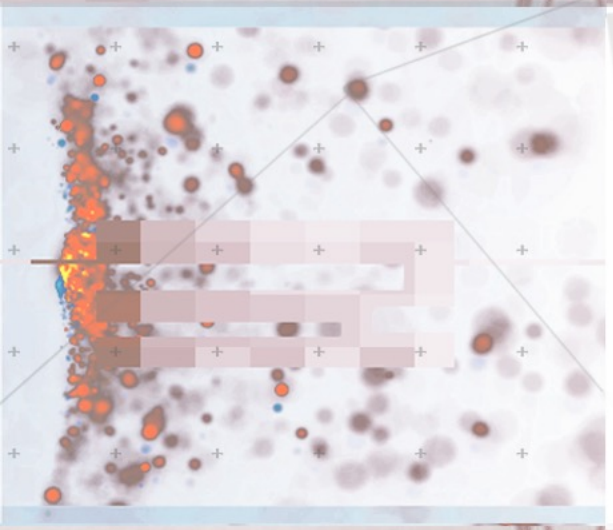


Linear Classifier and Support Vector Machines

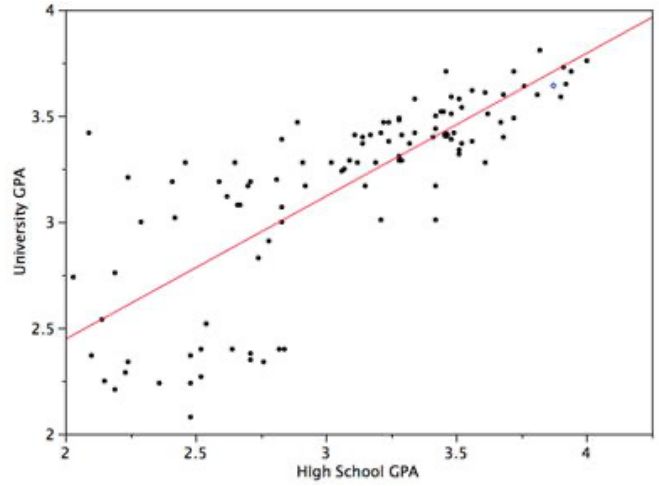
Outline

- Linear Classifier
- Support Vector Machines

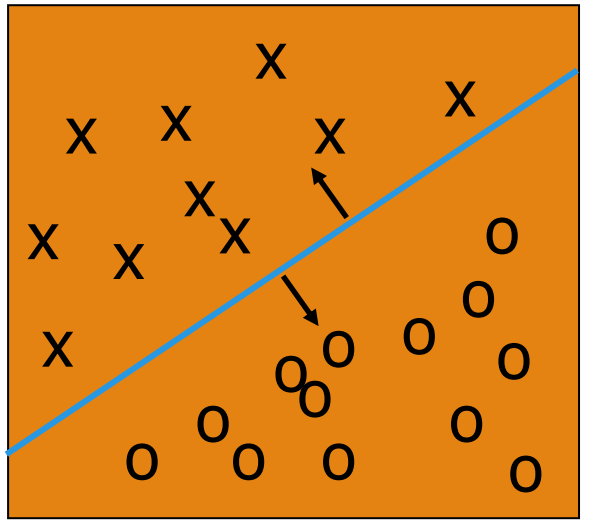
Linear Classifier

Linear Regression vs. Linear Classifier

- Linear regression
  - Data modeled to fit a straight line
    - *Linear equation: $Y = w X + b$*
  - Often uses the least-square method to fit the line
  - Used to predict continuous values

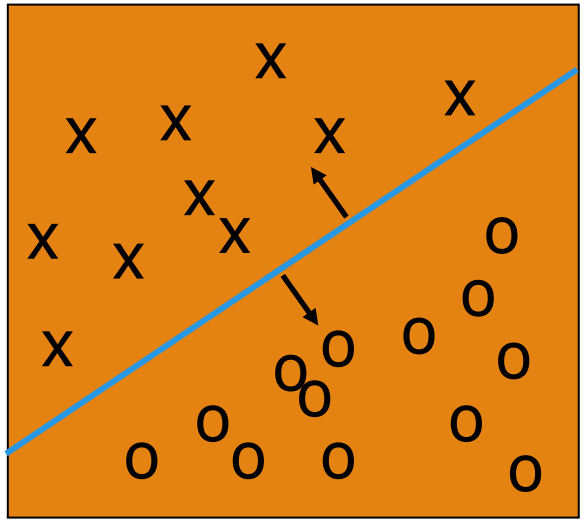


- Linear Classifier
  - Built a classification model using a straight line
  - Used for (categorical data) binary classification



Linear Classifier: General Ideas

- Binary Classification
  - $f(x)$ is a linear function based on the example's attribute values
    - The prediction is based on the value of $f(x)$
    - Data above the blue line belongs to class 'x' (i.e., $f(x) > 0$)
    - Data below blue line belongs to class 'o' (i.e., $f(x) < 0$)
- Classical Linear Classifiers
  - Linear Discriminant Analysis (LDA) (not covered)
  - Logistic Regression
  - Perceptron
  - SVM

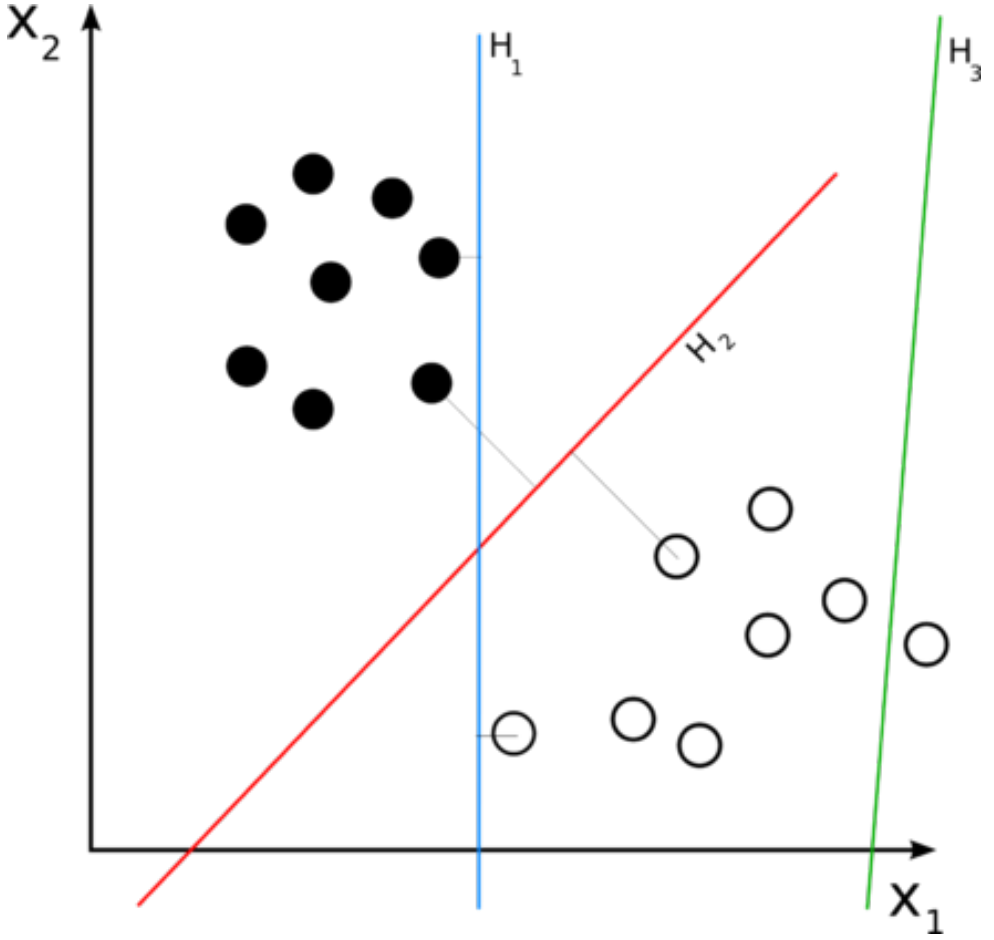


Linear Classifier: An Example

- A toy rule to determine whether a faculty member has tenure
  - Year >= 6 or Title = "Professor" ⇔ Tenure
- How to express the rule as a linear classifier?
  - Features
    - $\mathrm{x}_1 (x_1 \geq 0)$ is an integer denoting the year
    - x_2 is a Boolean denoting whether the title is "Professor"
  - A feasible linear classifier: $f(x) = (x_1 - 5) + 6 \cdot x_2$
    - When x_2 is True, because $x_1 \geq 0$, $f(x)$ is always greater than 0
    - When x_2 is False, because $f(x) > 0 \Leftrightarrow x_1 \geq 6$
  - There are many more feasible classifiers
    - $f(x) = (x_1 - 5.5) + 6 \cdot x_2$
    - $f(x) = 2 \cdot (x_1 - 5) + 11 \cdot x_2$
    - ……

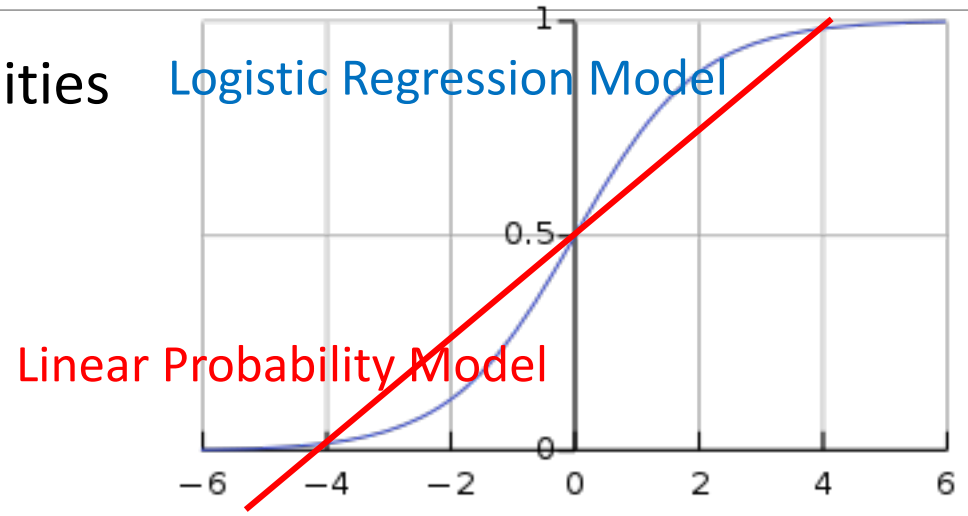
Key Question: Which Line Is Better?

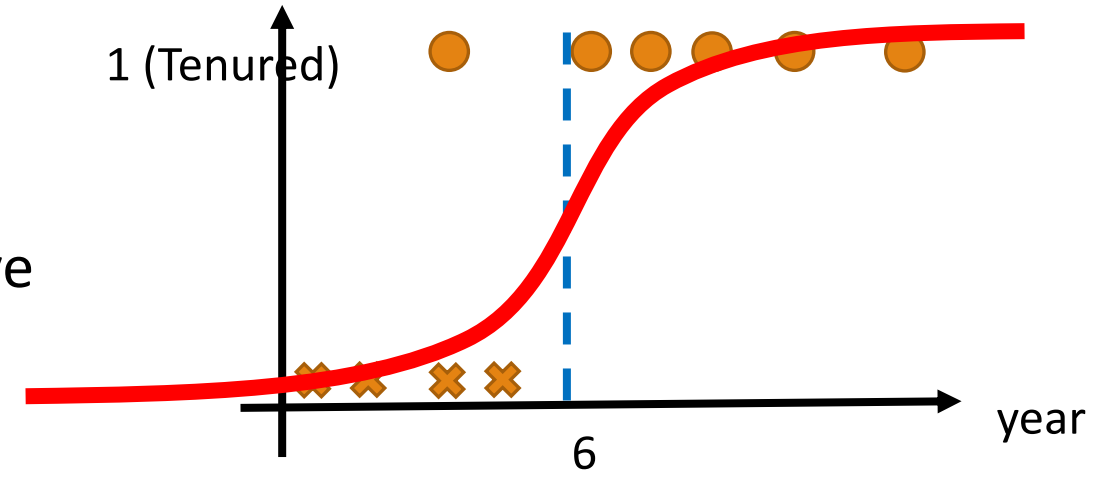
- There might be many feasible linear functions
  - Both H_1 and H_2 will work
- Key question: Which one is better?
  - H_2 looks "better" in the sense that it is also furthest from both groups
  - We will introduce more in the SVM section



Logistic Regression: General Ideas

- Key Idea: Turns linear predictions into probabilities
  - Sigmoid function:
    - $S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$
    - Projects $(-\infty, +\infty)$ to $[0, 1]$
  - Compare to linear probability model
    - Smoother transition
- Logistic regression on our example
  - Only consider year as feature, we have





Logistic Regression: Maximum Likelihood

- The prediction function to learn
  - $p(Y = 1 \,|X = x; \boldsymbol{w}) \;=\; S(w_0 + \sum_{i=1}^{n} w_i \cdot x_i)$
  - $\boldsymbol{w} = (w_0, w_1, w_2, \ldots, w_n)$ are the parameters
- Maximum Likelihood
  - Log likelihood:

$$l(w) = \sum_{i=1}^{N} y_i \log p(Y = 1|X = x_i; \boldsymbol{w}) + (1 - y_i) \log\big(1 - p(Y = 1|X = x_i; \boldsymbol{w})\big)$$

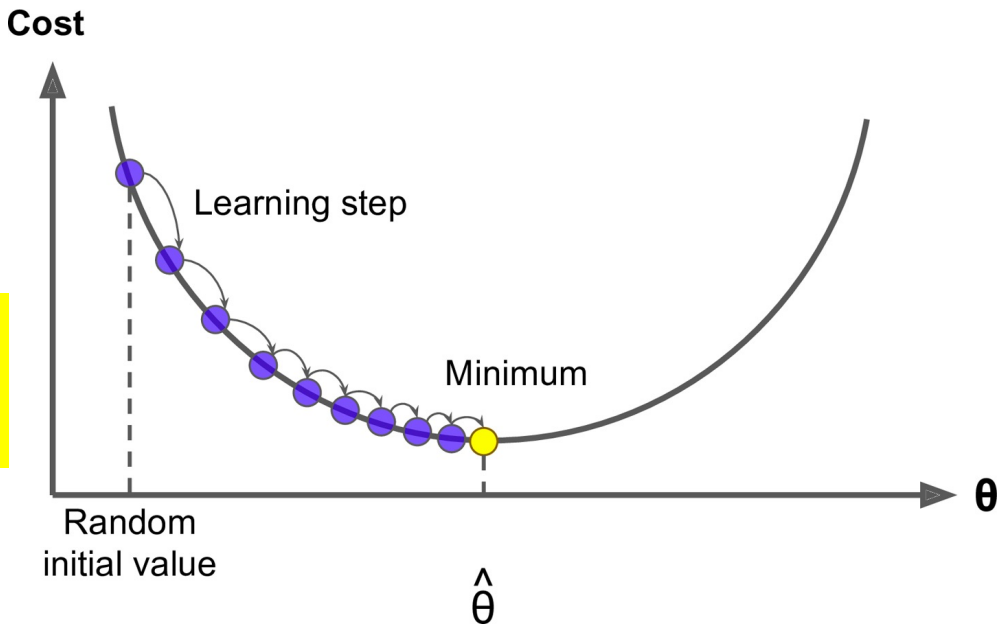
- How to solve it effectively?
  - Gradient Descent
  - Update *w* based on training data
    - Chain-rule for the gradient

Gradient Descent

- Gradient Descent is an iterative optimization algorithm for finding the minimum of a function (e.g., the negative log likelihood)
- For a function F(x) at a point **a**, F(x) *decreases fastest* if we go in the direction of the negative gradient of **a**

$$\mathbf{a}_{n+1} = \mathbf{a}_n - \gamma \nabla F(\mathbf{a}_n)$$

When the gradient is zero, we arrive at the local minimum

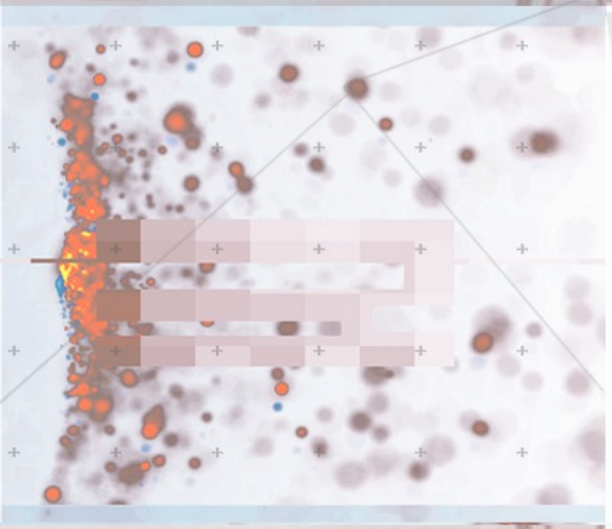


Generative vs. Discriminative Classifiers

- X: observed variables (features) Y: target variables (class labels)
- A generative classifier models p(Y, X)
  - It models how the data was "generated“, and what is the likelihood this or that class generated this instance, and pick the one with higher probability
  - Naïve Bayes
  - Bayesian Networks
- A discriminative classifier models p(Y|X)
  - It uses the data to create a decision boundary
  - Logistic Regression
  - Support Vector Machines

Further Comments on Discriminative Classifiers

- Strength
  - Prediction accuracy is generally high
    - As compared to generative models
    - Robust, works when training examples contain errors
  - Fast evaluation of the learned target function
    - Comparing to Bayesian networks (which are normally slow)
- Criticism
  - Long training time
  - Difficult to understand the learned function (weights)
    - Bayesian networks can be used easily for pattern discovery
  - Not easy to incorporate domain knowledge
    - Easy in the form of priors on the data or distributions

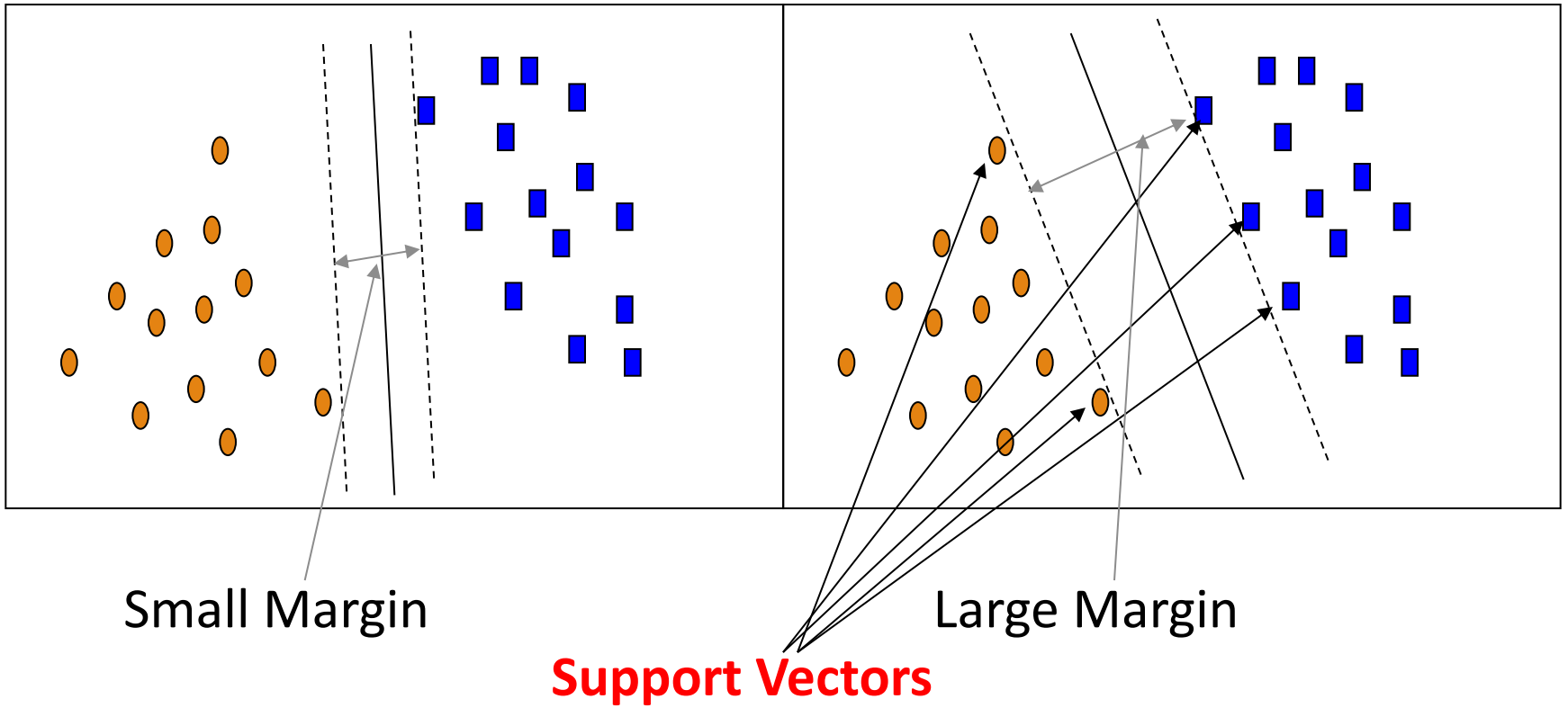


Support Vector Machines

SVM—Support Vector Machines

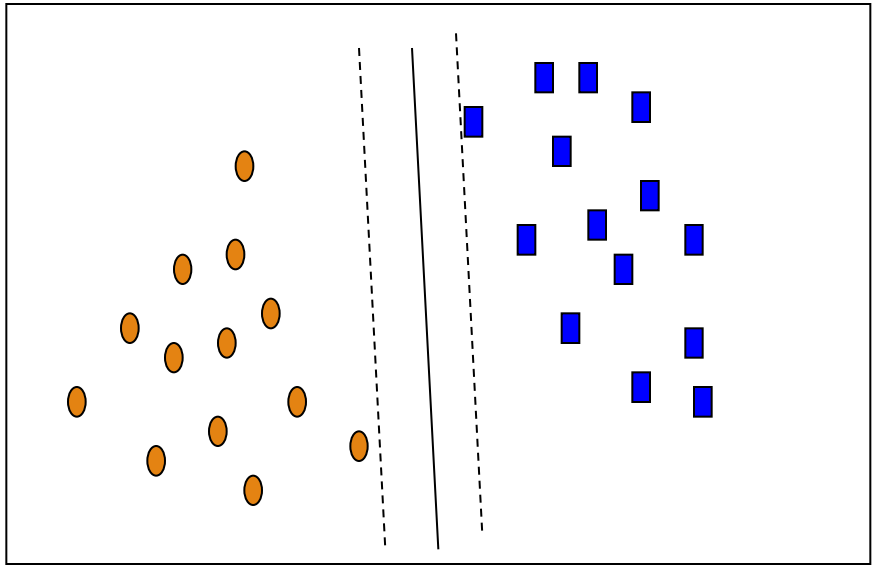
- A popularly adopted classification method for both linear and nonlinear data
  - Proposed by Vapnik and colleagues (1992)—groundwork from Vapnik & Chervonenkis' statistical learning theory in 1960s
- For nonlinear classification, it uses a nonlinear mapping to transform the original training data into a higher dimension space
  - With the new dimension, it searches for the linear optimal separating **hyperplane** (i.e., "decision boundary")
  - With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated by a hyperplane
- SVM finds the hyperplane using **support vectors** ("essential" training instances) and **margins** (defined by the support vectors)

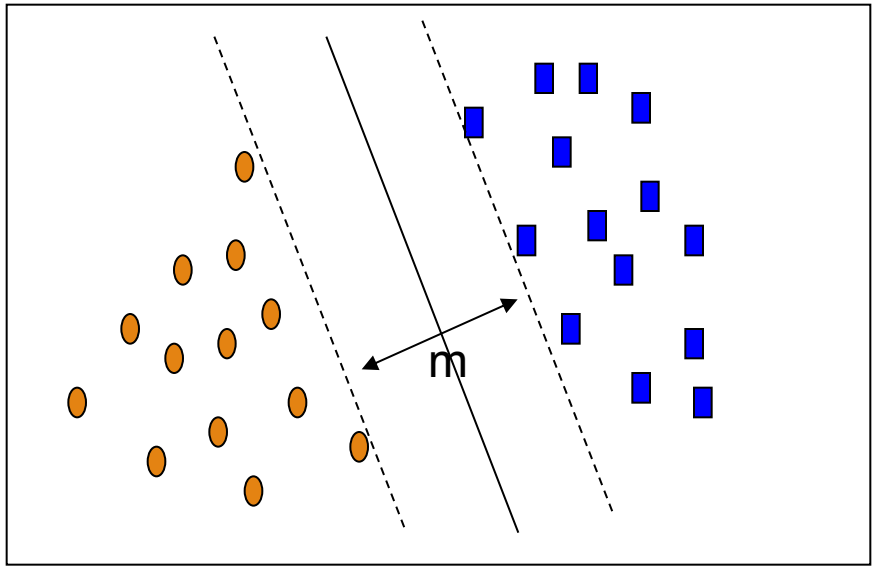
SVM—General Philosophy



- The best hyperplane is the one that represents the largest separation, or **margin**, between the two classes
- Samples on the margin are called the **support vectors**

SVM—When Data Is Linearly Separable





Let data D be ($\mathbf{X}_1$, y_1), …, ($\mathbf{X}_{|D|}$, $y_{|D|}$), where $\mathbf{X}_i$ is the set of training instances associated with the class labels y_i

There are infinite lines (hyperplanes) separating the two classes but we want to find the best one (the one that minimizes classification error on unseen data)

SVM searches for the hyperplane with the largest margin, i.e., **maximum marginal hyperplane** (MMH)

SVM—Linearly Separable

- A separating hyperplane can be written as

  $\mathbf{W} \bullet \mathbf{X} + b = 0$

  where $\mathbf{W}=\{w_1, w_2, \ldots, w_n\}$ is a weight vector and b a scalar (bias)
- For 2-D, it can be written as: $w_0 + w_1 x_1 + w_2 x_2 = 0$
  - The hyperplane defining the sides of the margin:

    $H_1: w_0 + w_1 x_1 + w_2 x_2 \geq 1$ for $y_i = +1$, and

    $H_2: w_0 + w_1 x_1 + w_2 x_2 \leq -1$ for $y_i = -1$
- Any training tuples that fall on hyperplanes H_1 or H_2 (i.e., the sides defining the margin) are **support vectors**
- This becomes a **constrained (convex) quadratic optimization** problem:
  - Quadratic objective function and linear constraints → *Quadratic Programming (QP)* → Lagrangian multipliers

SVM—Linearly Inseparable

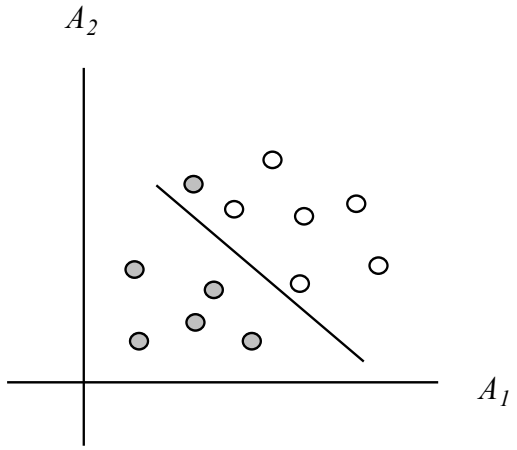
- Transform the original input data into a higher dimensional space

Example 6.8 Nonlinear transformation of original input data into a higher dimensional space. Consider the following example. A 3D input vector $\mathbf{X} = (x_1, x_2, x_3)$ is mapped into a 6D space Z using the mappings $\phi_1(X) = x_1, \phi_2(X) = x_2, \phi_3(X) = x_3, \phi_4(X) = (x_1)^2, \phi_5(X) = x_1x_2$, and $\phi_6(X) = x_1x_3$. A decision hyperplane in the new space is $d(\mathbf{Z}) = \mathbf{WZ} + b$, where $\mathbf{W}$ and $\mathbf{Z}$ are vectors. This is linear. We solve for $\mathbf{W}$ and b and then substitute back so that we see that the linear decision hyperplane in the new $(\mathbf{Z})$ space corresponds to a nonlinear second order polynomial in the original 3-D input space,

$$\begin{aligned} d(Z) &= w_1x_1 + w_2x_2 + w_3x_3 + w_4(x_1)^2 + w_5x_1x_2 + w_6x_1x_3 + b \\ &= w_1z_1 + w_2z_2 + w_3z_3 + w_4z_4 + w_5z_5 + w_6z_6 + b \end{aligned}$$

■

- Search for a linear separating hyperplane in the new space



Kernel Functions for Nonlinear Classification

- SVMs can efficiently perform a non-linear classification using kernel functions, implicitly mapping their inputs into high-dimensional feature spaces
- Instead of computing the dot product on the transformed data, it is mathematically equivalent to applying a kernel function $K(\mathbf{X}_i, \mathbf{X}_j)$ to the original data, i.e.,
  - $K(\mathbf{X}_i, \mathbf{X}_j) = \Phi(\mathbf{X}_i)\ \Phi(\mathbf{X}_j)$
- Typical Kernel Functions

$$\text{Polynomial kernel of degree } h: \quad K(X_i, X_j) = (X_i \cdot X_j + 1)^h$$

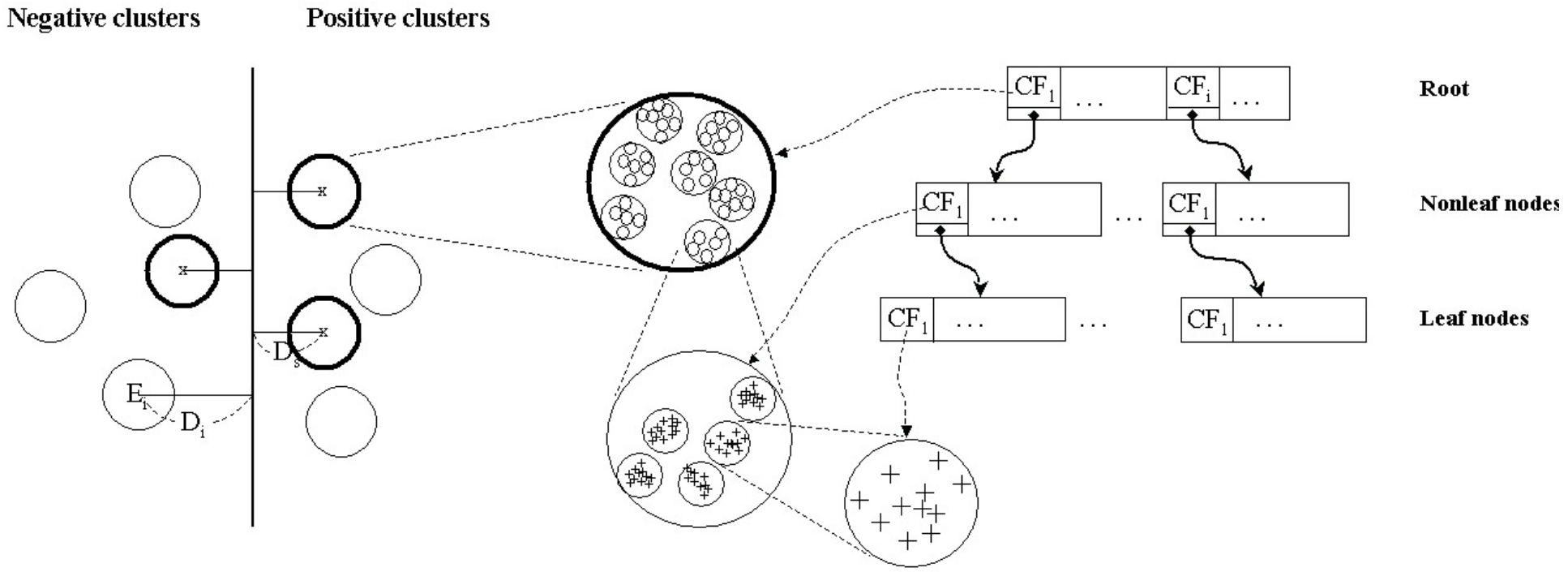
$$\text{Gaussian radial basis function kernel}: \quad K(X_i, X_j) = e^{-\|X_i - X_j\|^2/2\sigma^2}$$

$$\text{Sigmoid kernel}: \quad K(X_i, X_j) = \tanh(\kappa X_i \cdot X_j - \delta)$$

Is SVM Scalable on Massive Data?

- SVM is effective on high dimensional data
  - The **complexity** of trained classifier is characterized by the number of support vectors rather than the dimensionality of the data
  - The **support vectors** are the essential or critical training examples—they lie closest to the decision boundary (MMH)
  - Thus, an SVM with a small number of support vectors can have good generalization, even when the dimensionality of the data is high
- SVM is not scalable to the number of data objects in terms of training time and memory usage
  - Scaling SVM by a hierarchical micro-clustering approach
    - H. Yu, J. Yang, and J. Han, "Classifying Large Data Sets Using SVM with Hierarchical Clusters", KDD'03

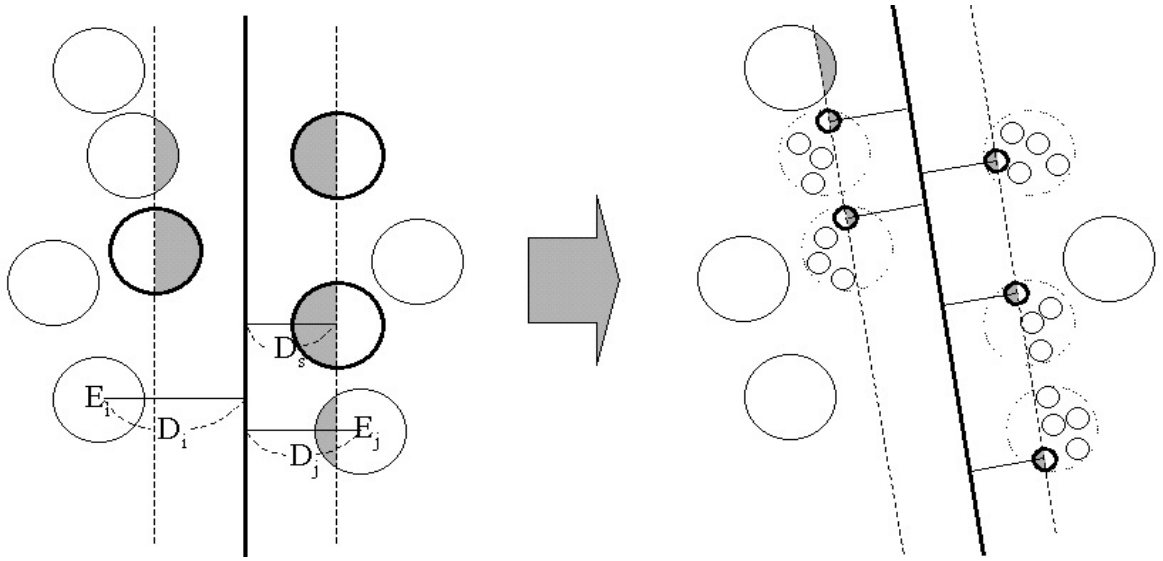
Scaling SVM by Hierarchical Micro-Clustering



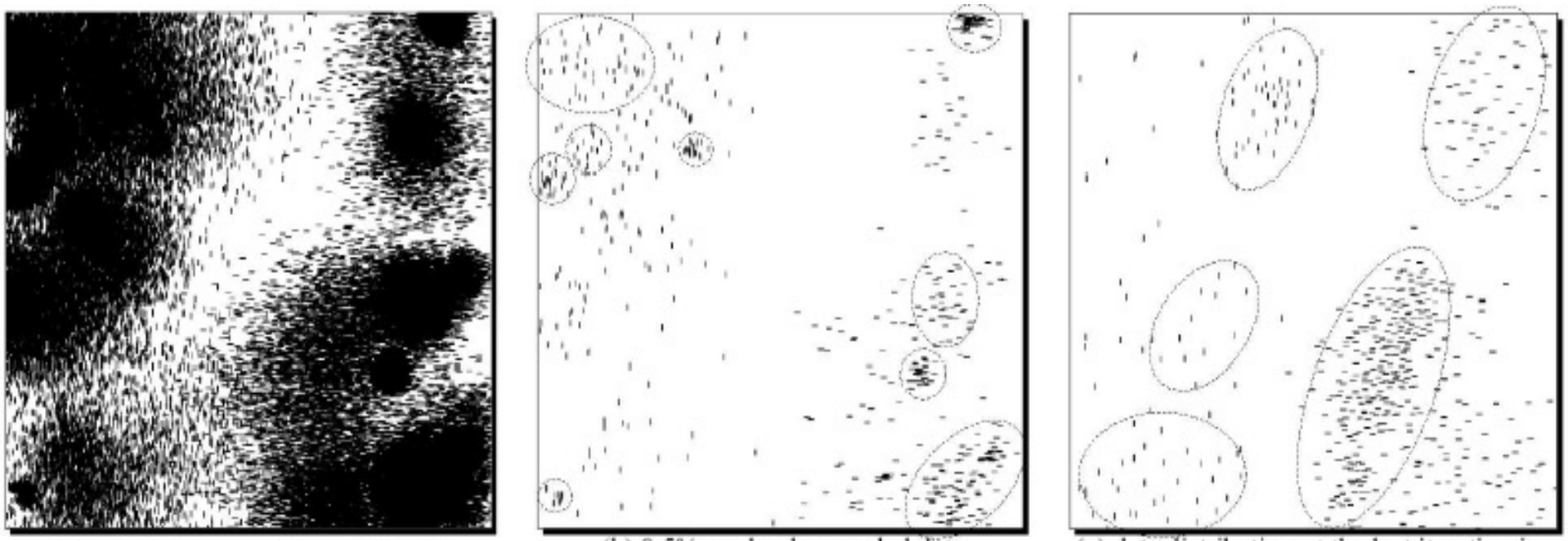
- Construct two CF-trees (i.e., statistical summary of the data) from positive and negative data sets independently (with one scan of the data set)
- Micro-clustering: Hierarchical indexing structure
  - Provide finer samples closer to the boundary and coarser samples farther from the boundary

Selective Declustering: Ensure High Accuracy

- Decluster only the cluster whose subclusters have possibilities to be the support cluster of the boundary
  - "Support cluster": The cluster whose centroid is a support vector
- De-cluster only the cluster E_i such that
  - $D_i - R_i < D_s$, where D_i is the distance from the boundary to the center point of E_i and R_i is the radius of E_i



Accuracy and Scalability on Synthetic Dataset



(a) original data set ($N = 113601$)

(b) 0.5% randomly sampled data ($N = 603$)

(c) data distribution at the last iteration in CB-SVM ($N = 597$)

Figure 6: Synthetic data set in a two-dimensional space. '|': positive data; '−': negative data

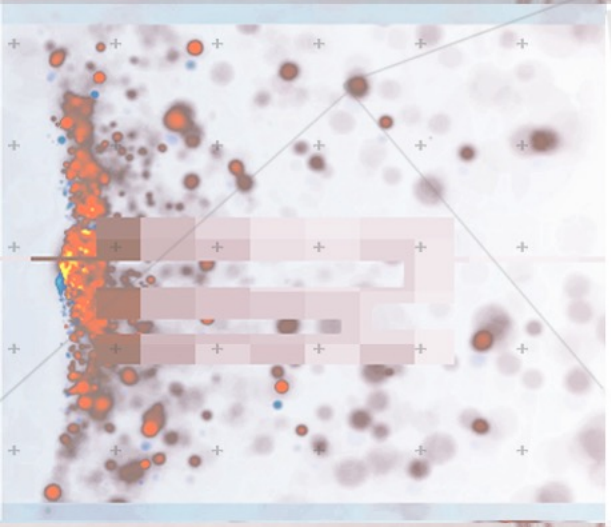
- Experiments on large synthetic data sets shows better accuracy than random sampling approaches and far more scalable than the original SVM algorithm

SVM: Features and Applications

- Features and challenges
  - Training can be slow but accuracy is high owing to their ability to model complex nonlinear decision boundaries (margin maximization)
  - However, the parameters of a solved model are difficult to interpret
- SVM can also be used for classifying multiple (> 2) classes and for regression analysis (with additional parameters)
- SVM has been used in many applications
  - Handwritten digit recognition, object recognition, speaker identification, benchmarking time-series prediction tests
  - Document classification, biomedical data analysis

SVM Related Links

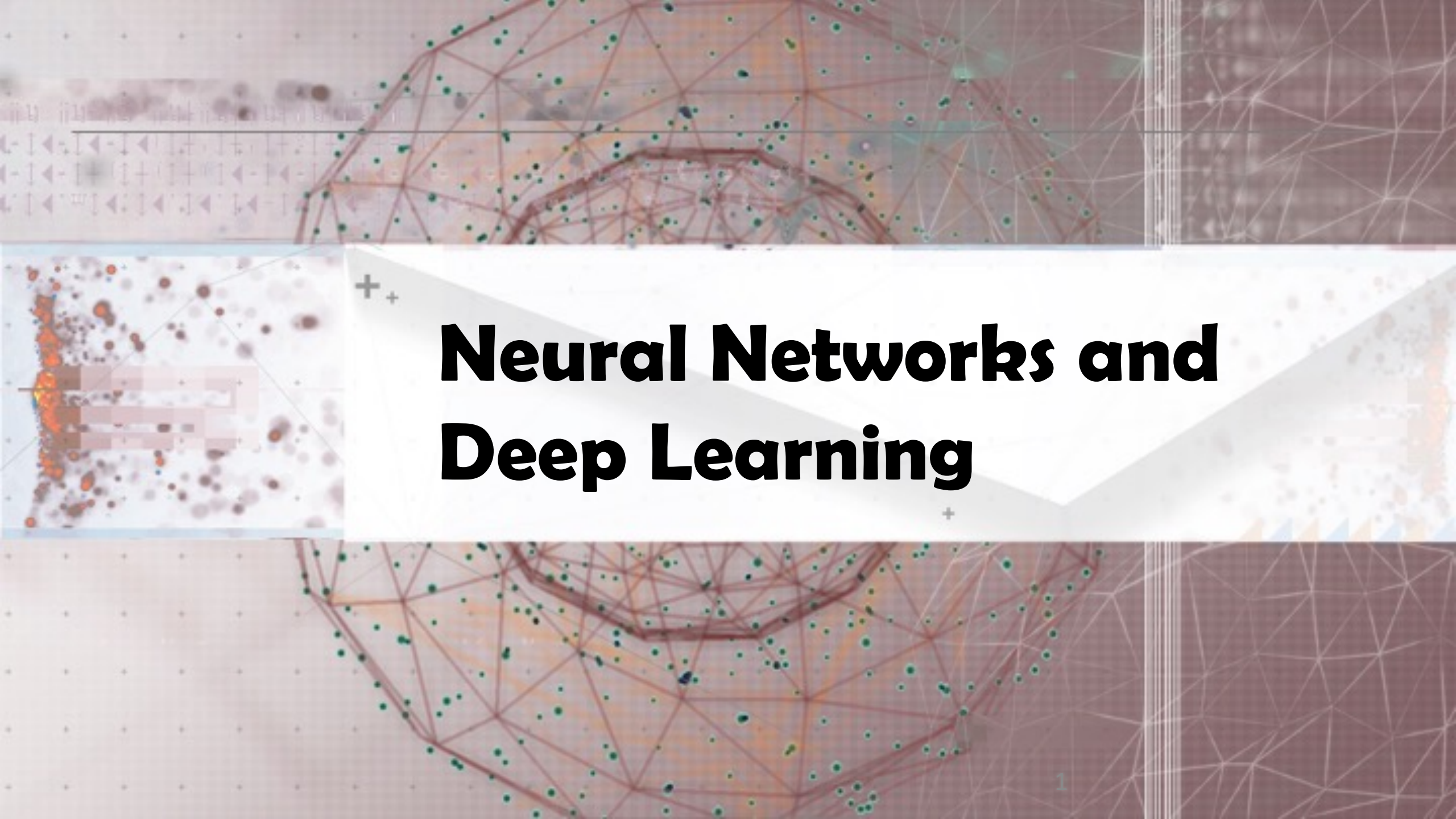
- SVM Website: http://www.kernel-machines.org/
- Representative implementations
  - **LIBSVM**: an efficient implementation of SVM, multi-class classifications, nu-SVM, one-class SVM, including also various interfaces with java, python, etc.
  - **SVM-light**: simpler but performance is not better than LIBSVM, support only binary classification and only in C
  - **SVM-torch**: another recent implementation also written in C



Summary

Recommended Readings

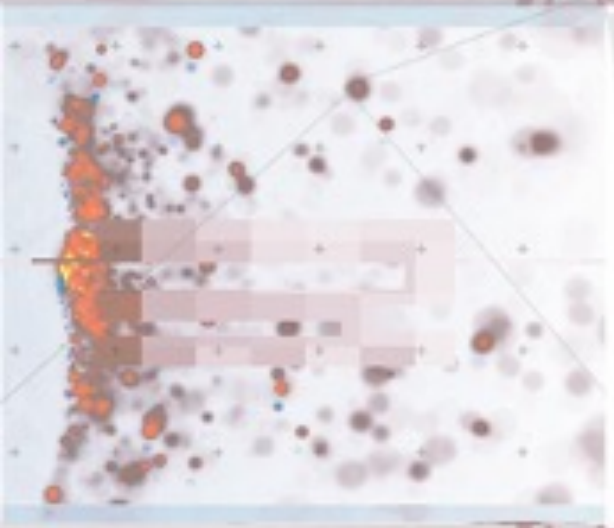
- N. S. Altman, “An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression”, The American Statistician. **46** (3): 175–185, 1992
- C. Cortes and V. Vapnik, Support Vector Machine, Machine Learning, 1995
- D. W. Hosmer Jr, S. Lemeshow, R. X. Sturdivant, Applied Logistic Regression, John Wiley, 2013
- D. G. Kleinbaum, M. Klein. Logistic Regression: A Self-Learning Text. Springer, 2010
- A. Y. Ng and M. I. Jordan, On Discriminative vs. Generative Classifiers: A Comparison of Logistic Regression and Naïve Bayes, NIPS, 2002
- B. Scholkopf and A.J. Smola, Learning with Kernels: Support Vector Machines, Regularization, Optimization, And Beyond, The MIT Press, 2001
- S. Tong and D. Koller, Support Vector Machine Active Learning With Applications to Text Classification, Journal of Machine Learning Research, 2001
- G.-X. Yuan, C.-H. Ho and C.-J. Lin. “Recent Advances of Large-Scale Linear Classification”. Proc. IEEE. 100 (9), 2012



Neural Networks and Deep Learning

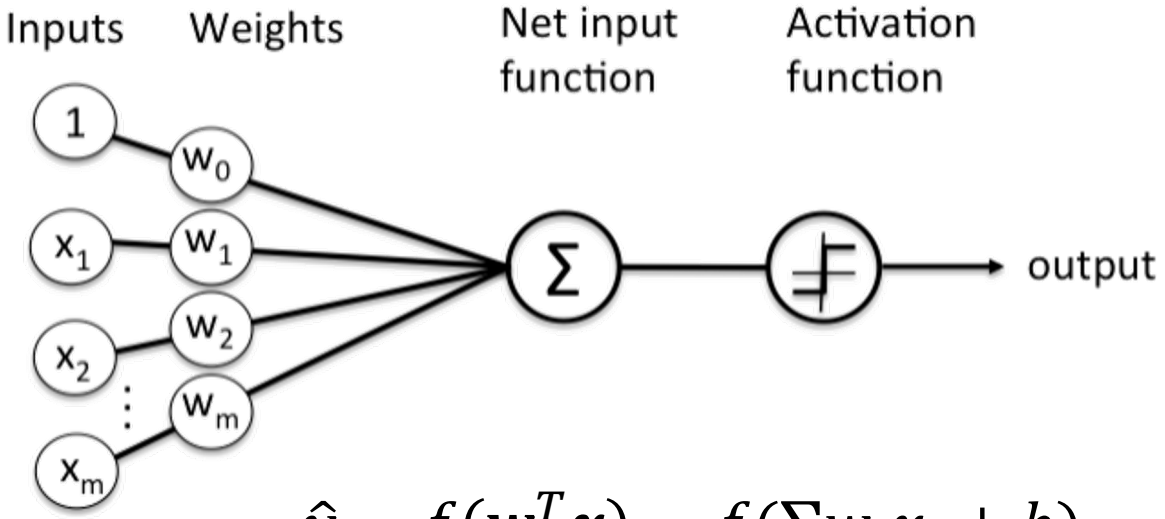
Neural Networks and Deep Learning

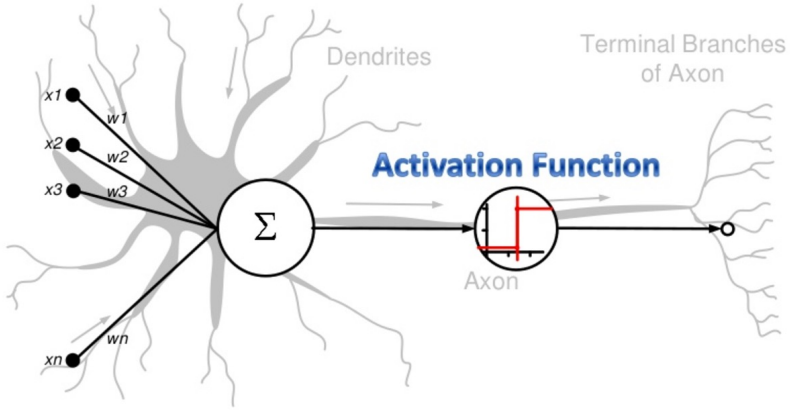
- Neural Networks and Activation Functions
- Learning Neural Network Parameters
- Deep Learning: Convolutional Neural Networks
- Deep Learning: Recurrent Neural Networks



Neural Networks and Activation Functions

A Neuron (Perceptron) and Its Function





The name comes from analogy of a biological neuron

$$\hat{y} = f(\boldsymbol{w}^T \boldsymbol{x}) = f(\sum w_i x_i + b)$$

Non-linear Activation Function
Adding non-linearity to the model

Weights

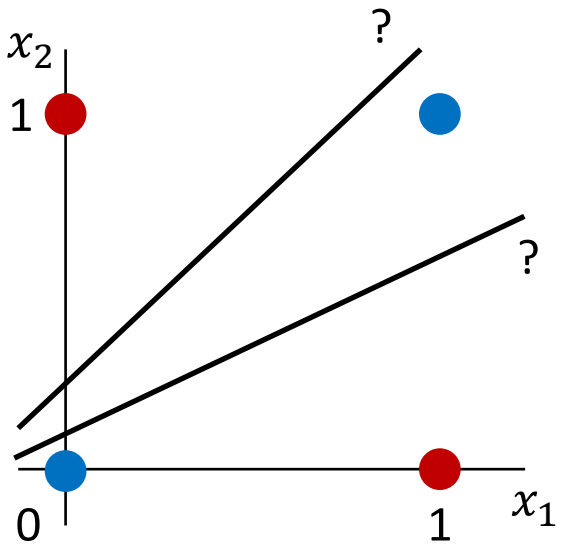
Bias
A measure of how easy it is to get the perceptron to output a 1

- A neuron: Weighted combination of inputs + bias + non-linear activation function

More on activation functions: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0
Figures adapted from https://pt.slideshare.net/hammawan/deep-neural-networks

Logistic Regression and Its Limitation

- Recall the logistic regression classifier
  - $p(Y = 1 \,|X = \boldsymbol{x}; \boldsymbol{w}) = g(x) \;=\; \text{Sigmoid}(w_0 + \sum_{i=1}^{n} w_i \cdot x_i)$
- The linear model $g(x)$ has limited approximation power, and may fail for even some simple problems
- Example: The XOR problem
  - Solve for $y \;= x_1 \;\text{XOR}\; x_2$



The XOR problem:
Not linearly separable

- Can we build more powerful models that can approximate *any* function?

Solving the XOR Problem by Stacking Neurons

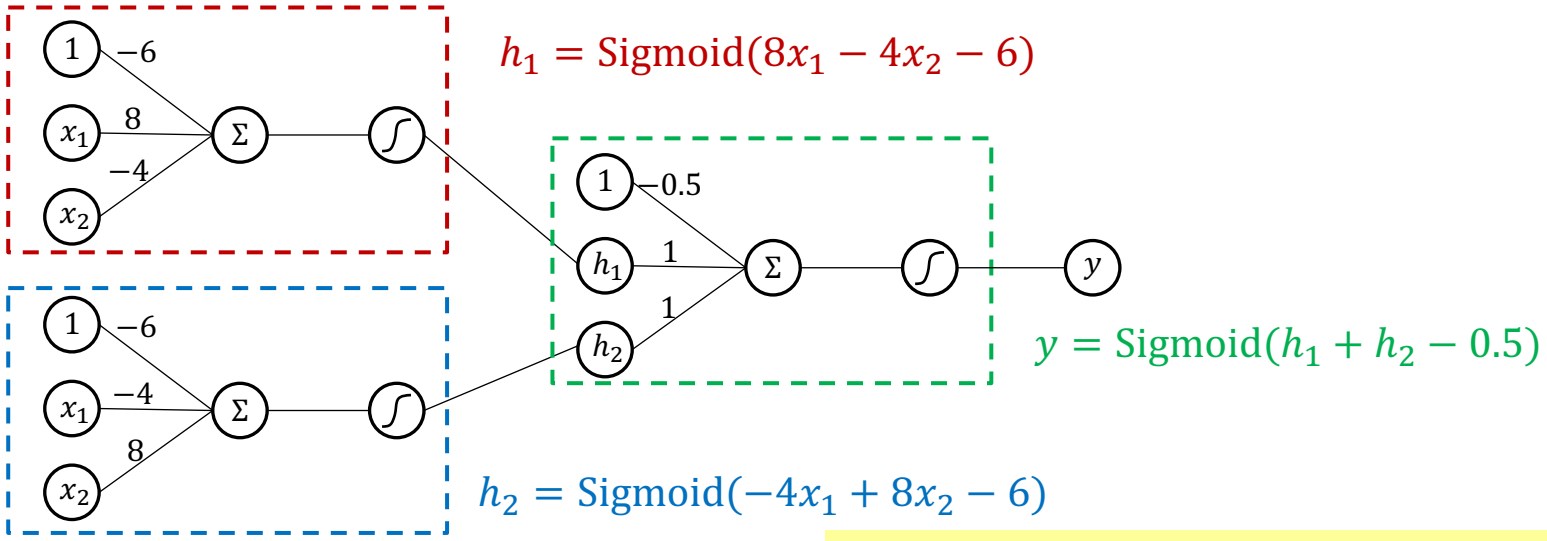
- Each neuron can be used as a basic component, combining multiple neurons gives us more power
- The XOR problem can be rewritten as

$$y = x_1 \text{ XOR } x_2 \quad = (x_1 = 1 \text{ AND } x_2 = 0) \text{ OR } (x_1 = 0 \text{ AND } x_2 = 1)$$

Multiple neurons in one *layer*

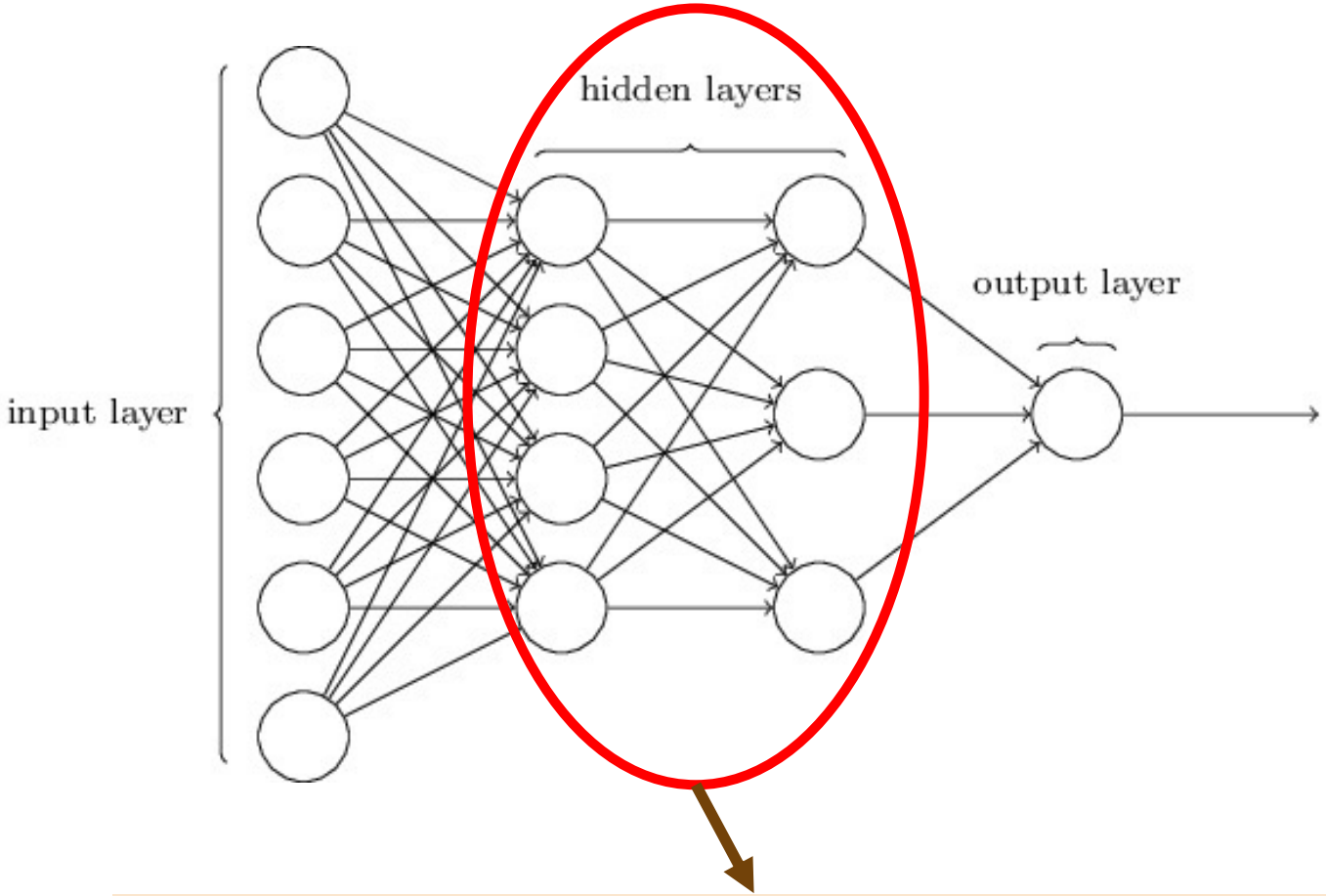
- We can use two neurons to generate intermediate results h_1 and h_2
  - And then use a third neuron to generate the final prediction y from h_1 and h_2

Multiple layers



From: https://courses.cs.washington.edu/courses/cse446/16sp/logistic_regression_3.pdf

From Neurons to Neural Networks



Stacking multiple layers of neurons (adding hidden layers): A multilayer perceptron

- Using multiple neurons in a *layer*, and stacking multiple layers of neurons makes a *neural network*
- The simplest neural network: a multilayer perceptron (MLP), connecting everything together
- MLP can engage in sophisticated decision making and have the power to approximate any function

Play with neural network: http://playground.tensorflow.org

Figure adapted from https://dev.to/liashchynskyi/creating-of-neural-network-using-javascript-in-7minutes-o21

Why Non-Linear Activation Function?

- Imagine a neural network with no non-linear activation functions
- A single layer

$$\boldsymbol{x} \mapsto \boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1$$

- Multiple layers

$$\boldsymbol{x} \mapsto \boldsymbol{W}_N(\ldots \boldsymbol{W}_2(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2 \ldots) + \boldsymbol{b}_N$$

$$= (\boldsymbol{W}_N \ldots \boldsymbol{W}_2\boldsymbol{W}_1)\boldsymbol{x} + (\boldsymbol{b}_N + \cdots + \boldsymbol{W}_N \ldots \boldsymbol{W}_2\boldsymbol{b}_1)$$

  - It is still a linear function of $\boldsymbol{x}$, which is undesirable
- Using non-linear activation functions is essential for the expressive power of neural networks

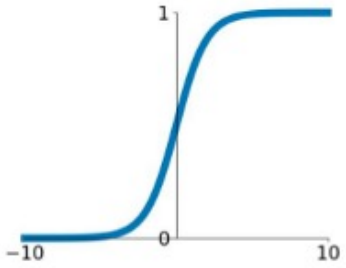
Common Non-Linear Activation Functions

- Examples of activation functions

More on activation functions: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

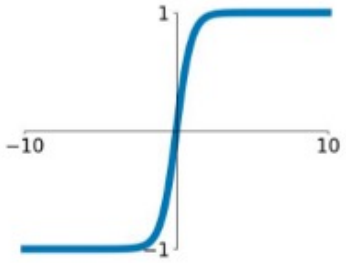
Sigmoid

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



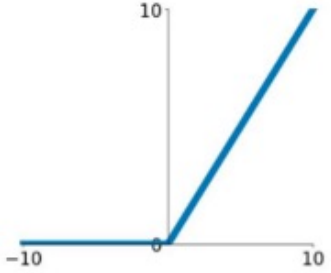
tanh

$$\tanh(x)$$



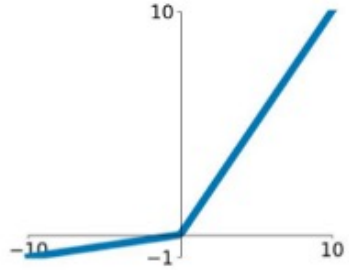
ReLU

$$\max(0, x)$$



Leaky ReLU

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



Maxout

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

ELU

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

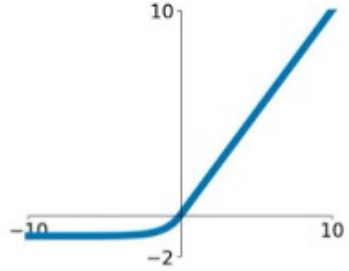
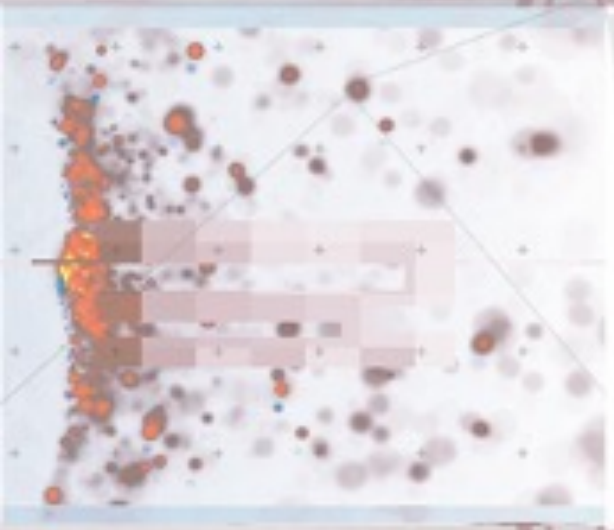
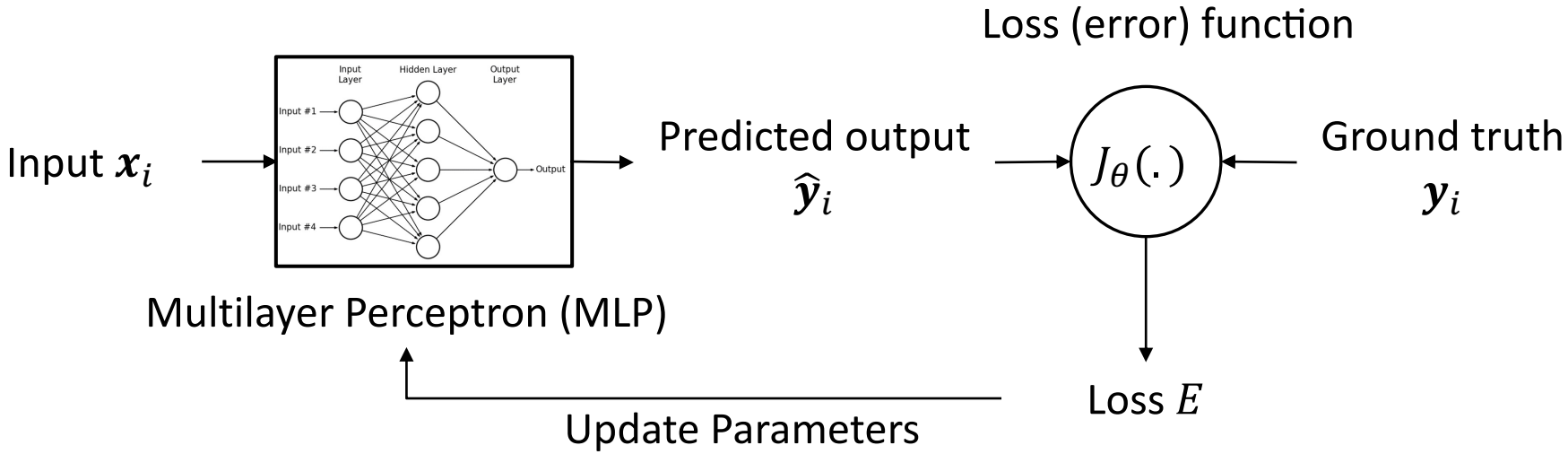


Figure adapted from https://medium.com/dataseries/basic-overview-of-convolutional-neural-network-cnn-4fcc7dbb4f17



Learning Neural Network Parameters

Learning Neural Network Parameters

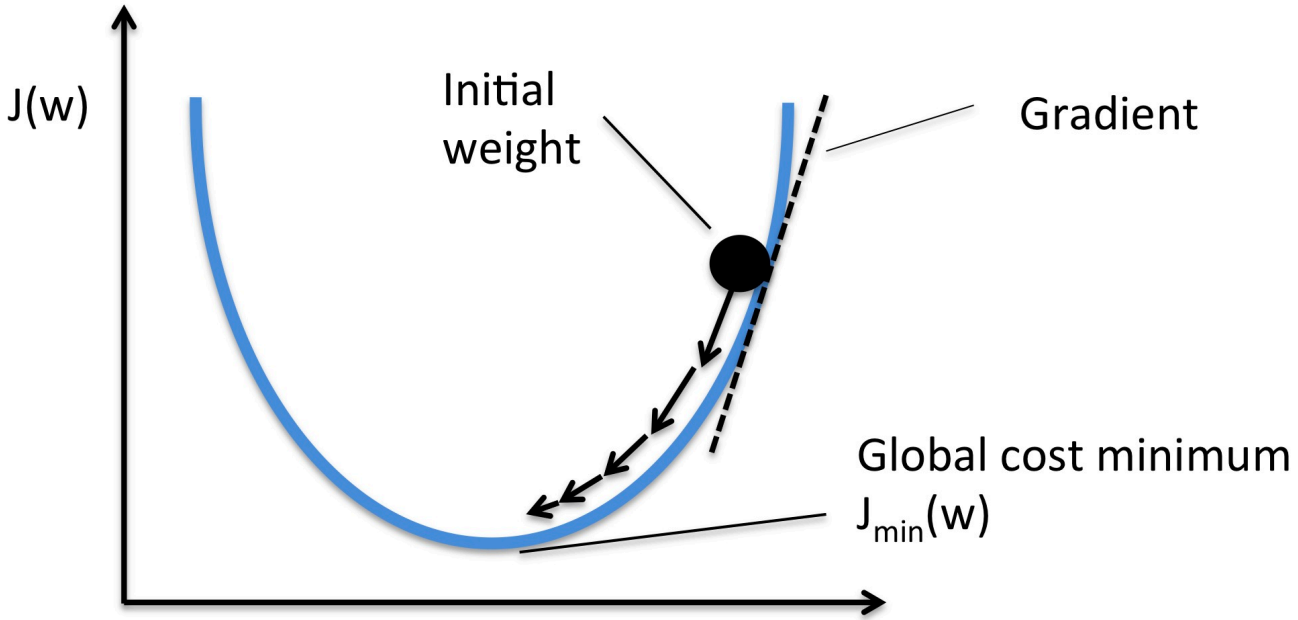


- **Gradient Descent Algorithm**
- **Input**: Training sample $\boldsymbol{x}_i$ and its label $\boldsymbol{y}_i$

1. **Feed Forward**: Get prediction $\hat{\boldsymbol{y}}_i = \mathrm{MLP}(x_i)$, and loss $E = J(\hat{\boldsymbol{y}}_i, \boldsymbol{y}_i)$
2. **Compute Gradient**: For each parameter θ_i (weights, bias), compute its gradient $\frac{\partial}{\partial \theta_i} J_\theta$
3. **Update Parameter**: $\theta_i = \theta_i - \alpha \cdot \frac{\partial}{\partial \theta_i} J_\theta$

Explained later

Empirical Explanation of Gradient Descent



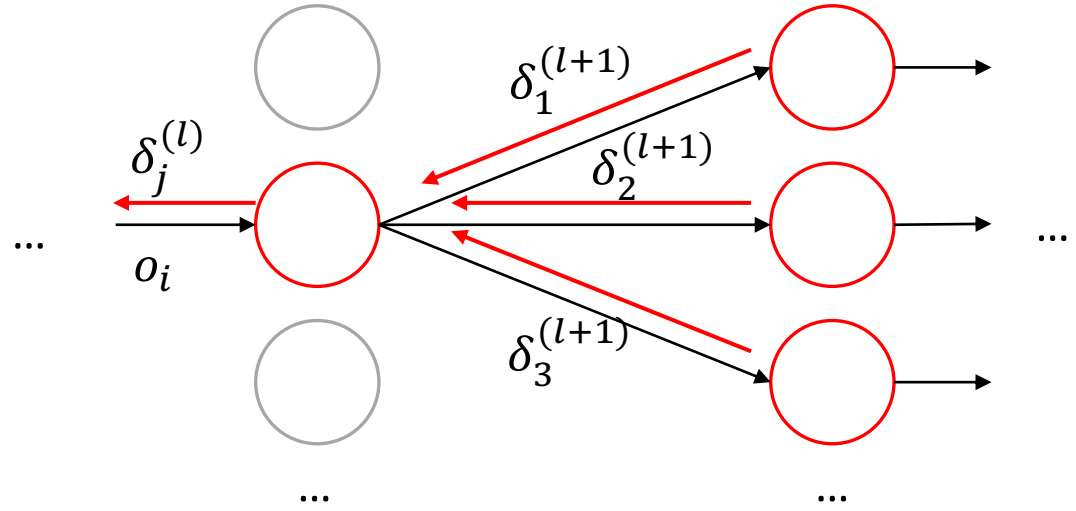
$$\theta_i = \theta_i - \alpha \cdot \frac{\partial^{w}}{\partial \theta_i} J_\theta$$

learning rate – "step size" of the optimization

- The loss function J: a function of the model parameters
- Objective: Minimize J
- Gradient: Measures how much the output of a function changes if you change the inputs a little bit
- We update the parameters, based on their gradients, so that the loss function is going "downhill"

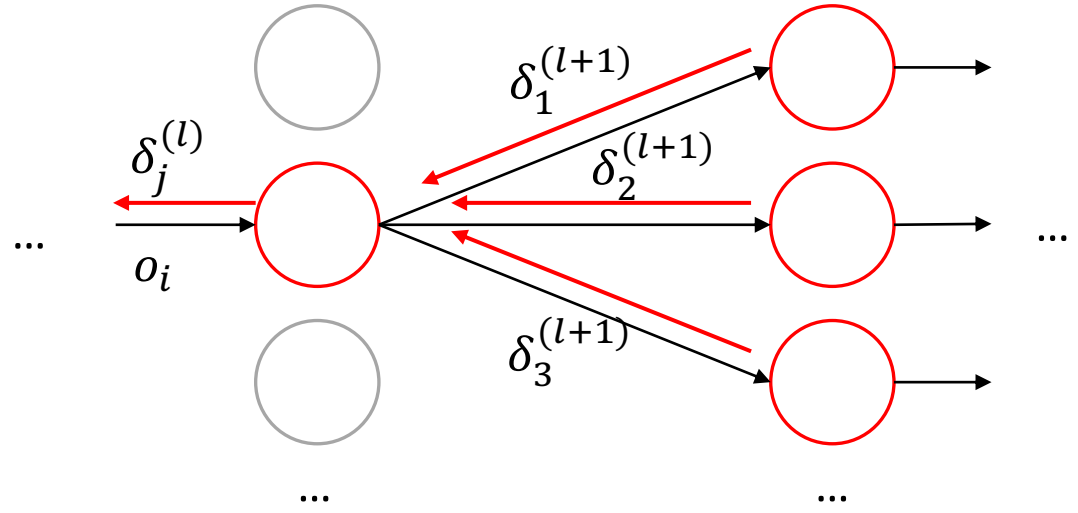
Figure adapted from https://www.upgrad.com/blog/machine-learning-interview-questions-answers-ii/

Gradient Computation: Backpropagation



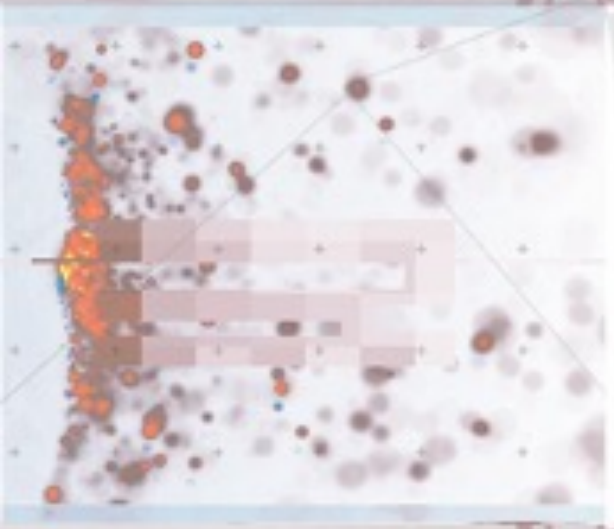
- The gradient of w_{ij} in the l-th layer (corresponding to unit j in layer l, connected to unit i in layer l-1) is a function of
  - All "error" terms from layer l+1 $\delta_k^{(l+1)}$: An auxiliary term for computation, not to be confused with gradients
  - Output from unit i in layer l-1 (input to unit j in layer l): Can be stored at the feed forward phase of computation

Gradient Computation: Backpropagation



This process can be handled by modern Deep Learning libraries such as PyTorch and TensorFlow automatically, once the feedforward process is specified.

- The "error" terms $\delta_j^{(l)}$ is a function of
  - All $\delta_k^{(l+1)}$ in the layer l+1, if layer l is a hidden layer
  - The overall loss value, if layer l is the output layer
- We can compute the error at the output, and distributed backwards throughout the network's layers (backpropagation)



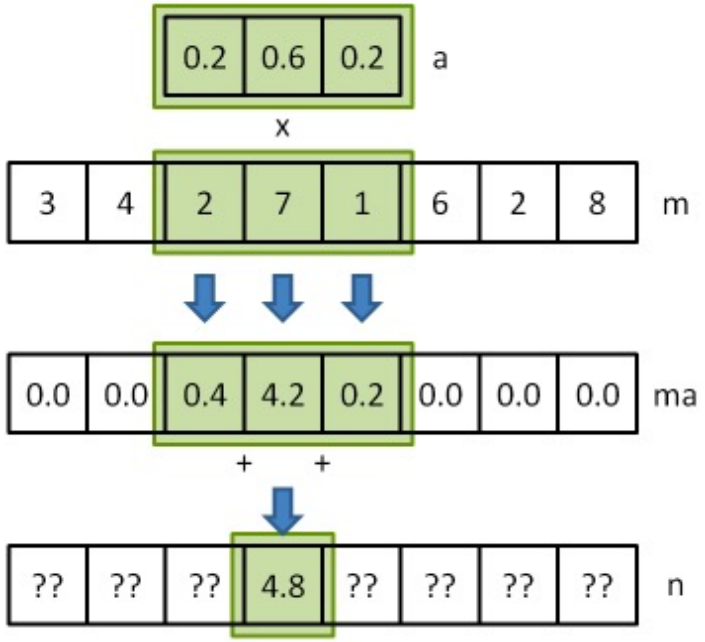
Deep Learning: Convolutional Neural Networks

From Neural Networks to Deep Learning

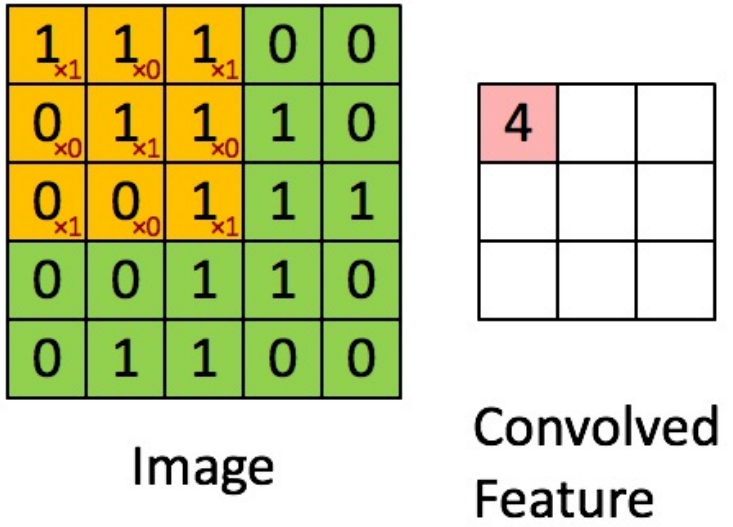
- **Deep Learning:** Training (deep) neural networks with
  - More neurons, more layers
  - More complex ways to connect layers
- Advantages
  - **Tremendous improvement of performance** in
    - Image recognition, natural language processing, AI game playing…
  - **Requires no (or less) feature engineering**, making end-to-end models possible
- Several factors lead to deep learning's success
  - Very large data sets
  - Massive amounts of computation power (GPU acceleration)
  - Advanced neural network structures and tricks
    - Convolutional neural networks, recurrent neural networks, …
    - Dropout, ReLU, residual connection, … (not covered)

Convolutional Neural Networks (CNN)

- What is convolution?



1D Convolution



Image

Convolved Feature

2D Convolution

Ack. figure adapted from: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

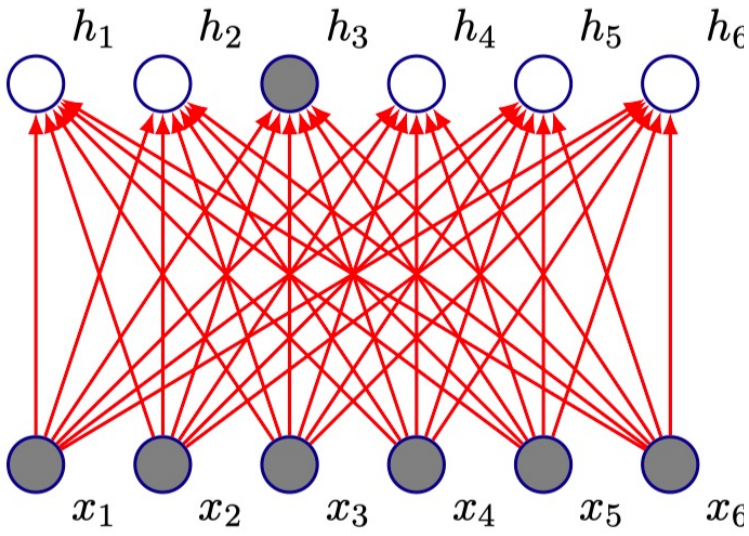
- The outputs are computed by sliding a **kernel** (of weights) on the inputs, and computing weighted sum locally

CNN Motivation

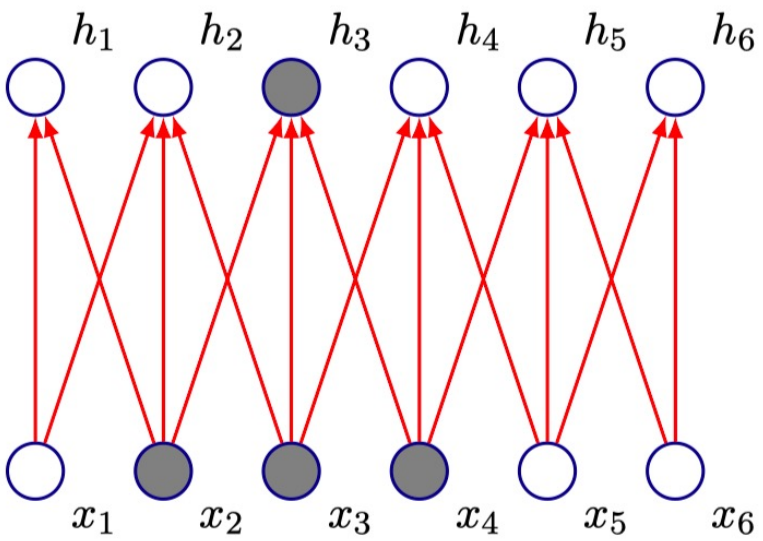
- Why not deep MLP?
  - **Computationally expensive** (Long training time)
  - **Hard to train** (slow convergence, local minima)
- Motivations of convolution
  - Sparse interactions
  - Parameter sharing
  - Equivariant representations
- The properties of CNNs are well aligned with properties of many forms of data (e.g., images, text), making them very successful

CNN Motivation: Sparse Interactions

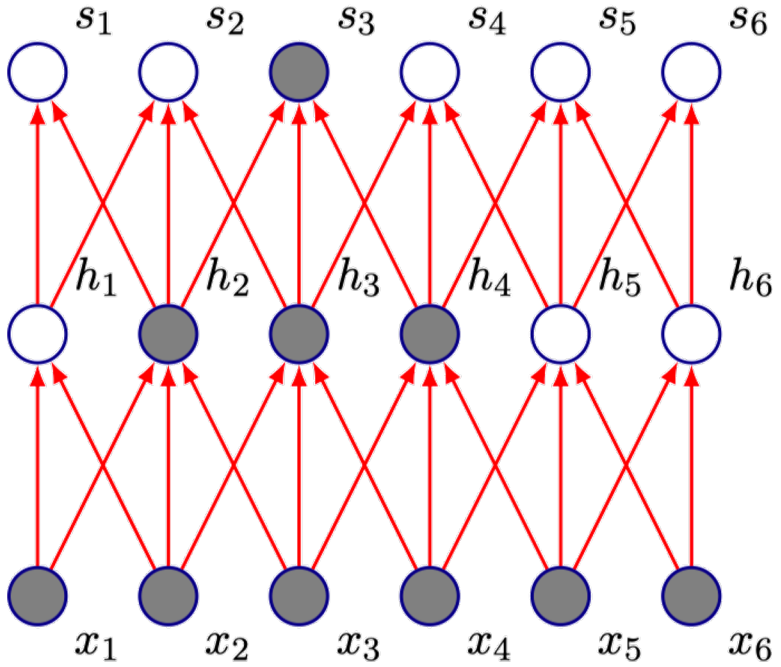
- Motivations of convolution
  - Sparse interactions
    - Ex. 1D convolution with kernel size 3
    - Units in deeper layers still connect to a wide range of inputs



MLP



CNN



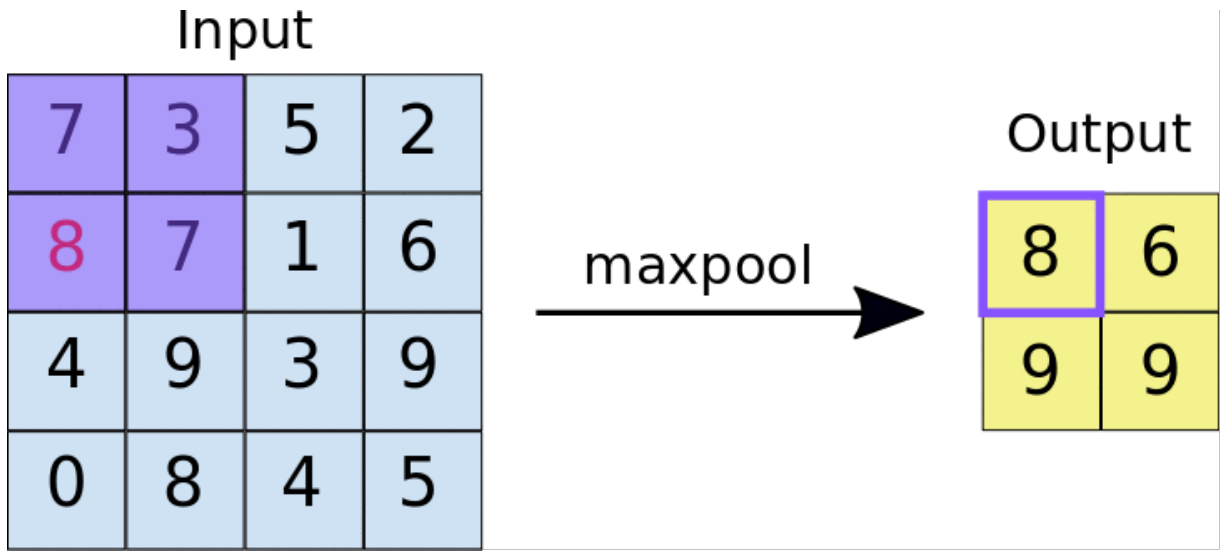
More on convolutional neural networks http://cs231n.github.io/convolutional-networks/

CNN Motivation: Parameter Sharing & Equivariance

- Motivations of convolution
  - **Parameter sharing**
    - Each kernel is used on all locations of input
    - Reduce # of parameters
  - **Equivariance**
    - Same input at different location gives same output
    - Ex. A cat at the upper right corner and at the lower left corner of an image, will produce the same outputs
    - Ex. “University of Illinois” at the start of the sentence and at the end of the sentence produce the same outputs

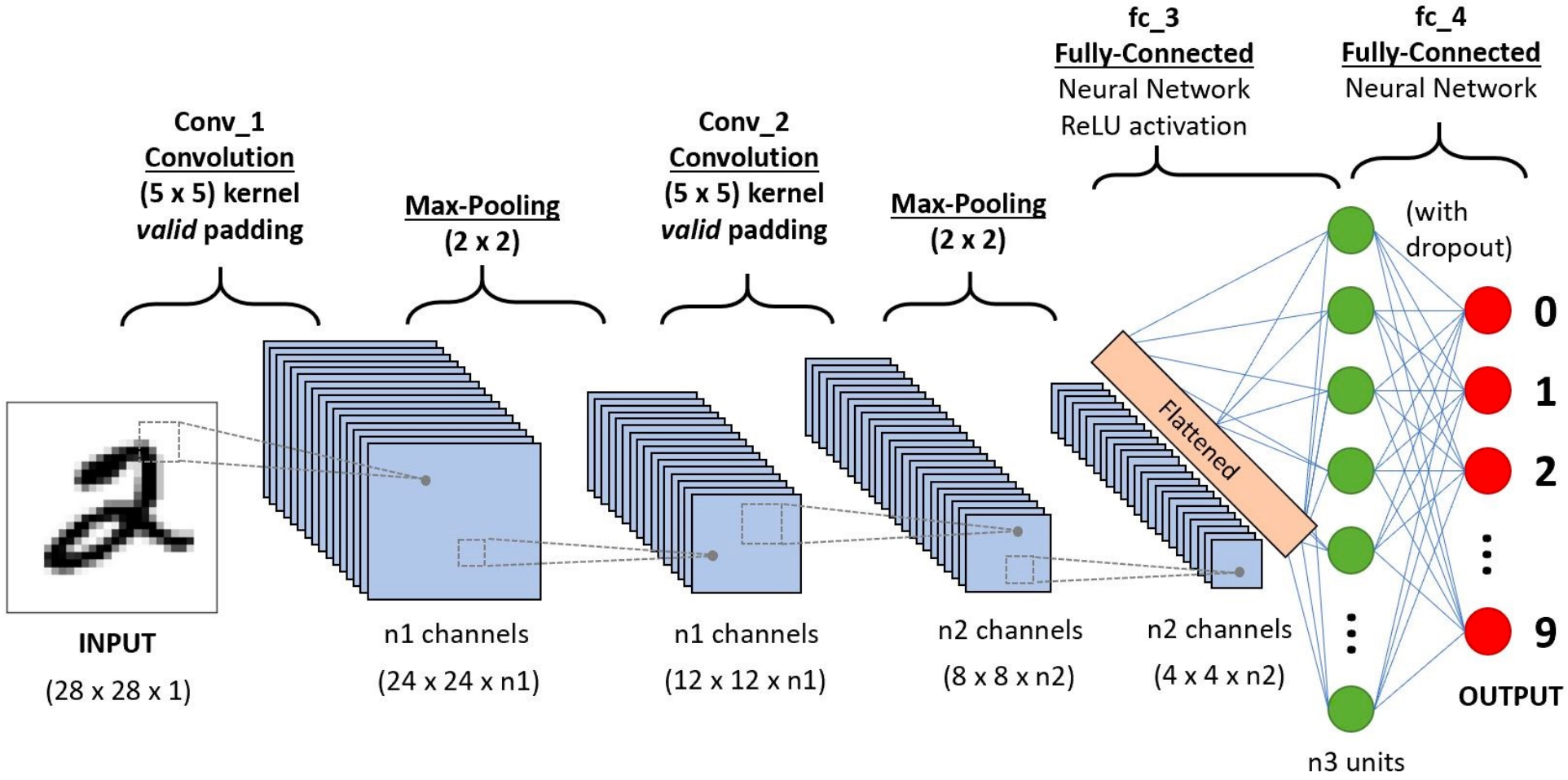
CNN: Pooling Layer

- Pooling (Subsampling)
  - Pool hidden units in the same neighborhood
    - Introduces invariance to local translations
    - Reduces the number of hidden units in hidden layer



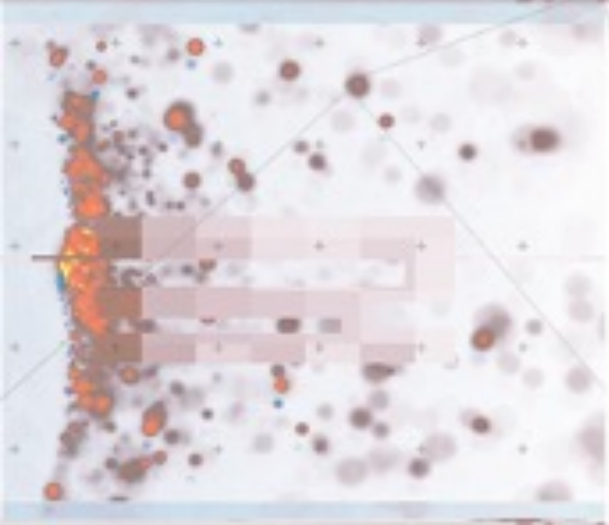
Ack. figure adapted from: https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks

CNN for Image Recognition: Example



An example CNN for handwritten digit recognition

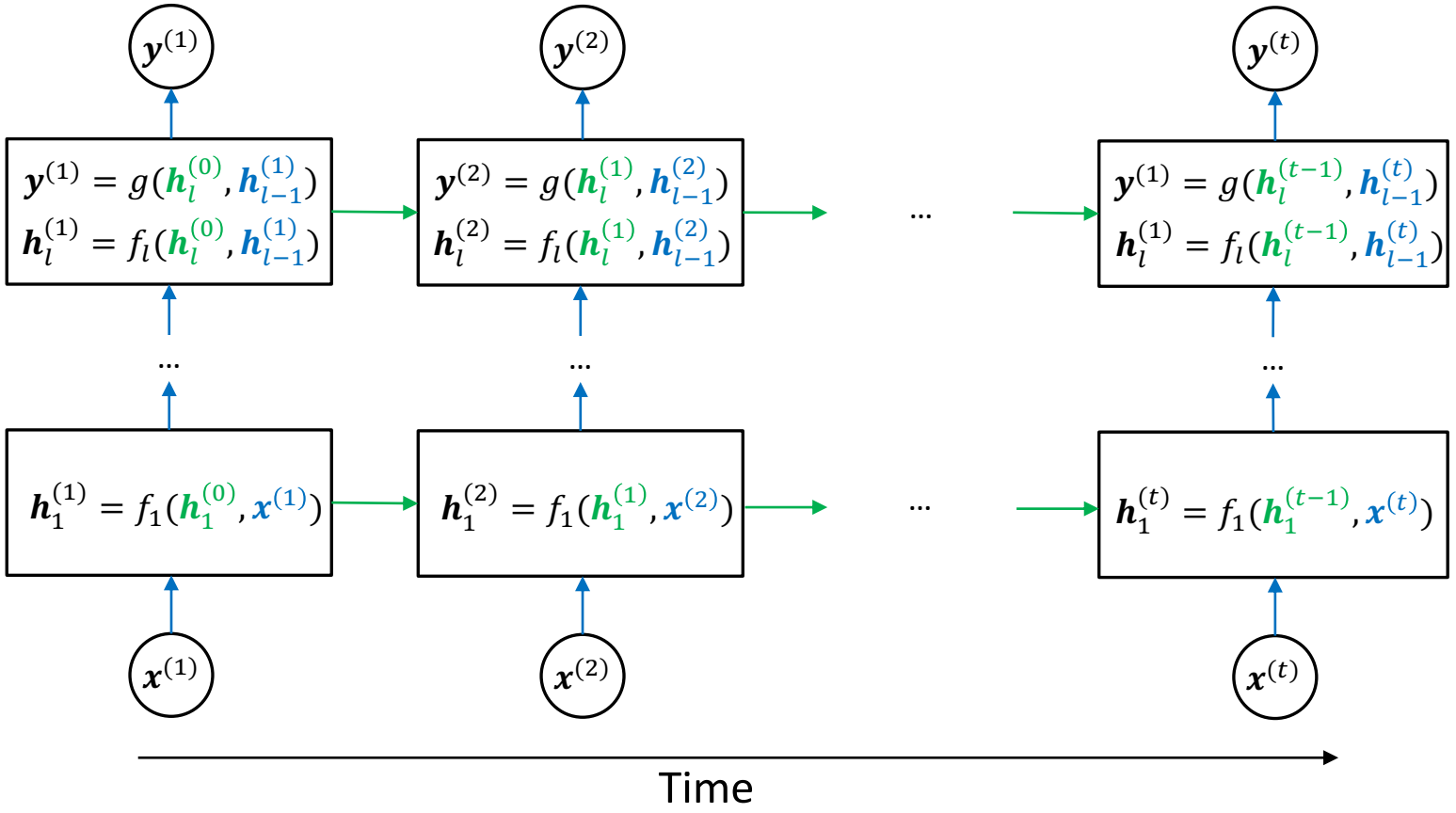
Ack. figure adapted from: https://laptrinhx.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-755556893/



Deep Learning: Recurrent Neural Networks

Recurrent Neural Networks

- Handling sequences with **Recurrent Neural Networks (RNN)**
  - At each time step, the input and the previous hidden state are fed into the network



Recurrent Neural Networks: General Concepts

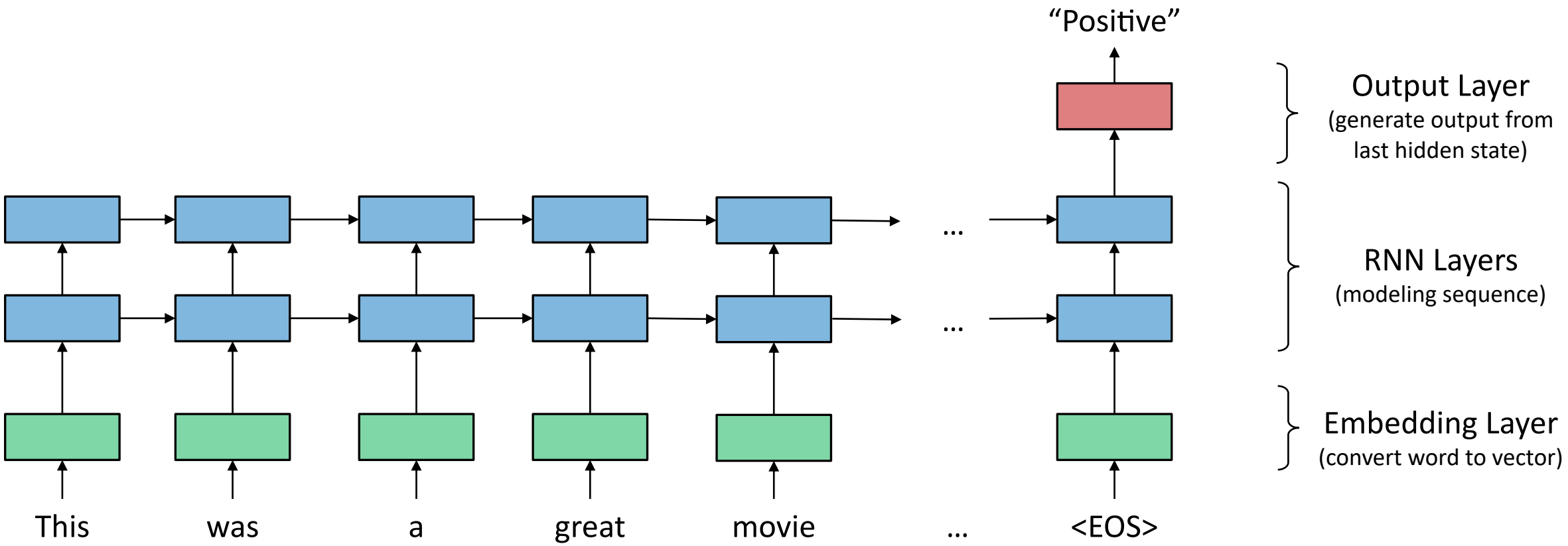
- Modeling the time dimension:
  - **Feedback loops** connected to past decisions
  - **Long-term dependencies**: Use hidden states to preserve sequential information
- RNNs are trained to model sequences: Output at each timestamp t is based on the current input ($\boldsymbol{x}^{(t)}$) and the previous state ($\boldsymbol{h}^{(t-1)}$)

$$\boldsymbol{h}^{(t)} = \sigma(\boldsymbol{W}\boldsymbol{x}^{(t)} + \boldsymbol{U}\boldsymbol{h}^{(t-1)})$$

  - Compute a gradient with the algorithm BPTT (backpropagation through time)
- Major obstacles of RNN: Vanishing and Exploding Gradients
  - When the gradient becomes too large or too small, it is difficult to model long-range dependencies (10 timestamps or more)
  - Solution: Use a variant of RNN: LSTM (1997, by Hochreiter and Schmidthuber)

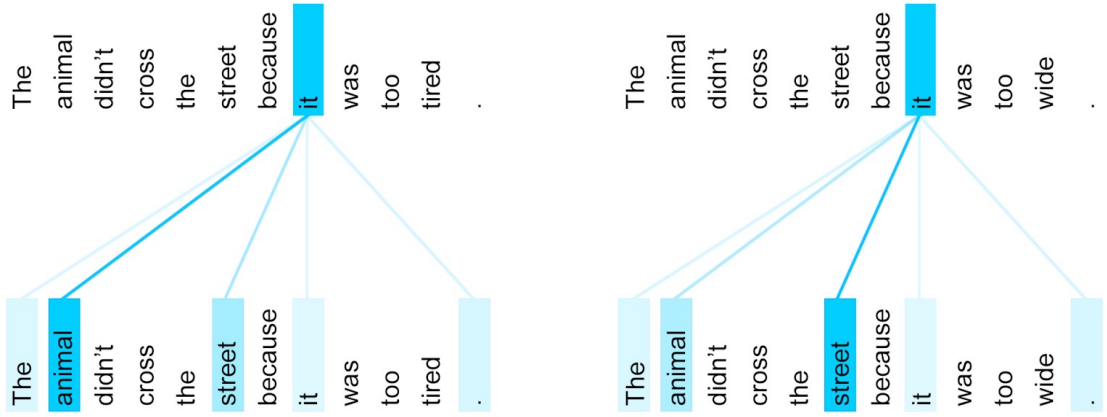
Ref.: https://deeplearning4j.org/lstm.html

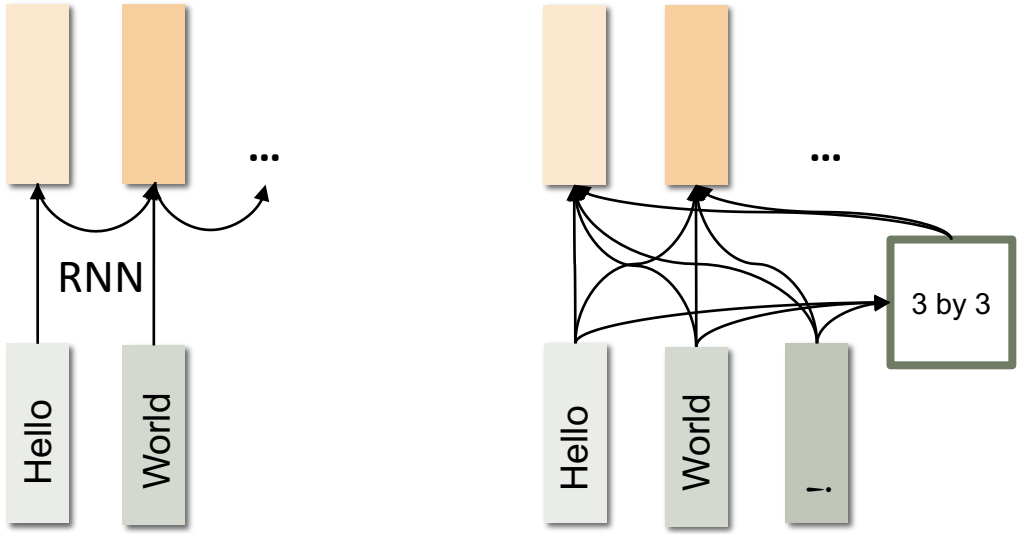
RNN for Sentiment Classification: Example



State-of-the-art Model: Transformers

- Problem with RNNs:
  - The output vector at timestamp *t* depends on the output at timestamp *t – 1*
  - It presumes an order for inputs and prevents parallelization for scalable training
- Solution: Attention is all you need (https://arxiv.org/abs/1706.03762)
  - Explicitly calculate dependencies & handle sequential inputs without “recurrent”
  - The independent computation of outputs allows parallelization





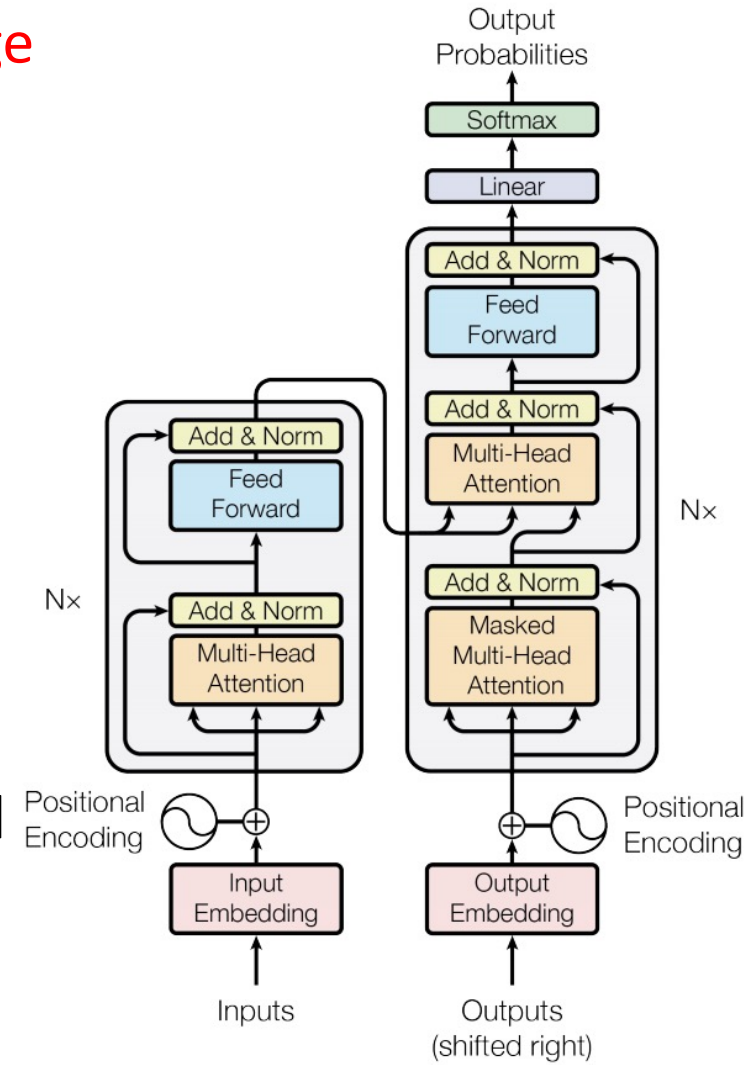
Reading material: The Illustrated Transformer

https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

More reading: https://www.programmersought.com/article/10956032922/

Transformer + Language Model

- The Transformer architecture enables scalable training of large models, but where can we find big data to train the models?
- Language Model: learning from unlabeled text data
  - Task: Given a sentence with randomly masked words, the model is asked to read the seen words (encoder), and predict the masked words (decoder)
    - Ex. The man [MASK] to the store.
  - All existing text corpora can be our training data!
  - The learned (pretrained) encoder/decoder serves as a good initialization of models for downstream applications
- Reading materials
  - Transformer → BERT → GPT-2



Ack. Figures from Vaswani et al., *NIPS* 2017

Summary: Deep Learning Recap

- Pros
  - Very good performance on certain tasks, for certain types of data
    - Images: image recognition, segmentation, …
    - Text (sometimes): machine translation, language modeling,…
    - …
  - Requires very little feature engineering
  - Good generalization
    - Ex. models trained on ImageNet dataset for classification can help tasks such as segmentation
- Cons
  - Requires huge amounts of computation power
  - Black box model
  - Hard to tune the architecture and hyperparameters for new tasks