ELEC 825
Machine Learning and Deep Learning

Basics of Deep Learning

Xiaodan Zhu

ECE, Queen’s University

Fall, 2021

Announcement

- For course related questions, use the following email to contact me:

  elec825.instructor@queensu.ca

- Register your group.
- You can use the above spreadsheet to find group members (use the “find group members” tab at the bottom of the spreadsheet)
- Register topics
- **Is everyone in Kingston now?**

Last Lecture

- Introduction
  - About you, about me
  - What this course is about?
  - Course learning outcomes
  - How to evaluate?
  - Resources
- Machine Learning: An Overview
  - Types of learning tasks
  - Representing the world
  - Typical machine learning models
  - Deep learning

Last Lecture

- Introduction
  - About you, about me
  - What this course is about?
  - Course learning outcomes
  - How to evaluate?
  - Resources
- Machine Learning: An Overview
  - Types of learning tasks
  - Representing the world
  - Typical machine learning models
  - **Deep Learning**

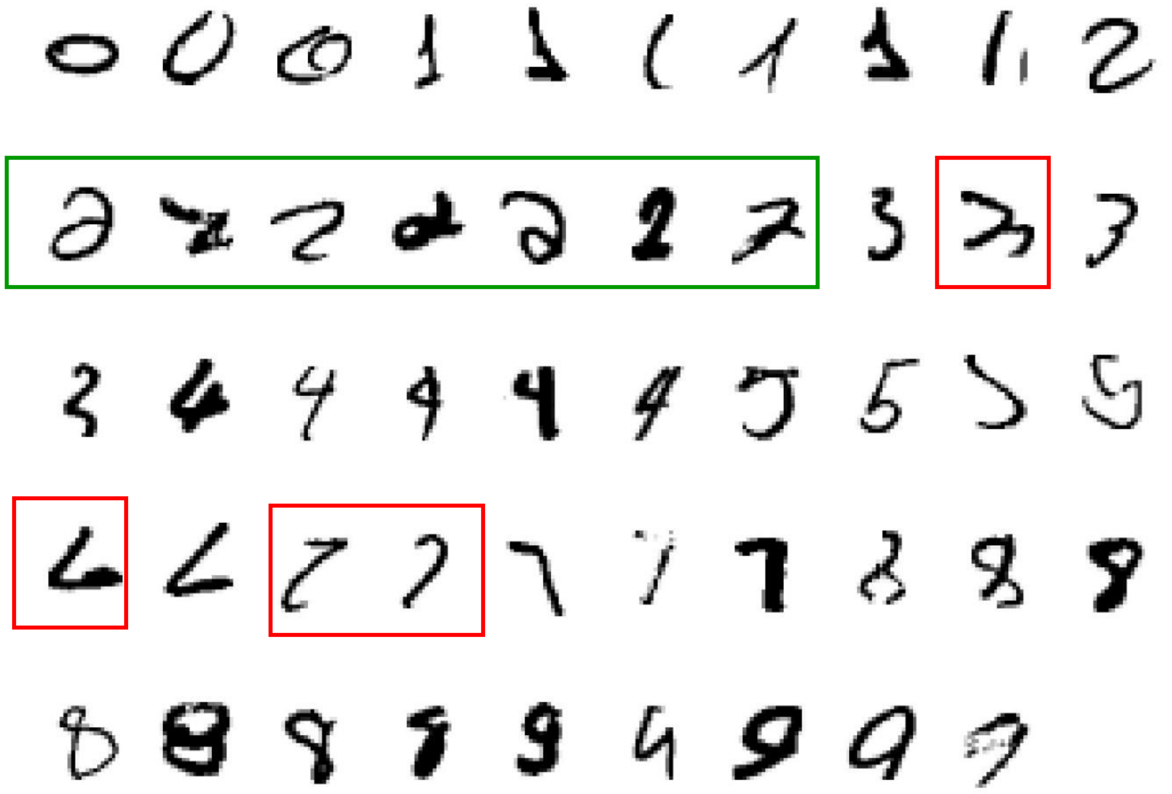
OnQ

- Main website for this course
- Information about this course will be posted on OnQ:
  - Slides
  - Seminar materials
  - Project information
  - Grades
  - …
- OnQ will be used to send course notifications
- On your OnQ page, please subscribe to receive instant notifications via email/SMS

Review the Last lecture

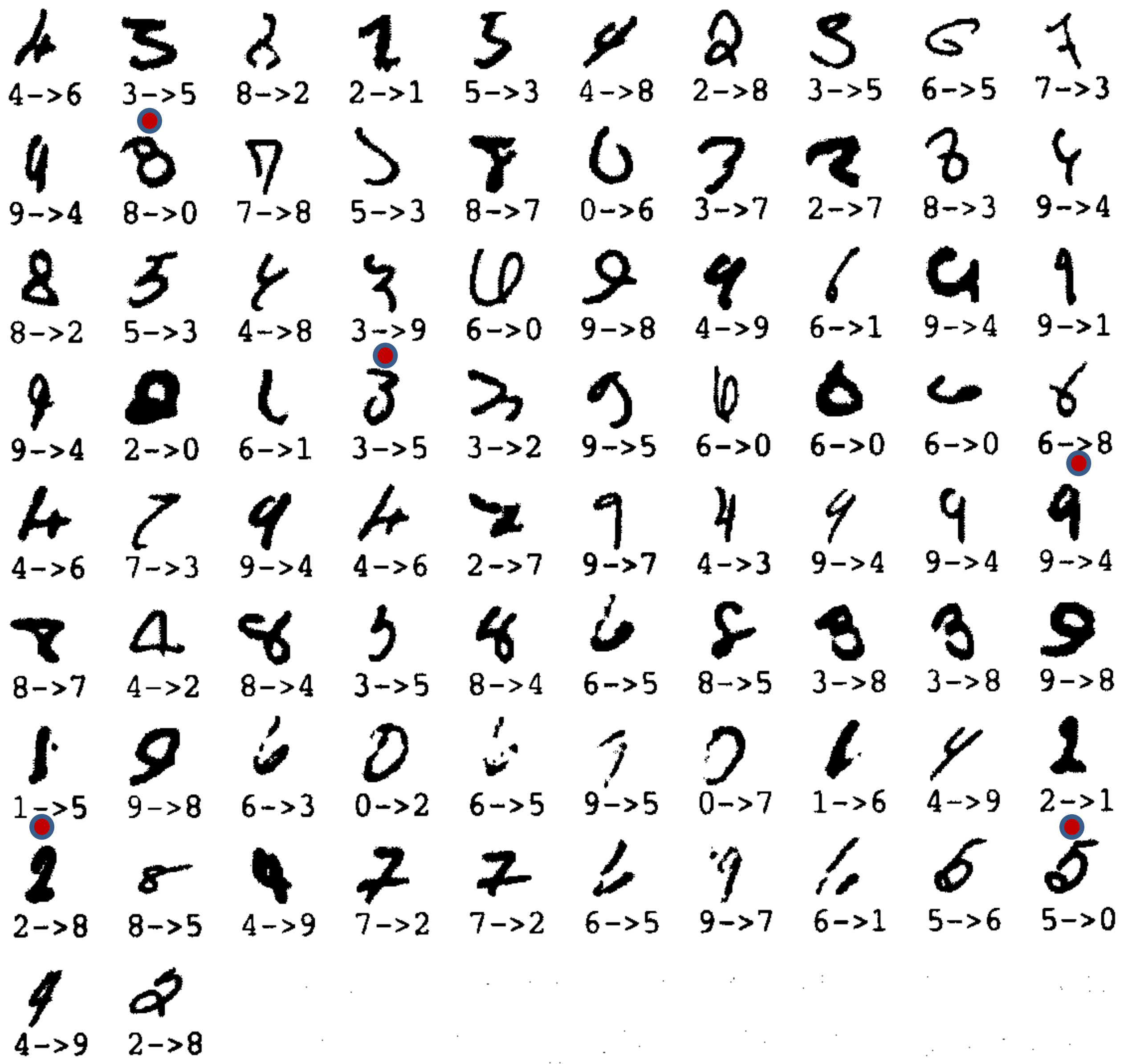
Machine Learning: An Overview

Example 1: We want to teach computers to recognize digits, but… it is very hard to say what makes a “2”.



It is very hard to say what makes a “2” and in general hard to teach computers by writing rules, without letting them learn from data. (In history, many efforts have been exerted to teach computers rules.)

Teach Computers to Learn Digits: More Examples



- Boundaries between classes can be complicated (see the left top example marked with a red dot.)
- Also some hard decisions need to "combine" complicated, multiple "features" (see the case at bottom left marked with a red dot).

Machine Learning

- It is very hard to write programs that solve many interesting problems:
  - We don’t know what program to write because we don’t know how our brain does it
  - Even if we had a good idea about how to do it, the program might be extremely complicated.
- Instead of writing a program by hand, we can let **machines to learn** (from data).
  - We collect lots of examples that specify the correct output for a given input.
  - A machine learning algorithm then takes these examples and produces a program that does the job.
  - If we do it right, the program works for new cases as well as the ones we trained it on.
  - If the data changes the program can change too by training on the new data.

Types of Learning Tasks

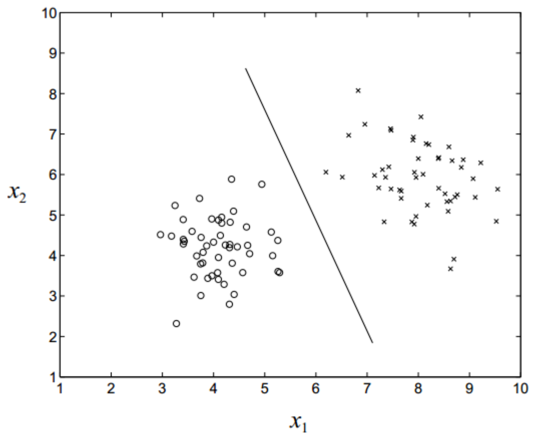
Types of Learning Tasks

- **Supervised Learning**: given examples of inputs and corresponding desired outputs, predict outputs on future inputs.
  - Examples: classification, regression, time series prediction
- **Unsupervised Learning**: given only inputs, automatically discover representations, features, structures, etc.
  - Example: clustering
  - Low-dimensional manifold learning
- **Reinforcement Learning**: given sequences of inputs, actions from a fixed set, and scalar rewards/punishments, learn to select action sequences in a way that maximizes expected reward.

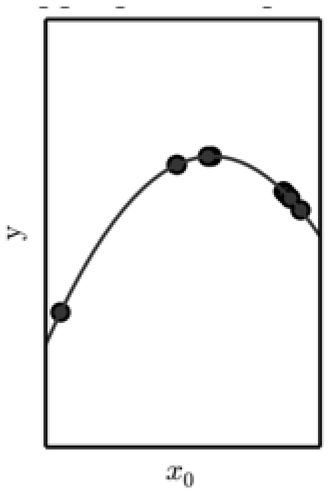
  (This course will focus on supervised and unsupervised learning.)

Supervised Learning

- **Classification**: Outputs are categorical; inputs are anything. Goal is to select correct class for new inputs.
  - The simplest case is a choice between 1 and 0.
  - We can also have multiple alternative labels.



- **Regression**: Outputs are continuous; inputs are anything (but usually continuous values). Goal is to predict outputs accurately for new inputs
  - The price of a stock in 6 months time.
  - The temperature at noon tomorrow.
  - Sentiment intensity towards a movie/restaurant/political party/stock.



Unsupervised Learning

- For several decades years, unsupervised learning was largely ignored by the machine learning community.
  - Some widely used definitions of machine learning actually excluded it.
  - Many researchers thought that clustering was the only form of unsupervised learning.

- It is a very interesting direction to work on.
  - For many tasks we may never have (or never have enough) training data for supervised learning.

Representing the World

Representing the World

- Now your data are in a "feature space", and traditionally, there are a lot of "feature engineering" to do to find good features.
- We will talk later in the deep learning part that good features can be automatically learned (representation learning).

Features **Class labels**

	categorical	categorical	continuous	categorical
Tid	**Refund**	**Marital Status**	**Taxable Income**	**Cheat?**
1	Yes	Single	125K	**No**
2	No	Married	100K	**No**
3	No	Single	70K	**No**
4	Yes	Married	120K	**No**
5	No	Divorced	95K	**Yes**
6	No	Married	60K	**No**
7	Yes	Divorced	220K	**No**
8	No	Single	85K	**Yes**
9	No	Married	75K	**No**
10	No	Single	90K	**Yes**

a "**data point**" ⇨

Each data point in this example lives in a **three-dimensional** space

Typical Machine Learning Models

Several Typical Machine Learning Models

- **Supervised Learning**:
  - Classification
    - Generative: Use Bayes' theorem to find the posterior class probabilities $p(C_k|x)$:

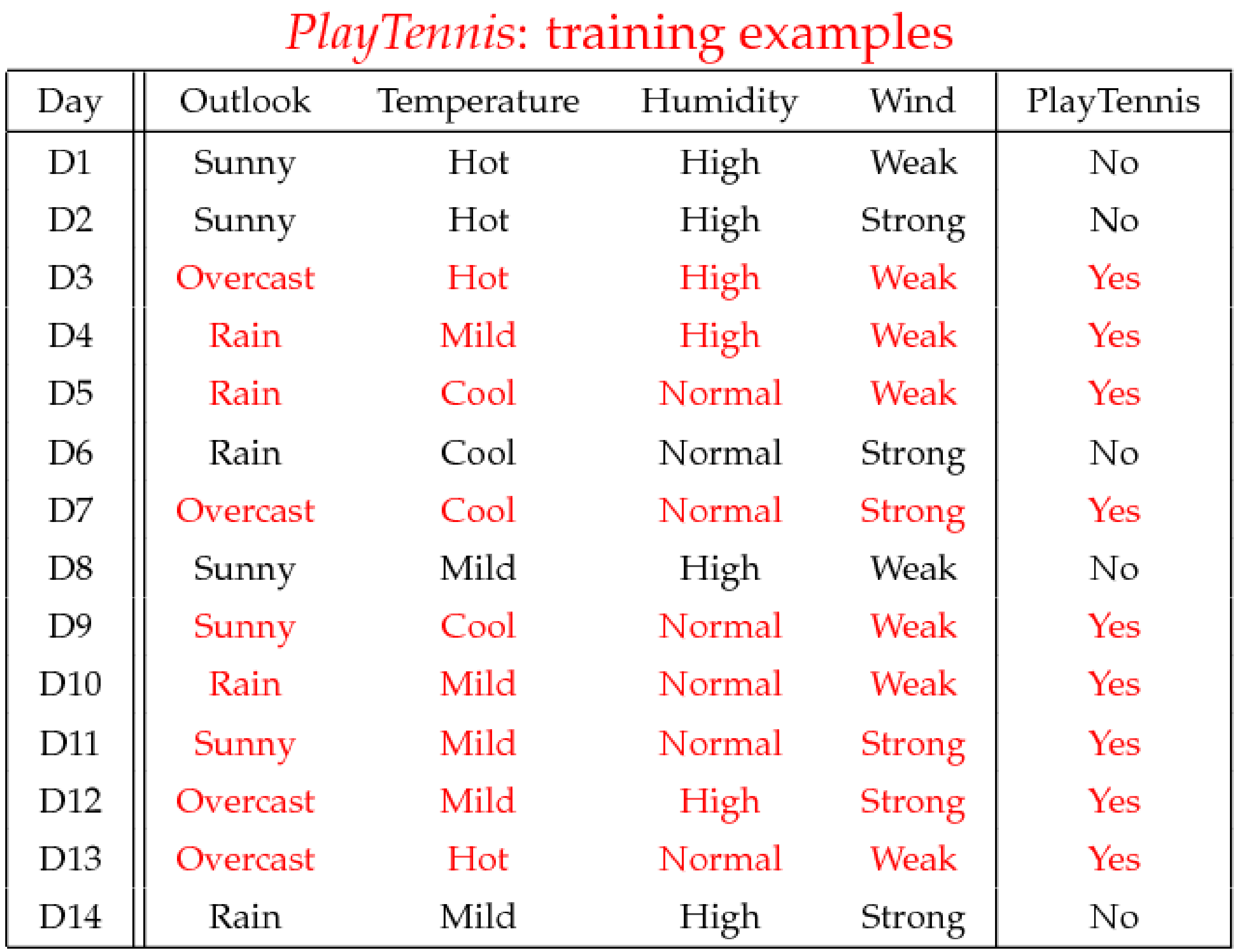
$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathbf{x})}$$

      – Example: **naïve Bayes**, (conditional) Gaussian classifiers, KNN
    - Discriminative models: learn $f(C_k|x)$ directly.
      – Example: **perceptron**, logistic regress, **SVM**.
  - Regression: constant, linear, high-order polynomial models
  - Sequence/structured prediction: HMM (also used in unsupervised setup)
- **Unsupervised Learning**:
  - Clustering: **K-means**, mixture of Gaussians
  - Dimensionality reduction: PCA

Naïve Bayes

Naive Bayes Example

Play Tennis or not?



PlayTennis: training examples

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Naive Bayes Example – Test Phase

– Given a new instance, predict its label

x’=(Outlook=*Sunny,* Temperature=*Cool,* Humidity=*High,* Wind=*Strong*)

Naive Bayes Example – Test Phase

– Given a new instance, predict its label

x'=(Outlook=*Sunny,* Temperature=*Cool,* Humidity=*High,* Wind=*Strong*)

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathbf{x})}$$

To make the predication, we need to know: $p(C_k)$ and $p(\mathbf{x}|C_k)$

Naive Bayes Example – Test Phase

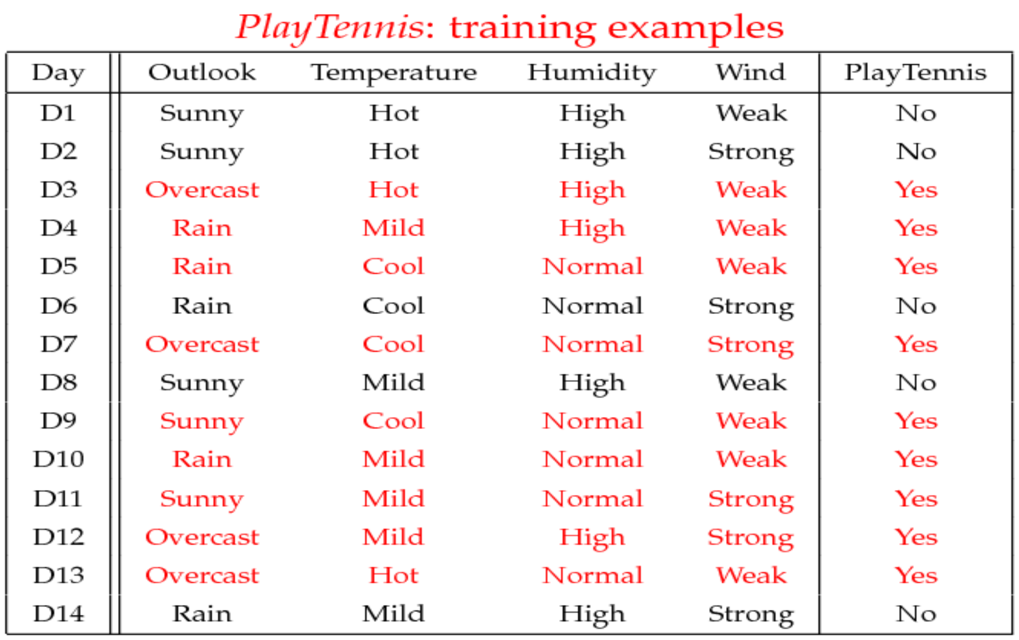
– Given a new instance, predict its label

x'=(Outlook=*Sunny,* Temperature=*Cool,* Humidity=*High,* Wind=*Strong*)

$$p(C_k|\mathbf{x}) = \frac{p(\mathbf{x}|C_k)p(C_k)}{p(\mathbf{x})}$$

p(C_k) is easy to know from the training data:

***P*(Play=*Yes*)** = 9/14 ***P*(Play=*No*)** = 5/14



PlayTennis: training examples

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Naive Bayes Example – Test Phase

- Given a new instance, predict its label

  x'=(Outlook=*Sunny,* Temperature=*Cool,* Humidity=*High,* Wind=*Strong*)

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathbf{x})}$$

How about **$p(x|C_k)$** ?

Naive Bayes Example – Training Phase

Finding $p(\mathbf{x}|C_k)$

- Assumption: conditioned on class, features/variables are independent.

$$p(\mathbf{x}|C_k) = \boxed{p(x_1, \ldots, x_{\mathrm{D}}|C_k) \underset{\Uparrow}{=} p(x_1|C_k) \ldots p(x_D|C_k)} = \prod_{i=1}^{D} p(x_i|C_k)$$

Naive Bayes Assumption

Sounds crazy right? Right! But it often works reasonably well.

Naive Bayes – Training

So, to compute $p(\mathbf{x}|C_k)$, we need to know $p(x_i|C_k)$

Given Play=*Yes*

Outlook	
Sunny	2/9
Overcast	4/9
Rain	3/9

Temperature	
Hot	2/9
Mild	4/9
Cool	3/9

Wind	
Strong	3/9
Weak	6/9

Humidity	
High	3/9
Normal	6/9

$\Uparrow$ P(Wind=*Weak*|Play=*Yes*)

Given Play=*No*

Outlook	
Sunny	3/5
Overcast	0/5
Rain	2/5

Temperature	
Hot	2/5
Mild	2/5
Cool	1/5

Wind	
Strong	4/5
Weak	1/5

Humidity	
High	3/5
Normal	2/5

Naive Bayes Example – Test Phase

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathbf{x})}$$

– Given a new instance, predict its label

x'=(Outlook=*Sunny,* Temperature=*Cool,* Humidity=*High,* Wind=*Strong*)

– Look up the tables obtained in the learning phase

P(Outlook=*Sunny*|Play=*Yes*) = 2/9	P(Outlook=*Sunny*|Play=*No*) = 3/5
P(Temperature=*Cool*|Play=*Yes*) = 3/9	P(Temperature=*Cool*|Play==*No*) = 1/5
P(Huminity=*High*|Play=*Yes*) = 3/9	P(Huminity=*High*|Play=*No*) = 4/5
P(Wind=*Strong*|Play=*Yes*) = 3/9	P(Wind=*Strong*|Play=*No*) = 3/5
P(Play=*Yes*) = 9/14	P(Play=*No*) = 5/14

– Make decision by taking the largest posterior

P(*Yes*|**x**') = [P(*Sunny*|Yes)P(*Cool*|*Yes*)P(*High*|Yes)P(*Strong*|*Yes*)]P(Play=*Yes*) = 0.0053

P(*No*|**x**') = [P(*Sunny*|No) P(*Cool*|No)P(*High*|*No*)P(*Strong*|*No*)]P(Play=*No*) = 0.0206

Given the fact P(*Yes*|**x**') < P(*No*|**x**'), we label **x**' to be "*No*".

Several Typical Machine Learning Models

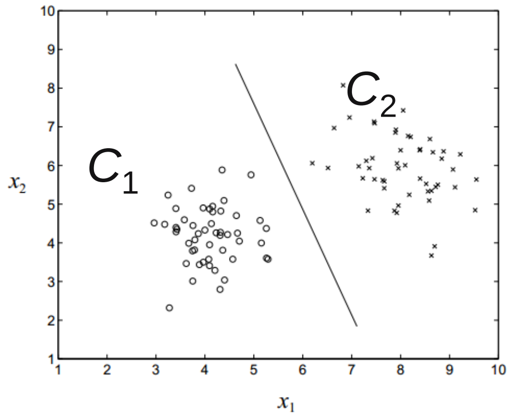
- **Supervised Learning**:
  - Classification
    - Generative: Use Bayes' theorem to find the posterior class probabilities $p(C_k|x)$:

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathbf{x})}$$

      – Example: naïve Bayes, (conditional) Gaussian classifiers, KNN
    - **Discriminative models:** learn $f(C_k\,|x)$ directly.
      – Example: **perceptron**, logistic regress, SVM.
  - Regression: constant, linear, high-order polynomial models
  - Sequence/structured prediction: HMM (also used in unsupervised setup)
- **Unsupervised Learning**:
  - Clustering: K-means, mixture of Gaussians
  - Dimensionality reduction: PCA

Linear Classifiers

- Goal: find the line (or hyperplane) which can “best” (under some criterion/objective) separate two classes:



$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0$$

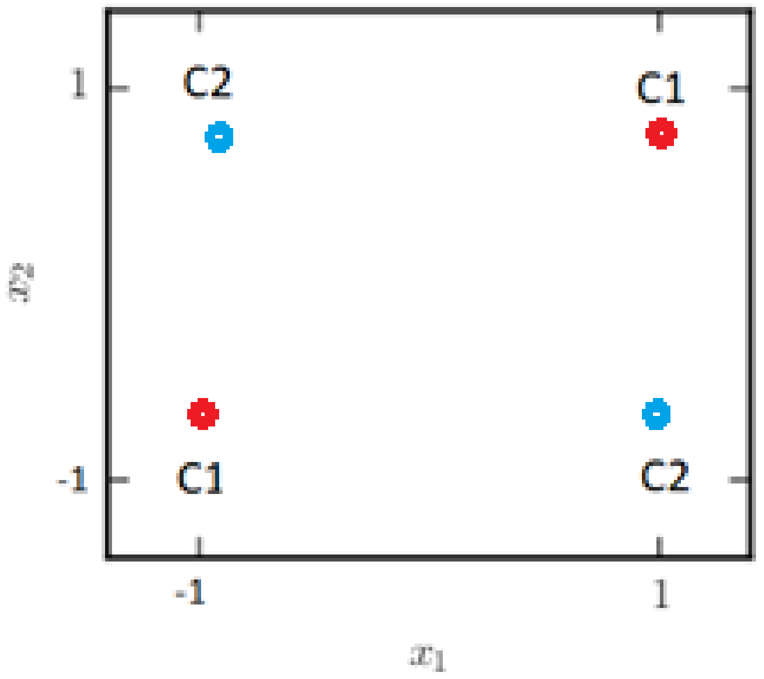
For example, we can assign an input vector **x** class C_1 if $y(\mathbf{x}) > 0$ and to class C_2 otherwise.

- The vector **w** controls the decision boundary (it is a vector perpendicular to the decision boundary); ω_0 is called *bias*.
- There are many different criteria/objects to place the hyperplane.
- Discuss one model: **Perceptron**

We can see perceptron …

- Aims to place a decision boundary to minimize the amount of error made by the model.
- For datapoints that are correctly classified, the error is zero. (Perceptron does not try to change the decision boundary).
- For misclassified data points, Perceptron tries to minimize error by adjusting the decision boundary.
  - We will not discuss the specific error equation here. But if you are interested in, you can check the Bishop textbook for details

Data that is Not Linearly Separable



Many datasets can not be separated by a linear classifier. The above is a typical example.

Several Typical Machine Learning Models

- **Supervised Learning**:
  - Classification
    - Generative: Use Bayes' theorem to find the posterior class probabilities $p(C_k|x)$:

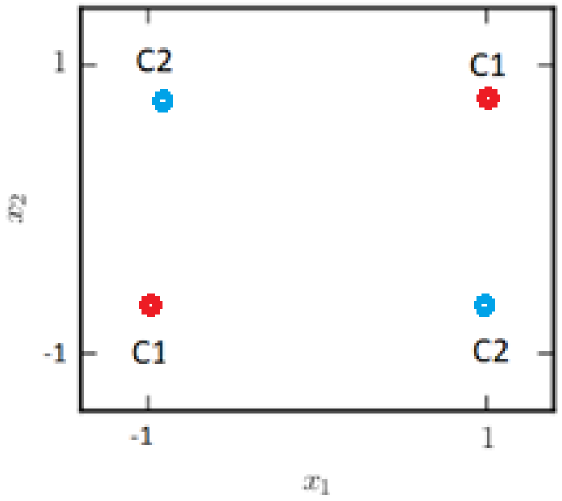
      $$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathbf{x})}$$

      – Example: naïve Bayes, (conditional) Gaussian classifiers, KNN
    - **Discriminative models:** learn $f(C_k\,|x)$ directly.
      – Example: perceptron, logistic regress, **SVM**.

    * The above models can also be viewed from the parametric/non-parametric perspective.
  - Regression: constant, linear, high-order polynomial models
  - Sequence/structured prediction: HMM (also used in unsupervised setup)
- **Unsupervised Learning**:
  - Clustering: K-means, mixture of Gaussians
  - Dimensionality reduction: PCA

Algorithms Based on Vector Data

- Remember our normal approach to classification is:
  - Represent the input (a datapoint) of a problem as a vector **x**
- We can often make our algorithms much more powerful by representing the data into a richer (larger) space which includes some fixed, possibly nonlinear functions of the original inputs/measurements.
- For example, if we have an input vector x = (x_1, x_2, x_3), we can enrich it as ϕ(x) = (1, x_1, x_2, x_3, x_1^2, x_2^2, x_3^2, x_1x_2, x_2x_3, x_1x_3).



- Remember we have discussed earlier that the mapping can make non linearly separable data to be linearly separable in a higher dimensional space.

Several Typical Machine Learning Models

- **Supervised Learning**:
  - Classification
    - Generative: Use Bayes' theorem to find the posterior class probabilities $p(C_k|x)$:

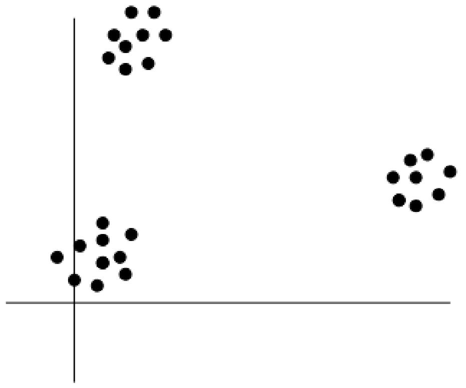
$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathbf{x})}$$

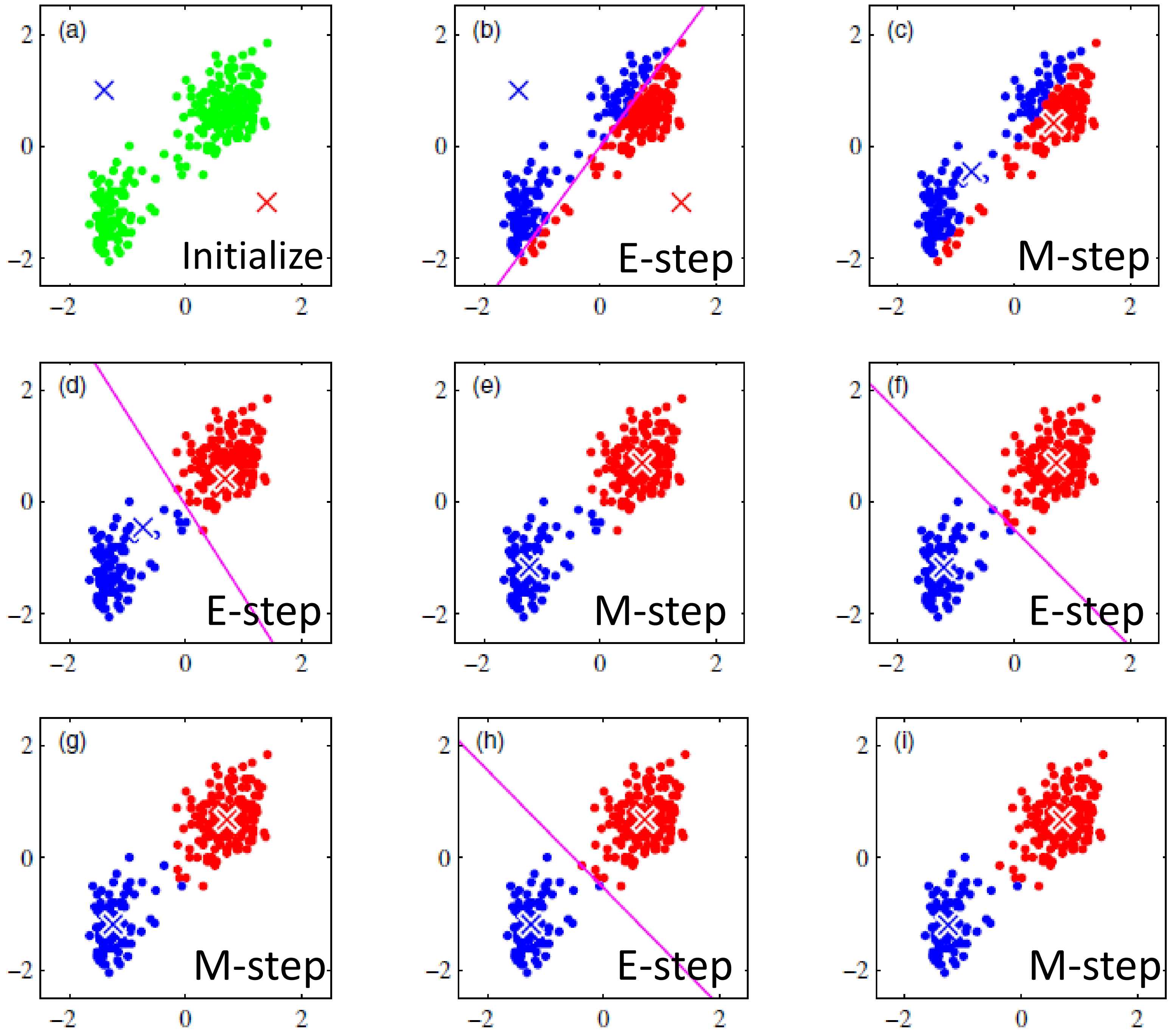
      - Example: naïve Bayes, (conditional) Gaussian classifiers, KNN
    - Discriminative models: learn *$f(C_k$ |x)* directly.
      - Example: perceptron, logistic regress, SVM.

    * The above models can also be viewed from the parametric/non-parametric perspective.
  - Regression: constant, linear, high-order polynomial models
  - Sequence/structured prediction: HMM (also used in unsupervised setup)
- **Unsupervised Learning**:
  - Clustering: **K-means**, mixture of Gaussians
  - Dimensionality reduction: PCA

Clustering

- A typical unsupervised learning problem is clustering.
  - Clustering: grouping similar training cases together and identifying a "prototype" example to represent each group (e.g., the mean of a cluster).
- Several approaches: fixed number of clusters, agglomerative, divisive, ...
- All require a way to measure distance between two data points, e.g. the Euclidean distance.



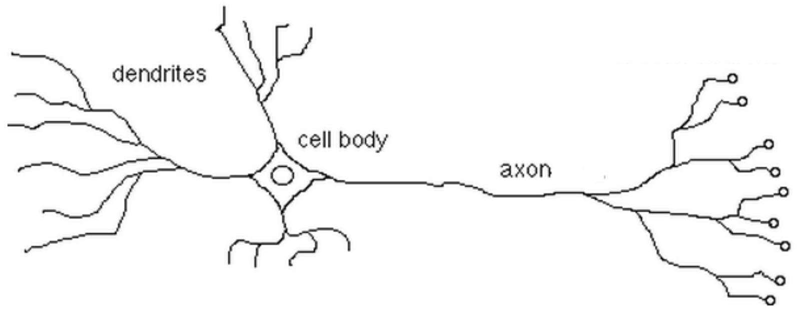

(a)
Initialize
(b)
E-step
(c)
M-step
(d)
E-step
(e)
M-step
(f)
E-step
(g)
M-step
(h)
E-step
(i)
M-step
2
0
−2

Deep Learning

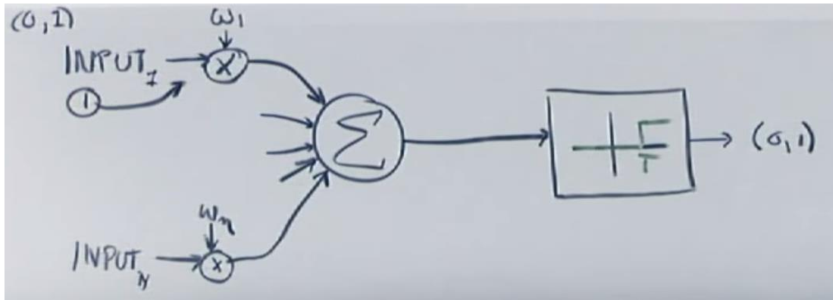
Last lecture, we have introduced some basic background of deep learning.

Today, let’s continue to introduce basic components and algorithms in deep learning that lay the foundation for the coming content we will discuss in this course …

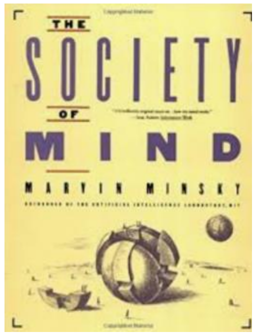
Biological Neuron



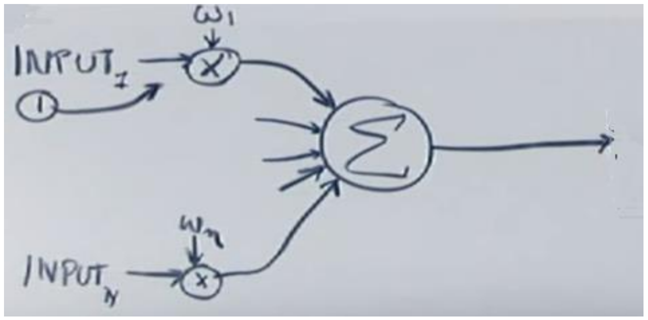
Artificial Neuron



A network of simple, non-intelligent decisions can lead to intelligence.
(An interesting book: *The Society of Mind*, by Marvin Minsky)



Linear Neurons



bias

ith input

$$y = \mathbf{w}^{\mathrm{T}}\mathbf{x} = b + \sum_i x_i w_i$$

output

index over input connections

weight on ith input

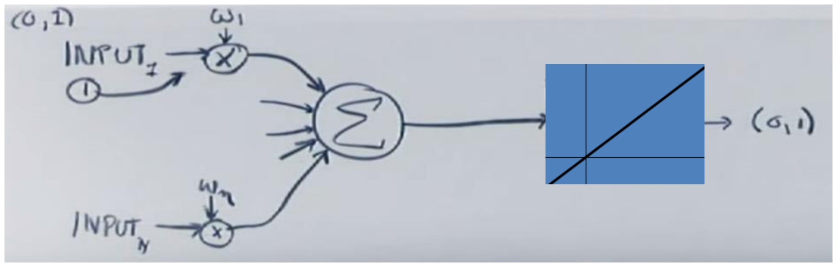
- A linear neuro just computes a weighted sum of input features, which we have actually seen in a couple of times in this class.
- These are simple but computationally limited.

Q: What is multilayer of linear neurons?

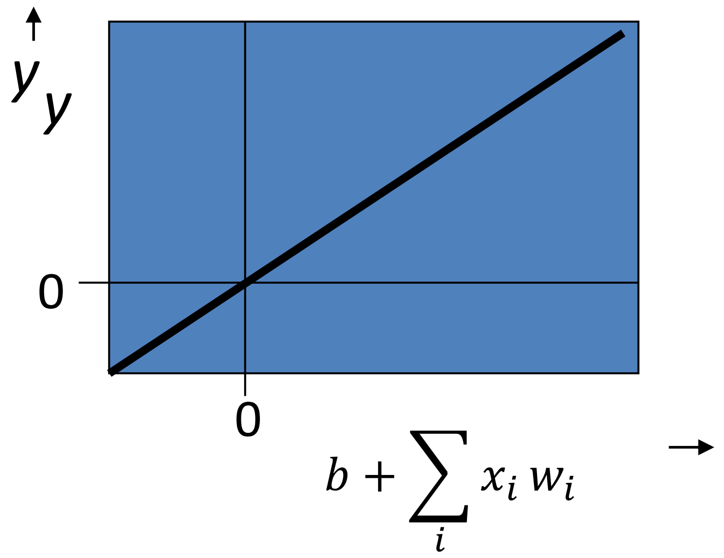
A: Still a linear model.

Linear Neurons

- Artificial neurons often have an activation layer applied over the weighted sum.
- We can think a linear neuron has a linear activation layer



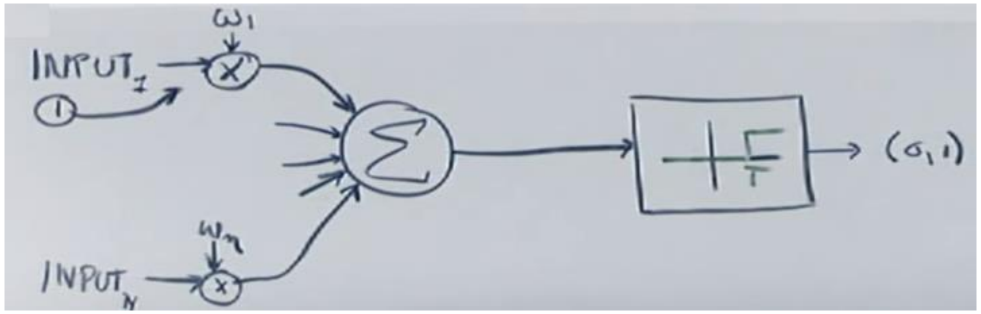
$$y = b + \sum_i x_i w_i$$



(weighted sum of input)

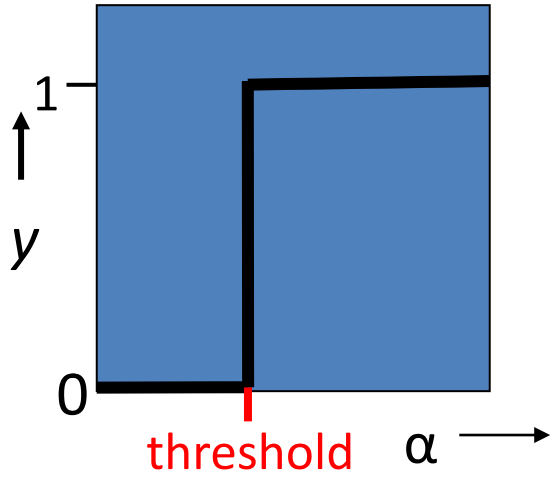
Binary Threshold Neurons

- Use a threshold to determine if the output is 1 or 0.



$$α = b + \sum_i x_i \, w_i$$

$$y = \begin{cases} 1 \text{ if } α > \text{threshold} \\ 0 \text{ otherwise} \end{cases}$$

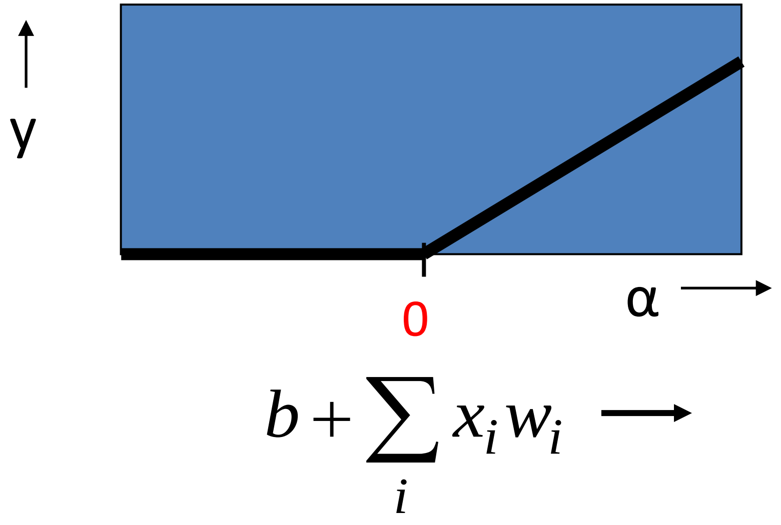


Rectified Linear Neurons
(sometimes called linear threshold neurons)

They compute a weighted sum of their inputs.
The output is a result of a non-linear function as follows:

$$\alpha = b + \sum_i x_i w_i$$

$$y = \begin{cases} \alpha \text{ if } \alpha >= 0 \\ 0 \text{ otherwise} \end{cases}$$



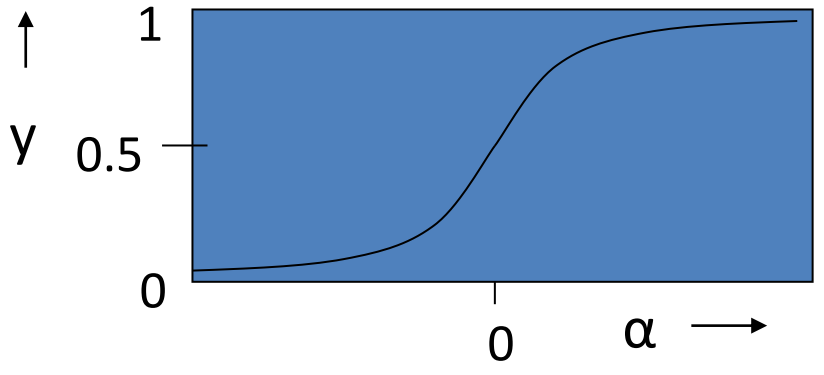
i.e., y = max(0, α)

Sigmoid Neurons

- These give a real-valued output that is a smooth and bounded function of their total input.
  - Typically they use the logistic function
  - They have nice derivatives which make learning easy.

$$\alpha = b + \sum_i x_i w_i$$

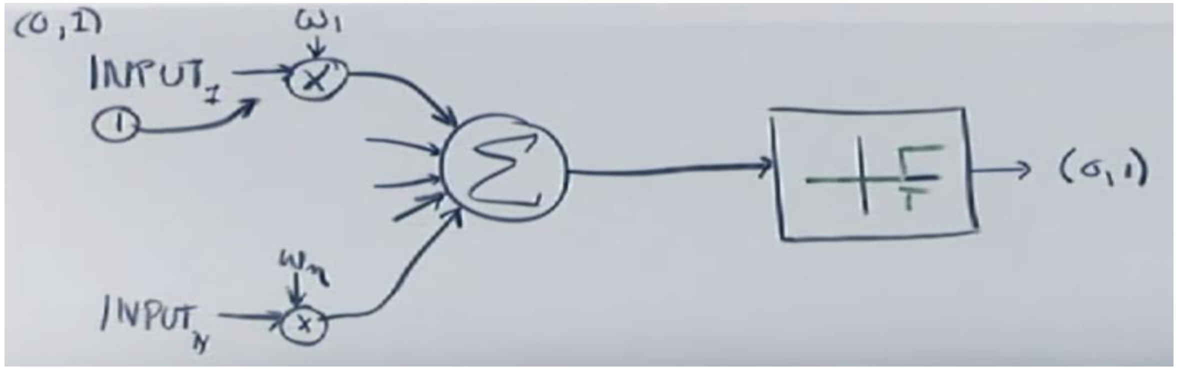
$$y = \frac{1}{1 + e^{-a}}$$



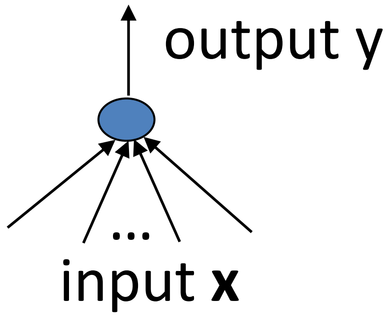
Hidden units

- In general you can use a validation set to select activation function for your task.
- We often uses ReLUs
- Many activation functions perform comparably to ReLUs. New hidden units that perform comparably are rarely interesting.

So far, we have discussed different types of individual neurons

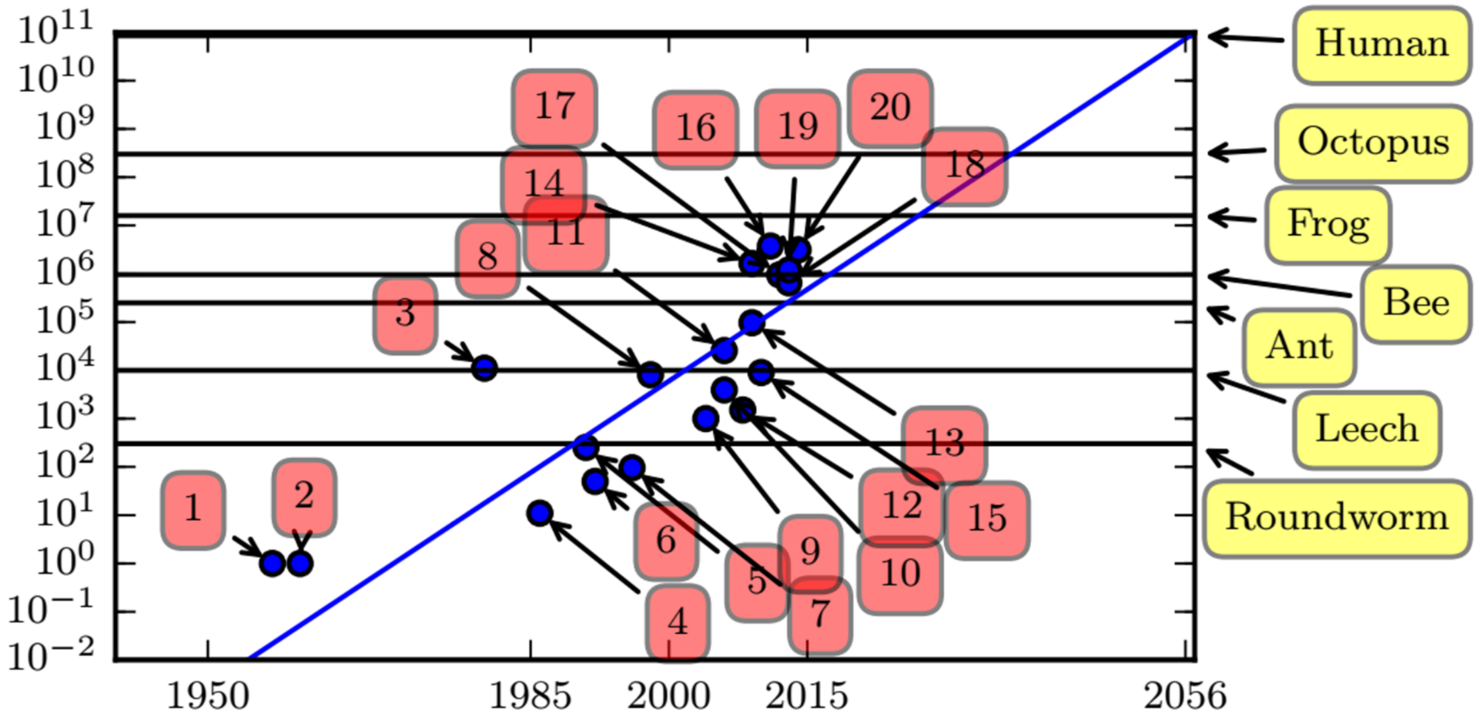


For simplicity, a neuron is often drawn as follows, where the circle includes both the weighted sum operation and the activation function:



We will use multiple neurons to construct complicated neural networks.

Number of Neurons



Increasing neural network size over time. The 20 blue circles are different neural network models; e.g., #20 is GoogLeNet proposed in 2014 for image recognition. Since the introduction of hidden units, artificial neural networks have doubled in size roughly every 2.4 years.

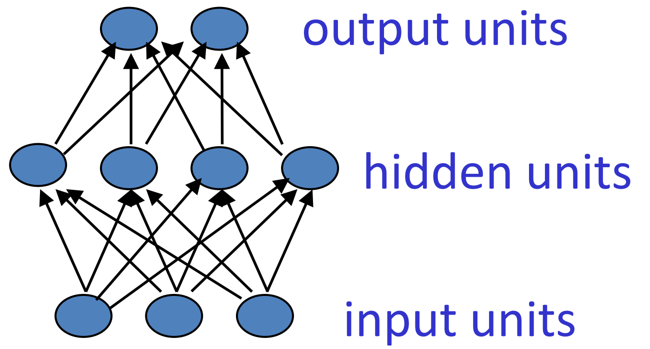
* Refer to Chapter 1 of the GoodFellow & Bengio textbook we listed in our first lecture for details. The above figure is taken from and credited to that book.

How to Structure Your Networks?

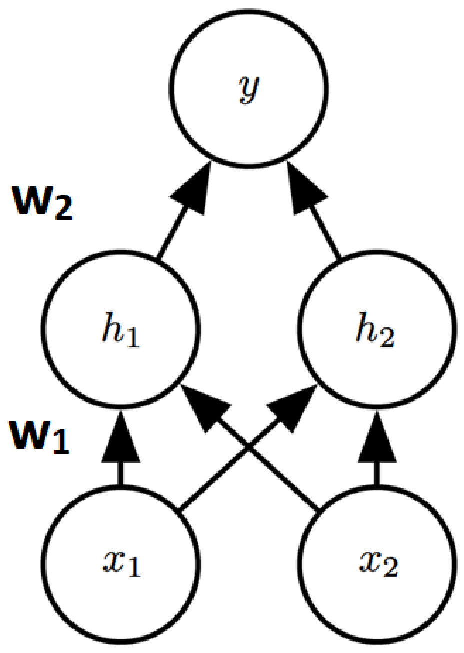
- We have shown that the number of neurons is relevant to “intelligence” of systems or living beings.
- The structures of networks also matter.

Feed-Forward (FF) Neural Networks

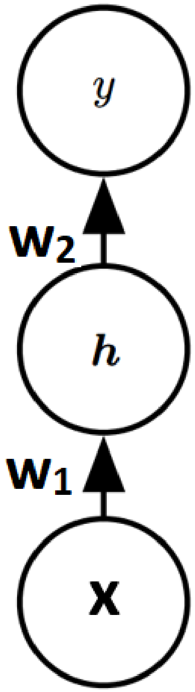
- These are the commonest type of neural network in practical applications.
    - The first layer is the input and the last layer is the output.
    - In the following example, we have one hidden layer.
    - If there is more than one hidden layer, we call them **"deep" neural networks.**
    - They compute a series of transformations.
    - The inputs of the neurons in each layer are a non-linear function of the outputs of the layer below.



Network with One Hidden Layer



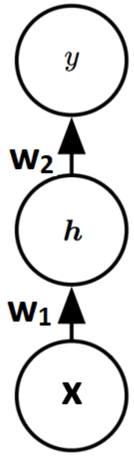
Can be drawn as:

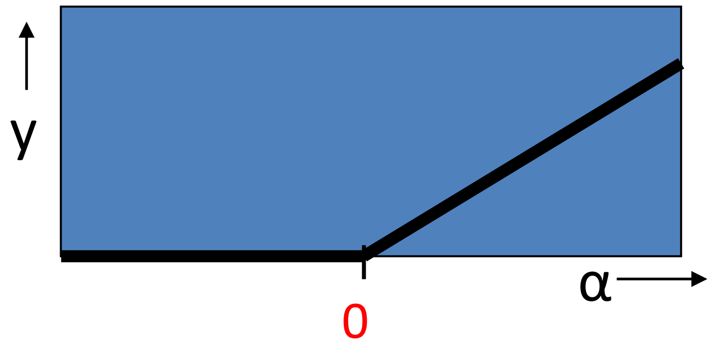


W₁ and W₂ are model parameters.

- So $\boldsymbol{h} = f_1(\mathbf{W}_1^T \mathbf{x} + c)$, where $f_1(.)$ is an activation function. Note that $\mathbf{W}_1$ is a matrix.
- $\boldsymbol{y} = f_2(\mathbf{W}_2^T \boldsymbol{h} + b)$, where $f_2(.)$ is also an activation function. Note that $\mathbf{W}_2$ is another matrix.

Network with One Hidden Layer





- More specifically, suppose: (1) $f_1(.)$ is the Rectified Linear Unit (ReLU), and (2) $f_2(.)$ is the linear activation function, we have:

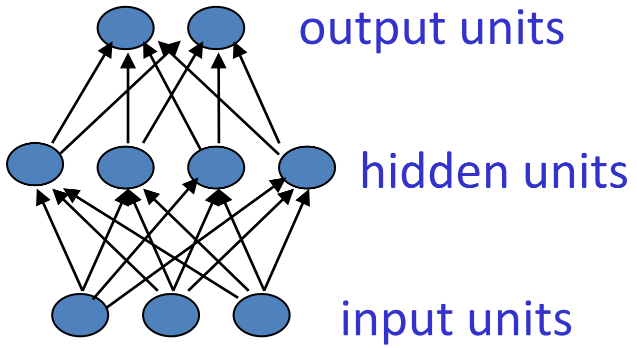
$$\boldsymbol{y} = f_2(\mathbf{W_2}^T \boldsymbol{h} + b)$$

$$= f_2(\mathbf{W_2}^T f_1(\mathbf{W_1}^T \mathbf{x} + c) + b)$$

$$= \mathbf{W_2}^T \max(0, \mathbf{W_1}^T \mathbf{x} + c) + b$$

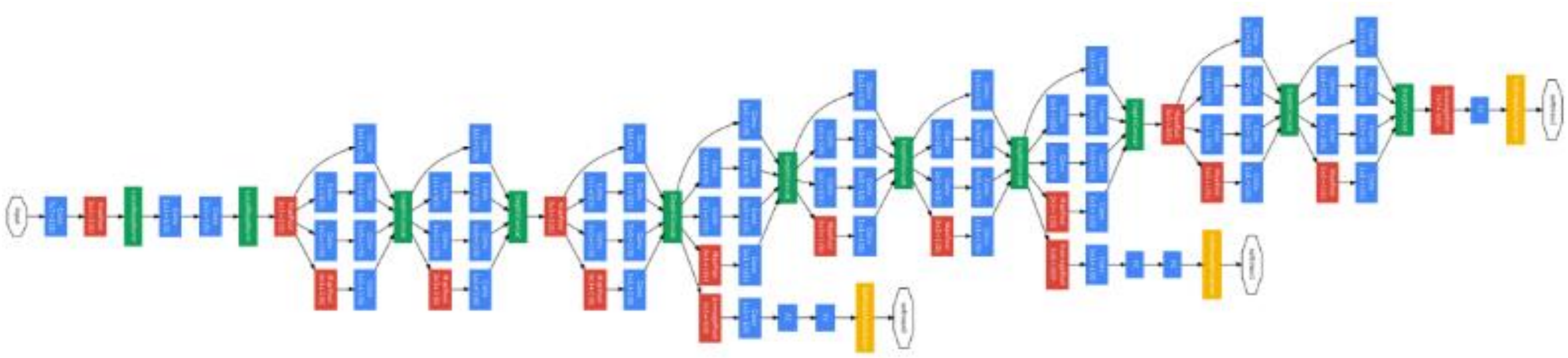
Feed-Forward Neural Networks

- Feed-Forward Networks are **fully connected** (between two adjacent layers).



Q: How many layers do you need?

GoogLeNet



For example example, GoogLeNet is a very deep neural nets proposed to perform image classification.

Universal Approximator Theorem

- However, it has been proven that networks with one hidden layer is enough to *represent* an approximation of any function to an arbitrary degree of accuracy.
  - Remember that supervised machine learning models are trying to find the function underlying training data.

- Q: So why we want a deep network?

Better Generalization with Greater Depth

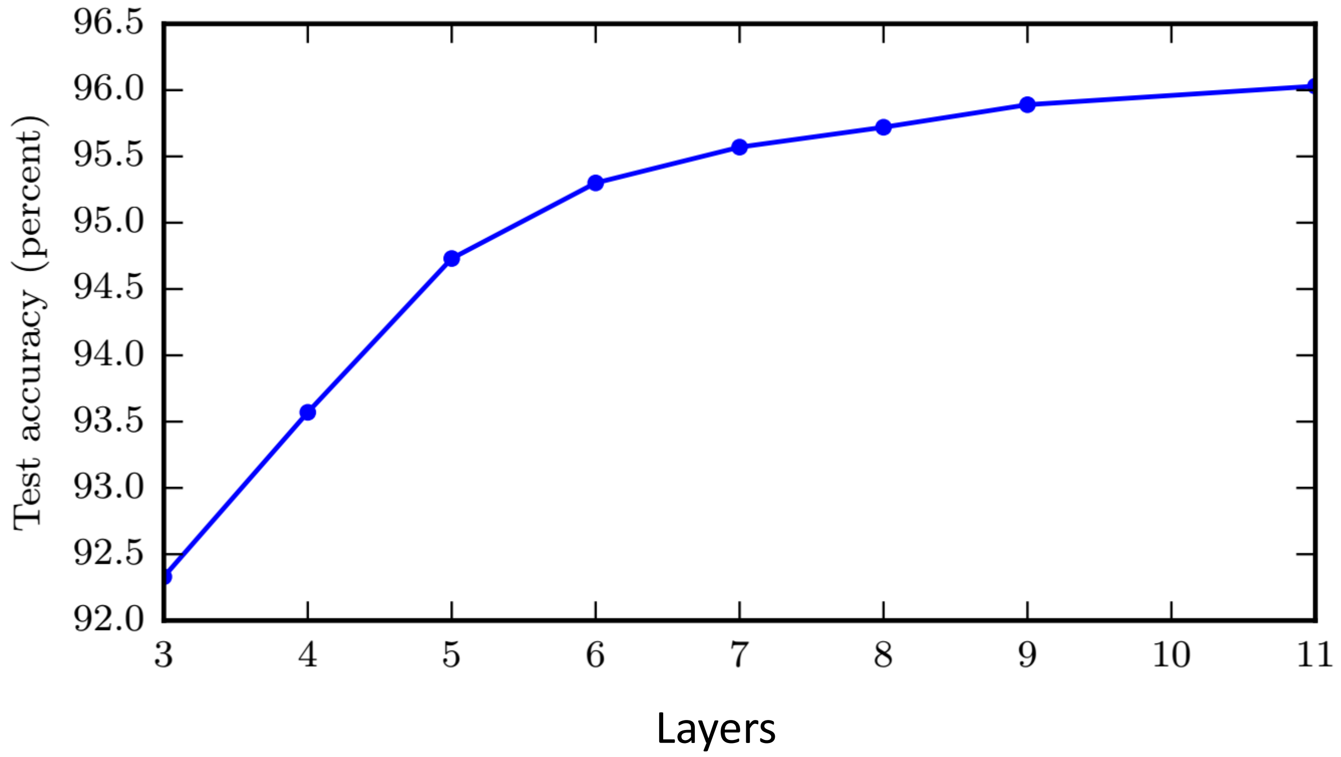


Figure 6.6 of the Goodfellow & Bengio textbook (the book is listed in the slides of our first lecture): Empirical results showing that deeper networks generalize better when used to transcribe multi-digit numbers from photographs of addresses. The test set accuracy consistently increases with increasing depth.

Large, Shallow Models Overfit More

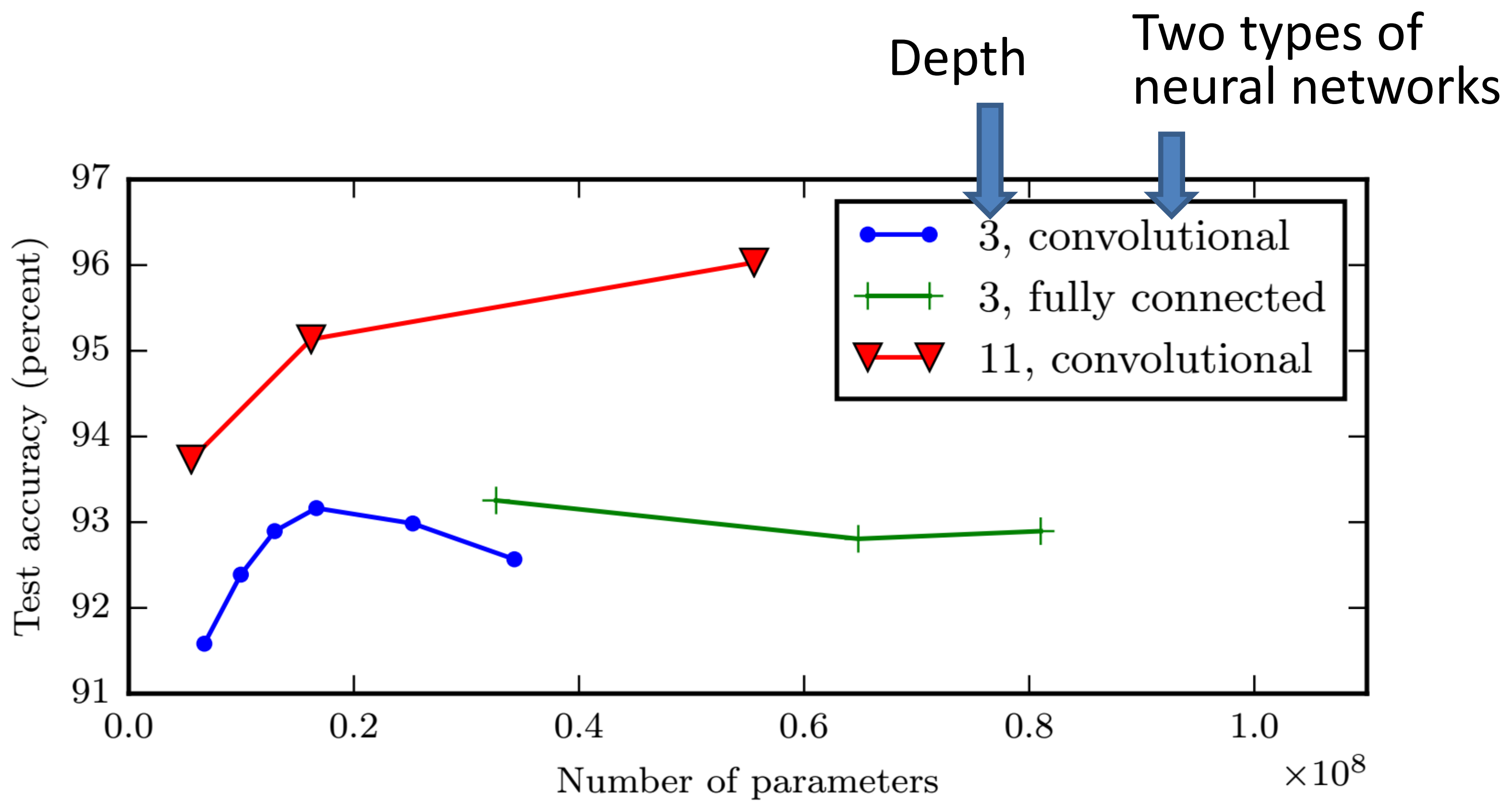


Figure 6.7 of the Goodfellow & Bengio textbook. So far, just think convolutional networks as a type of sparsely connected feed-forward network; we will discuss details later in the course. We observe in the figure that shallow models in this context (on blue line) overfit at around 20 million parameters while deep ones (red) can benefit from having over 60 million. While for the same numbers of parameters, deep models (red) achieve better performance than shallow ones (blue).

Deep Network

- Q: So why we want a deep network rather than that with only one hidden layer?

  A:

  - Shallow nets may overfit more easily.
  - Deep nets often achieve better performance with same numbers of parameters.

- Q: But how deep?

  A: No universal answer; often task dependent.

  For some problems, the depth of models are naturally given by the problems; e.g., in recurrent neural nets we will discuss, a networks often has a depth “by time” (number of time steps).

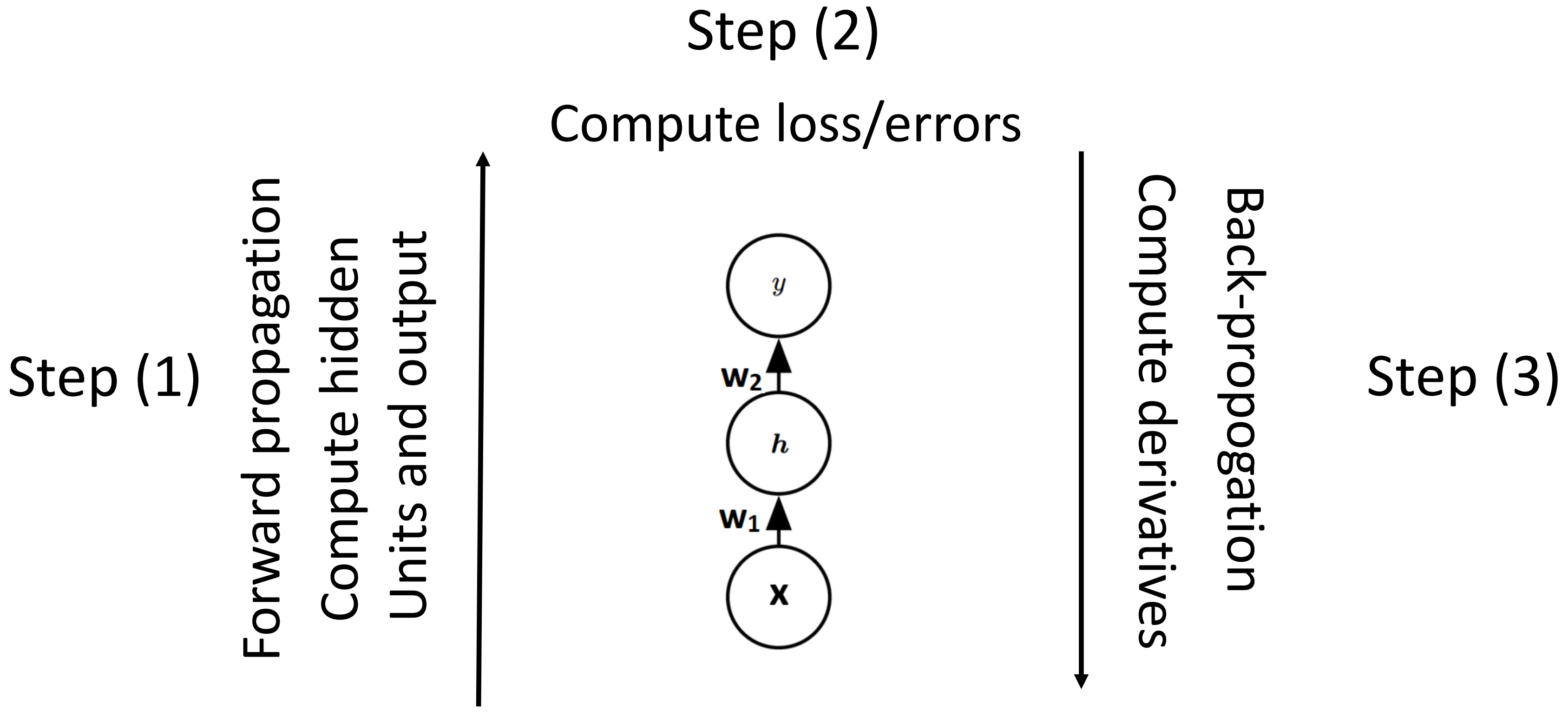
Backprobagation

Training Feed Forward Neural Nets: The Key Ideas

- We aim to **minimize** errors.
- This is an optimization problem.
  - Remember what we learn in mathematics: for some (error) functions, we can <u>compute the derivative</u> w.r.t. model parameters and set that to be zero; we get an equation.
  - Solve that equation and get model parameters corresponding to where the derivative is zero (where the error is smallest.)
- In general, for many other complicated error functions, you cannot solve the above equation and you actually do not need to solve the optimization problem yourself.
  - There are many optimization algorithms you can use to find good parameters that (approximately) minimize the errors, but you often need to provide the optimization algorighms with the derivatives.
- In feed-forward neural networks, derivatives are computed with an algorithm called:

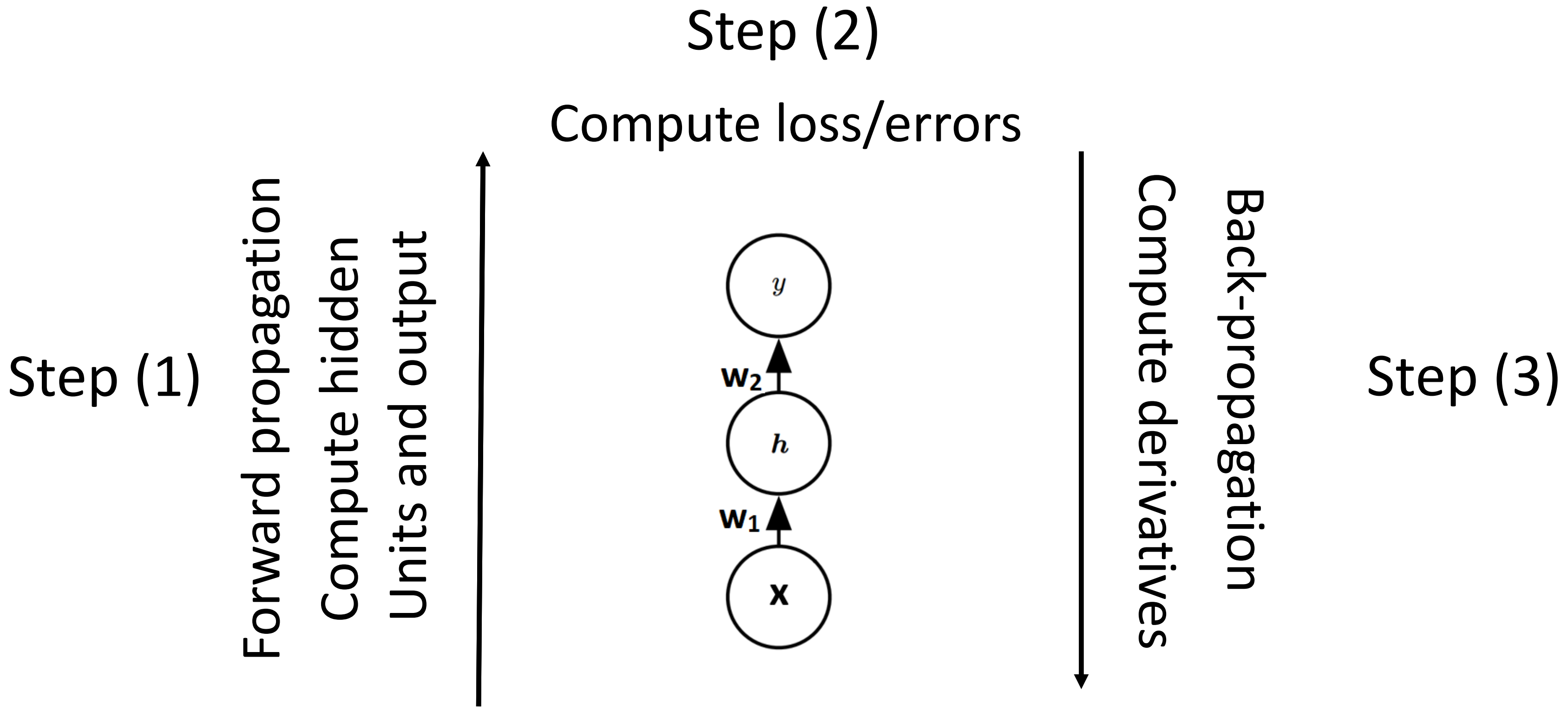
Backward Propagation (a.k.a. Backprop)

Training Feed-Forward Networks: Forward and Backward Propagation



- Step (4): Use some existing optimization algorithms to find good parameters (those minimizing errors).
- Note that when you use some neural network toolkits, you often just need to define network architecture; the tools can do forward and backward propagation and optimization for you.

Training Feed-Forward Networks: Forward and Backward Propagation

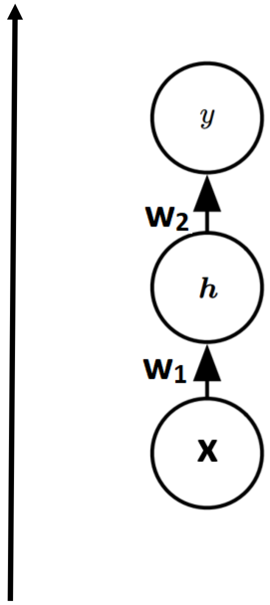


- In neural networks optimization algorithms often do not guarantee finding globally optimal solution
  - Often, the error function is not convex and have many local optimums.

Step 1: Forward Propagation

Step (1)

Forward propagation
Compute hidden
Units and output



For the left example, forward propagation computes:

- $\boldsymbol{h} = f_1(\mathbf{W_1}^T \mathbf{x} + c)$
- $\boldsymbol{y} = f_2(\mathbf{W_2}^T \boldsymbol{h} + b)$

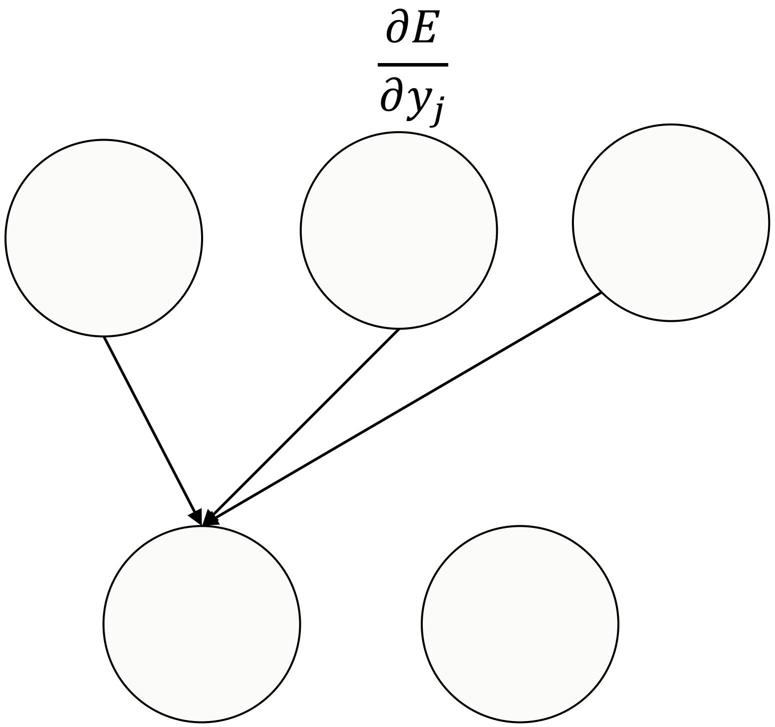
For deeper networks, just do the same computation in the same bottom-up way.

Step 2: Compute Errors and Derivatives w.r.t. Output Variables

- First compute the discrepancy between each output y_j and its target value t_j (ground truth).
  - There are many error functions to use.
  - Suppose we use squared error here.
- Compute the derivative of errors w.r.t. each output y_j.
- Then we will use back propagation to compute the derivative of errors w.r.t. parameters at lower layers of the network.

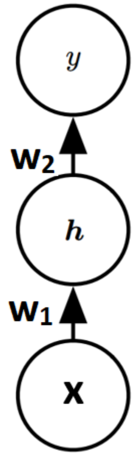
$$E = \sum_j \frac{1}{2} (y_j - t_j)^2$$

$$\frac{\partial E}{\partial y_j} = y_j - t_j$$



Backward Propagation (Backprop)

- Backpropagation applies the derivative chain rule to the network.



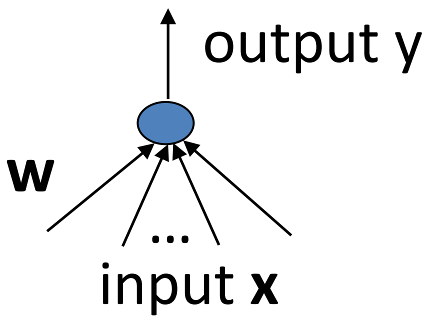
$$\frac{\partial \mathbf{E}}{\partial \mathbf{h}} = \frac{\partial \mathbf{E}}{\partial \boldsymbol{y}} \frac{\partial \boldsymbol{y}}{\partial \mathbf{h}}$$

$$\frac{\partial \mathbf{E}}{\partial \mathbf{x}} = \frac{\partial \mathbf{E}}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{x}}$$

- For deeper networks, the computation is performed in a similar top-down way.
- With backprop, error derivatives for hidden units can be computed efficiently.

Backprop Details

- Let's first see how derivative is performed on one neuron.
- Suppose we have a logistic neuron where the activation function is a logistic function.



$$y = \frac{1}{1 + e^{-a}}$$

$$\alpha = b + \sum_i x_i \, w_i$$

Derivative of Logistic Neuron

- The derivatives of the logistic activation.

$$y = \frac{1}{1+e^{-\alpha}}$$

$$\frac{dy}{d\alpha} = \frac{1}{1+e^{-\alpha}} - \frac{1}{(1+e^{-\alpha})^2}$$

$$= y - y^2$$

$$= y(1-y)$$

This form is very convenient: as *y* has been computed in forward propagation, the derivative is ready to use!

Derivative of Logistic Neural

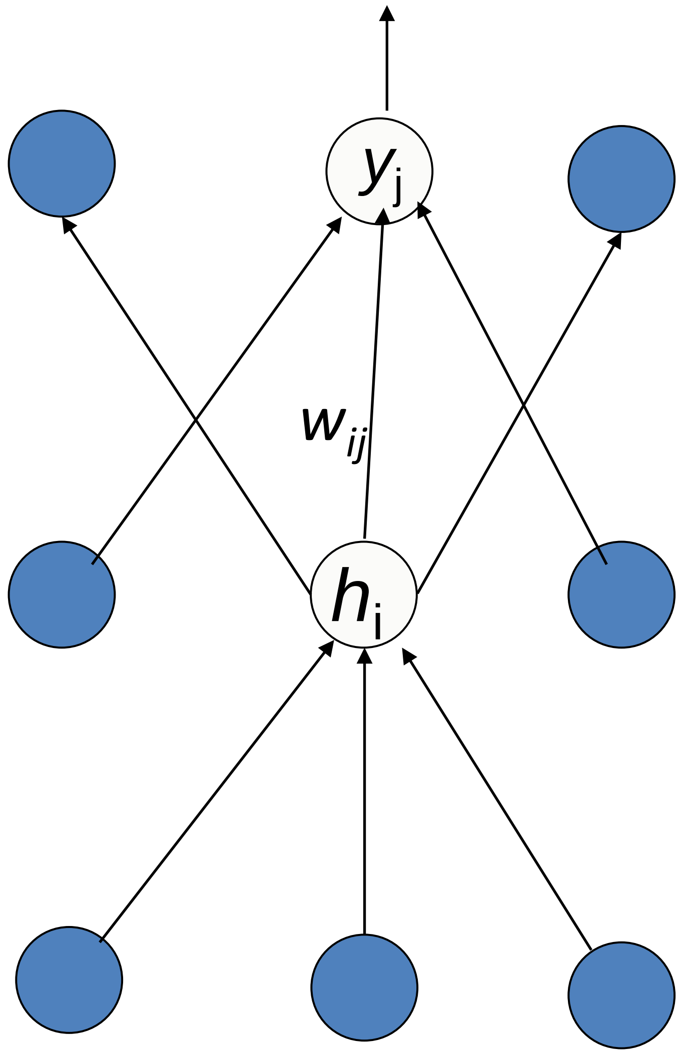
- The derivative of the weighted sum:

$$\alpha = b + \sum_i x_i w_i$$

$$\frac{\partial \alpha}{\partial w_i} = x_i \qquad \frac{\partial \alpha}{\partial x_i} = w_i$$

We can see that once we have the error derivatives for a hidden unit, it’s easy to get the error derivatives for the weights going into a hidden unit.

The Error Derivatives at the Top Two Layers



$$\frac{\partial E}{\partial \alpha_j} = \frac{\partial E}{\partial y_j}\frac{dy_j}{d\alpha_j} = \frac{\partial E}{\partial y_j} y_j(1 - y_j)$$

Again, y_j has been computed in forward propagation.

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial \alpha_j}\ \frac{\partial \alpha_j}{\partial w_{ij}}\ = \ \frac{\partial E}{\partial \alpha_j} h_i$$

Also, h_j has been computed in forward propagation.

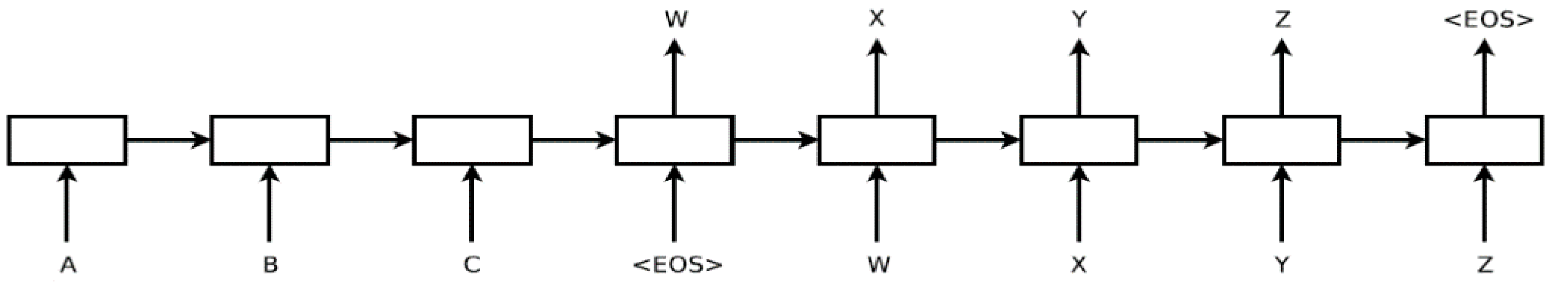
$$\frac{\partial E}{\partial h_i} = \sum_j \frac{\partial E}{\partial \alpha_j}\frac{d\alpha_j}{dh_i} = \sum_j \frac{\partial E}{\partial \alpha_j} w_{ij}$$

We keep performing this top-down on the network.

Recurrent Neural Networks (RNN)

Modelling Sequences: Motivations

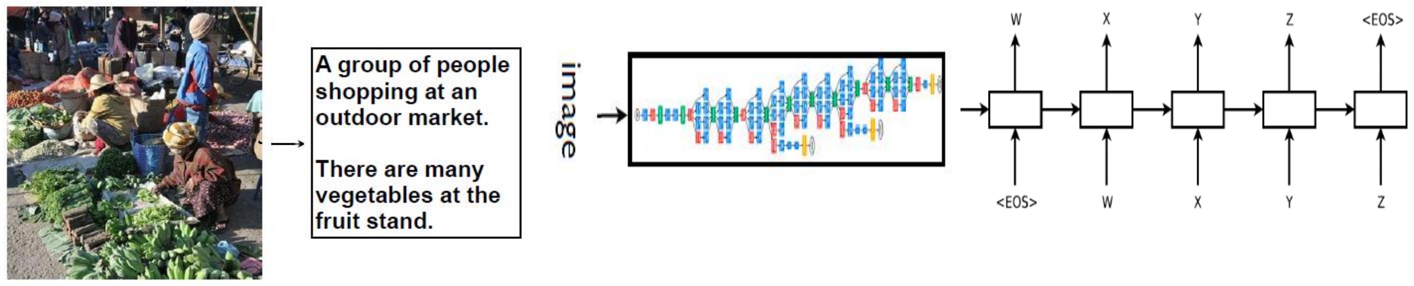
- E.g., translate a French sentence to English



A French sentence with three words A B C

An English sentence with words W X Y Z.

- Image-to-text conversion: predict next words based both on the image and previous words :


A group of people shopping at an outdoor market.
There are many vegetables at the fruit stand.
image
W
X
Y
Z
<EOS>
<EOS>
W
X
Y
Z

Modelling Sequences: Motivations

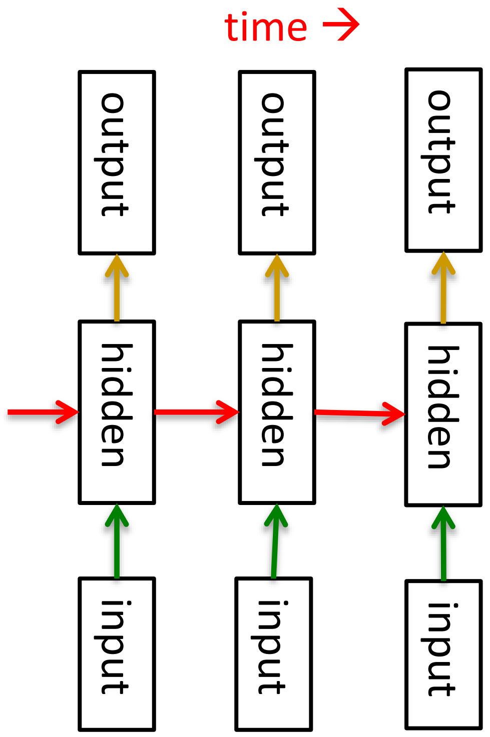
- More examples: We can teach a model by training it to predict the next term in the input sequence.
    - Recently many people call this *self-supervised* learning. It blurs the distinction between supervised and unsupervised learning.
    - It uses methods designed for supervised learning, but doesn't require a separate training signal.
    - This technology achieves dramatic improvement recently on many problems.

Example sequence:

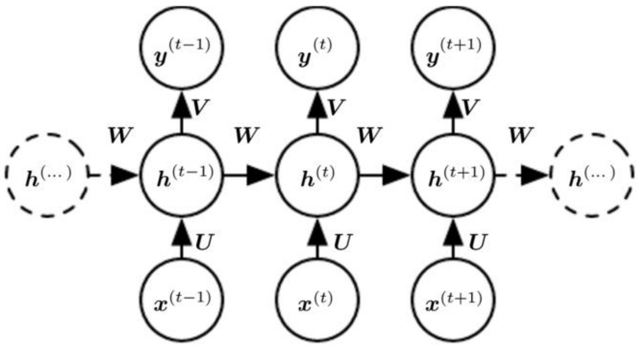
"Shared joy <u>is</u> a double joy; shared sorrow <u>is</u> half <u>...</u>"

Recurrent Neural Networks (RNNs)

- Recurrent neural networks (RNN) are a type of neural networks designed to model sequences.
- At each time step, RNN has one layer of hidden units to remember history information
- So the depth of the network along the time line corresponds to the number of timesteps (the length of input sequences).



Details at Each Time Step



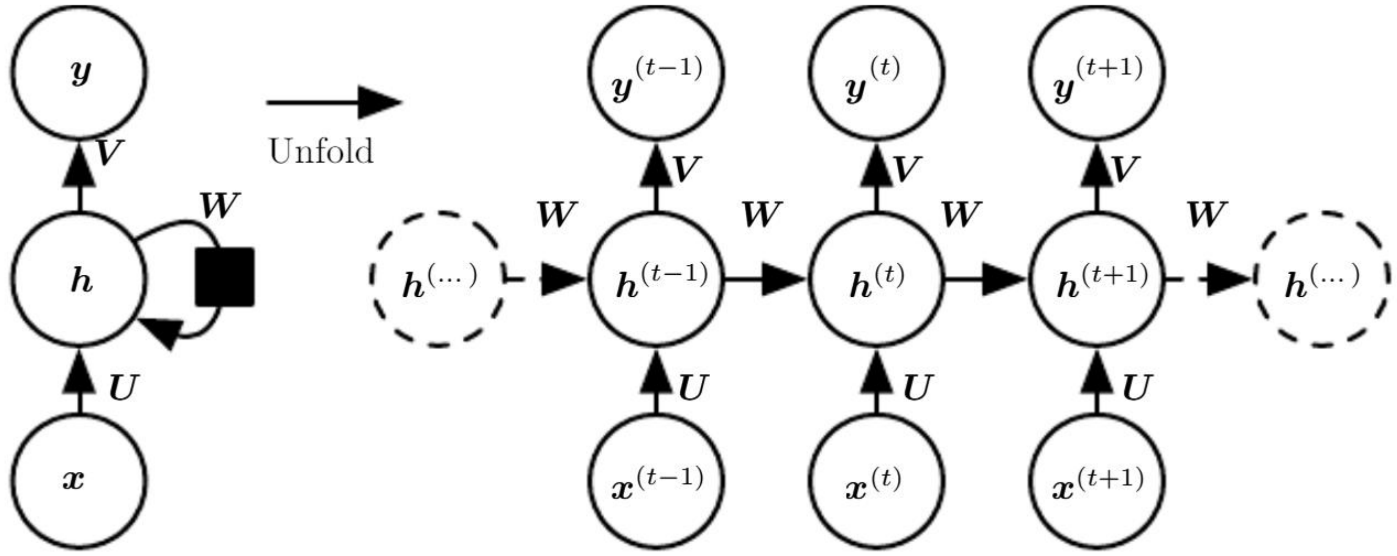
$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\boldsymbol{y}^{(t)} &= \mathrm{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$

Note that **W**, **U** and **V** are weights (model parameters). In RNN, they are shared at all different time steps. You can see that layer **h** uses a *tanh* activation here but you can use other activation functions we have learned.

$$y_i^{(t)} = softmax\left(\boldsymbol{o}^{(t)}\right)_i = \frac{e^{(o_i^{(t)})}}{\sum_{j=1}^{m} e^{(o_j^{(t)})}}$$

Where $\boldsymbol{o}^{(t)}$={$o_1^{(t)}$,…, $o_m^{(t)}$} has *m* number of units/neurons.

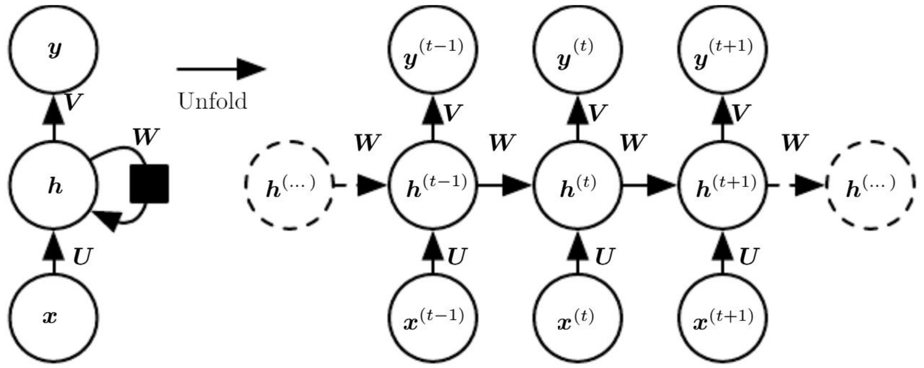
Two Ways to Present RNNs



- RNN can be drawn in the folded way (left), or unfolded way (right) where each node is associated with one particular time step.
- Both graphs describe RNN maps an input sequence of **x** to a corresponding sequence of output **y**. Both describe that **h** is updated when an input is taken in.
- At each time step, an error/loss can be used to measure how far each **y** is from the corresponding training target/true label
    - But as we discussed earlier, in some applications, **y** is just the next input and we use that as your training target.

Forward Propagation of RNNs

- From the unfolded version of RNN graph, we see that RNN can be seen as a special types of feed-forward models.
  - But remember that all time steps share the same transition and emission functions.
- Again, the forward propagation of RNNs is simple:



$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\boldsymbol{y}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$

  - You can see that layer **h** uses a *tanh* activation here but you can use other activation function as well. Note that **W**, **U** and **V** are shared at all different time steps.

Backward Propagation of RNNs

- You can do backward propagation similarly as in feed forward nets, but remember that:
  - A RNN keep reusing the same weights at different time steps.
- It is easy to modify the backprop algorithm to incorporate the constraints between the weights.
  - We compute the gradients as usual, and then modify the gradients so that they satisfy the constraints.

    To constrain weights at different times to be same:

    $$\mathbf{W}^{(t)} = \mathbf{W}^{(t+1)}, \text{we need:} \quad \Delta\mathbf{W}^{(t)} = \Delta\mathbf{W}^{(t+1)}$$

    $$\text{we compute:} \quad \frac{\partial E}{\partial \mathbf{W}^{(t)}} \quad and \quad \frac{\partial E}{\partial \mathbf{W}^{(t+1)}}$$

    $$\text{use} \quad \frac{\partial E}{\partial \mathbf{W}^{(t)}} + \frac{\partial E}{\partial \mathbf{W}^{(t+1)}} \quad \text{for } \mathbf{W}^{(t)} \text{ and } \mathbf{W}^{(t+1)}$$

  - So if the weights started off satisfying the constraints, they will continue to satisfy them.

Difficulties in Training RNNs

- In principle, RNNs were known to be more powerful than other sequence models (e.g. hidden Markov models).
- Q: But why RNNs were not widely used until rencent years, e.g., in speech recognition?

  A: It was difficult to train RNNs:

  (1) training is time consuming and less tractable when computers were slow,

  (2) The difficulty is also due to a phenomenon called vanishing/exploding gradient.