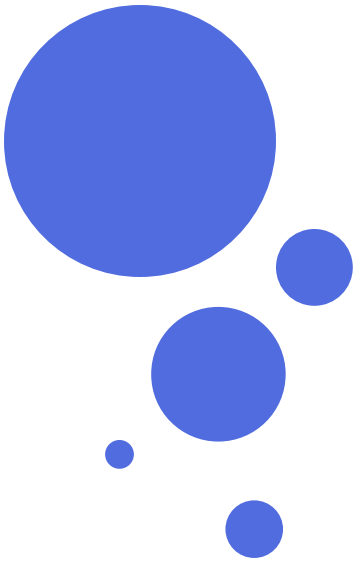


Lecture 23: Final Exam Review

Dr. Chengjiang Long
Computer Vision Researcher at Kitware Inc.
Adjunct Professor at RPI.
Email: **longc3@rpi.edu**

Final Project Presentation Agenda on May 1st

No.	Start time	Duration	Project name	Authors
1	1:30pm	00:10:00	Handwritten digits recognition	Kimberly Oakes
2	1:40pm	00:10:00	Character Recognition	Xiangyang Mou, Tong Jian
3	1:50pm	00:10:00	Handdrawn recognition	Deniz Koyuncu
4	2:00pm	00:10:00	Human Face Recognition	Chao-Ting Hsueh, Huaiyuan Chu, Yilin Zhu
5	2:10pm	00:10:00	Head Pose Estimation	Lisa Chen
6	2:20pm	00:10:00	Facial expressions expression	Cameron Mine
7	2:30pm	00:10:00	Kickstarter: succeed or fail?	Jeffrey Chen and Steven Sperazza
8	2:40pm	00:10:00	Tragedy of Titanic: a person on board can survive or not.	Ziyi Wang, Dewei Hu
9	2:50pm	00:10:00	Classifying groceries by image using CNN	Rui Li, Yan Wang
10	3:00pm	00:10:00	Neural Style Transfer for Video	Sarthak Chatterjee and Ashraful Islam
11	3:10pm	00:10:00	Feature selection	Zijun Cui

Guideline for the final project presentation

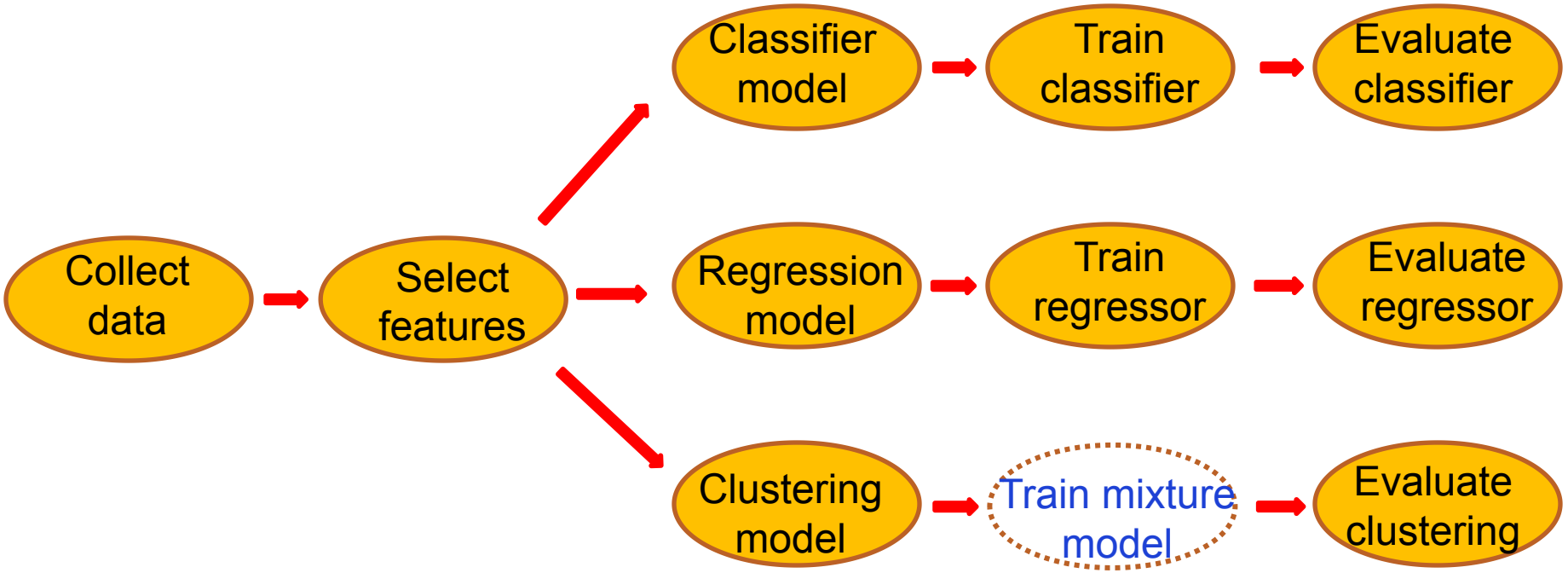
- Briefly review the importance, the problem you solved and the objective in your project - 1 or 3 slides (can used some of your proposal slides)
- The details of your solutions you used to solve the project - at least 2 slides.
- Experiment part - at least 3 slides.
  - the detailed information data set.
  - data split.
  - details about feature extraction.
  - hyper-parameter selection.
  - showing comparison results.
  - discuss based on your experimental observations.
- Conclusion and future work - 1 slide.
- Share the mistakes you encountered and the lessons you learn during completing the final project to others - 1 slide.
- List the references. - 1 slide

8-10 min presentation, including Q&A. I would like to recommend you to use informative figures as possible as you can to share what you are going to do with the other classmates.

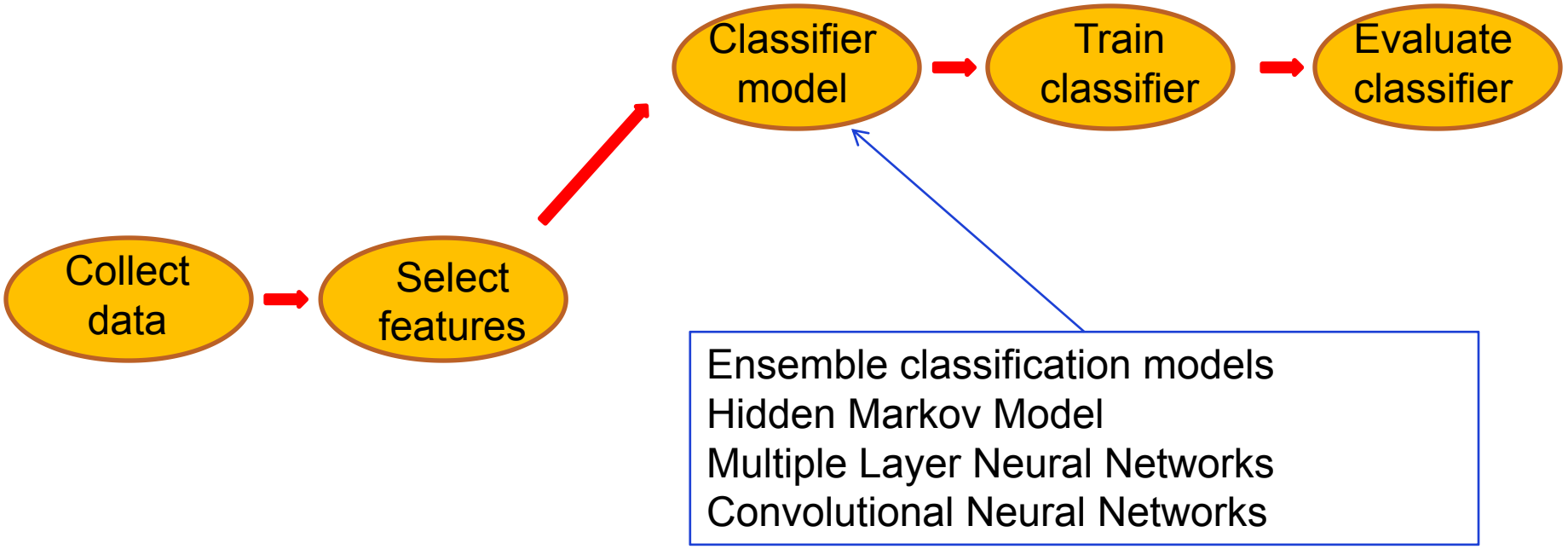
Guideline for the final project report (May 2)

- Title
- Abstract
- Introduction (can include related work)
- Related work (if not included in introduction section, list it as a seperate section)
- Techniques
- Experimens (describe the detailed information data set, data split, details about feature extraction and hyper-parameter selection. Show comparison results. and discuss based on your experimental observations).
- Conclusion and Future work.
- References.

Pattern recognition design cycle



Pattern recognition design cycle

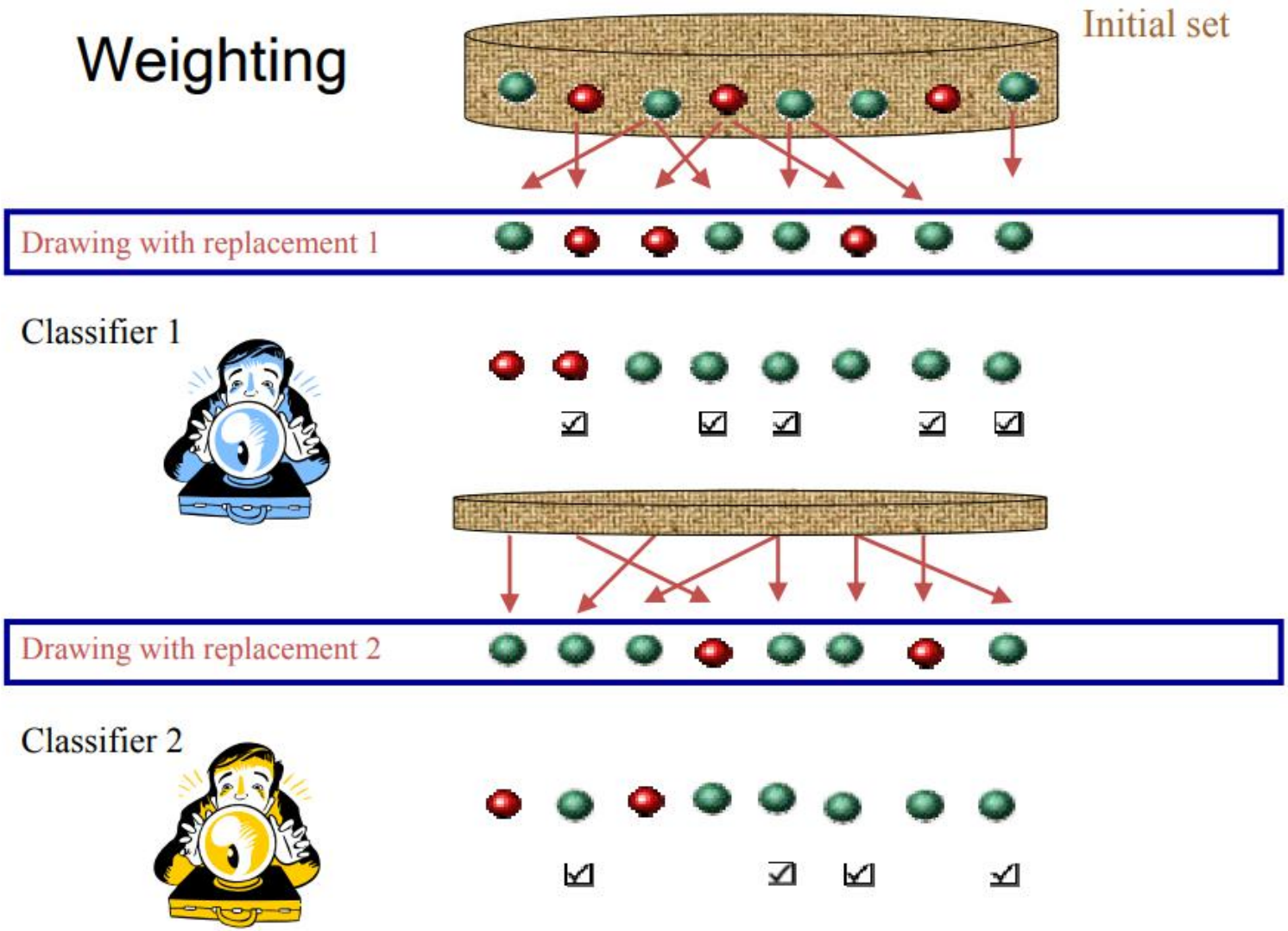


The Bagging Algorithm

- B bootstrap samples
- From which we derive:
  - B Classifiers $\in \{-1, 1\}: c^1, c^2, c^3, \ldots, c^B$
  - B Estimated probabilities $\in [0, 1]: p^1, p^2, p^3, \ldots, p^B$
- The aggregate classifier becomes:

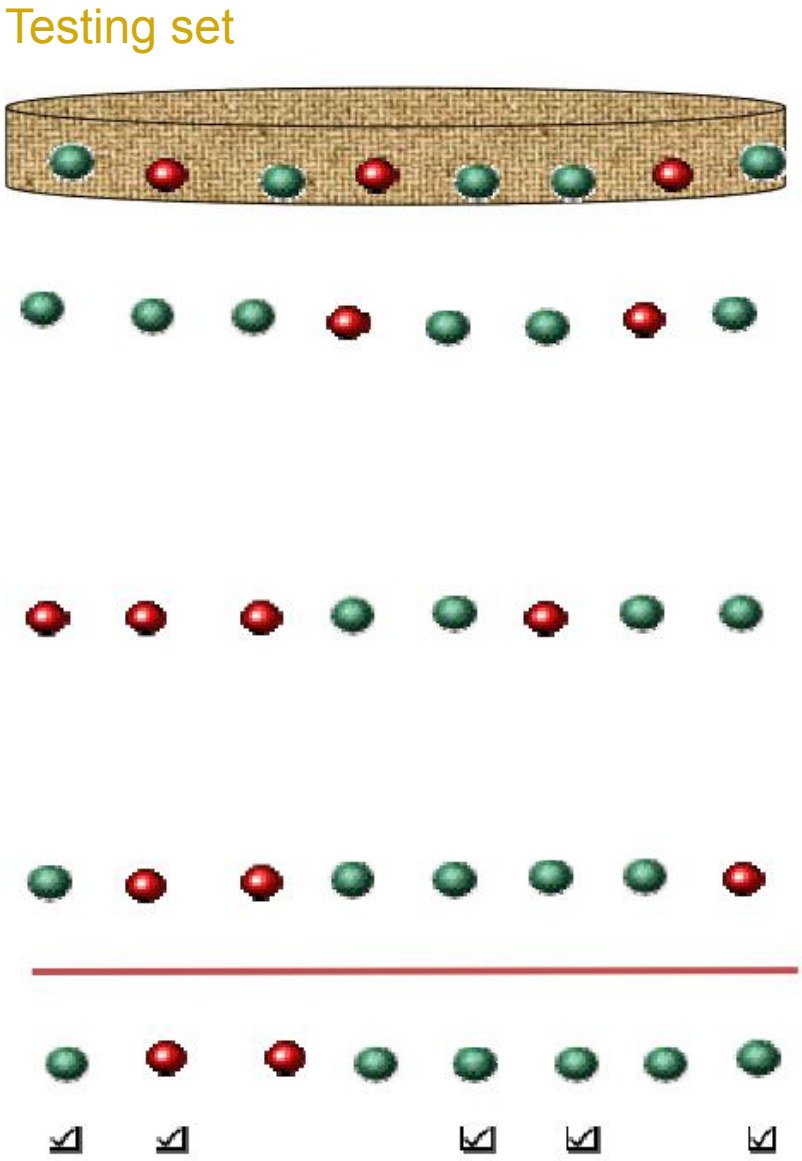
$$c_{bag}(x) = sign\left(\frac{1}{B}\sum_{b=1}^{B} c^b(x)\right) \quad \text{or} \quad p_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B} p^b(x)$$

Example (1)


Weighting
Initial set
Drawing with replacement 1
Classifier 1
Drawing with replacement 2
Classifier 2

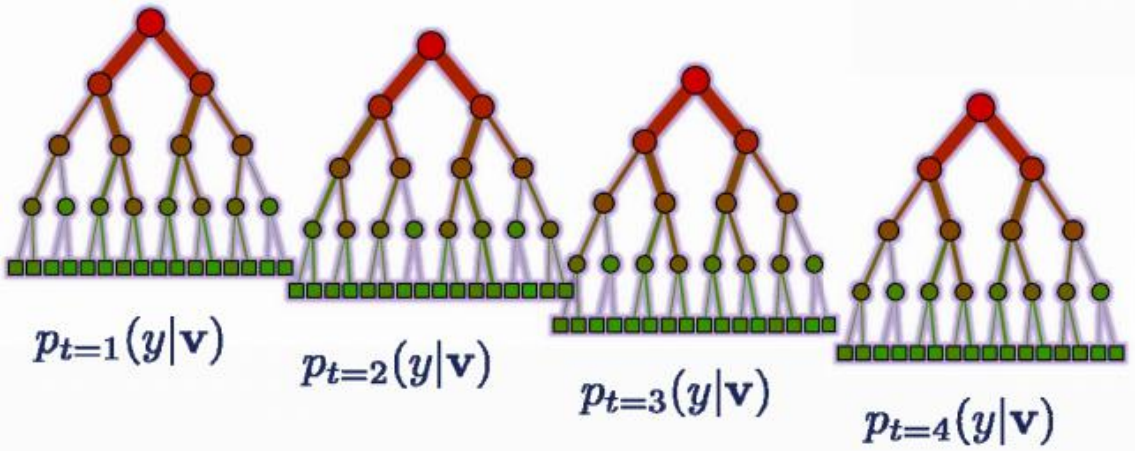
Example (2)


Aggregation
Sign
Classifier 1
+
Classifier 2
+
Classifier 3
+
…
+
Classifier T
=
Final rule

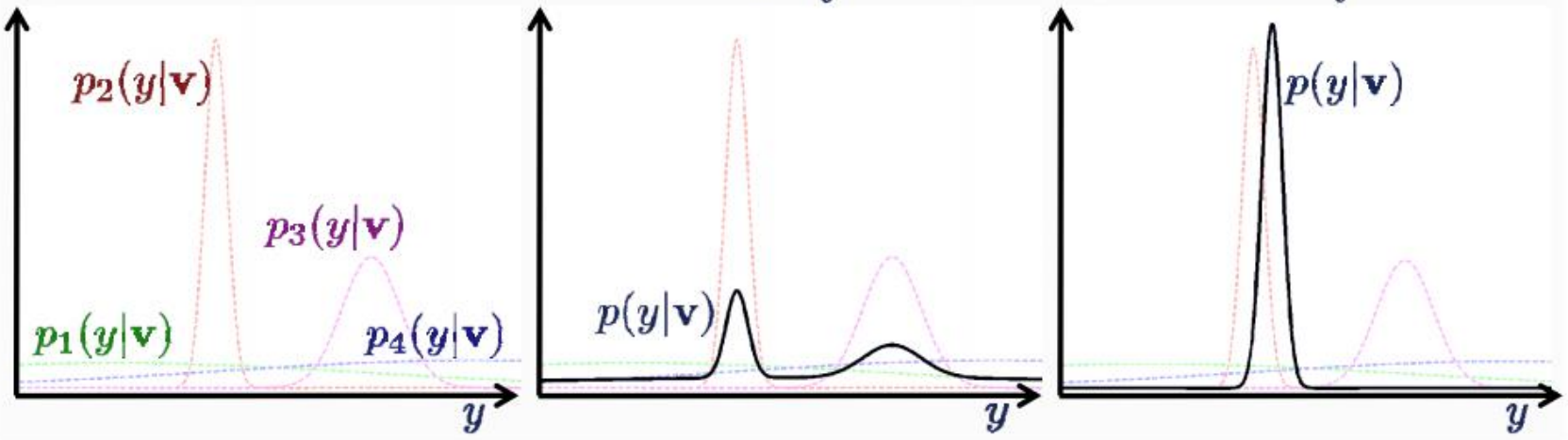

Testing set

Random Forest

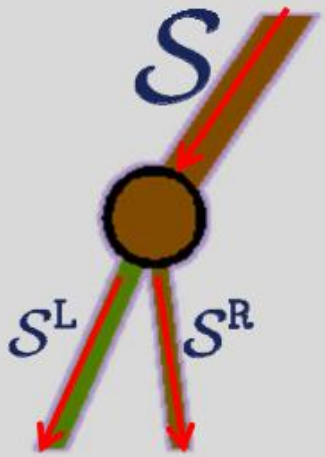
An example forest to predict continuous variables



$$p(y|\mathbf{v}) = \frac{1}{T}\sum_{t}^{T} p_t(y|\mathbf{v}) \qquad p(y|\mathbf{v}) = \frac{1}{Z}\prod_{t}^{T} p_t(y|\mathbf{v})$$



Training and Information Gain



Information gain

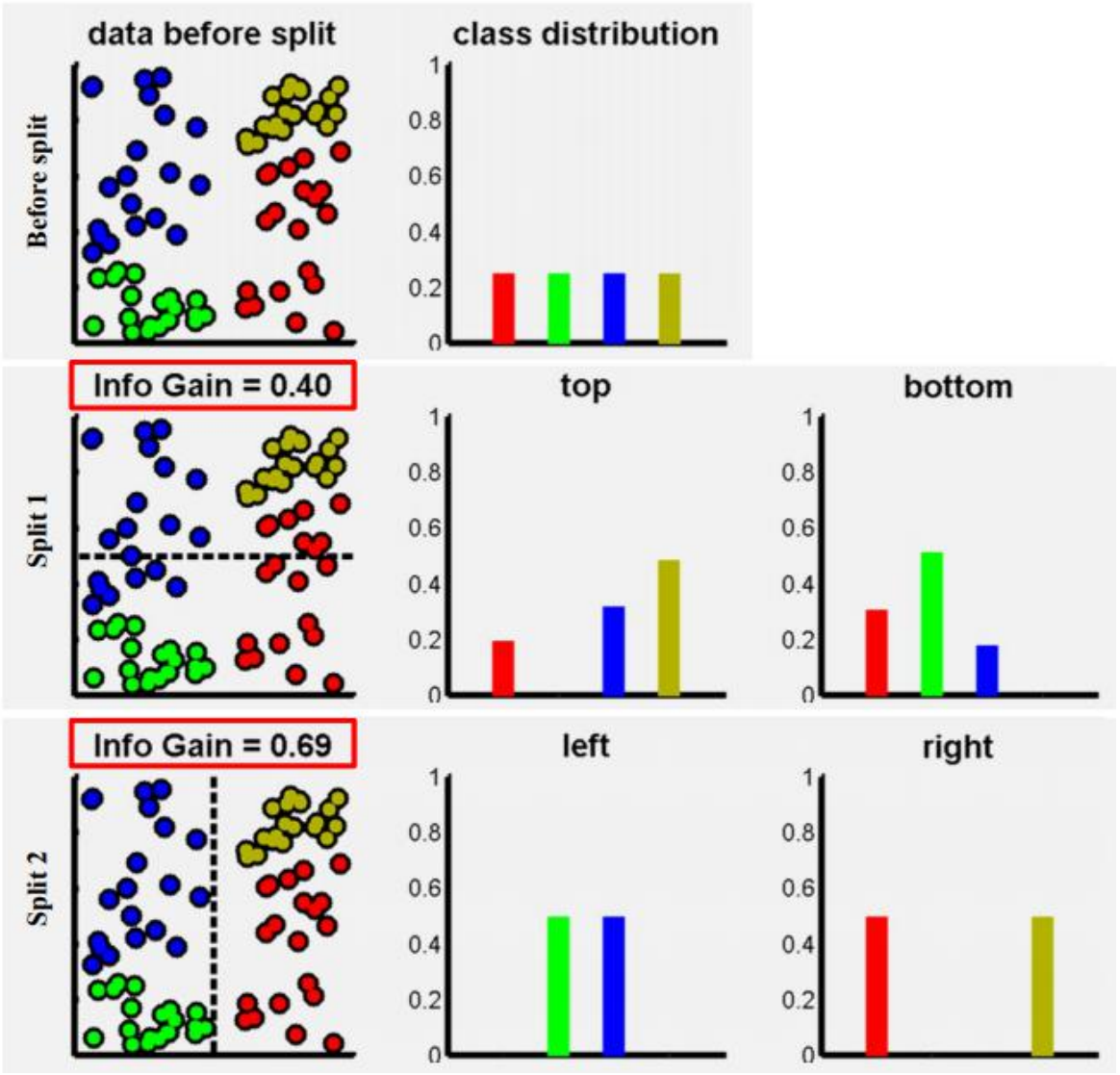
$$I(\mathcal{S},\boldsymbol{\theta}) = H(\mathcal{S}) - \sum_{i\in\{\mathrm{L},\mathrm{R}\}} \frac{|\mathcal{S}^i|}{|\mathcal{S}|} H(\mathcal{S}^i)$$

Shannon's entropy

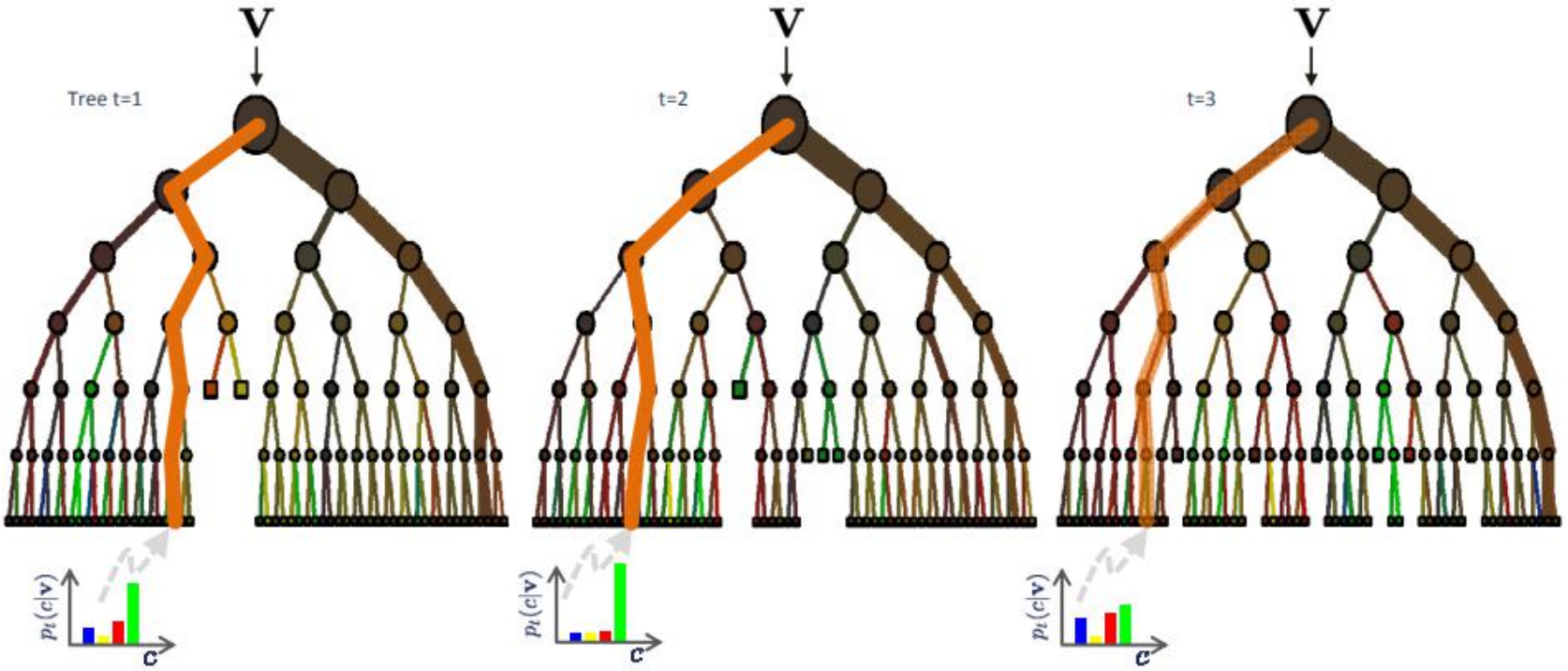
$$H(\mathcal{S}) = -\sum_{c\in\mathcal{C}} p(c)\log(p(c))$$

Node training

$$\boldsymbol{\theta} = \arg\max_{\boldsymbol{\theta}\in\mathcal{T}_j} I(\mathcal{S}_j,\boldsymbol{\theta})$$

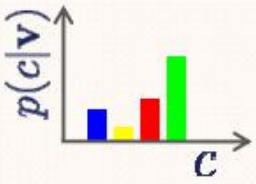


Classification Forest: Ensemble Model



The ensemble model

Forest output probability $p(c|\mathbf{v}) = \frac{1}{T}\sum_{t}^{T} p_t(c|\mathbf{v})$



AdaBoost

- Constructing Dt

$$D_1(i) = \frac{1}{m}$$

$$D_{t+1} = \frac{D_t(i)}{Z_t} c(x)$$

$$c(x) = \begin{cases} e^{-\alpha_t} & : y_i = h_t(x_i) \\ e^{\alpha_t} & : y_i \neq h_t(x_i) \end{cases}$$

$$D_{t+1} = \frac{D_t(i)}{Z_t} e^{-\alpha_t y_i h_t(x_i)}$$

where Zt is a normalization constant

Proof:

$$\begin{aligned} Z_t &= \sum_{i:y_i \neq h_t(x_i)} D_t(i) e^{\alpha_t} + \sum_{i:y_i = h_t(x_i)} D_t(i) e^{-\alpha_t} \\ &= \epsilon_t e^{\alpha_t} + (1 - \epsilon_t) e^{-\alpha_t} \\ &= 2\sqrt{\epsilon_t(1 - \epsilon_t)} = \sqrt{1 - (1 - 2\epsilon_t)^2} \end{aligned}$$

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

The AdaBoost Algorithm

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:

- Train base learner using distribution D_t.
- Get base classifier $h_t : X \to \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:

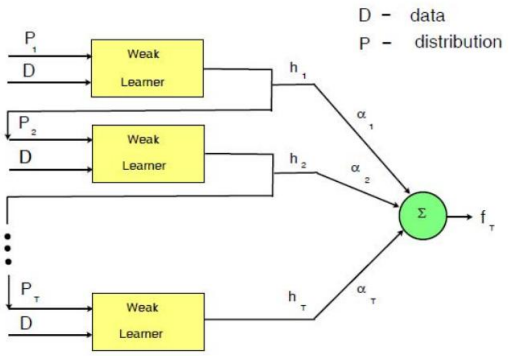
$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

with minimum ϵ_t

Output the final classifier:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$



The AdaBoost Algorithm

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:

- Train base learner using distribution D_t.
- Get base classifier $h_t : X \to \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

$$\epsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i]$$

$$\epsilon_t = \frac{1}{\sum_{i=1}^{n} D_t(i)} \sum_{i=1}^{m} D_t(i) \delta(h_t(x_i) \neq y_i)$$

Analyzing training error

- What α_t to choose for hypothesis h_t?

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

ε_t - weighted training error

- If each weak learner h_t is slightly better than random guessing ($\varepsilon_t < 0.5$), then training error of AdaBoost decays exponentially fast in number of rounds T.

$$\frac{1}{m} \sum_{i=1}^{m} \delta(H(x_i) \neq y_i) \leq \exp\left(-2 \sum_{t=1}^{T} (1/2 - \epsilon_t)^2 \right)$$

Training Error

What α_t to choose for hypothesis h_t?

- We can tighten this bound greedily, by choosing αt and ht on each iteration to minimize Zt.

$$Z_t = \sum_{i=1}^{m} D_t(i) \exp(-\alpha_t y_i h_t(x_i))$$

- For boolean target function, this is accomplished by [Freund & Schapire '97]:

$$\boxed{\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)}$$

Proof:

$$\begin{aligned} Z_t &= \sum_{i:y_i \neq h_t(x_i)} D_t(i) e^{\alpha_t} + \sum_{i:y_i = h_t(x_i)} D_t(i) e^{-\alpha_t} \\ &= \epsilon_t e^{\alpha_t} + (1-\epsilon_t) e^{-\alpha_t} \end{aligned}$$

$$\frac{\partial Z_t}{\alpha_t} = \epsilon_t e^{\alpha_t} - (1-\epsilon_t) e^{-\alpha_t} = 0 \qquad \Rightarrow e^{2\alpha_t} = \frac{1-\epsilon_t}{\epsilon_t}$$

What α_t to choose for hypothesis h_t?

- We can tighten this bound greedily, by choosing αt and ht on each iteration to minimize Zt.

$$Z_t = \sum_{i=1}^{m} D_t(i) \exp(-\alpha_t y_i h_t(x_i))$$

- For boolean target function, this is accomplished by [Freund & Schapire '97]:

$$\boxed{\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)}$$

Proof:

$$\begin{aligned} Z_t &= \sum_{i:y_i \neq h_t(x_i)} D_t(i) e^{\alpha_t} + \sum_{i:y_i = h_t(x_i)} D_t(i) e^{-\alpha_t} \\ &= \epsilon_t e^{\alpha_t} + (1 - \epsilon_t) e^{-\alpha_t} \\ &= 2\sqrt{\epsilon_t (1 - \epsilon_t)} = \sqrt{1 - (1 - 2\epsilon_t)^2} \end{aligned}$$

Dumb classifiers made Smart

- Training error of final classifier is bounded by:

$$\frac{1}{m}\sum_{i=1}^{m}\delta(H(x_i)\neq y_i) \leq \prod_t Z_t = \prod_t \sqrt{1-(1-2\epsilon_t)^2}$$

$$\leq \exp\left(-2\sum_{t=1}^{T}\underbrace{(1/2-\epsilon_t)^2}\right)$$

grows as ε_t moves away from 1/2

If each classifier is (at least slightly) better than random $\varepsilon_t < 0.5$

AdaBoost will achieve zero *training error* exponentially fast (in number of rounds T) !!

What about test error?

Hidden Markov Models

- Parameters – stationary/homogeneous markov model (independent of time t)

Initial probabilities

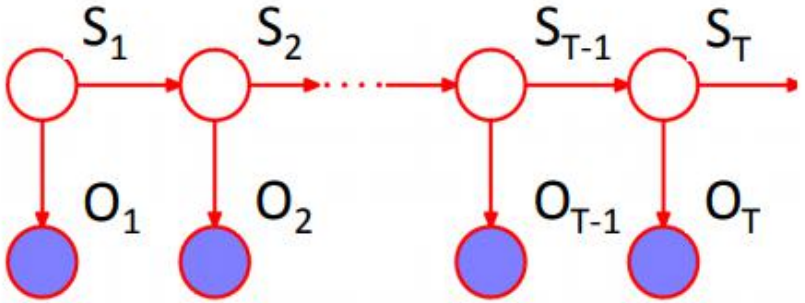
$p(S_1 = i) = \pi_i$

Transition probabilities

$p(S_t = j | S_{t-1} = i) = p_{ij}$

Emission probabilities

$p(O_t = y | S_t = i) = q_i^y$



$$p(\{S_t\}_{t=1}^T, \{O_t\}_{t=1}^T) = p(S_1) \prod_{t=2}^T p(S_t|S_{t-1}) \prod_{t=1}^T p(O_t|S_t)$$

Three main problems in HMMs

- **Evaluation** – Given HMM parameters & observation seqn $\{O_t\}_{t=1}^T$

  find $p(\{O_t\}_{t=1}^T)$ prob of observed sequence

- **Decoding** – Given HMM parameters & observation seqn $\{O_t\}_{t=1}^T$

  find $\arg \max_{s_1,\ldots,s_T} p(\{S_t\}_{t=1}^T | \{O_t\}_{t=1}^T)$ most probable

  sequence of hidden states

- **Learning** – Given HMM with unknown parameters and $\{O_t\}_{t=1}^T$ observation sequence

  find $\arg \max_{\theta} p(\{O_t\}_{t=1}^T | \theta)$ parameters that maximize

  likelihood of observed data

HMM Algorithms

- Evaluation
  – What is the probability of the observed sequence? **Forward Algorithm**

- Decoding
  – What is the probability that the third roll was loaded given the observed sequence? **Forward–Backward Algorithm**
  – What is the most likely die sequence given the observed sequence? **Viterbi Algorithm**

- Learning
  – Under what parameterization is the observed sequence most probable? **Baum–Welch Algorithm (EM)**

Notations

$$p(\{O_t\}_{t=1}^T) = \sum_k \underbrace{p(\{O_t\}_{t=1}^T, S_T = k)}_{\alpha_T^k}$$

Compute recursively

$$\begin{aligned} p(S_t = k, \{O_t\}_{t=1}^T) &= p(O_1, \ldots, O_t, S_t = k, O_{t+1}, \ldots, O_T) \\ &= \underbrace{p(O_1, \ldots, O_t, S_t = k)}_{\alpha_t^k} \underbrace{p(O_{t+1}, \ldots, O_T | S_t = k)}_{\beta_t^k} \end{aligned}$$

$$\arg \max_{\{S_t\}_{t=1}^T} p(\{S_t\}_{t=1}^T | \{O_t\}_{t=1}^T) = \arg \max_{\{S_t\}_{t=1}^T} p(\{S_t\}_{t=1}^T, \{O_t\}_{t=1}^T)$$

$$= \arg \max_k \underbrace{\max_{\{S_t\}_{t=1}^{T-1}} p(S_T = k, \{S_t\}_{t=1}^{T-1}, \{O_t\}_{t=1}^T)}_{V_T^k}$$

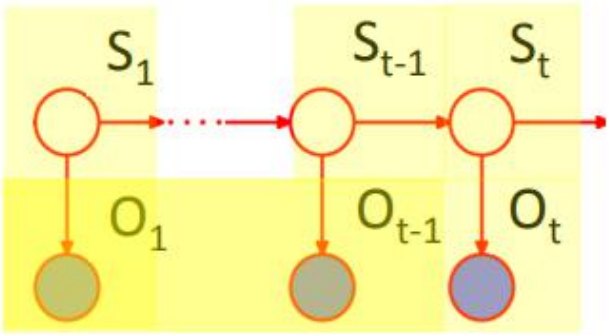
Compute recursively

Forward Probability

$$p(\{O_t\}_{t=1}^T) = \sum_k p(\{O_t\}_{t=1}^T, S_T = k) = \sum_k \alpha_T^k$$

Compute forward probability α_t^k recursively over t

$$\alpha_t^k := p(O_1, \ldots, O_t, S_t = k)$$



Introduce S_{t-1}

Chain rule

Markov assumption

$$= p(O_t|S_t = k) \sum \alpha_{t-1}^i p(S_t = k|S_{t-1} = i)$$

Forward Algorithm

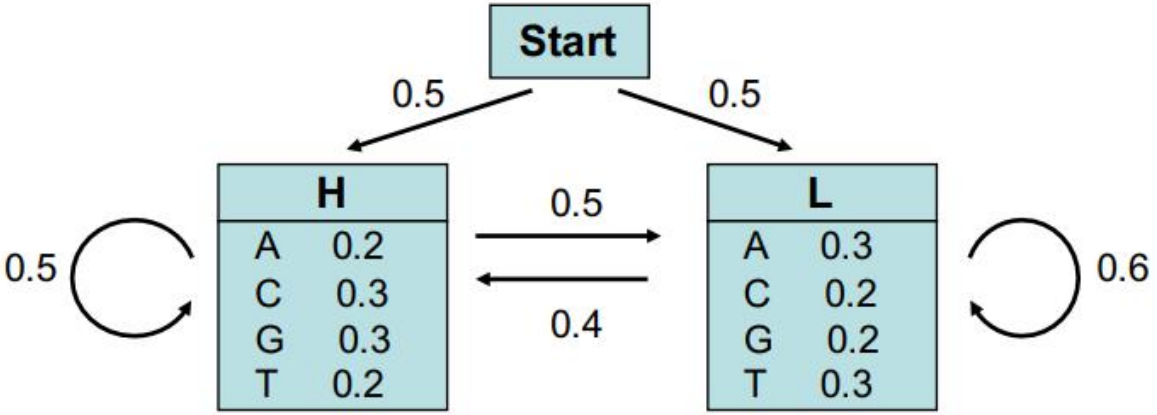
Can compute α_t^k for all k, t using dynamic programming:

- Initialize: $\alpha_1^k = p(O_1 | S_1 = k)\, p(S_1 = k)$ for all k
- Iterate: for t = 2, …, T

$$\alpha_t^k = p(O_t | S_t = k) \sum_i \alpha_{t-1}^i \, p(S_t = k | S_{t-1} = i) \quad \text{for all k}$$

- Termination: $p(\{O_t\}_{t=1}^T) = \sum_k \alpha_T^k$

Forward Algorithm Example

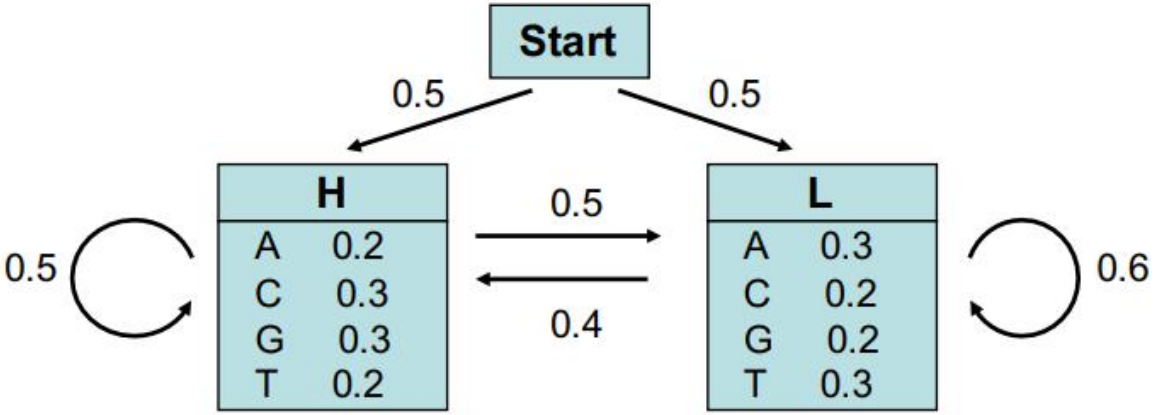


Forward algorithm

Consider now the sequence S= **GGCA**

	Start	G	G	C	A
H	0	0.5*0.3=0.15			
L	0	0.5*0.2=0.1			

Forward Algorithm Example

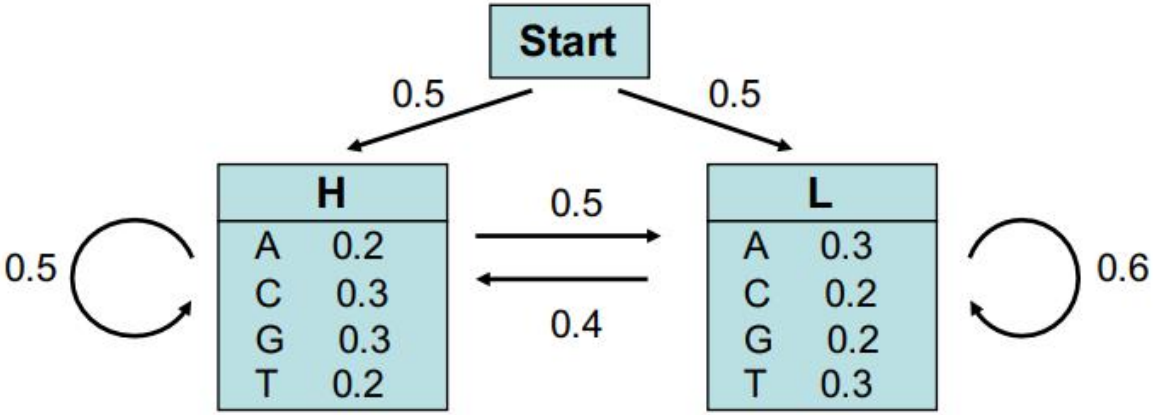


Forward algorithm

Consider now the sequence S= **GGCA**

	Start	G	G	C	A
H	0	0.5*0.3=0.15 →	0.15*0.5*0.3 + 0.1*0.4*0.3=0.0345		
L	0	0.5*0.2=0.1 →			

Forward Algorithm Example

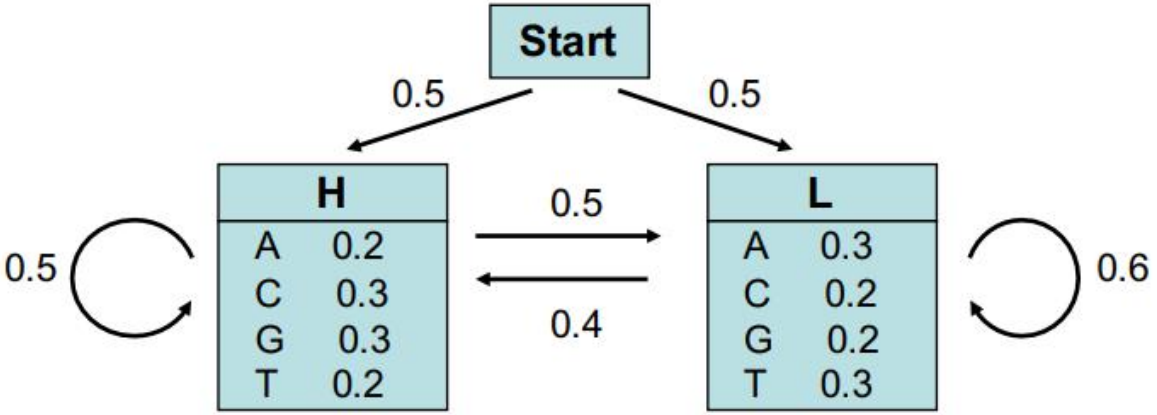


Forward algorithm

Consider now the sequence S= **GGCA**

	Start	G	G	C	A
H	0	0.5*0.3=0.15	0.15*0.5*0.3 + 0.1*0.4*0.3=0.0345		
L	0	0.5*0.2=0.1	0.1*0.6*0.2 + 0.15*0.5*0.2=0.027		

Forward Algorithm Example

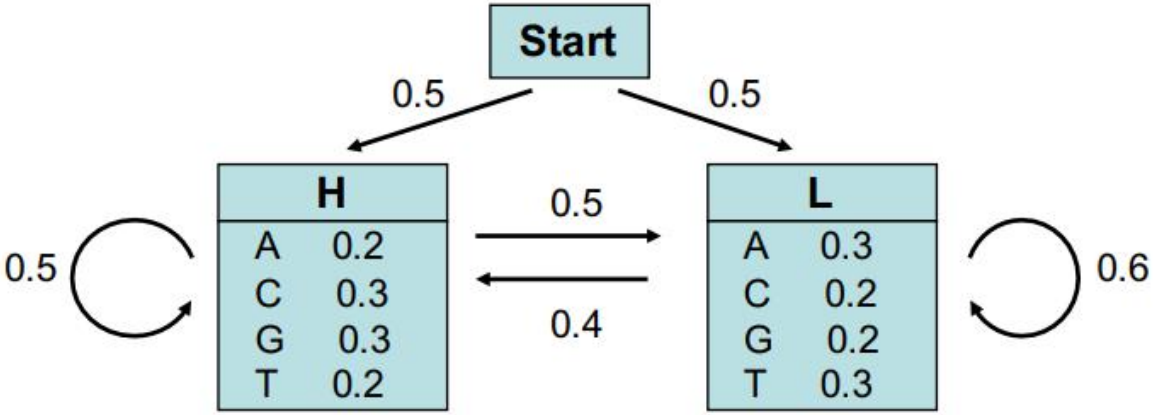


Forward algorithm

Consider now the sequence S= **GGCA**

	Start	G	G	C	A
H	0	0.5*0.3=0.15	0.15*0.5*0.3 + 0.1*0.4*0.3=0.0345	... + ...	
L	0	0.5*0.2=0.1	0.1*0.6*0.2 + 0.15*0.5*0.2=0.027	... + ...	

Forward Algorithm Example



Forward algorithm

Consider now the sequence S= **GGCA**

	Start	G	G	C	A
H	0	0.5*0.3=0.15	0.15*0.5*0.3 + 0.1*0.4*0.3=0.0345	... + ...	0.0013767
L	0	0.5*0.2=0.1	0.1*0.6*0.2 + 0.15*0.5*0.2=0.027	... + ...	0.0024665

=> The probability that the sequence S was generated by the HMM model is thus P(S)=0.0038432.

$\Sigma = 0.0038432$

Viterbi Algorithm

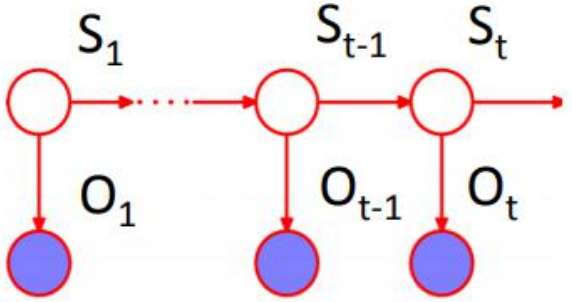
$$\max_{\{S_t\}_{t=1}^T} p(\{S_t\}_{t=1}^T, \{O_t\}_{t=1}^T) = \max_k V_T^k$$

Compute probability V_t^k recursively over t

$$V_t^k := \max_{S_1,\ldots,S_{t-1}} p(S_t = k, S_1, \ldots, S_{t-1}, O_1, \ldots, O_t)$$

. Bayes rule

. Markov assumption



$$= p(O_t|S_t = k) \max_i p(S_t = k|S_{t-1} = i) V_{t-1}^i$$

Viterbi Algorithm

Can compute V_t^k for all k, t using dynamic programming:

- Initialize: $V_1^k = p(O_1|S_1=k)p(S_1 = k)$ for all k
- Iterate: for t = 2, …, T

$$V_t^k = p(O_t|S_t = k) \max_i p(S_t = k|S_{t-1} = i) V_{t-1}^i \quad \text{for all k}$$

- Termination:

$$\max_{\{S_t\}_{t=1}^T} p(\{S_t\}_{t=1}^T, \{O_t\}_{t=1}^T) = \max_k V_T^k$$

Traceback:

$$S_T^* = \arg\max_k V_T^k$$

Viterbi Algorithm Example

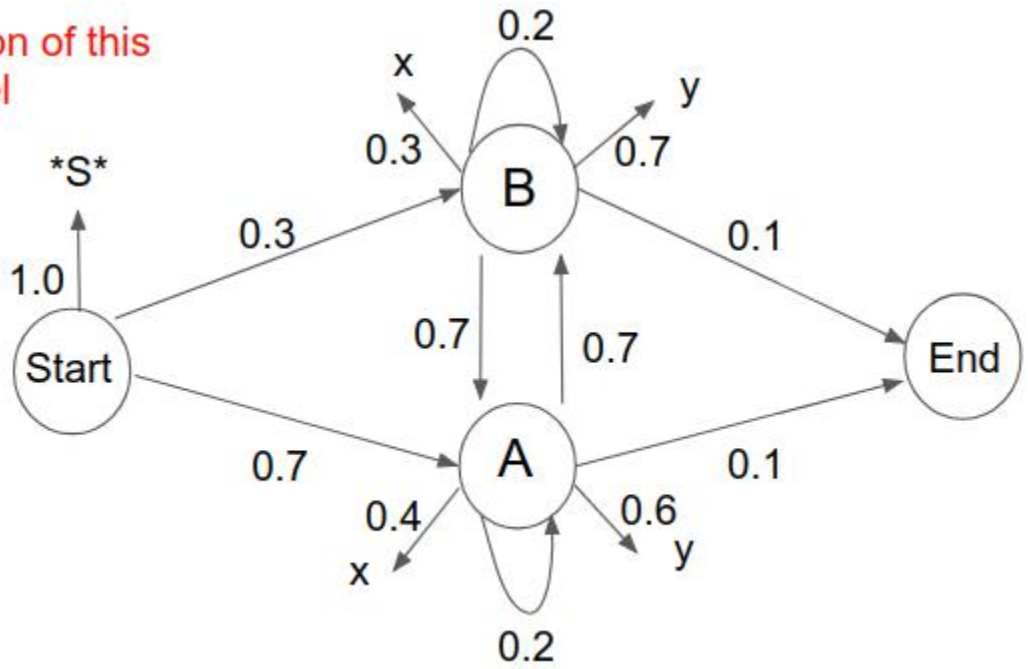
Transition Probability Table

	A	B	End
Start	0.7	0.3	0
A	0.2	0.7	0.1
B	0.7	0.2	0.1

Emission Probability Table

	S	x	y
Start	1	0	0
A	0	0.4	0.6
B	0	0.3	0.7

A graph representation of this Hidden Markov Model



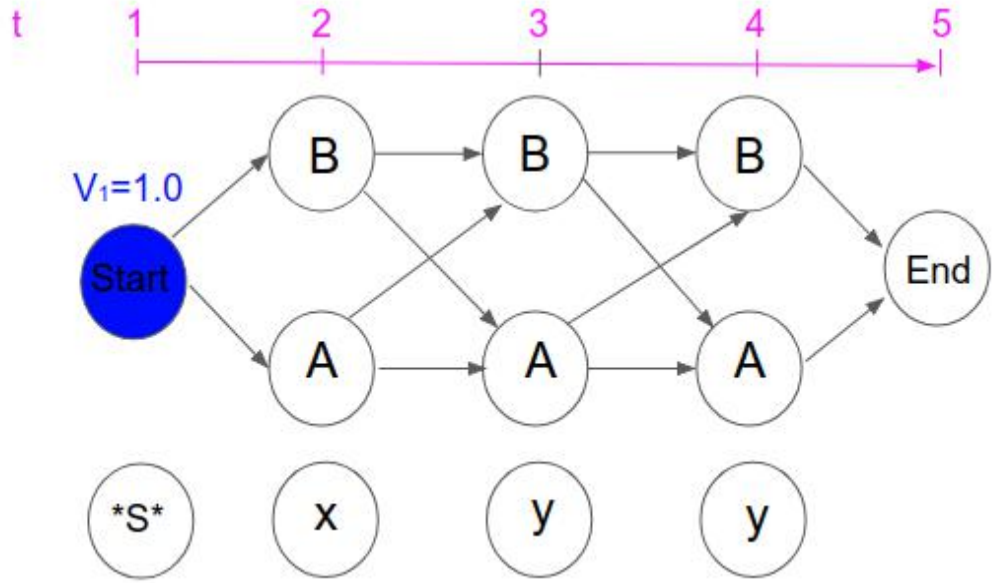
Viterbi Algorithm Example

Transition Probability Table

	A	B	End
Start	0.7	0.3	0
A	0.2	0.7	0.1
B	0.7	0.2	0.1

Emission Probability Table

	S	x	y
Start	1	0	0
A	0	0.4	0.6
B	0	0.3	0.7



Initialization: $V_1^k = p(O_1|S_1 = k)p(S_1 = k)$ for all k

$$V_1^{Start} = p(\star S \star | S_1 = Start)p(S_1 = Start)$$
$$= 1.0 \times 1.0$$
$$= 1.0$$

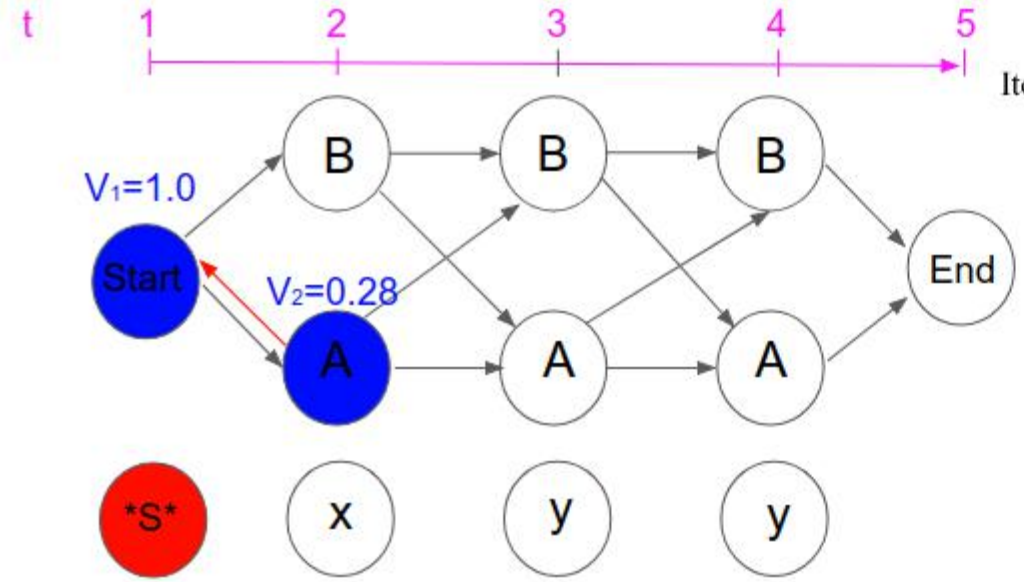
Viterbi Algorithm Example

Transition Probability Table

	A	B	End
Start	0.7	0.3	0
A	0.2	0.7	0.1
B	0.7	0.2	0.1

Emission Probability Table

	S	x	y
Start	1	0	0
A	0	0.4	0.6
B	0	0.3	0.7



Iterate: for t=2, ..., T

$$V_t^k = p(O_t|S_t = k) \max_i p(S_t = k|S_{t-1} = i) V_{t-1}^i \quad \text{for all } k$$

$$\begin{aligned} V_2^A &= p(x|S_2 = A)p(S_2 = A|S_1 = Start)V_1^{Start} \\ &= 0.4 \times \boxed{0.7 \times 1.0} \\ &= 0.28 \end{aligned}$$

Start ← A

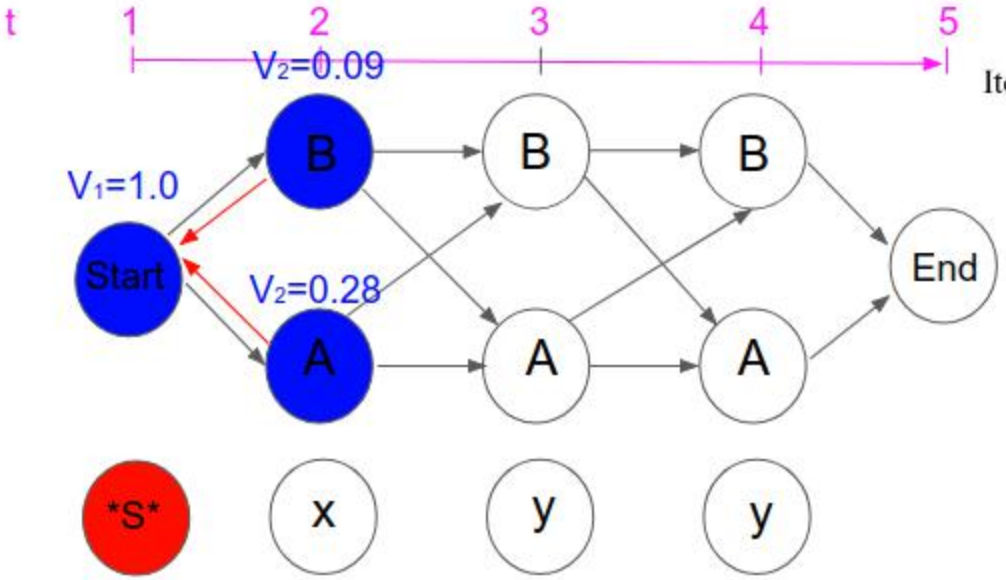
Viterbi Algorithm Example

Transition Probability Table

	A	B	End
Start	0.7	0.3	0
A	0.2	0.7	0.1
B	0.7	0.2	0.1

Emission Probability Table

	S	x	y
Start	1	0	0
A	0	0.4	0.6
B	0	0.3	0.7



Iterate: for t=2, ..., T

$$V_t^k = p(O_t|S_t = k) \max_i p(S_t = k|S_{t-1} = i) V_{t-1}^i \quad \text{for all } k$$

$$V_2^B = p(x|S_2 = B)p(S_2 = B|S_1 = Start)V_1^{Start}$$
$$= 0.3 \times \boxed{0.3 \times 1.0}$$
$$= 0.09$$

Start ← B

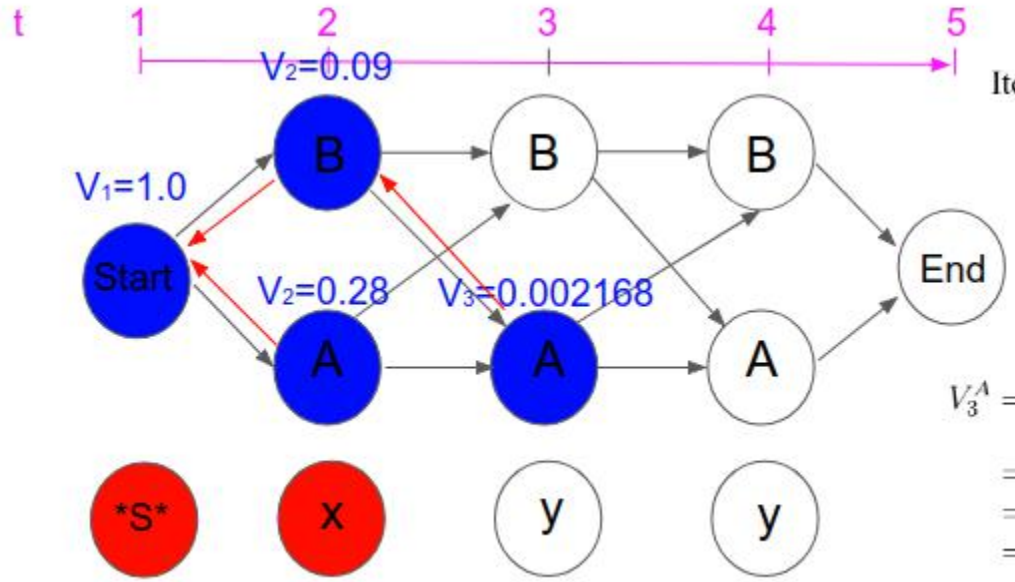
Viterbi Algorithm Example

Transition Probability Table

	A	B	End
Start	0.7	0.3	0
A	0.2	0.7	0.1
B	0.7	0.2	0.1

Emission Probability Table

	S	x	y
Start	1	0	0
A	0	0.4	0.6
B	0	0.3	0.7



Iterate: for t=2, ..., T

$$V_t^k = p(O_t|S_t = k) \max_i p(S_t = k|S_{t-1} = i) V_{t-1}^i \quad \text{for all } k$$

$$V_3^A = p(y|S_3 = A) \max\{p(S_3 = A|S_2 = A)V_2^A, p(S_3 = A|S_2 = B)V_2^B\}$$

$$= 0.6 \times \max\{0.2 \times 0.28, 0.7 \times 0.09\}$$
$$= 0.6 \times \max\{0.056, 0.063\}$$
$$= 0.0021168$$

B ← A

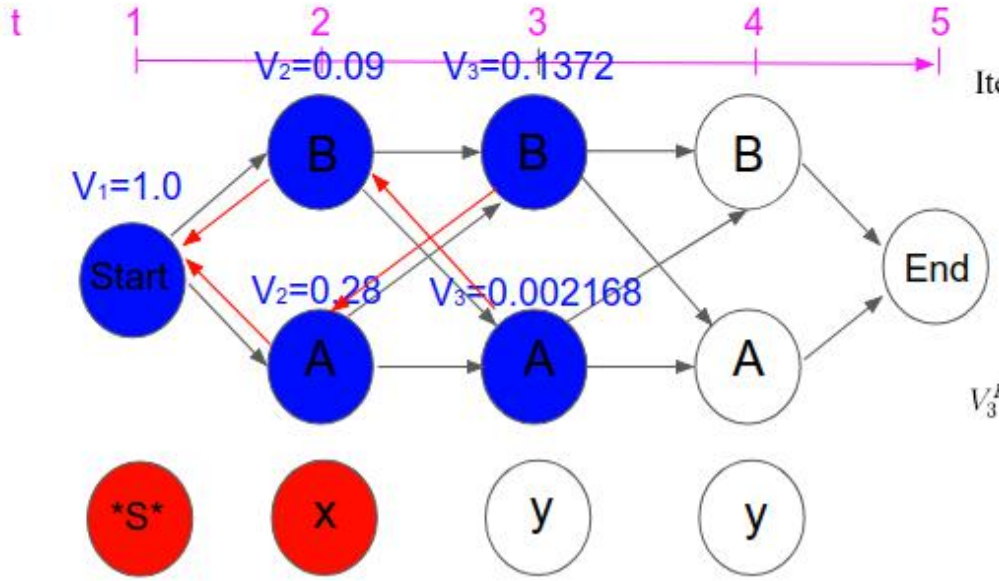
Viterbi Algorithm Example

Transition Probability Table

	A	B	End
Start	0.7	0.3	0
A	0.2	0.7	0.1
B	0.7	0.2	0.1

Emission Probability Table

	S	x	y
Start	1	0	0
A	0	0.4	0.6
B	0	0.3	0.7



Iterate: for t=2, ..., T

$$V_t^k = p(O_t|S_t = k) \max_i p(S_t = k|S_{t-1} = i) V_{t-1}^i \quad \text{for all } k$$

$$V_3^B = p(y|S_3 = B) \max\{p(S_3 = B|S_2 = A)V_2^A, p(S_3 = B|S_2 = B)V_2^B\}$$
$$= 0.7 \times \max\{0.7 \times 0.28, 0.2 \times 0.09\}$$
$$= 0.7 \times \max\{0.196, 0.018\}$$
$$= 0.1372$$

A ← B

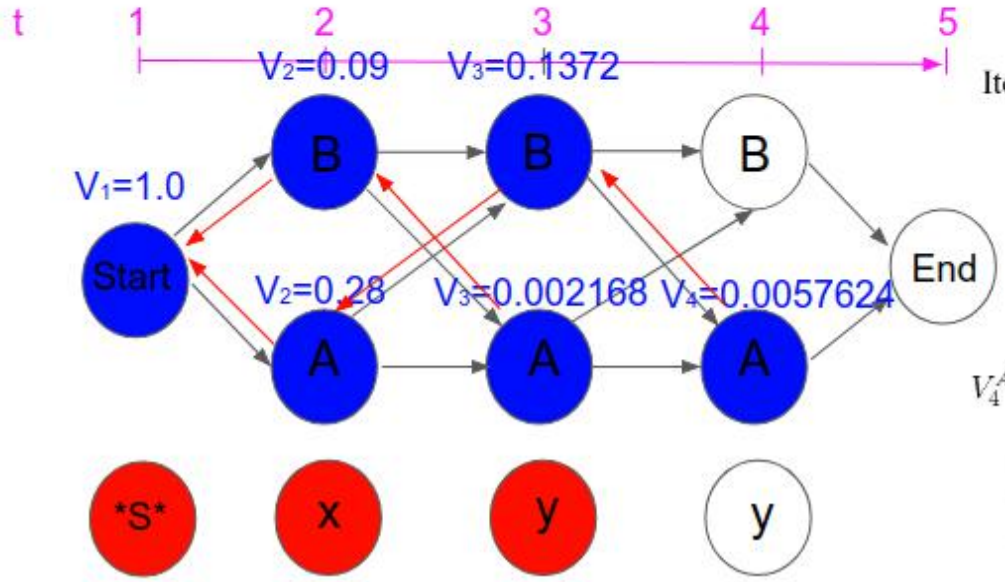
Viterbi Algorithm Example

Transition Probability Table

	A	B	End
Start	0.7	0.3	0
A	0.2	0.7	0.1
B	0.7	0.2	0.1

Emission Probability Table

	S	x	y
Start	1	0	0
A	0	0.4	0.6
B	0	0.3	0.7



Iterate: for t=2, ..., T

$$V_t^k = p(O_t|S_t = k) \max_i p(S_t = k|S_{t-1} = i) V_{t-1}^i \quad \text{for all } k$$

$$V_4^A = p(y|S_4 = A) \max\{p(S_4 = A|S_3 = A)V_3^A, p(S_4 = A|S_2 = B)V_3^B\}$$

$$= 0.6 \times \max\{0.2 \times 0.002168, 0.7 \times 0.1372\}$$
$$= 0.6 \times \max\{0.0004336, 0.009604\}$$
$$= 0.0057624$$

B ← A

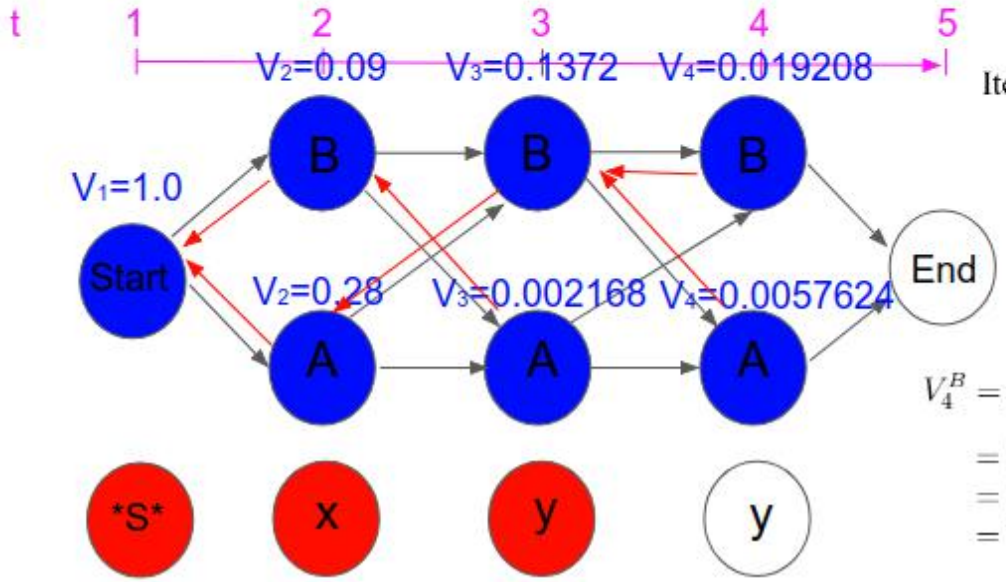
Viterbi Algorithm Example

Transition Probability Table

	A	B	End
Start	0.7	0.3	0
A	0.2	0.7	0.1
B	0.7	0.2	0.1

Emission Probability Table

	S	x	y
Start	1	0	0
A	0	0.4	0.6
B	0	0.3	0.7



Iterate: for t=2, ..., T

$$V_t^k = p(O_t|S_t = k) \max_i p(S_t = k|S_{t-1} = i) V_{t-1}^i \quad \text{for all } k$$

$$V_4^B = p(y|S_4 = B) \max\{p(S_4 = B|S_3 = A)V_3^A, p(S_4 = B|S_2 = B)V_3^B\}$$

$$= 0.7 \times \max\{0.7 \times 0.002168, 0.2 \times 0.1372\}$$

$$= 0.7 \times \max\{0.0015176, 0.02744\}$$

$$= 0.019208$$

B ← B

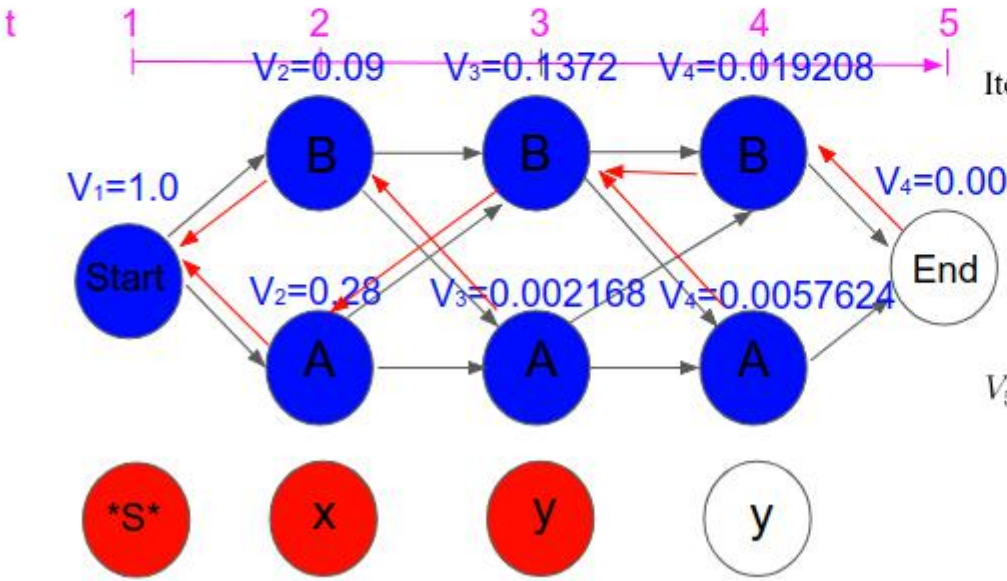
Viterbi Algorithm Example

Transition Probability Table

	A	B	End
Start	0.7	0.3	0
A	0.2	0.7	0.1
B	0.7	0.2	0.1

Emission Probability Table

	S	x	y
Start	1	0	0
A	0	0.4	0.6
B	0	0.3	0.7



Iterate: for t=2, ..., T

$$V_t^k = p(O_t|S_t = k) \max_i p(S_t = k|S_{t-1} = i) V_{t-1}^i \quad \text{for all } k$$

$$V_5^{End} = \max\{p(S_5 = End|S_4 = A)V_4^A, p(S_5 = End|S_4 = B)V_4^B\}$$

$$= \max\{0.1 \times 0.0057624, 0.1 \times 0.019208\}$$
$$= 0.0019208$$

B ← End

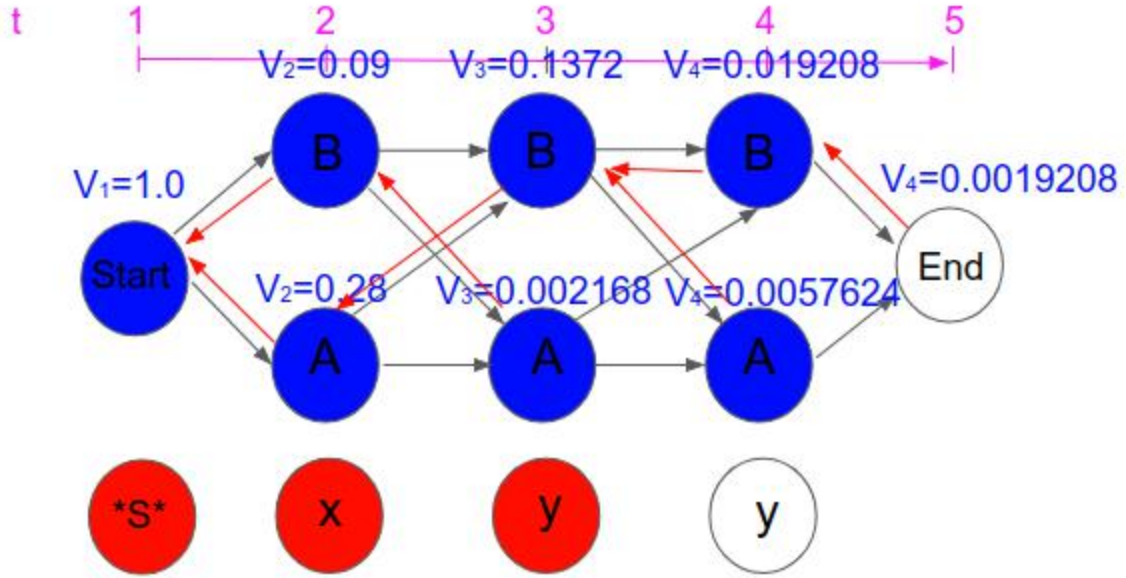
Viterbi Algorithm Example

Transition Probability Table

	A	B	End
Start	0.7	0.3	0
A	0.2	0.7	0.1
B	0.7	0.2	0.1

Emission Probability Table

	S	x	y
Start	1	0	0
A	0	0.4	0.6
B	0	0.3	0.7

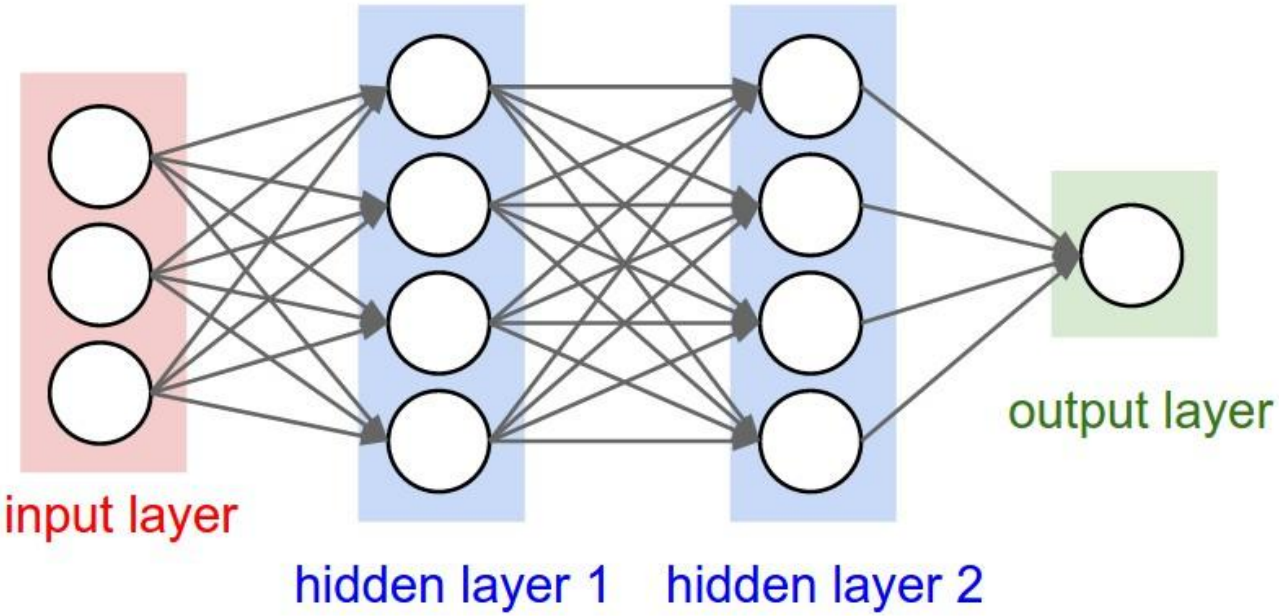


Viterbi sequence: xyy
p(**ABB**, xyy) = 0.0019208

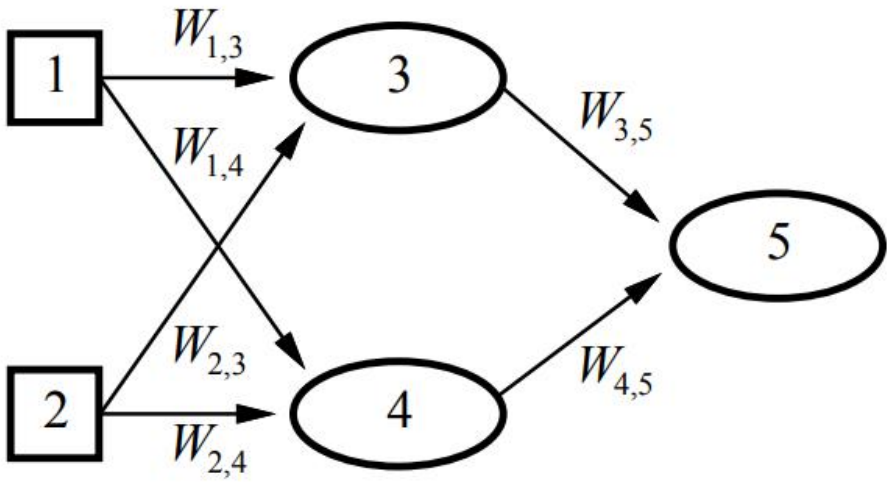
Via traceback, we can get the hidden state sequence: **ABB**.

Network structures

- Feed–forward networks:
  - single–layer perceptrons
  - multi–layer perceptrons
- Feed–forward networks implement functions, have no internal state



Feed-forward example



- Feed–forward network = a parameterized family of nonlinear functions:

$$o_5 = g(w_{35}o_3 + w_{45}o_4)$$
$$= g(w_{35}g(w_{13}x_1 + w_{23}x_2) + w_{45}g(w_{14}x_1 + w_{24}x_2))$$

- Adjusting weights changes the function: do learning this way!

Backpropagation learning algorithm BP

- Solution to credit assignment problem in MLP. *Rumelhart, Hinton and Williams (1986) (*though actually invented earlier in a PhD thesis relating to economics)
- **BP has two phases**:

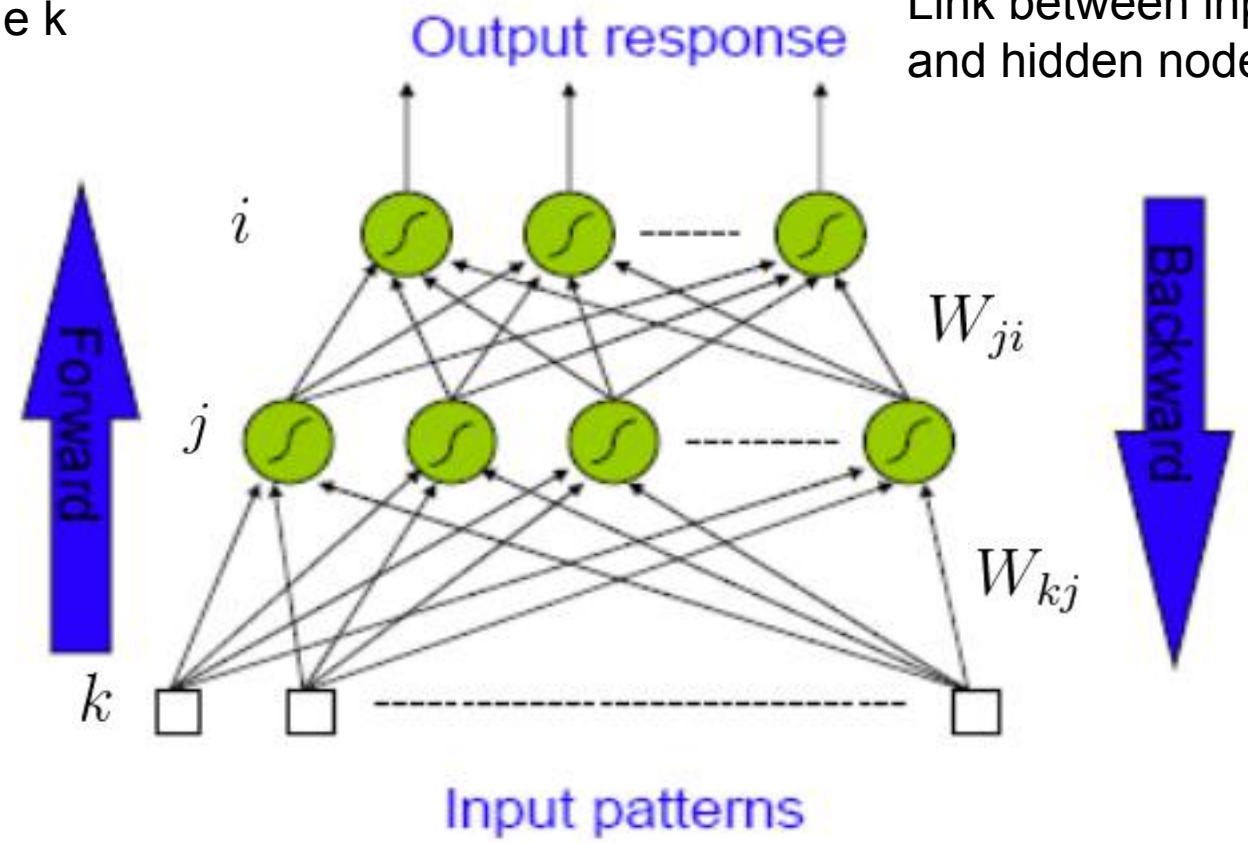
Forward pass phase: computes 'functional signal', feed forward propagation of input pattern signals through network.

Backward pass phase: computes 'error signal', ***propagates*** the error ***backwards*** through network starting at output units (where the error is the difference between actual and desired output values)

Conceptually: Forward Activity - Backward Error

Output node i
Hidden node j
Input node k

Link between hidden node j and output node i: Wji
Link between input node k and hidden node j: Wkj



Back-propagation derivation

- The squared error on a single example is defined as

$$E = \frac{1}{2}\sum_i (y_i - o_i)^2$$

where the sum is over the nodes in the output layer.

$$\begin{aligned}
\frac{\delta E}{\delta W_{kj}} &= -(y_i - o_i)\frac{\delta o_i}{\delta W_{ji}} = -(y_i - o_i)\frac{\delta g(in_i)}{\delta W_{ji}} \\
&= -(y_i - o_i)g'(in_i)\frac{\delta in_i}{\delta W_{ji}} = -(y_i - o_i)g'(in_i)\frac{\delta}{\delta W_{ji}}(\sum_j W_{ji}o_j) \\
&= -(y_i - o_i)g'(in_i)o_j \\
&= -\sigma_i o_j
\end{aligned}$$

Back-propagation derivation contd.

$$\frac{\delta E}{\delta W_{kj}} = -\sum_i (y_i - o_i)\frac{\delta o_i}{\delta W_{kj}} = -\sum_i (y_i - o_i)\frac{\delta g(in_i)}{\delta W_{kj}}$$

$$= -\sum_i (y_i - o_i) g'(in_i)\frac{\delta in_i}{\delta W_{kj}} = -\sum_i (y_i - o_i) g'(in_i)\frac{\delta}{\delta W_{kj}}(\sum_j W_{ji} o_j)$$

$$= -\sum_i \sigma_i \frac{\delta}{\delta W_{kj}}(\sum_j W_{ji} o_j) = -\sum_i \sigma_i W_{ji}\frac{\delta o_j}{\delta W_{kj}}$$

$$= -\sum_i \sigma_i W_{ji}\frac{\delta g(in_j)}{\delta W_{kj}} = -\sum_i \sigma_i W_{ji} g'(in_j)\frac{\delta in_j}{\delta W_{kj}}$$

$$= -\sum_i \sigma_i W_{ji} g'(in_j)\frac{\delta}{\delta W_{kj}}(\sum_k W_{kj} o_k) = -\sum_i \sigma_i W_{ji} g'(in_j) o_k$$

$$= -\sigma_j o_k$$

Back-propagation Learning

- Output layer: same as the single-layer perceptron

$$W_{ji} = W_{ji} + \eta \sigma_i o_j$$

where $\sigma_i = -(y_i - o_i) g'(in_i)$

- Hidden layer: back-propagation the error from the output layer.

$$\sigma_j = -\sum_i \sigma_i W_{ji} g'(in_j)$$

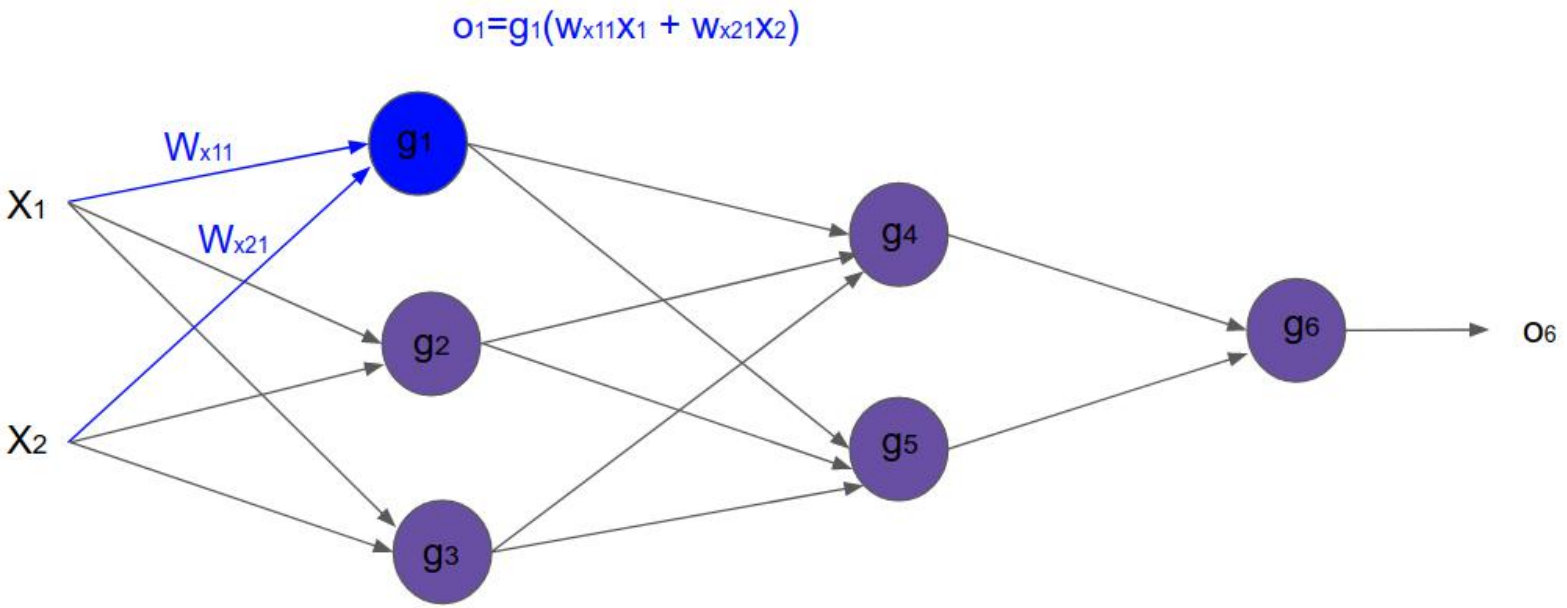
- Upadate rules for weights in the hidden layers.

$$W_{kj} = W_{kj} + \eta \sigma_j o_k$$

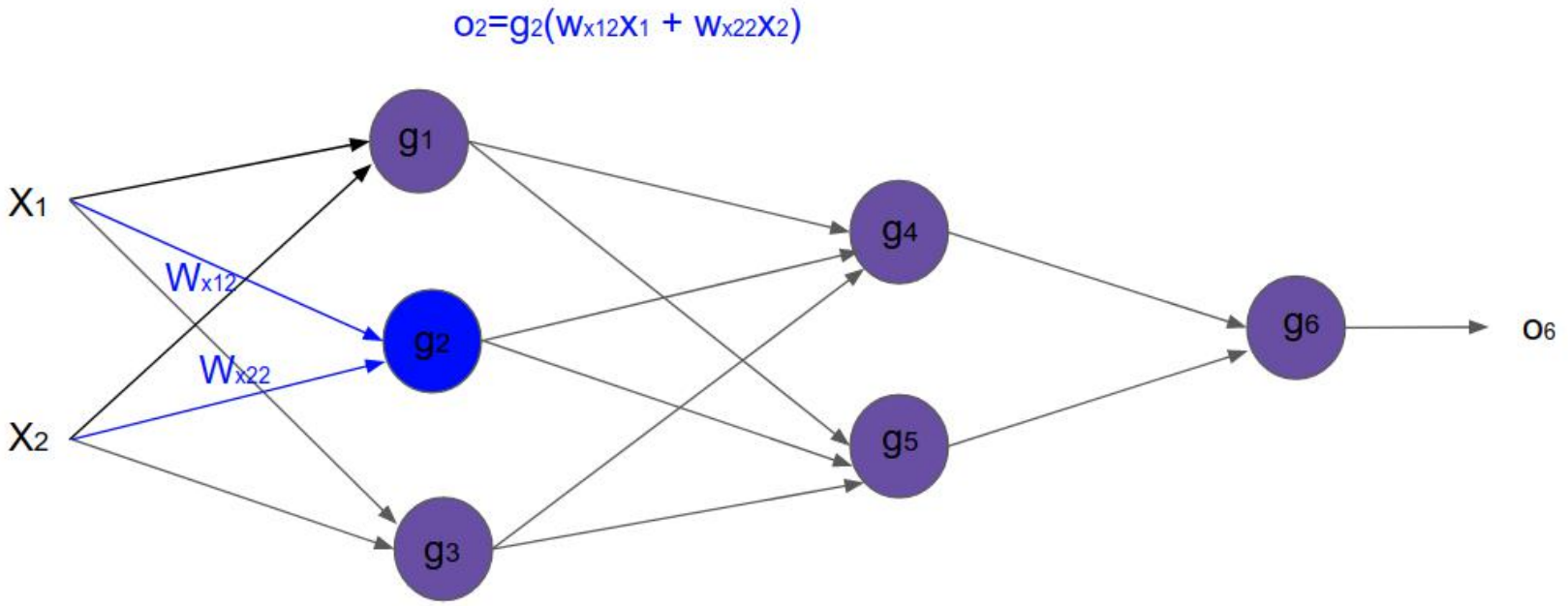
(Most neuroscientists deny that back-propagation occurs in the brain)

Learning Algorithm: Backpropagation

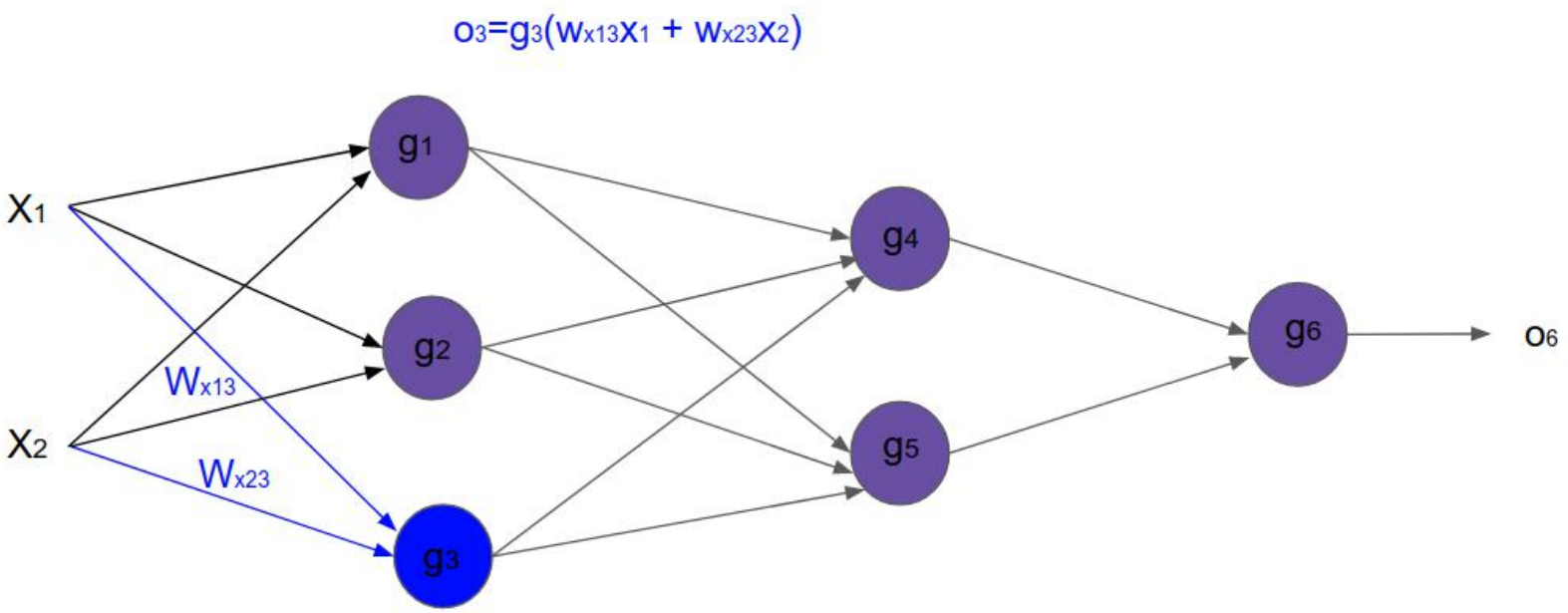
- Pictures below illustrate how signal is propagating through the network, Symbols $w_{(xm)n}$ represent weights of connections between network input x_m and neuron n in input layer.Symbols o_n represents output signal of neuron n.



Learning Algorithm: Backpropagation

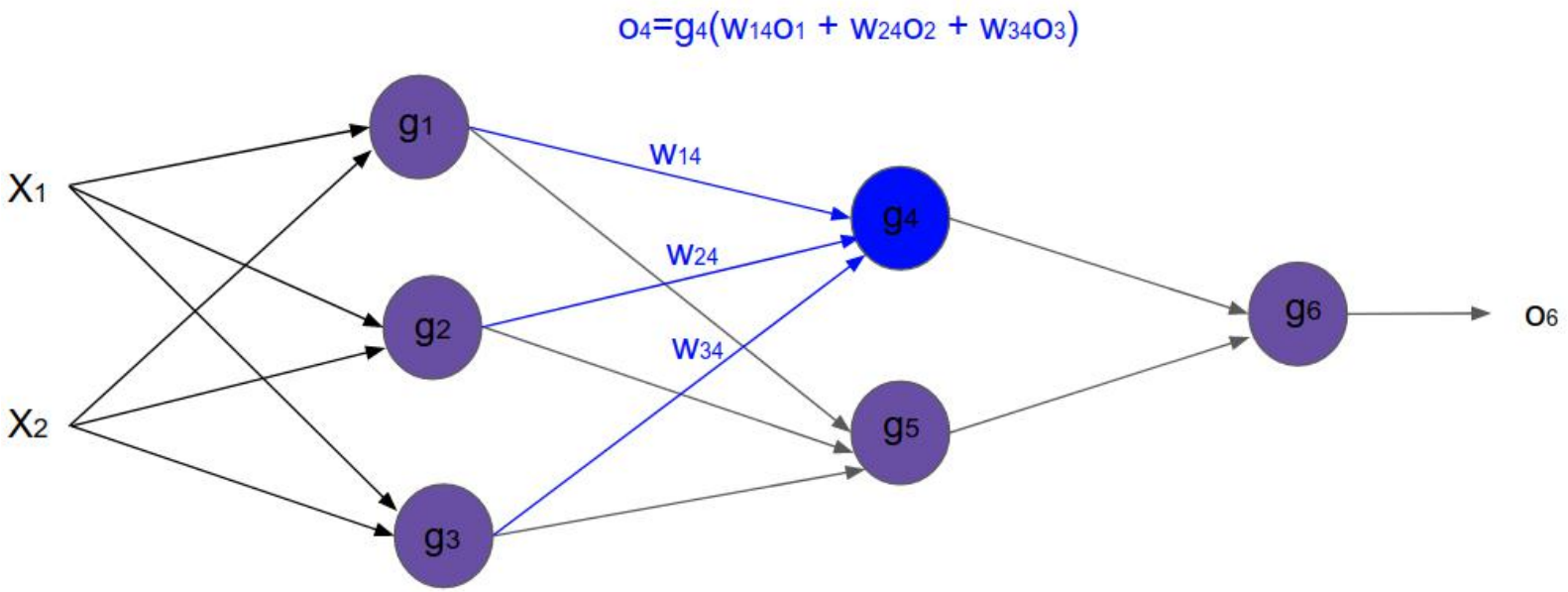


Learning Algorithm: Backpropagation

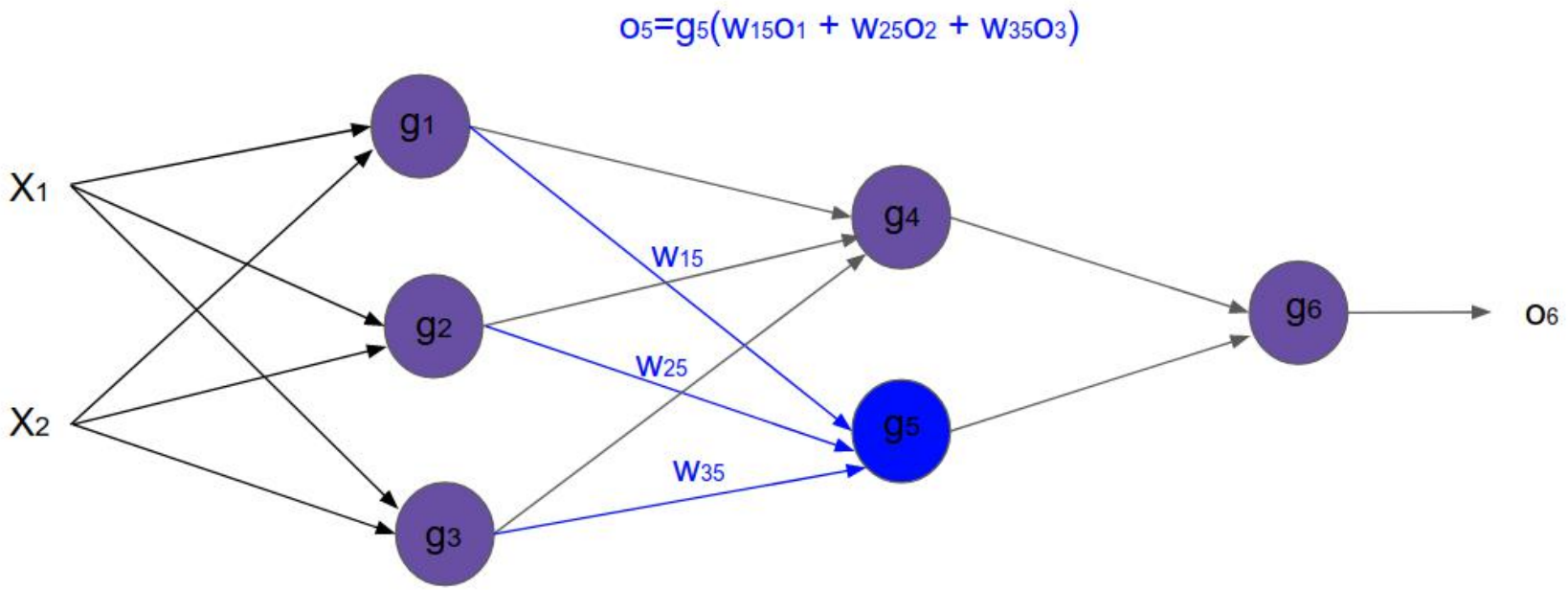


Learning Algorithm: Backpropagation

- Propagation of signals through the hidden layer. Symbols w_{mn} represent weights of connections between output of neuron m and input of neuron n in the next layer.

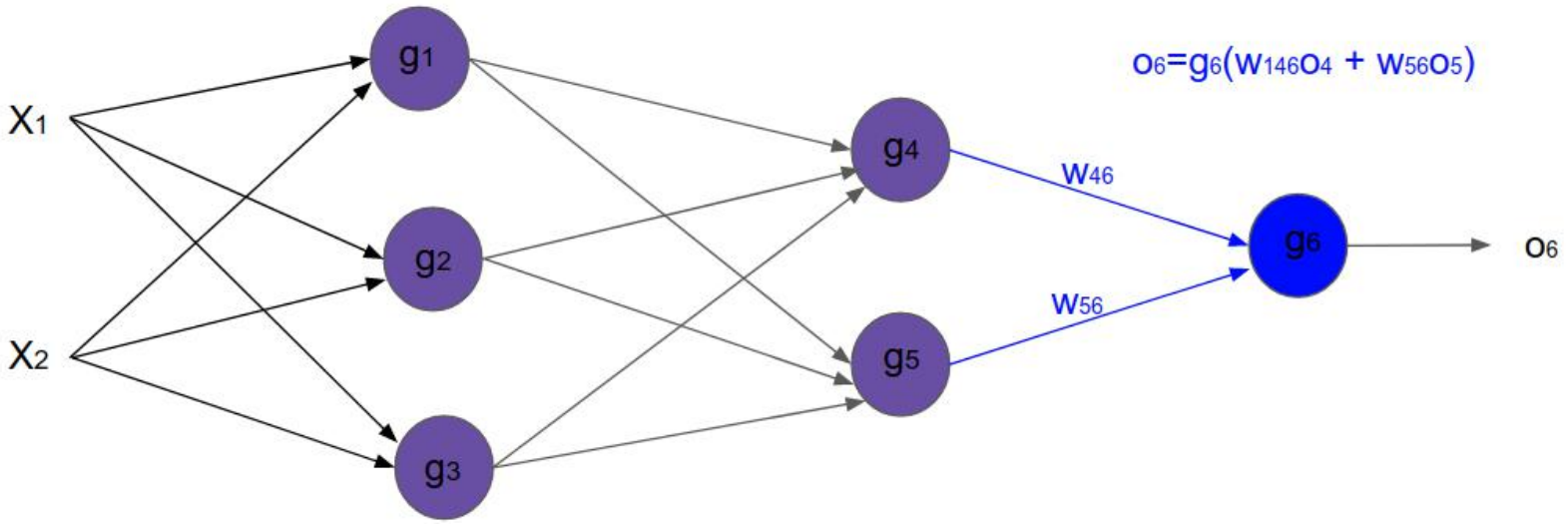


Learning Algorithm: Backpropagation



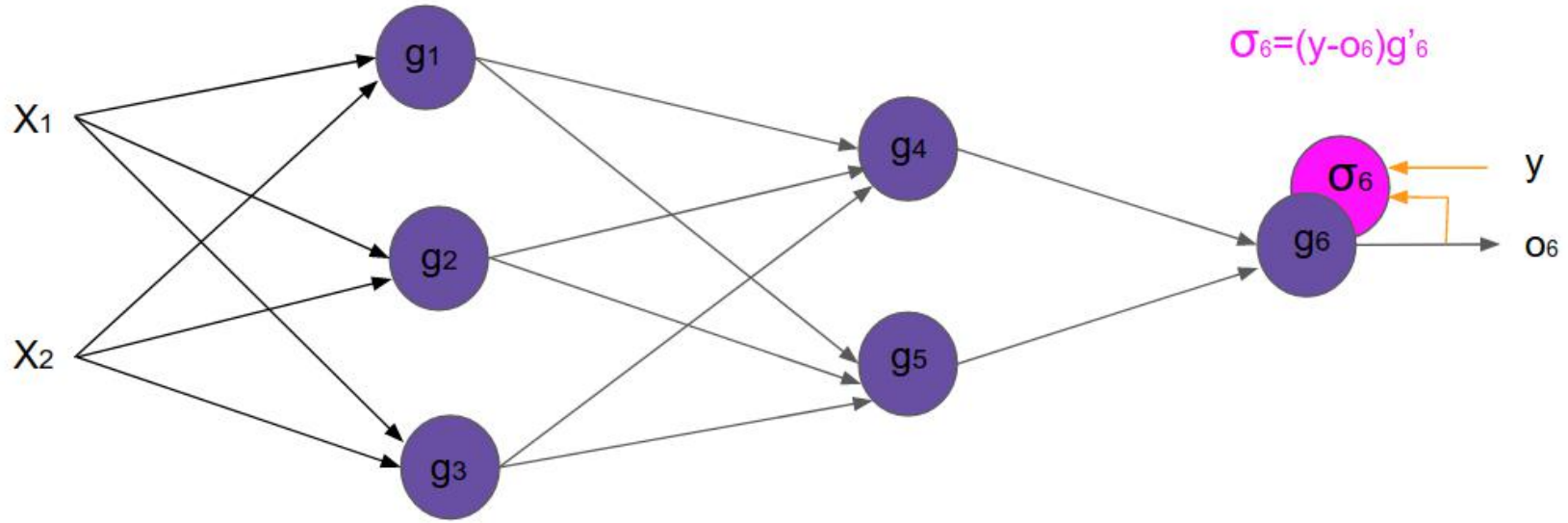
Learning Algorithm: Backpropagation

- Propagation of signals through the output layer.



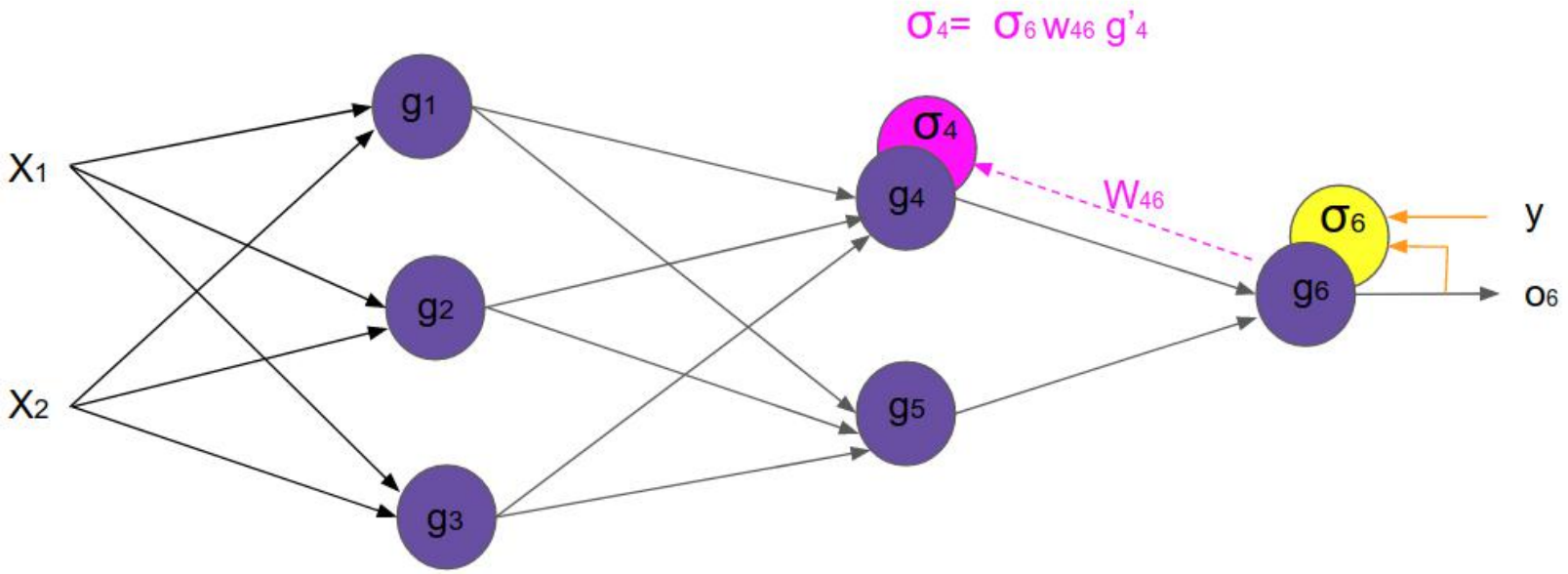
Learning Algorithm: Backpropagation

- In the next algorithm step the output signal of the network *y* is compared with the desired output value (the target), which is found in training data set. The difference is called error signal of output layer neuron



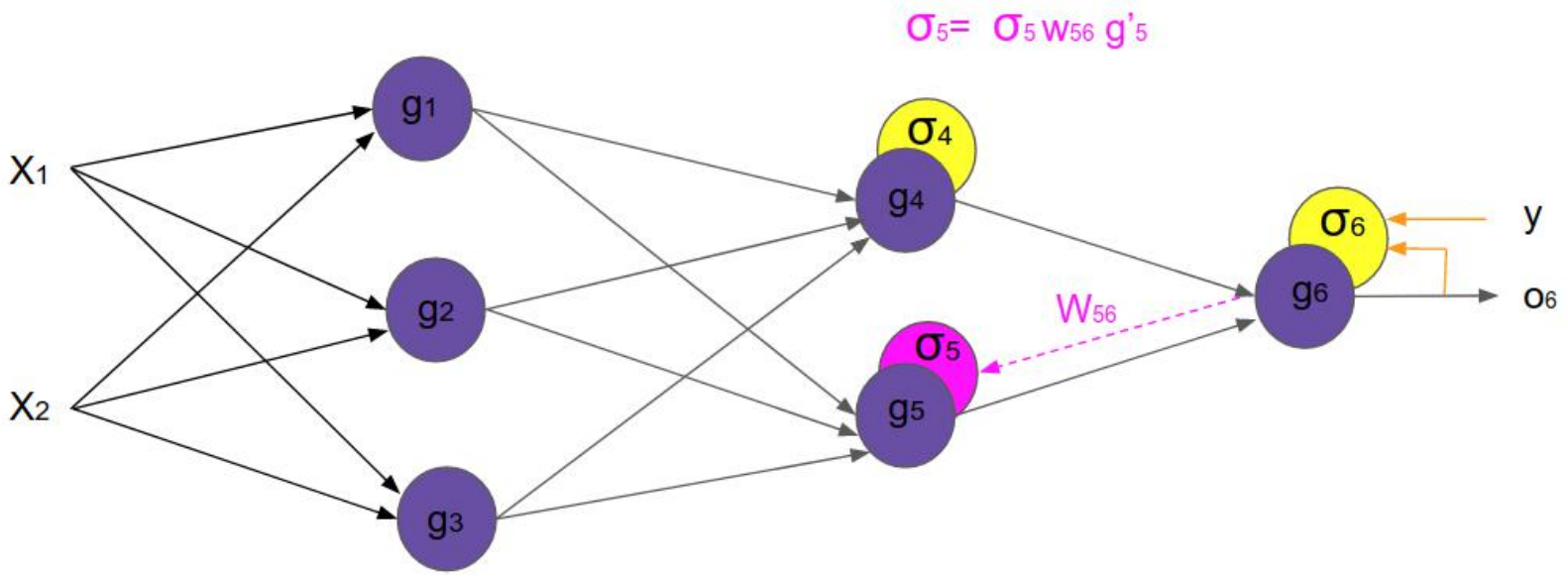
Learning Algorithm: Backpropagation

- The idea is to propagate error signal (computed in single step) back to all neurons, which output signals were input for discussed neuron.



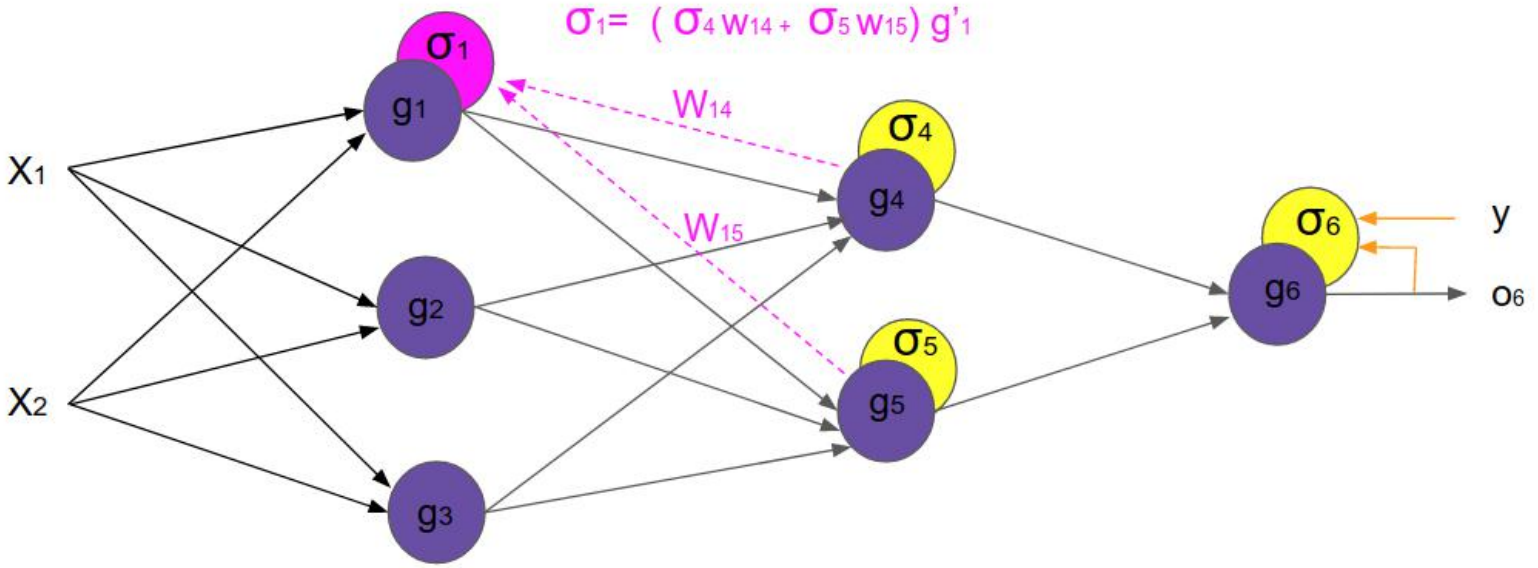
Learning Algorithm: Backpropagation

- The idea is to propagate error signal (computed in single step) back to all neurons, which output signals were input for discussed neuron.



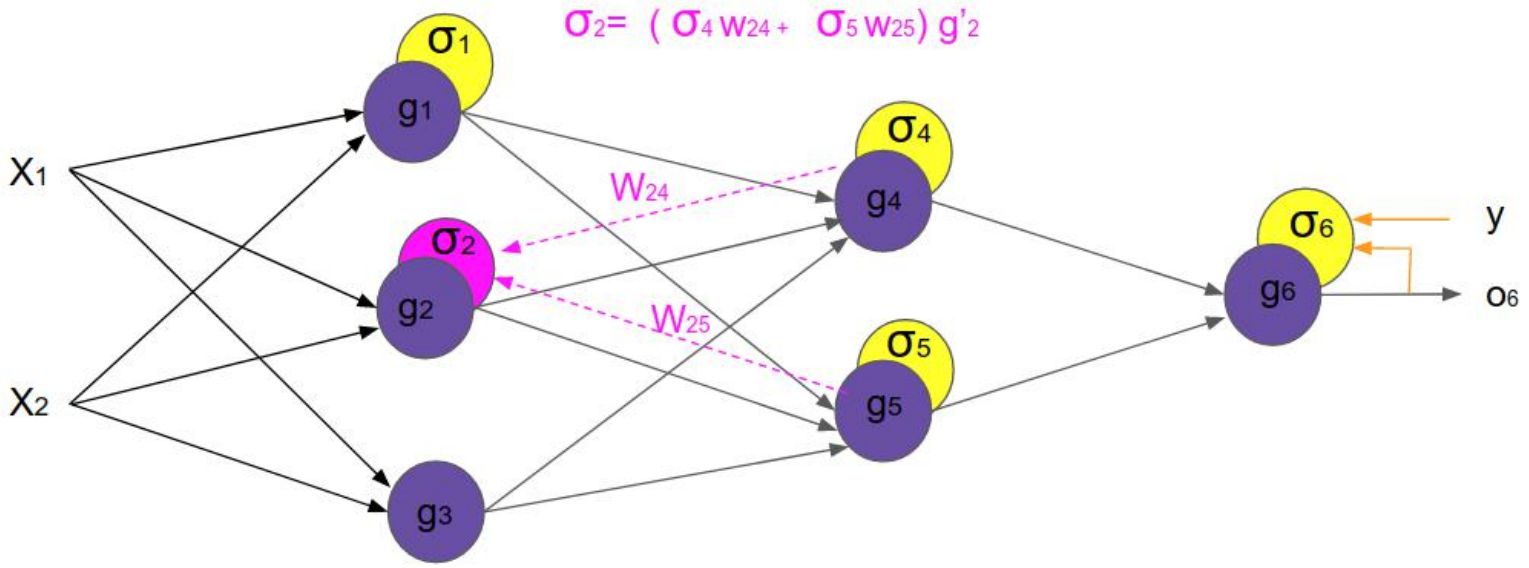
Learning Algorithm: Backpropagation

- The weights' coefficients w_{mn} used to propagate errors back are equal to this used during computing output value. Only the direction of data flow is changed (signals are propagated from output to inputs one after the other). This technique is used for all network layers. If propagated errors came from few neurons they are added. The illustration is below:



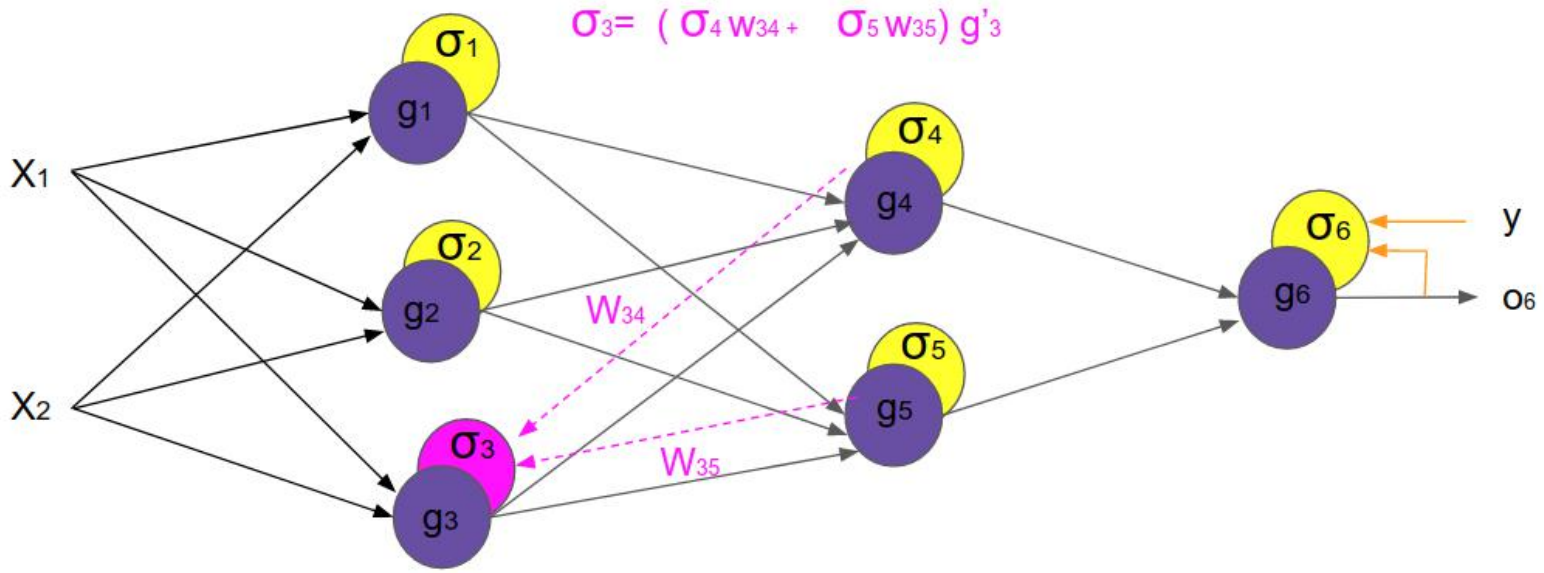
Learning Algorithm: Backpropagation

- The weights' coefficients w_{mn} used to propagate errors back are equal to this used during computing output value. Only the direction of data flow is changed (signals are propagated from output to inputs one after the other). This technique is used for all network layers. If propagated errors came from few neurons they are added. The illustration is below:



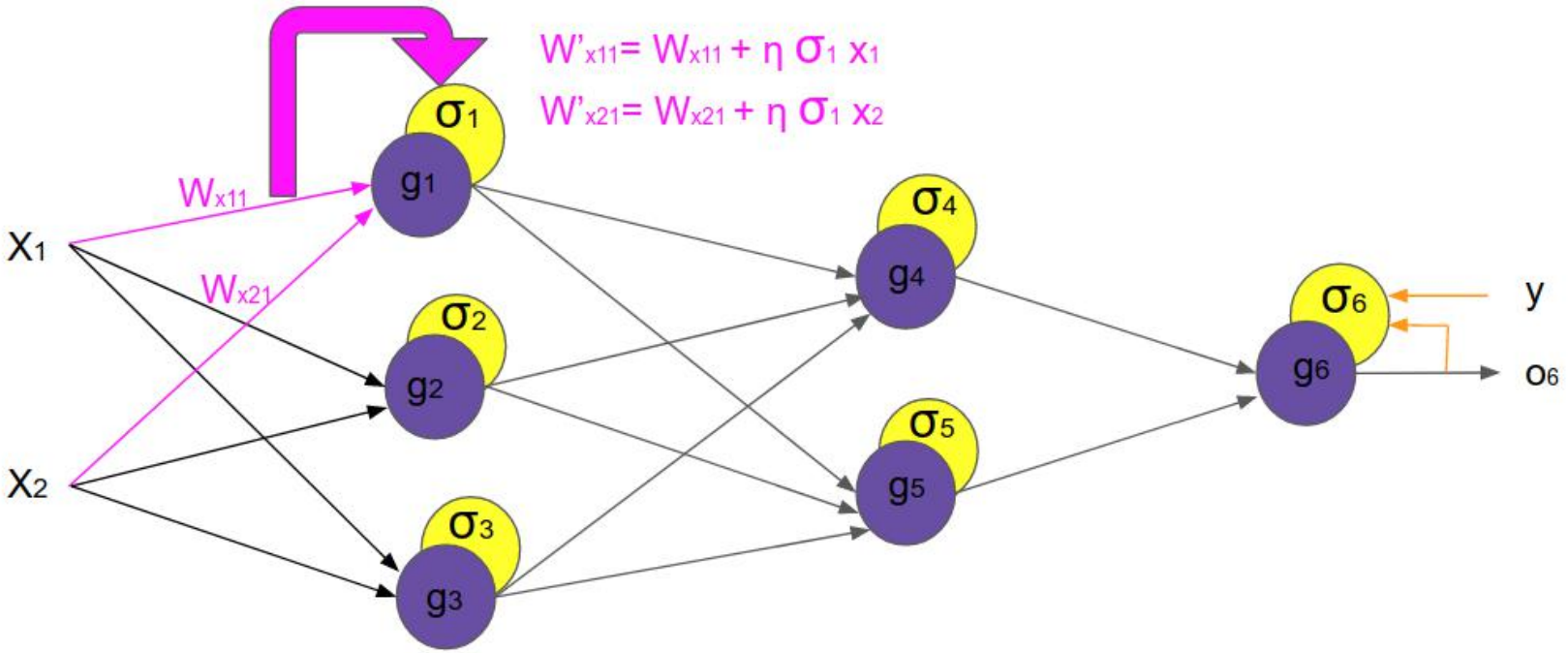
Learning Algorithm: Backpropagation

- The weights' coefficients w_{mn} used to propagate errors back are equal to this used during computing output value. Only the direction of data flow is changed (signals are propagated from output to inputs one after the other). This technique is used for all network layers. If propagated errors came from few neurons they are added. The illustration is below:



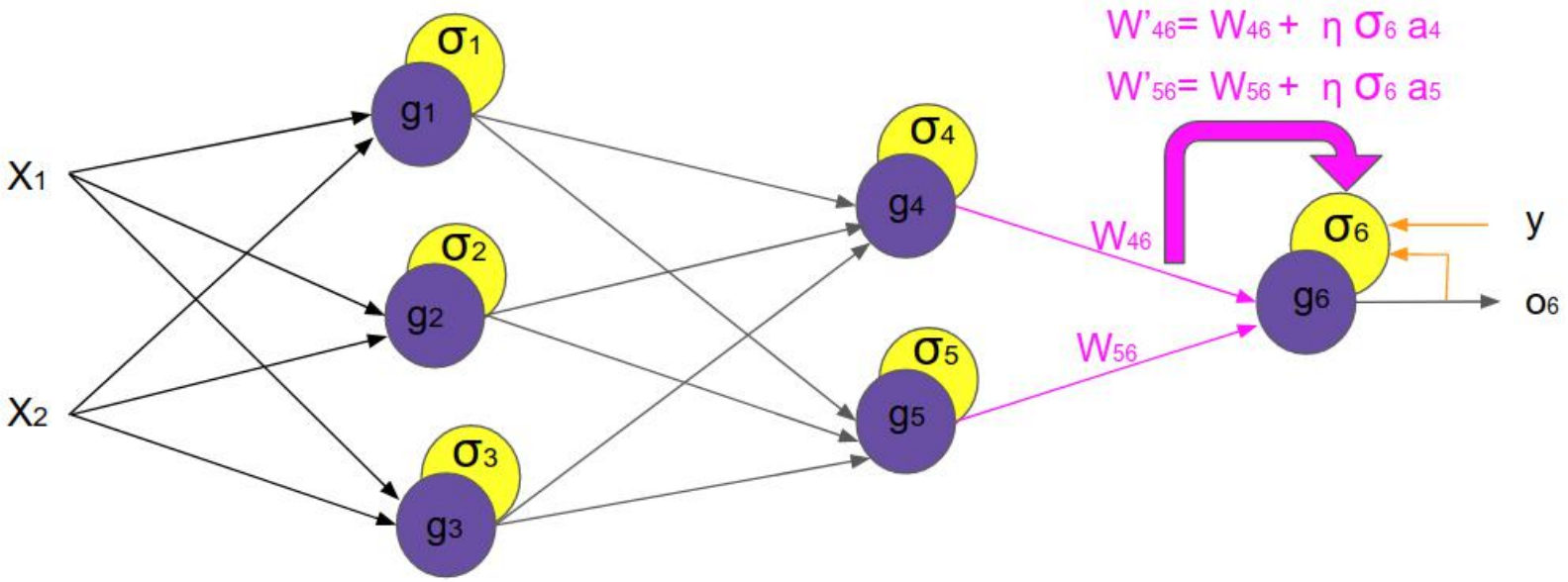
Learning Algorithm: Backpropagation

- When the error signal for each neuron is computed, the weights coefficients of each neuron input node may be modified.



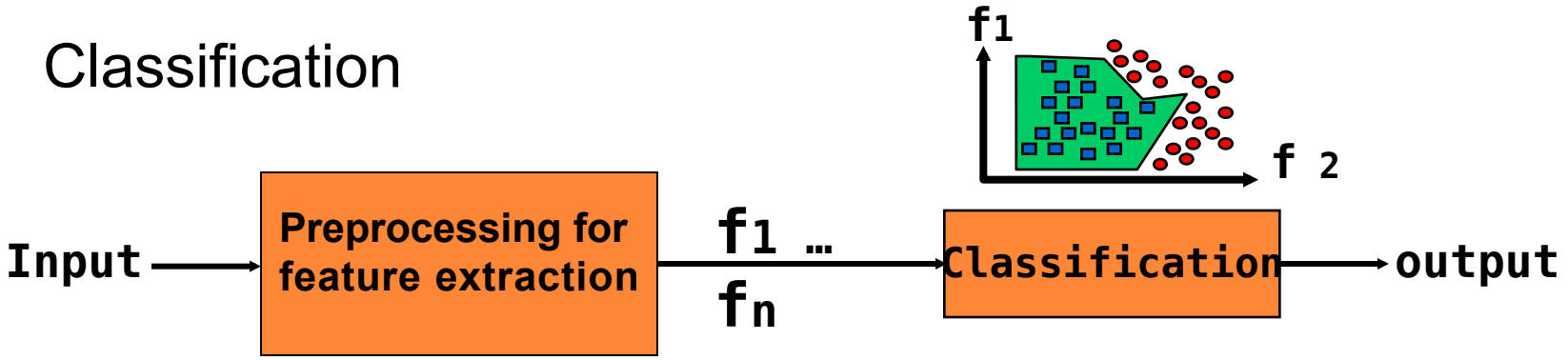
Learning Algorithm: Backpropagation

- When the error signal for each neuron is computed, the weights coefficients of each neuron input node may be modified.



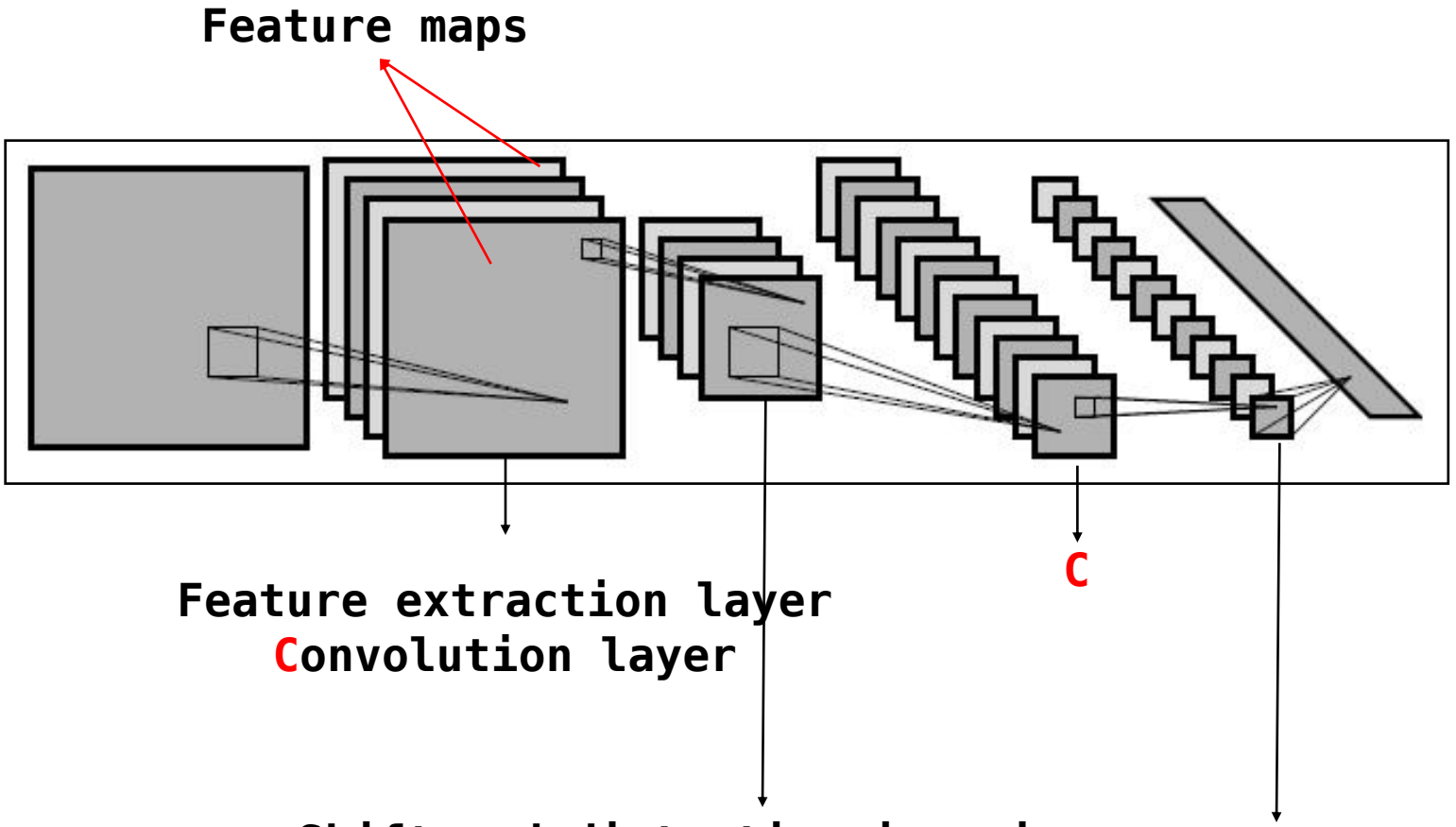
Classification

- Classification



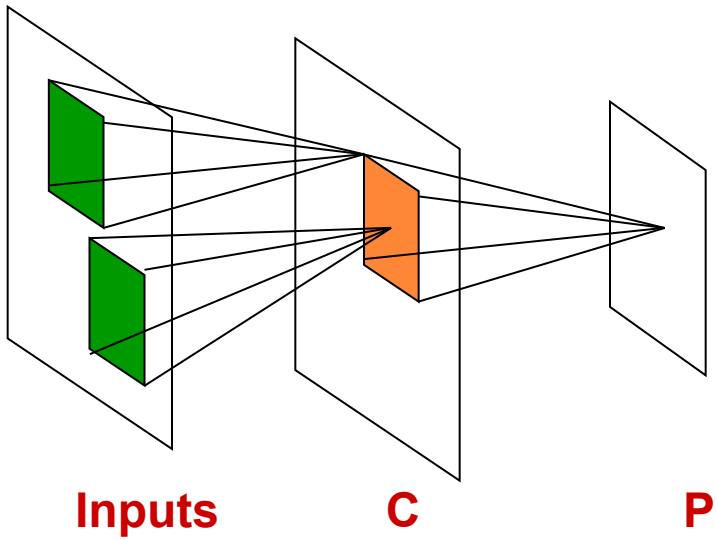
- Convolutional neural network

CNN's Topology

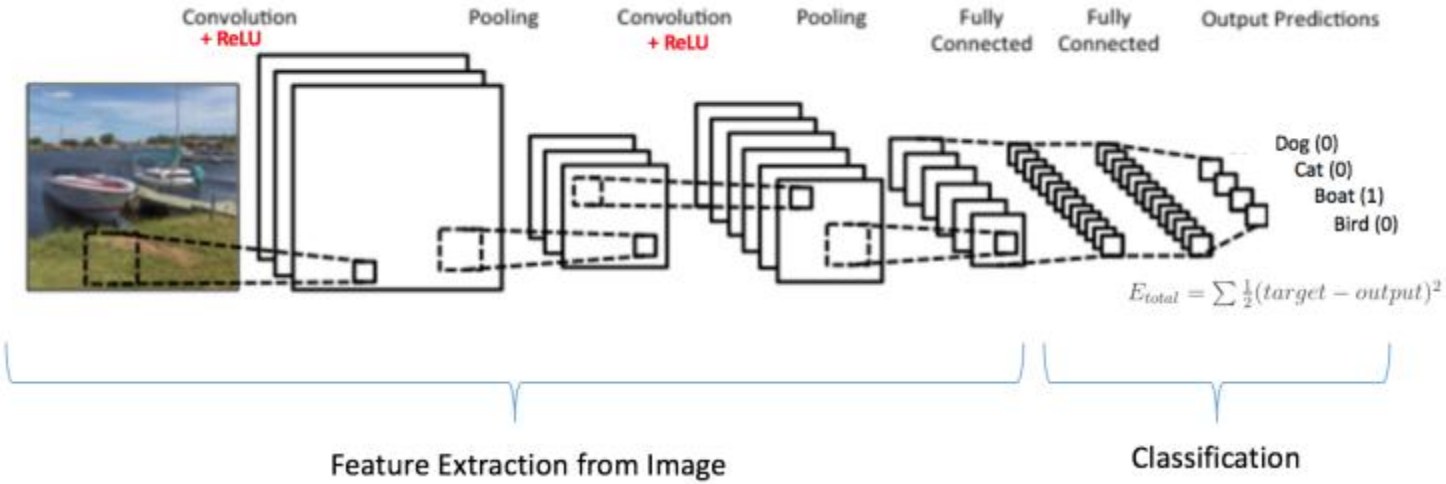


Feature extraction

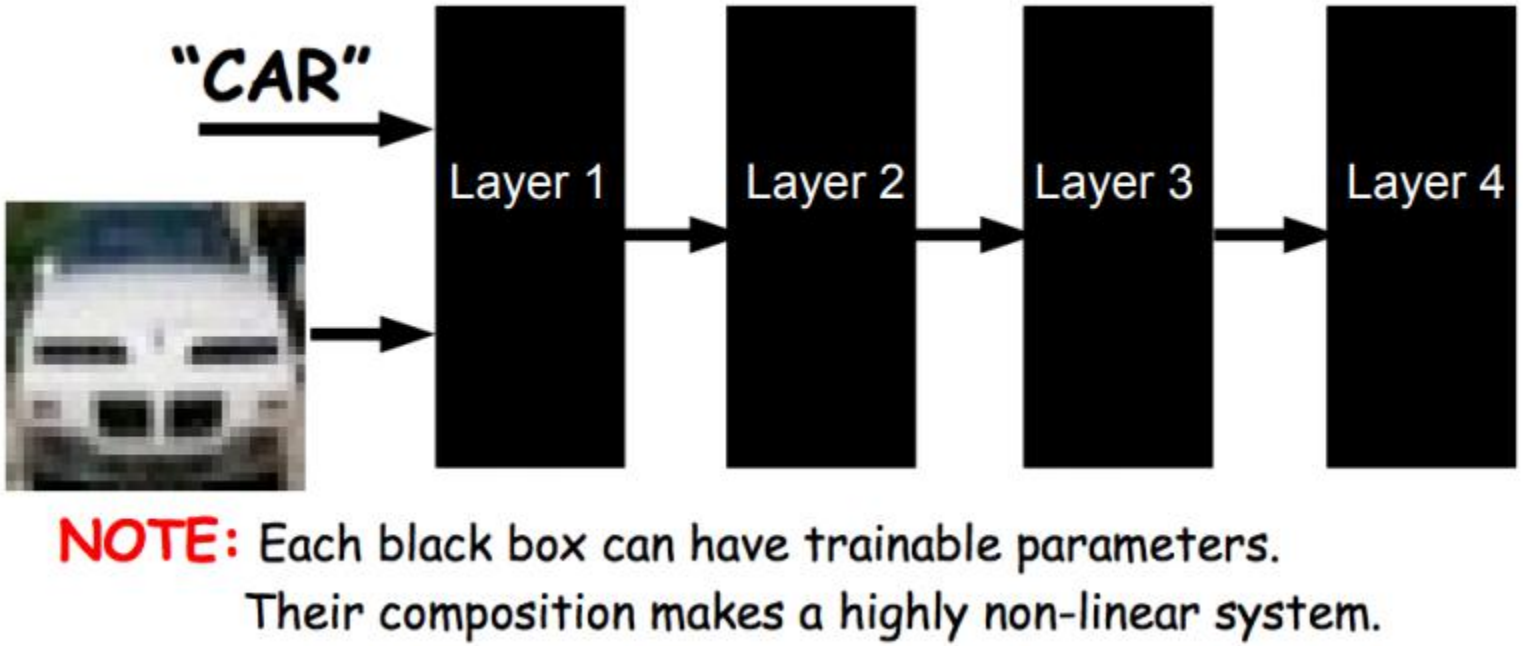
- Shared weights: all neurons in a feature share the same weights (but not the biases).
- In this way all neurons detect the same feature at different positions in the input image.
- Reduce the number of free parameters.



Putting it all together

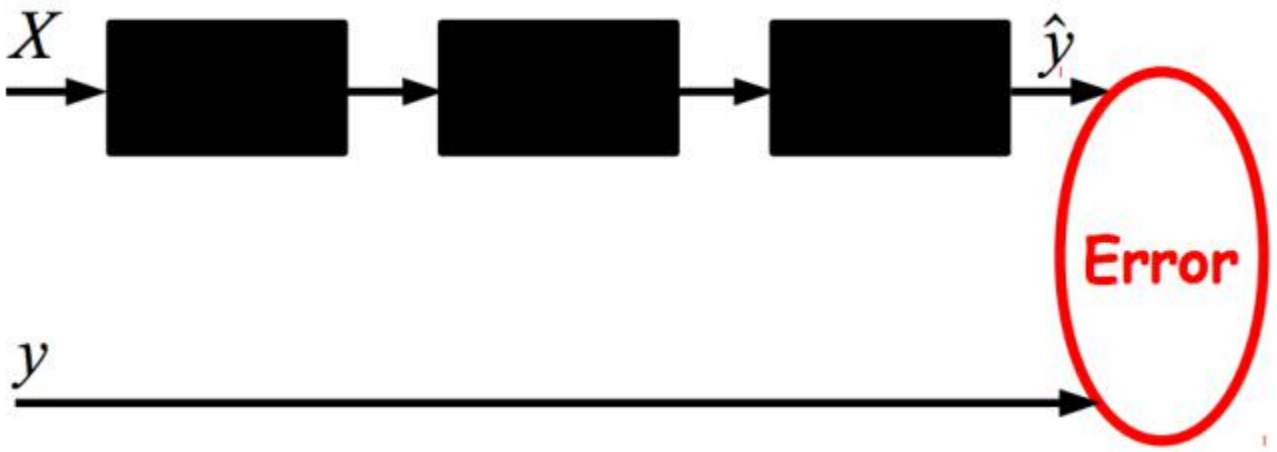


Intuition behind Deep Neural Nets



- The final layer outputs a probability distribution of categories.

Joint training architecture overview

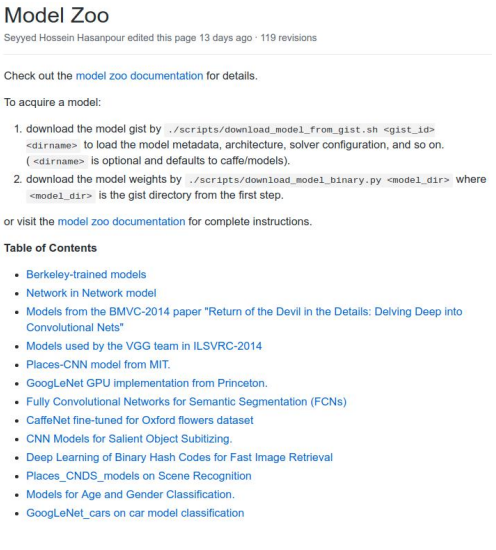


NOTE: Multi-layer neural nets with more than two layers are nowadays called **deep nets**!!

NOTE: User must specify number of layers, number of hidden units, type of layers and loss function.

Lots of pretrained ConvNets

- Caffe models: https://github.com/BVLC/caffe/wiki/Model-Zoo
- TensorFlow models: https://github.com/tensorflow/models/tree/master/research/slim
- PyTorch models:https://github.com/Cadene/pretrained-models.pytorch



Model Zoo

Seyyed Hossein Hasanpour edited this page 13 days ago · 119 revisions

Check out the model zoo documentation for details.

To acquire a model:

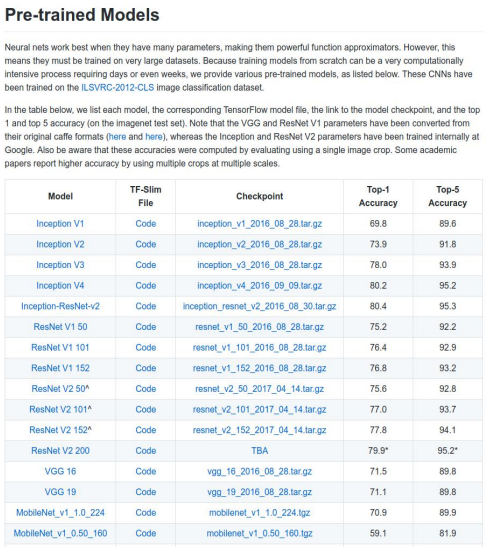
1. download the model gist by `./scripts/download_model_from_gist.sh <gist_id> <dirname>` to load the model metadata, architecture, solver configuration, and so on. (`<dirname>` is optional and defaults to caffe/models).
2. download the model weights by `./scripts/download_model_binary.py <model_dir>` where `<model_dir>` is the gist directory from the first step.

or visit the model zoo documentation for complete instructions.

Table of Contents

- Berkeley-trained models
- Network in Network model
- Models from the BMVC-2014 paper "Return of the Devil in the Details: Delving Deep into Convolutional Nets"
- Models used by the VGG team in ILSVRC-2014
- Places-CNN model from MIT.
- GoogLeNet GPU implementation from Princeton.
- Fully Convolutional Networks for Semantic Segmentation (FCNs)
- CaffeNet fine-tuned for Oxford flowers dataset
- CNN Models for Salient Object Subitizing.
- Deep Learning of Binary Hash Codes for Fast Image Retrieval
- Places_CNDS_models on Scene Recognition
- Models for Age and Gender Classification.
- GoogLeNet_cars on car model classification

Caffe



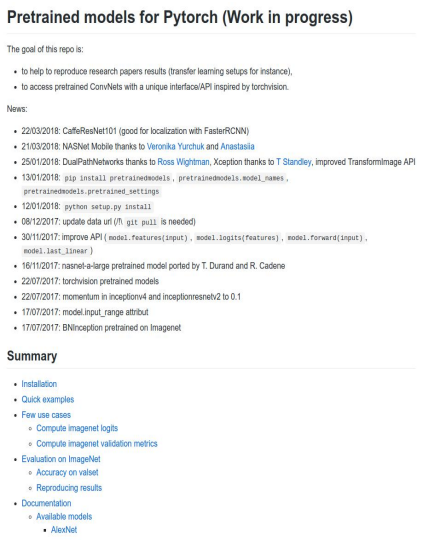
Pre-trained Models

Neural nets work best when they have many parameters, making them powerful function approximators. However, this means they must be trained on very large datasets. Because training models from scratch can be a very computationally intensive process requiring days or even weeks, we provide various pre-trained models, as listed below. These CNNs have been trained on the ILSVRC-2012-CLS image classification dataset.

In the table below, we list each model, the corresponding TensorFlow model file, the link to the model checkpoint, and the top 1 and top 5 accuracy (on the imagenet test set). Note that the VGG and ResNet V1 parameters have been converted from their original caffe formats (here and here), whereas the Inception and ResNet V2 parameters have been trained internally at Google. Also be aware that these accuracies were computed by evaluating using a single image crop. Some academic papers report higher accuracy by using multiple crops at multiple scales.

Model	TF-Slim File	Checkpoint	Top-1 Accuracy	Top-5 Accuracy
Inception V1	Code	inception_v1_2016_08_28.tar.gz	69.8	89.6
Inception V2	Code	inception_v2_2016_08_28.tar.gz	73.9	91.8
Inception V3	Code	inception_v3_2016_08_28.tar.gz	78.0	93.9
Inception V4	Code	inception_v4_2016_09_09.tar.gz	80.2	95.2
Inception-ResNet-v2	Code	inception_resnet_v2_2016_08_30.tar.gz	80.4	95.3
ResNet V1 50	Code	resnet_v1_50_2016_08_28.tar.gz	75.2	92.2
ResNet V1 101	Code	resnet_v1_101_2016_08_28.tar.gz	76.4	92.9
ResNet V1 152	Code	resnet_v1_152_2016_08_28.tar.gz	76.8	93.2
ResNet V2 50^	Code	resnet_v2_50_2017_04_14.tar.gz	75.6	92.8
ResNet V2 101^	Code	resnet_v2_101_2017_04_14.tar.gz	77.0	93.7
ResNet V2 152^	Code	resnet_v2_152_2017_04_14.tar.gz	77.8	94.1
ResNet V2 200	Code	TBA	79.9*	95.2*
VGG 16	Code	vgg_16_2016_08_28.tar.gz	71.5	89.8
VGG 19	Code	vgg_19_2016_08_28.tar.gz	71.1	89.8
MobileNet_v1_1.0_224	Code	mobilenet_v1_1.0_224.tgz	70.9	89.9
MobileNet_v1_0.50_160	Code	mobilenet_v1_0.50_160.tgz	59.1	81.9

TensorFlow



Pretrained models for Pytorch (Work in progress)

The goal of this repo is:

- to help to reproduce research papers results (transfer learning setups for instance),
- to access pretrained ConvNets with a unique interface/API inspired by torchvision.

News:

- 22/03/2018: CaffeResNet101 (good for localization with FasterRCNN)
- 21/03/2018: NASNet Mobile thanks to Veronika Yurchuk and Anastasiia
- 25/01/2018: DualPathNetworks thanks to Ross Wightman, Xception thanks to T Standley, improved TransformImage API
- 13/01/2018: `pip install pretrainedmodels`, `pretrainedmodels.model_names`, `pretrainedmodels.pretrained_settings`
- 12/01/2018: `python setup.py install`
- 08/12/2017: update data url (/!\ `git pull` is needed)
- 30/11/2017: improve API (`model.features(input)`, `model.logits(features)`, `model.forward(input)`, `model.last_linear`)
- 16/11/2017: nasnet-a-large pretrained model ported by T. Durand and R. Cadene
- 22/07/2017: torchvision pretrained models
- 22/07/2017: momentum in inceptionv4 and inceptionresnetv2 to 0.1
- 17/07/2017: model.input_range attribut
- 17/07/2017: BNInception pretrained on Imagenet

Summary

- Installation
- Quick examples
- Few use cases
  - Compute imagenet logits
  - Compute imagenet validation metrics
- Evaluation on ImageNet
  - Accuracy on valset
  - Reproducing results
- Documentation
  - Available models
    - AlexNet

PyTorch

- AlexNet
- BNInception
- CaffeResNet101
- DenseNet121
- DenseNet161
- DenseNet169
- DenseNet201
- DenseNet201
- DualPathNet68
- DualPathNet92
- DualPathNet98
- DualPathNet107
- DualPathNet113
- FBResNet152
- InceptionResNetV2
- InceptionV3
- InceptionV4
- NASNet-A-Large
- NASNet-A-Mobile
- ResNeXt101_32x4d
- ResNeXt101_64x4d
- ResNet101
- ResNet152
- ResNet18
- ResNet34
- ResNet50
- SqueezeNet1_0
- SqueezeNet1_1
- VGG11
- VGG13
- VGG16
- VGG19
- VGG11_BN
- VGG13_BN
- VGG16_BN
- VGG19_BN
- Xception

Disadvantages

- From a memory and capacity standpoint the CNN is not much bigger than a regular two layer network.
- At runtime the convolution operations are computationally expensive and take up about 67% of the time.
- CNN's are about $3X$ slower than their fully connected equivalents (size wise).

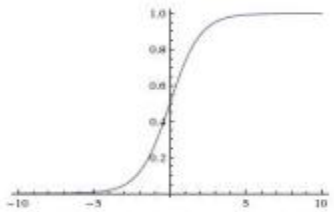
Disadvantages

- Convolution operation
  - 4 nested loops (2 loops on input image & 2 loops on kernel)
- Small kernel size
  - make the inner loops very inefficient as they frequently JMP.
- Cash unfriendly memory access
  - Back-propagation require both row-wise and column-wise access to the input and kernel image.
  - 2D Images represented in a row-wise/ serialized order.
  - Column-wise access to data can result in a high rate of cash misses in memory subsystem.

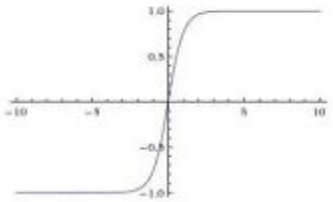
Activation Functions

Sigmoid

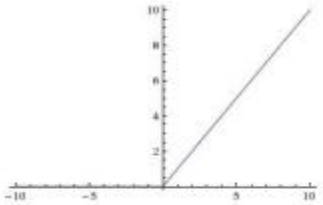
$\sigma(x) = 1/(1+e^{-x})$



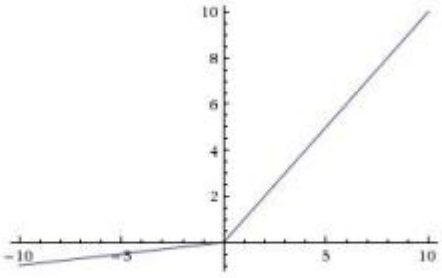
tanh tanh(x)



ReLU max(0,x)

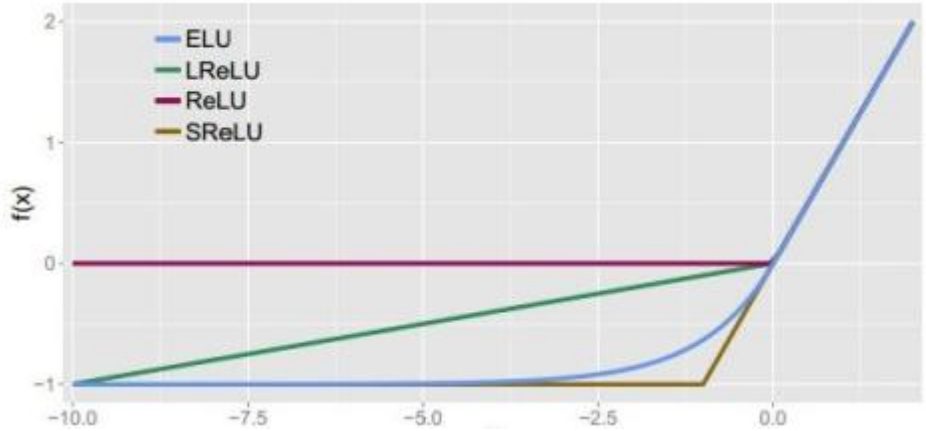


Leaky ReLU
max(0.1x, x)



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

ELU $f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \le 0 \end{cases}$



SReLU (Shift Rectified Linear Unit)

max(-1, x)

In practice

- Use ReLU. Be careful with your learning rates
- Try out Leaky ReLU / Maxout / ELU
- Try out tanh but don't expect much
- Don't use sigmoid

Mini-batch SGD

- Loop:

1. Sample a batch of data
2. Forward prop it through the graph, get loss
3. Backprop to calculate the gradients
4. Update the parameters using the gradient

This method takes the best of both batch and SGD, and performs an update for every mini-batch of n.

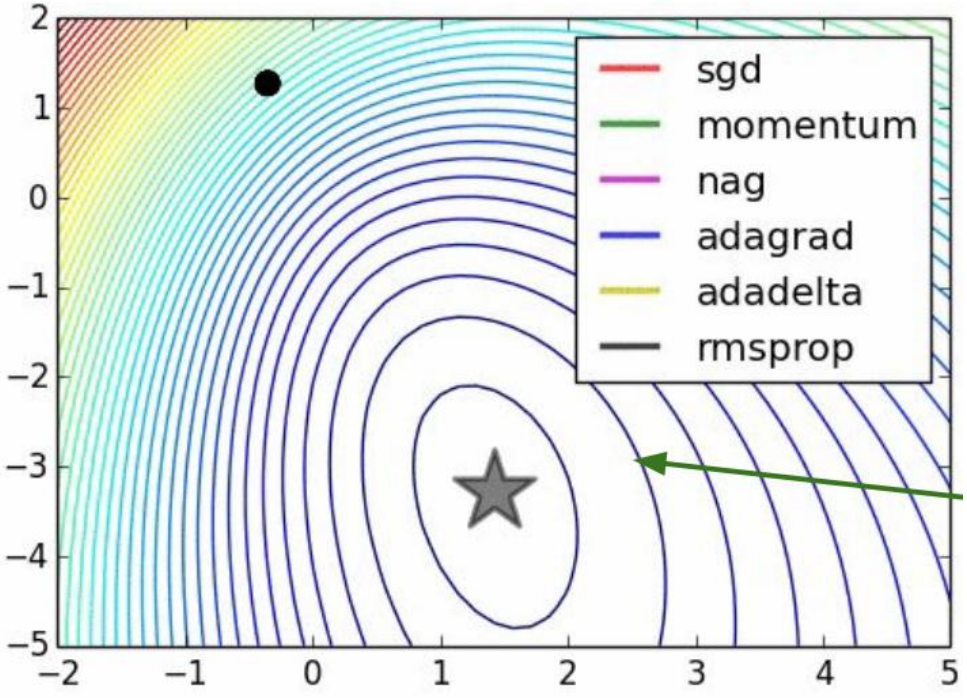
Update equation

$$\theta = \theta - \eta * \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

Code

```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for batch in get_batches(data, batch_size=50):
    params_grad = evaluate_gradient(loss_function, batch, params)
    params = params - learning_rate * params_grad
```

Overview of gradient descent optimization algorithms



- Gradient descent optimization algorithms
  - Momentum
  - Nesterov accelerated gradient
  - Adagrad
  - Adadelta
  - RMSprop
  - Adam
  - AdaMax
  - Nadam
  - AMSGrad
  - Visualization of algorithms
  - Which optimizer to choose?

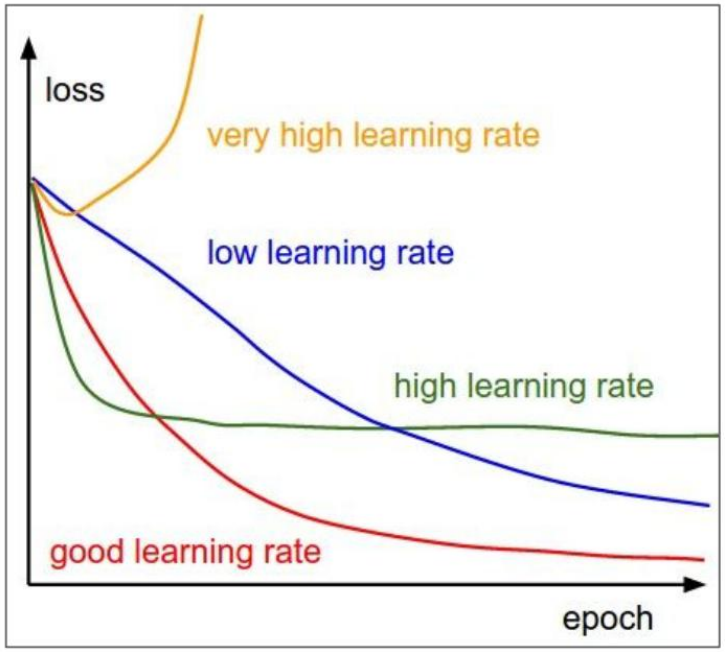
Link: http://ruder.io/optimizing-gradient-descent/

Which Optimizer to Use?

- If your input data is sparse, then you likely achieve the best results using one of the adaptive learning-rate methods.
- RMSprop is an extension of Adagrad that deals with its radically diminishing learning rates. Adam, finally, adds bias-correction and momentum to RMSprop. Insofar, RMSprop, Adadelta, and Adam are very similar algorithms that do well in similar circumstances.
- Experiments show that bias-correction helps Adam slightly outperform RMSprop towards the end of optimization as gradients become sparser. Insofar, Adam might be the best overall choice.
- Interestingly, many recent papers use SGD without momentum and a simple learning rate annealing schedule. As has been shown, SGD usually achieves to find a minimum, but it might take significantly longer than with some of the optimizers, is much more reliant on a robust initialization and annealing schedule, and may get stuck in saddle points rather than local minima.
- If you care about fast convergence and train a deep or complex neural network, you should choose one of the adaptive learning rate methods

Learning rate

- SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have learning rate as a hyperparameter.



=> Learning rate decay over time!

step decay:
e.g. decay learning rate by half every few epochs.

exponential decay:

$$\alpha = \alpha_0 e^{-kt}$$

1/t decay:

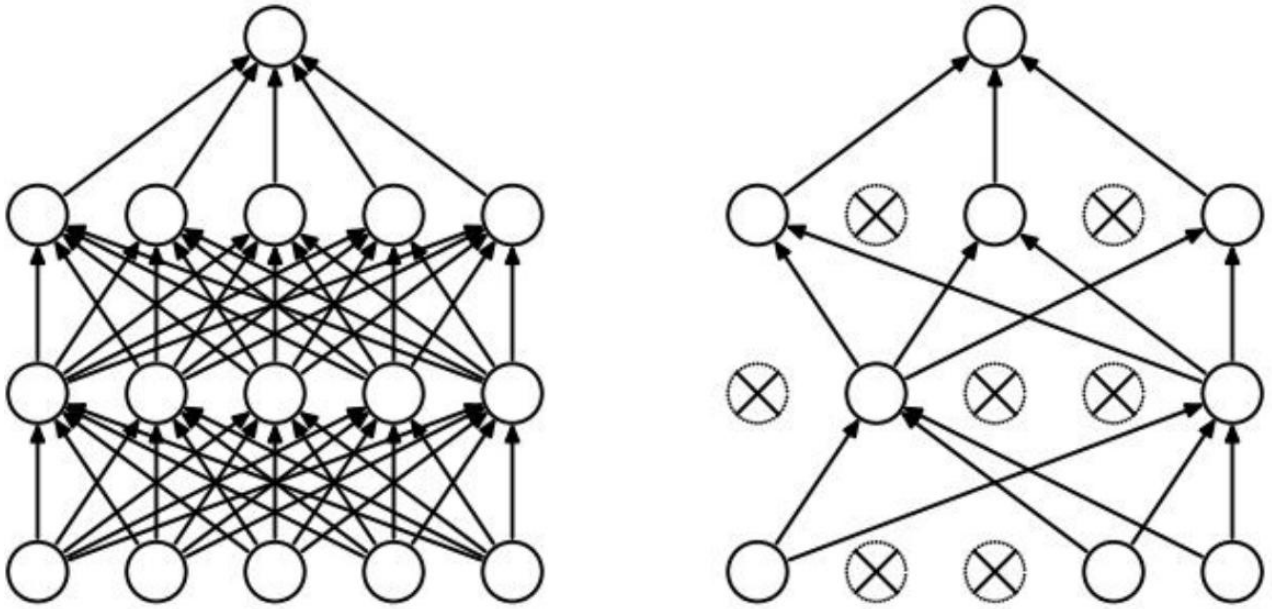
$$\alpha = \alpha_0 / (1 + kt)$$

L-BFGS

- **Usually works very well in full batch, deterministic mode.**
  - i.e. if you have a single, deterministic f(x) then L-BFGS will probably work very nicely
- **Does not transfer very well to mini-batch setting**.
  - Gives bad results. Adapting L-BFGS to large-scale, stochastic setting is an active area of research
- **In practice**:
  - Adam is a good default choice in most cases
  - If you can afford to do full batch updates then try out L-BFGS (and don't forget to disable all sources of noise)

Regularization: Dropout

- "randomly set some neurons to zero in the forward pass"



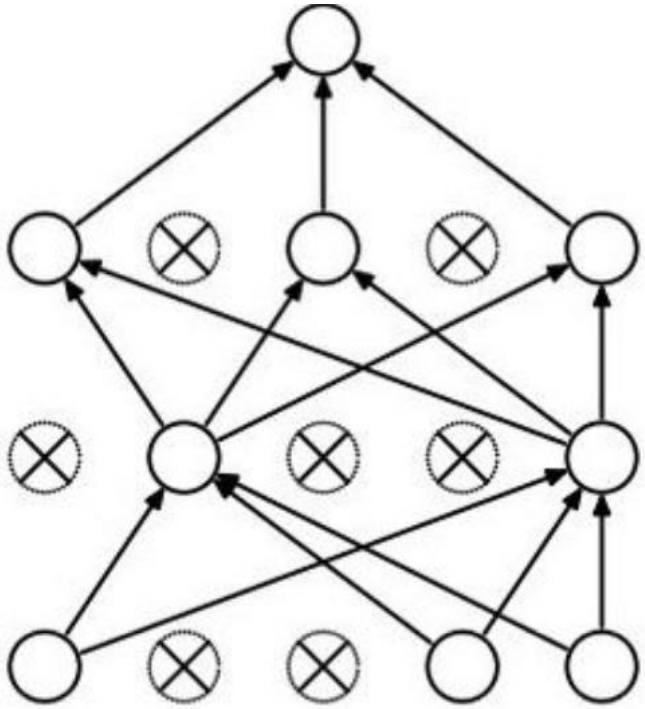
(a) Standard Neural Net

(b) After applying dropout.

[Srivastava et al., 2014]

Regularization: Dropout

- Wait a second… How could this possibly be a good idea?

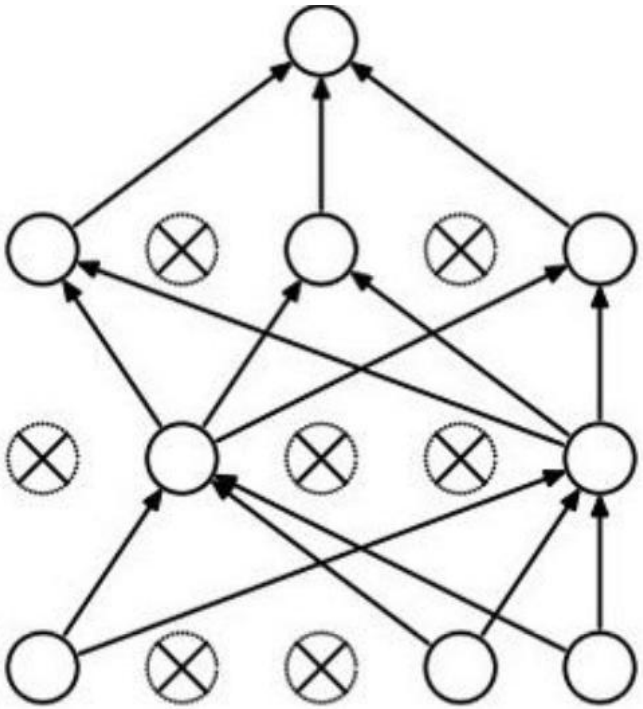


Another interpretation:

Dropout is training a large ensemble of models (that share parameters).

Each binary mask is one model, gets trained on only ~one datapoint

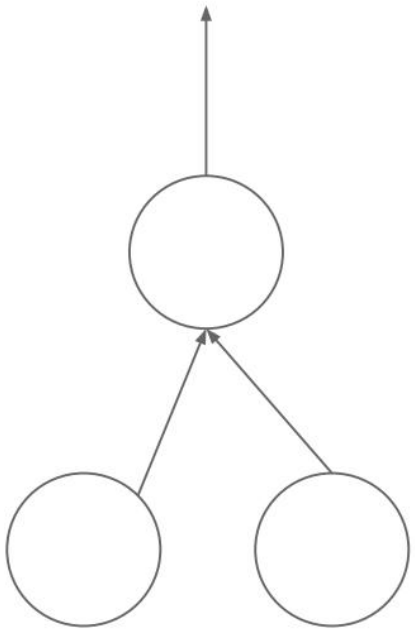
At test time….



- **Ideally**:
- want to integrate out all the noise
- **Monte Carlo approximation**:
- do many forward passes with different dropout masks, average all predictions

At test time….

- Can in fact do this with a single forward pass! (approximately)
- Leave all input neurons turned on (no dropout).

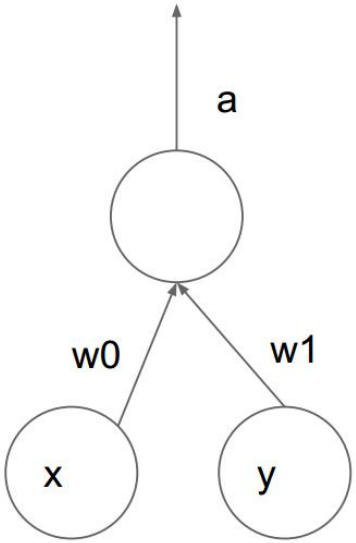


Q: Suppose that with all inputs present at test time the output of this neuron is x.

What would its output be during training time, in expectation? (e.g. if p = 0.5)

At test time….

- Can in fact do this with a single forward pass! (approximately)
- Leave all input neurons turned on (no dropout).



during test: **a = w0*x + w1*y**

during train:

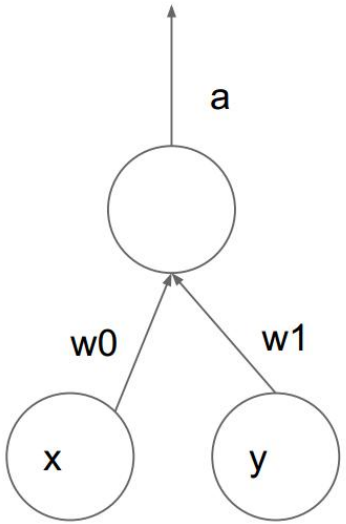
E[a] = ¼ * (w0*0 + w1*0
w0*0 + w1*y
w0*x + w1*0
w0*x + w1*y)

= ¼ * (2 w0*x + 2 w1*y)

= ½ * **(w0*x + w1*y)**

At test time….

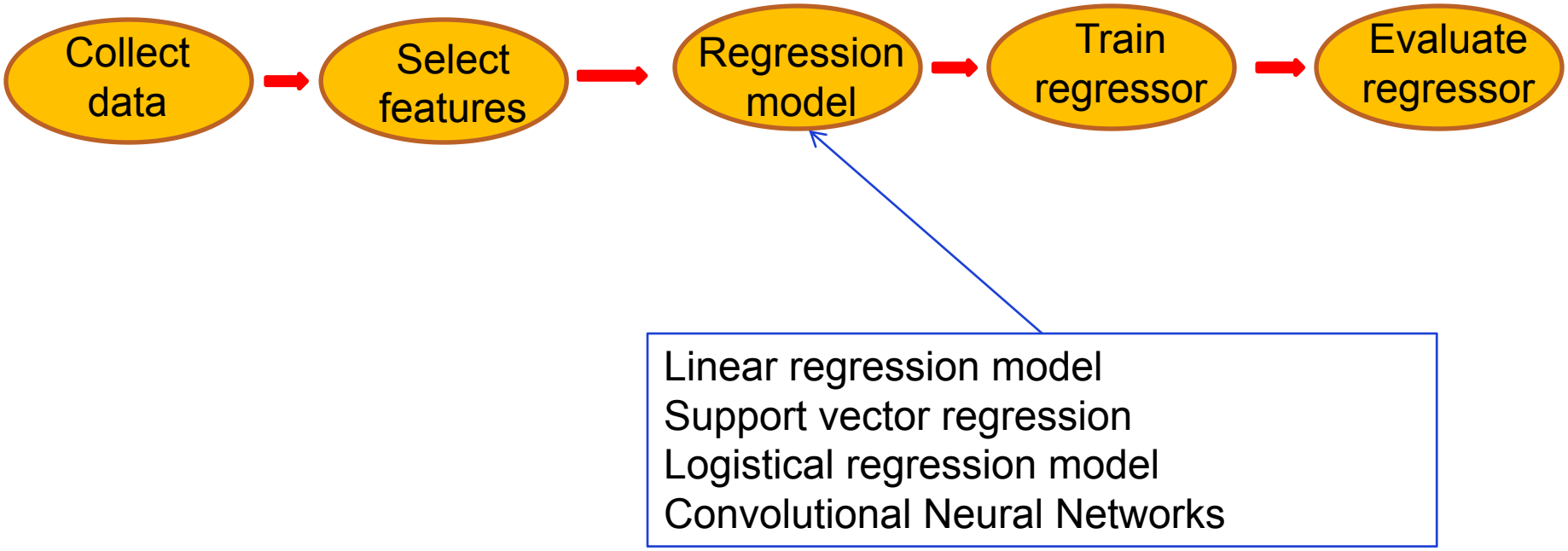
- Can in fact do this with a single forward pass! (approximately)
- Leave all input neurons turned on (no dropout).



during test: **a = w0*x + w1*y**
during train:
E[a] = ¼ * (w0*0 + w1*0
w0*0 + w1*y
w0*x + w1*0
w0*x + w1*y)
= ¼ * (2 w0*x + 2 w1*y)
= ½ * **(w0*x + w1*y)**

With p=0.5, using all inputs in the forward pass would inflate the activations by 2x from what the network was "used to" during training!
=> Have to compensate by scaling the activations back down by ½

Pattern recognition design cycle



Linear Regression

- Given data with n dimensional variables and 1 target-variable (real number)

$$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_m, y_m)\}$$

where $\mathbf{x} \in \Re^n, y \in \Re$

- The objective: Find a function f that returns the best fit.

$$f : \Re^n \rightarrow \Re$$

- To find the best fit, we minimize the sum of squared errors –> Least square estimation

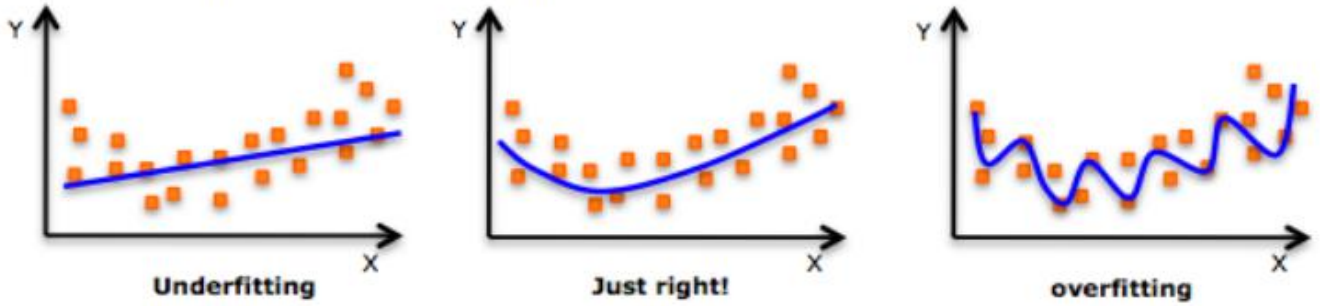
$$\min \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{m} (y_i - (\mathbf{w} \cdot \mathbf{x}_i + b))^2$$

- The solution can be found by solving

$$\hat{\mathbf{w}} = (X^T X)^{-1} X^T Y$$

Linear Regression

$$\min \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{m} (y_i - (\hat{\mathbf{w}} \cdot \mathbf{x}_i + \hat{b}))^2$$



To avoid over-fitting, a regularization term can be introduced (minimize a magnitude of w)

– LASSO: $\min \sum_{i=1}^{m} (y_i - \mathbf{w} \cdot \mathbf{x}_i - b)^2 + C \sum_{j=1}^{n} | w_j |$

– Ridge regression: $\min \sum_{i=1}^{m} (y_i - \mathbf{w} \cdot \mathbf{x}_i - b)^2 + C \sum_{j=1}^{n} | \mathbf{w}_j^2 |$

Support Vector Regression

- Find a function, f(x), with at most ε –deviation from the target y

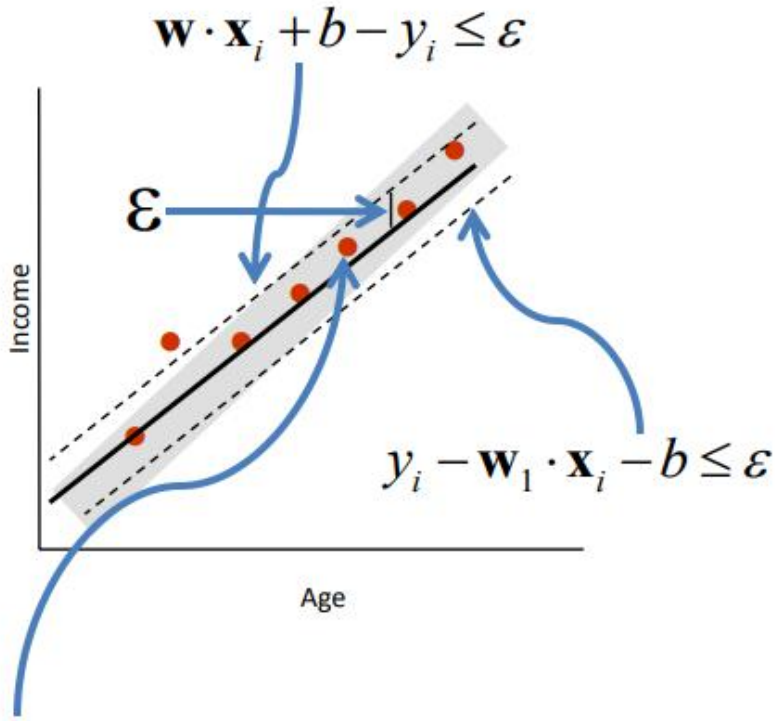
The problem can be written as a convex optimization problem

$$\min \frac{1}{2} \| \mathbf{w} \|^2$$

$$s.t.\ y_i - \mathbf{w}_1 \cdot \mathbf{x}_i - b \le \varepsilon;$$

$$\mathbf{w}_1 \cdot \mathbf{x}_i + b - y_i \le \varepsilon;$$

C: trade off the complexity



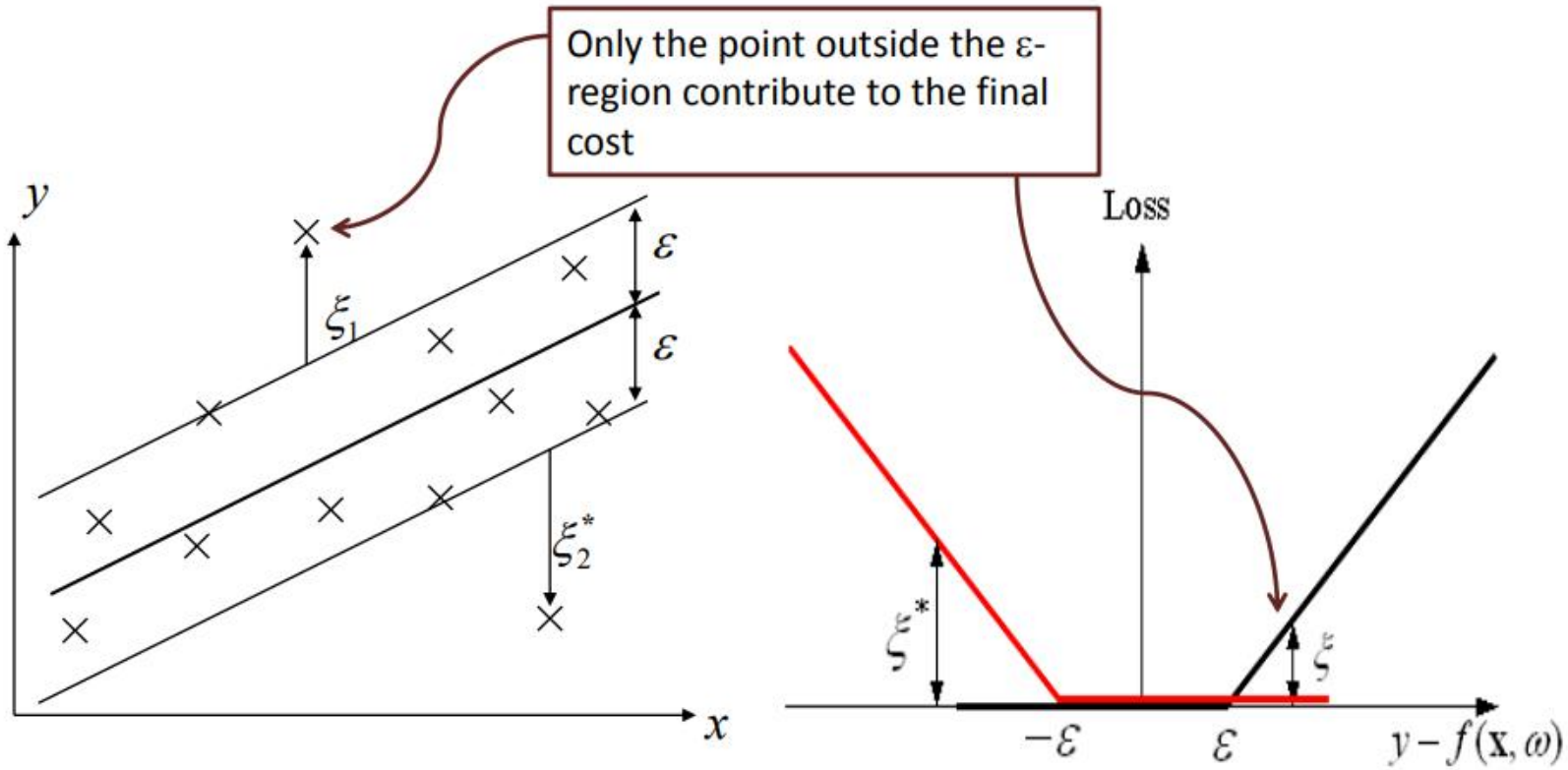
We do not care about errors as long as they are less than ε

What if the problem is not feasible?

We can introduce slack variables (similar to soft margin loss function).

Support Vector Regression

Assume linear parameterization $f(\mathbf{x}, \omega) = \mathbf{w} \cdot \mathbf{x} + b$

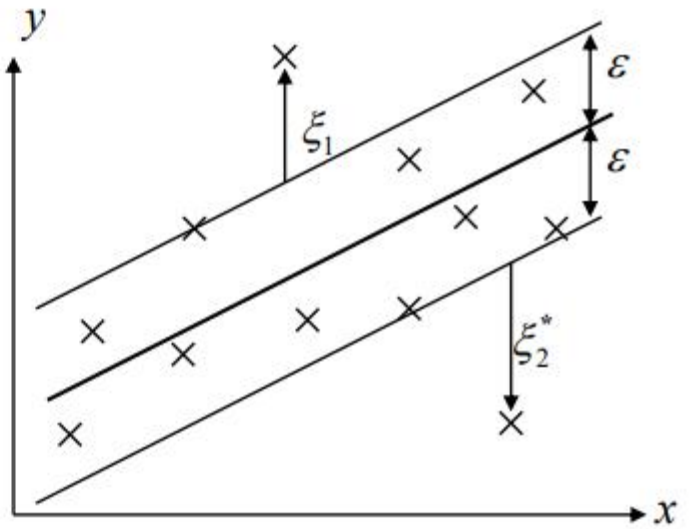


$$L_\varepsilon(y, f(\mathbf{x}, \omega)) = \max(|\, y - f(\mathbf{x}, \omega)\,| - \varepsilon, 0)$$

Soft margin

Given training data

$$(\mathbf{x}_i, y_i) \quad i = 1, \ldots, m$$



Minimize

$$\frac{1}{2} \| \mathbf{w} \|^2 + C \sum_{i=1}^{m} (\xi_i + \xi_i^*)$$

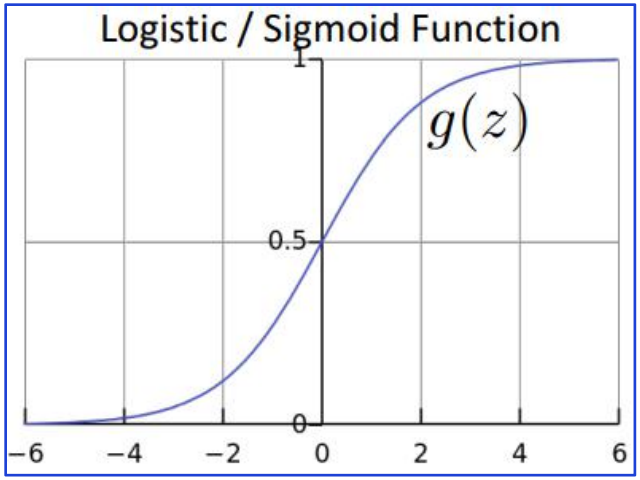
Under constraints

$$\begin{cases} y_i - (\mathbf{w} \cdot \mathbf{x}_i) - b \le \varepsilon + \xi_i \\ (\mathbf{w} \cdot \mathbf{x}_i) + b - y_i \le \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* \ge 0, i = 1, \ldots, m \end{cases}$$

Logistic Regression

Given $\left\{ \left(\boldsymbol{x}^{(1)}, y^{(1)} \right), \left(\boldsymbol{x}^{(2)}, y^{(2)} \right), \ldots, \left(\boldsymbol{x}^{(n)}, y^{(n)} \right) \right\}$

where $\boldsymbol{x}^{(i)} \in \mathbb{R}^d, \; y^{(i)} \in \{0, 1\}$



Model: $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = g\left(\boldsymbol{\theta}^{\mathsf{T}} \boldsymbol{x}\right)$

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

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} \qquad \boldsymbol{x}^{\mathsf{T}} = \begin{bmatrix} 1 & x_1 & \ldots & x_d \end{bmatrix}$$

Logistic Regression Objective Function

- Can't just use squared loss as in linear regression

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2$$

  – Using the logistic regression model

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{x}}}$$

results in a non-convex optimization

Deriving the Cost Function via Maximum Likelihood Estimation

- Likelihood of data is given by: $l(\boldsymbol{\theta}) = \prod_{i=1}^{n} p(y^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta})$
- So, looking for the $\boldsymbol{\theta}$ that maximizes the likelihood

$$\boldsymbol{\theta}_{\text{MLE}} = \arg\max_{\boldsymbol{\theta}} l(\boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{n} p(y^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

- Can take the log without changing the solution:

$$\boldsymbol{\theta}_{\text{MLE}} = \arg\max_{\boldsymbol{\theta}} \log \prod_{i=1}^{n} p(y^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

$$= \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{n} \log p(y^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

Deriving the Cost Function via Maximum Likelihood Estimation

- Expand as follows:

$$\boldsymbol{\theta}_{\text{MLE}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{n} \log p(y^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

$$= \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{n} \left[y^{(i)} \log p(y^{(i)} = 1 \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta}) + \left(1 - y^{(i)}\right) \log \left(1 - p(y^{(i)} = 1 \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta})\right) \right]$$

- Substitute in model, and take negative to yield

Logistic regression objective:

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

$$J(\boldsymbol{\theta}) = -\sum_{i=1}^{n} \left[y^{(i)} \log h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) + \left(1 - y^{(i)}\right) \log \left(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})\right) \right]$$

Regularized Logistic Regression

$$J(\boldsymbol{\theta}) = -\sum_{i=1}^{n}\left[y^{(i)}\log h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) + \left(1-y^{(i)}\right)\log\left(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})\right)\right]$$

- We can regularize logistic regression exactly as before

$$\begin{aligned} J_{\text{regularized}}(\boldsymbol{\theta}) &= J(\boldsymbol{\theta}) + \lambda\sum_{j=1}^{d}\theta_j^2 \\ &= J(\boldsymbol{\theta}) + \lambda\|\boldsymbol{\theta}_{[1:d]}\|_2^2 \end{aligned}$$

Another Interpretation

- Equivalently, logistic regression assumes that

$$\log \frac{p(y=1 \mid \boldsymbol{x}; \boldsymbol{\theta})}{p(y=0 \mid \boldsymbol{x}; \boldsymbol{\theta})} = \theta_0 + \theta_1 x_1 + \ldots + \theta_d x_d$$

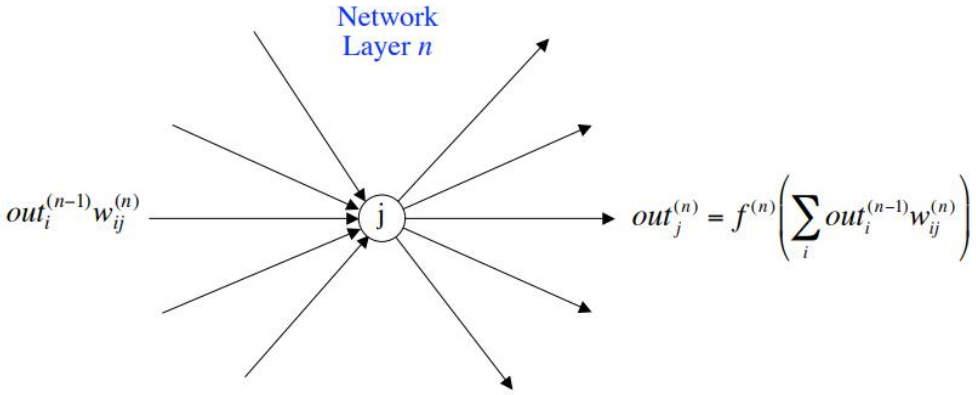
odds of y = 1

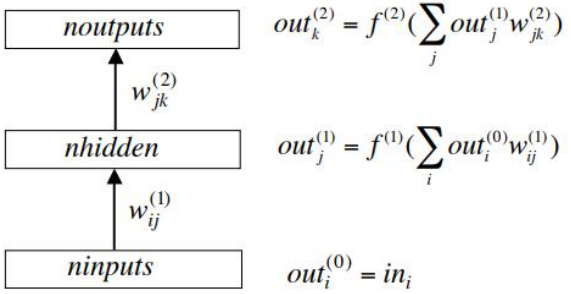
Side Note: the odds in favor of an event is the quantity $p / (1 - p)$, where p is the probability of the event

E.g., If I toss a fair dice, what are the odds that I will have a 6?

- In other words, logistic regression assumes that the log odds is a linear function of **x**

DNN Regression





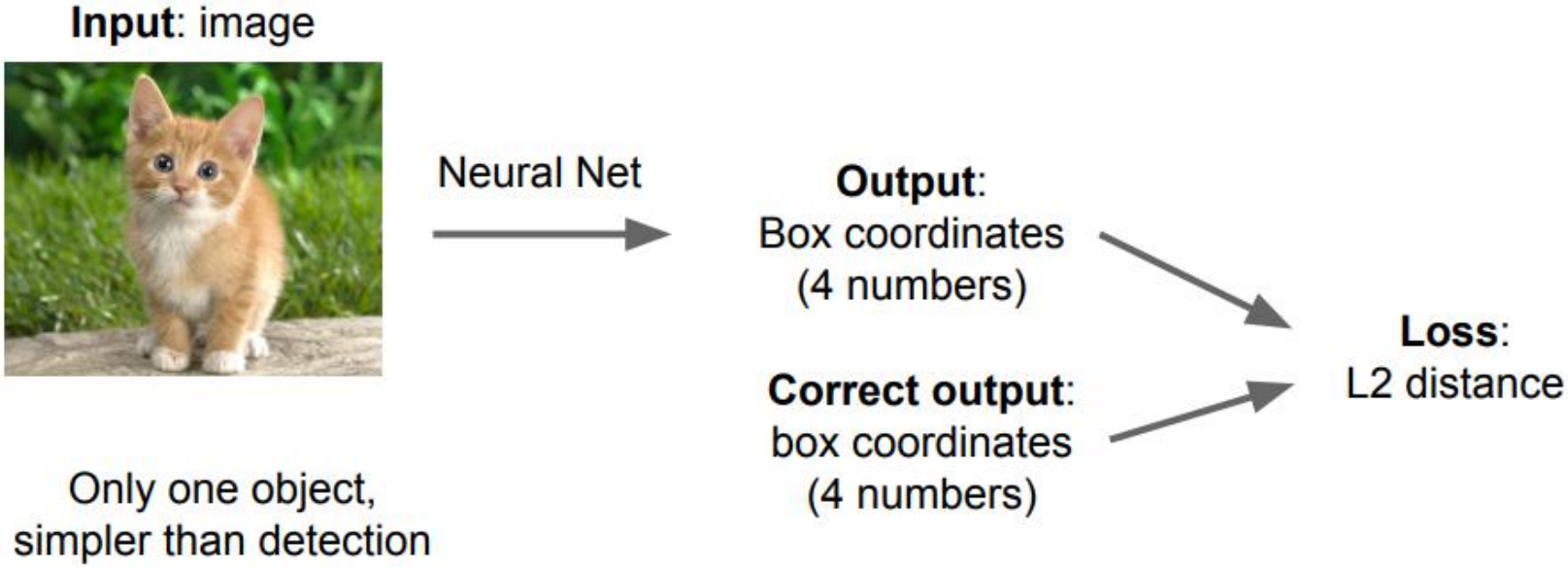
- For a two-layer MLP:

$$out_k^{(2)} = f^{(2)}\left(\sum_j out_j^{(1)}.w_{jk}^{(2)}\right) = f^{(2)}\left(\sum_j f^{(1)}\left(\sum_i in_i w_{ij}^{(1)}\right).w_{jk}^{(2)}\right)$$

- The network weights are adjusted to minimize an output cost function

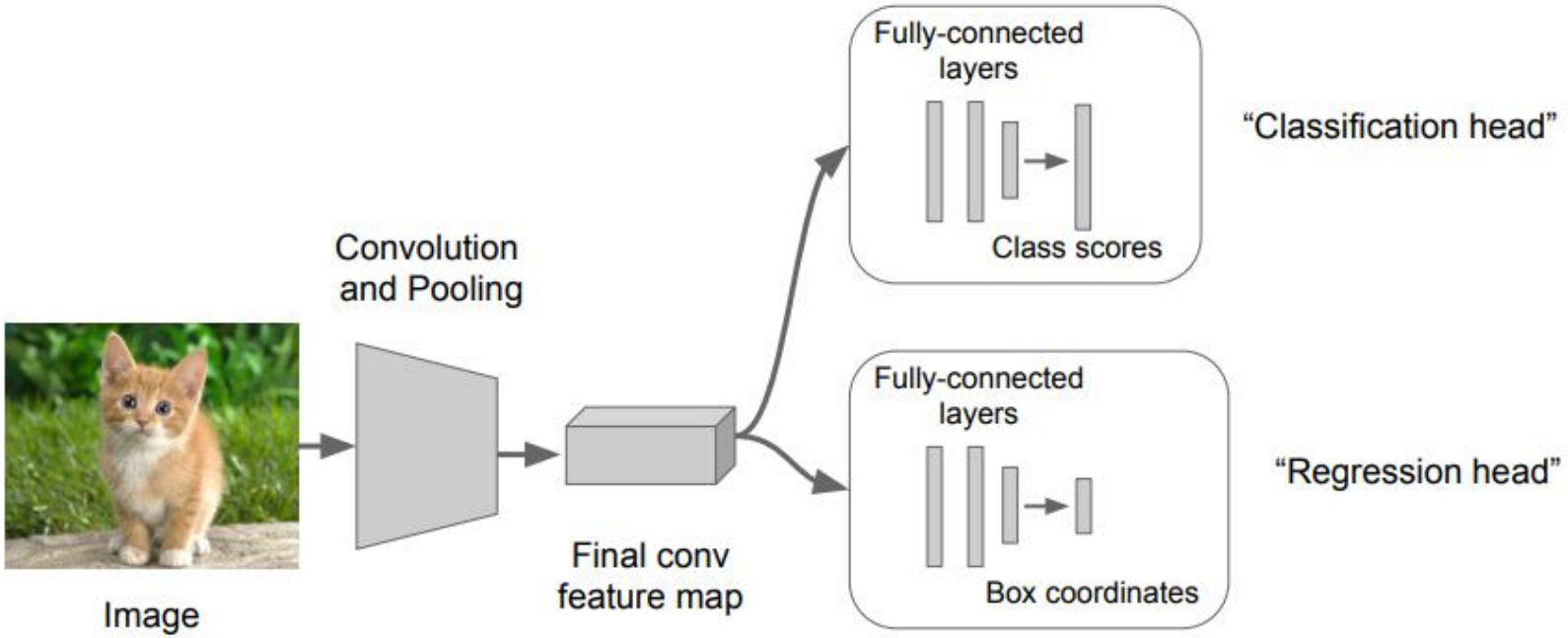
$$E_{SSE} = \frac{1}{2}\sum_p \sum_j \left(targ_j^p - out_j^{(N)p}\right)^2$$

Idea #1: Localization as Regression



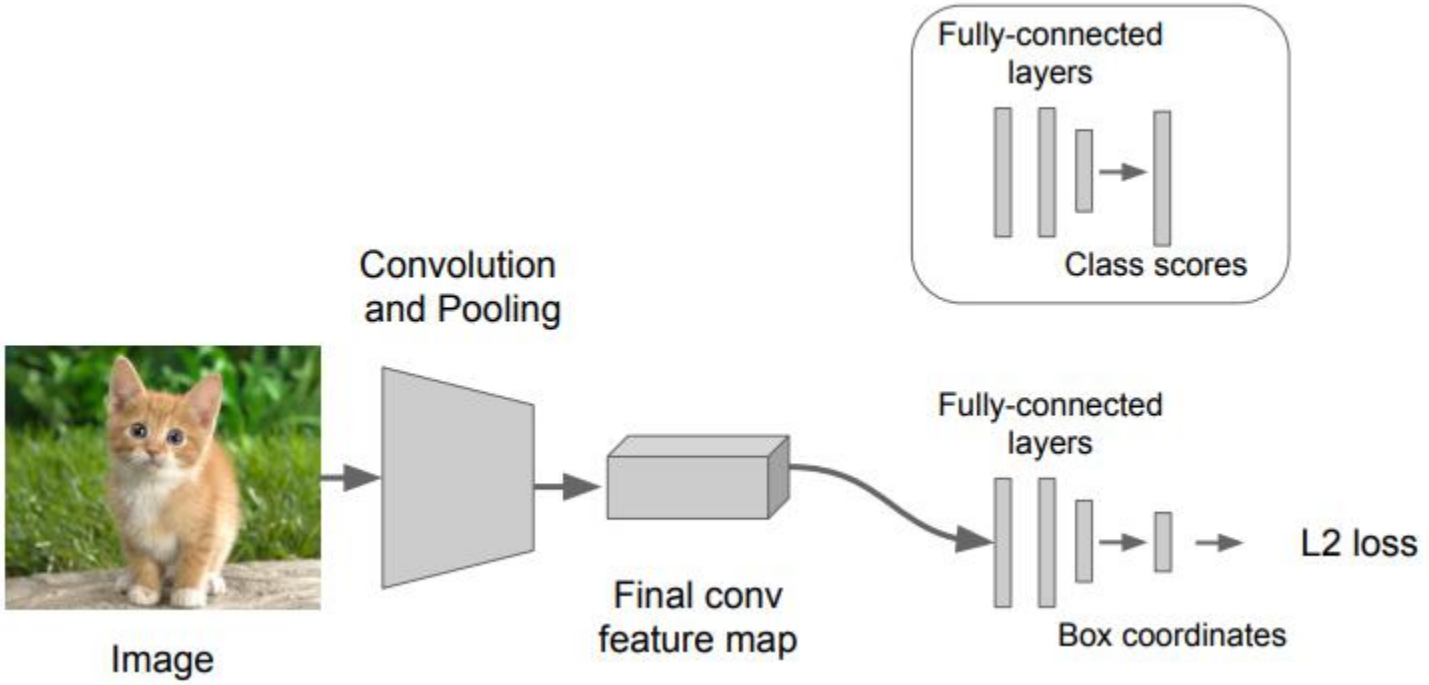
Simple Recipe for Classification + Localization

- Step 2: Attach new fully-connected "regression head" to the network

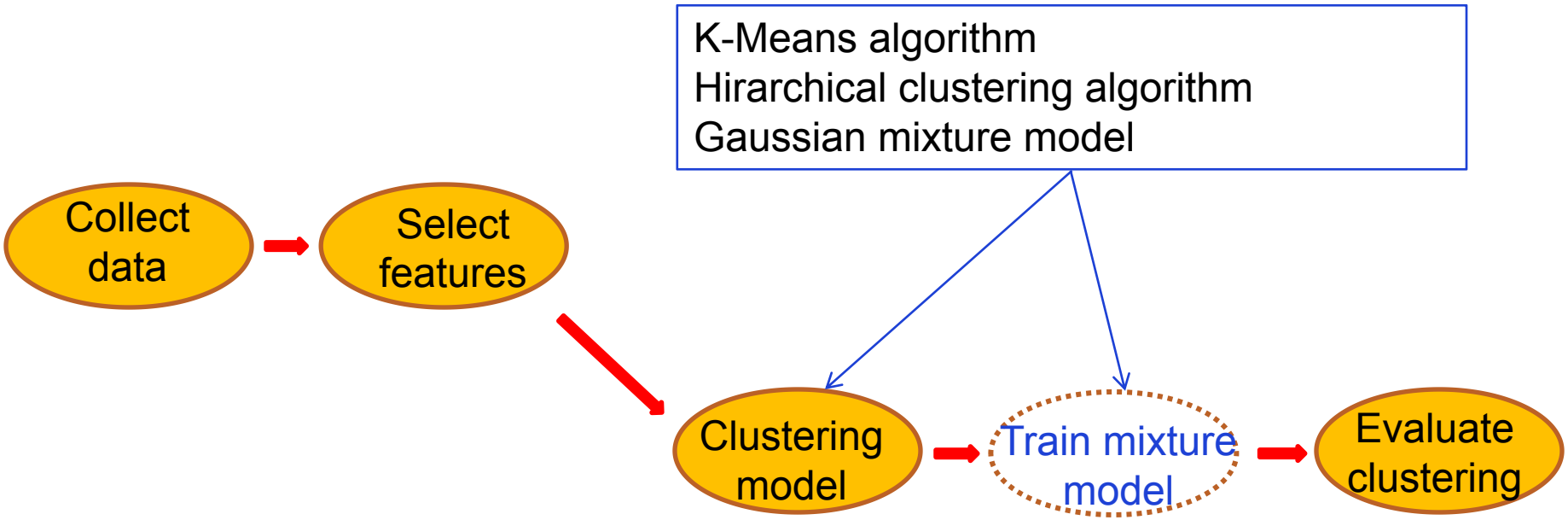


Simple Recipe for Classification + Localization

- Step 3: Train the regression head only with SGD and L2 loss



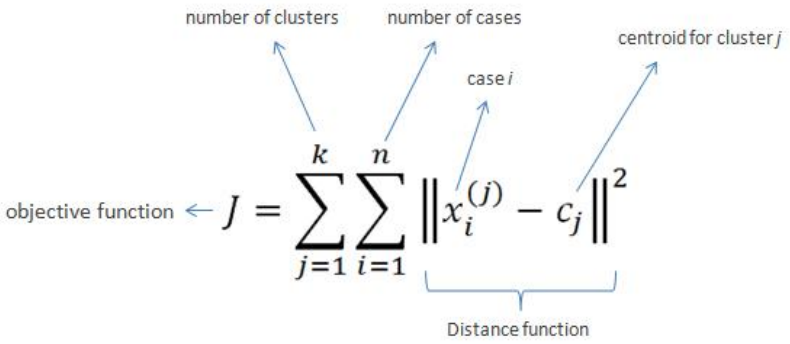
Pattern recognition design cycle

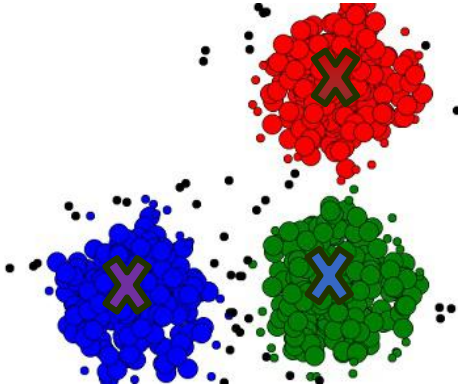


Clustering evaluation

- Intra-cluster cohesion (compactness)
  - Cohesion measures how near the data points in a cluster are to the cluster centroid.
  - Sum of squared error (SSE) is a commonly used measure.
- Inter-cluster seperation (isolation)
  - Separation means that different cluster centroids should be far away from one another.

Clustering is hard to evaluate. In most applications, expert judgements are still the key.





Data Clustering - Formal Definition

- Given a set of N unlabeled examples D = x_1, x_2, ..., x_N in a d-dimensional feature space, D is partitioned into a number of disjoint subsets Dj's:

$$D = \cup_{j=1}^{k} D_j \qquad D_i \cap D_j = \emptyset, i \neq j$$

- A partition is denoted by:

$$\pi = (D_1, D_2, ..., D_k)$$

and the problem of data clustering is thus formulated as

$$\pi^* = \underset{\pi}{\operatorname{argmin}}\ f(\pi)$$

where f(·) is formulated according to a given criterion.

K-means

- K-means is a most well-known and popular clustering algorithm.
- K-means procedure:
  - Start with some initial cluster centers
  - Iterate until there are no changes in any means
    - Assign/cluster each example to closest center
    - Recalculate centers as the mean of the points in a cluster

Pros and cons of K-means

Strengths:

- **Simple**: easy to understand and to implement.
- **Efficient**: time complexity is $O(tkn) \approx O(n)$.
  - n is the number of data point, k is the number of clusters, and t is the number of iterations.
  - Since both k and t are small, k-means is considered a linear algorithm.

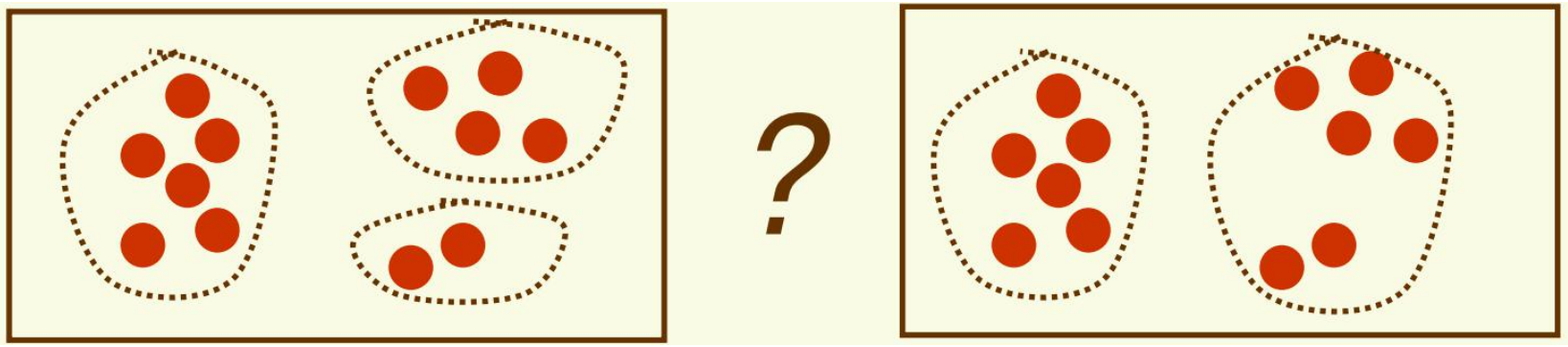
Note that: *it terminates at a local optimum if SSE is used. The global optimum is hard to find due to complexity*.

Weakneses:

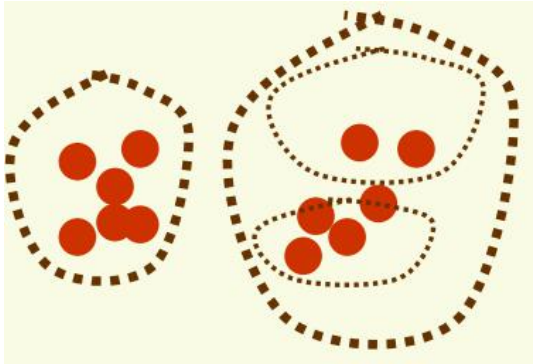
- The user needs to specify the value of K.
- Applicable only when mean is defined.
- The algorithm is sensitive to the initial seeds.
- The algorithm is sensitive to outliers.
  - Outliers are data points that are very far away from other data points.
  - Outliers could be errors in the data recording or some special data points with very different values.

Hierarchical Clustering

- Up to now, considered “flat” clustering

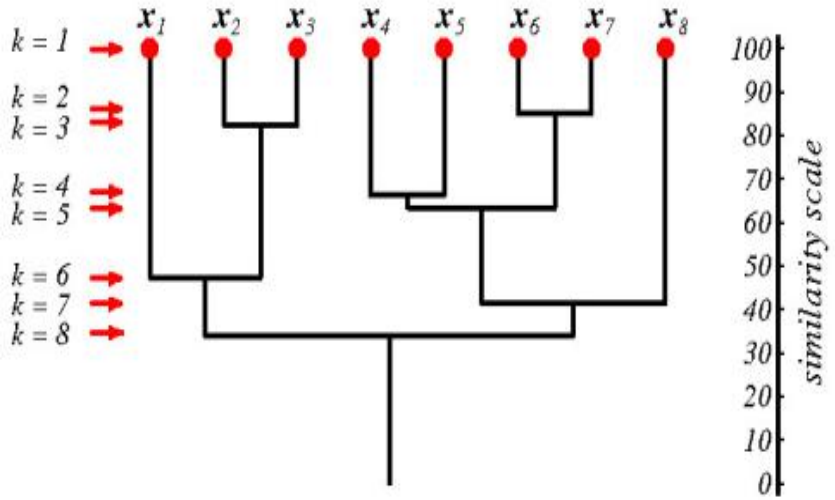


- For some data, hierarchical clustering is more appropriate than “flat” clustering
- Hierarchical clustering

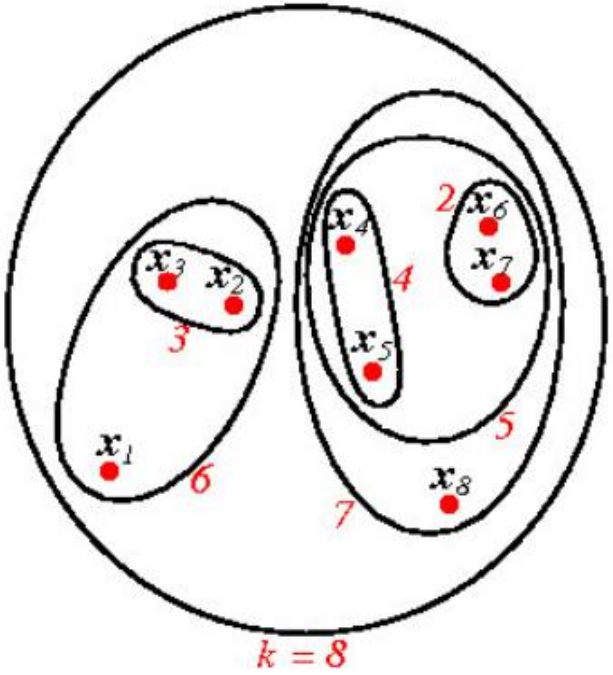


Hierarchical Clustering:

- Hierarchical cluster representation.



Dendrogram-Binary Tree



Venn diagram

Hierarchical Clustering

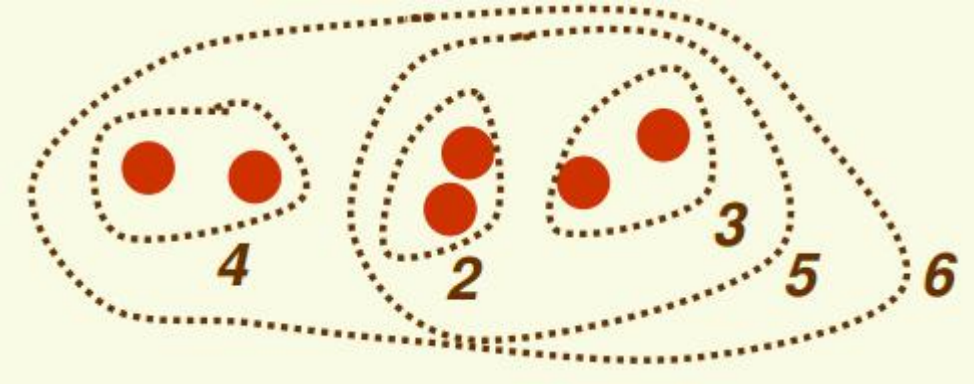
- Algorithms for hierarchical clustering can be divided into two types:
  - 1. Agglomerative (**bottom up**) procedures

    _x0001_ Start with n singleton clusters

    _x0001_ Form hierarchy by merging most similar clusters
  - 2. Divisive (**top bottom**) procedures

    _x0001_ Start with all samples in one cluster

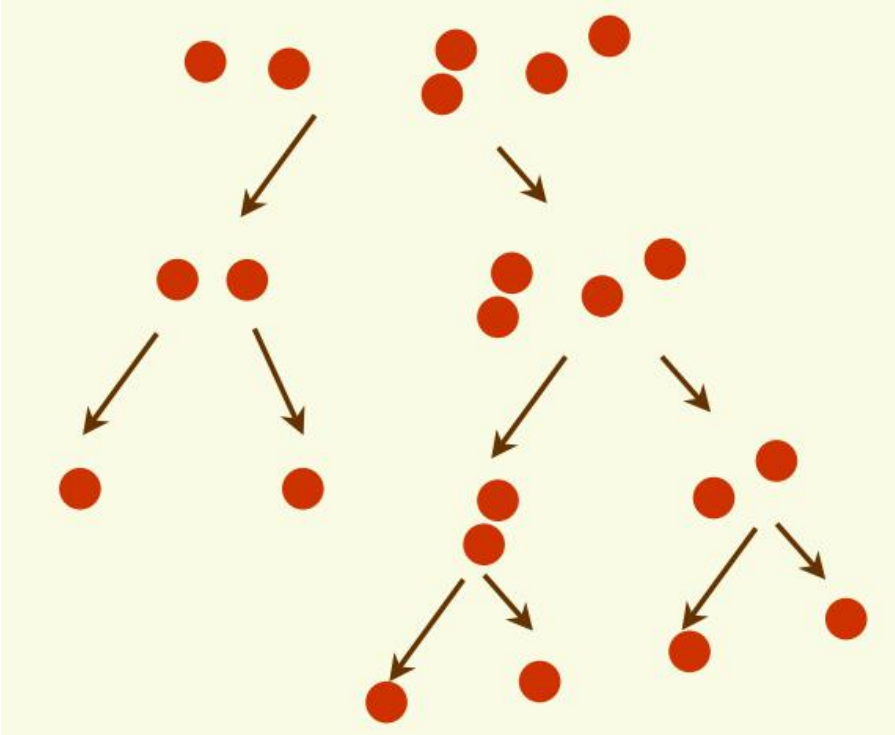
    _x0001_ Form hierarchy by splitting the “worst” clusters



Divisive Hierarchical Clustering

- Any “flat” algorithm which produces a fixed number of clusters can be used

set c = 2

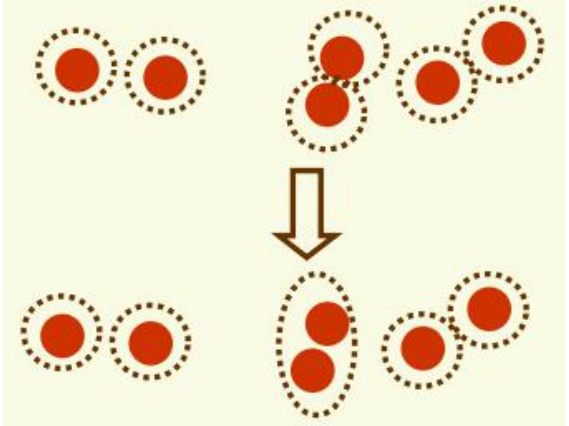


Agglomerative Hierarchical Clustering

- Initialize with each example in singleton cluster while there is more than 1 cluster
  - 1. find 2 nearest clusters
  - 2. merge them

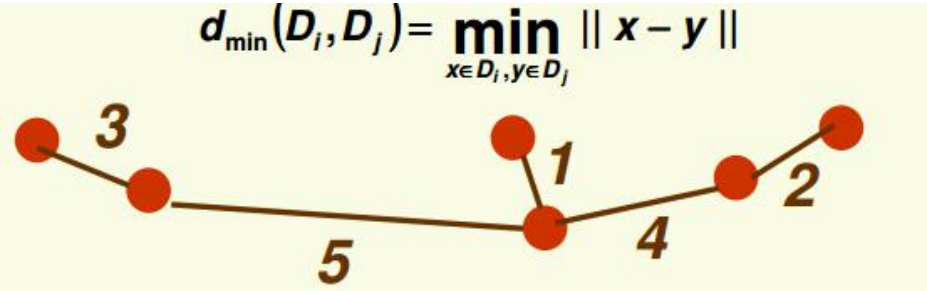
- Four common ways to measure cluster distance

minimum distance	$d_{\min}(D_i, D_j) = \min_{x \in D_i, y \in D_j} \| x - y \|$
maximum distance	$d_{\max}(D_i, D_j) = \max_{x \in D_i, y \in D_j} \| x - y \|$
average distance	$d_{avg}(D_i, D_j) = \frac{1}{n_i n_j} \sum_{x \in D_i} \sum_{y \in D_j} \| x - y \|$
mean distance	$d_{mean}(D_i, D_j) = \| \mu_i - \mu_j \|$

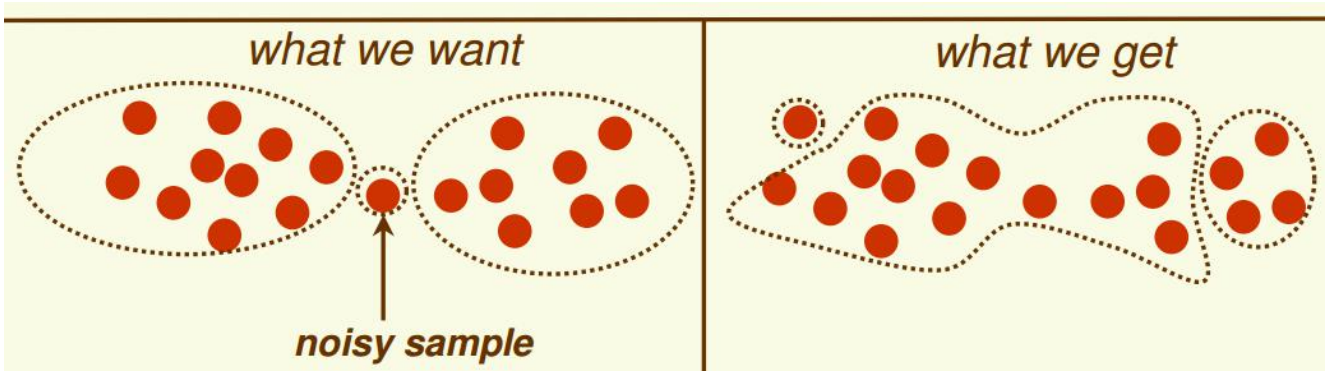


Single Linkage or Nearest Neighbor

- Agglomerative clustering with minimum distance



- generates minimum spanning tree
- encourages growth of elongated clusters
- disadvantage: very sensitive to noise

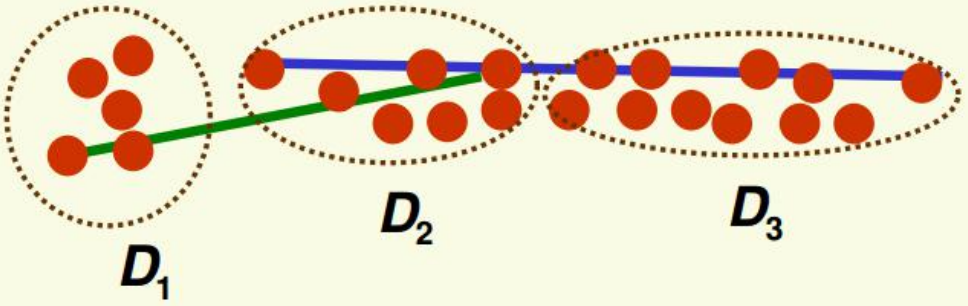


Complete Linkage or Farthest Neighbor

- Agglomerative clustering with maximum distance

$$d_{\max}(D_i, D_j) = \max_{x \in D_i, y \in D_j} \| x - y \|$$

- Encourages compact clusters
- Does not work well if elongated clusters present



- $d_{\max}(D_1, D_2) < d_{\max}(D_2, D_3)$
- thus D_1 and D_2 are merged instead of D_2 and D_3

Average and Mean Agglomerative Clustering

- Agglomerative clustering is more robust under the average or the mean cluster distance

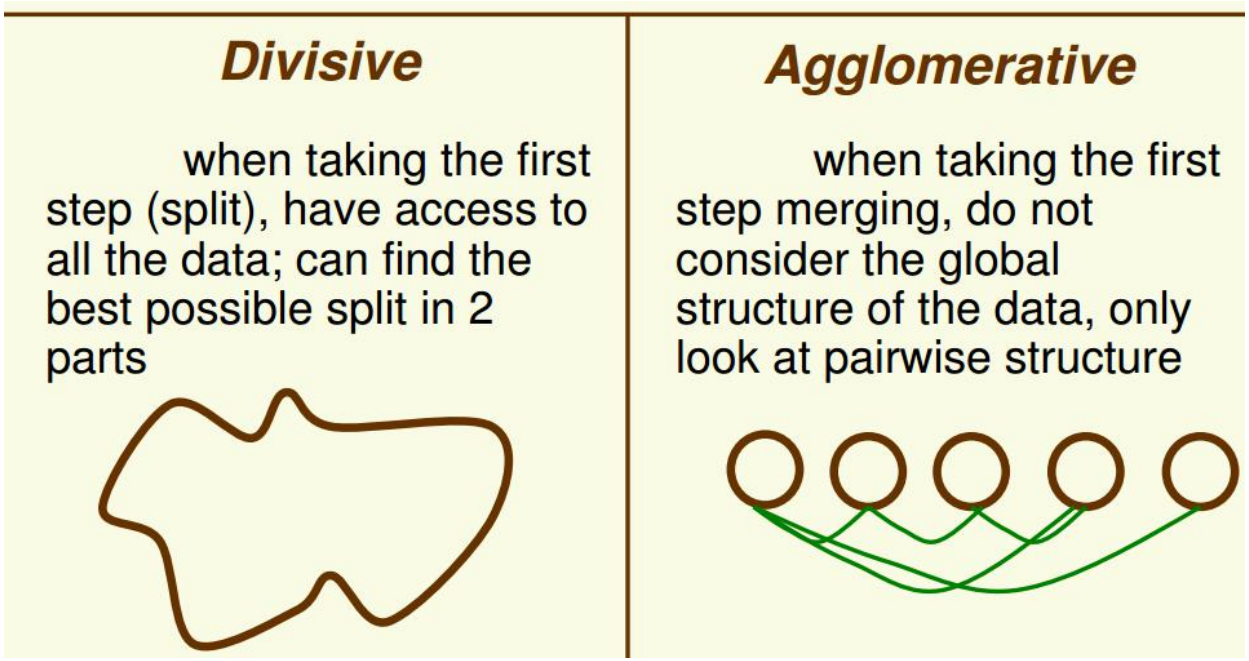
$$d_{avg}(D_i, D_j) = \frac{1}{n_i n_j} \sum_{x \in D_i} \sum_{y \in D_j} || x - y ||$$

$$d_{mean}(D_i, D_j) = || \mu_i - \mu_j ||$$

- Mean distance is cheaper to compute than the average distance
- Unfortunately, there is not much to say about agglomerative clustering theoretically, but it does work reasonably well in practice

Agglomerative vs. Divisive

- Agglomerative is faster to compute, in general
- Divisive may be less "blind" to the global structure of the data



Mixture Density Model

- Model data with density model.

component densities

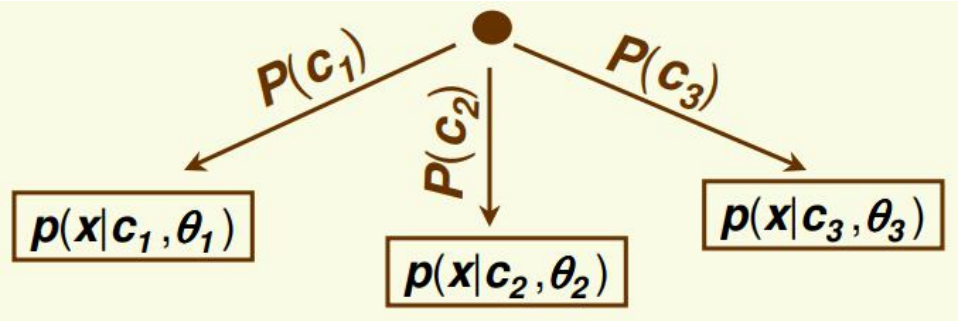
$$p(x \mid \theta) = \sum_{j=1}^{m} p(x \mid c_j, \theta_j) P(c_j)$$

mixing parameters

where $\theta = \{\theta_1, \ldots, \theta_m\}$

$$P(c_1) + P(c_2) + \ldots + P(c_m) = 1$$

- To generate a sample from distribution p(x|θ)
  - first select j with probability $p(c_j)$
  - then generate x according to probability law $p(x|c_j, \theta_j)$



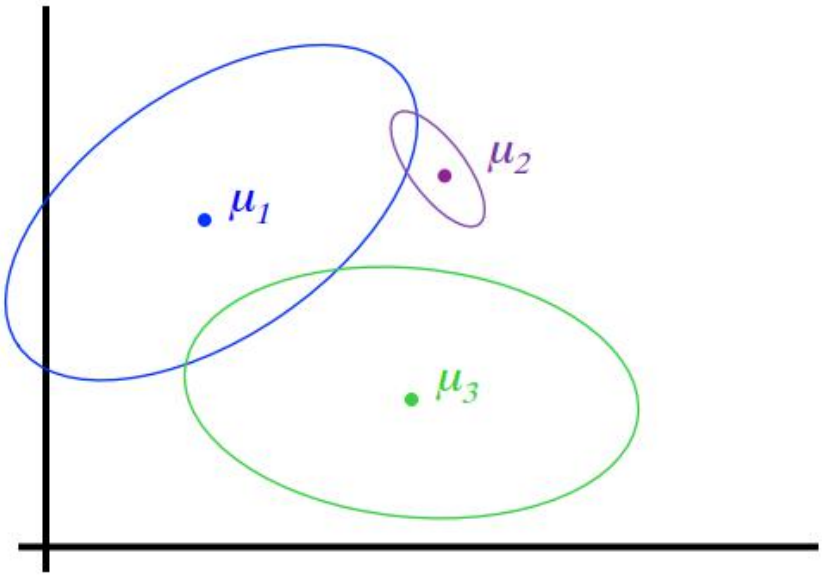
The General GMM Assumption

- P(Y): There are k components
- P(X|Y): Each component generates data from a **multivariate Gaussian** with mean μ_i and covariance matrix Σ_i

Each data point is sampled from a ***generative process***:

1. Choose component i with probability *P(y=i)*
2. Generate datapoint ~ N(m_i, Σ_i)

Gaussian mixture model (GMM)



ML Estimation for Mixture Density

$$p(x \mid \theta, \rho) = \sum_{j=1}^{m} p(x \mid c_j, \theta_j) P(c_j) = \sum_{j=1}^{m} p(x \mid c_j, \theta_j) \rho_i$$

- Can use Maximum Likelihood estimation for a mixture density; need to estimate

$$\theta_1, \ldots, \theta_m$$

$$\rho_1 = P(c_1), \ldots, \rho_m = P(c_m), \text{ and } \rho = \{\rho_1, \ldots, \rho_m\}$$

- As in the supervised case, form the logarithm likelihood function

$$l(\theta, \rho) = \ln p(D \mid \theta, \rho) = \sum_{k=1}^{n} \ln p(x_k \mid \theta, \rho) = \sum_{k=1}^{n} \ln \left[\sum_{j=1}^{m} p(x \mid c_j, \theta_j) \rho_i \right]$$

Expectation Maximization Algorithm

- EM is an algorithm for ML parameter estimation when the data has missing values. It is used when
  - 1. data is incomplete (has missing values)
    - some features are missing for some samples due to data corruption, partial survey responces, etc.
    - This scenario is very useful
  - 2. Suppose data X is complete, but $p(X|\theta)$ is hard to optimize. Suppose further that introducing certain hidden variables U whose values are missing, and suppose it is easier to optimize the "complete" likelihood function $p(X,U|\theta)$. Then EM is useful.
    - This scenario is useful for the mixture density estimation, and is subject of our lecture today
- Notice that after we introduce artificial (hidden) variables U with missing values, case 2 is completely equivalent to case 1

EM: Joint Likelihood

- Let $z_i = \{z_i^{(1)}, \ldots, z_i^{(m)}\}$, and $Z = \{z_1, \ldots, z_n\}$
- The complete likelihood is

$$p(X, Z \mid \theta) = p(x_1, \ldots, x_n, z_1, \ldots, z_n \mid \theta) = \prod_{i=1}^{n} p(x_i, z_i \mid \theta)$$

$$= \prod_{i=1}^{n} \underbrace{p(x_i \mid z_i, \theta)}_{\text{gaussian}} \underbrace{p(z_i)}_{\text{part of } \rho_c}$$

- If we actually observed Z, the log likelihood $\ln[p(X,Z \mid \theta)]$ would be trivial to maximize with respect to θ and ρ_i
- The problem, of course, is that the values of Z are missing, since we made it up (that is Z is hidden)

EM Algorithm

- EM solution is to iterate

1. start with initial parameters $\theta^{(0)}$

iterate the following 2 step until convergence

E. compute the expectation of log likelihood with respect to current estimate $\theta^{(t)}$ and $\boldsymbol{X}$. Let's call it $\boldsymbol{Q}(\theta|\theta^{(t)})$

$$\boldsymbol{Q}\left(\theta \mid \theta^{(t)}\right) = \boldsymbol{E}_{\boldsymbol{Z}}\left[\ln \boldsymbol{p}(\boldsymbol{X},\boldsymbol{Z} \mid \theta) \mid \boldsymbol{X},\theta^{(t)}\right]$$

M. maximize $\boldsymbol{Q}(\theta|\theta^{(t)})$

$$\theta^{(t+1)} = \arg\max_{\theta} \boldsymbol{Q}\left(\theta \mid \theta^{(t)}\right)$$

EM Algorithm and K-means

- k-means can be derived from EM algorithm
- Setting mixing parameters equal for all classes,

$$E_Z\left[z_i^{(k)}\right]=\frac{\rho_k \exp\left(-\frac{1}{2\sigma^2}(x_i-\mu_k)^2\right)}{\sum_{j=1}^{m}\rho_j \exp\left(-\frac{1}{2\sigma^2}(x_i-\mu_j)^2\right)}=\frac{\exp\left(-\frac{1}{2\sigma^2}(x_i-\mu_k)^2\right)}{\sum_{j=1}^{m}\exp\left(-\frac{1}{2\sigma^2}(x_i-\mu_j)^2\right)}$$

- If we let $\sigma \to \infty$, then

$$E_Z\left[z_i^{(k)}\right]=\begin{cases}1 & \text{if } \forall j,\ \| x_i-\mu_k \| > \| x_i-\mu_j \| \\ 0 & \text{otherwise}\end{cases}$$

  - so at the E step, for each current mean, we find all points closest to it and form new clusters
  - at the M step, we compute the new means inside current clusters

$$\mu_k=\frac{1}{n}\sum_{i=1}^{n}E_Z\left[z_i^{(k)}\right]x_i$$

Q & A