COL333/671: Introduction to AI

Semester I, 2024-25

Learning – II: Gradient-based Learning and Neural Networks

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

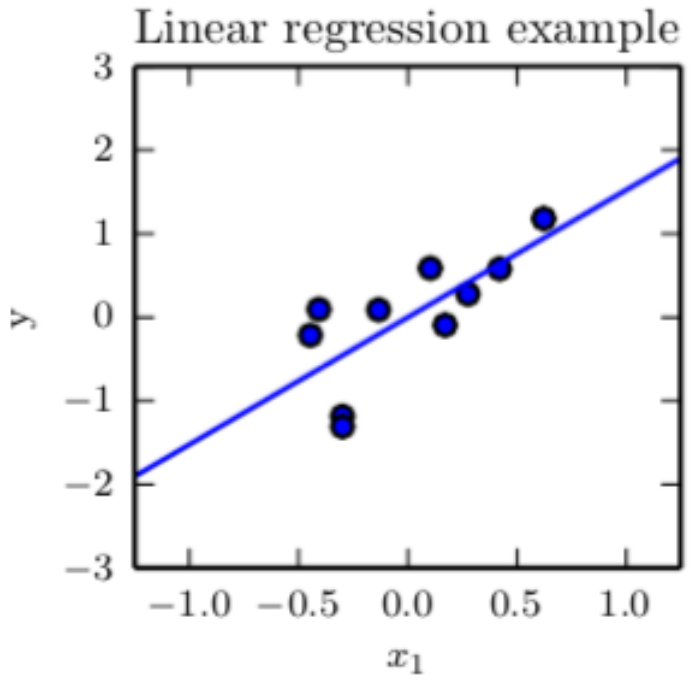
These slides are intended for teaching purposes only. Some material has been used/adapted from web sources and from slides by Doina Precup, Dorsa Sadigh, Percy Liang, Mausam, Parag, Emma Brunskill, Alexander Amini, Dan Klein, Anca Dragan, Nicholas Roy and others.

Learning a Best Fit Hypothesis: Linear Regression Task

Linear regression (no bias) $\hat{y} = \boldsymbol{w}^\top \boldsymbol{x}$

Linear regression (with bias) $\hat{y} = \boldsymbol{w}^\top \boldsymbol{x} + b$

Error term/Loss term $\text{MSE}_{\text{test}} = \frac{1}{m} ||\hat{\boldsymbol{y}}^{(\text{test})} - \boldsymbol{y}^{(\text{test})}||_2^2,$



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.

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

Linear Regression Example

Learning/Training:
Optimize the error w.r.t. the model parameters, w

$$\nabla_{\boldsymbol{w}} \text{MSE}_{\text{train}} = 0$$

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

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

$$\Rightarrow \nabla_{\boldsymbol{w}} \left(\boldsymbol{X}^{(\text{train})} \boldsymbol{w} - \boldsymbol{y}^{(\text{train})} \right)^\top \left(\boldsymbol{X}^{(\text{train})} \boldsymbol{w} - \boldsymbol{y}^{(\text{train})} \right) = 0 \tag{5.9}$$

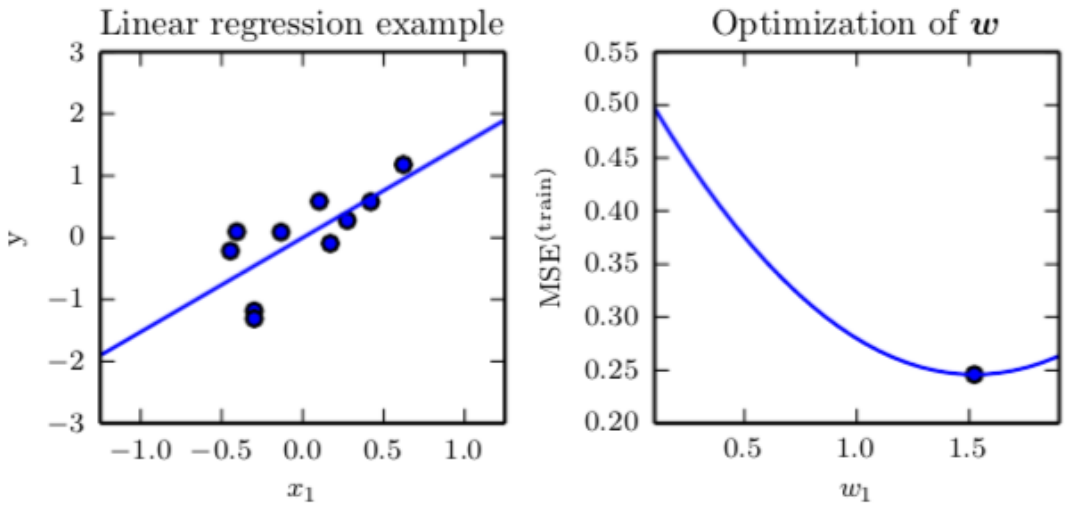
$$\Rightarrow \nabla_{\boldsymbol{w}} \left(\boldsymbol{w}^\top \boldsymbol{X}^{(\text{train})\top} \boldsymbol{X}^{(\text{train})} \boldsymbol{w} - 2\boldsymbol{w}^\top \boldsymbol{X}^{(\text{train})\top} \boldsymbol{y}^{(\text{train})} + \boldsymbol{y}^{(\text{train})\top} \boldsymbol{y}^{(\text{train})} \right) = 0 \tag{5.10}$$

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

$$\Rightarrow \boldsymbol{w} = \left(\boldsymbol{X}^{(\text{train})\top} \boldsymbol{X}^{(\text{train})} \right)^{-1} \boldsymbol{X}^{(\text{train})\top} \boldsymbol{y}^{(\text{train})} \tag{5.12}$$

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

Optimal w and the implied linear model



Function/model space **Parameter space**

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

Classification Task

Softmax applied at the last stage

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

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

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

pg. 148

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

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

Classification of an image as containing certain species.

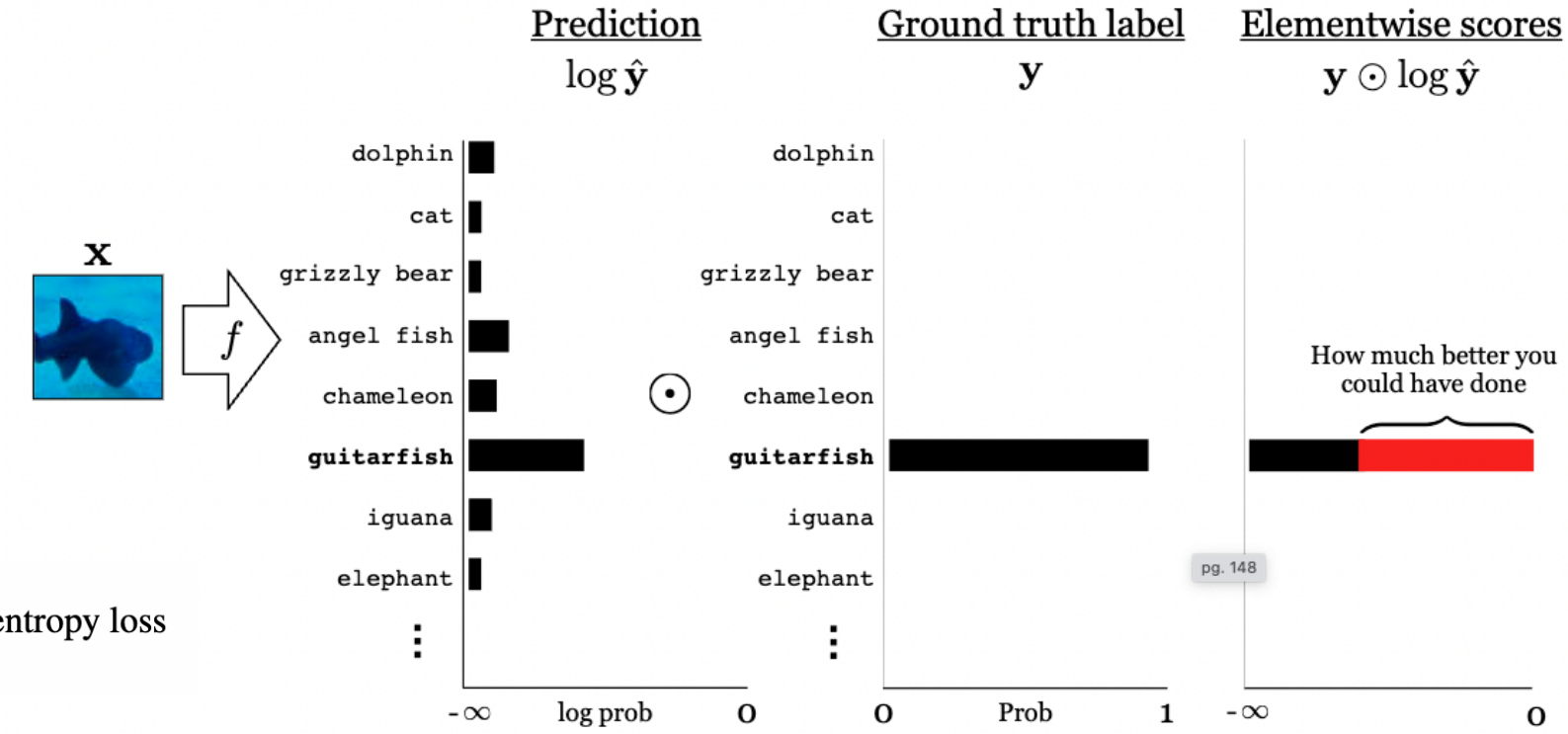


Image courtesy: https://mitpress.mit.edu/9780262048972/foundations-of-computer-vision/

How much to fit the data?

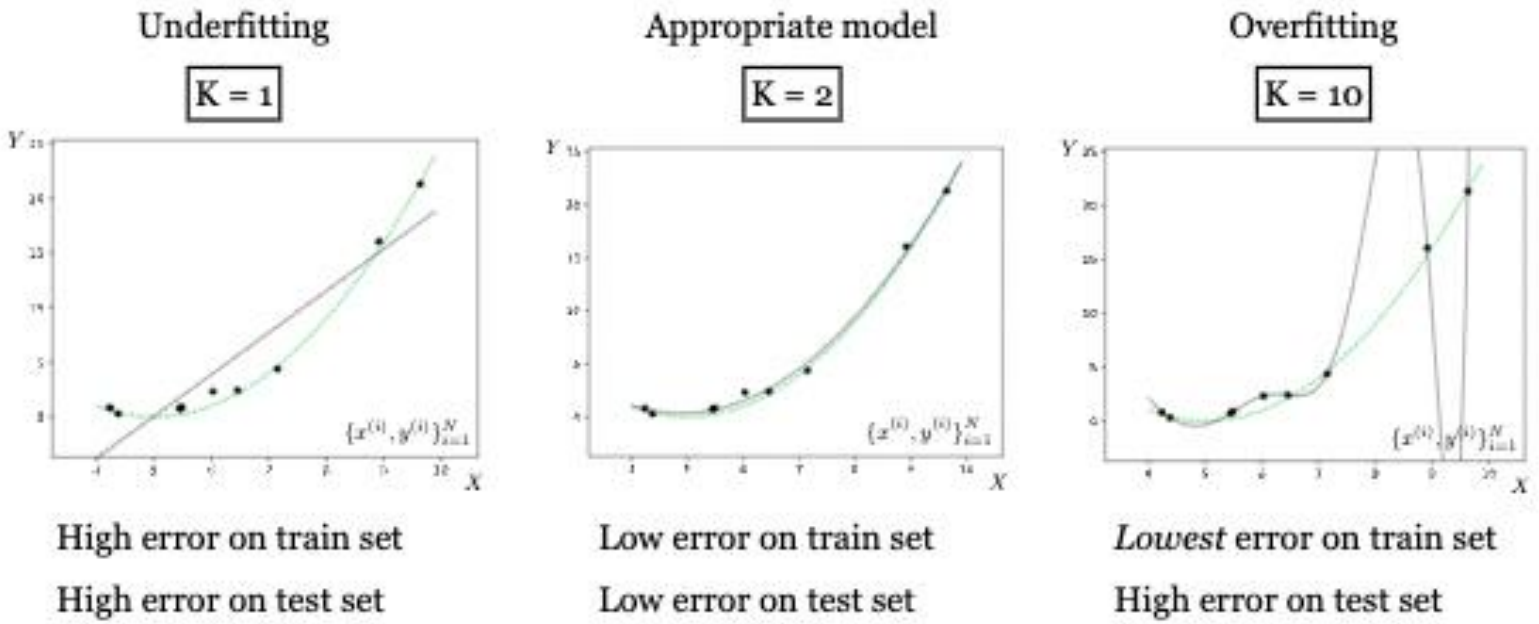
Overfitting and underfitting with polynomial functions

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



High error on train set
High error on test set

Low error on train set
Low error on test set

Lowest error on train set
High error on test set

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

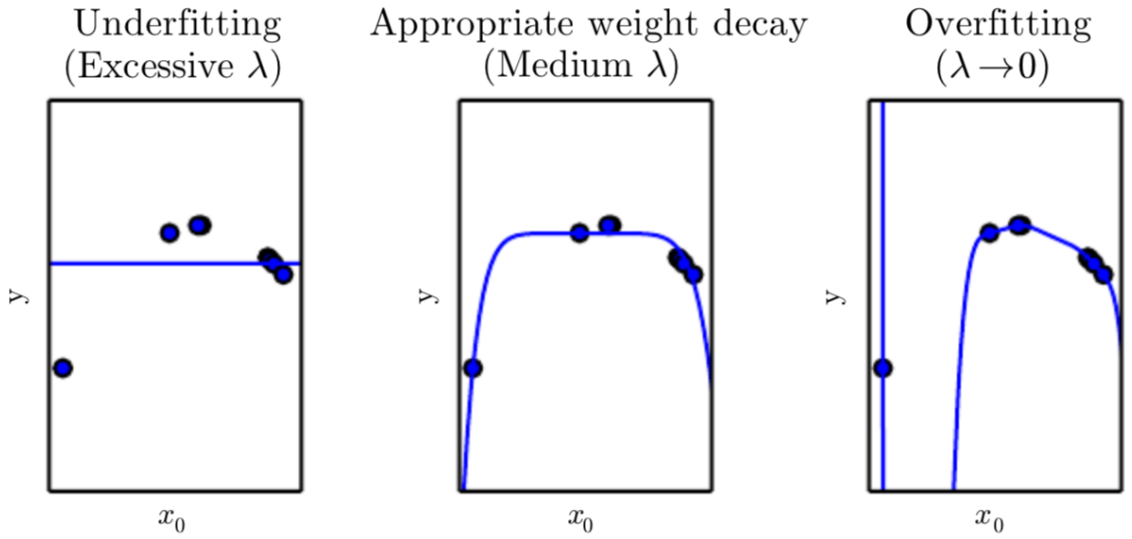
Regularization

- 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.

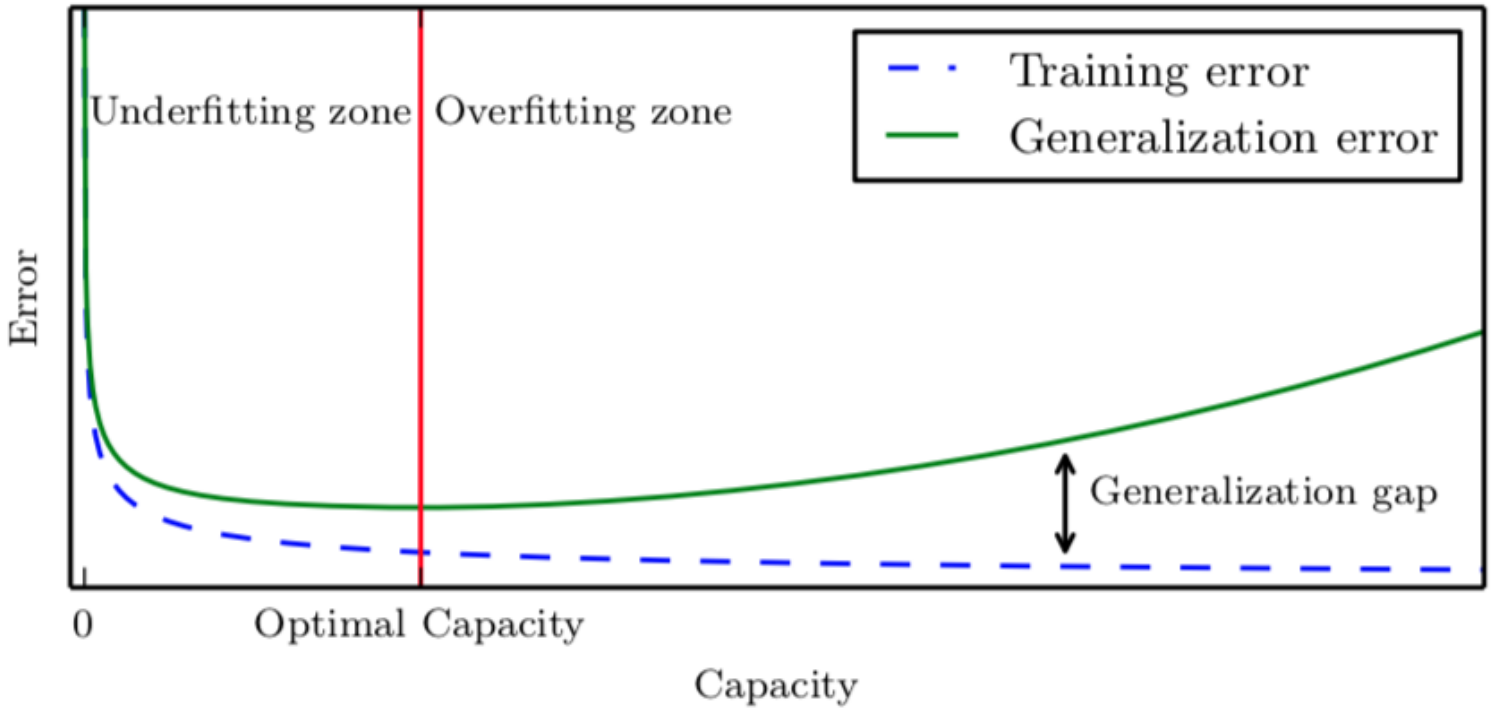
$$J(\boldsymbol{w}) = \mathrm{MSE}_{\mathrm{train}} + \lambda \boldsymbol{w}^\top \boldsymbol{w}$$



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.



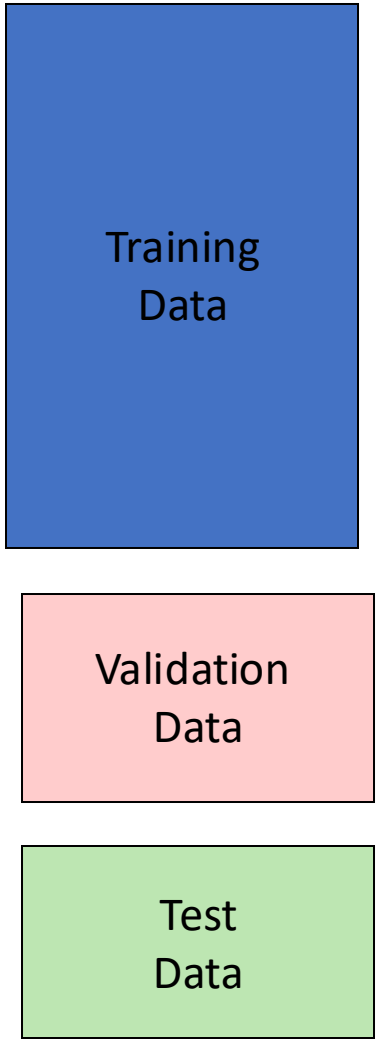
We can train a model only on the training set. The test set is not available during training.
How do we know the generalization error when we cannot use the test set?

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



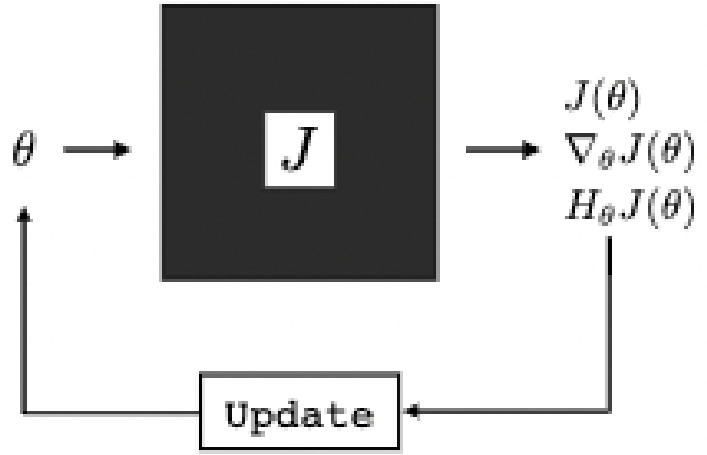
$$J_{\text{approx}} = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(f_\theta(x^{(i)}_{(\text{train})}), y^{(i)}_{(\text{train})})$$

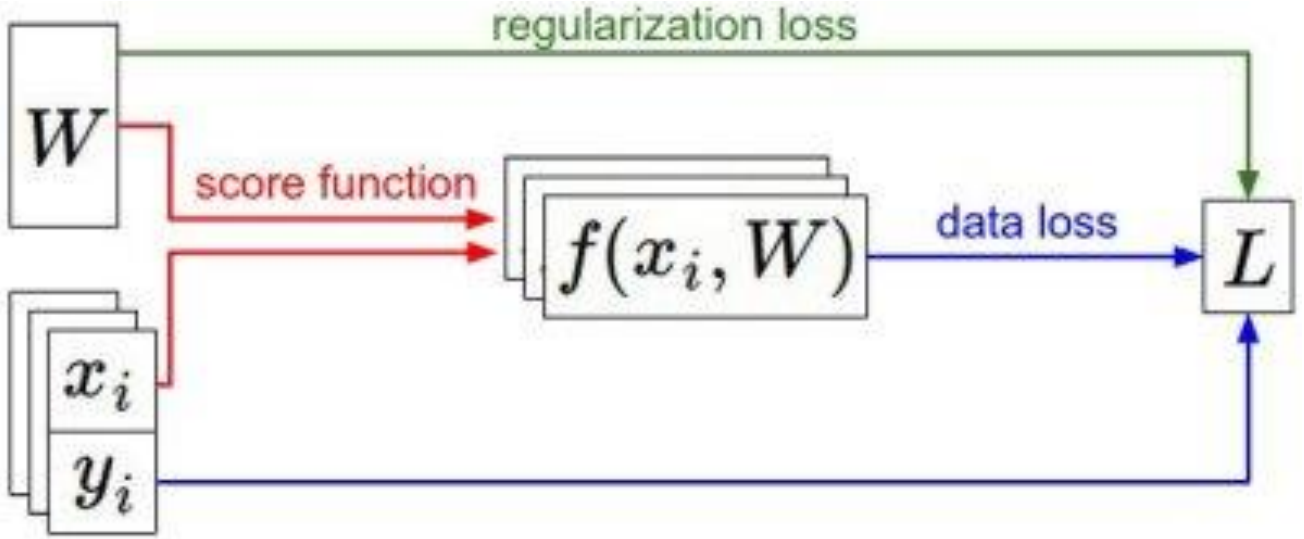
$$J_{\text{gen}} = \mathbb{E}_{x,y\sim p_{\text{data}}}[\mathcal{L}(f_\theta(x), y)]$$

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

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

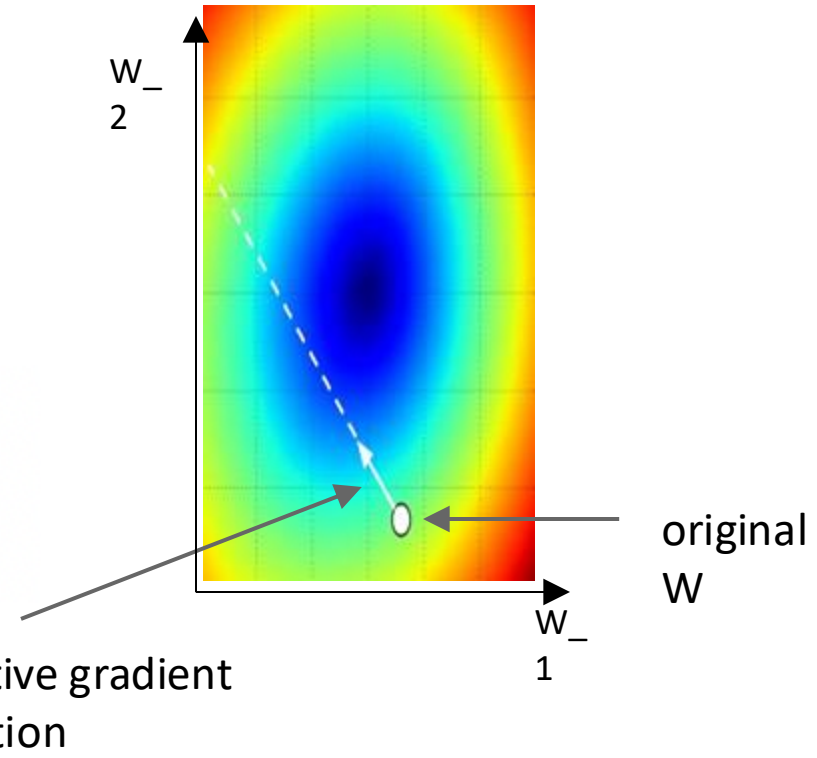




Gradient Descent

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

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



Learning rate schedules

Gradient descent with adaptive learning rate.

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

Types of learning rate schedules.

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

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

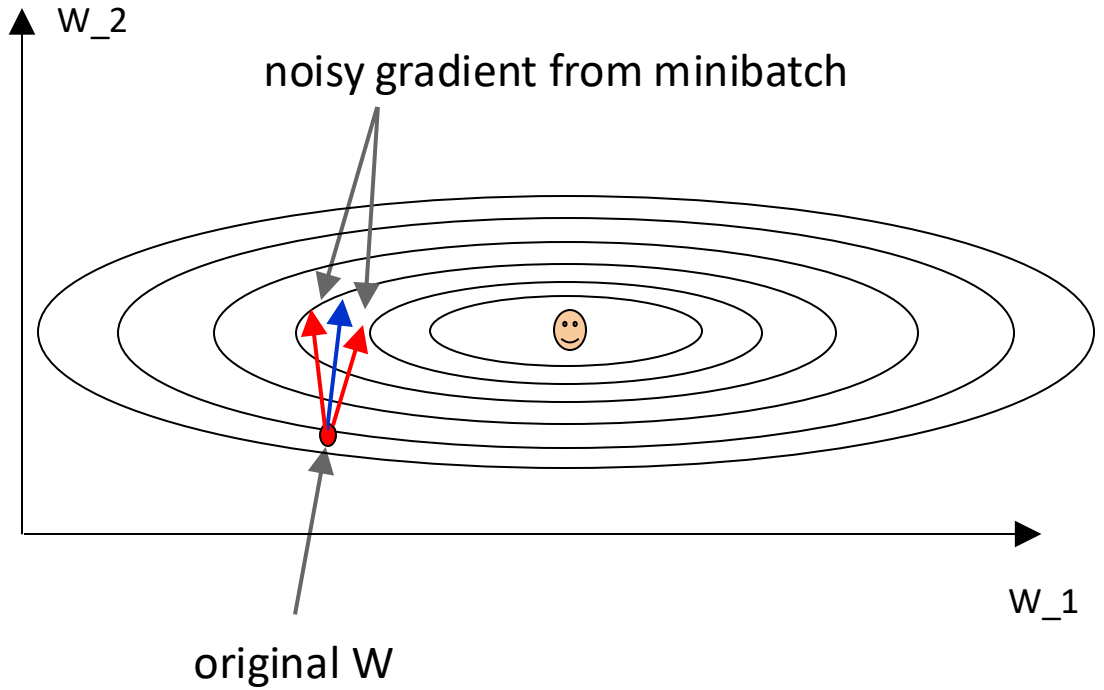
$$\texttt{lr}(\eta^0, k) = \frac{(K-k)}{K}\eta^0 \quad / \quad \text{linear decay}$$

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.

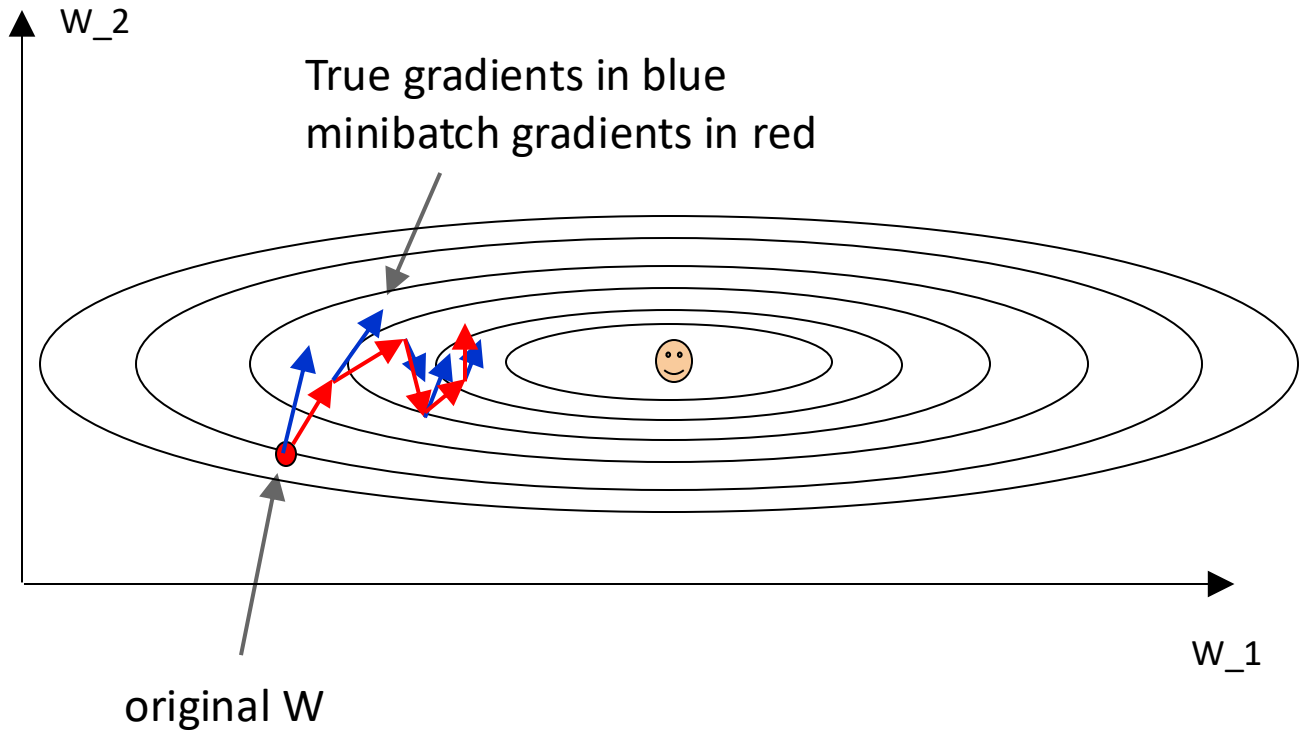
```
# Vanilla Minibatch Gradient Descent

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

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

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

Gradient for an additive cost function

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

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

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

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

Theta denotes the parameters. Same as w used in the previous slides.

Stochastic Gradient Descent

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

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

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

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

Gradient Descent with Momentum

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

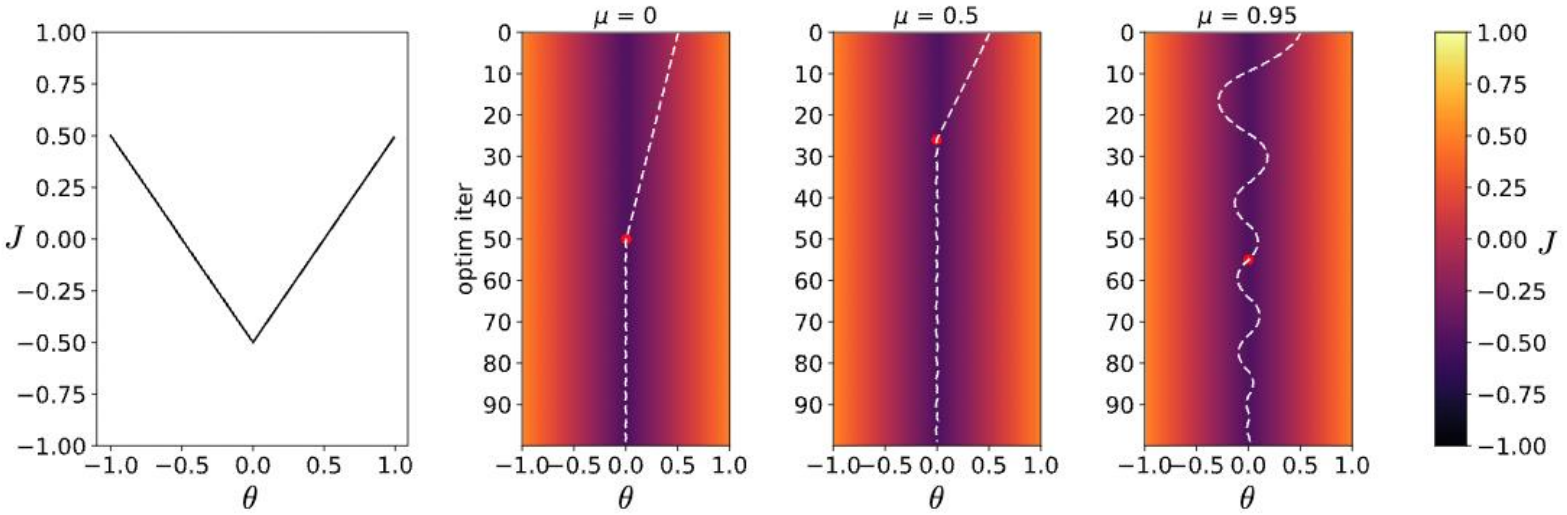
Output: trained parameter vector $\theta^* = \theta^K$

$\mathbf{v}^0 = \mathbf{0}$

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

  $\mathbf{v}^{k+1} = \mu \mathbf{v}^k - \eta \nabla_\theta J(\theta^k)$

  $\theta^{k+1} \leftarrow \theta^k + \mathbf{v}^{k+1}$

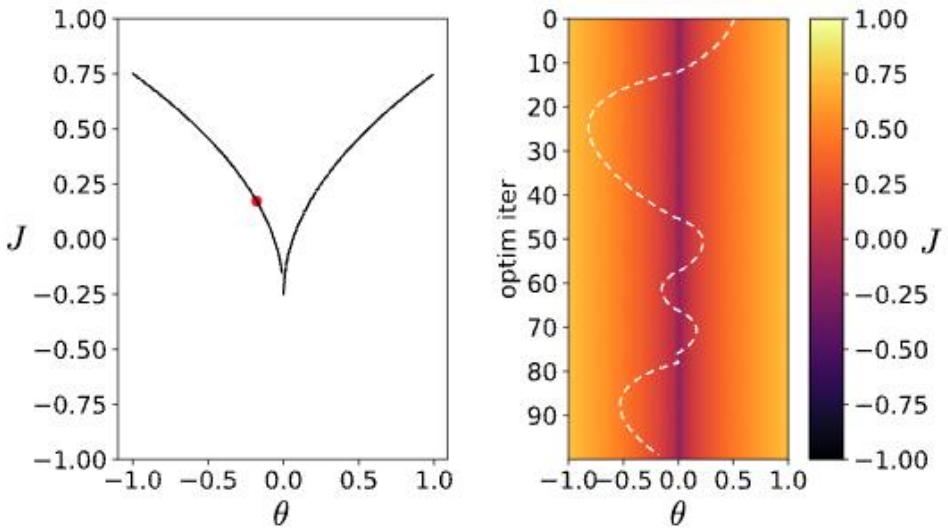


Gradient Clipping

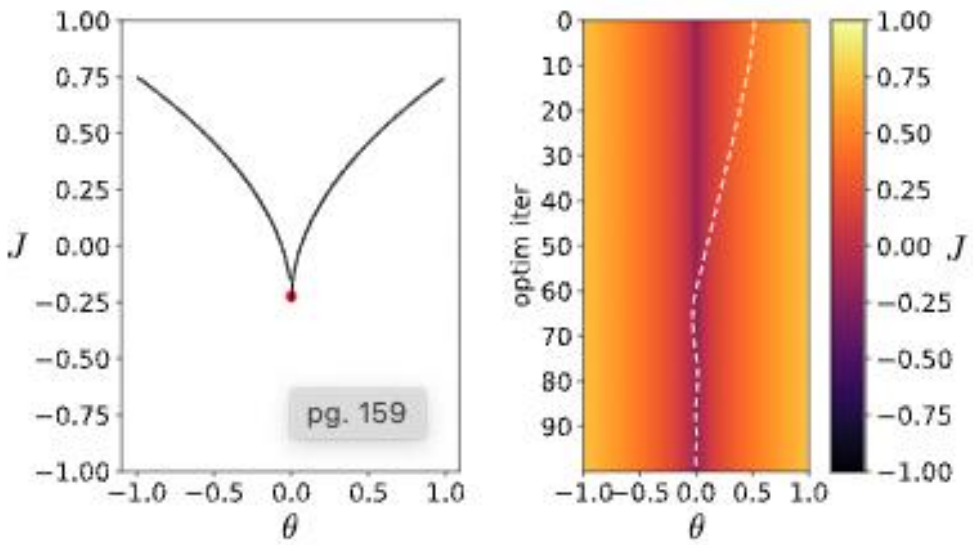
`clip` is the "clipping" function: $\texttt{clip}(v, -m, m) = \max(\min(v, m), -m)$

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

Without clipping – instabilities

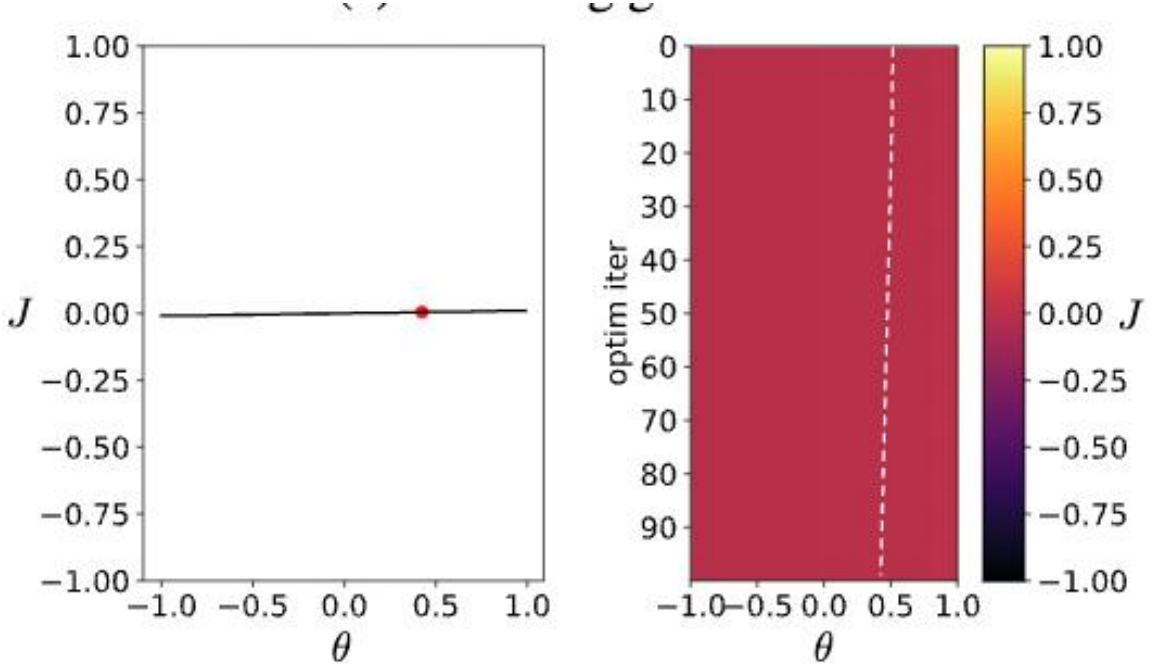


With clipping – stable convergence

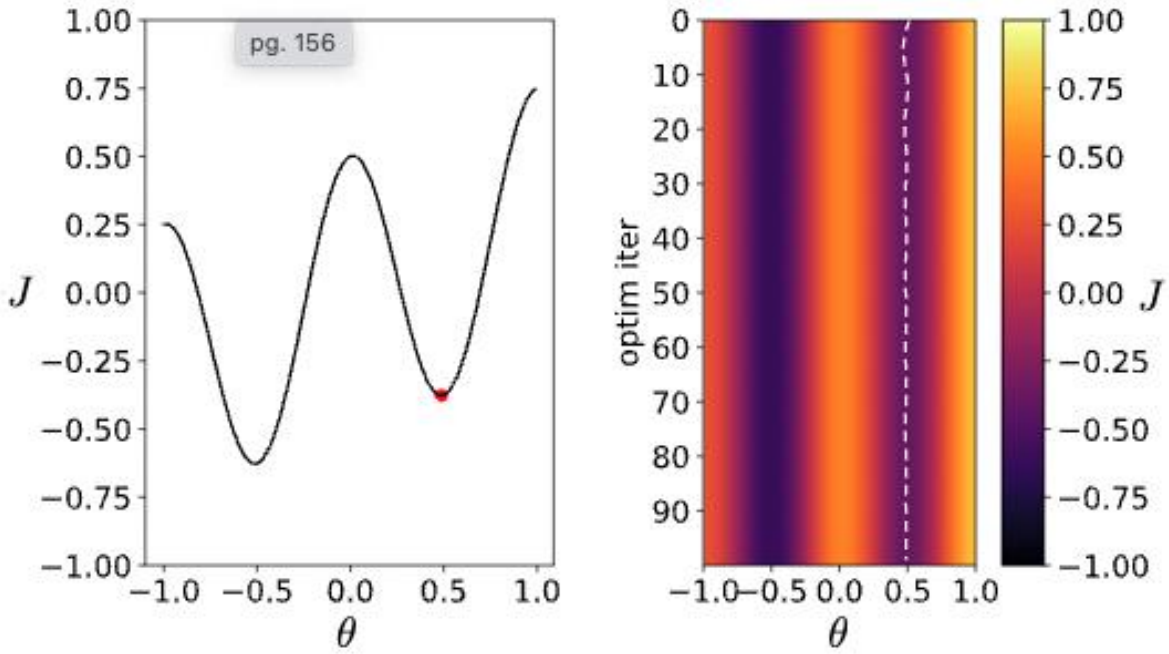


Other Aspects

Vanishing gradients



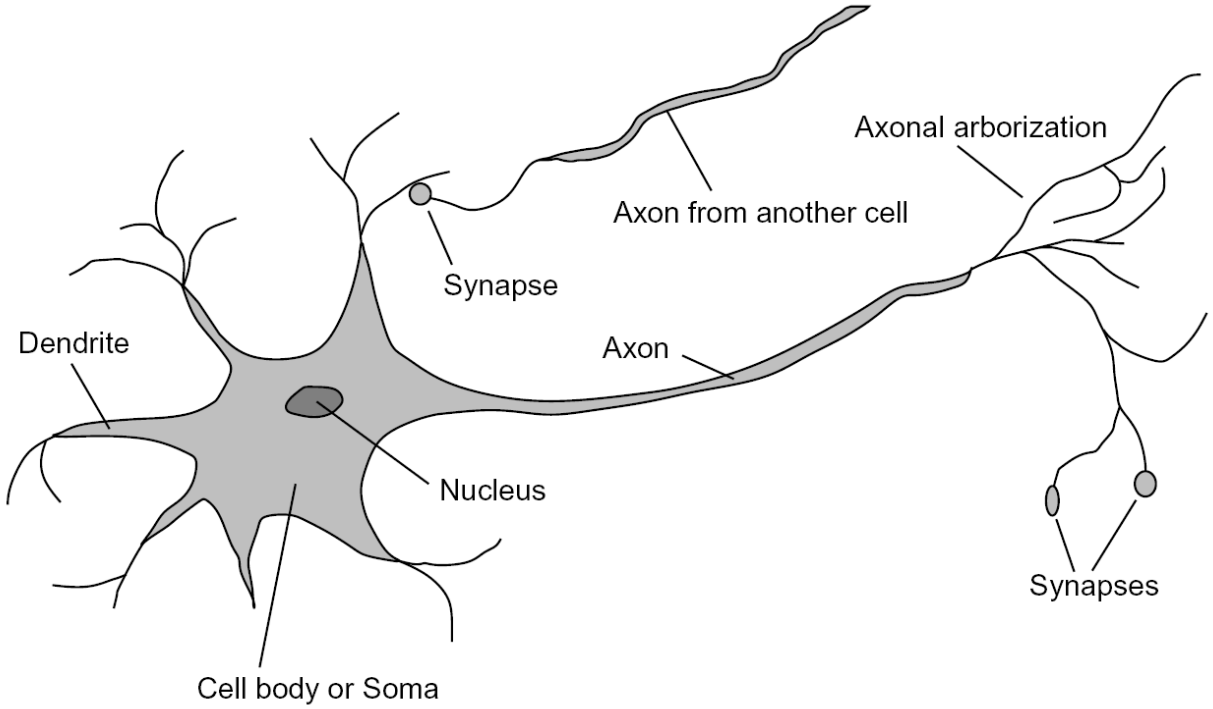
Multiple local minima



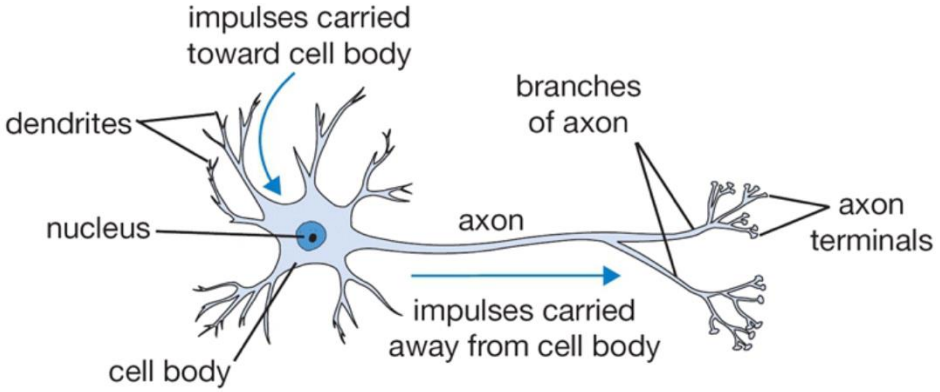
Neural Networks: *Biological*

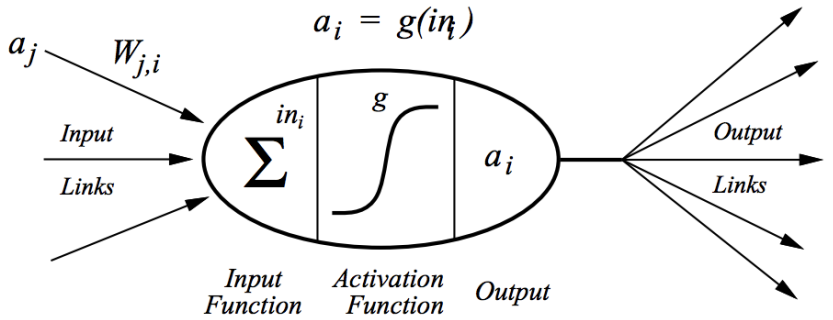
Neurons in the brain

- Activations and inhibitions
- Parallelism
- Connected networks



Modeling a Neuron





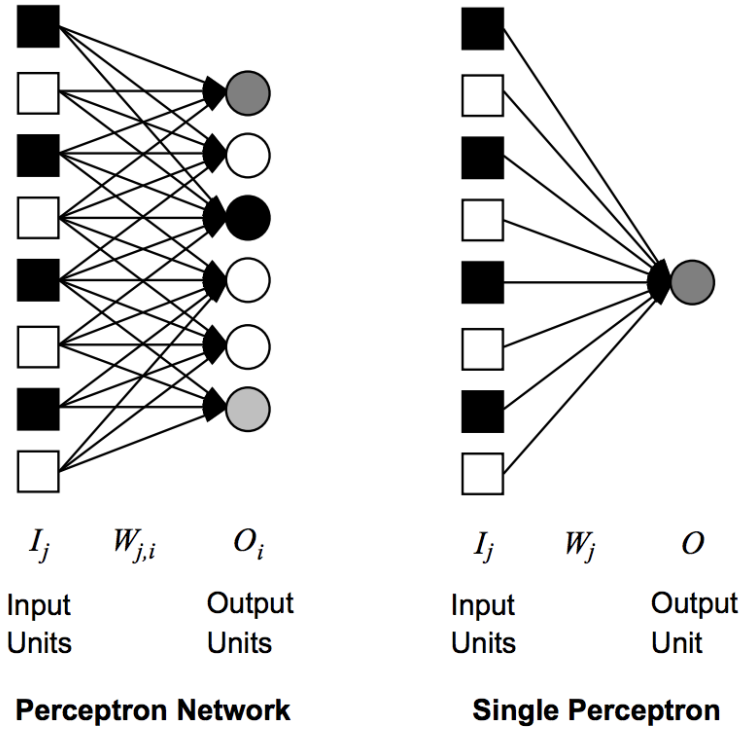
$$a_i = g(\sum_j W_{j,i} a_j)$$

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: 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.



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
```

x → $f(x)$ → y

```
Hello,

Do you want free printr
cartriges?  Why pay more when
you can get them ABSOLUTELY
FREE!  Just
```

→

```
# free      : 2
YOUR_NAME   : 0
MISSPELLED  : 2
FROM_FRIEND : 0
...
```

→ SPAM or +

→

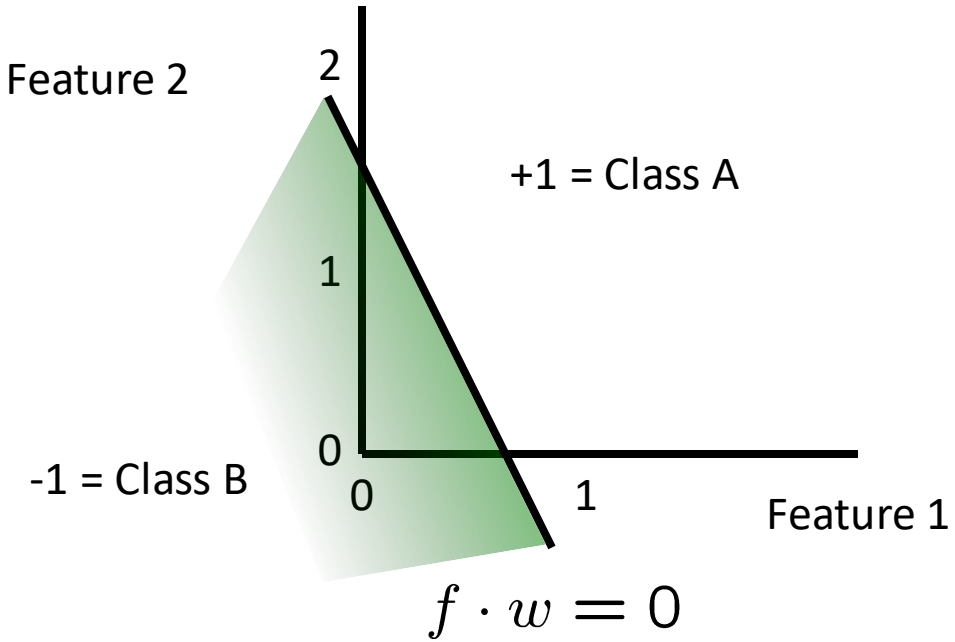
```
PIXEL-7,12  : 1
PIXEL-7,13  : 0
...
NUM_LOOPS   : 1
...
```

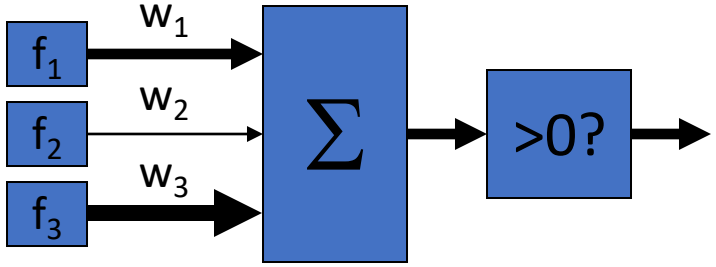
→ “2”

Classification with a Perceptron

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

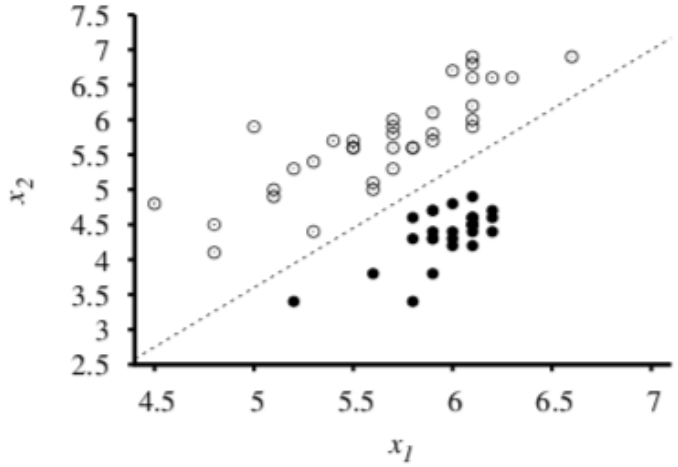
$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$





Perceptron

Decision rule (Binary case)



Binary classification task

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

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

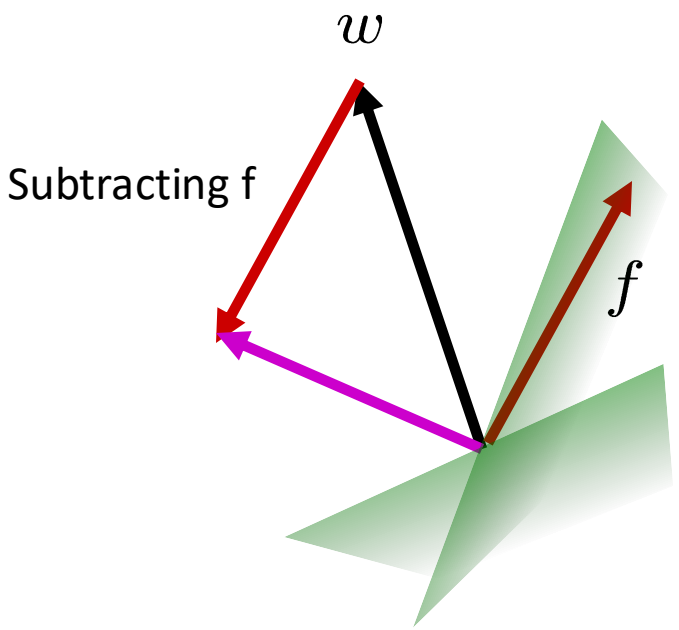
One side of the decision boundary is class. A and the other is class B. A threshold is introduced.

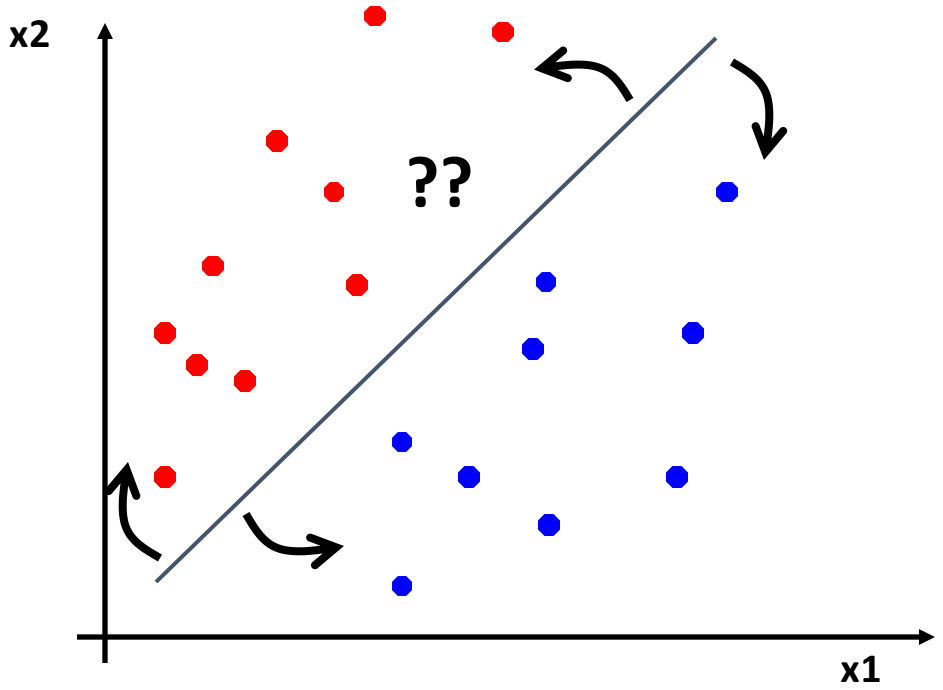
How to arrive at the separator?

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/

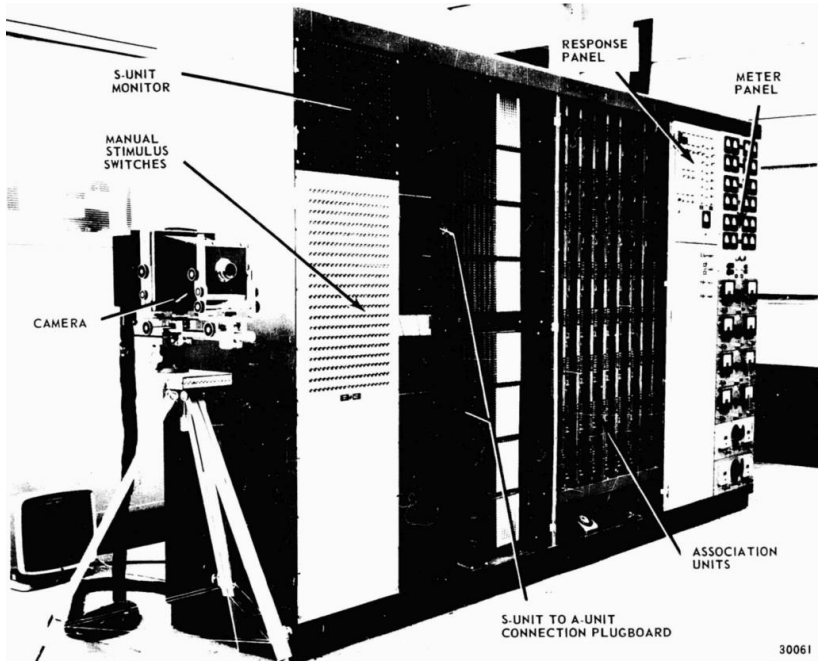
Some History

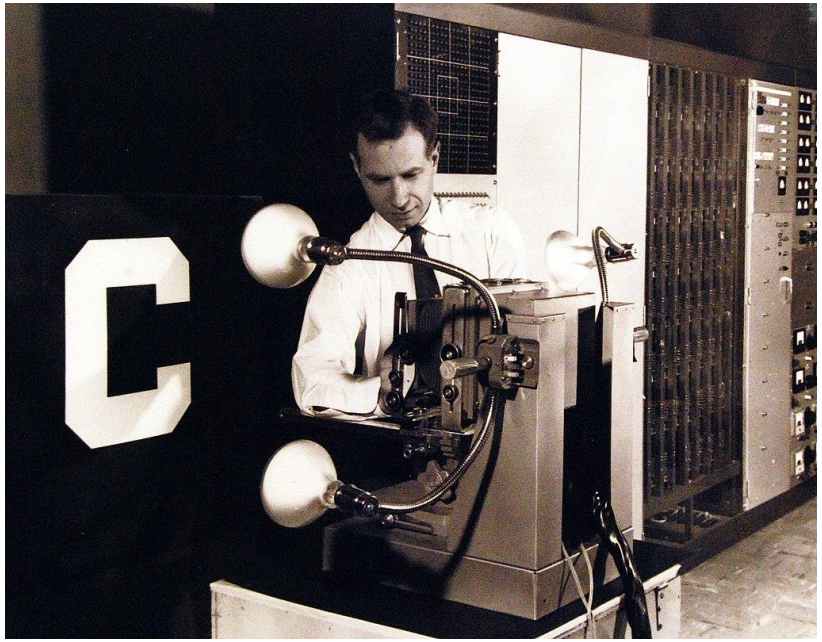
A physical realization of the perceptron machine.

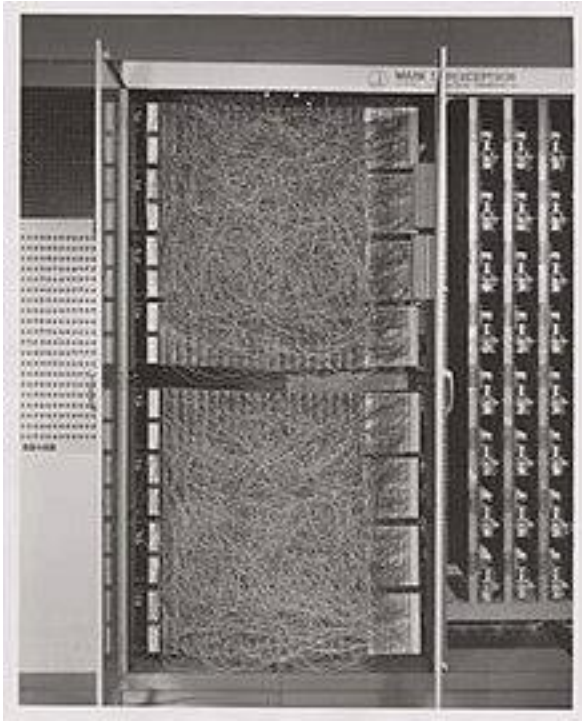
The perceptron: A probabilistic model for information storage and organization in the brain.

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)



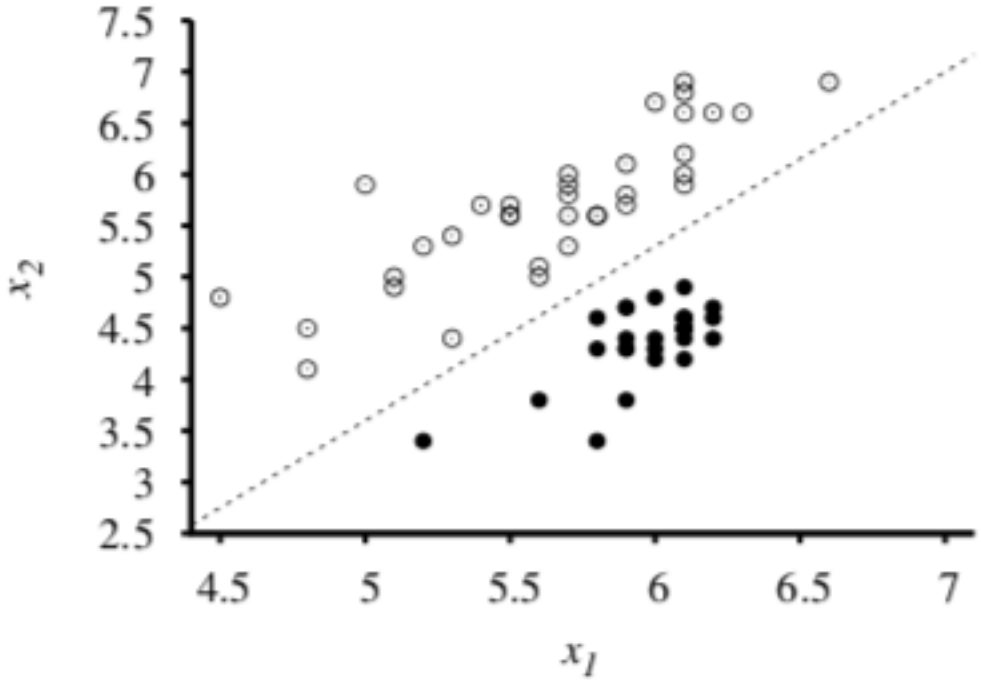




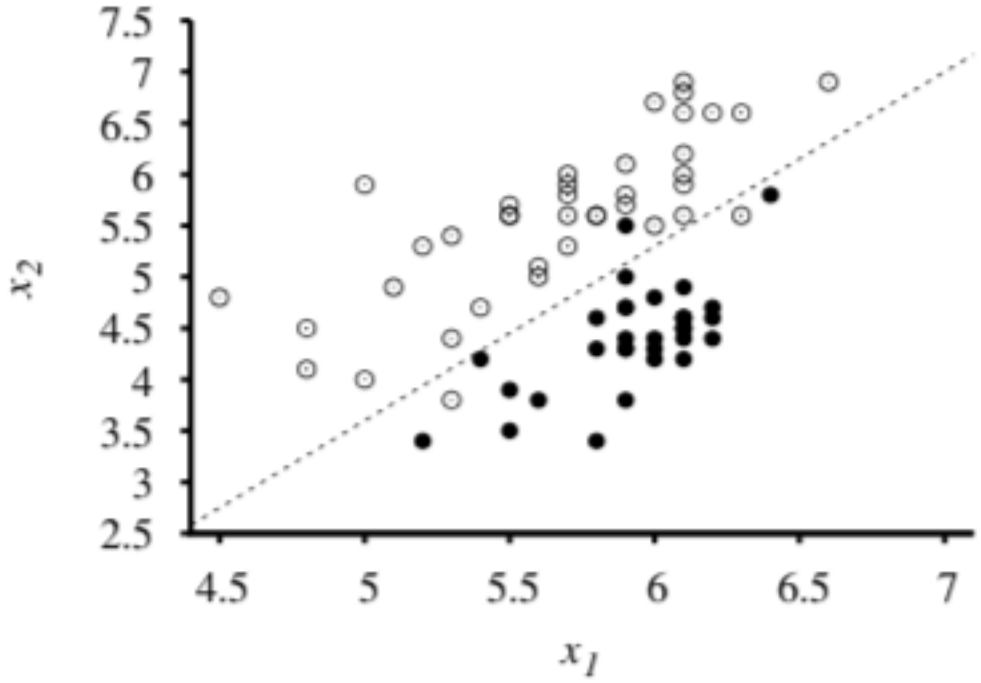
https://en.wikipedia.org/wiki/Perceptron#:~:text=The%20first%20hardware%20implementation%20was,demonstrated%20on%2023%20June%201960.

Perceptron

Case: Linearly-separable

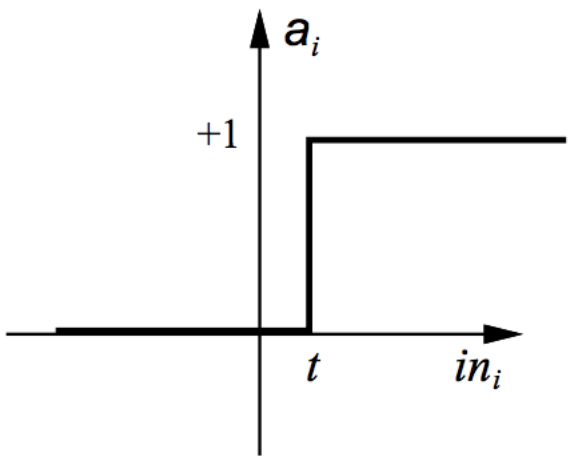


Case: Not Linearly-separable

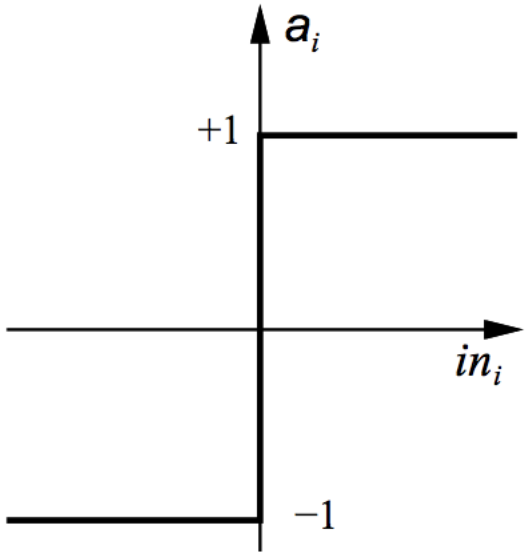


Perceptron learning rule converges to a perfect linear separator when the data points are linearly separable. Problem when there is non-separable data.

Threshold Functions



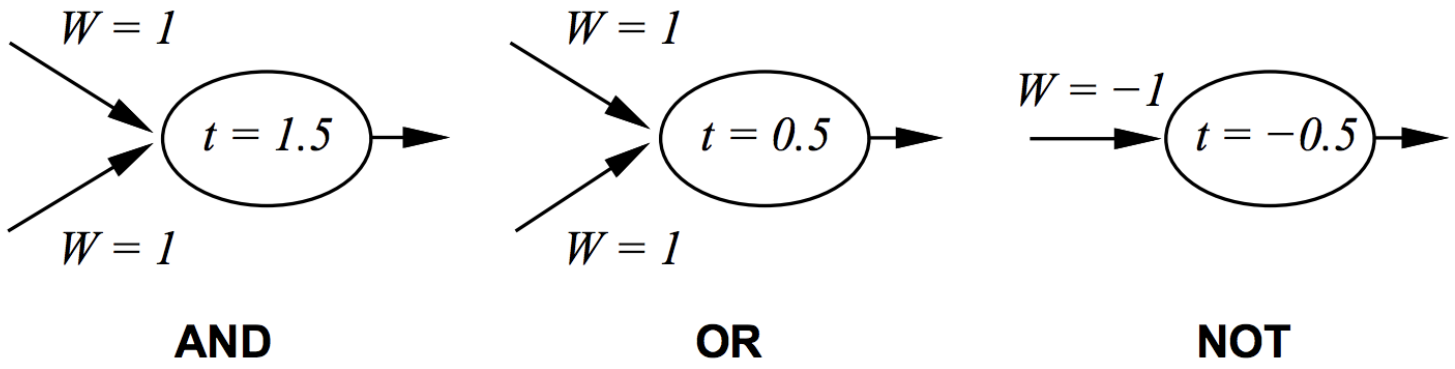
(a) Step function

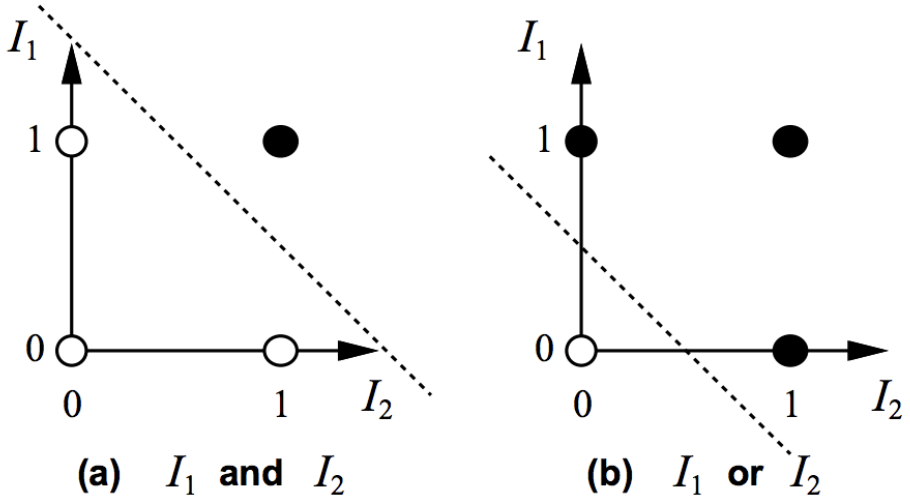


(b) Sign function

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

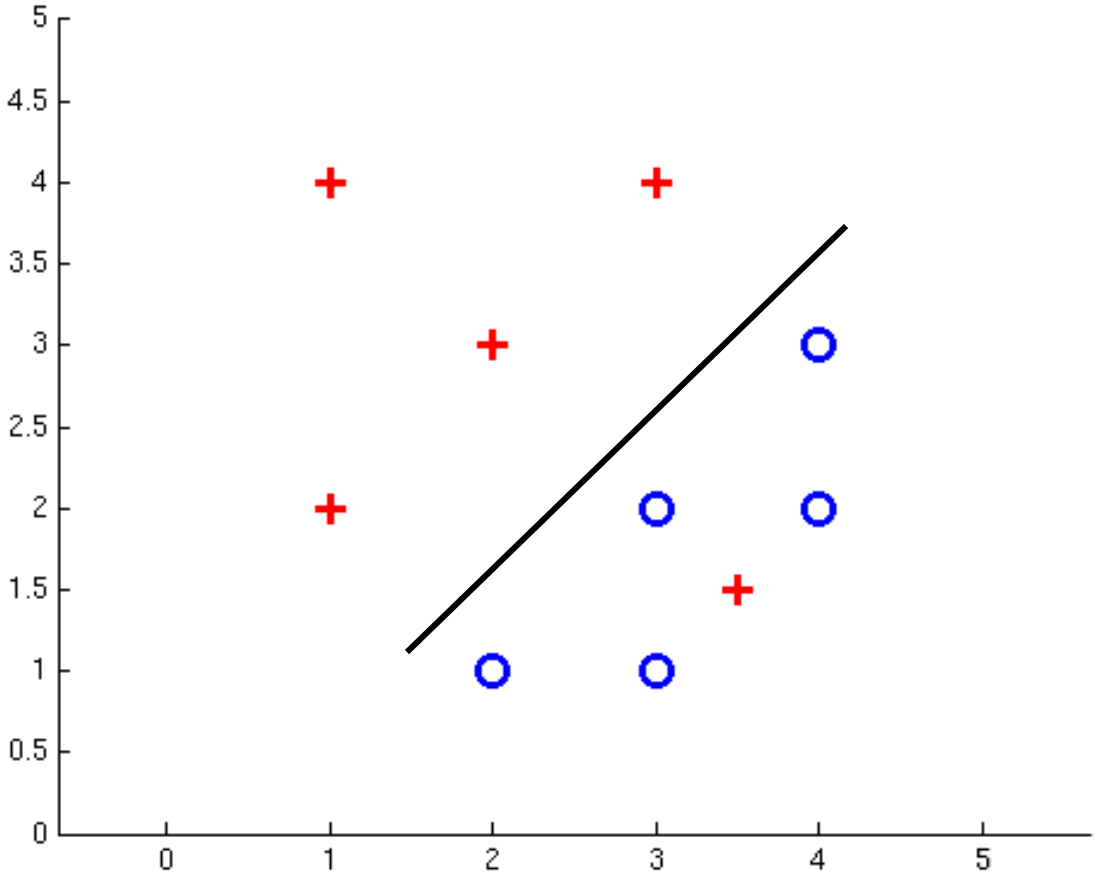
Boolean Functions and Perceptron



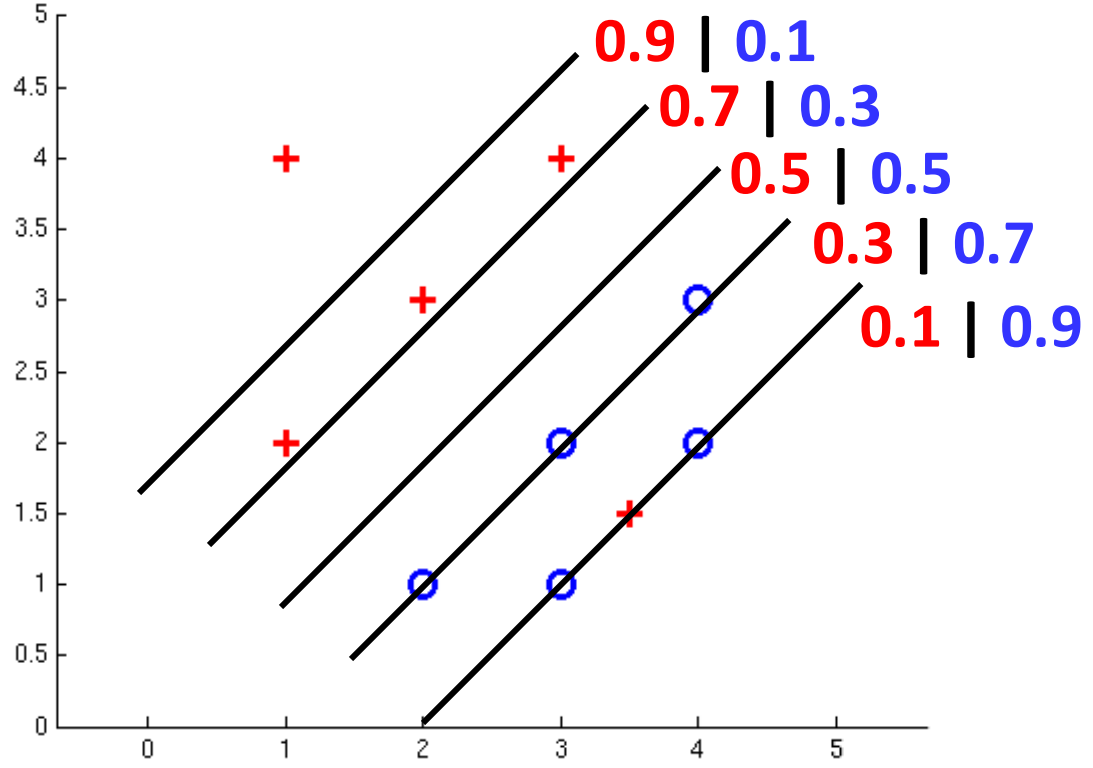


Non-separable case

Deterministic Decisions



Probabilistic Decisions

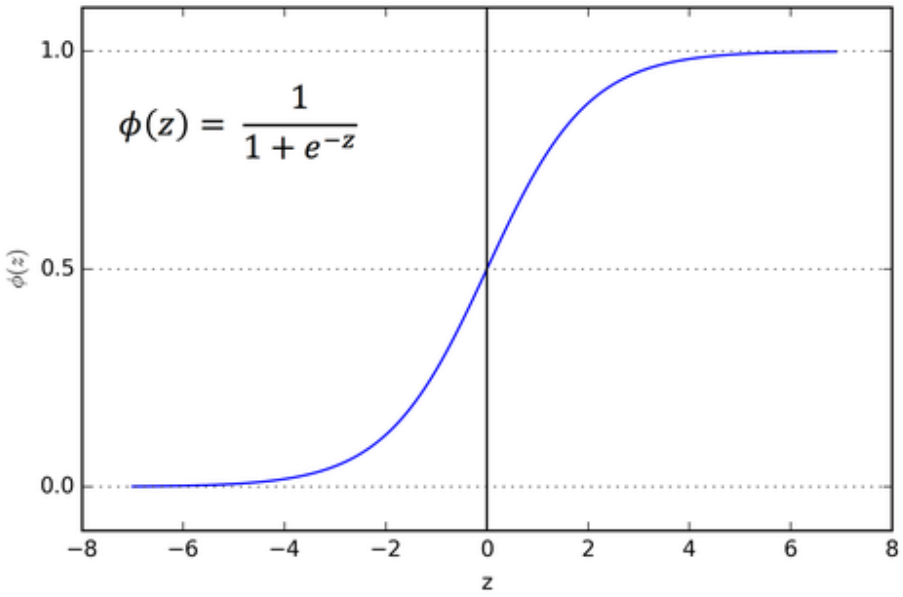


Logistic Output

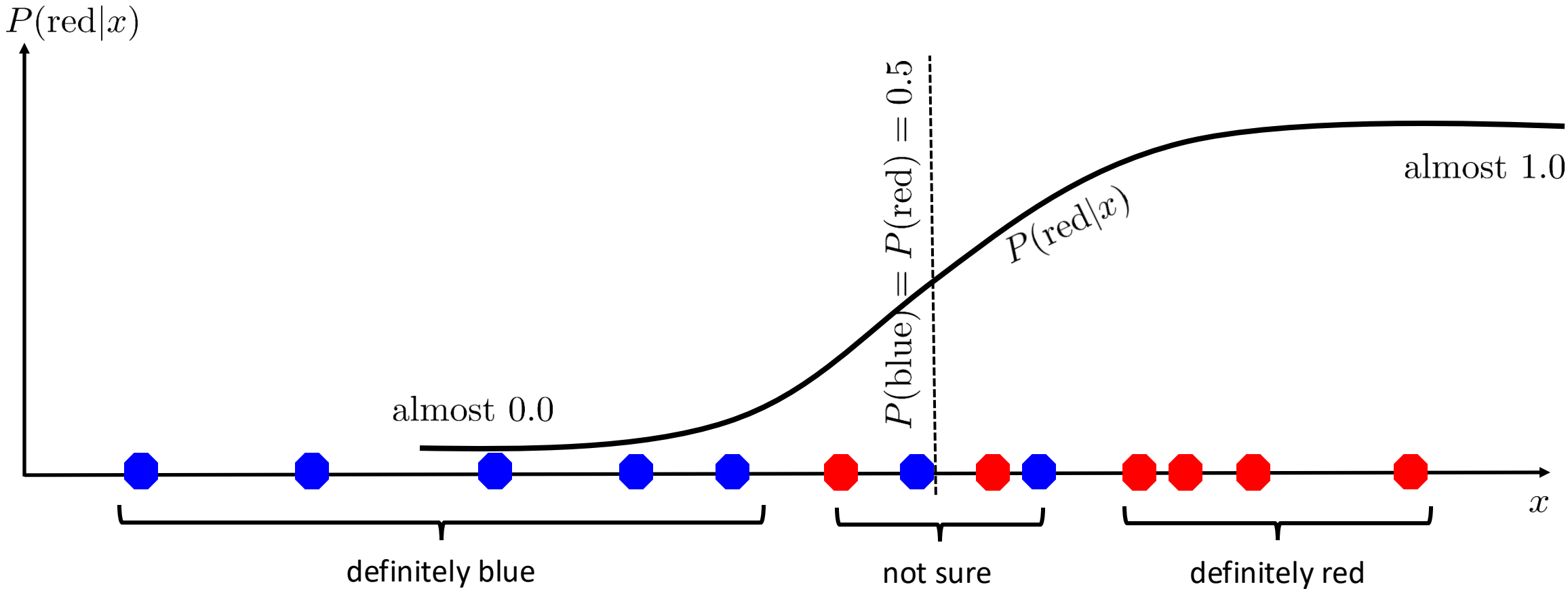
- **Logistic Function**
  - Very positive values. Probability -> 1
  - Very negative values. Probability -> 0.
  - Makes the prediction. Converts to a probability
  - Softens the decision boundary.
- **Logistic Regression**
  - Fitting the weights of this model to minimize loss on a data set is called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

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

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

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

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.

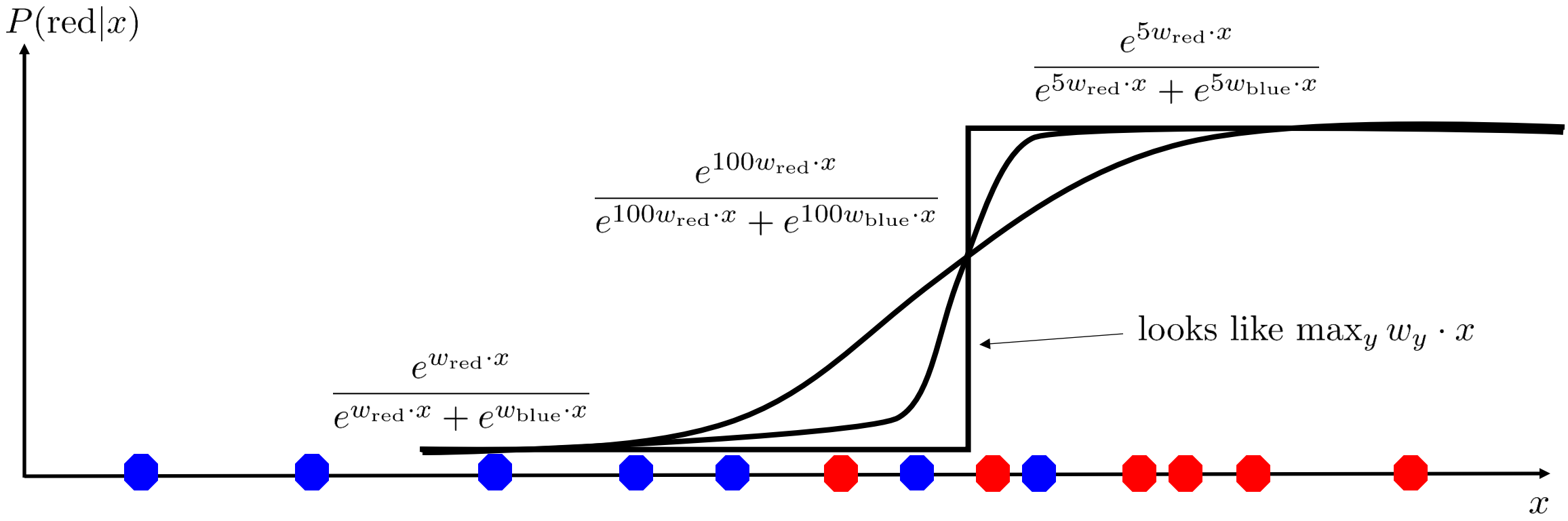
Prediction of the unnormalized probabilities.

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

Exponentiate and normalize the values.

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

Softmax Example



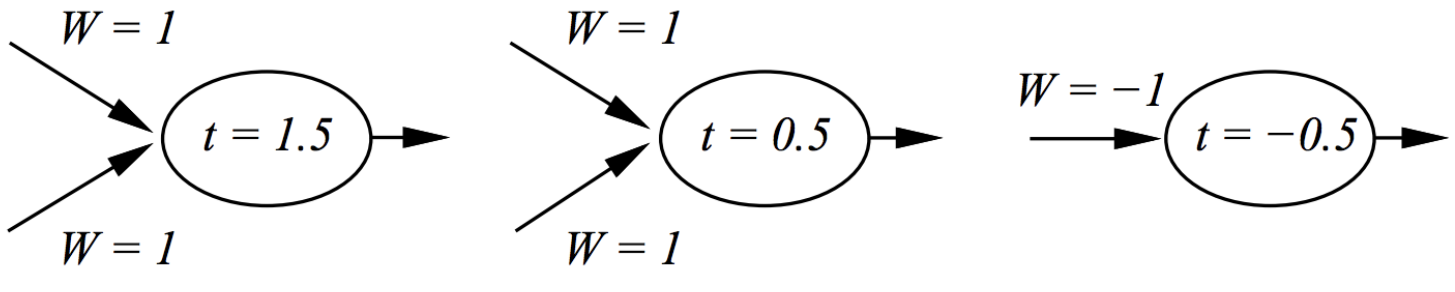
$$P(\text{red}|x) = \frac{e^{w_{\text{red}}\cdot x}}{e^{w_{\text{red}}\cdot x} + e^{w_{\text{blue}}\cdot x}}$$

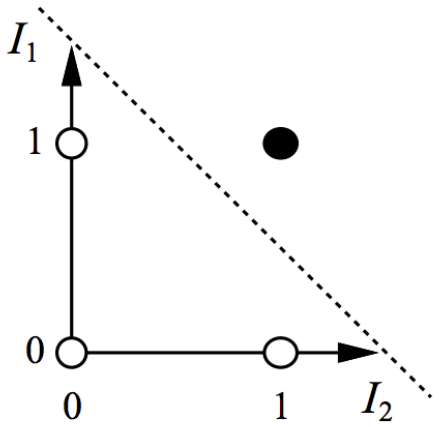
Multi-class Setting

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

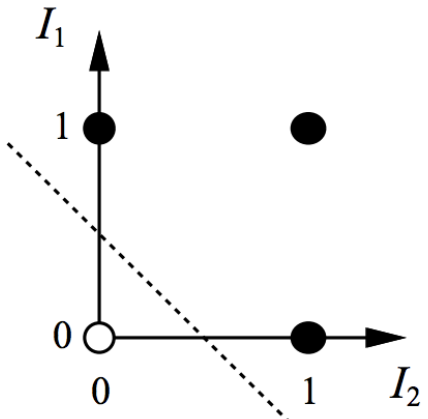
$$P(y^{(i)}|x^{(i)}; w) = \frac{e^{w_{y^{(i)}} \cdot f(x^{(i)})}}{\sum_y e^{w_y \cdot f(x^{(i)})}}$$

Can a perceptron learn XOR?

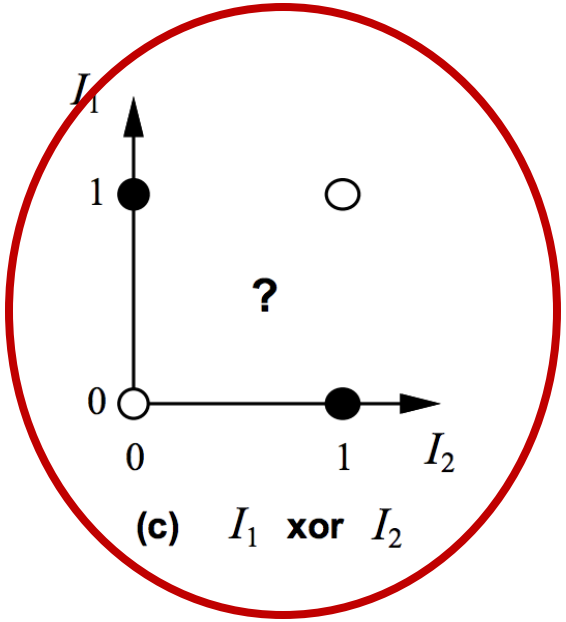




(a) I_1 and I_2



(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 wher the training set is separable.
- Need a non-linear function to describe the features.
- Applying a non-linear kernel map. Affine transformation.

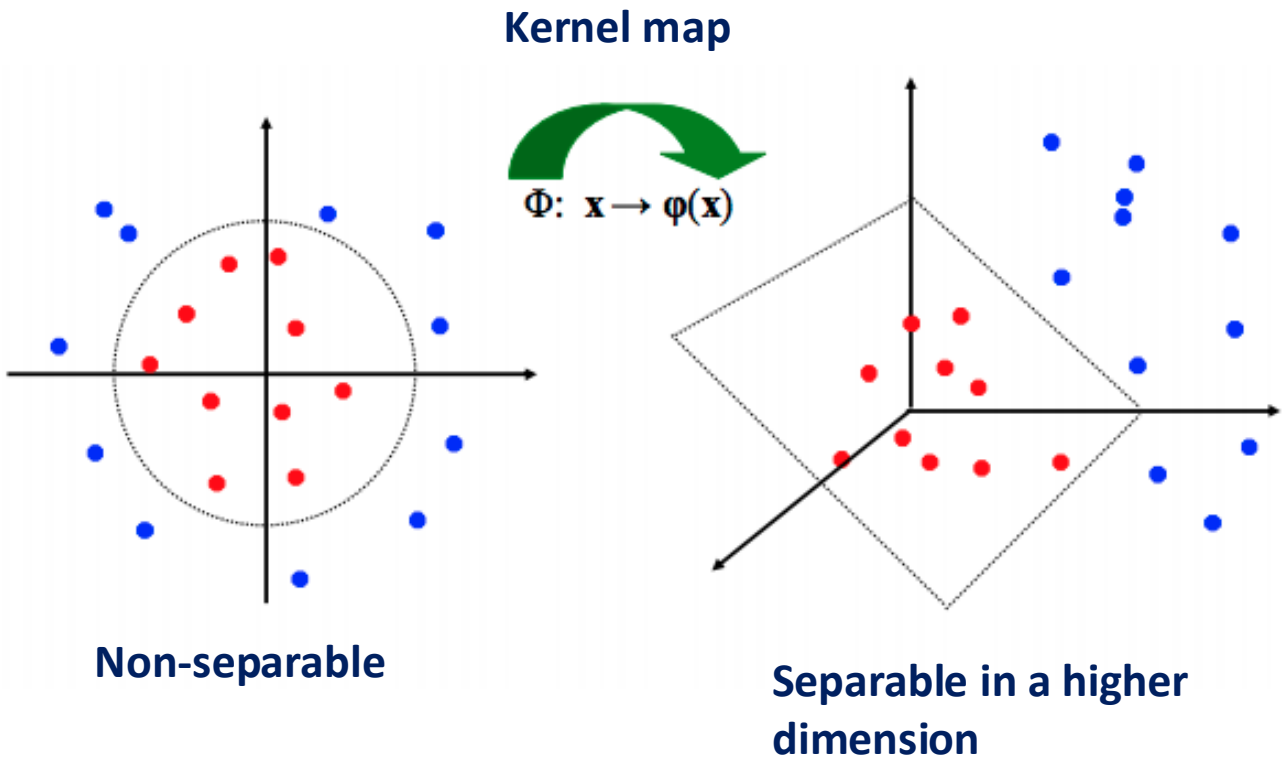
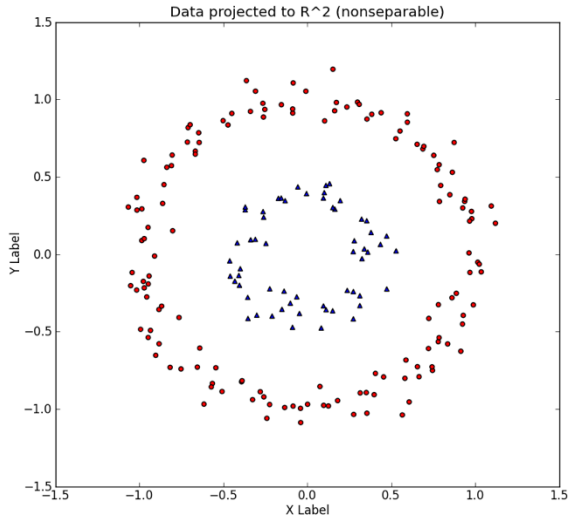
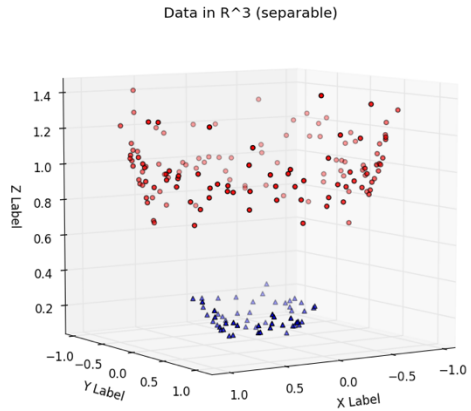


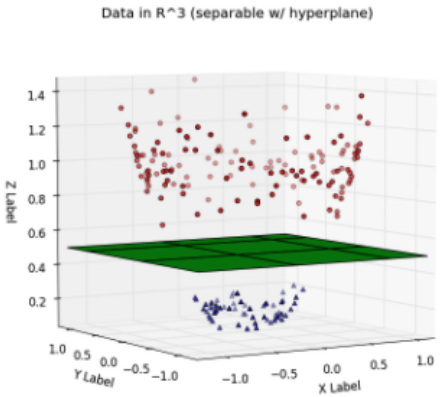
Figure from Ray Mooney

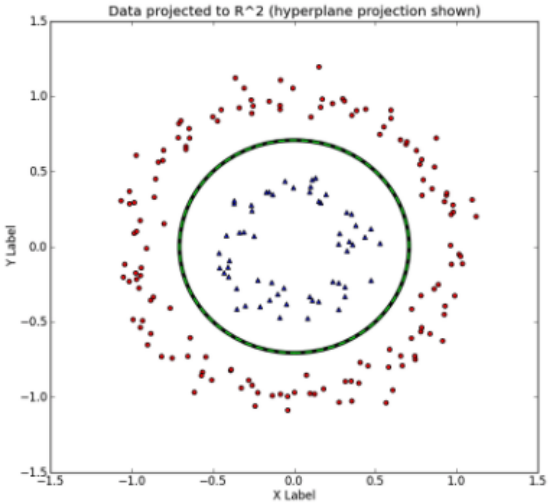
Example: Kernel Map



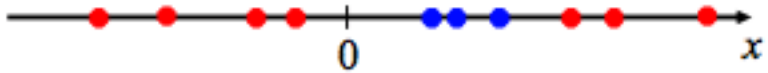


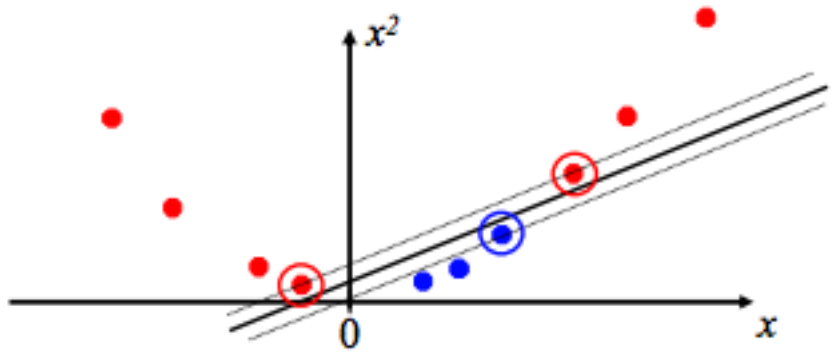
Left) A dataset in $\mathbb{R}^2$, not linearly separable. (Right) The same dataset transformed by the transformation:
$[x_1, x_2] = [x_1, x_2, {x_1}^2 + {x_2}^2]$.





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.

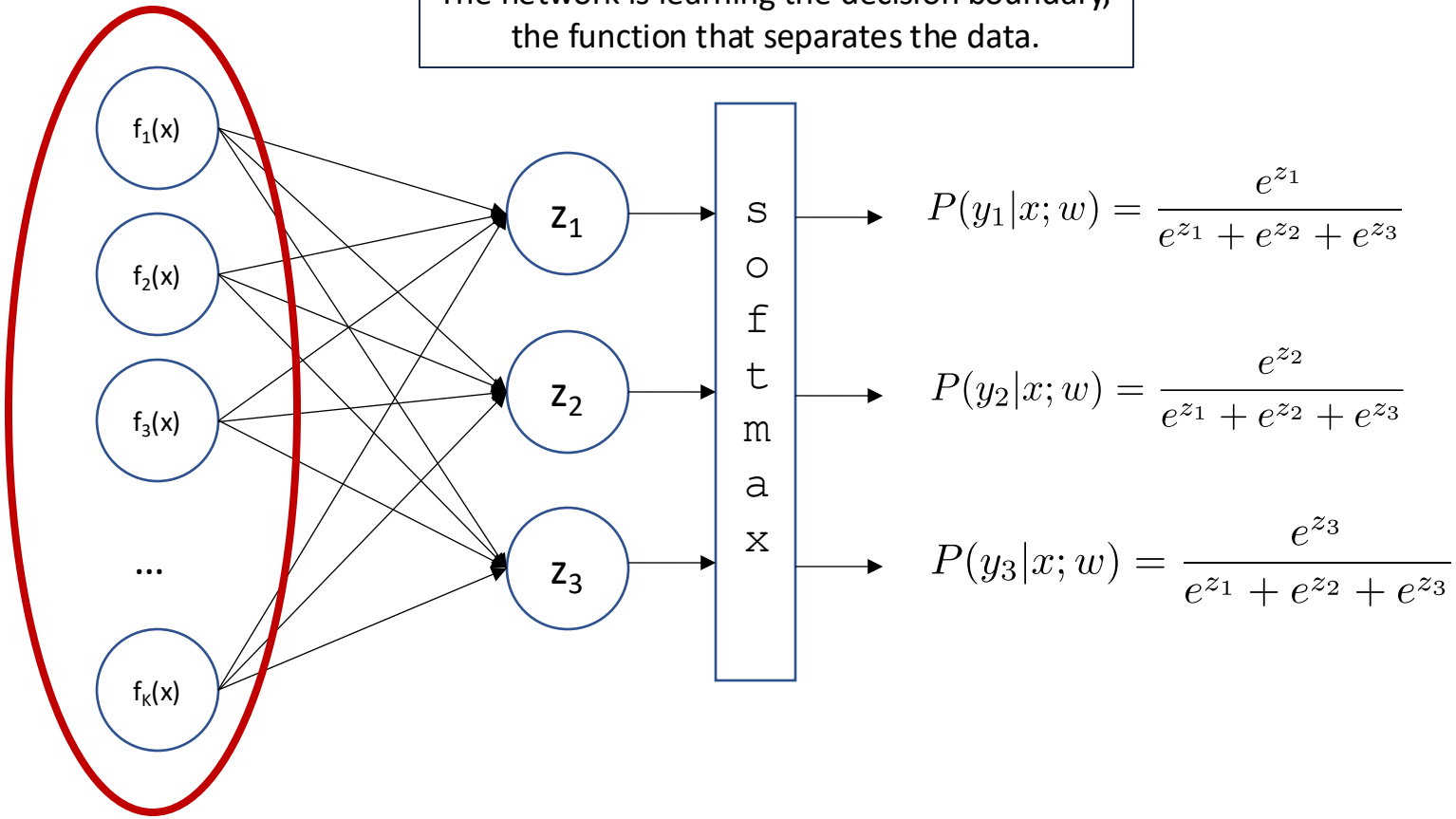




Features ``represent" an input data point

The network is learning the decision boundary, the function that separates the data.

Data point as a vector of features as input.

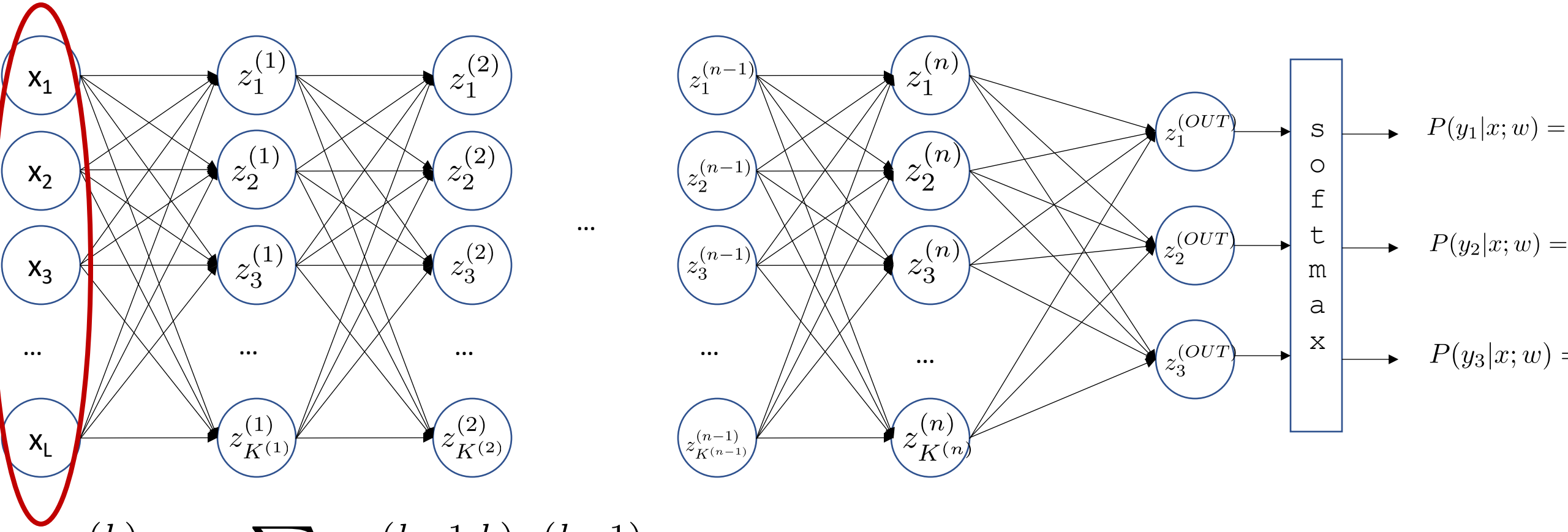


Output: class probabilities, in general the output depends on the machine learning task we are performing.

- We are designing features that represent a data point. We hope that the feature represent enables good classification performance (requires hand-crafting).
- As we will see, a neural network "learns" features to represent a data point for a machine learning task.

“Deep” Neural Networks

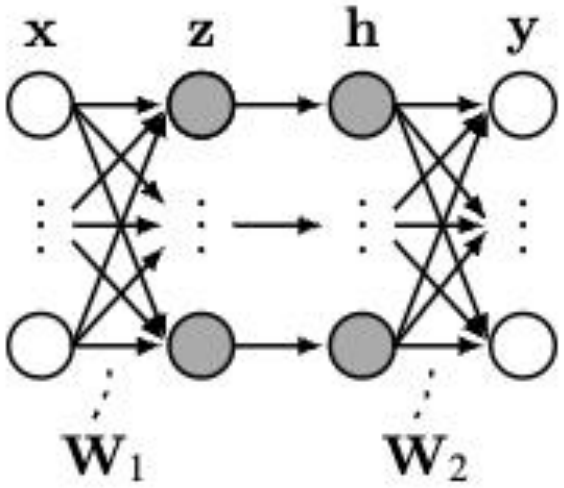
- Connect input data to output. Structure these models by composing many units.
- Feature learning is implicit in the network.
- Paradigm is called *deep learning.*



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

g = nonlinear activation function

Multi-layer Perceptrons (MLPs)

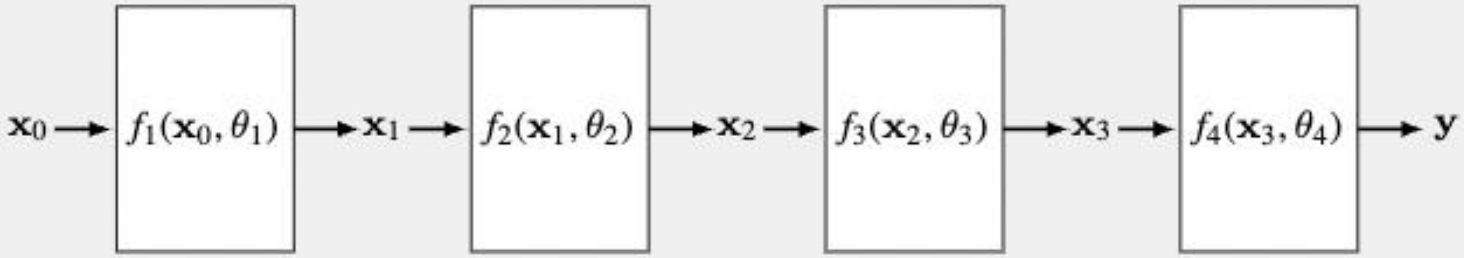


$$\mathbf{z} = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1 \quad / \text{ linear layer}$$

$$\mathbf{h} = g(\mathbf{z}) \quad / \text{ activation function}$$

$$\mathbf{y} = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2 \quad / \text{ linear layer}$$

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

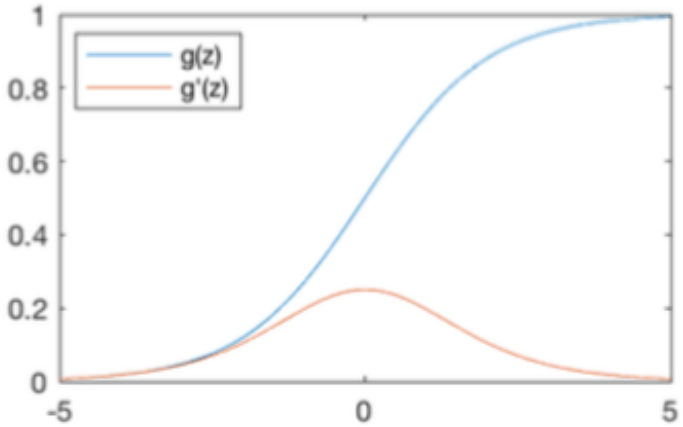


$$\mathbf{x}_{l+1} = f_{l+1}(\mathbf{x}_l, \theta_{l+1})$$

Common for networks to be “deep” -> Leads to expressive models.

Activation Functions

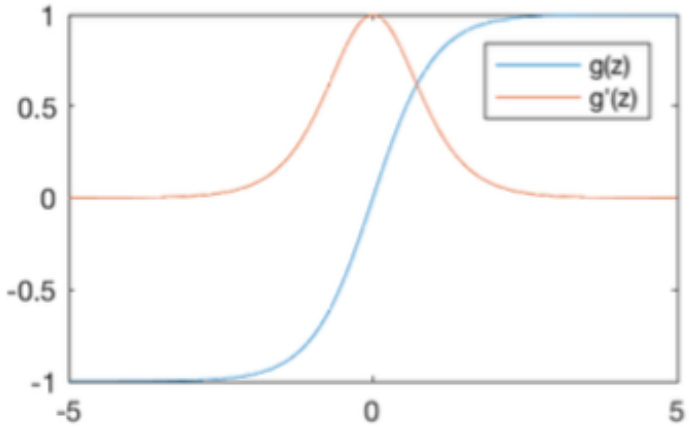
Sigmoid Function



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

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

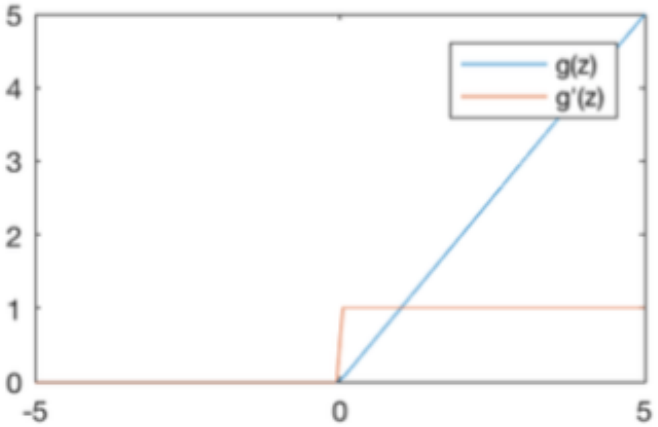
Hyperbolic Tangent



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

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

Rectified Linear Unit (ReLU)



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

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

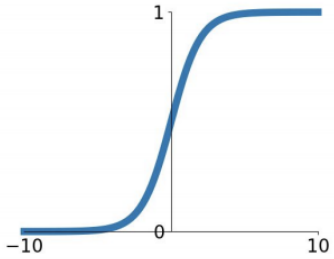
[source: MIT 6.S191 introtodeeplearning.com]

Note: Activations introduce non-linearity. Each unit ”knows” its derivative. Used later to compute gradients automatically.

Common Activation Functions

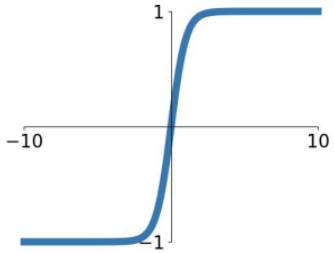
Sigmoid

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



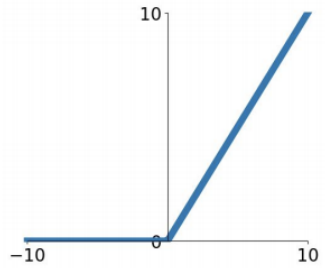
tanh

$$\tanh(x)$$



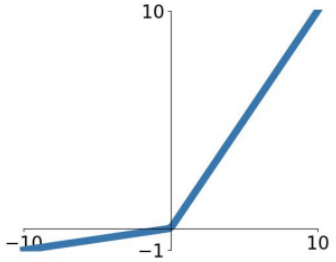
ReLU

$$\max(0, x)$$



Leaky ReLU

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

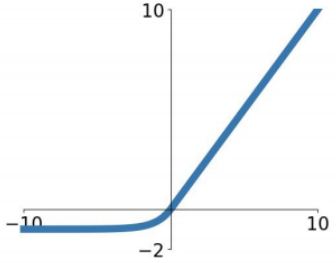


Maxout

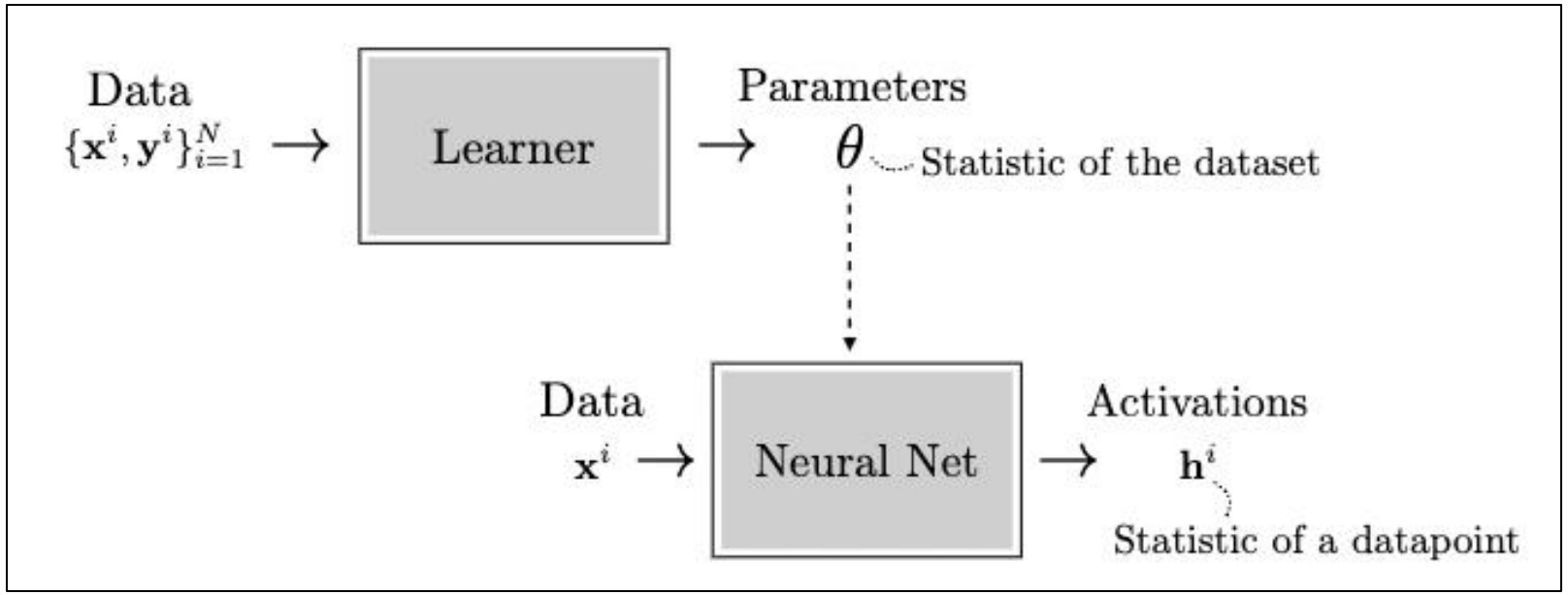
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

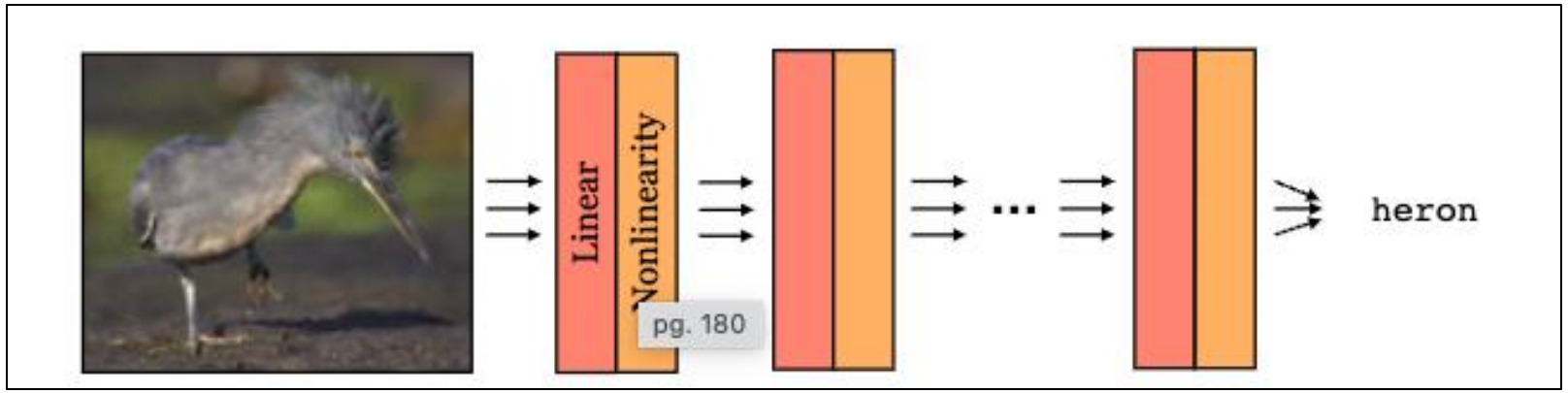
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Deep Neural Networks: *In essence*



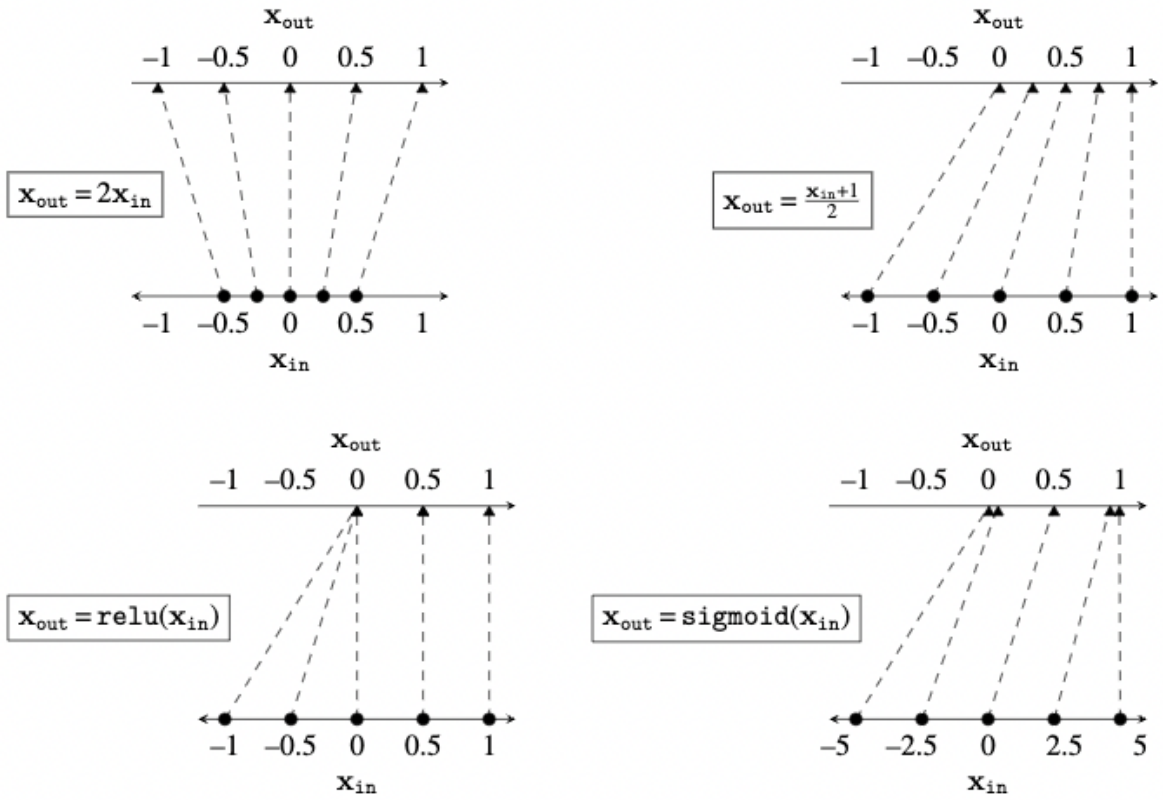
- During training one learns the parameters of the network.
- At inference time, directly apply the weights to form the predication.



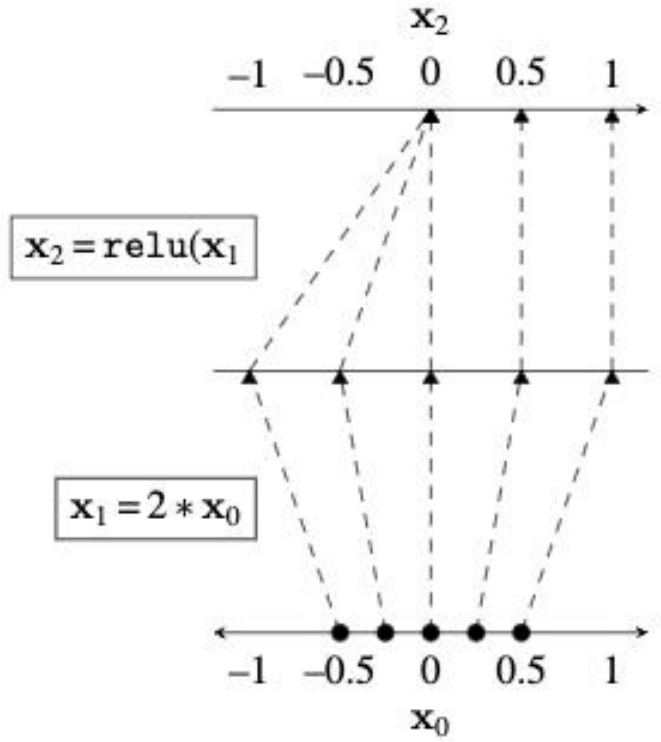
- Deep nets consist of linear layers interleaved with non-linearities.

Neural Networks are Distribution Transformers

What does a long sequence of linear and non-linear function compositions lead to?



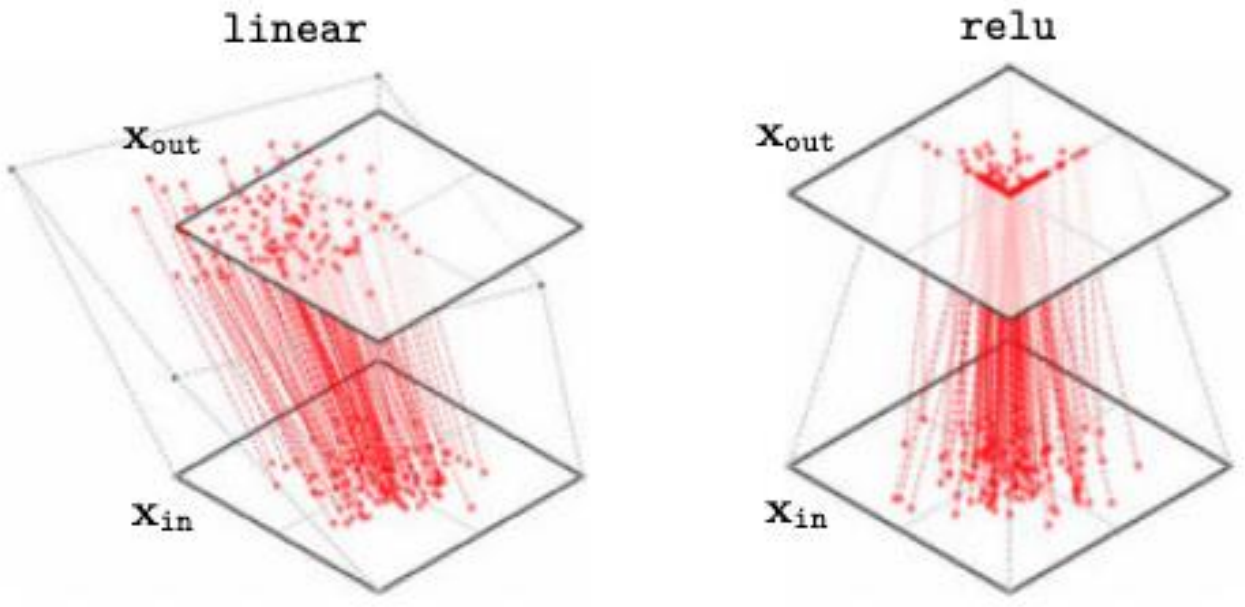
Mapping plots for simple functions that can be used in neural layers



Mapping plot for a linear-relu composition

Mapping diagrams for a data distribution

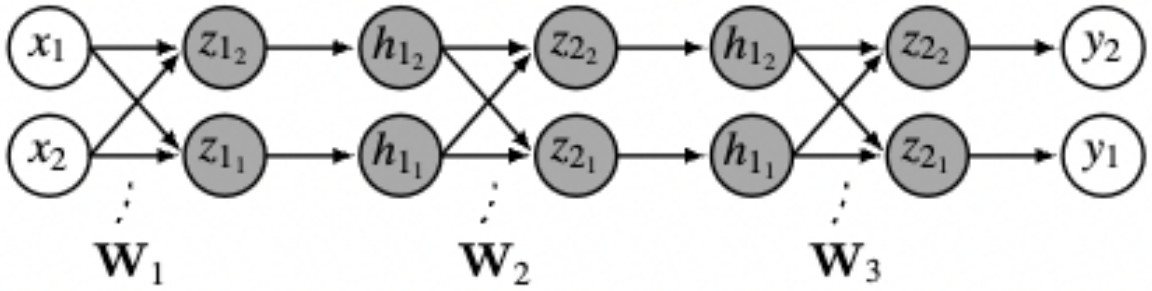
2D mapping diagram for several neural layers.



- 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.

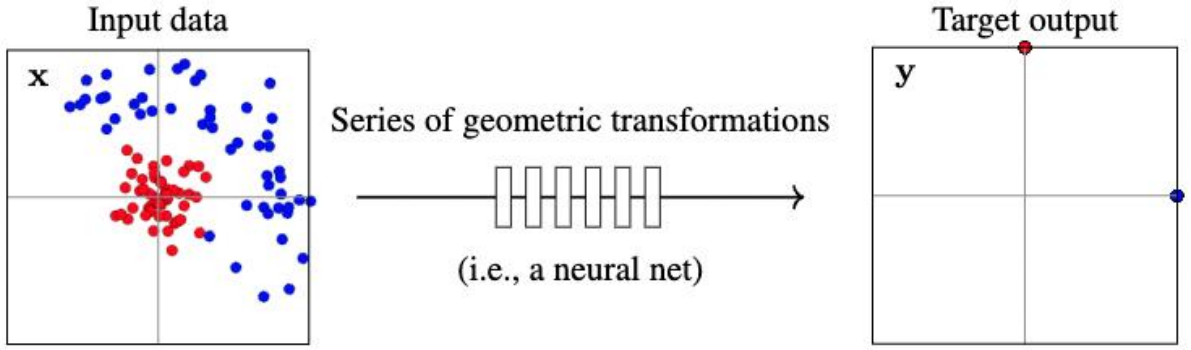
Binary Classification Example

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



Linear-relu-linear-relu-linear architecture

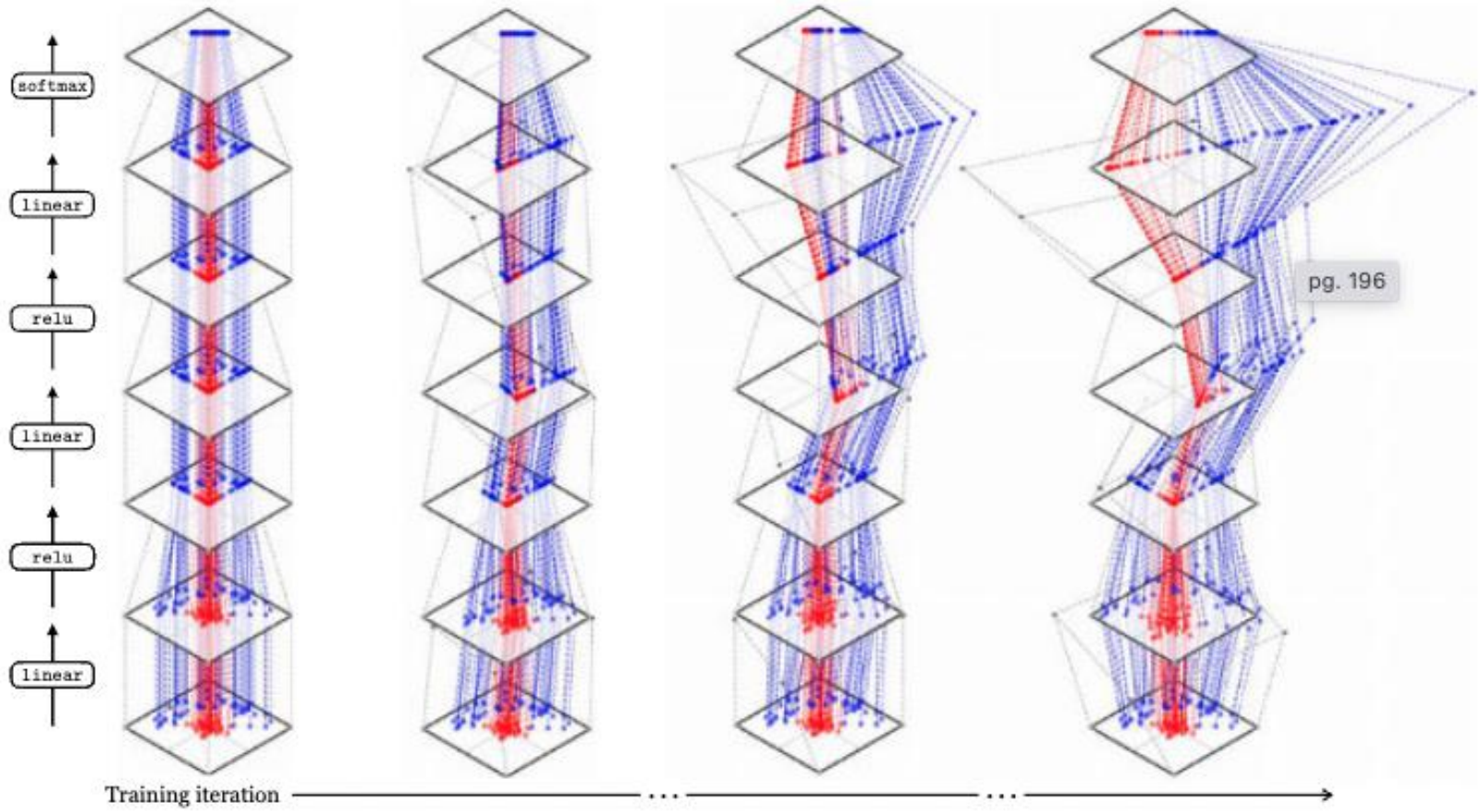
$$\mathbf{z}_1 = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1 \quad / \quad \texttt{linear}$$
$$\mathbf{h}_1 = \texttt{relu}(\mathbf{z}_1) \quad / \quad \texttt{relu}$$
$$\mathbf{z}_2 = \mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2 \quad / \quad \texttt{linear}$$
$$\mathbf{h}_2 = \texttt{relu}(\mathbf{z}_2) \quad / \quad \texttt{relu}$$
$$\mathbf{z}_3 = \mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3 \quad / \quad \texttt{linear}$$
$$\mathbf{y} = \texttt{softmax}(\mathbf{z}_3) \quad / \quad \texttt{softmax}$$



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

Goal of neural network classifier is to arrange the input data distribution to match the target label distribution (two points in the right image.)

Remapping input data layer by layer

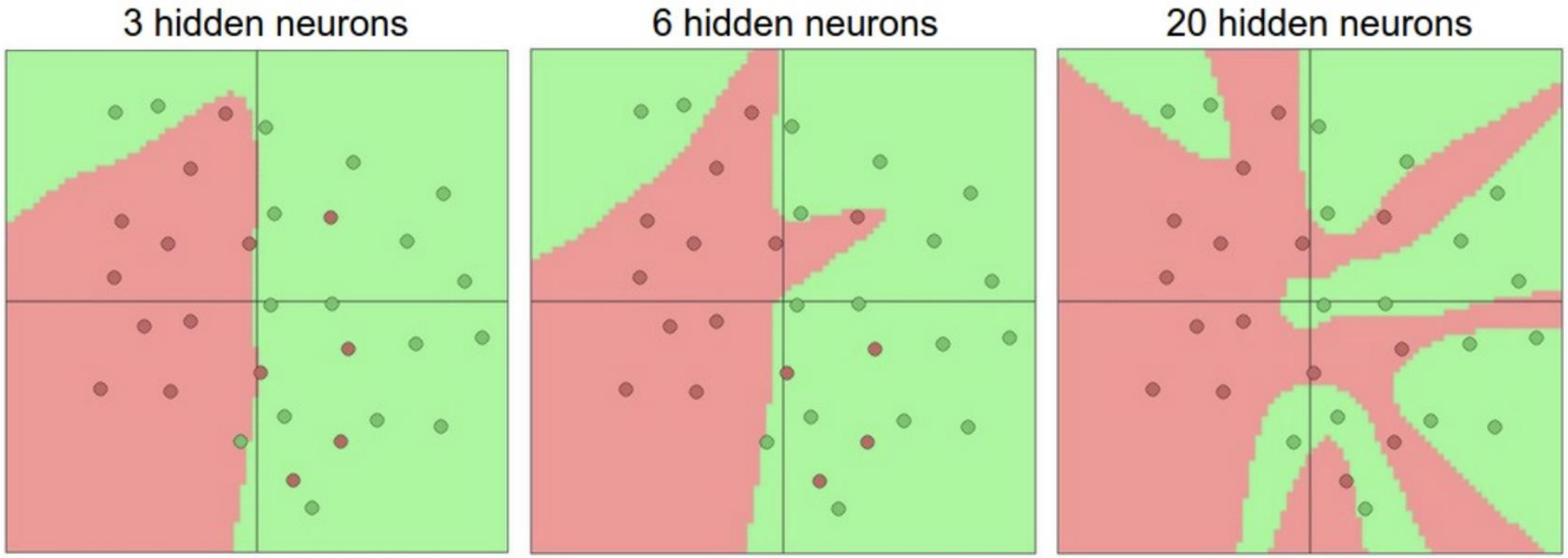


Target output moves the red and blue points closer to the ground truth data.
As training progresses the network gradually achieves this separation.

The outputs of the final layers before classification can be considered as learned features for the data.

Representing 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&regDataset=reg-plane&learningRate=0.03&regularizationRate=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

Learning XOR

XOR is not linearly separable.

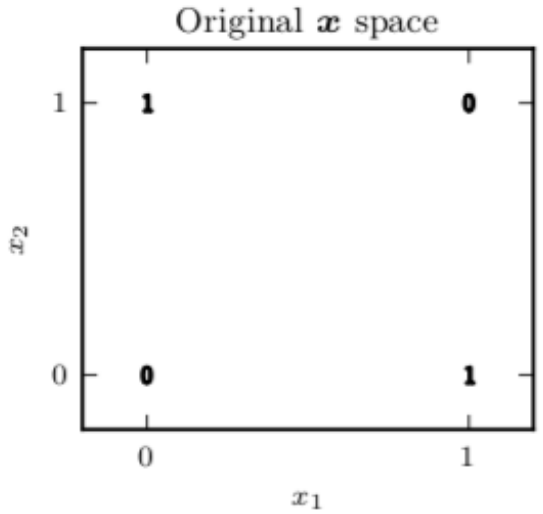


Figure 6.1, left

Rectified Linear Activation

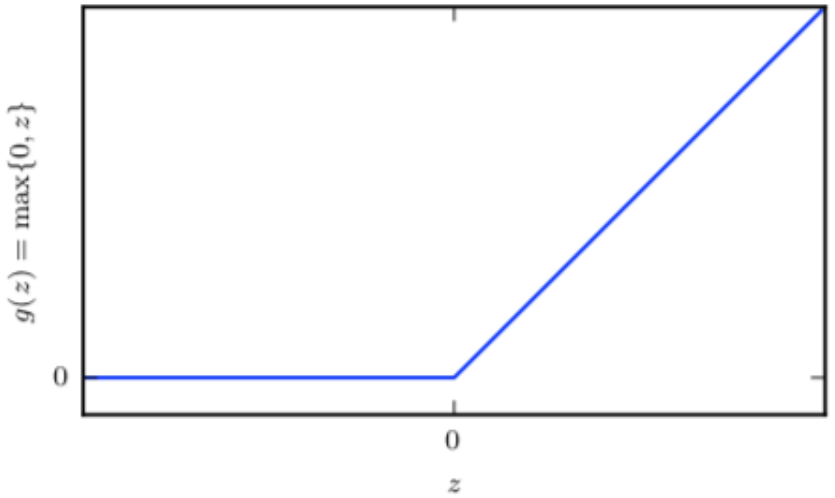


Figure 6.3

Network Diagram

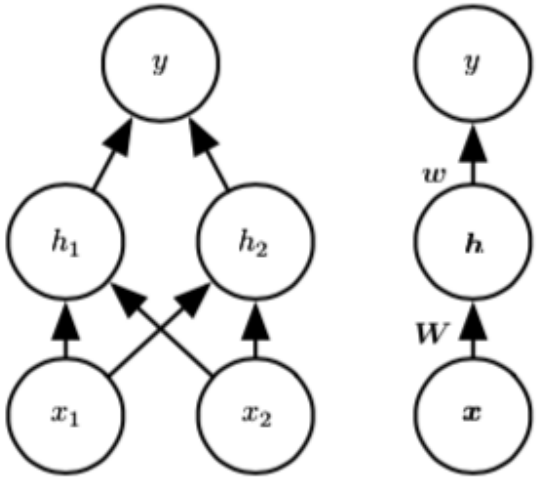


Figure 6.2

More compact representation

Example from Ch 6, DL Book

Learning XOR

Network Diagram

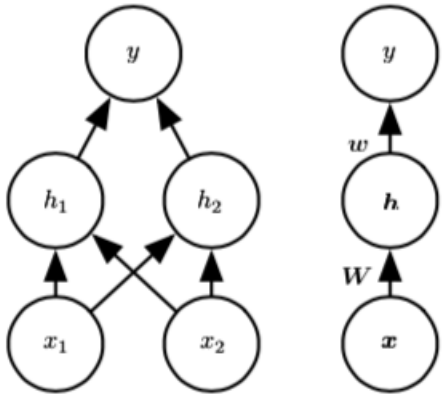


Figure 6.2

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

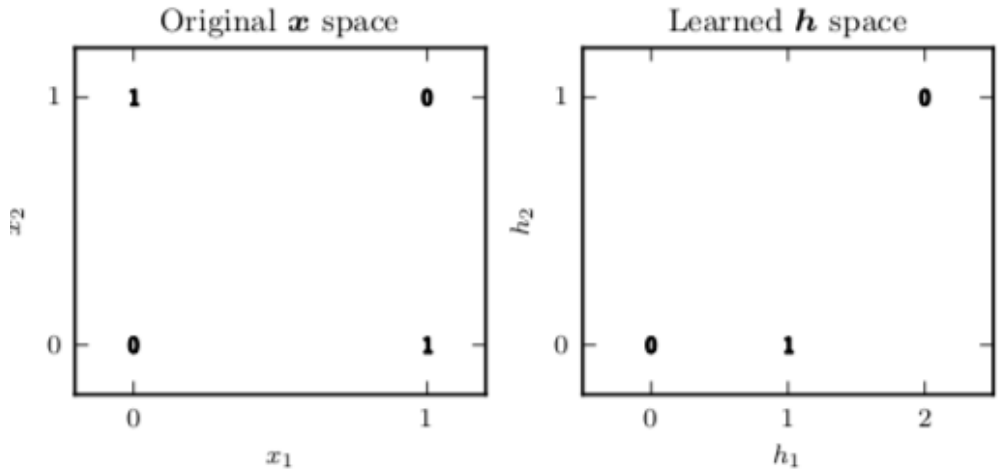


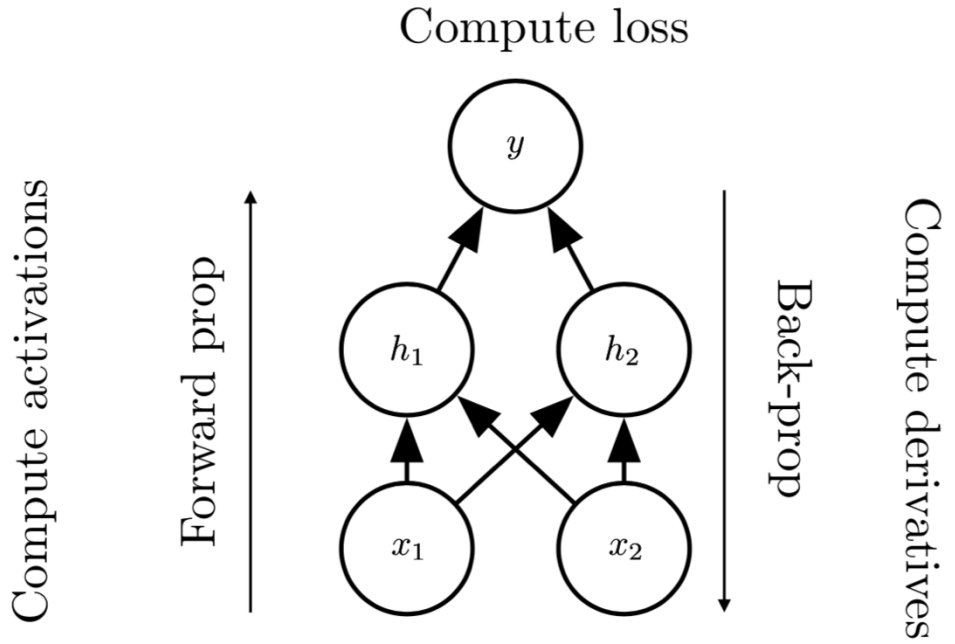
Figure 6.1

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

Backpropagation and Computation Graphs

- Training NNs
  - 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?*



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

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

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

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.

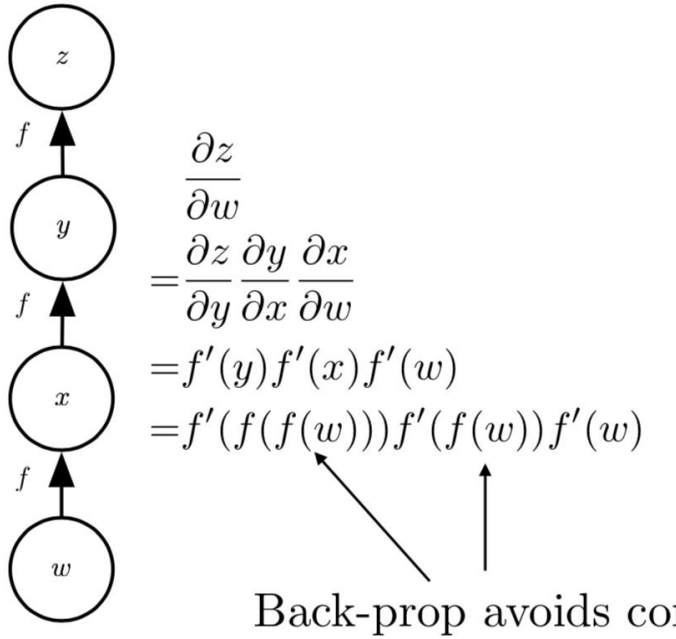


Figure 6.9

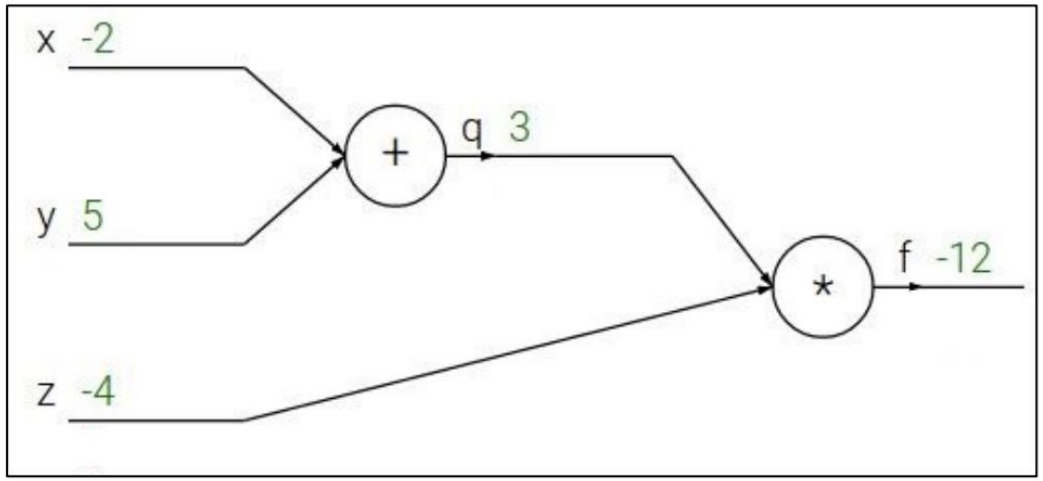
Material from Ch 6, DL Book

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

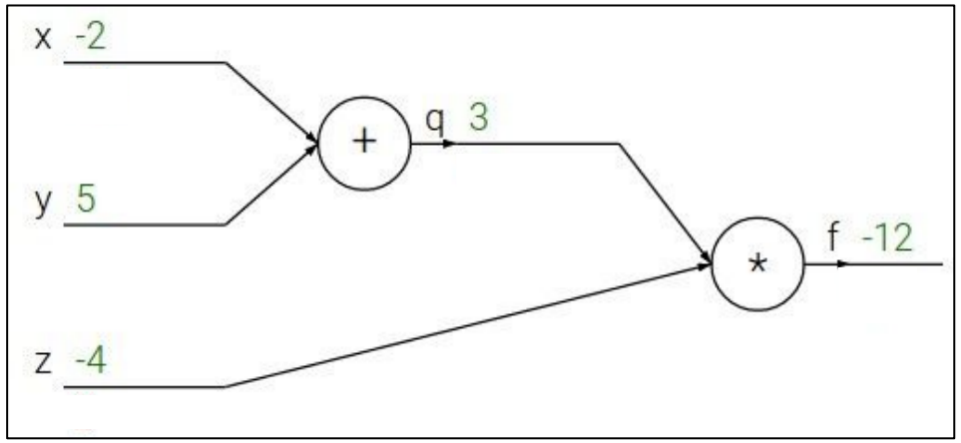
$$f(x, y, z) = (x + y)z$$

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

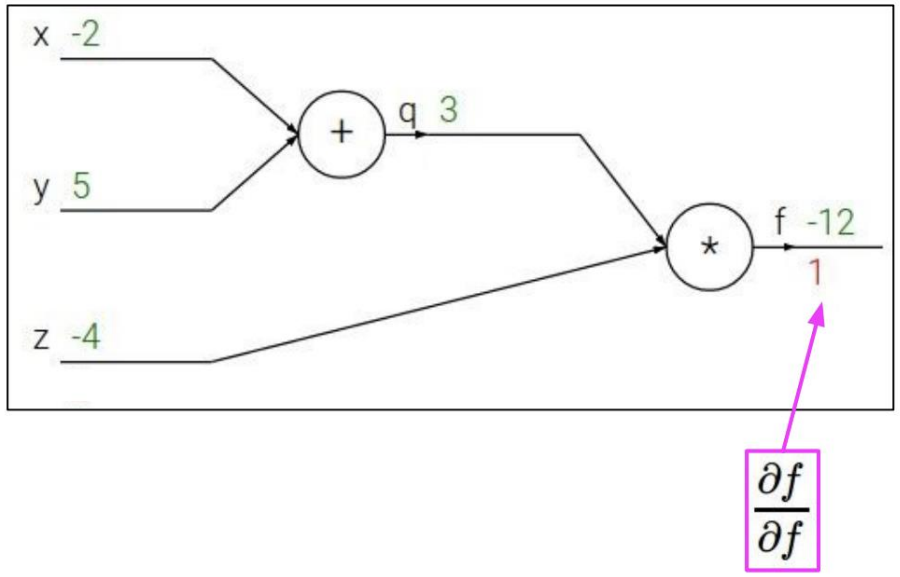
$$f(x, y, z) = (x + y)z$$

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



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

Backpropagation: Toy Example

Backpropagation: a simple example

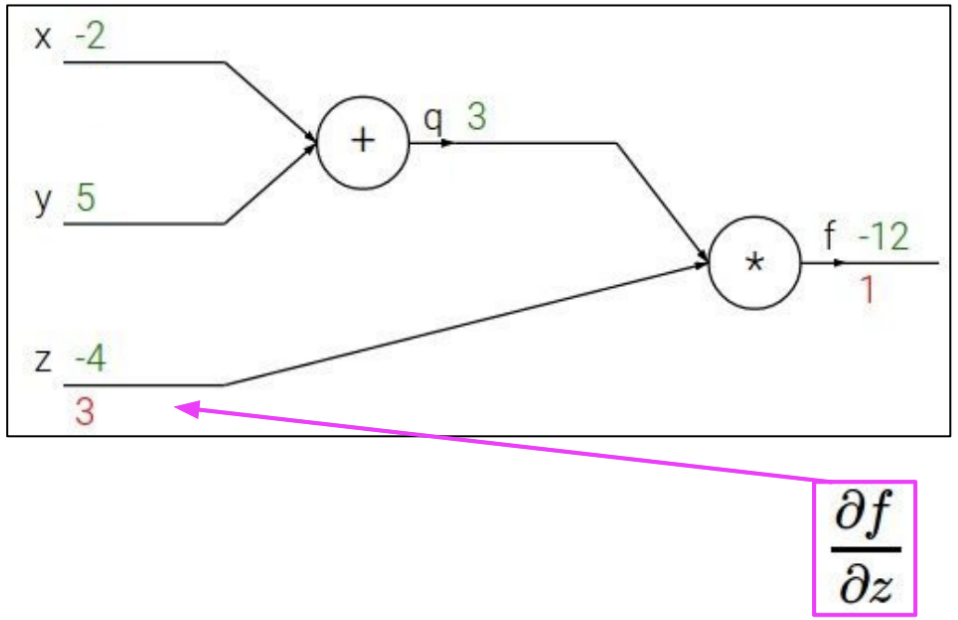
$$f(x, y, z) = (x + y)z$$

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

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

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

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



Backpropagation: Toy Example

Backpropagation: a simple example

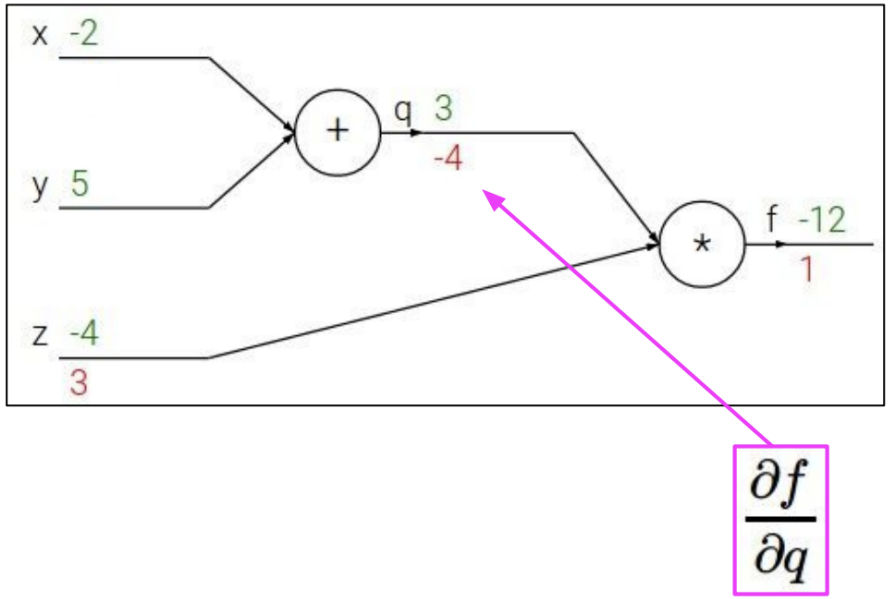
$$f(x, y, z) = (x + y)z$$

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



asdf

Backpropagation: Toy Example

Backpropagation: a simple example

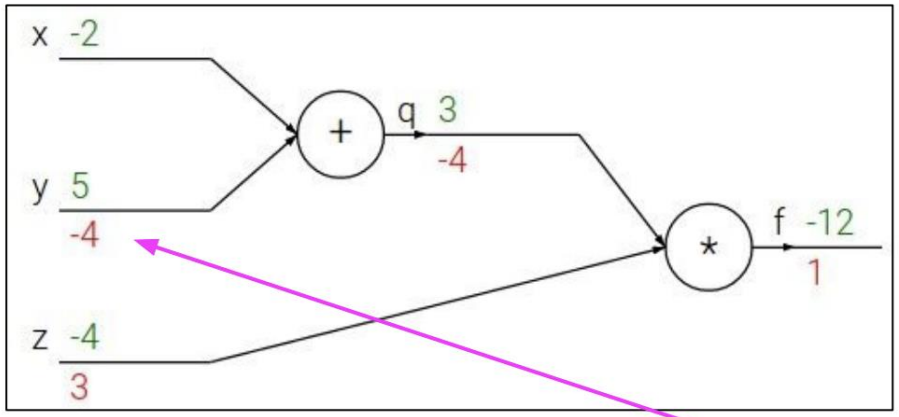
$$f(x, y, z) = (x + y)z$$

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

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

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

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



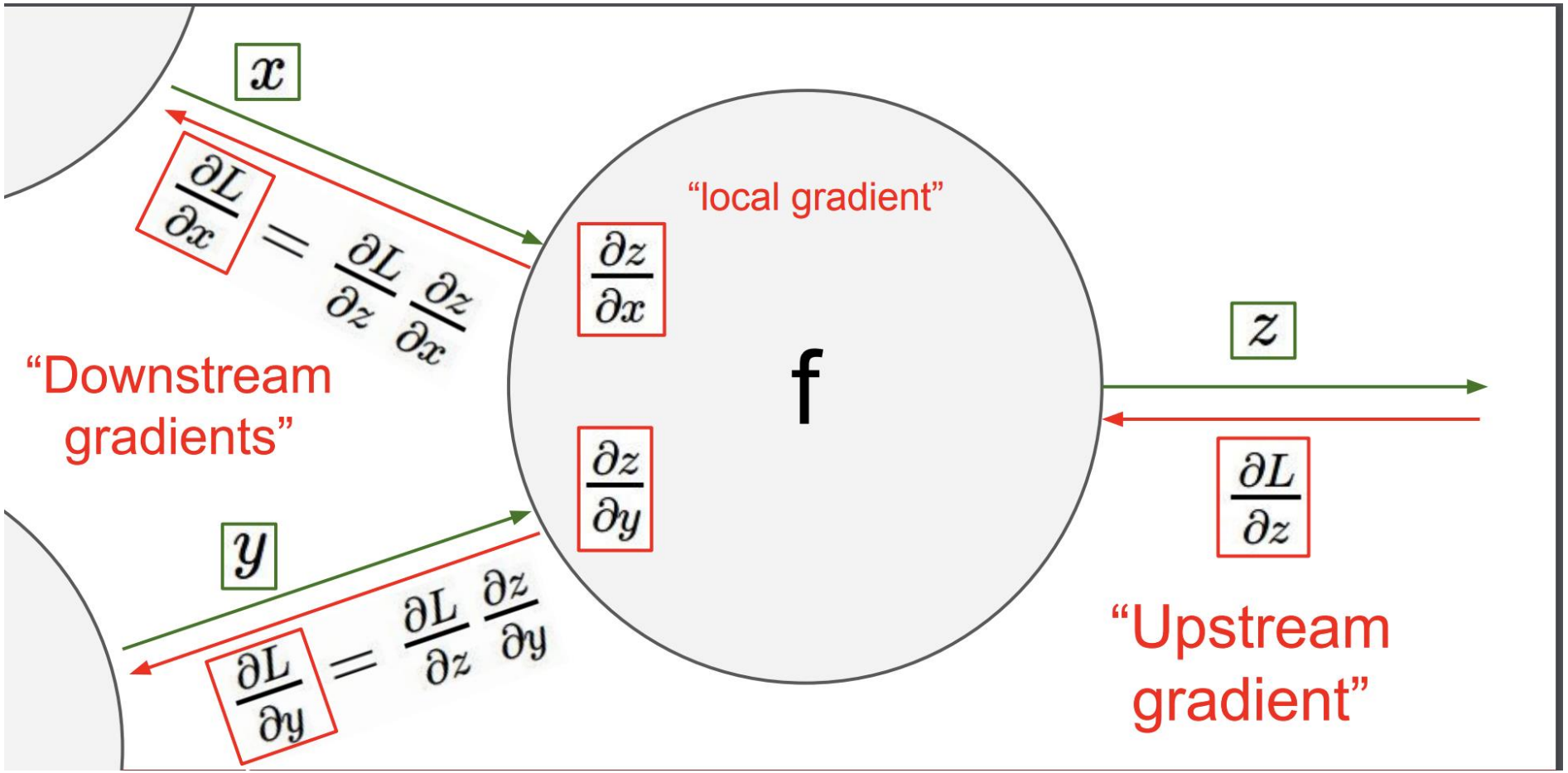
Chain rule:

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

Upstream gradient

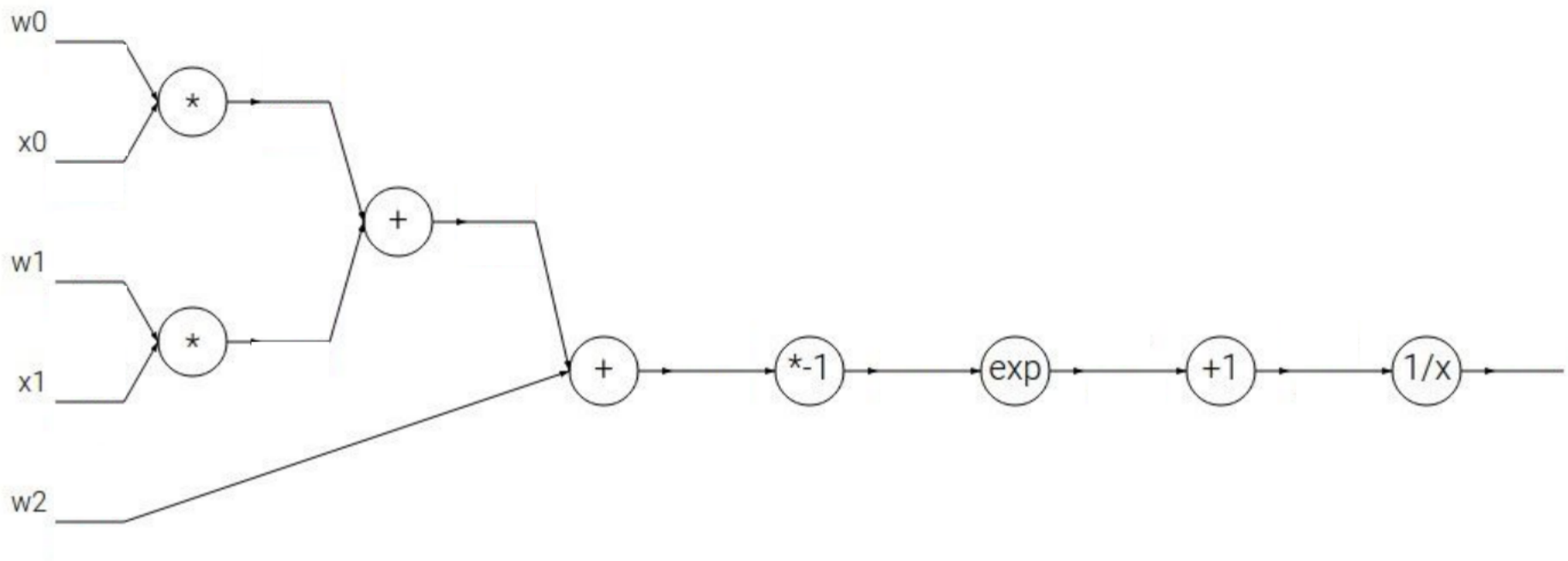
Local gradient

Backpropagation



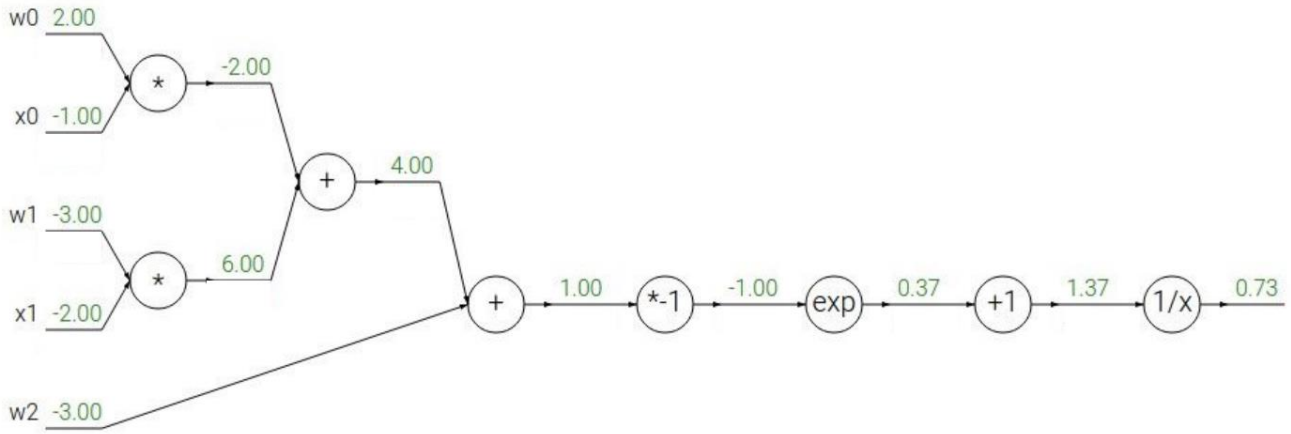
Backpropagation: Example

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



Backpropagation: Example

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



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

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

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

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

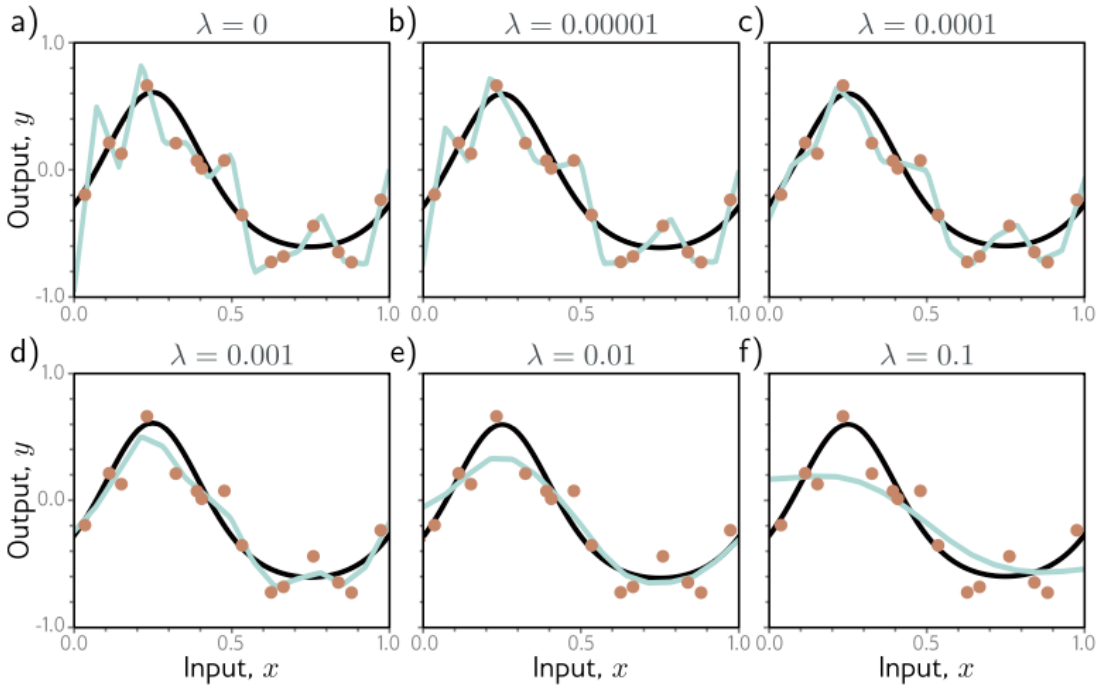
Deep Neural Networks: Highly expressive

- 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.

Preventing Over-fitting: Weight Decay

L2-regularisation

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[\sum_{i=1}^{I} \ell_i[\mathbf{x}_i, \mathbf{y}_i] + \lambda \sum_{j} \phi_j^2 \right]$$



Core Idea:

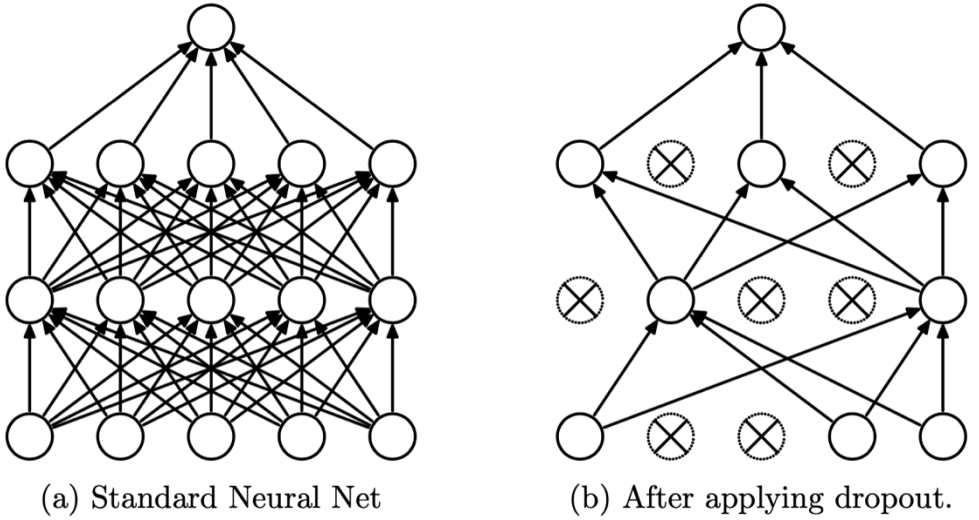
- Encourage smaller weights so that the output function is mother.
- Output is the weighted sum of the previous layer.
- If the input magnitudes are small then the output variation is small.

Preventing Over-fitting: Early Stopping

- Stop the training procedure before convergence.
- Reduces overfitting if the model has already captured the coarse shape of the function before it starts to fit to noise.
- Single hyper-parameter – number of steps before learning is terminated.
  - Chosen empirically using a validation set (without the need to train multiple models)
  - The model is trained once, the performance on the validation set is monitored every T iterations, and the associated parameters are stored. The stored parameters where the validation performance was best are selected.
- Intuition
  - Weights are often initialized to small values. With early stopping they do not get a chance to become large.
  - Activations remain in the linear range without saturating.

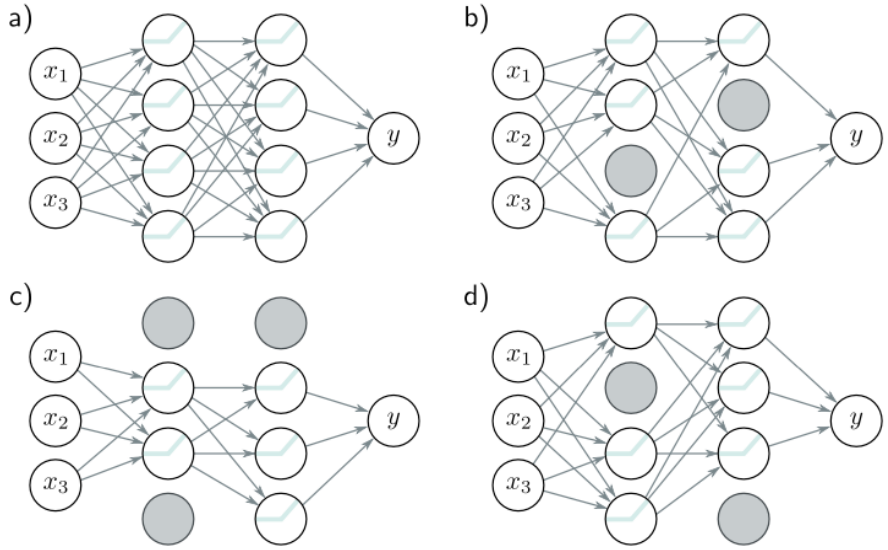
Preventing Over-fitting: Dropouts

- A way to reduce the test set error at the cost of making it hard to fit on the training data.
- Each step of training
  - Dropout applies one step of back prop. To a new version of the network.
  - Created by deactivating a randomly chosen subset of units.
- During training, dropout samples from an exponential number of different “thinned” networks.
- At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single un thinned network that has smaller weights.



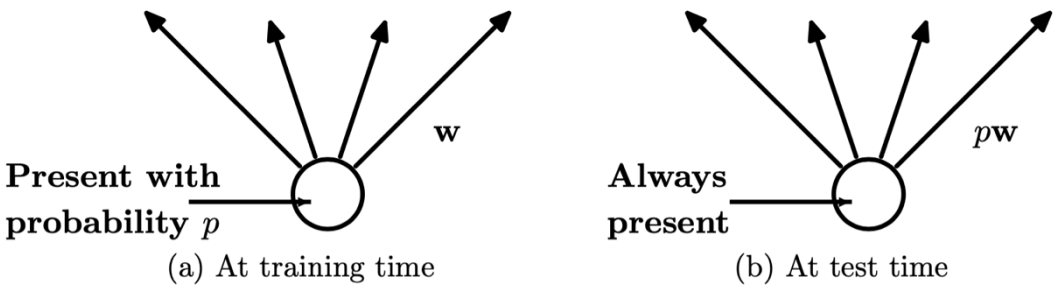
(a) Standard Neural Net

(b) After applying dropout.



Preventing Over-fitting: Dropouts

- If a unit is retained with probability p during training, the outgoing weights of that unit are multiplied by p at test time. This ensures that for any hidden unit the expected output is the same as the actual output at test time.
- 2^n networks with shared weights can be combined into a single neural network to be used at test time.



(a) At training time (b) At test time

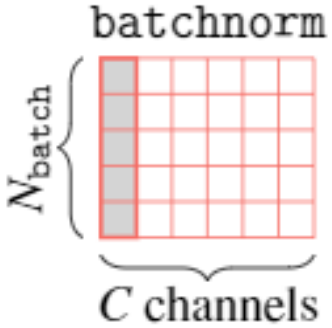
Intuition

- Forces neurons to be useful as a whole paying attention to what others are learning.

https://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

Improving SGD Convergence

- **Batch-normalization**
  - Standardizes each neural activation with respect to its mean and variance.
  - Over the mini-batch of data points.
  - In effect adjusts the distribution observed by the deeper layers from the outputs coming from the earlier layers.
    - A problem called internal covariate shift.
- **L2- Normalization**
  - Projects the inputs onto a unit hypersphere.
  - Useful for bounding activations to unit vectors.
- The above two can be considered as a *non-linear layers* in the neural network.



$$x_{\text{out}}[i] = \gamma \frac{x_{\text{in}}[i] - \mathbb{E}[x_{\text{in}}[i]]}{\sqrt{\text{Var}[x_{\text{in}}[i]]}} + \beta$$

$$x_{\text{out}}[i] = \frac{x_{\text{in}}[i]}{\|x_{\text{in}}\|_2}$$

Neural Networks: Successes