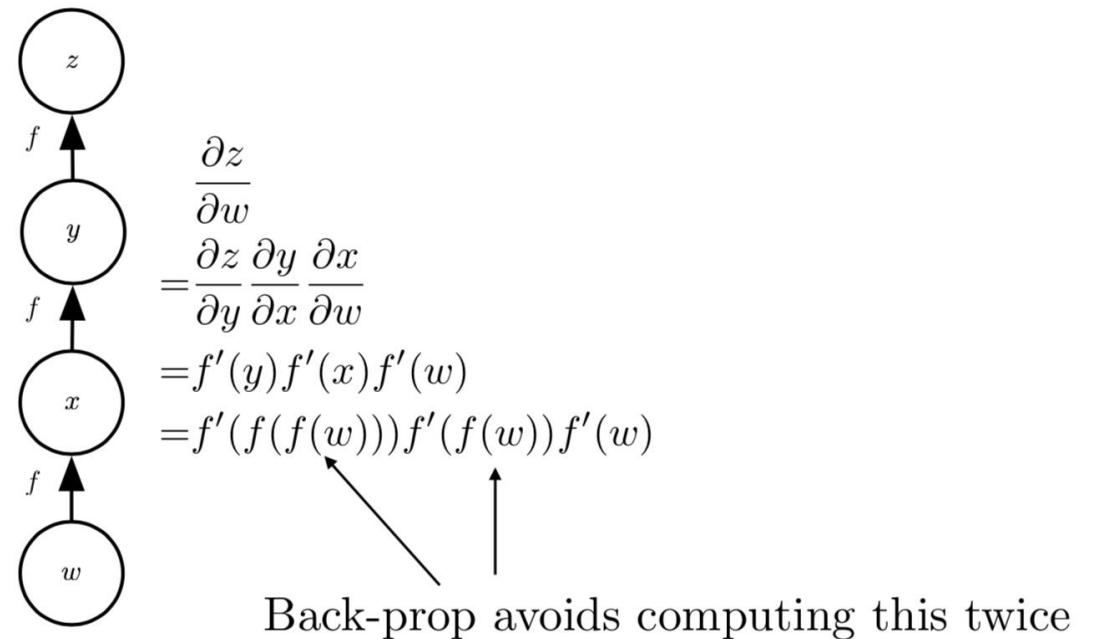


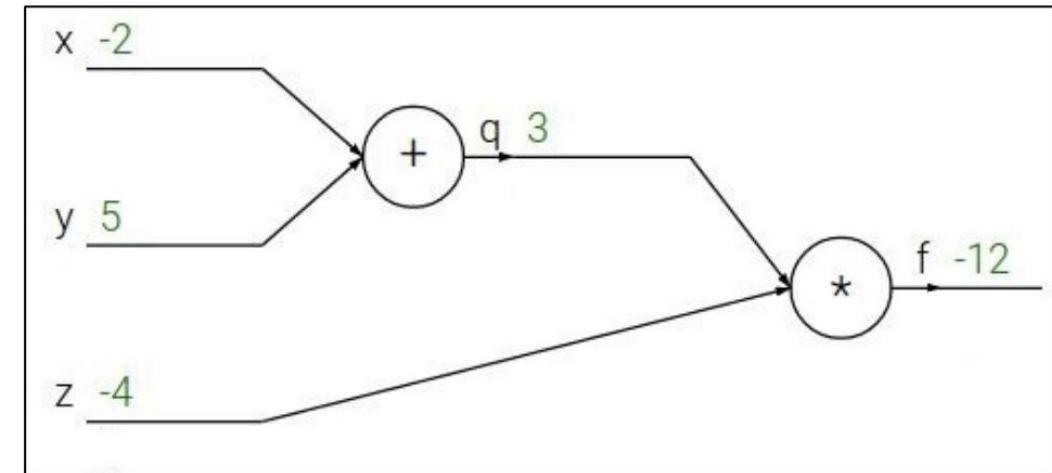
Figure 6.9

Backpropagation: Toy Example

Backpropagation: a simple example

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

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



Backpropagation: Toy Example

Backpropagation: a simple example

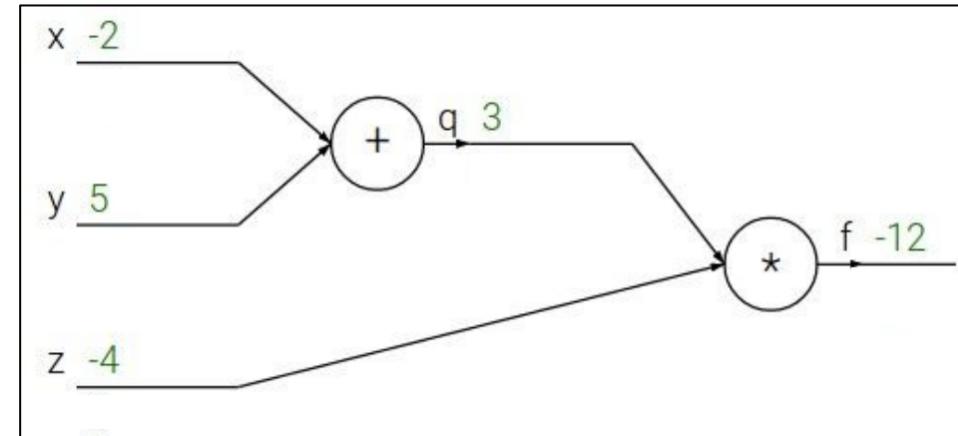
$$f(x, y, z) = (x + y)z$$

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

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

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

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



Backpropagation: Toy Example

Backpropagation: a simple example

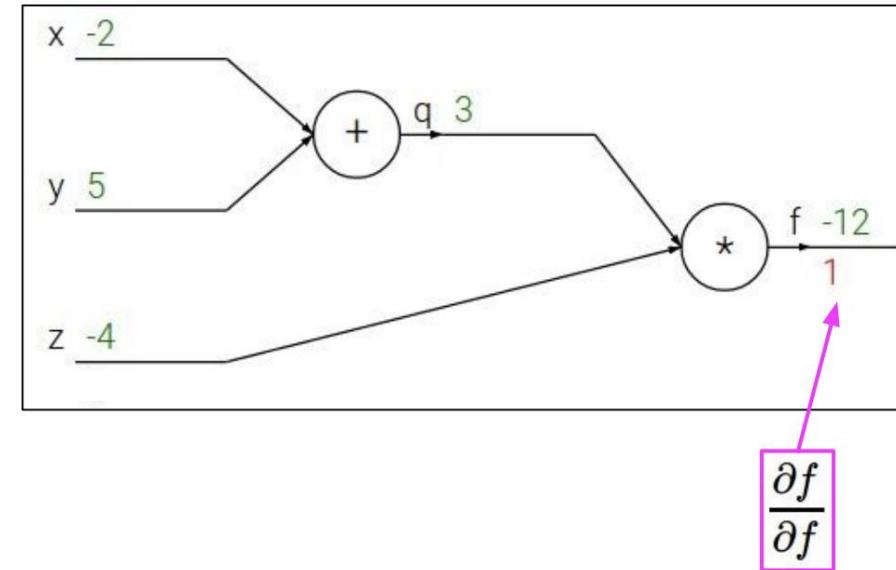
$$f(x, y, z) = (x + y)z$$

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

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

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

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



Once this gradient is computed, the downstream gradients can "share" this computation.

Backpropagation: Toy Example

Backpropagation: a simple example

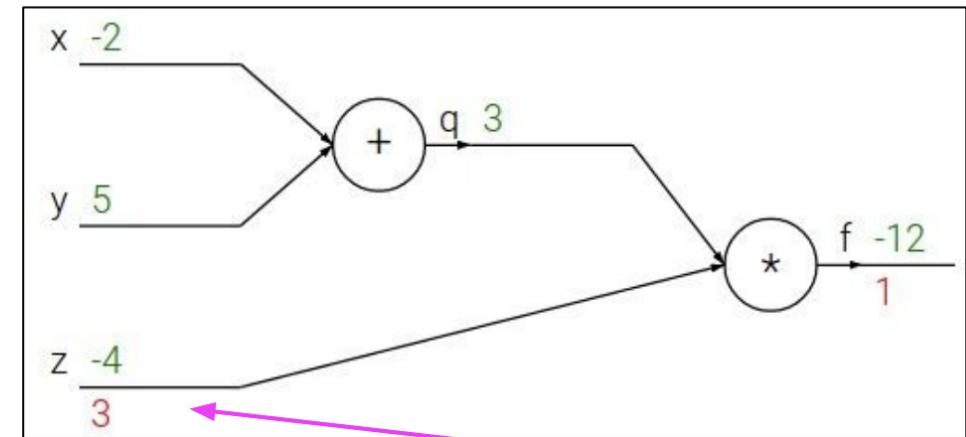
$$f(x, y, z) = (x + y)z$$

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

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

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

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

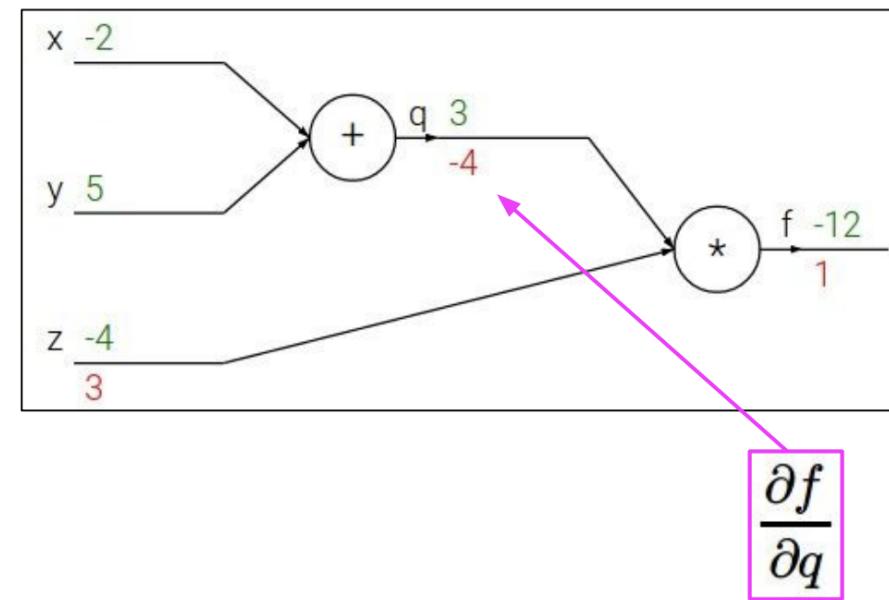
$$f(x, y, z) = (x + y)z$$

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

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

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

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



asdf

Backpropagation: Toy Example

Backpropagation: a simple example

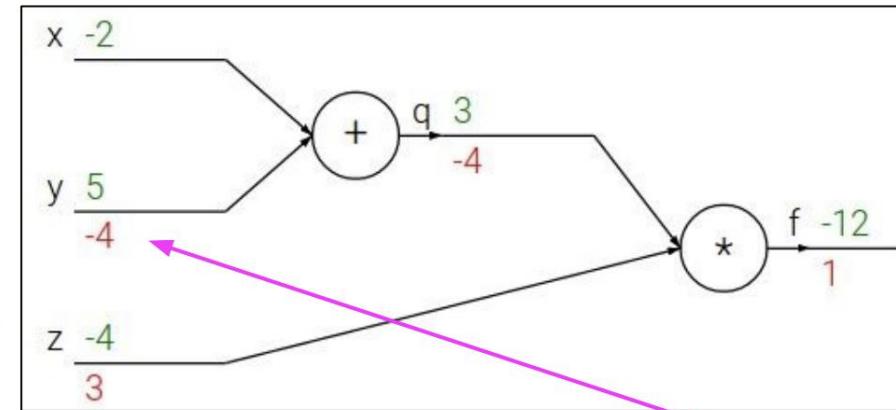
$$f(x, y, z) = (x + y)z$$

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

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

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

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



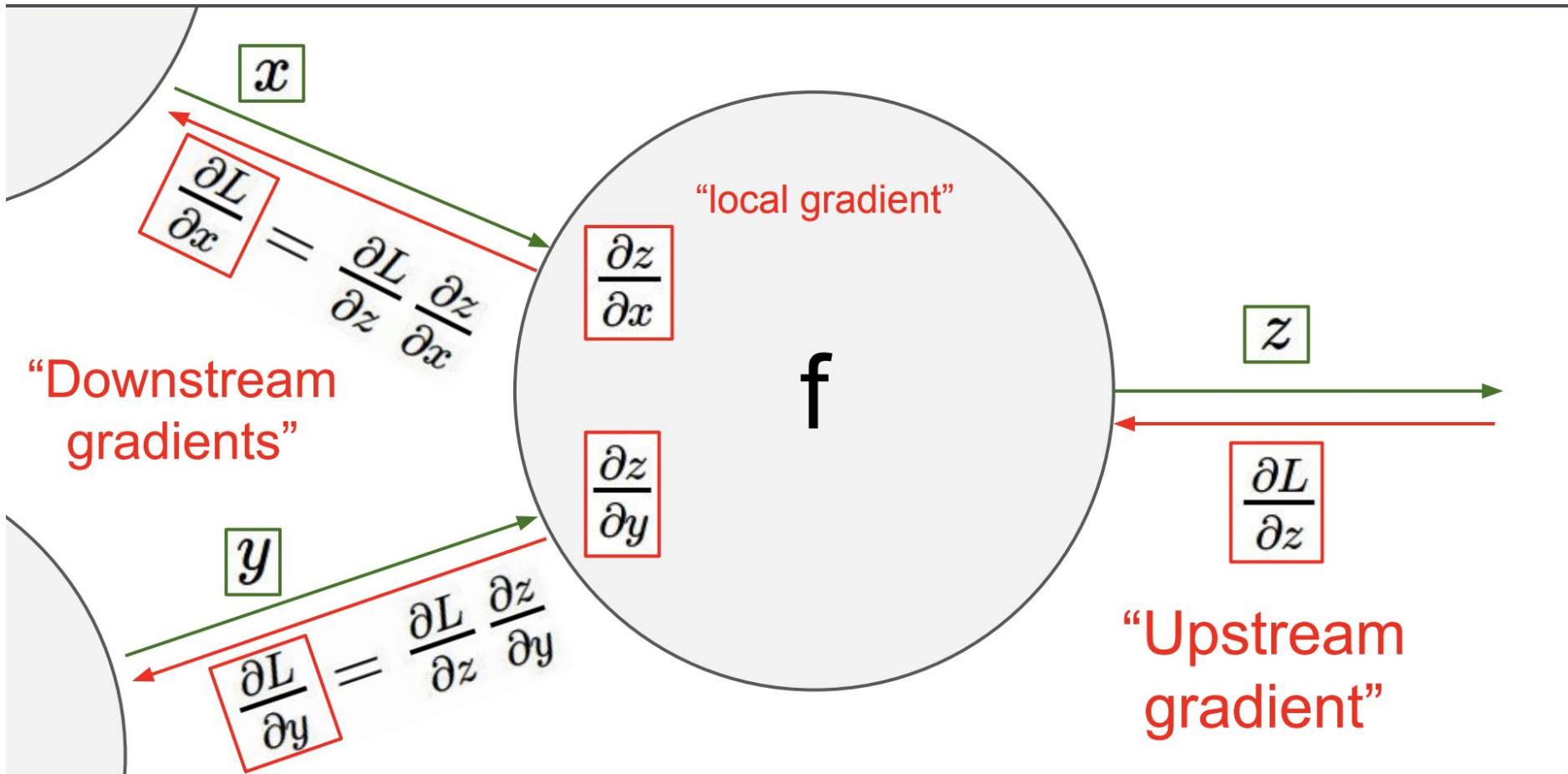
Chain rule:

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

Upstream gradient Local gradient

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

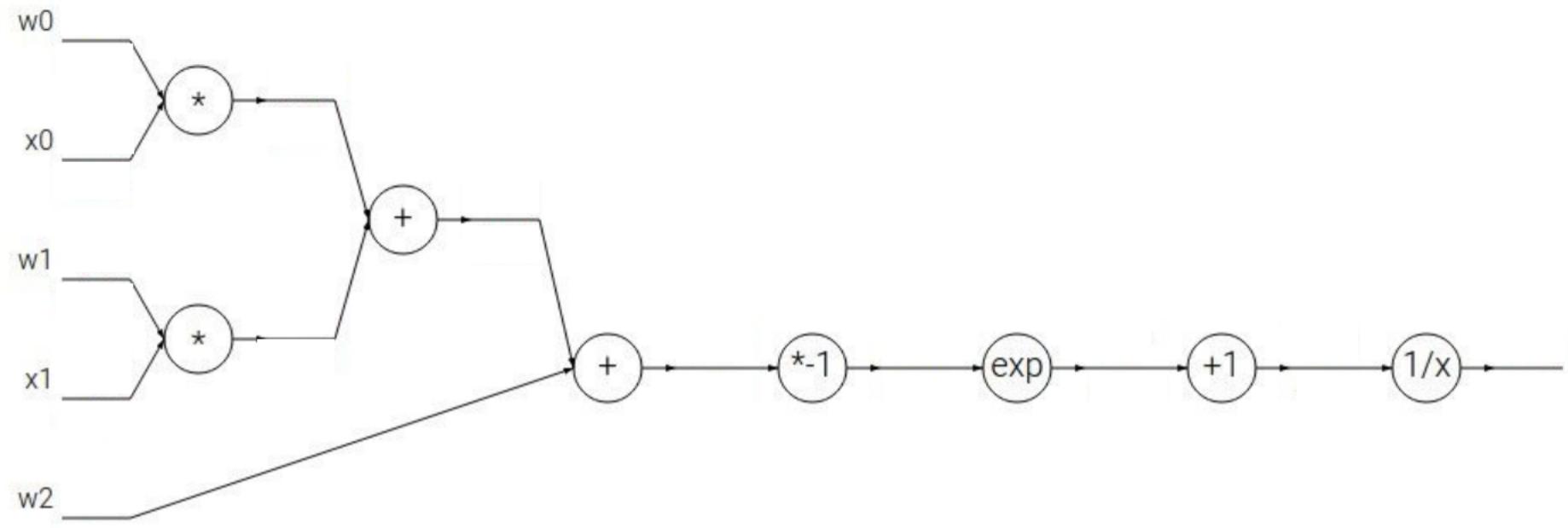
Backpropagation



Backpropagation: Example

Another example:

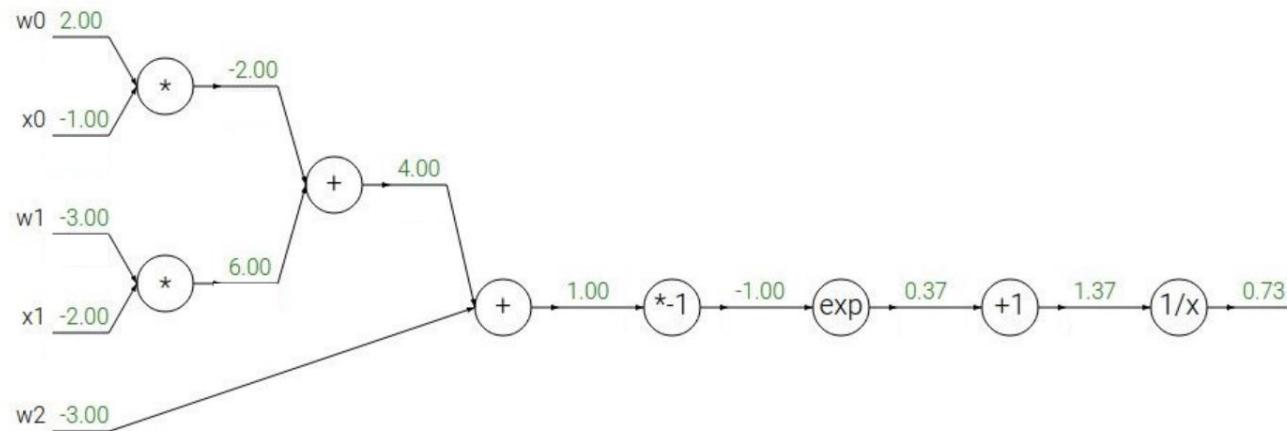
$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



Backpropagation: Example

Another example:

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



$$f(x) = e^x$$

→

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

→

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

→

$$\frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x$$

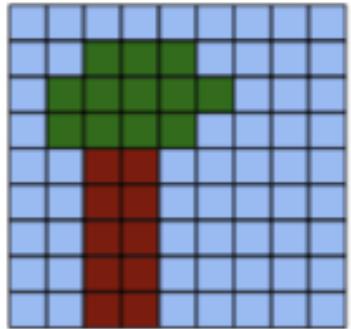
→

$$\frac{df}{dx} = 1$$

Other Links

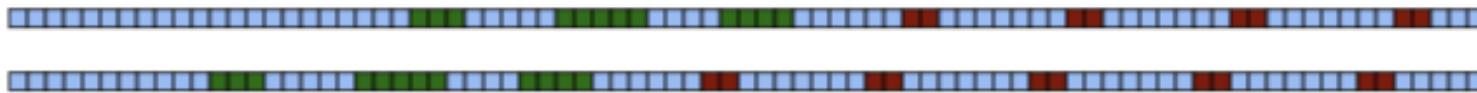
- Visualization
 - <http://playground.tensorflow.org>
- Libraries
 - <https://pytorch.org/>
 - <https://www.tensorflow.org/>
 - <https://pypi.org/project/Theano/>
 - <http://pyro.ai/>

Structuring neural networks: *How to exploit locality and translational Invariance?*



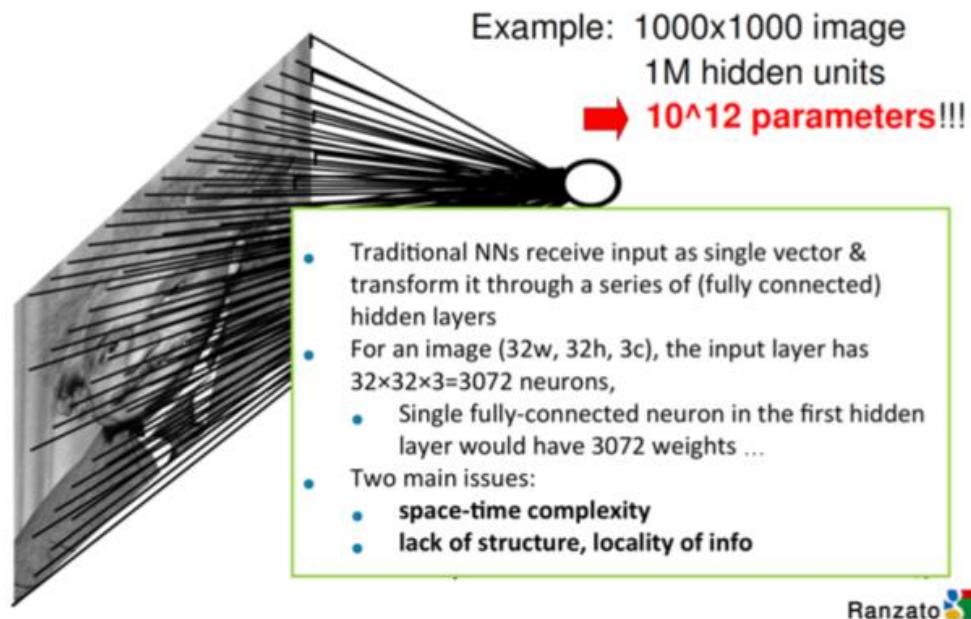
A digital image is a 2D grid of pixels.

A neural network expects a vector of numbers as input.

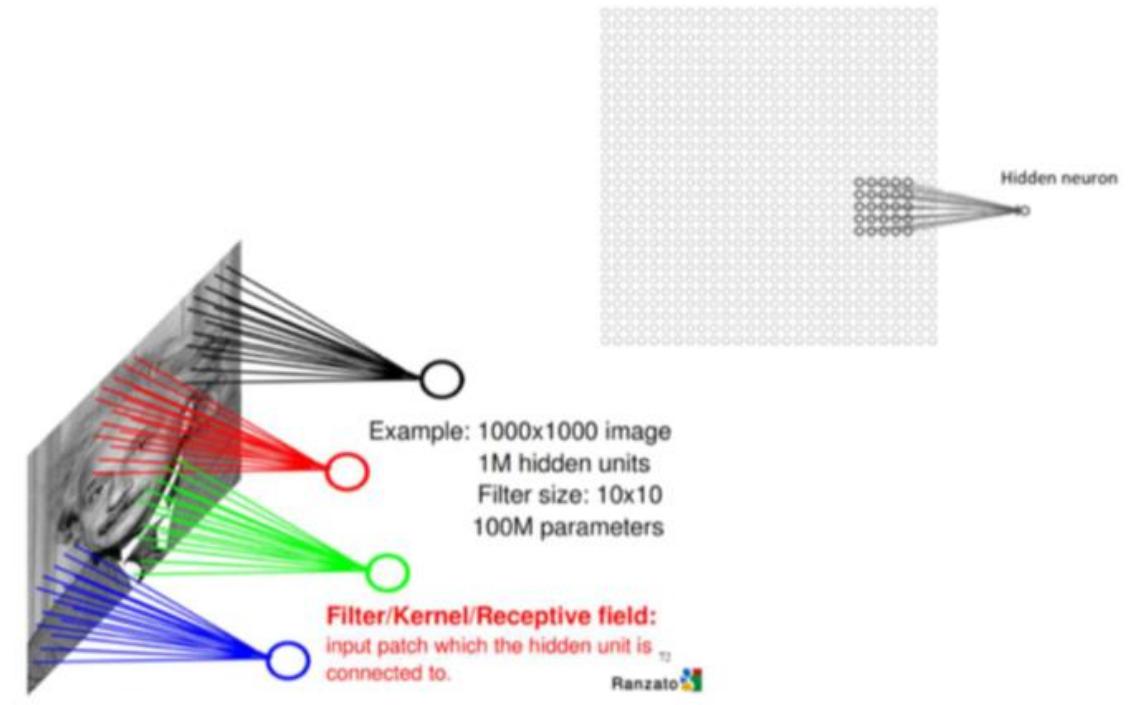


Locality of Information: Receptive Fields

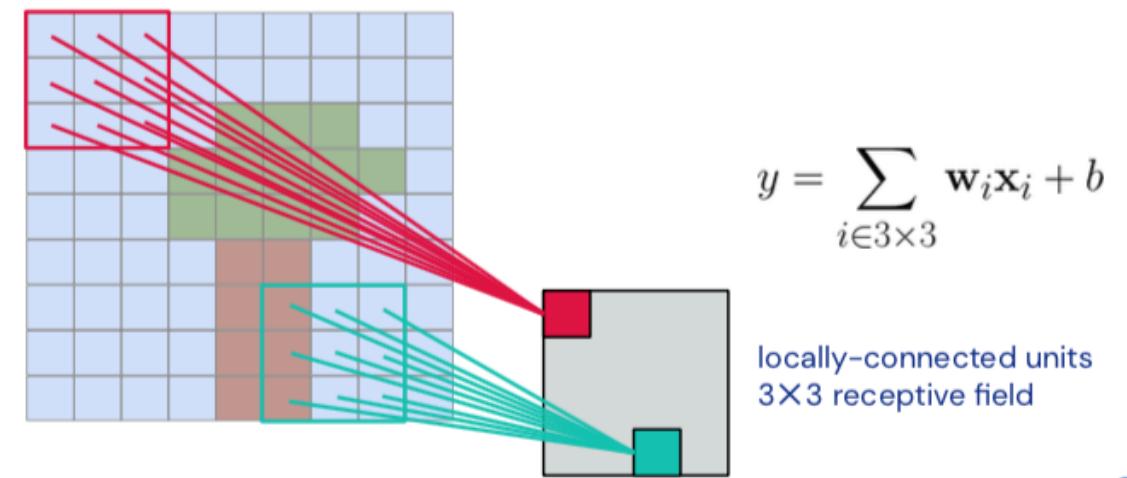
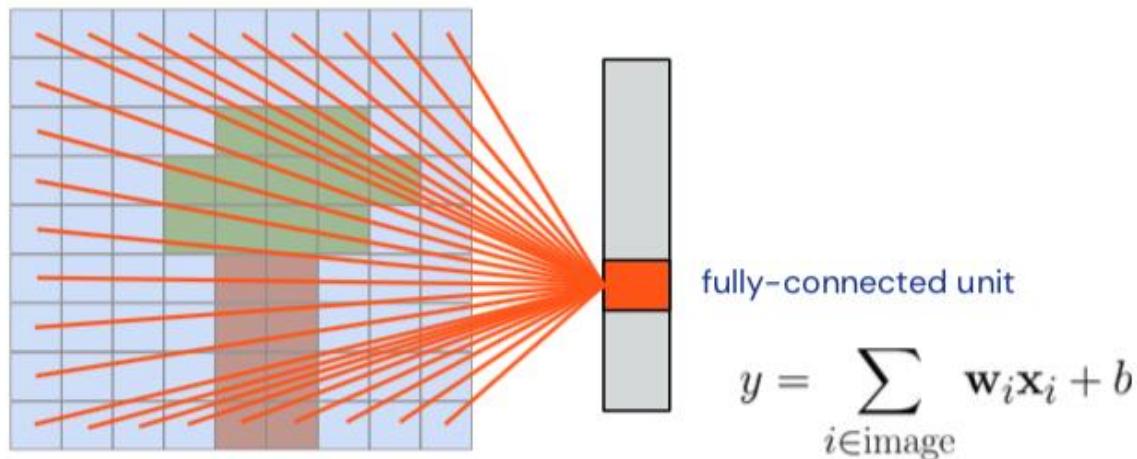
Fully connected network.



Convolutional NN

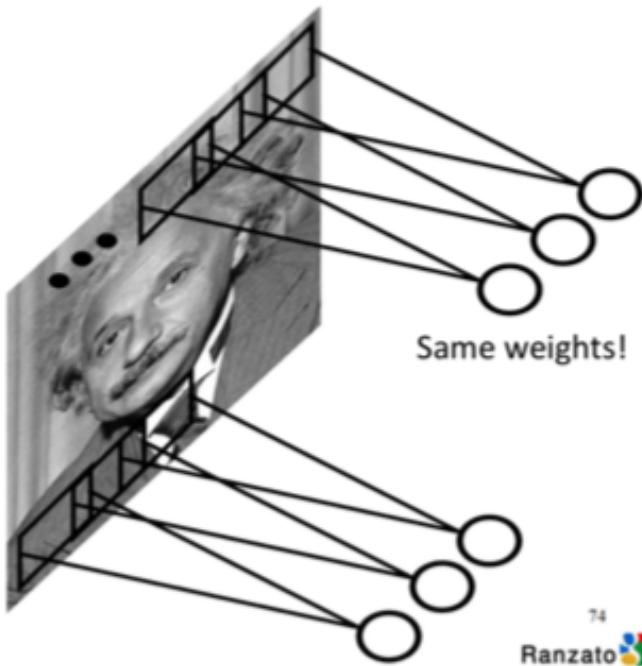


From Globally to Locally Connected

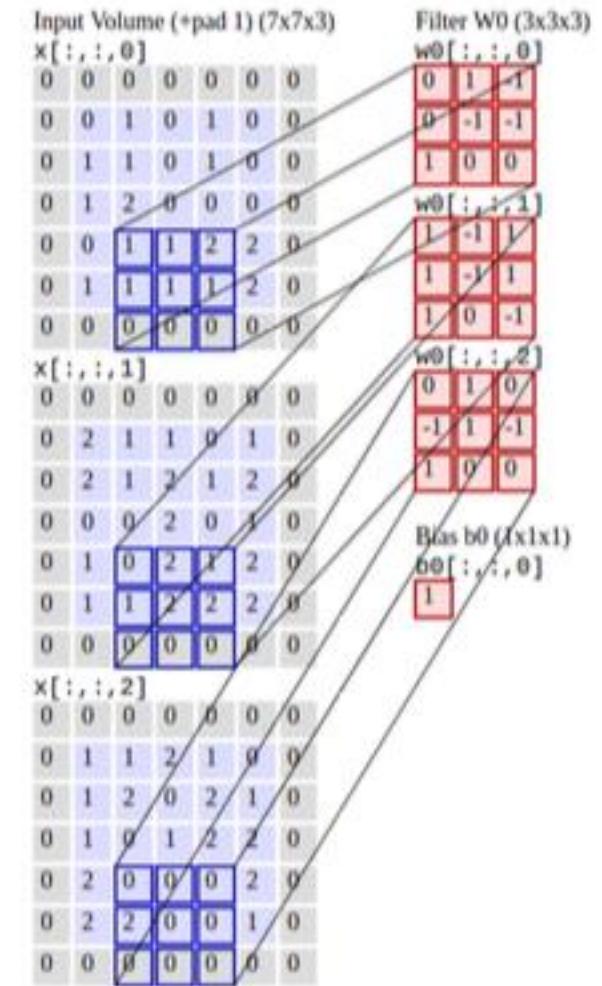


Convolutional NN

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

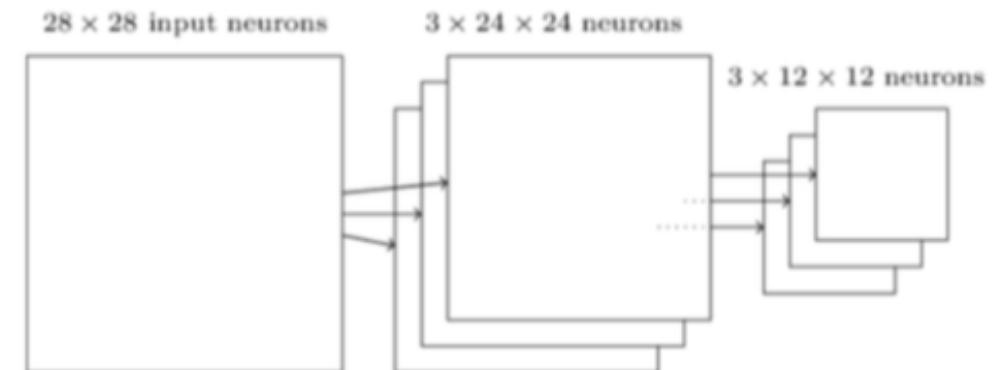
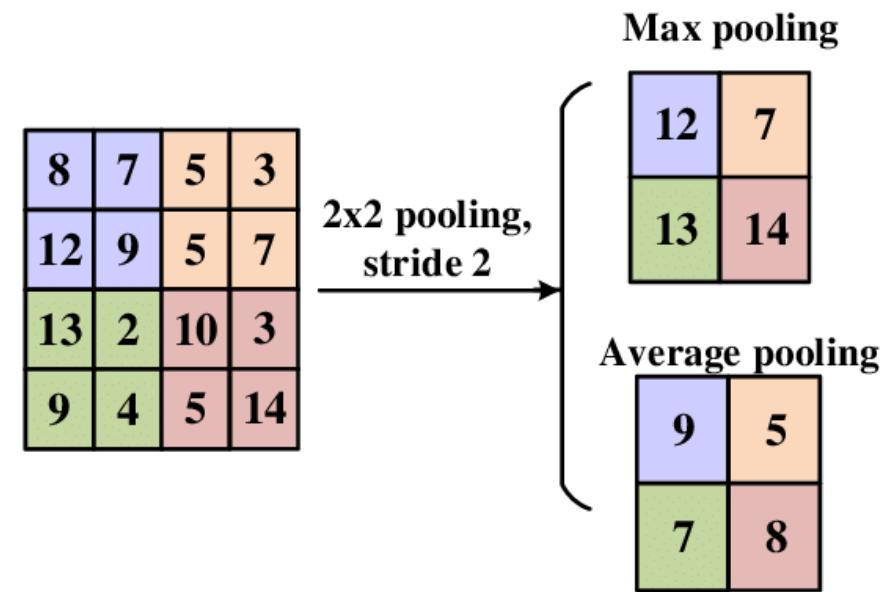


Pooling layers

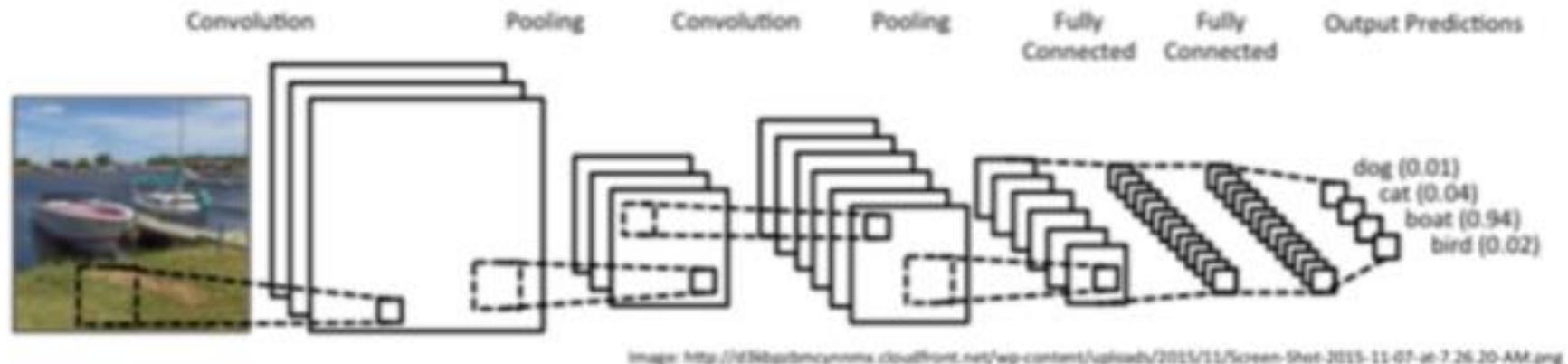
Pooling layers are usually used immediately after convolutional layers.

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

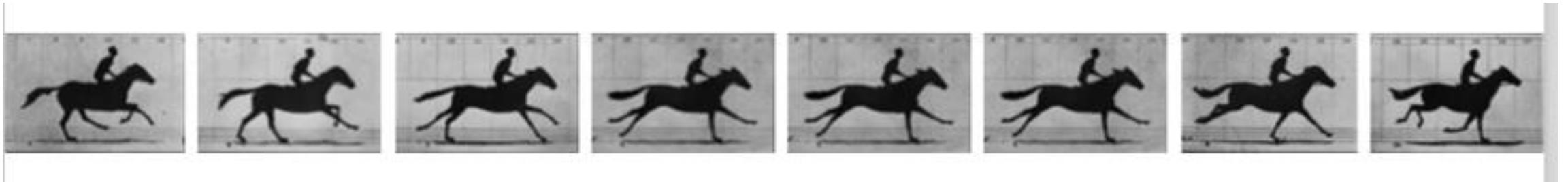


Convolution NN



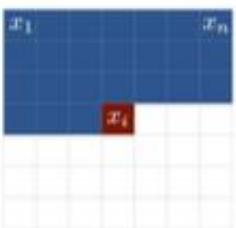
- Consider local structure and common extraction of features
- Not fully connected. Locality of processing
- Weight sharing for parameter reduction
- Learn the parameters of multiple convolutional filter banks
- Compress to extract salient features & favor generalization

Sequences



"Sequences really seem to be everywhere! We should learn how to model them. What is the best way to do that? Stay tuned!"

Words, letters



Images



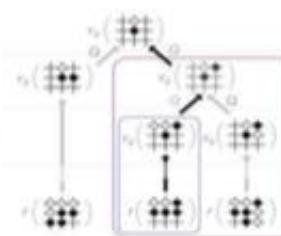
Speech



Programs



Videos



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 $p(\mathbf{x})$ from conditionals

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

Modeling

Modeling word

Modeling word probabilities

Modeling word probabilities is

Modeling word probabilities is really

Modeling word probabilities is really difficult

$$p(x_1)$$

$$p(x_2|x_1)$$

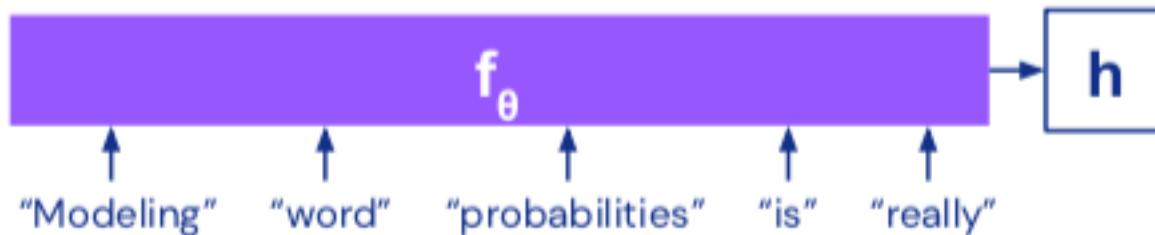
$$p(x_3|x_2, x_1)$$

$$p(x_4|x_3, x_2, x_1)$$

$$p(x_5|x_4, x_3, x_2, x_1)$$

$$p(x_6|x_5, x_4, x_3, x_2, x_1)$$

Vectorizing the conditional likelihood

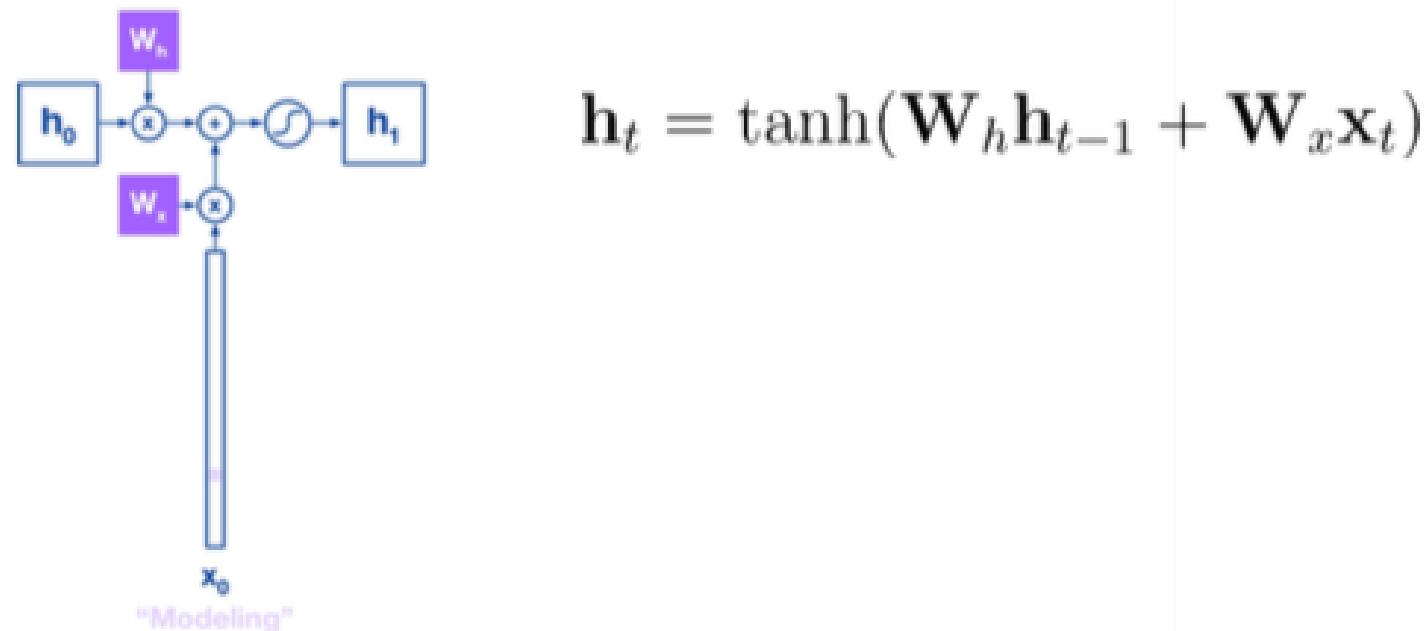


Desirable properties for f_θ :

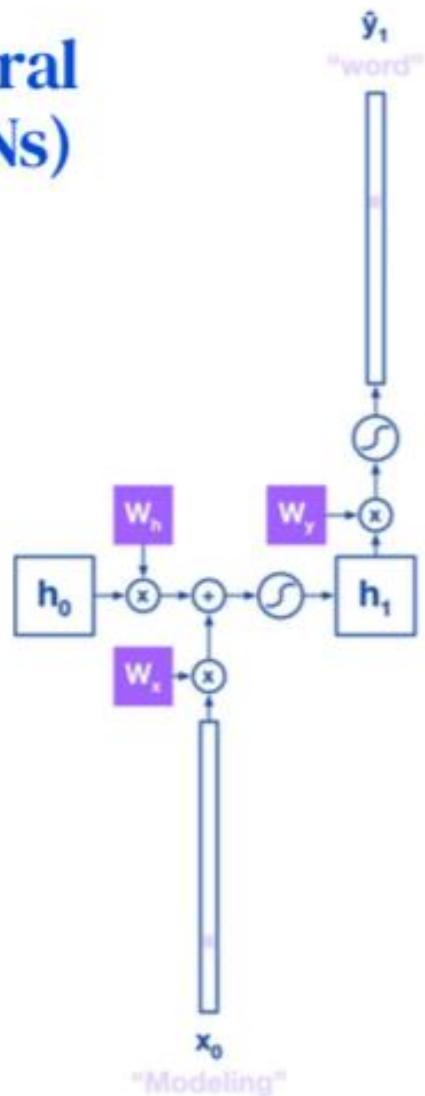
- Order matters
- Variable length
- Learnable (differentiable)
- Individual changes can have large effects
(non-linear/deep)

Recurrent Neural Networks (RNNs)

Persistent state variable \mathbf{h} stores information from the context observed so far.



Recurrent Neural Networks (RNNs)

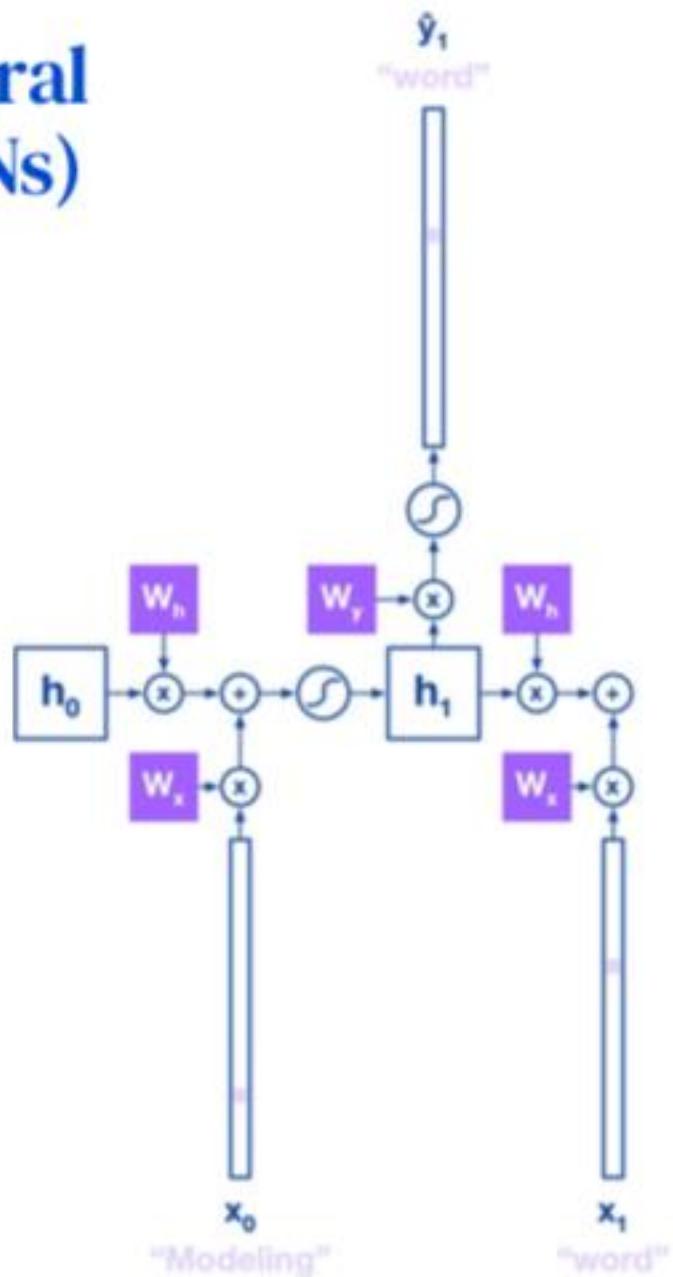


RNNs predict the target y (the next word) from the state h .

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

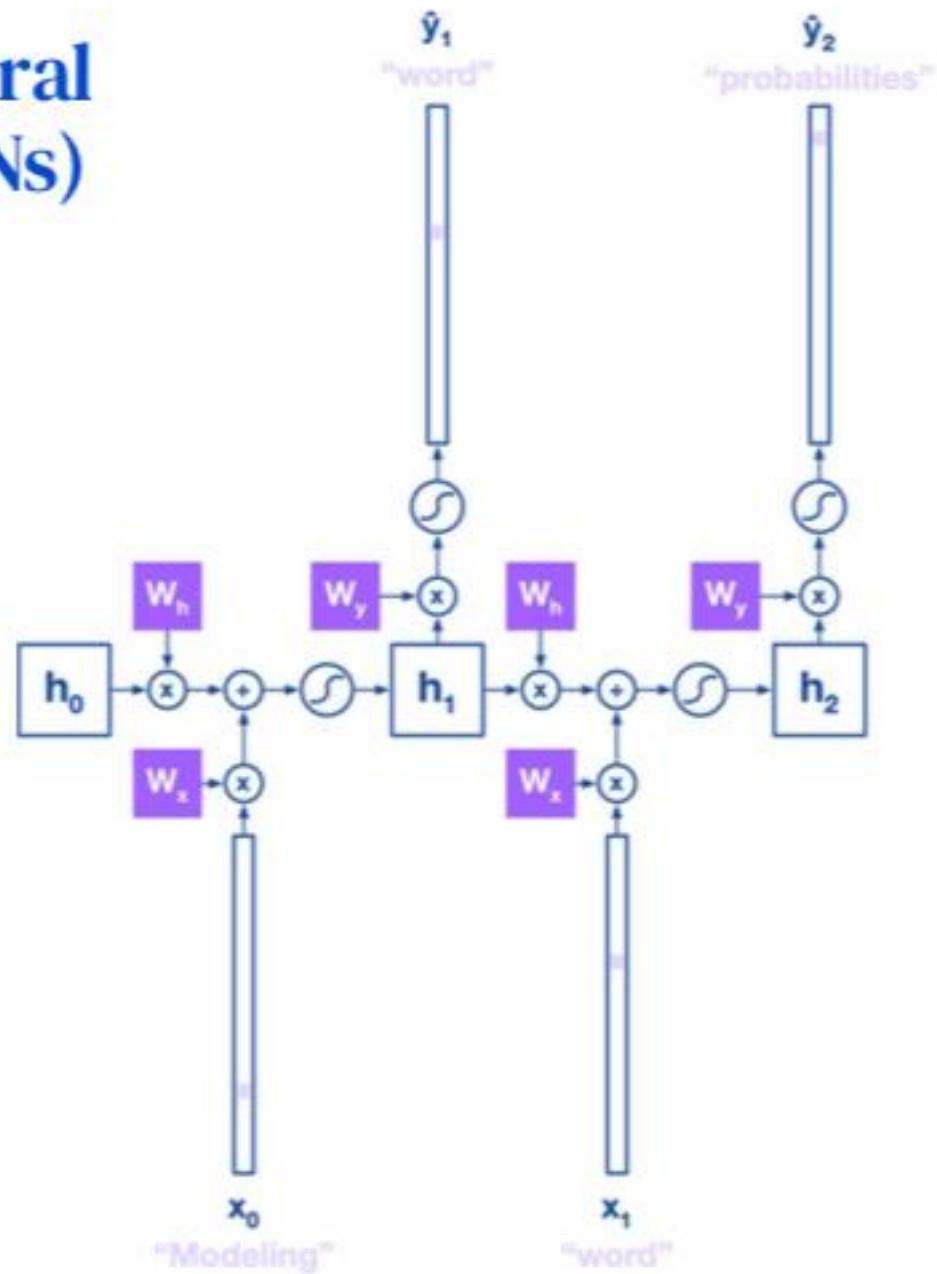
Softmax ensures we obtain a distribution over all possible words.

Recurrent Neural Networks (RNNs)



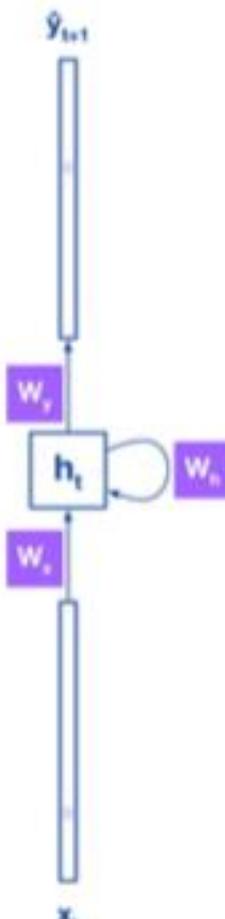
Input next word in sentence x_1

Recurrent Neural Networks (RNNs)

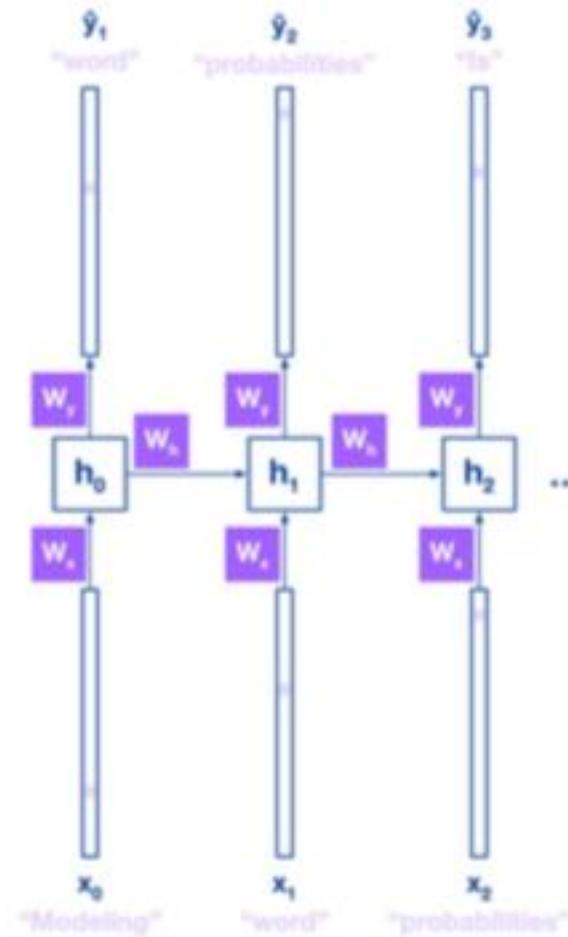


Recurrent Neural Networks (RNNs)

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:

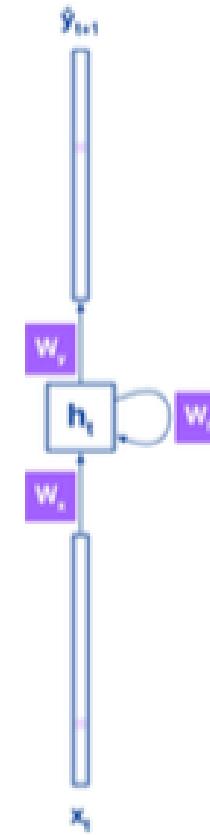
For one word:

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

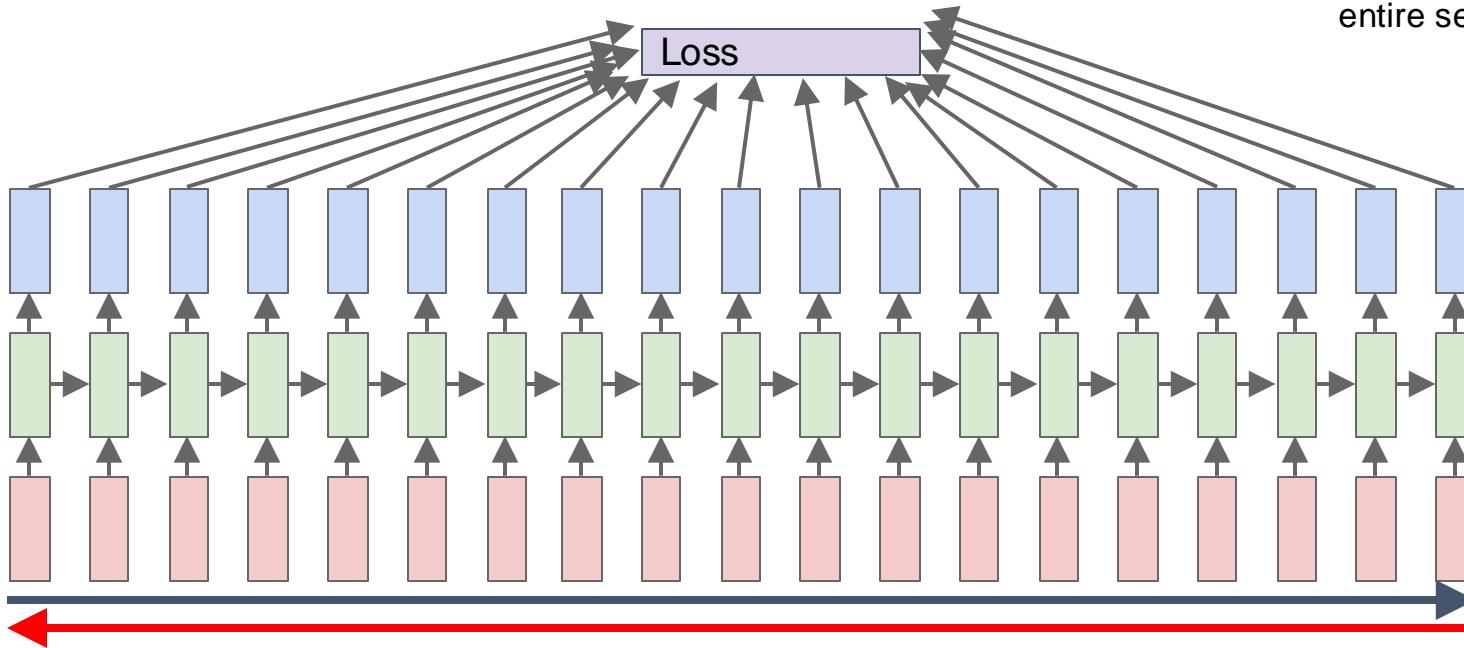
For the sentence:

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

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



Backprop through Time

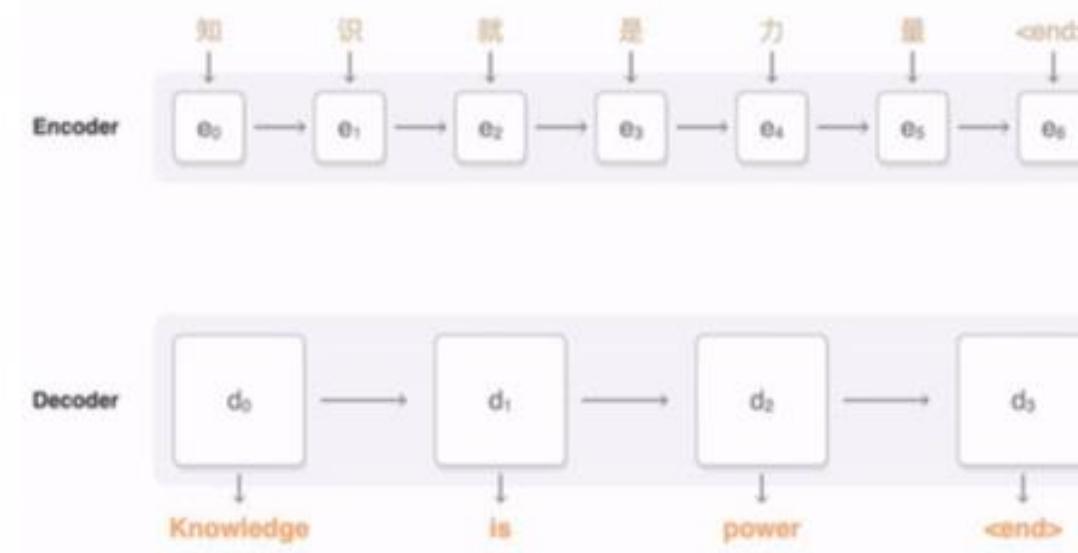


RNNs can have long or short dependencies. When there are long dependencies, gradients have trouble back-propagating through.

Other models such as LSTMs and beyond address that problem.

Applications

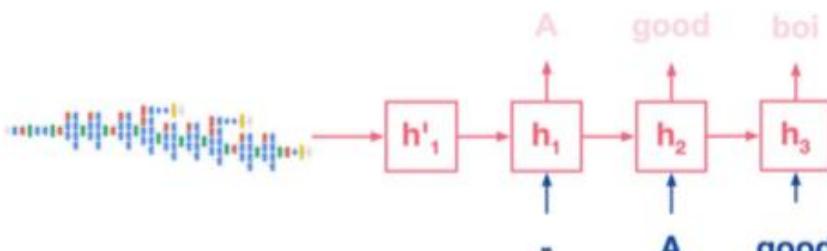
Google Neural Machine Translation



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.