ECS 171 Machine Learning

Lecture9: DNN (L1,L2 regularization, Dropout), Intro to Graphical Models, Bayesian Networks

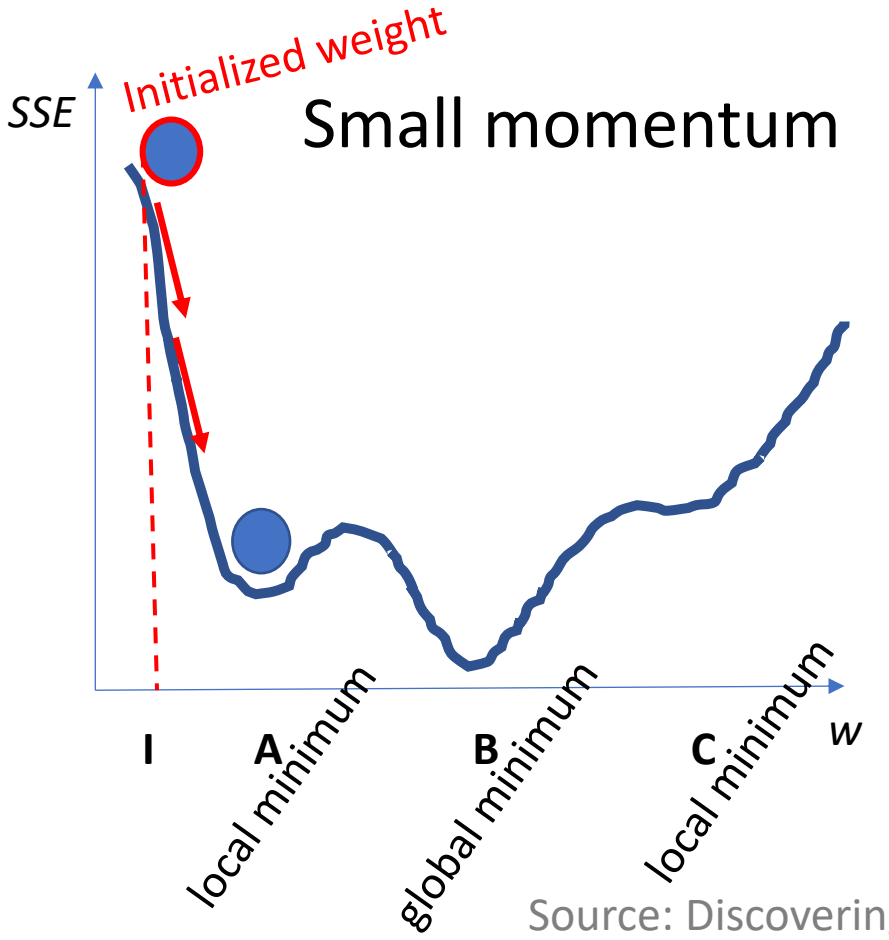
Instructor: Dr. Setareh Rafatirad

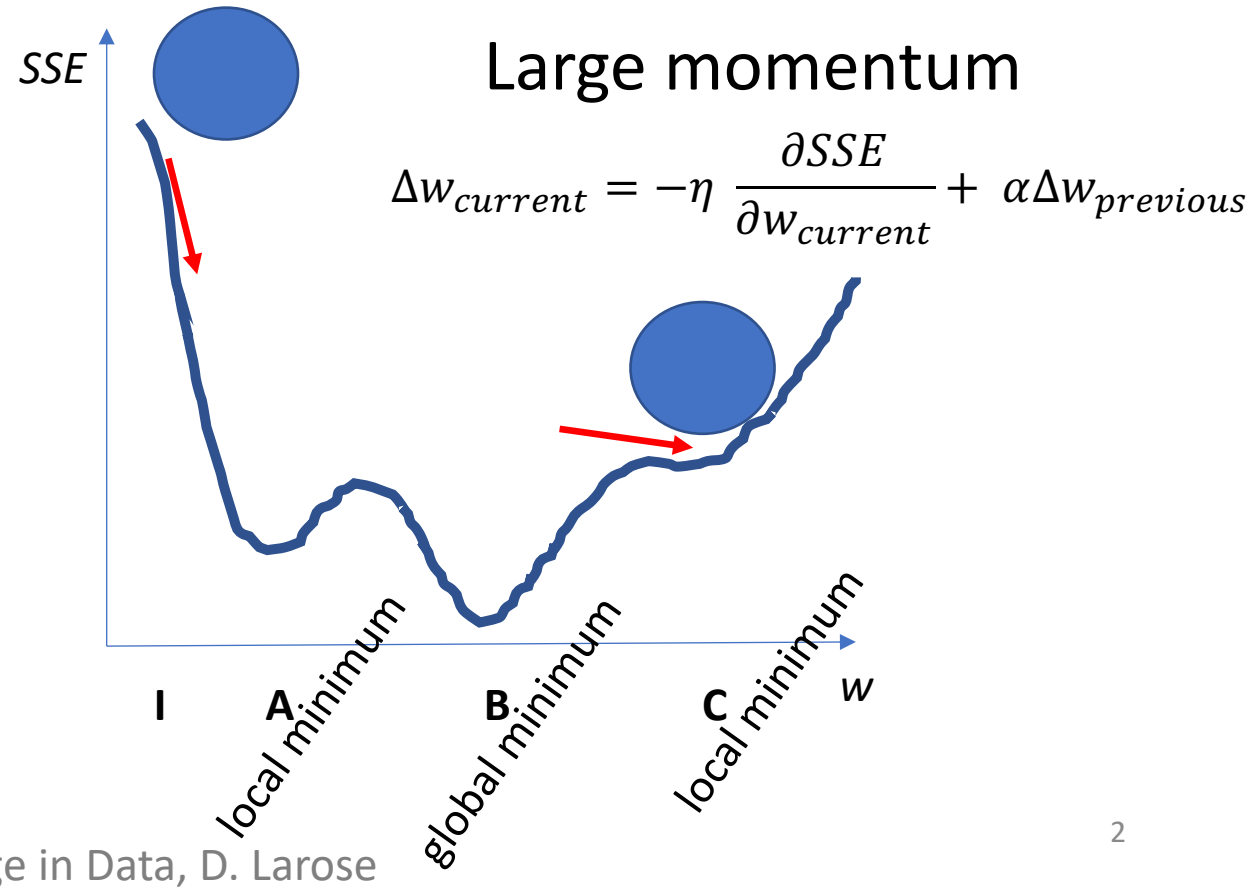
Momentum Term Influence Recap

Backpropagation is made more powerful with the addition of a momentum term α.
The momentum term represents inertia.

- "momentum" helps to know the direction of the next step with the knowledge of the previous steps.
- It helps to prevent oscillations.
- Momentum can also be implemented with mini-batch GD.

$$\Delta w_{current} = -\eta \frac{\partial SSE}{\partial w_{current}} + \alpha \Delta w_{previous}$$





$$\Delta w_{current} = -\eta \frac{\partial SSE}{\partial w_{current}} + \alpha \Delta w_{previous}$$

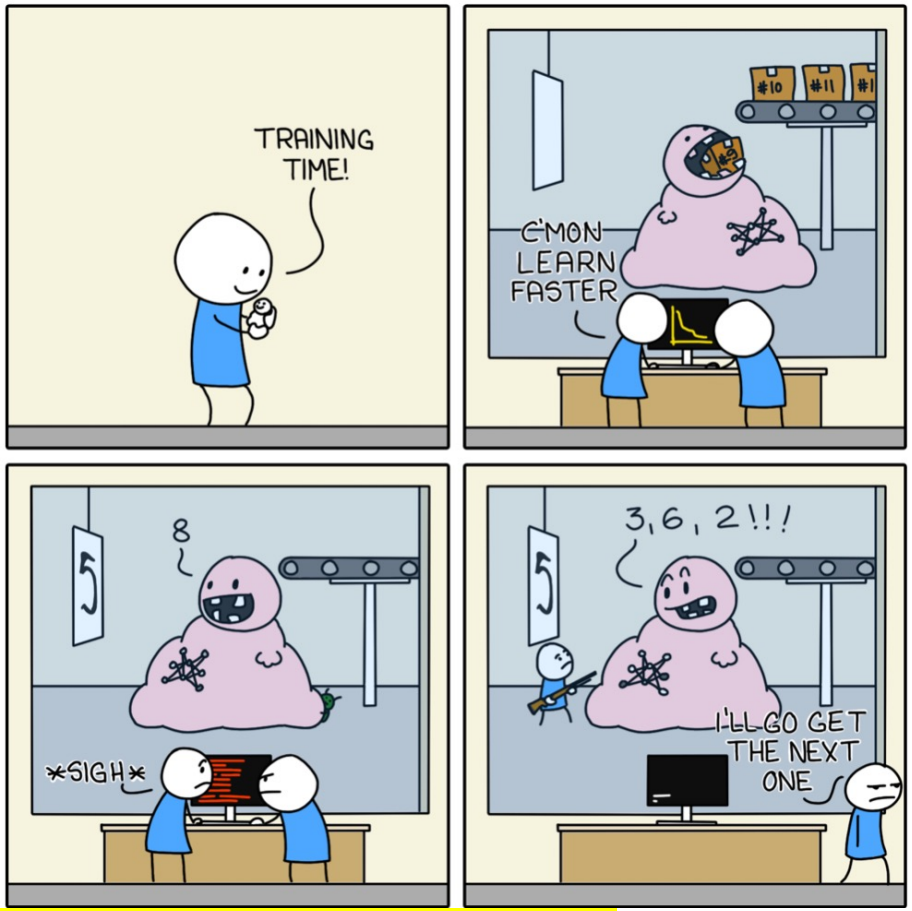
Source: Discovering Knowledge in Data, D. Larose

Momentum with Mini-Batch GD Recap

Backpropagation is made more powerful with the addition of a <u>momentum term</u> α. The momentum term represents inertia.

η : *learning rate* ; $0 \leq \eta \leq 1$ $\qquad$ $0 \leq \alpha < 1$

$$\Delta w_{current} = -\eta \frac{\partial SSE}{\partial w_{current}} + \alpha \underbrace{\Delta w_{previous}}_{\text{Previous weight adjustment}}$$



(Reed & Marks) prove that including momentum in backpropagation algorithm results in the adjustment becoming exponential average of all pervious adjustments:

$$\Delta w_{current} = -\eta \sum_{k=0}^{\infty} \alpha^k \frac{\partial SSE}{\partial w_{current-k}}$$

large values of α allows the algorithm to "remember" more terms in the adjustment history.

small values of α reduce the inertial effects as well as the influence of the recent adjustments, until α=0.

Momentum with Mini-Batch Gradient Descent Recap

mini-batches contain 2 to several hundred samples
For large models, the choice of the mini-batch size is constrained by computational resources.
Batch size impacts the stability and speed of learning.

$$if\ \nabla_\theta L(\theta) = gradient\ of\ the\ loss,$$
$$then\ momentum\ is\ implemented\ by\ computing\ a\ moving\ average:$$
$$\Delta\theta = -\eta\nabla_\theta L(\theta) + \alpha\Delta\theta^{old}\ ;\ \alpha \in [0,1] \quad \equiv \quad \Delta w_{current} = -\eta\ \frac{\partial SSE}{\partial w_{current}} + \alpha\Delta w_{previous}$$

α: momentum term

- *Commonly, momentum is first initialized to 0.9 , but it is then tuned similar to learning rate, during the training process. A typical choice of momentum is between 0.5 and 0.9.*
- *Learning rate may be adopted over epochs t to give η_t . In the first few epochs, a fixed learning rate is often used. Then a <u>decaying</u> schedule is followed:*

$$\eta_t = \frac{\eta_0}{1+\varepsilon t}, or\ \eta_t = \frac{\eta_0}{t^\varepsilon}, (0.5 < \varepsilon \leq 1)$$

η_t : learning rate at epoch t, it can be different at each epoch

Mini-Batch Gradient Descent Recap

- Deep learning models are often optimized using mini-batch gradient descent.
- Uses a small subset of the data and updates the weights based on the average gradient of the subset. The rest is the same (initialize the parameters, enter a parameter update loop, terminate by monitoring a test/validation set). In contrast with stochastic GD , the main loop iterates over the mini-batches.
- The mini-batches depending on the batch size, are obtained from the training set. Batches are randomly selected non-overlapping subsets of training set.
- Weights (parameters) are updated after processing each batch.
- Pros
  - more efficient than stochastic gradient descent since the batching allows the efficiency of not having all training data in memory and algorithm implementations.
  - More accurate than batch gradient descent
- Cons
  - Requires an additional hyper-parameter called mini-batch size
  - Error information must be accumulated across mini-batches of training data (similar to batch GD)

Regularization in DNN

- Regularization decreases model variance
- Regularizers allow you to apply penalties on layer parameters
- Regularization Techniques in Deep Learning
  - L2 and L1 regularization
  - Dropout
  - Early stopping
  - Data augmentation (a form of adding prior knowledge to a model)

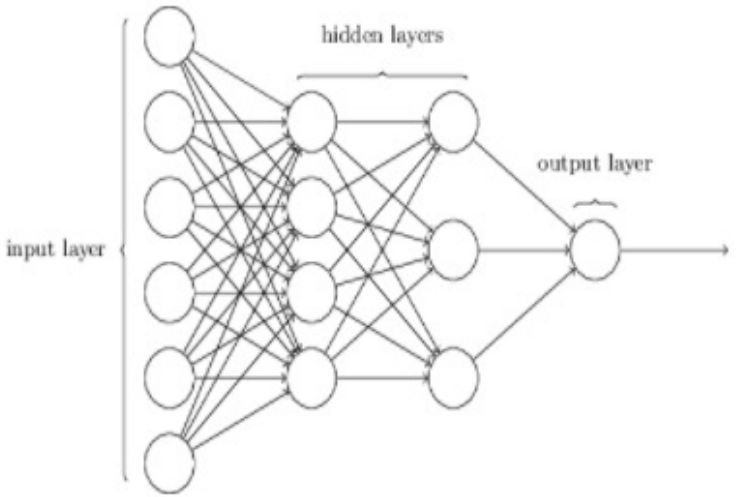
Cost function = $L(\theta)$ + Regularization term

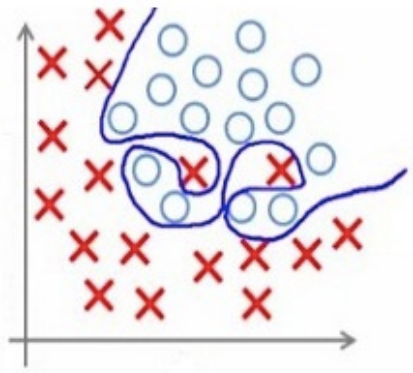
update rule:

$$\theta_{new} = \theta - \eta_t \left[\frac{1}{B_k} \sum_{i \in I} \left[\frac{\partial L(f(x_i;\theta), y_i)}{\partial \theta} \right] + \frac{B_k}{N} \lambda \frac{\partial R(\theta)}{\partial \theta} \right]$$

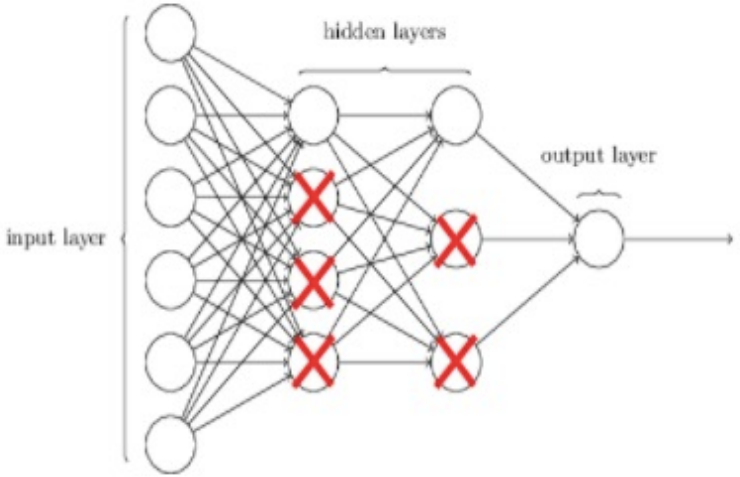
B_k: Batch size
θ: model's weight
L: Loss function
$R(\theta)$: Regulariser with weight λ (regularization parameter)

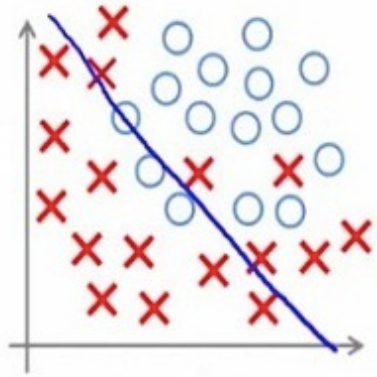
Objective: use optimized regularization coefficient and obtain a generalized model.





Over-fitting





Under-fitting

When the regularization coefficients are too high, some of the weights matrices are nearly equal to zero. Can you explain why?

L2 & L1 Regularization

- The regularization term differs in L1 and L2.
  - https://keras.io/api/layers/regularizers/
- L2 regularization is also known as *weight decay* as it forces the weights to decay towards zero (but not exactly zero).

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum ||w||^2$$

λ= regularization parameter

- L1 regularization may reduce the weights to zero.

Regularization term

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum ||w||$$

m = training examples with n features

lambda is the regularization parameter.

Weight Regularization in Keras

- L1: Sum of the absolute weights.
- L2 :Sum of the squared weights.
- L1L2: Sum of the absolute and the squared weights.

In DNN, a weight regularizer can be added to each hidden layer:

```
keras.regularizers.l1(0.01)
keras.regularizers.l2(0.01)
keras.regularizers.l1_l2(l1=0.01, l2=0.01)
```

- https://keras.io/api/layers/regularizers/

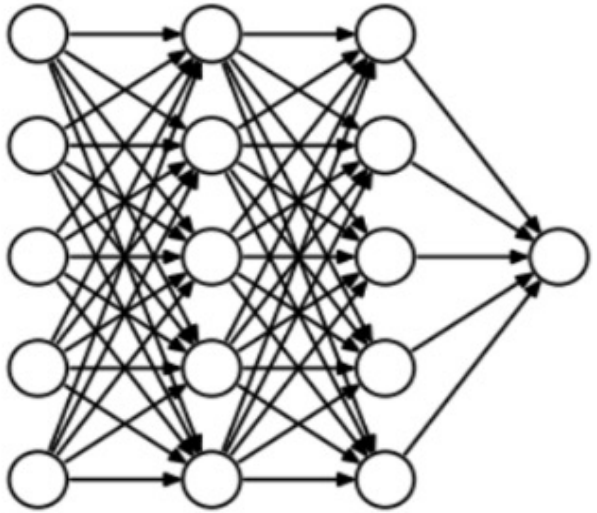
Sample code to apply L2 regularization (Directly calling a regularizer)

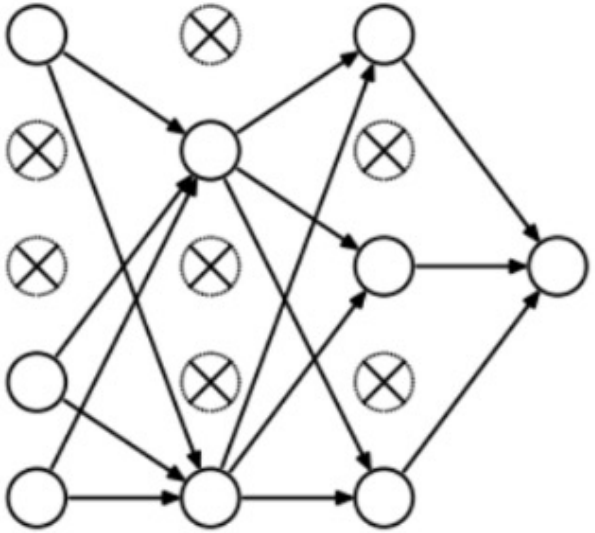
```
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense
from keras.regularizers import l2

def create_model():
    # Create model
    model = Sequential(name=name)
    model.add(Dense(nodes, input_dim=input_dim, activation='relu’))
    model.add(Dense(output_dim=output_dim, activation='relu’),
      kernel_regularizer=regularizers.l2(0.01))
    model.add(Dense(n output_dim=output_dim, activation='relu’),
      kernel_regularizer=regularizers.l2(0.01))
    model.add(Dense(output_dim, activation='sigmoid’) )

    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
    return model
```

Dropout Regularization





- Frequently used in Deep Learning.
- **At every iteration, some nodes are randomly selected and removed along with all of their incoming and outgoing connections.**
- So each round (iteration) has a different set of nodes and this results in a different set of outputs. So it creates a different model at each iteration (one feed-forward followed by one backprop pass).
- Dropout outperforms a normal network model. In this sense, it can be thought of as *ensemble model* because it generates multiple models.
- **The probability of choosing the number of nodes to be dropped is the hyperparameter of the dropout function.**
- One can define the probability of dropping to have a value (such as 0.25) and tune it in further for better results using grid search method.
- In keras, one can implement dropout using keras core layer: https://keras.io/api/layers/core_layers/#dropout https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/

Dropout Regularization in keras used in DNN

```
# dropout in the input layer with weight constraint
def create_model():
        # create model
        model = Sequential()
        model.add(Dropout(0.2, input_shape=(60,)))
        model.add(Dense(60, activation='relu', kernel_constraint=maxnorm(3)))
        model.add(Dense(30, activation='relu', kernel_constraint=maxnorm(3)))
        model.add(Dense(1, activation='sigmoid'))
        # Compile model
        sgd = SGD(lr=0.1, momentum=0.9)
        model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])
        return model
```

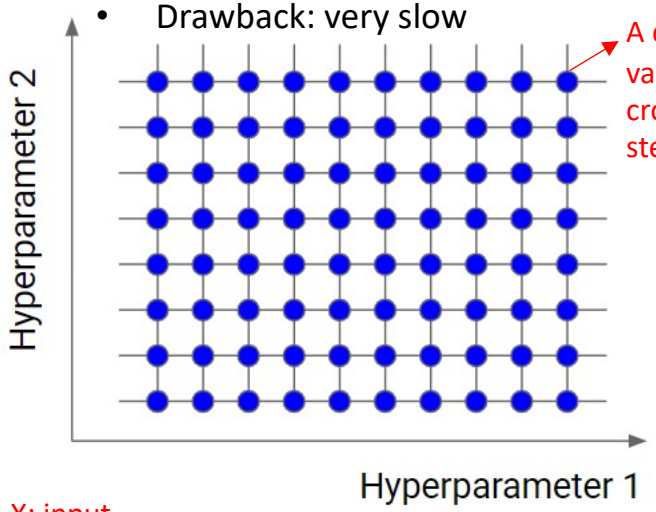
Random Search vs Grid Search

The point of the grid that maximizes the average value for some performance metrics in cross-validation, is the optimal combination of values for the hyperparameters.

Grid Search

from sklearn.model_selection import GridSearchCV

- Supports discrete values
- Gives the best combination of values of the hyperparameters
- Drawback: very slow



A combination of hyperparameter values. Each point needs k-fold cross-validation (i.e., k training steps)

X: input
Y: output
Search space:

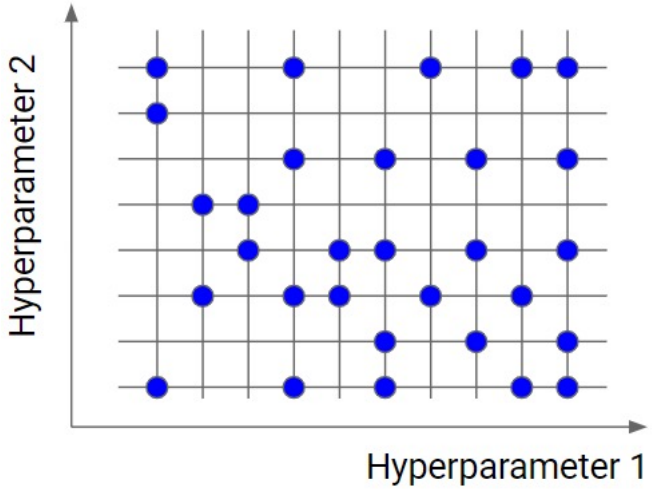
Best: 0.708333 using {'optimizer': 'Nadam'}

```
# define the grid search parameters
optimizer = ['gradient_descent_v2', 'Adagrad', 'Adadelta', 'Adam', 'Nadam']
param_grid = dict(optimizer=optimizer)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(X, Y)
```

Random Search

from sklearn.model_selection import RandomizedSearchCV

- Supports discrete or distribution of values
- Selects a randomly selected subset of the points, the smaller this subset, the faster. The larger this subset, it gets closer to grid search.



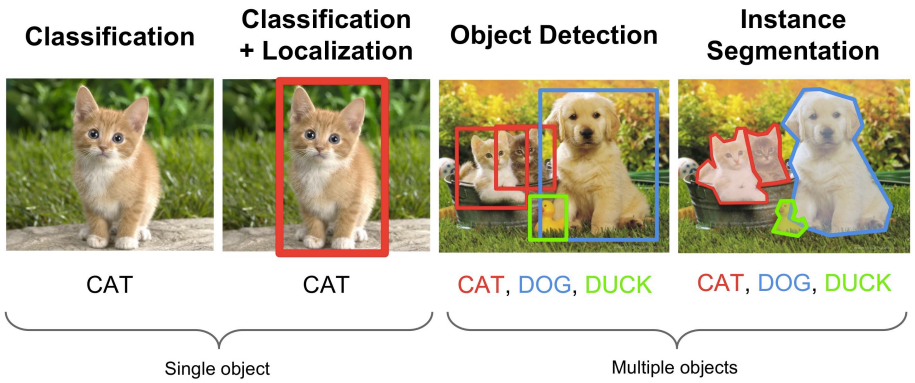
See
gridsearch_example.ipynb

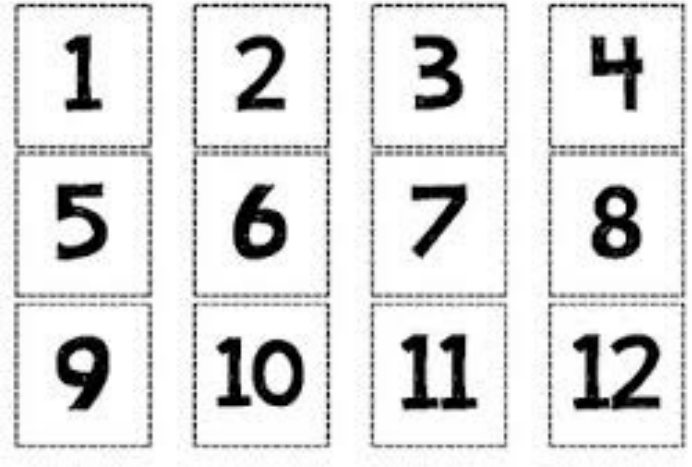
Posted on Canvas

Graphical Models

Bayesian Network

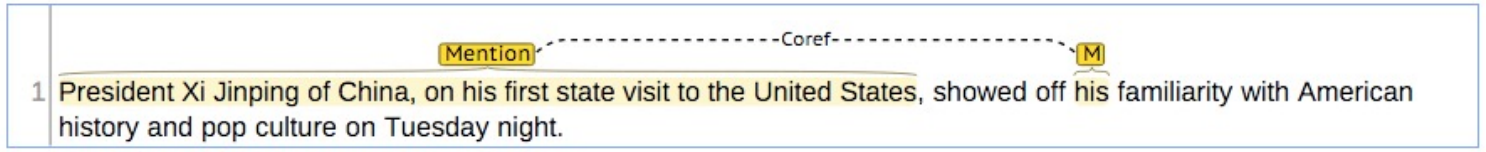
Machine Learning Problems



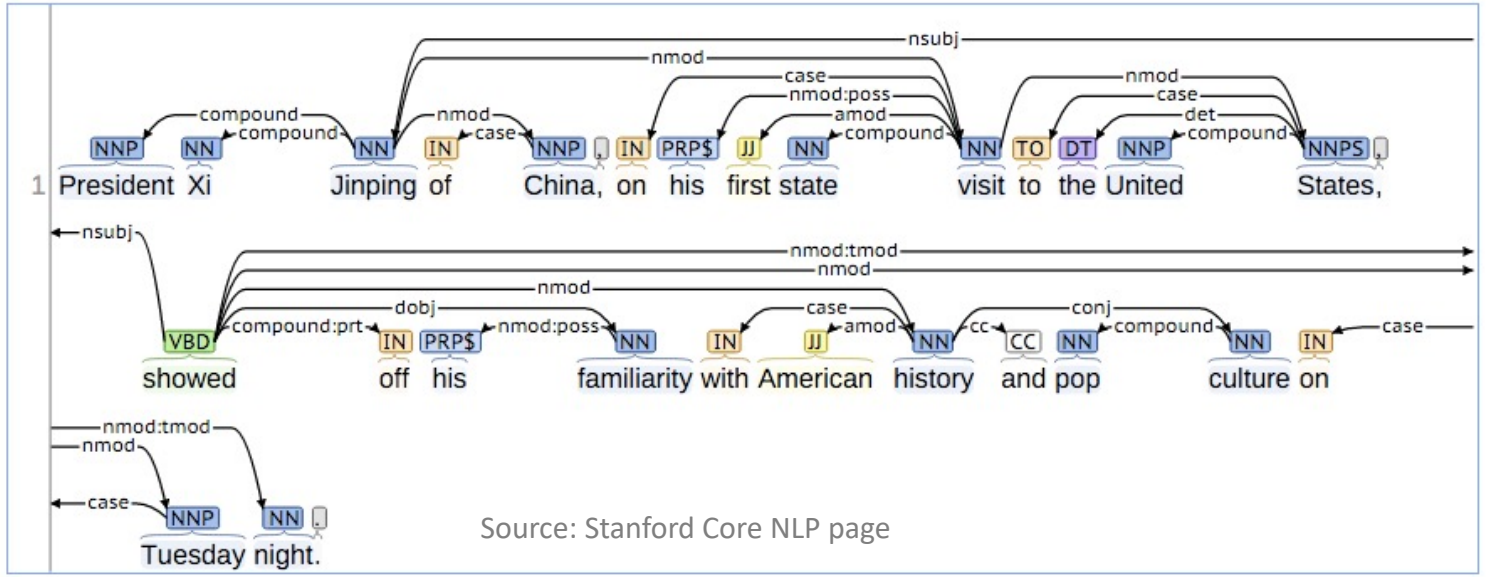


Named Entity Recognition:

Coreference:

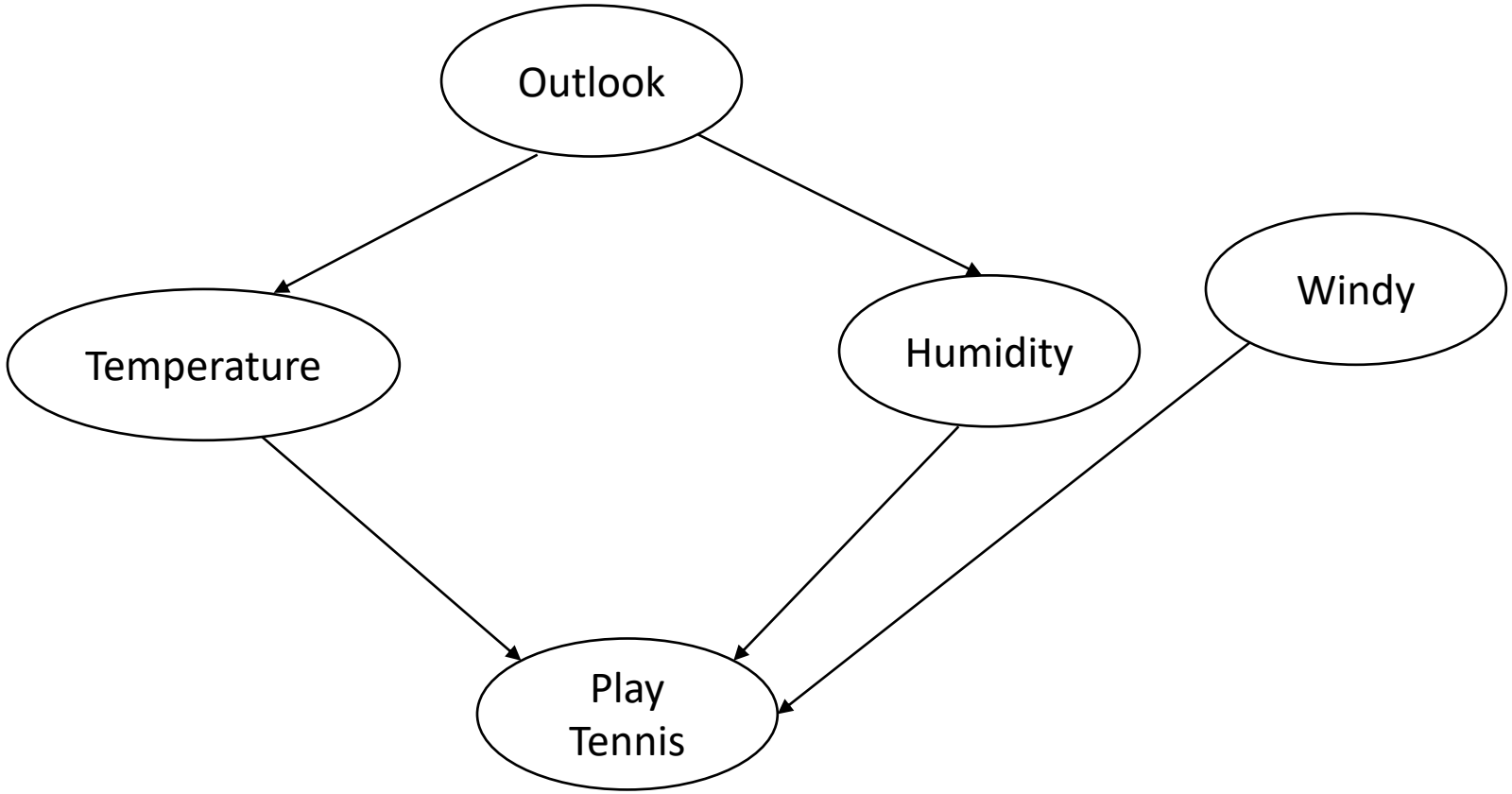


Basic Dependencies:



Source: Stanford Core NLP page

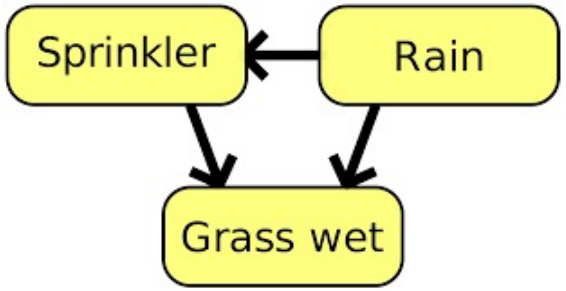
Probabilistic Graphical Models (PGM)



Benefits of PGM

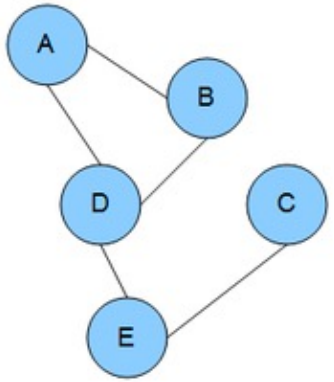
- "PGM is the diagrammatic representation of probability distribution" (Bishop, 2006)
  - Learning dependencies
  - Visualizing a probability model
  - Graphical manipulations over latent variables
  - Obtaining insights such as conditional independence
- Categorization of Probabilistic Graphical Models
  - Bayesian networks (Directed Graph)
  - Markov networks (Undirected Graph)

Bayesian Network



https://goo.gl/images/Q3wuzF

Markov Network



https://goo.gl/images/NBB17Y

Joint Probability Distribution

$$P(S,R,G) = \underbrace{P(R).P(S|R).P(G|S)}_{\text{Product of factors}}$$

Joint Probability

- Joint probability is the probability of two (or more) events occurring simultaneously:
  - "Product rule" or "chain rule" of probability is used for calculation of joint probability.
  - If A and B are independent: $P(A,B) = P(A) * P(B)$
  - If A and B are dependent: $P(A,B) = P(A|B) * P(B)$
  - Symmetric: $P(A,B) = P(B,A) = P(B|A) * P(A)$.
- Example: the probability that it rains (A) and the sky is cloudy (B): P(rain,cloudy) ; s.t. P(rain | cloudy) = 1/13 and P(cloudy) = 1/2 ➔ P(A,B) = 1/13 x 1/2 = 1/26

Marginal Probability

- Marginal Probability is the probability of an event irrespective of the outcome of another variables (unconditional probability).
  - Example: the probability that a card drawn from a pile of cards is “blue”.
- “Sum rule” for calculating marginal probability.

P(A=blue) = P(A=Blue|B=1) + P(A=Blue|B=2) =
.06+ .04 =.1 or 10%

A = color

B = value

	Red	Green	Blue	Total
1	0.12	0.42	0.06	0.6
2	0.08	0.28	0.04	0.4
Total	0.2	0.7	0.1	1.0

Conditional Probability

- Conditional Probability is the probability of one event with some relationship to one or more other events.
- P(A|B) = P(A,B) / P(B)
  - Example:

What is the probability of a randomly selected person is female given that they own a pet?

P(A=Female|B=True) = P(A=Female,B=True)/ P(B=True) =

.41/ (.45+.41) = .41/ .86 = .477 or 47.7%

B = Own a pet

A = Gender

	True	False	Total
Male	0.45	0.06	0.51
Female	0.41	0.08	0.49
Total	0.86	0.14	1.0

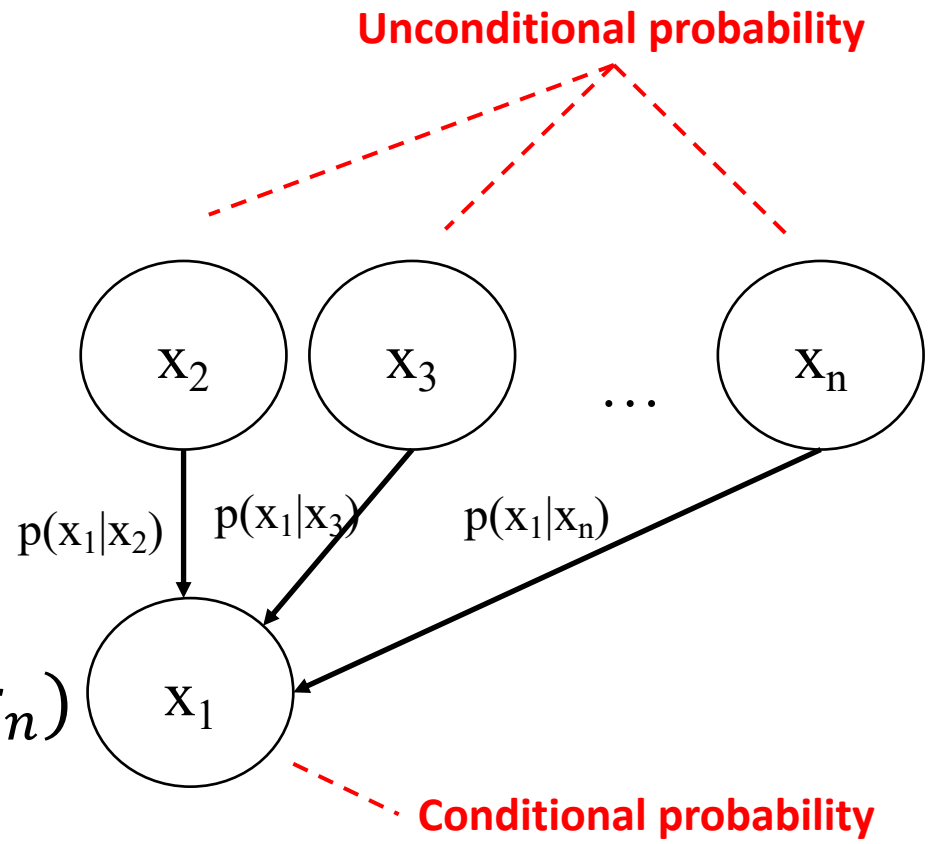
Bayesian Networks

- A Directed Acyclic Graph G and a compact representation of a probability distribution over *n* variables $x_1, x_2, x_3, \ldots, x_n$

- The generalization of random processes that depend on each other.
  - Example 1: rainy weather pattern: Dark clouds increase the probability of raining later the same day.
  - Example 2: The probability of detecting a malware is influenced by the values of internal CPU events in a microprocessor

Bayesian Networks cont.

- Vertices : Variables
- Edges: represent a conditional probability
  - An edge from y to x represents p(x|y)
  - For a vertex x_1, the conditional probability is:



$$p(x_1|x_2, x_3, \dots, x_n)$$

Important Property of BN:

$$P(x_1, x_2, \dots, x_n) = \prod_{i=1}^{n} P(x_i|Parents(x_i))$$

The joint distribution contains the information we need to compute any probability of interest in a BN.

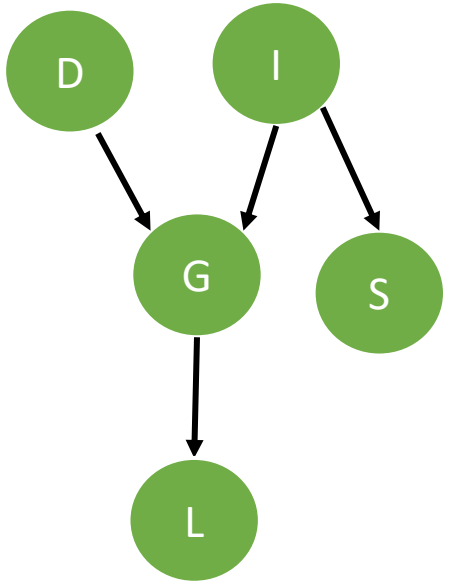
Advantages of Bayesian Networks

- A complete model for a domain
- Directed Acyclic Graph (DAG) where a node can represent a single variable or a set of variables.
- Answer probabilistic queries and compute Inference e.g., what is $P(X|e)$?
- Interoperable structure

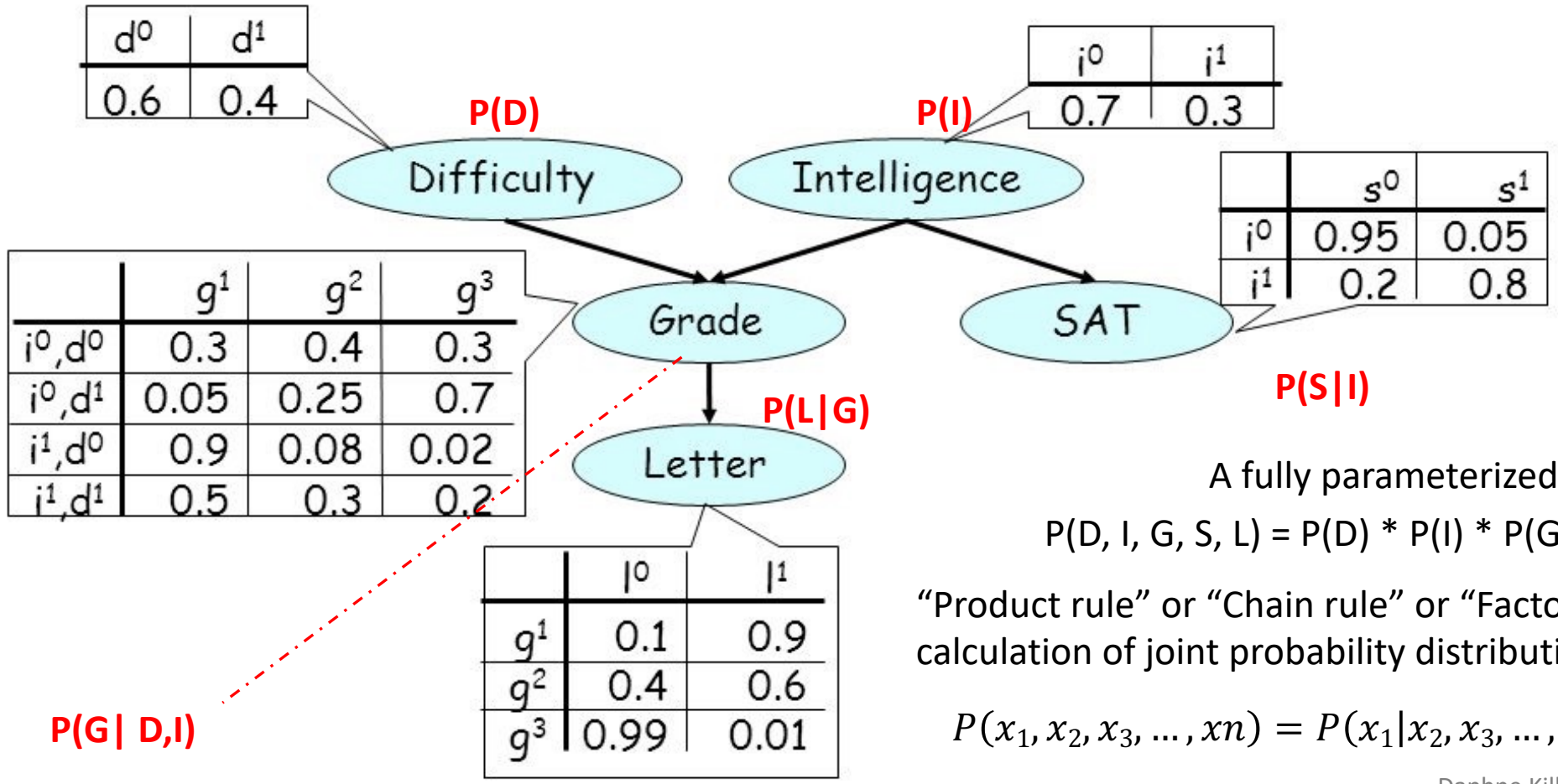
An Example of a Bayesian Belief Network

- Probability Distribution in the Student Example, P(D, I, G, S, L)

Random Variables	Possible Values
Difficulty of the class	0 (easy) and 1 (difficult)
Intelligence of a student	0 (not intelligent) and 1 (intelligent)
Grade of the student in the class	1 (good), 2 (average), and 3 (bad)
SAT score of the student	0 (low score) and 1 (high score)
Letter of recommendation	0 (not a good letter) and 1 (a good letter)



The Student Network



d^0	d^1
0.6	0.4

i^0	i^1
0.7	0.3

	s^0	s^1
i^0	0.95	0.05
i^1	0.2	0.8

	g^1	g^2	g^3
i^0,d^0	0.3	0.4	0.3
i^0,d^1	0.05	0.25	0.7
i^1,d^0	0.9	0.08	0.02
i^1,d^1	0.5	0.3	0.2

	l^0	l^1
g^1	0.1	0.9
g^2	0.4	0.6
g^3	0.99	0.01

A fully parameterized Bayesian network

P(D, I, G, S, L) = P(D) * P(I) * P(G|D,I) * P(S|I) * P(L|G)

"Product rule" or "Chain rule" or "Factoring" is used for calculation of joint probability distribution in a BN :

$$P(x_1, x_2, x_3, \dots, xn) = P(x_1|x_2, x_3, \dots, xn).P(x_2, x_3, \dots, xn)$$

Daphne Killer, Stanford University

$$P(x_1, x_2, \dots, x_n) = \prod_{i=1}^{n} P(x_i|Parents(x_i))$$

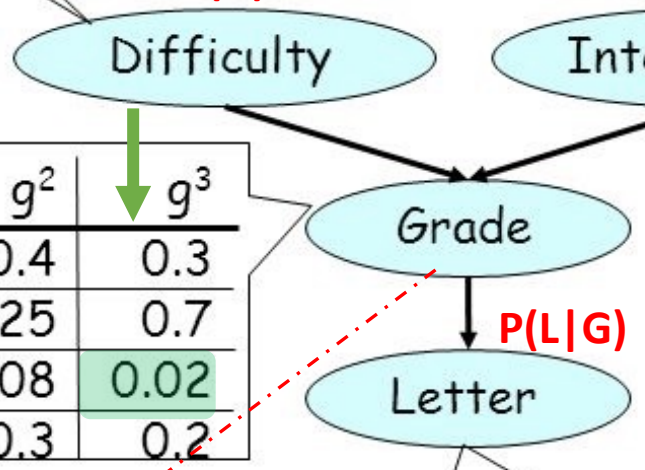
The Student Network

d^0	d^1
0.6	0.4

P(D)

i^0	i^1
0.7	0.3

P(I)



	s^0	s^1
i^0	0.95	0.05
i^1	0.2	0.8

P(S|I)

	g^1	g^2	g^3
i^0,d^0	0.3	0.4	0.3
i^0,d^1	0.05	0.25	0.7
i^1,d^0	0.9	0.08	0.02
i^1,d^1	0.5	0.3	0.2

P(L|G)

P(G| D,I)

	l^0	l^1
g^1	0.1	0.9
g^2	0.4	0.6
g^3	0.99	0.01

$$P(D, I, G, S, L) = P(D) * P(I) * P(G|D,I) * P(S|I) * P(L|G)$$

$P(d^0, i^1, g^3, s^1, l^1) = ?$

$P(d^0, i^1, g^3, s^1, l^1) = P(D= d^0) * P(I= i^1) * P(G= g^3 | D= d^0, I = i^1) * P(S=s^1 | I=i^1) * P(L= l^1 | G=g^3)$

Daphne Killer, Stanford University

$P(d^0, i^1, g^3, s^1, l^1)$ = 0.6 * 0.3 * 0.02 * 0.8* 0.01