Introduction to Machine Learning

Yifeng Tao

School of Computer Science

Carnegie Mellon University

Logistics

- Course website:

http://www.cs.cmu.edu/~yifengt/courses/machine-learning

Slides uploaded after lecture

- Time: Mon-Fri 9:50-11:30am lecture, 11:30-12:00pm discussion
- Contact: yifengt@cs.cmu.edu

Introduction to Machine Learning

Course Information

- Instructor: Yifeng Tao
- Time: Mon-Fri 9:50-11:30AM / 9:50-12:00PM, May 13-24 2019
- Location: Institute of Industrial and Systems Engineering, Northeastern University

Course Description

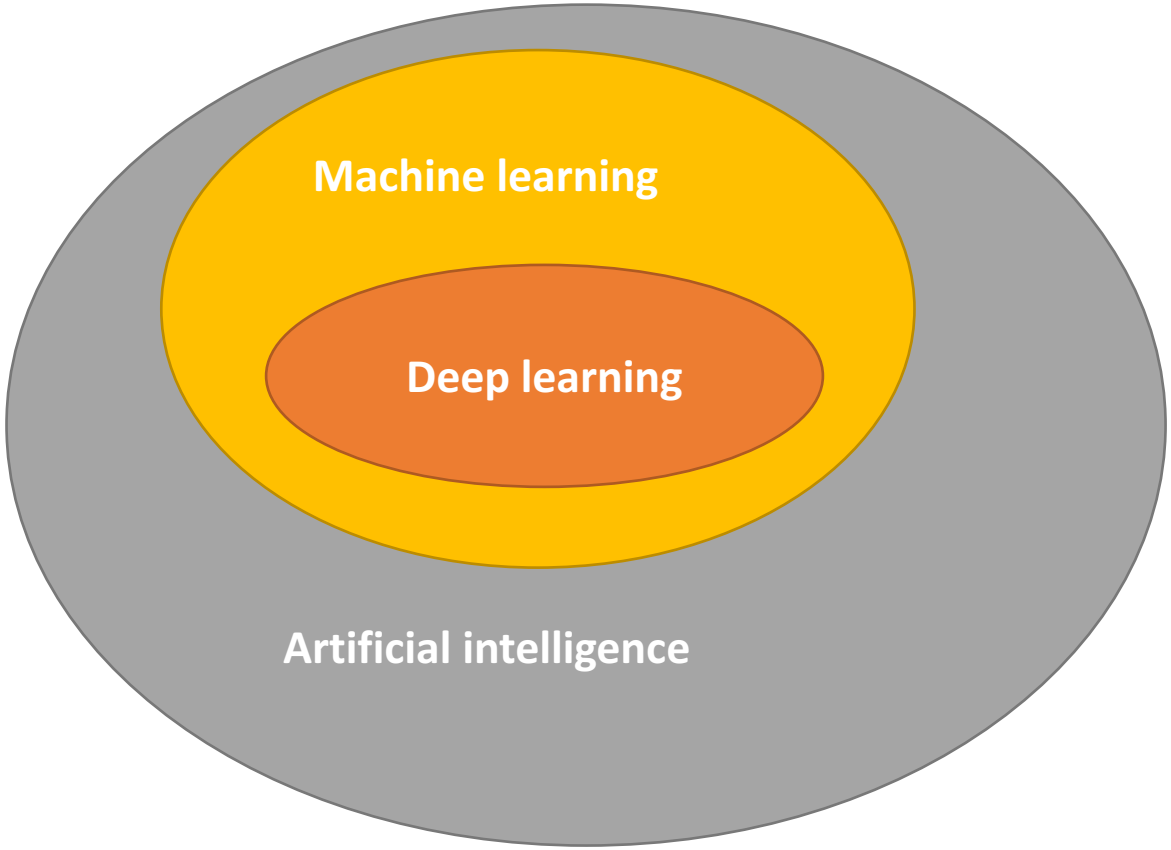
The recent advancement of machine learning, especially the development of deep learning, has essentially influenced the area of computer vision, natural language processing, and computational biology. In this series of lectures and seminars of "Introduction to Machine Learning", I will introduce the general knowledge of machine learning, such as supervised learning, unsupervised learning, deep learning, as well as specific topics of machine learning application in precision medicine and clinical text mining.

Syllabus

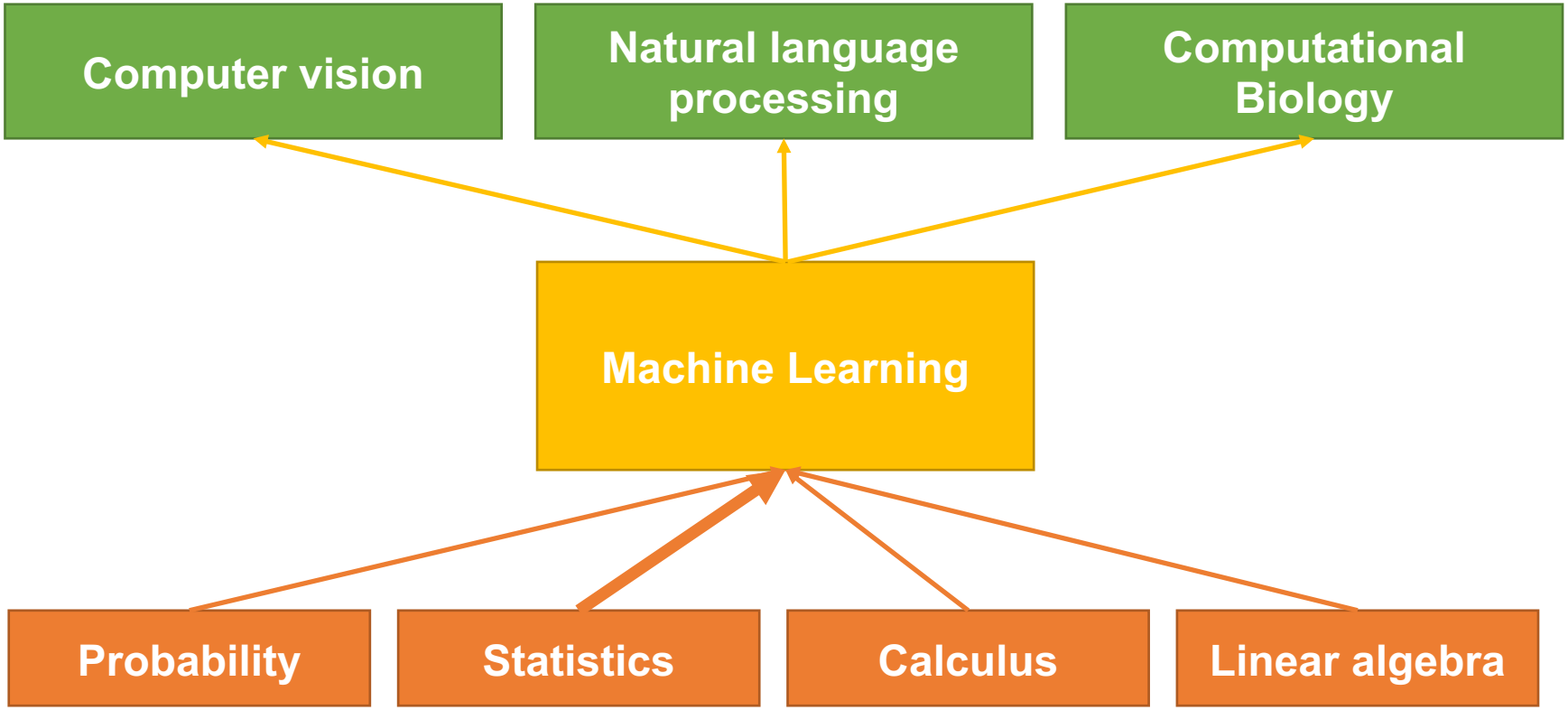
Date	Lecture	Topics	Slides
Mon, May 13	Lecture 1	Supervised learning: linear models	[Link]
Tue, May 14	Lecture 2	Kernel machines: SVMs and duality	[Link]
Wed, May 15	Lecture 3	Unsupervised learning: latent space analysis and clustering	[Link]
Thu, May 16	Lecture 4	Decision tree, kNN and model selection	[Link]
Fri, May 17	Lecture 5	Learning theory	[Link]
Mon, May 20	Lecture 6	Neural network (basics)	[]
Tue, May 21	Lecture 7	Deep learning in CV and NLP	[]
Wed, May 22	Lecture 8	Probabilistic graphical models	[]
Thu, May 23	Lecture 9	Reinforcement learning and its application in clinical text mining	[]
Fri, May 24	Lecture 10	Attention mechanism and transfer learning in precision medicine	[]

What is machine learning?

- What are we talking when we talk about AI and ML?



What is machine learning

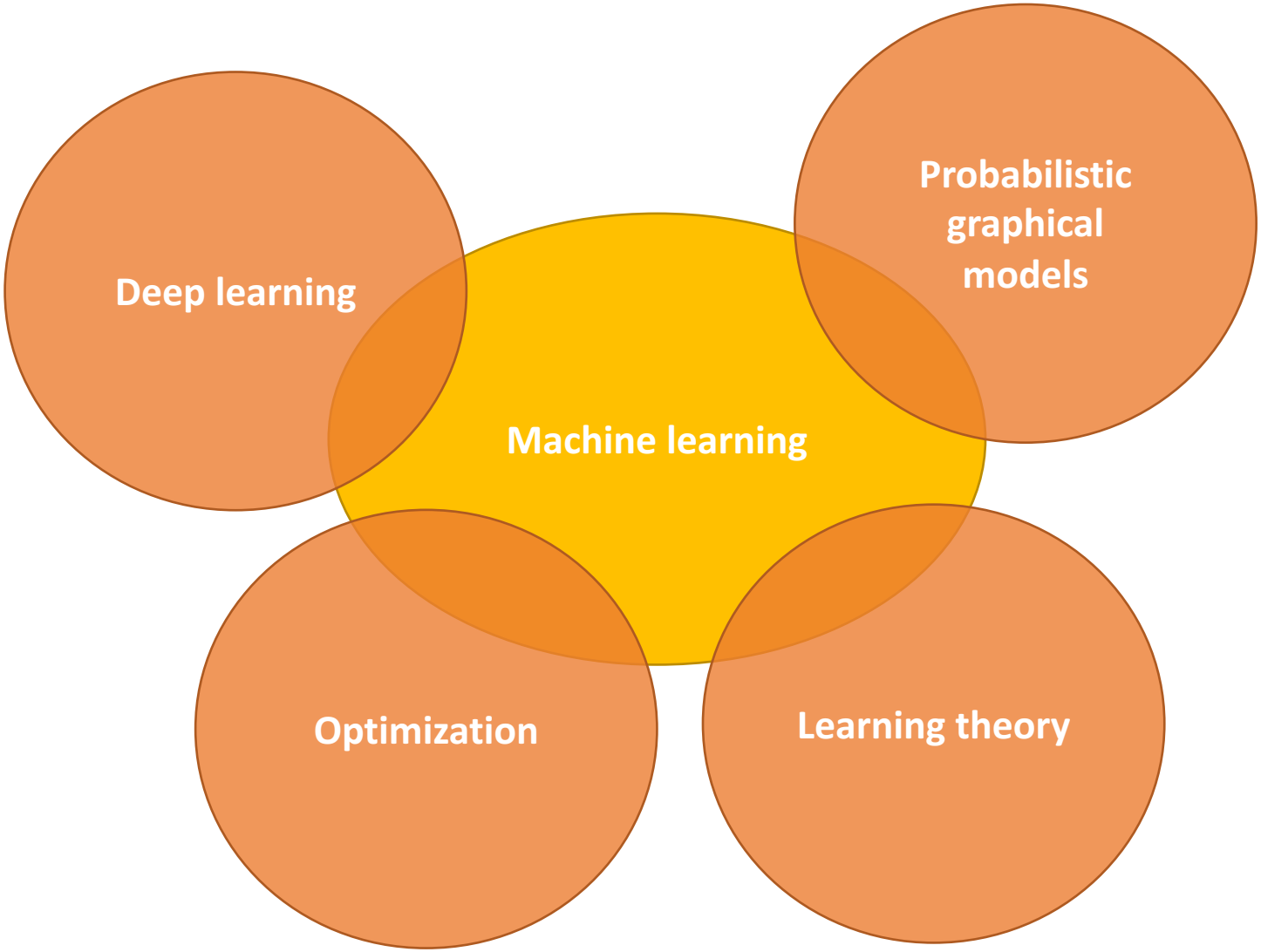


Where are we?

- Supervised learning: linear models
- Kernel machines: SVMs and duality
- Unsupervised learning: latent space analysis and clustering
- Supervised learning: decision tree, kNN and model selection
- Learning theory: generalization and VC dimension
- **Neural network (basics)**
- Deep learning in CV and NLP
- Probabilistic graphical models

- Reinforcement learning and its application in clinical text mining
- Attention mechanism and transfer learning in precision medicine

What's more after introduction?



What's more after introduction?

- Supervised learning: linear models
- Kernel machines: SVMs and duality
- → **Optimization**
- Unsupervised learning: latent space analysis and clustering
- Supervised learning: decision tree, kNN and model selection
- Learning theory: generalization and VC dimension
- → **Statistical machine learning**
- Neural network (basics)
- Deep learning in CV and NLP
- → **Deep learning**
- **Probabilistic graphical models**

Curriculum for an ML Master/Ph.D. student in CMU

- 10701 Introduction to Machine Learning:
  - http://www.cs.cmu.edu/~epxing/Class/10701/
- 35705 Intermediate Statistics:
  - http://www.stat.cmu.edu/~larry/=stat705/
- 36708 Statistical Machine Learning:
  - http://www.stat.cmu.edu/~larry/=sml/
- 10725 Convex Optimization:
  - http://www.stat.cmu.edu/~ryantibs/convexopt/
- 10708 Probabilistic Graphical Models:
  - http://www.cs.cmu.edu/~epxing/Class/10708-17/
- 10707 Deep Learning:
  - https://deeplearning-cmu-10707.github.io/
- Books:
  - Bishop. Pattern Recognition and Machine Learning
  - Goodfellow et al. Deep learning

Neural network (basics)

Yifeng Tao

School of Computer Science

Carnegie Mellon University

Slides adapted from Eric Xing, Maria-Florina Balcan, Russ Salakhutdinov, Matt Gormley

A Recipe for Supervised Learning

- 1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{N}$$

- 2. Choose each of these:
  - Decision function $\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$
  - Loss function $\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$

- 3. Define goal and train with SGD:
  - (take small steps opposite the gradient)

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

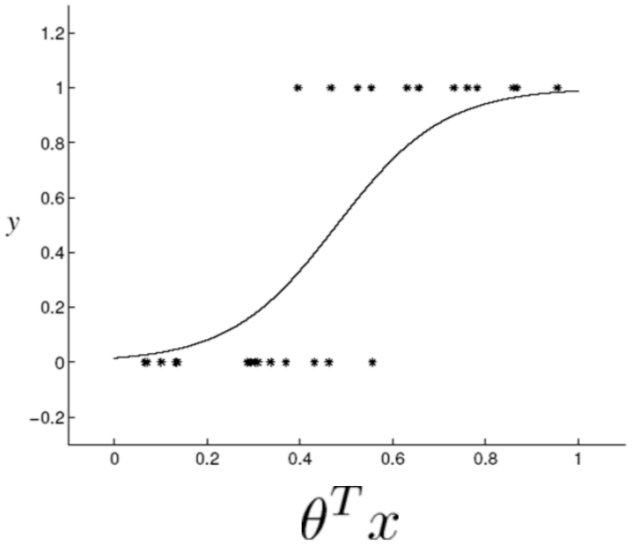
[Slide from Matt Gormley et al.]

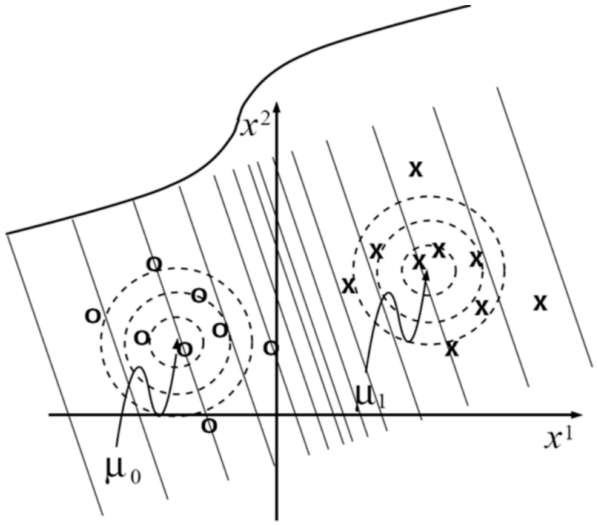
Logistic Regression

- The prediction rule:

$$p(y=1 \mid x_n) = \frac{1}{1+\exp\left\{-\sum_{i=1}^{M}\theta_i x_i - \theta_0\right\}} = \frac{1}{1+e^{-\theta^T x}}$$

- In this case, learning *P(y|x)* amounts to learning conditional probability over two Gaussian distribution.
- Limitation: only simple data distribution.



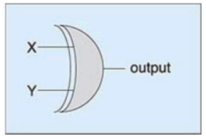


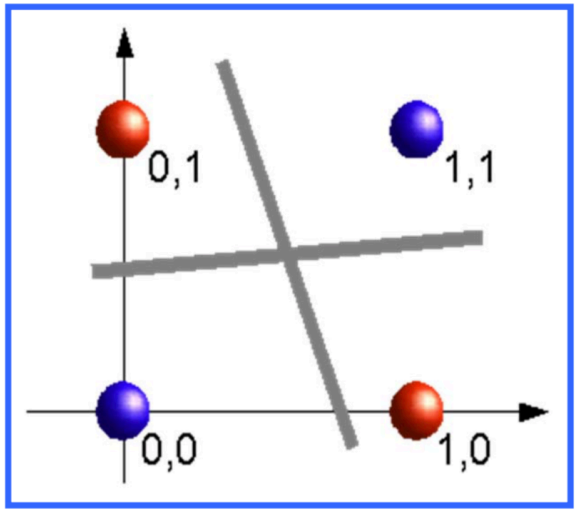
[Slide from Eric Xing et al.]

Learning highly non-linear functions

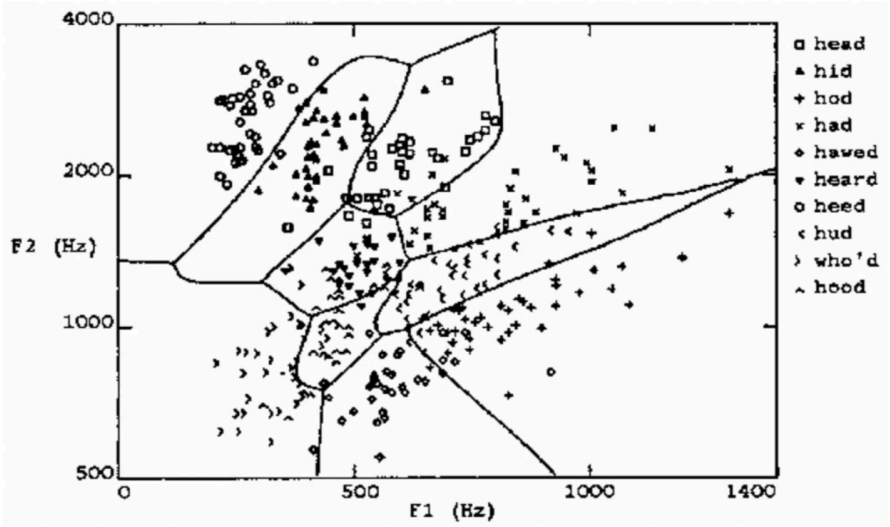
- f: X → y
- f might be non-linear function
- X continuous or discrete vars
- y continuous or discrete vars

The XOR gate



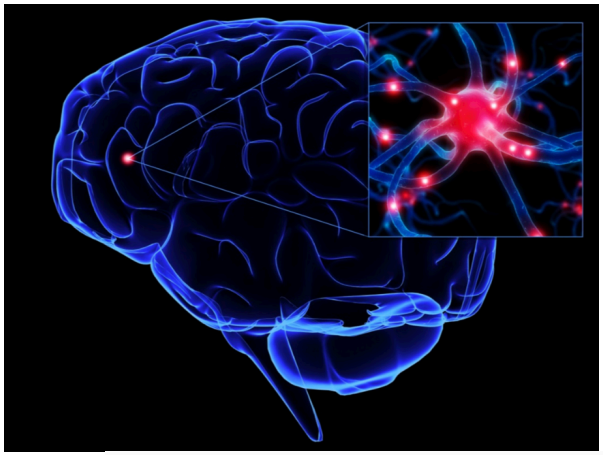


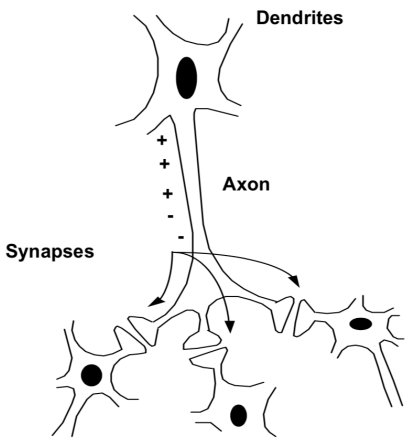
Speech recognition

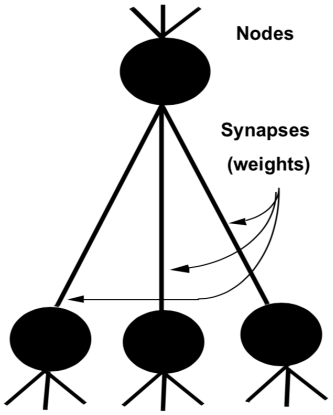


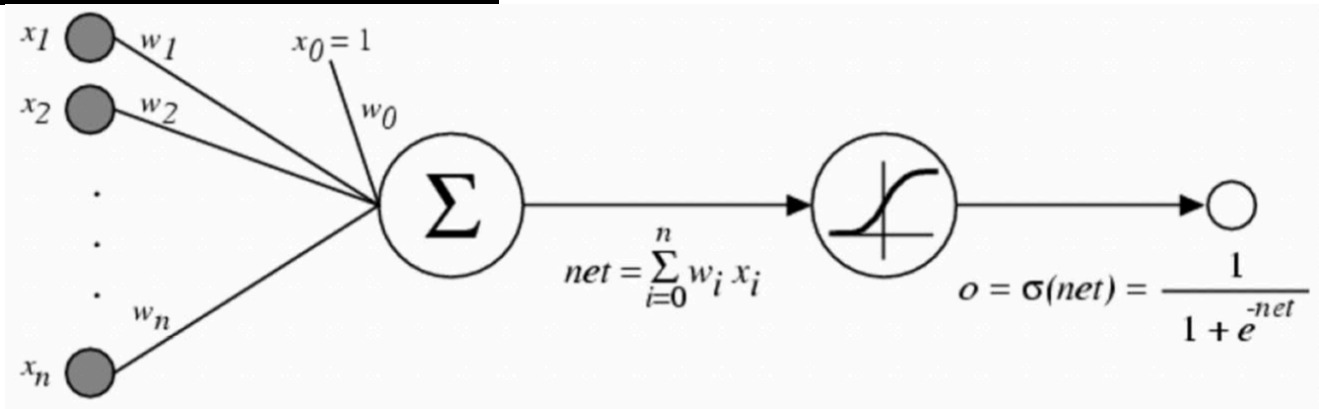
[Slide from Eric Xing et al.]

From biological neuron networks to artificial neural networks







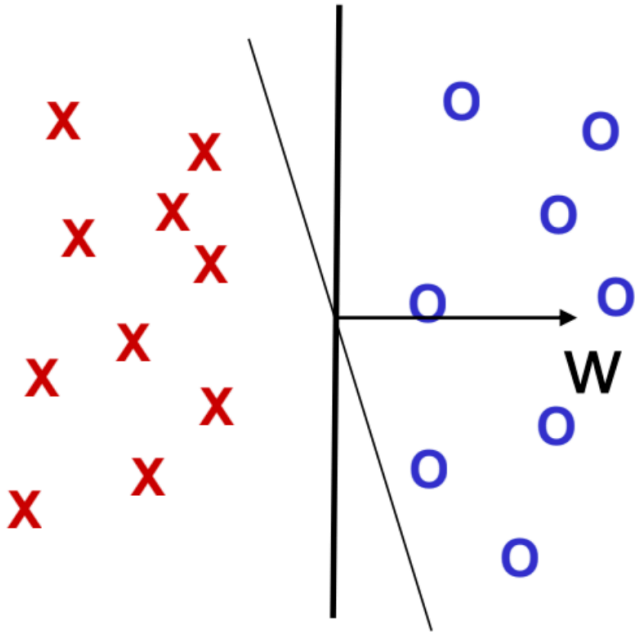


- Signals propagate through neurons in brain.
- Signals propagate through perceptrons in artificial neural network.

[Slide from Eric Xing et al.]

Perceptron Algorithm and SVM

- Perceptron: simple learning algorithm for supervised classification analyzed via geometric margins in the 50's [Rosenblatt'57].
- Similar to SVM, a linear classifier based on analysis of margins.
- Originally introduced in the online learning scenario.
  - Online learning model
  - Its guarantees under large margins

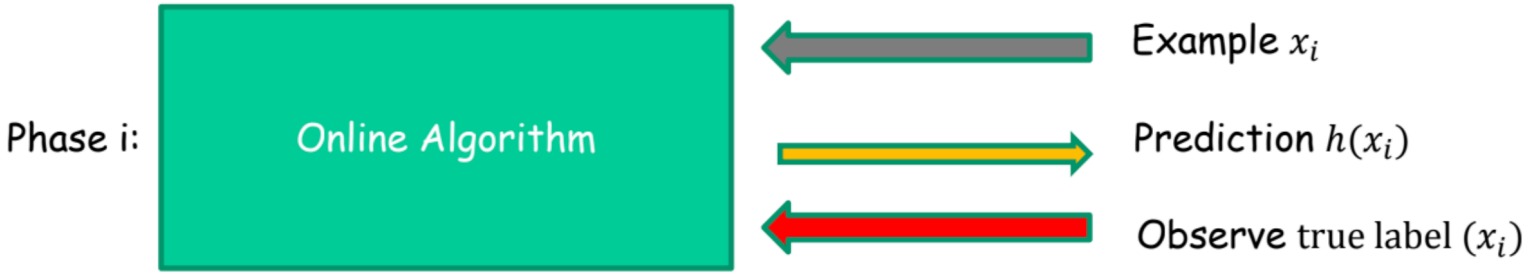


[Slide from Maria-Florina Balcan et al.]

The Online Learning Algorithm

- Example arrive sequentially.
- We need to make a prediction.
- Afterwards observe the outcome.

- For i=1, 2, ..., :



- Application:
  - Email classification
  - Recommendation systems
  - Add placement in a new market

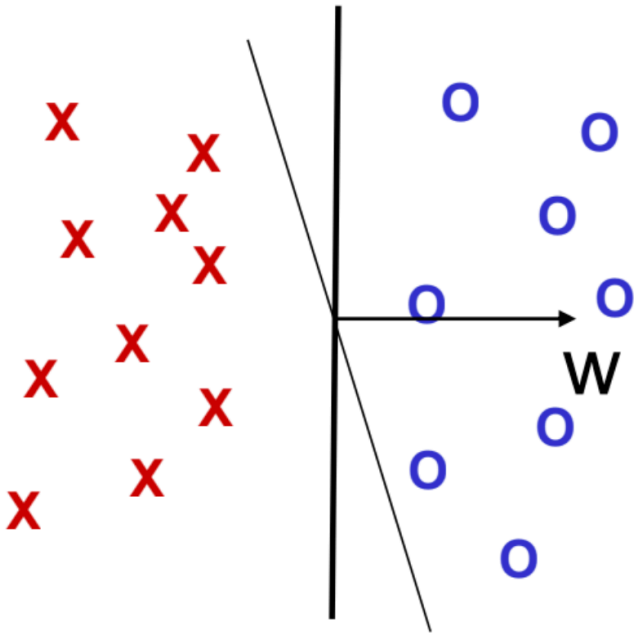
[Slide from Maria-Florina Balcan et al.]

Linear Separators: Perceptron Algorithm

- $h(x) = w^T x + w_0$, if $h(x) \geq 0$, then label x as +, otherwise label it as –

- Set t=1, start with the all zero vector w_1.
- Given example x, predict positive iff $w_t^T x \geq 0$
- On a mistake, update as follows:
  - Mistake on positive, then update $w_{t+1} \leftarrow w_t + x$
  - Mistake on negative, then update $w_{t+1} \leftarrow w_t - x$

- Natural greedy procedure:
  - If true label of x is +1 and w_t incorrect on x we have $w_t^T x < 0$, $w_{t+1}^T x \leftarrow w_t^T x + x^T x = w_t^T x + ||x||^2$, so more chance w_{t+1} classifies x correctly.
  - Similarly for mistakes on negative examples.



[Slide from Maria-Florina Balcan et al.]

Perceptron: Example and Guarantee

- **Example:**

$(-1,2)$ − ✗ $w_1 = (0,0)$

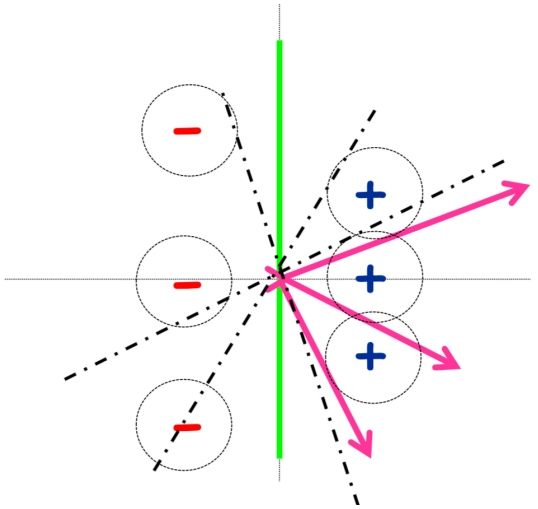
$(1,0)$ + ✓ $w_2 = w_1 - (-1,2) = (1,-2)$

$(1,1)$ + ✗ $w_3 = w_2 + (1,1) = (2,-1)$

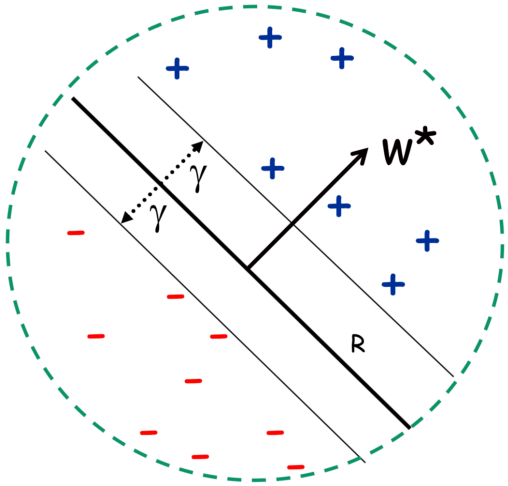
$(-1,0)$ − ✓

$(-1,-2)$ − ✗ $w_4 = w_3 - (-1,-2) = (3,1)$

$(1,-1)$ + ✓



- **Guarantee**: If data has margin γ and all points inside a ball of radius R, then Perceptron makes ≤ $(R/\gamma)^2$ mistakes.
  - Normalized margin: multiplying all points by 100, or dividing all points by 100, doesn't change the number of mistakes; algo is invariant to scaling.



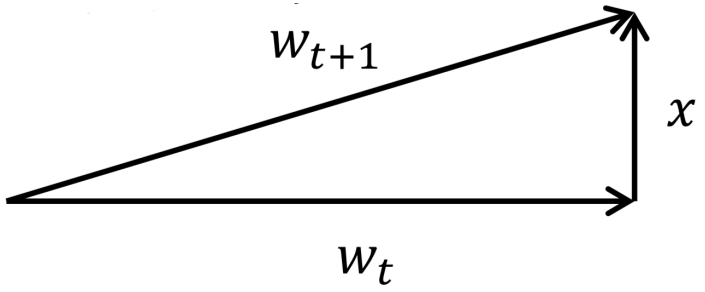
[Slide from Maria-Florina Balcan et al.]

Perceptron: Proof of Mistake Bound

- **Guarantee**: If data has margin γ and all points inside a ball of radius R, then Perceptron makes $\leq (R/\gamma)^2$ mistakes.
- **Proof:**
  - **Idea**: analyze $w_t^T w^*$ and $\|w_t\|$, where w^* is the max-margin sep, $\|w^*\|=1$.
  - Claim 1: $w_{t+1}^T w^* \geq w_t^T w^* + \gamma$. (because $x^T w^* \geq \gamma$)
  - Claim 2: $w_{t+1}^2 \leq w_t^2 + R^2$. (by Pythagorean Theorem)

  - After M mistakes:
  - $w_{M+1}^T w^* \geq \gamma M$ (by Claim 1)
  - $\|w_{M+1}\| \leq R\sqrt{M}$ (by Claim 2)
  - $w_{M+1}^T w^* \leq \|w_{M+1}\|$ (since w^* is unit length)

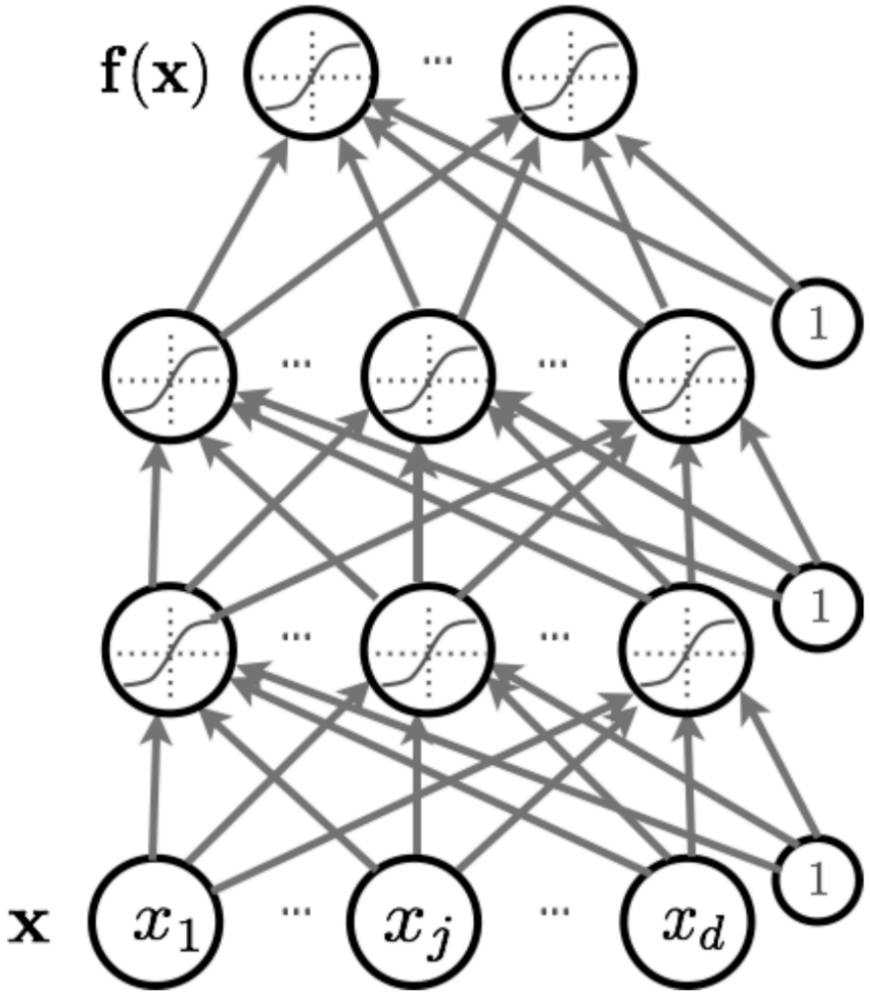
  - So, $\gamma M \leq RM$, so $M \leq (R/\gamma)^2$.



[Slide from Maria-Florina Balcan et al.]

Multilayer perceptron (MLP)

- A simple and basic type of feedforward neural networks
- Contains many perceptrons that are organized into layers
- MLP "perceptrons" are not perceptrons in the strict sense



[Slide from Russ Salakhutdinov et al.]

Artificial Neuron (Perceptron)

- Neuron pre-activation (or input activation):

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

- Neuron output activation:

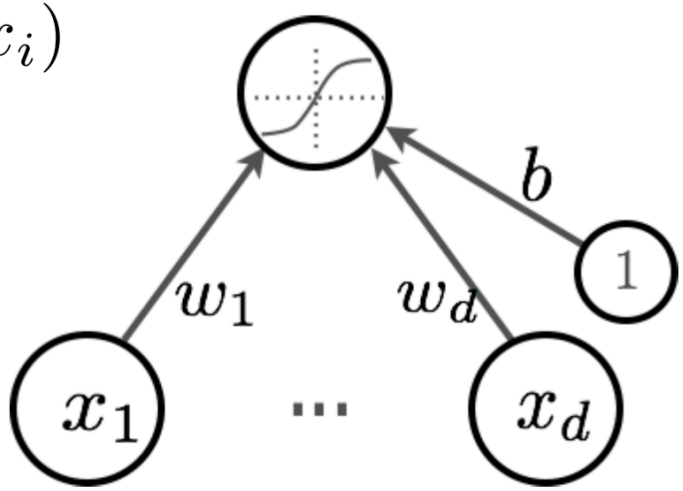
$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

where

$\mathbf{w}$ are the weights (parameters)

b is the bias term

$g(\cdot)$ is called the activation function



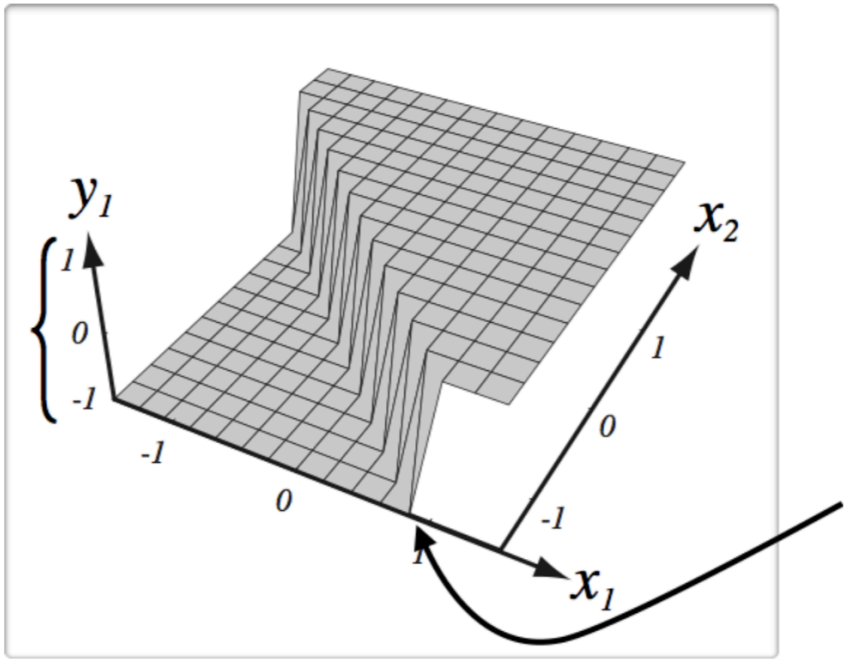
[Slide from Russ Salakhutdinov et al.]

Artificial Neuron (Perceptron)

- Output activation of the neuron:

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

Range is determined by $g(\cdot)$



(from Pascal Vincent's slides)

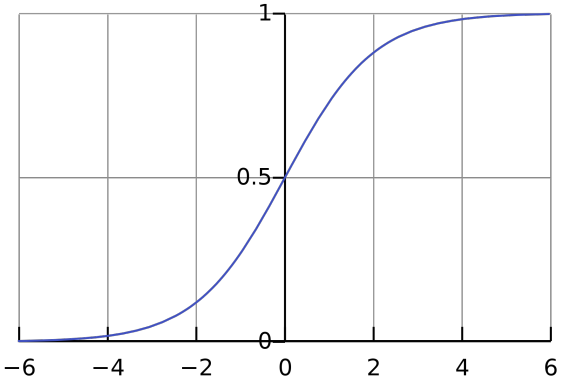
Bias only changes the position of the riff

[Slide from Russ Salakhutdinov et al.]

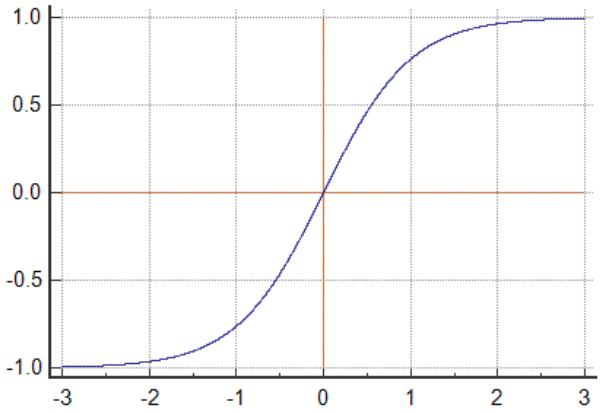
Activation Function

- sigmoid activation function:
  - Squashes the neuron's output between 0 and 1
  - Always positive
  - Bounded
  - Strictly increasing
  - Used in classification output layer

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



- tanh activation function:
  - Squashes the neuron's output between -1 and 1
  - Bounded
  - Strictly increasing
  - A linear transformation of sigmoid function

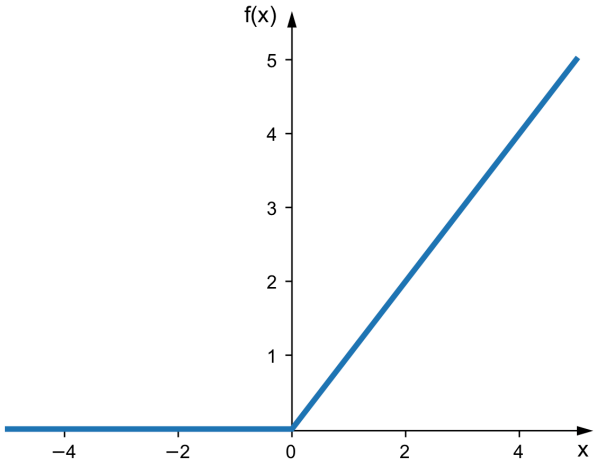


[Slide from Russ Salakhutdinov et al.]

Activation Function

- Rectified linear (ReLU) activation:
  - Bounded below by 0 (always non-negative)
  - Tends to produce units with sparse activities
  - Not upper bounded
  - Strictly increasing

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

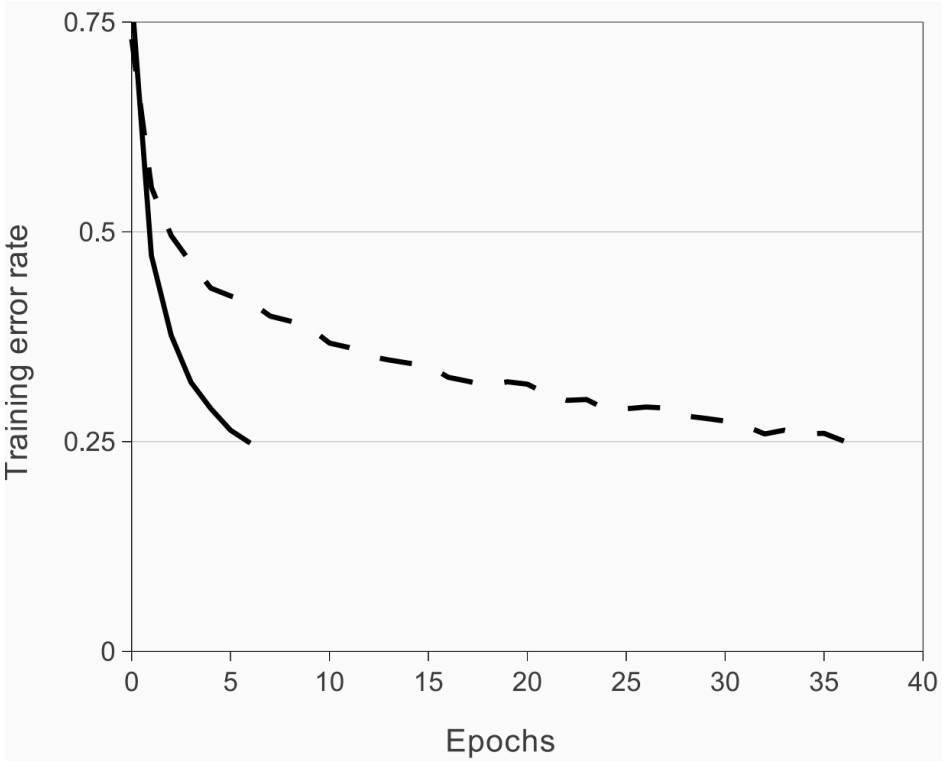


- Most widely used activation function
- Advantages:
  - Biological plausibility
  - Sparse activation
  - Better gradient propagation: vanishing gradient in sigmoidal activation

[Slide from Russ Salakhutdinov et al.]

Activation Function in Alexnet

- A four-layer convolutional neural network
  - ReLU: solid line
  - Tanh: dashed line



[Slide from https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf]

Single Hidden Layer MLP

- Hidden layer pre-activation:

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

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

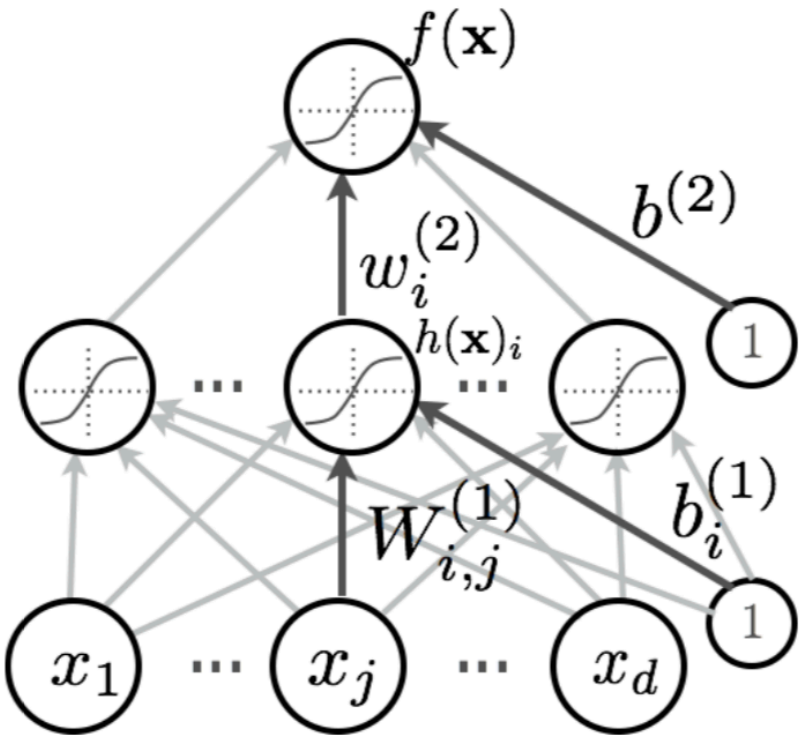
- Hidden layer activation:

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

- Output layer activation:

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

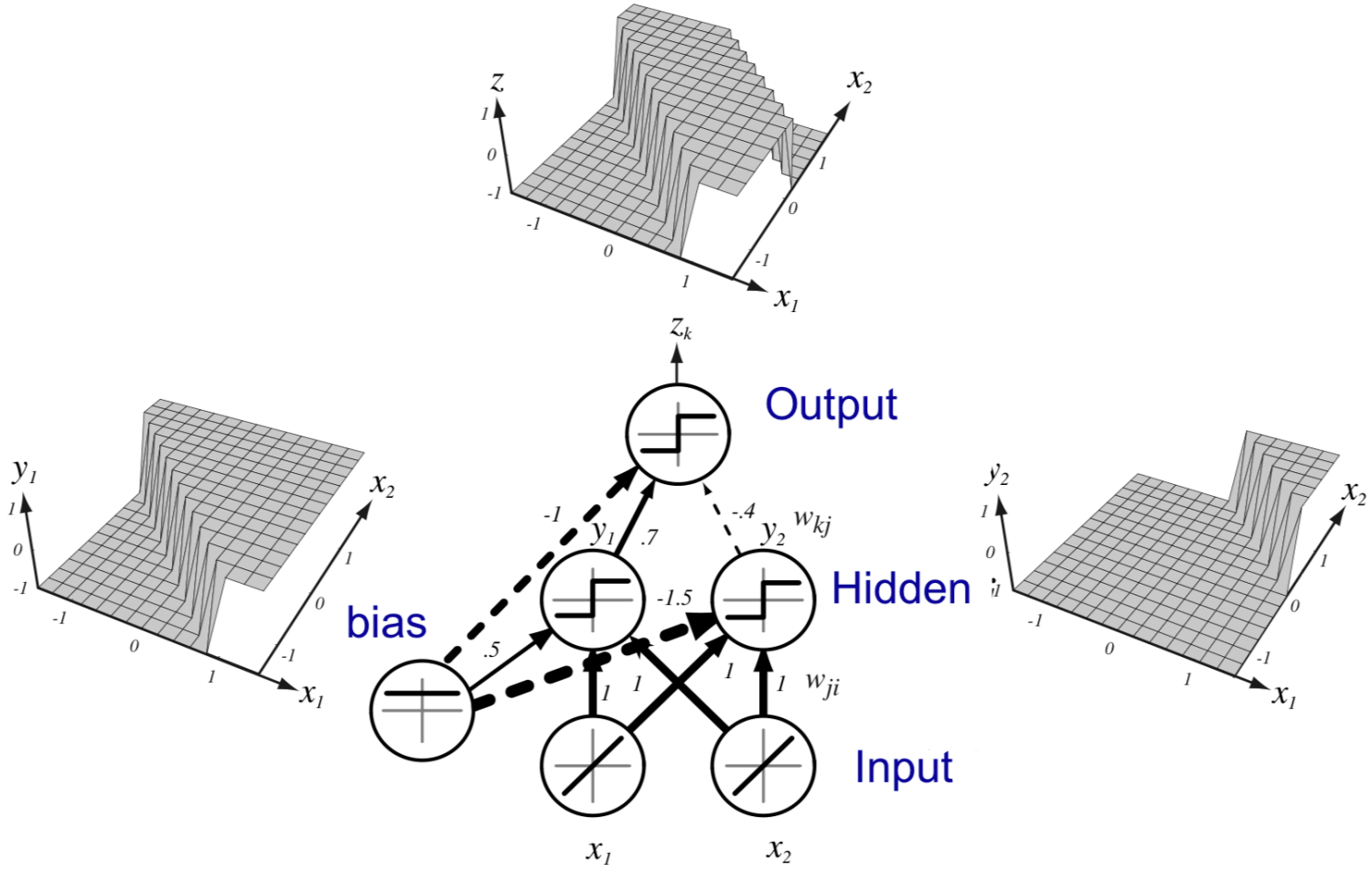
Output activation function



[Slide from Russ Salakhutdinov et al.]

Capacity of MLP

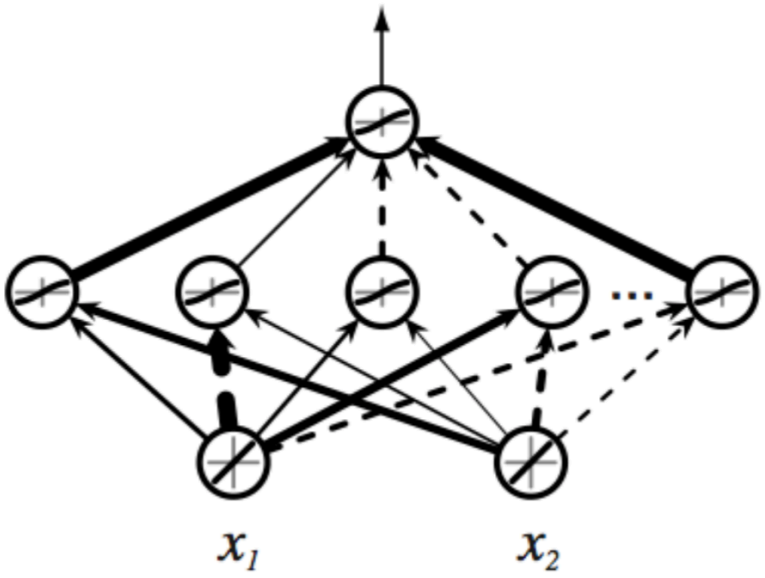
- Consider a single layer neural network

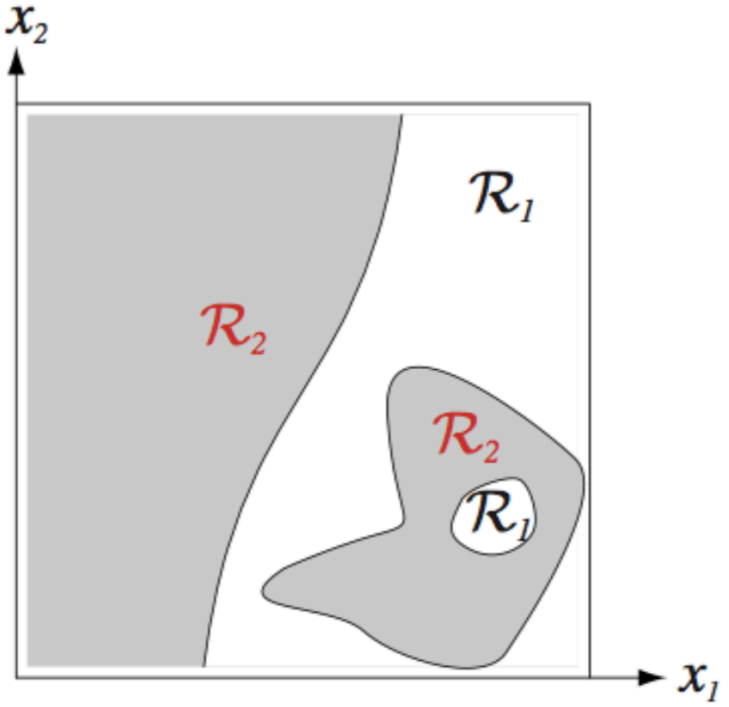


[Slide from Russ Salakhutdinov et al.]

Capacity of Neural Nets

- Consider a single layer neural network





[Slide from Russ Salakhutdinov et al.]

MLP with Multiple Hidden Layers

- Consider a network with L hidden layers.
  - layer pre-activation for k>0

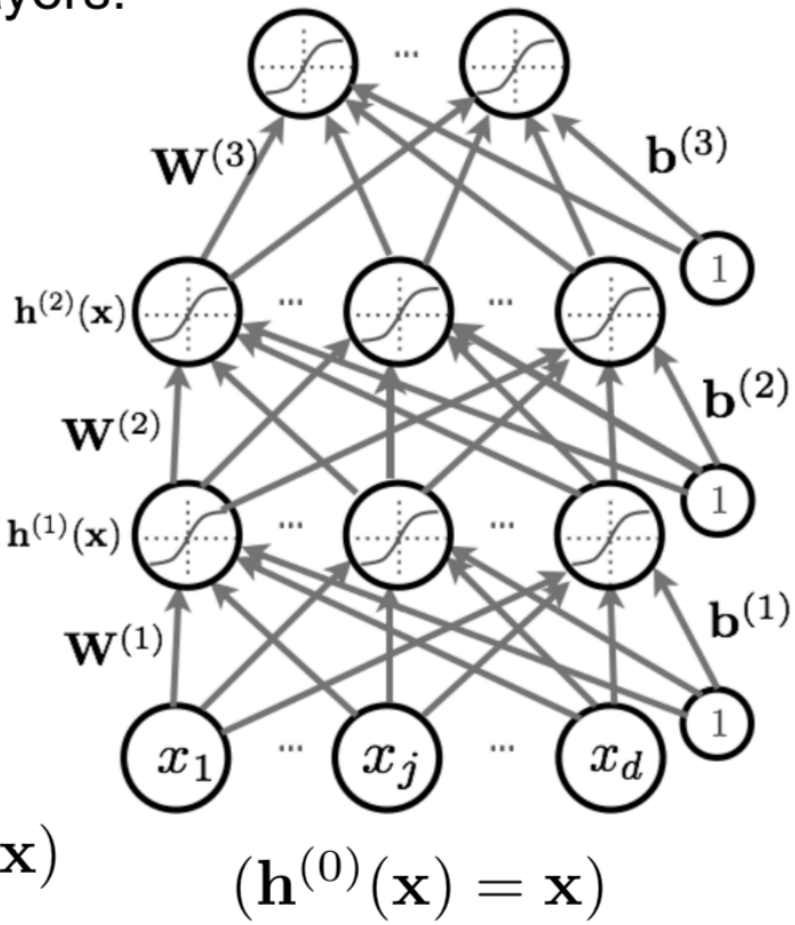
$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$

  - hidden layer activation from 1 to L:

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

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

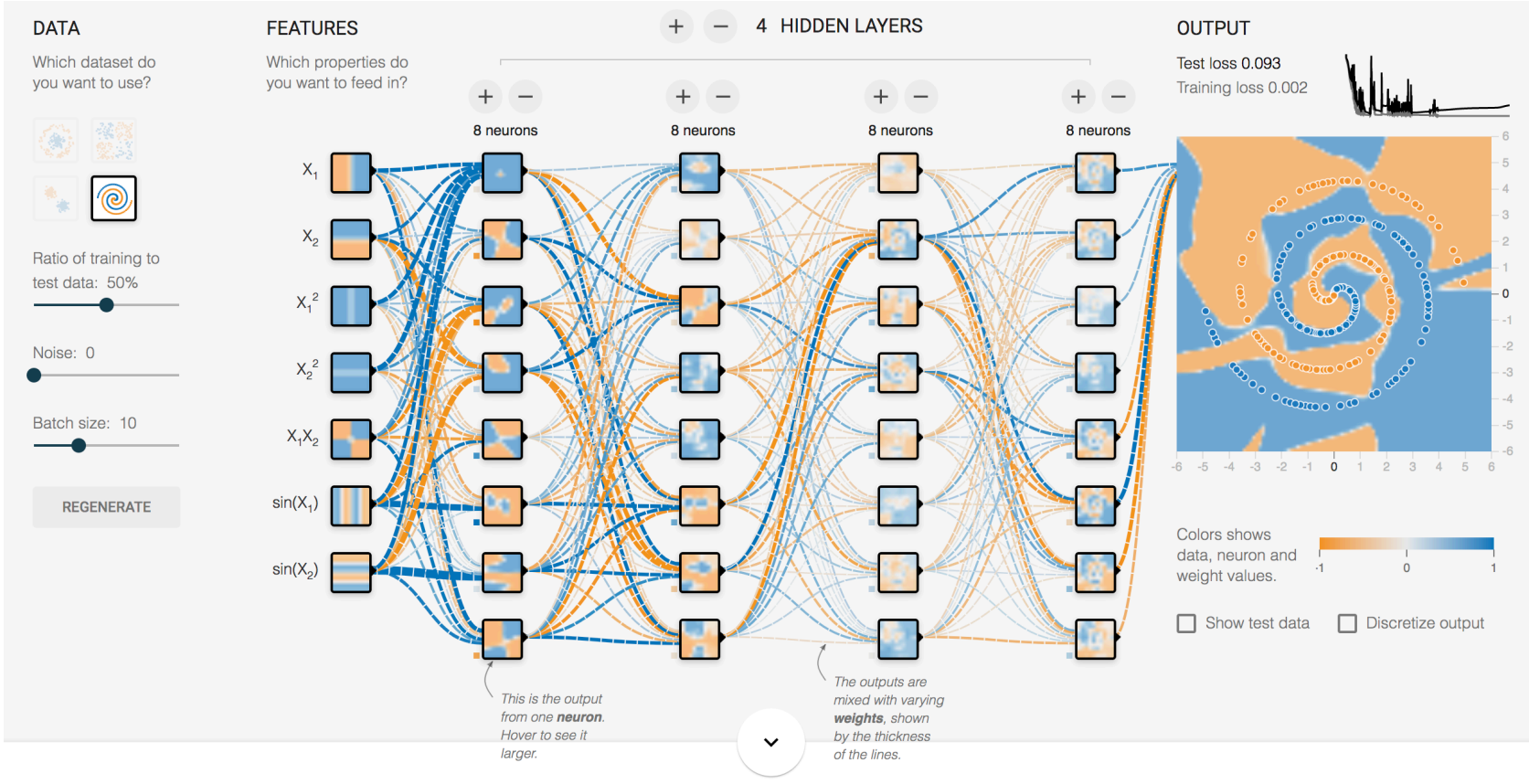
$$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$



[Slide from Russ Salakhutdinov et al.]

Capacity of Neural Nets

- Deep learning playground



[Slide from https://playground.tensorflow.org]

Training a Neural Network

- Empirical Risk Minimization:

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

- To train a neural net, we need:
  - Loss function: $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
  - A procedure to compute gradients: $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
  - Regularizer and its gradient: $\Omega(\boldsymbol{\theta}), \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$

[Slide from Russ Salakhutdinov et al.]

Stochastic Gradient Descent

- Perform updates after seeing each example:
  - Initialize: $\boldsymbol{\theta} \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \ldots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$
  - For t=1:T
    - for each training example $(\mathbf{x}^{(t)}, y^{(t)})$

$$\Delta = -\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) - \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \Delta$$

Training epoch = Iteration of all examples

- To train a neural net, we need:
  - Loss function: $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
  - A procedure to compute gradients: $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
  - Regularizer and its gradient: $\Omega(\boldsymbol{\theta}), \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$

[Slide from Russ Salakhutdinov et al.]

Mini-batch SGD

- Make updates based on a mini-batch of examples (instead of a single example)
  - The gradient is the average regularized loss for that mini-batch
  - Can give a more accurate estimate of the gradient
  - Can leverage matrix/matrix operations, which are more efficient

[Slide from Russ Salakhutdinov et al.]

Backpropagation

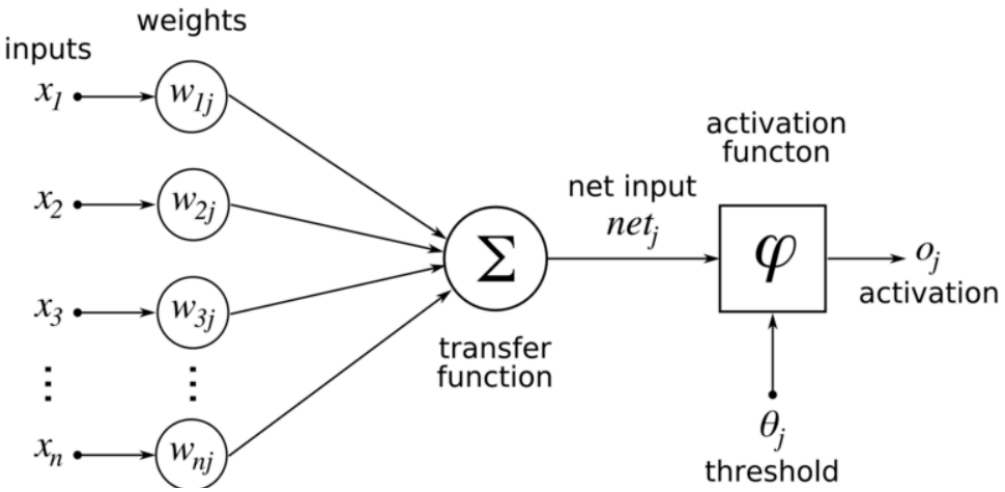
- Method used to train neural networks following gradient descent
- Essentially implementation of chain rule and dynamic programming

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\boxed{\frac{\partial o_j}{\partial \text{net}_j}\frac{\partial \text{net}_j}{\partial w_{ij}}}$$

- The derivative of last two terms:

$$\frac{\partial \text{net}_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\left(\sum_{k=1}^{n} w_{kj} o_k\right) = \frac{\partial}{\partial w_{ij}} w_{ij} o_i = o_i$$

$$\frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial \varphi(\text{net}_j)}{\partial \text{net}_j}$$



[Slide from https://en.wikipedia.org/wiki/Backpropagation]

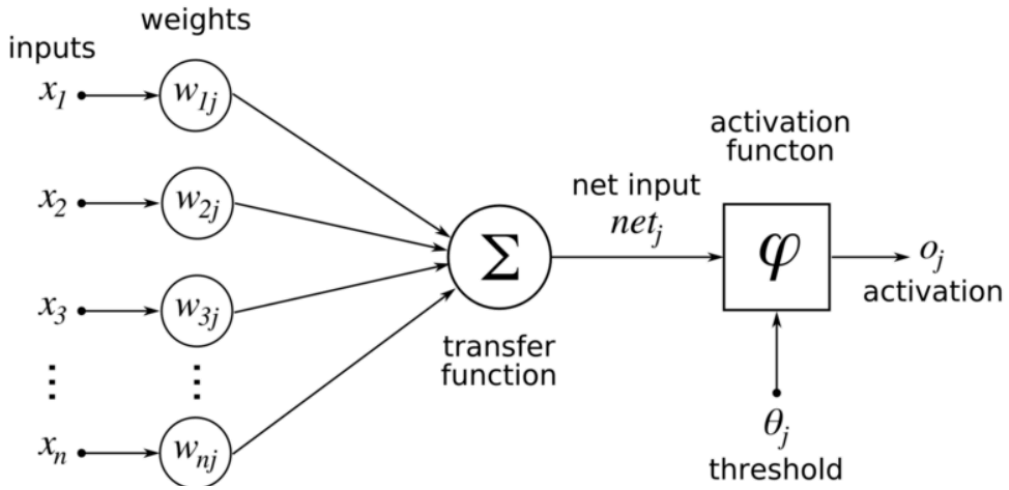
Backpropagation

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial w_{ij}} = \boxed{\frac{\partial E}{\partial o_j}}\frac{\partial o_j}{\partial \text{net}_j}\frac{\partial \text{net}_j}{\partial w_{ij}}$$

- If o_j is output → straightforward
- Else:

$$\frac{\partial E(o_j)}{\partial o_j} = \frac{\partial E(\text{net}_u, \text{net}_v, \ldots, \text{net}_w)}{\partial o_j}$$

$$\frac{\partial E}{\partial o_j} = \sum_{\ell \in L}\left(\frac{\partial E}{\partial \text{net}_\ell}\frac{\partial \text{net}_\ell}{\partial o_j}\right) = \sum_{\ell \in L}\left(\frac{\partial E}{\partial o_\ell}\frac{\partial o_\ell}{\partial \text{net}_\ell}\frac{\partial \text{net}_\ell}{\partial o_j}\right) = \sum_{\ell \in L}\left(\frac{\partial E}{\partial o_\ell}\frac{\partial o_\ell}{\partial \text{net}_\ell}w_{j\ell}\right)$$



[Slide from https://en.wikipedia.org/wiki/Backpropagation]

Weight Decay

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

- L2 regularization:

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

- L1 regularization:

$$\Omega(\boldsymbol{\theta}) = \sum_k \sum_i \sum_j |W_{i,j}^{(k)}|$$

  - Only applies to weights, not biases (weigh decay)

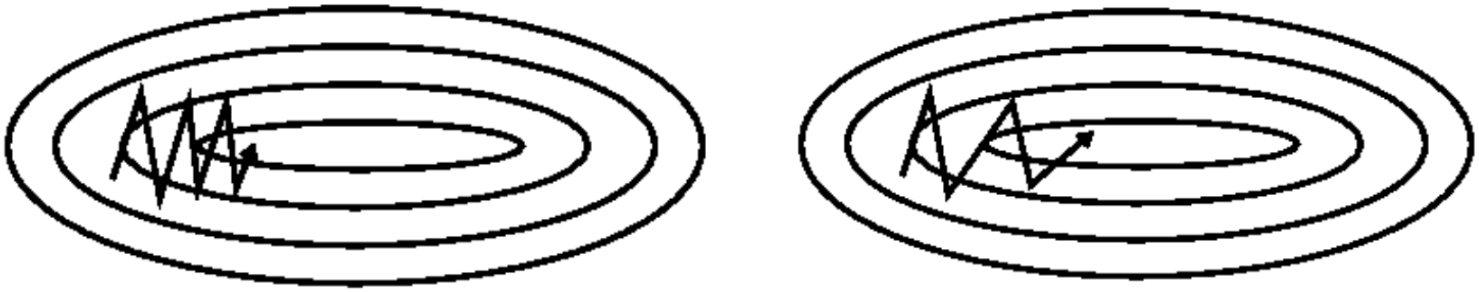
[Slide from Russ Salakhutdinov et al.]

Optimization: Momentum

- Momentum: Can use an exponential average of previous gradients:

$$\overline{\nabla}_{\boldsymbol{\theta}}^{(t)} = \nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}), y^{(t)}) + \beta \overline{\nabla}_{\boldsymbol{\theta}}^{(t-1)}$$

  - Can get pass plateous more quickly, by "gaining momentum"
  - Works well in positions with bad Hessian matrix
- SGD w/o momentum, SGD w/ momentum



[Slide from http://ruder.io/optimizing-gradient-descent/]

Momentum-based Optimization

- Nesterov accelerated gradient (NAG):

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$
$$\theta = \theta - v_t$$

- Adagrad:
  - smaller updates for params associated with frequently occurring features
  - larger updates for params associated with infrequent features

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

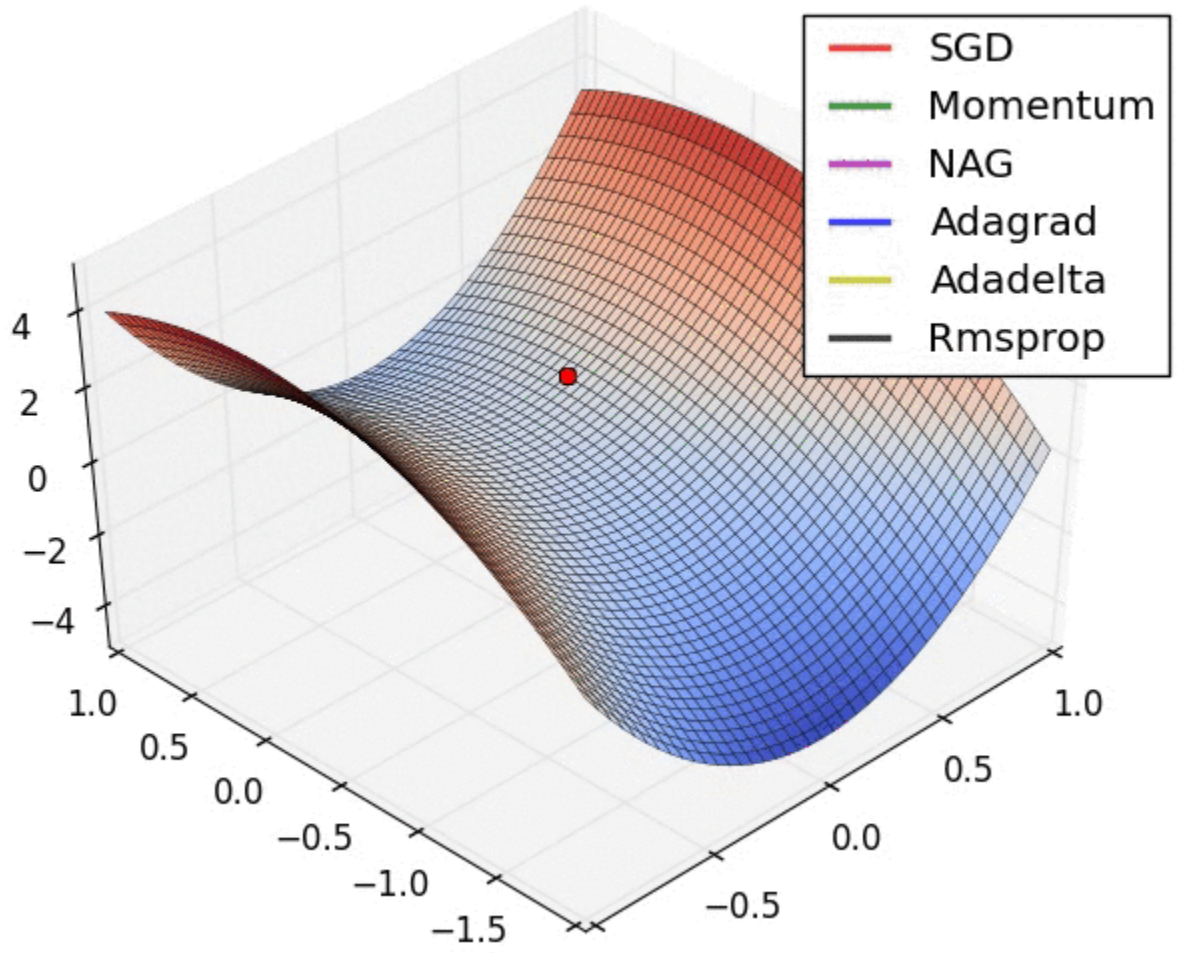
- RMSprop and Adadelta:
  - Reduce the aggressive, monotonically decreasing learning rate in Adagrad

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

- Adam

[Slide from http://ruder.io/optimizing-gradient-descent/]

Demo of Optimization Methods



[Slide from http://ruder.io/optimizing-gradient-descent/]

Take home message

- Perceptron is an online linear classifier
- Multilayer perceptron consists perceptrons with various activations
- Backpropagation is an implementation of calculating gradients of neural network in a backward and dynamic programming way
- Momentum-based mini-batch gradient descent methods are used in optimizing neural networks

- What's next?
  - Regularization in neural networks
  - Widely used NN architecture in practice

References

- Eric Xing, Tom Mitchell. 10701 Introduction to Machine Learning: http://www.cs.cmu.edu/~epxing/Class/10701-06f/
- Barnabás Póczos, Maria-Florina Balcan, Russ Salakhutdinov. 10715 Advanced Introduction to Machine Learning: https://sites.google.com/site/10715advancedmlintro2017f/lectures
- Matt Gormley. 10601 Introduction to Machine Learning: http://www.cs.cmu.edu/~mgormley/courses/10601/index.html
- Wikipedia