Multimodal Affective Computing

Lecture 12: Neural Network Predictive Models

Louis-Philippe Morency
Jeffrey Girard

Originally developed with help from
Stefan Scherer and Tadas Baltrušaitis

Outline

- Discriminative Graphical Models
  - Logistic classifier
  - Conditional random fields
  - L1 and L2 regularization
- Neural Networks
  - Multi-layer perceptron
  - Back-propagation
  - Convolutional neural networks
- Evaluation methods and error measures

- Next week: Multimodal deep learning

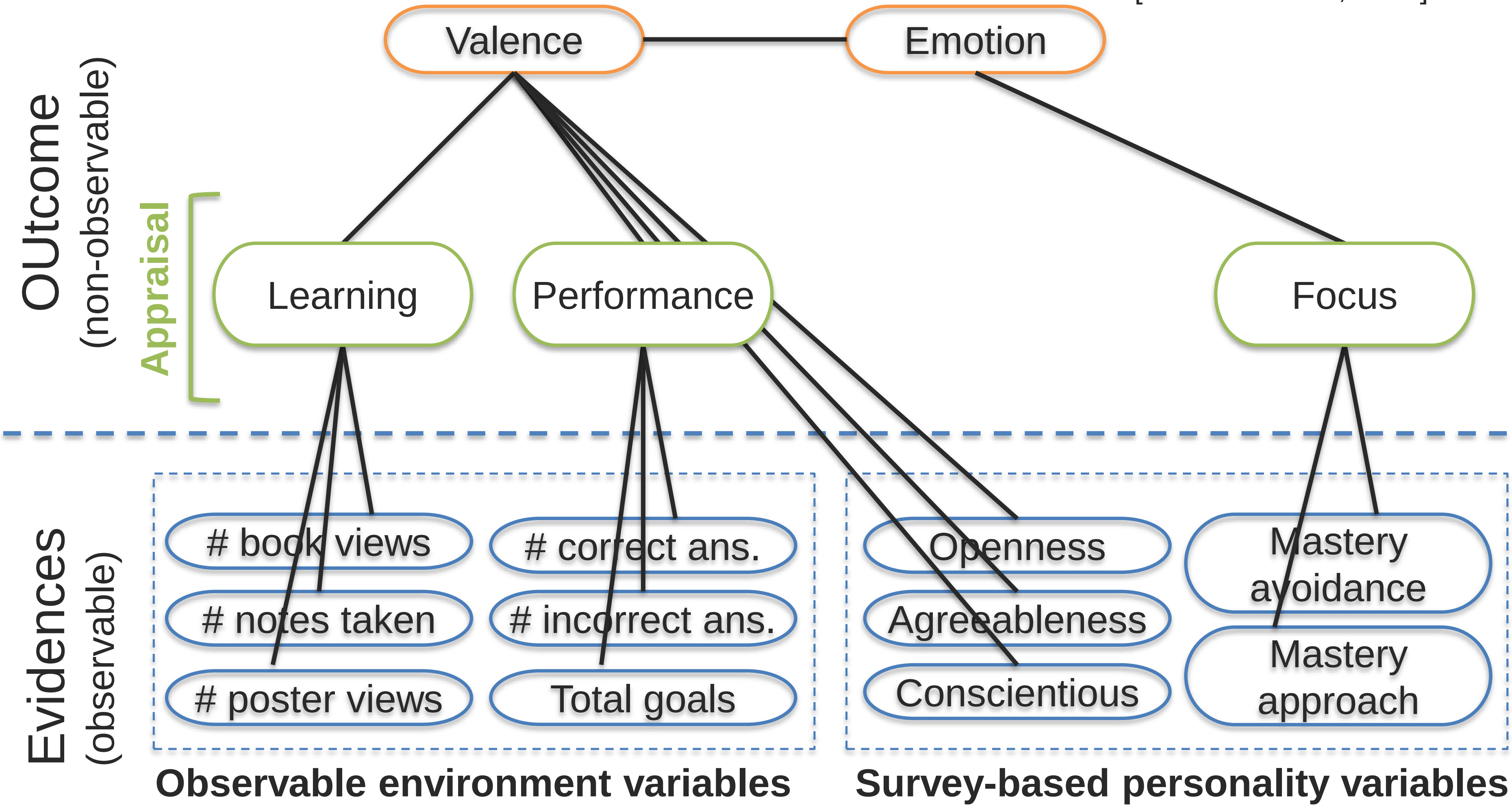
Upcoming Lectures

Classes	Tuesday	Thursday
Week 12 **4/02 & 4/04** ***midterm report***	Neural network predictive modeling • Multi-layer perceptron • Deep neural network • Convolutional neural network	Midterm presentations
Week 13 **4/09 & 4/11**	Multimodal deep learning • Multimodal representations • Attention and modality alignment • Temporal and multimodal fusion	NO CLASS
Week 14 **4/16 & 4/18**	Multimodal Behavior Generation • Guest lecture: Prof. Nakano • Generation based on user's attitude • Robot and virtual humans	Discussion (generation) • Jiang Liu • Ankit Shah
Week 15 **4/23 & 4/25**	Multimodal applications • Assessment in the clinical process • Biomarkers and behavioral indicators • Validation in the medical sciences	Discussion (applications) • Mingtong Zhang • Mahmoud Al Ismail
Week 16 **4/30 & 5/02** ***final report***	NO CLASS	Final presentations

Discriminative Graphical Models

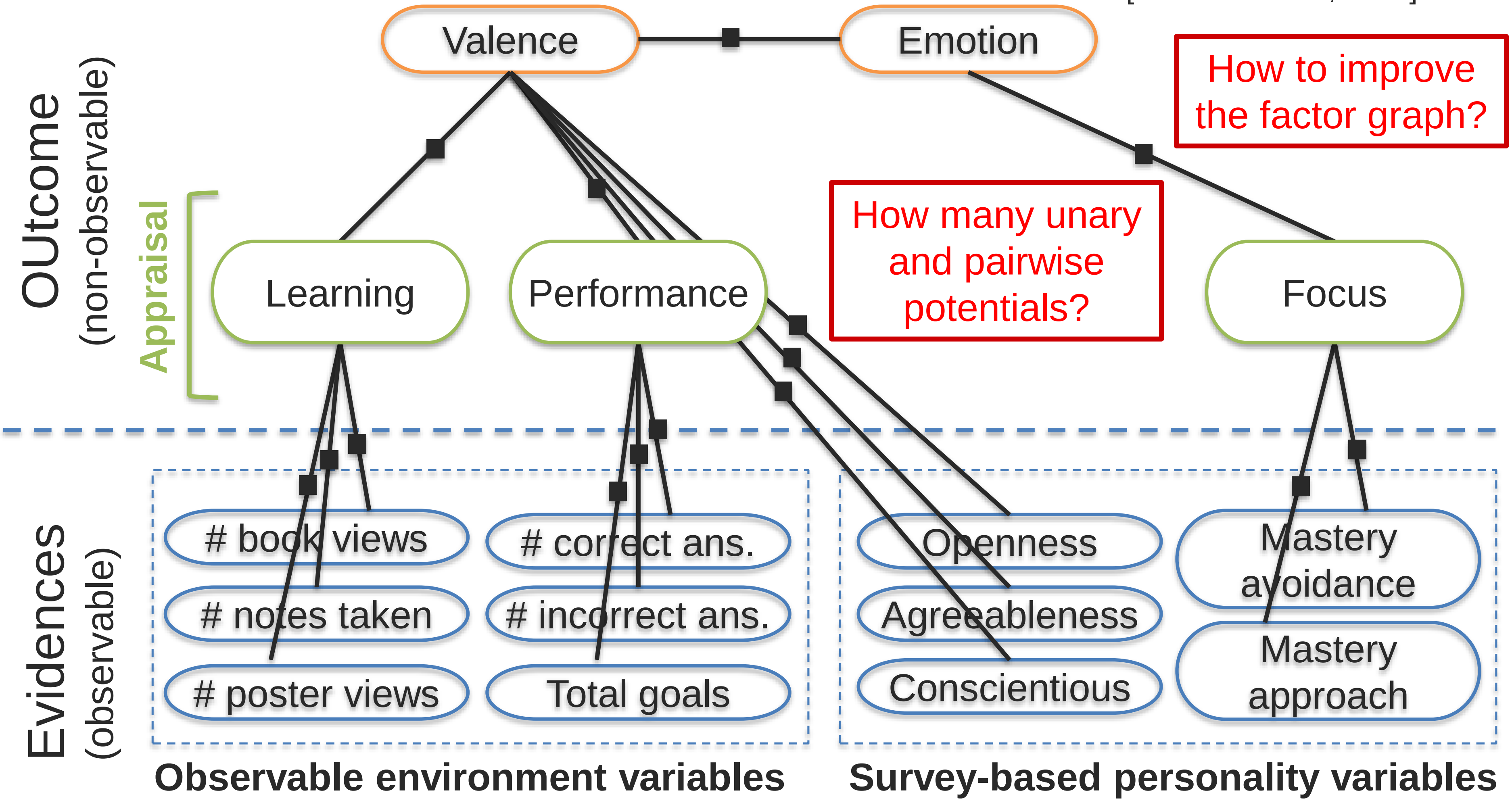
Example: Markov Random Field – Graphical Model

[Sabourin et al., 2011]



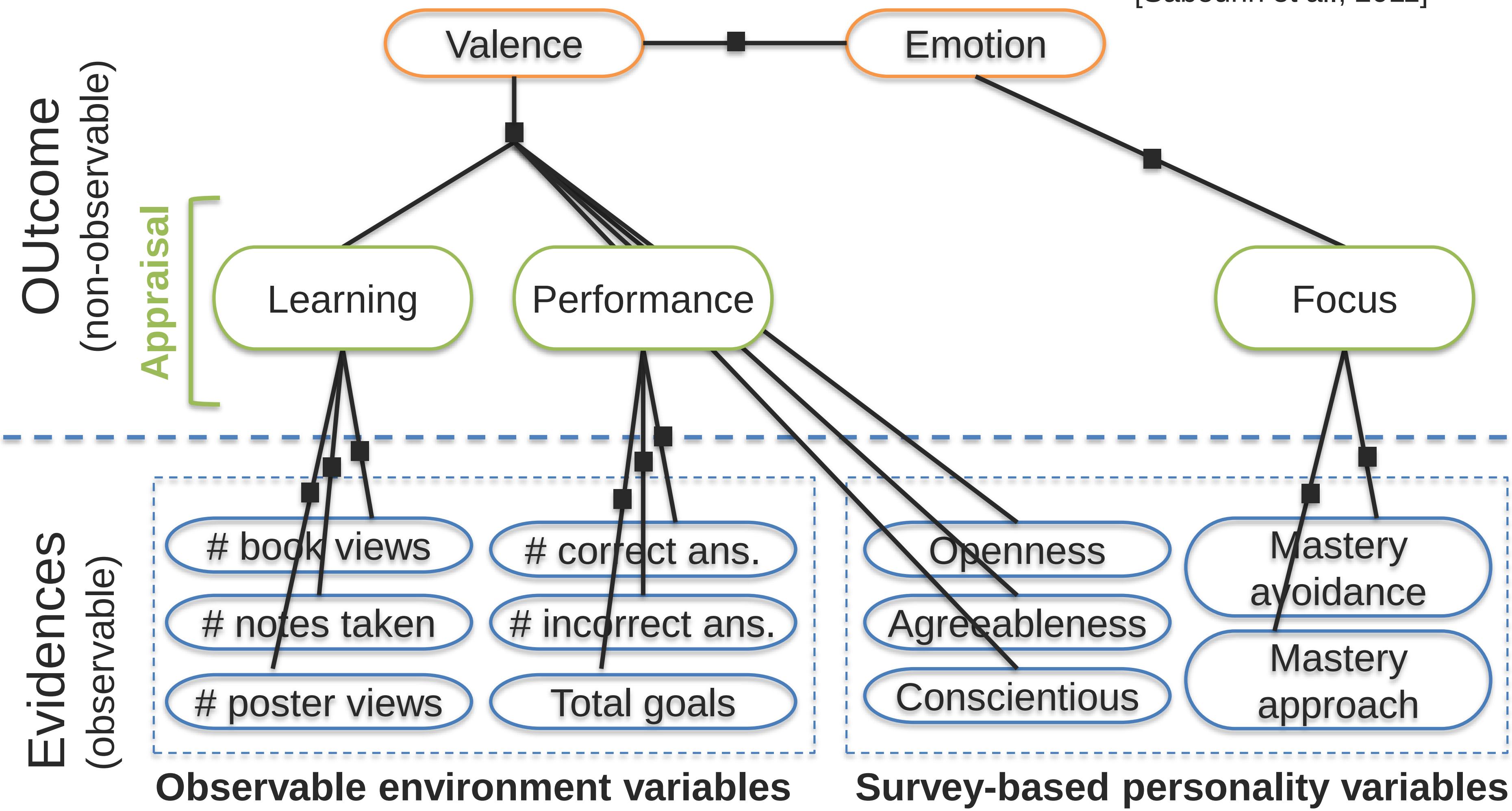
Example: Markov Random Field – Factor Graph

[Sabourin et al., 2011]



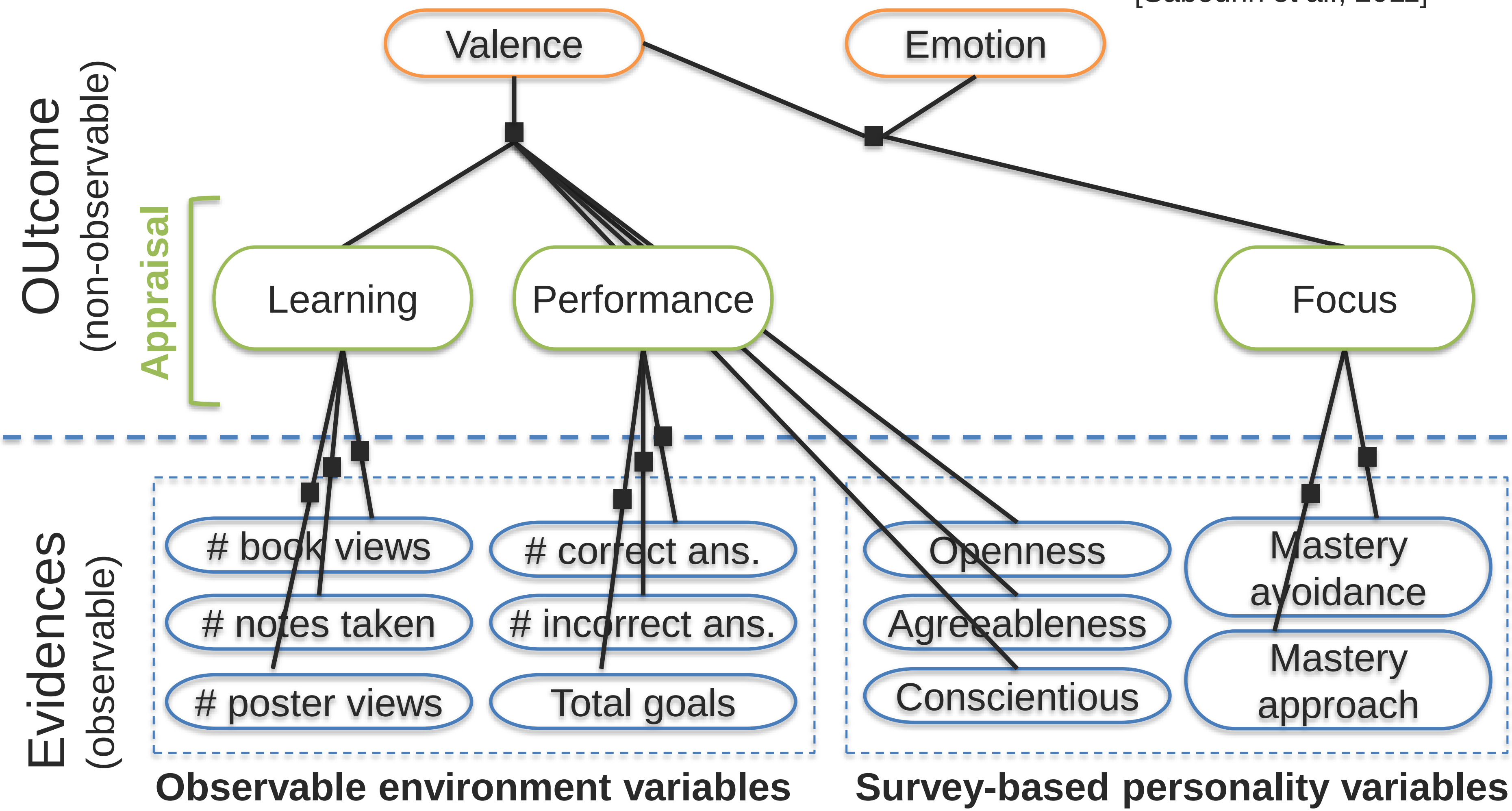
Example: Markov Random Field – Factor Graph

[Sabourin et al., 2011]

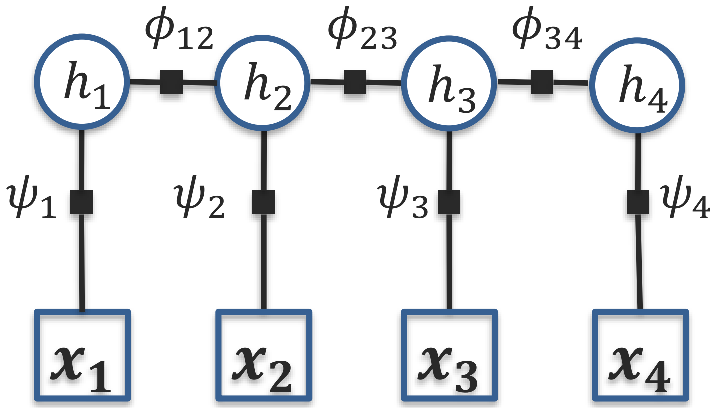


Example: Markov Random Field – Factor Graph

[Sabourin et al., 2011]



Generative versus Discriminative



Generative or Discriminative?

Answer: *It depends on the loss function!*

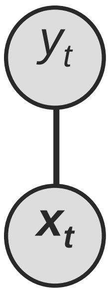
Generative loss function:
(joint probability)

$$L(\theta) = \sum_{j=1}^{N} P(\boldsymbol{h}^{(j)}, \boldsymbol{X}^{(j)}; \theta)$$

Discriminative loss function:
(conditional probability)

$$L(\theta) = \sum_{j=1}^{N} \log P\left(\boldsymbol{h}^{(j)} \middle| \boldsymbol{X}^{(j)}; \theta\right)$$

Discriminative Model: Logistic classifier



Label : {0:`Dominant`, 1:`Not-dominant`}

Observation vector: [speech-energy, gaze, turn-taking]

Score function:

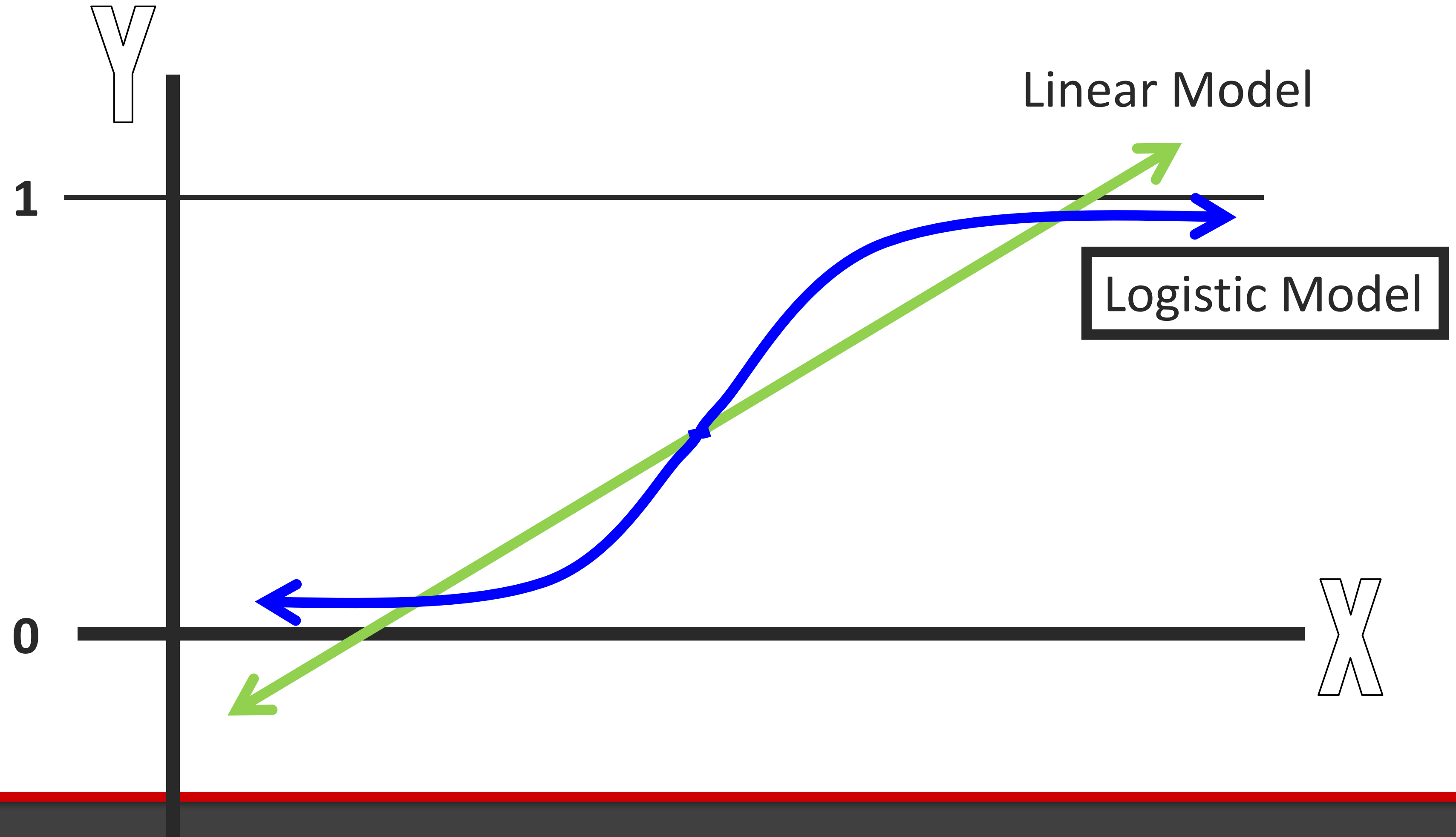
$$P(y_t = 1|\boldsymbol{x}_t) = \frac{1}{1 + \exp(-\boldsymbol{\theta}\boldsymbol{x}_t)}$$

Binary form

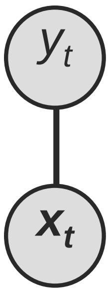
$$P(y_t = c|\boldsymbol{x}_t) = \frac{\exp(\boldsymbol{\theta}_c\boldsymbol{x}_t)}{\sum_{k=1}^{K}\exp(\boldsymbol{\theta}_k\boldsymbol{x}_t)}$$

Multinomial form

Comparing Linear and Logistic Models



Discriminative Model: Logistic classifier



Label : {0:`Dominant`, `1:Not-dominant`}

Observation vector: [speech-energy, gaze, turn-taking]

Score function:

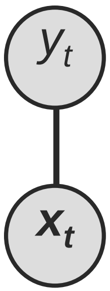
$$P(y_t = c|\boldsymbol{x}_t) = \frac{\exp(\boldsymbol{\theta}_c \boldsymbol{x}_t)}{\sum_{k=1}^{K} \boldsymbol{\theta}_c \boldsymbol{x}_t}$$

Familiar multinomial form

$$P(y_t|\boldsymbol{x}_t) = \frac{1}{\mathcal{Z}(\boldsymbol{x}_t)} \exp\left(\sum_{k=1}^{K} \theta_k f_k(y_t, \boldsymbol{x}_t)\right)$$

General form

Discriminative Model: Logistic classifier



Label : {0:`Dominant`, `1:Not-dominant`}

Observation vector: [speech-energy, gaze, turn-taking]

$$P(y_t|\boldsymbol{x}_t) = \frac{1}{\underbrace{Z(\boldsymbol{x}_t)}_{\text{Partition function}}} \exp\left(\sum_{k=1}^{K} \underbrace{\theta_k}_{\text{Weights}} \underbrace{f_k(y_t, \boldsymbol{x}_t)}_{\text{Feature functions}}\right) = \frac{1}{Z(\boldsymbol{x}_t)} \exp\left(\sum_{k=1}^{K} \phi_k(y_t, \boldsymbol{x}_t; \theta_k)\right)$$

Feature Functions

y_t Label : {0:`Dominant`, 1:`Not-dominant`} 2

$\boldsymbol{x_t}$ Observation vector: [speech-energy, gaze, turn-taking] 3

$$P(y_t|\boldsymbol{x_t}) = \frac{1}{Z(\boldsymbol{x_t})}\exp\left(\sum_{k=1}^{K}\theta_k f_k(y_t,\boldsymbol{x_t})\right)$$

$\boxed{K = 6}$

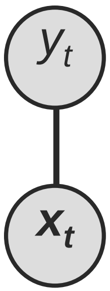
$$f_0(y_t,\boldsymbol{x_t}) = \begin{cases} x_{t0}, & y_t = 0 \\ 0, & Otherwise \end{cases} \qquad f_1(y_t,\boldsymbol{x_t}) = \begin{cases} x_{t0}, & y_t = 1 \\ 0, & Otherwise \end{cases}$$

$$f_2(y_t,\boldsymbol{x_t}) = \begin{cases} x_{t1}, & y_t = 0 \\ 0, & Otherwise \end{cases} \qquad f_3(y_t,\boldsymbol{x_t}) = \begin{cases} x_{t1}, & y_t = 1 \\ 0, & Otherwise \end{cases}$$

$$f_4(y_t,\boldsymbol{x_t}) = \begin{cases} x_{t2}, & y_t = 0 \\ 0, & Otherwise \end{cases} \qquad f_5(y_t,\boldsymbol{x_t}) = \begin{cases} x_{t2}, & y_t = 1 \\ 0, & Otherwise \end{cases}$$

	y_t=0	y_t=1
x_{t0}	θ_0	θ_1
x_{t1}	θ_2	θ_3
x_{t2}	θ_4	θ_5

Partition Function: Normalizing Constant



Label : {0:`Dominant`, 1:`Not-dominant`}

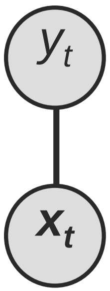
Observation vector: [speech-energy, gaze, turn-taking]

$$P(y_t|\boldsymbol{x}_t) = \frac{1}{Z(\boldsymbol{x}_t)} \exp\left(\sum_{k=1}^{K} \theta_k f_k(y_t, \boldsymbol{x}_t)\right)$$

$$Z(\boldsymbol{x}_t) = \sum_{y'=0}^{|Y|} \exp\left(\sum_{k=1}^{K} \theta_k f_k(y', \boldsymbol{x}_t)\right)$$

So that $P(y_t|\boldsymbol{x}_t)$ stays between 0 and 1.

Training and Loss Function



Label : {0:`Dominant`, `1:Not-dominant`}

Observation vector: [speech-energy, gaze, turn-taking]

$$P(y_t|\boldsymbol{x}_t) = \frac{1}{Z(\boldsymbol{x}_t)} \exp\left(\sum_{k=1}^{K} \theta_k f_k(y_t, \boldsymbol{x}_t)\right)$$

Loss function: Conditional log likelihood

$$L(\theta) = \sum_{j=1}^{N} \log P\left(\boldsymbol{y}^{(j)} \middle| \boldsymbol{X}^{(j)}; \theta\right) \; - R(\theta)$$

Regularization

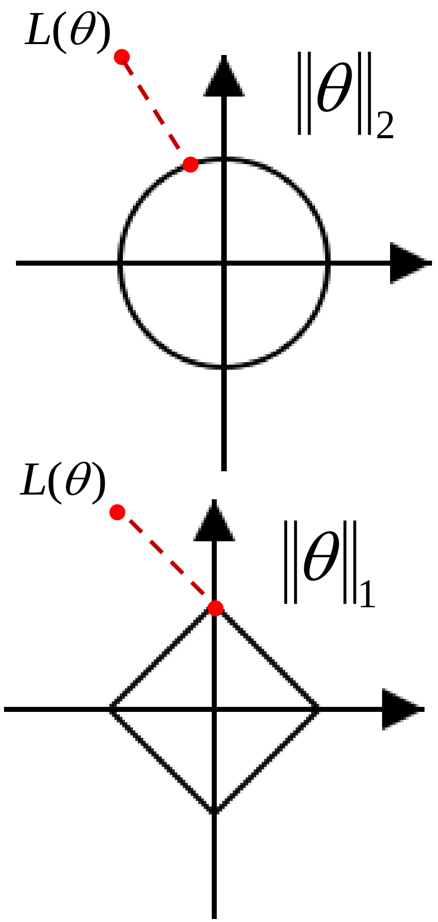
$$L(\theta) = \sum_{j=1}^{N} \log P\left(\boldsymbol{y}^{(j)} \middle| \boldsymbol{X}^{(j)}; \theta\right) - R(\theta)$$

- L-2 Norm (Gaussian prior):

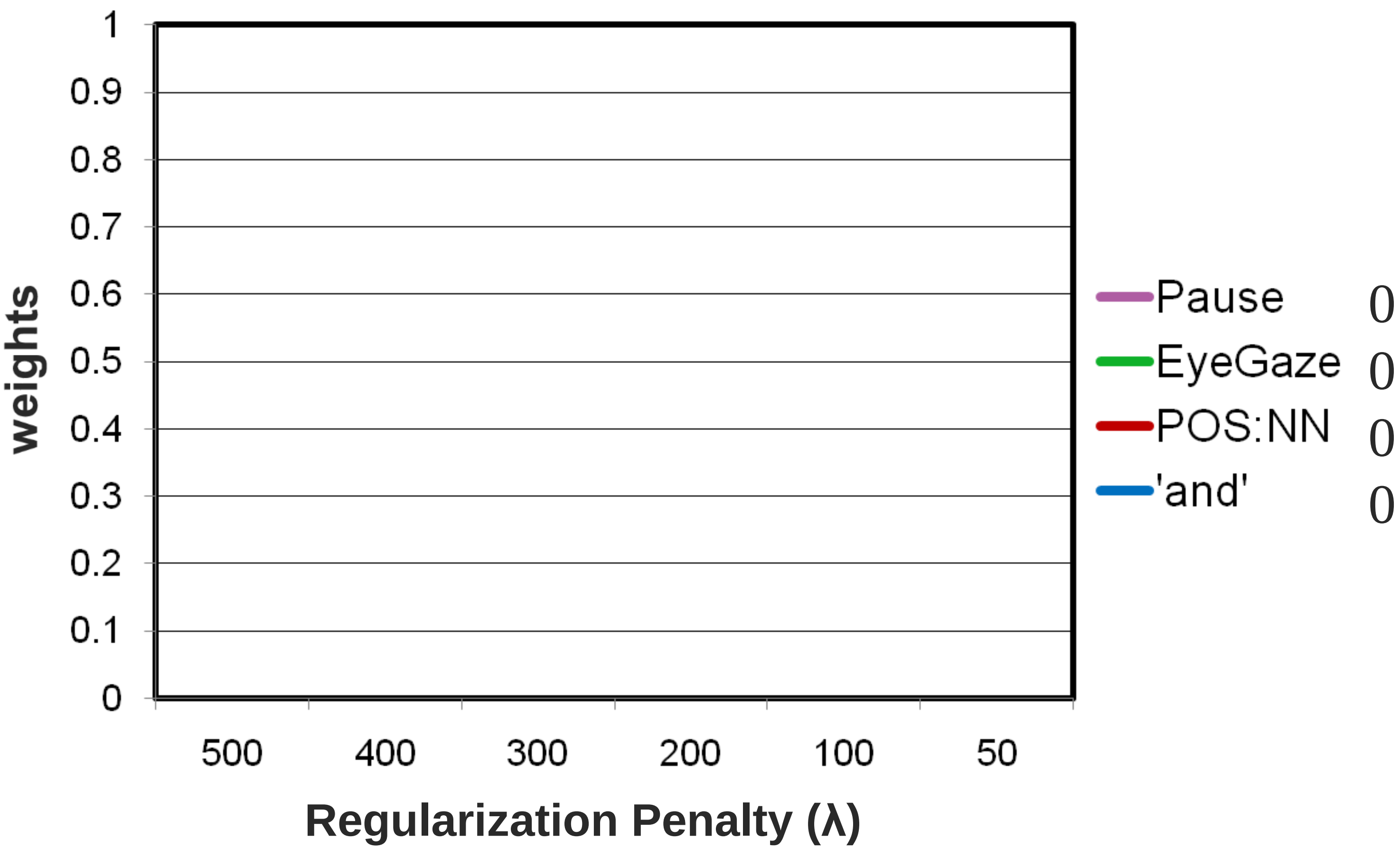
$$R(\theta) = \lambda \|\theta\|_2$$

- L-1 Norm (Laplacian prior):

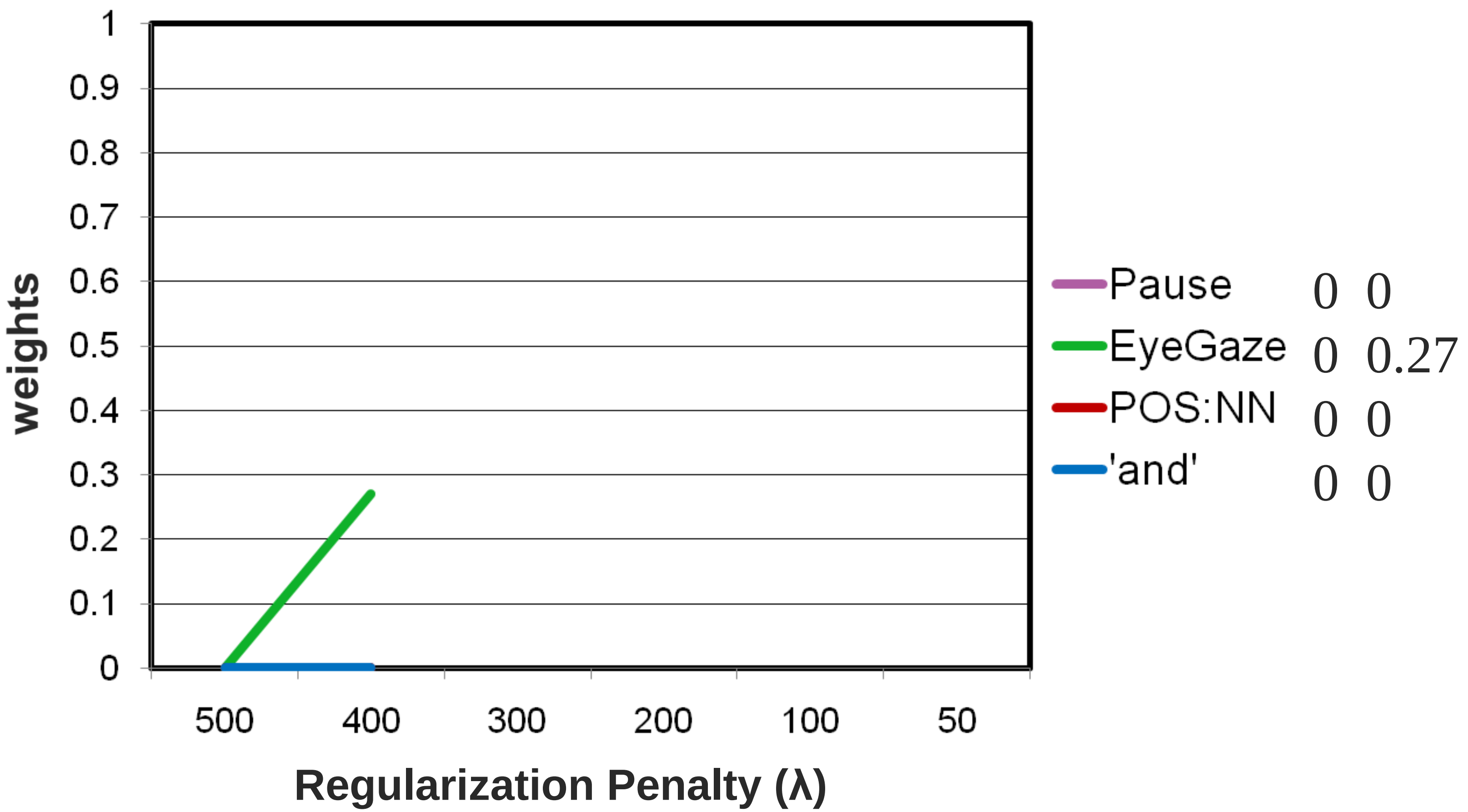
$$R(\theta) = \lambda \|\theta\|_1$$



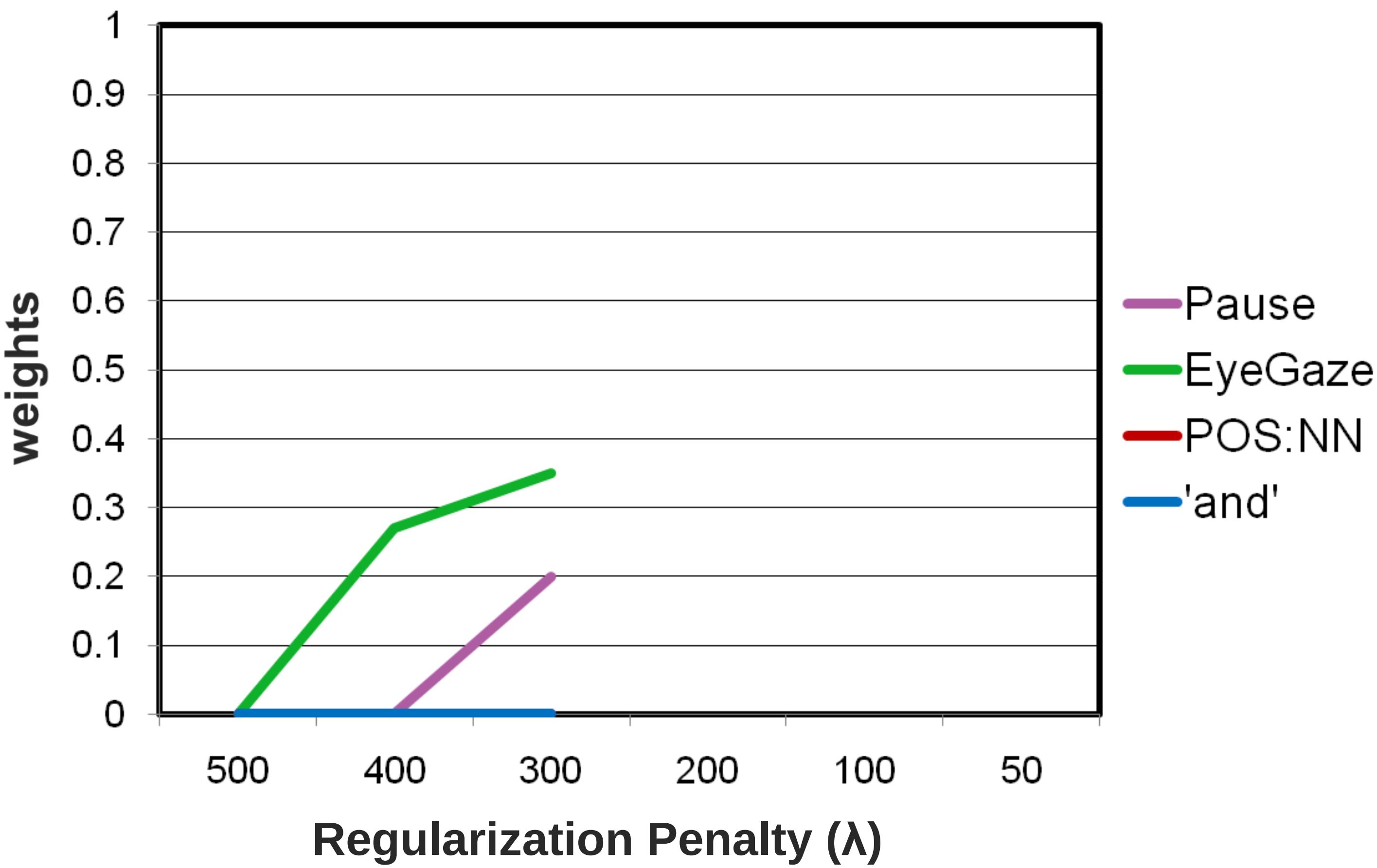
Sparse Feature Ranking



Sparse Feature Ranking

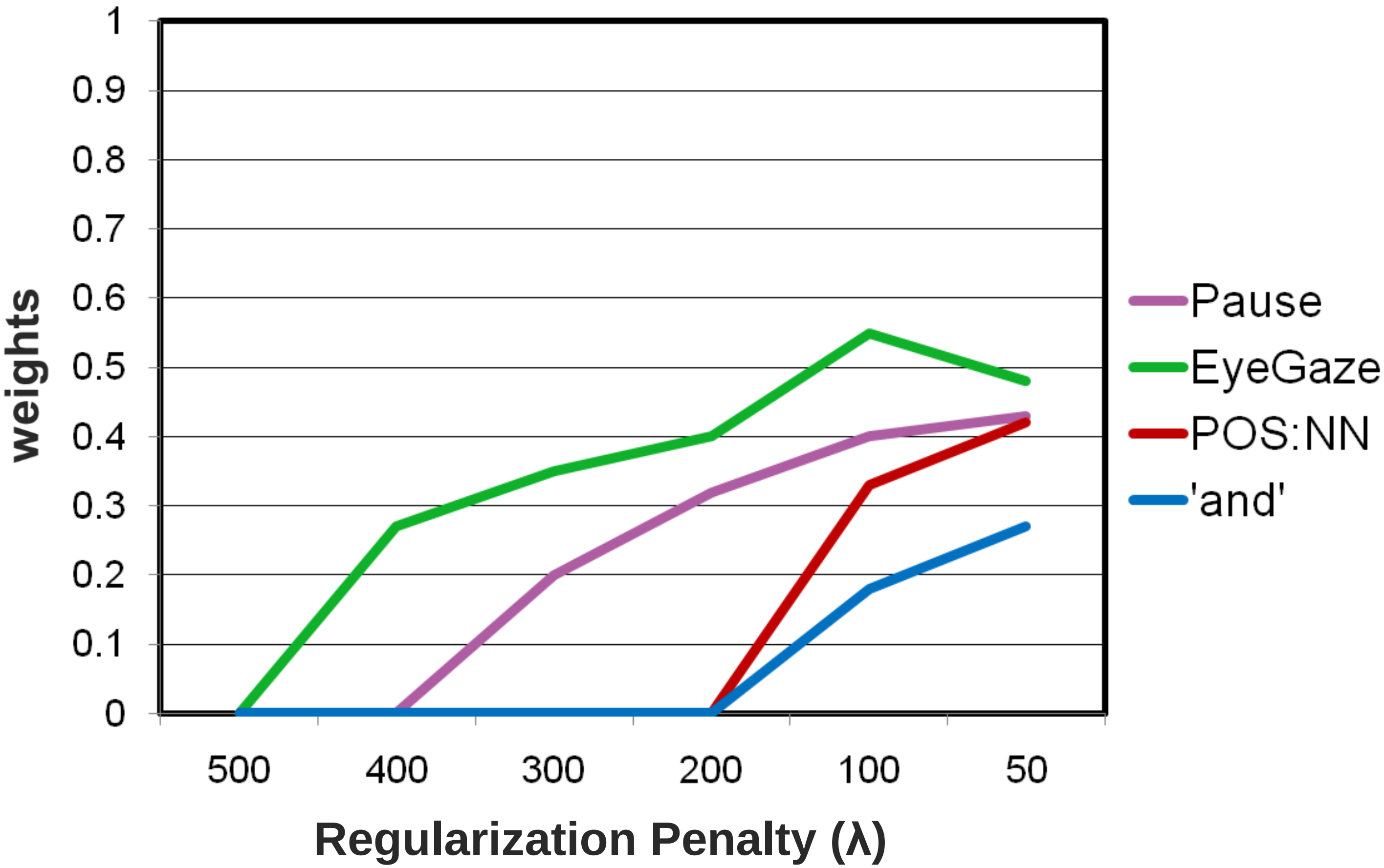


Sparse Feature Ranking



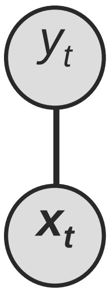
0	0	0.20
0	0.27	0.35
0	0	0
0	0	0

Sparse Feature Ranking



0	0	0.20	0.40
0	0.27	0.35	0.55
0	0	0	0.33
0	0	0	0.18

LASSO and ElasticNet



Label : {0:`Dominant`, `1:Not-dominant`}

Observation vector: [speech-energy, gaze, turn-taking]

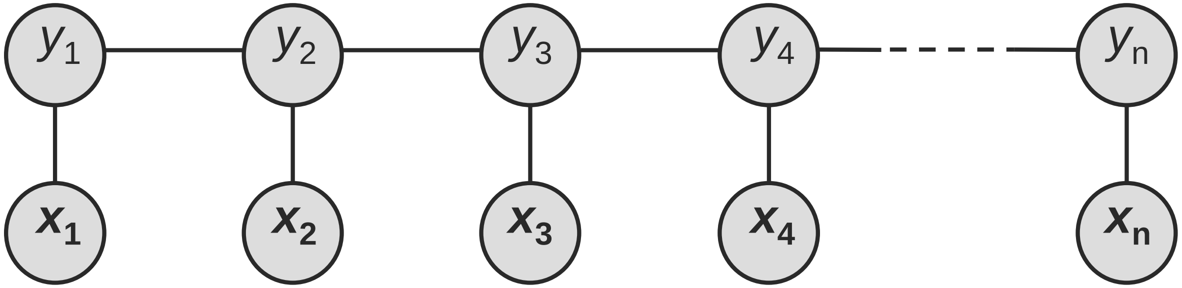
Lasso loss function: squared loss with L1 regularization

$$L(\theta) = \sum_{j=1}^{N} \left(y_j - f\left(\boldsymbol{x}_j; \theta\right)\right)^{\mathbf{2}} - \lambda \|\theta\|_1$$

ElasticNet: squared loss with L1 and L2 regularization

$$L(\theta) = \sum_{j=1}^{N} \left(y_j - f\left(\boldsymbol{x}_j; \theta\right)\right)^{\mathbf{2}} - \lambda \|\theta\|_1 - \lambda \|\theta\|_2$$

Conditional Random Fields (CRFs) [McCallum 2001]



$$P(\boldsymbol{y}|\boldsymbol{X}) = \frac{1}{Z(\boldsymbol{X})} \exp\left(\sum_{k=1}^{K} \theta_k f_k(y_t, \boldsymbol{x}_t) + \sum_{l=1}^{L} \lambda_l g_l(y_t, y_{t-1}) \right)$$

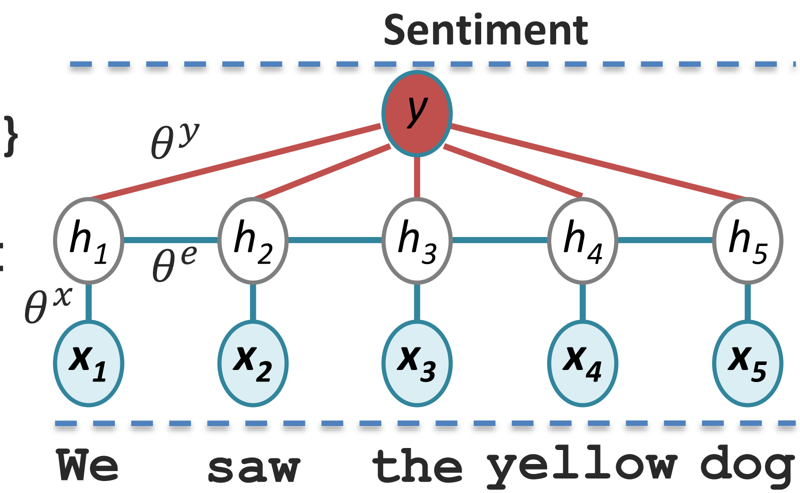
Hidden Conditional Random Field

Sequence label:

$y \in \mathcal{Y}$ for example, $\mathcal{Y}$: **{positive, negative}**

Latent variables with shared hidden states:

$\boldsymbol{h} = \{h_1, h_2, h_3, \ldots, h_t\}$ where $h_t \in \mathcal{H}$



$$p(y, \boldsymbol{h} \mid \mathbf{x}; \boldsymbol{\theta}) = \frac{1}{\mathcal{Z}(\boldsymbol{x}; \boldsymbol{\theta})} \exp\left\{\sum_t \boldsymbol{\theta}^x \cdot f^x(h_t, \mathbf{x}_t) + \underbrace{\sum_t \boldsymbol{\theta}^e \cdot f^e(h_t, h_{t-1}, y)} + \underbrace{\sum_t \boldsymbol{\theta}^y \cdot f^y(y, \boldsymbol{h}_t)}_{\text{Shared hidden states}}\right\}$$

$$p(y \mid \boldsymbol{x}; \boldsymbol{\theta}) = \sum_{\boldsymbol{h}} p($$

- **Inference is tractable: $O(YH^2T)$**
  - **Linear in sequence length T !**
- **Parameter learning ($\theta^x, \theta^e, \theta^y$):**
  - **Gradient descent or L-BFGS**

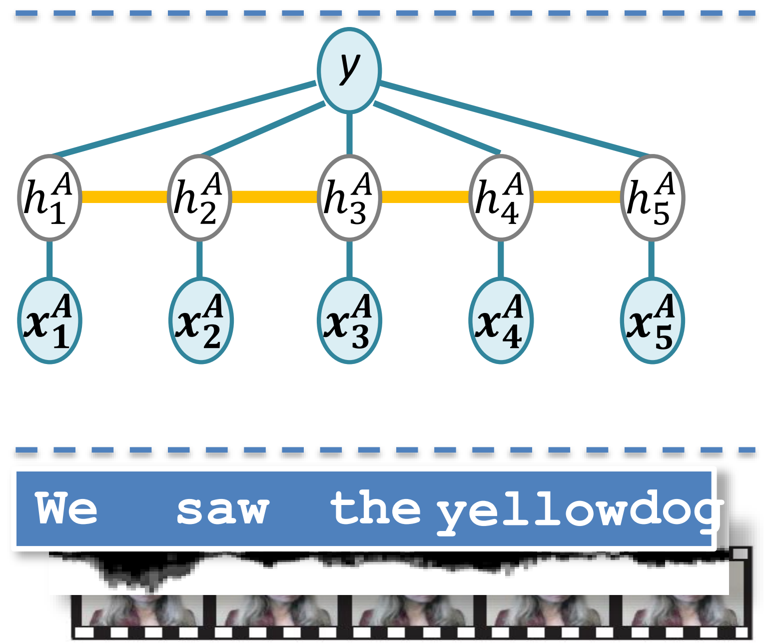
Learning Multimodal Structure

Modality-***private*** structure

- Internal grouping of observations

Modality-***shared*** structure

- Interaction and synchrony



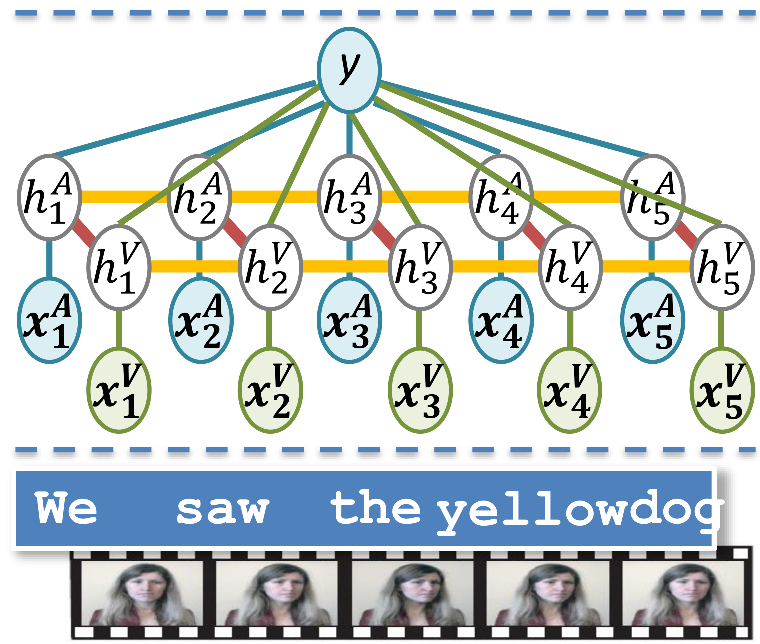
Multi-view Latent Variable Discriminative Models

Modality-*private* structure

- Internal grouping of observations

Modality-*shared* structure

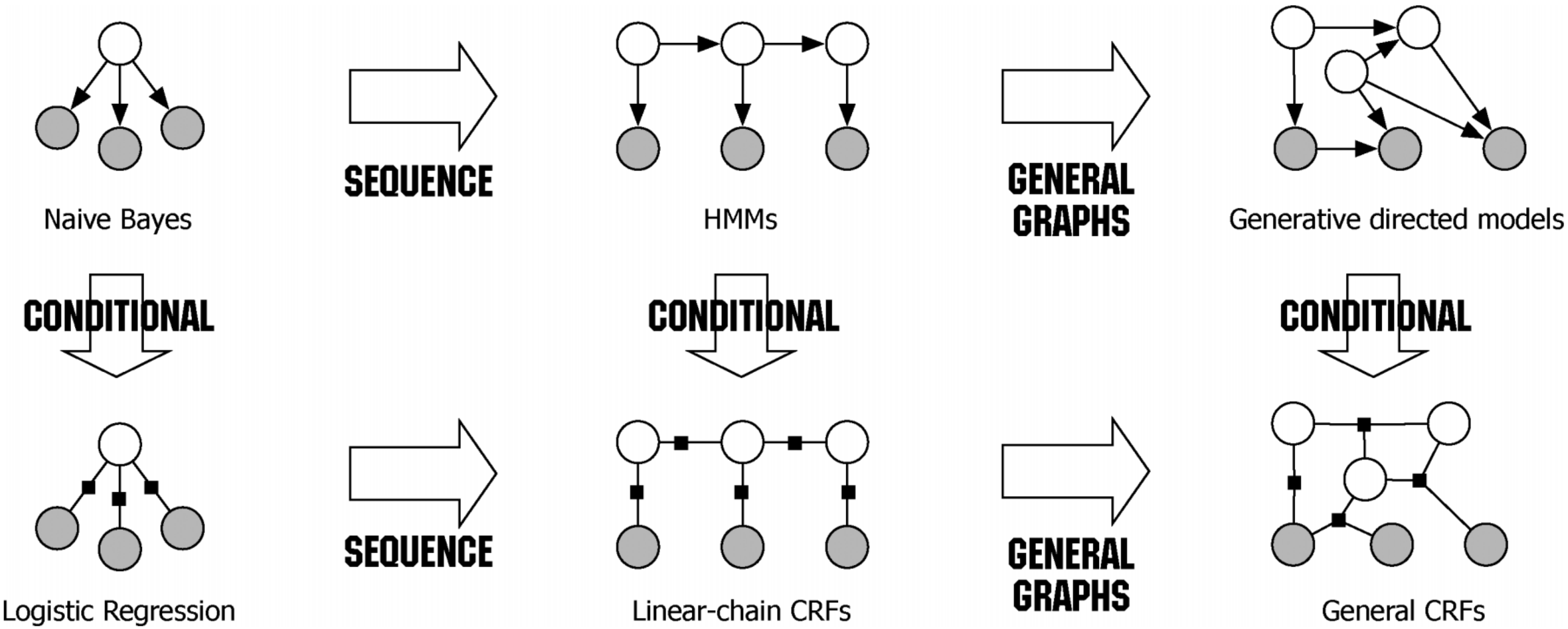
- Interaction and synchrony



$$p(y|\,\boldsymbol{x}^A, \boldsymbol{x}^V; \boldsymbol{\theta}) = \sum_{\boldsymbol{h}^A, \boldsymbol{h}^V} p\left(y, \boldsymbol{h}^A, \boldsymbol{h}^V \middle| \boldsymbol{x}^A, \boldsymbol{x}^V; \boldsymbol{\theta}\right)$$

➢ Approximate inference using loopy-belief

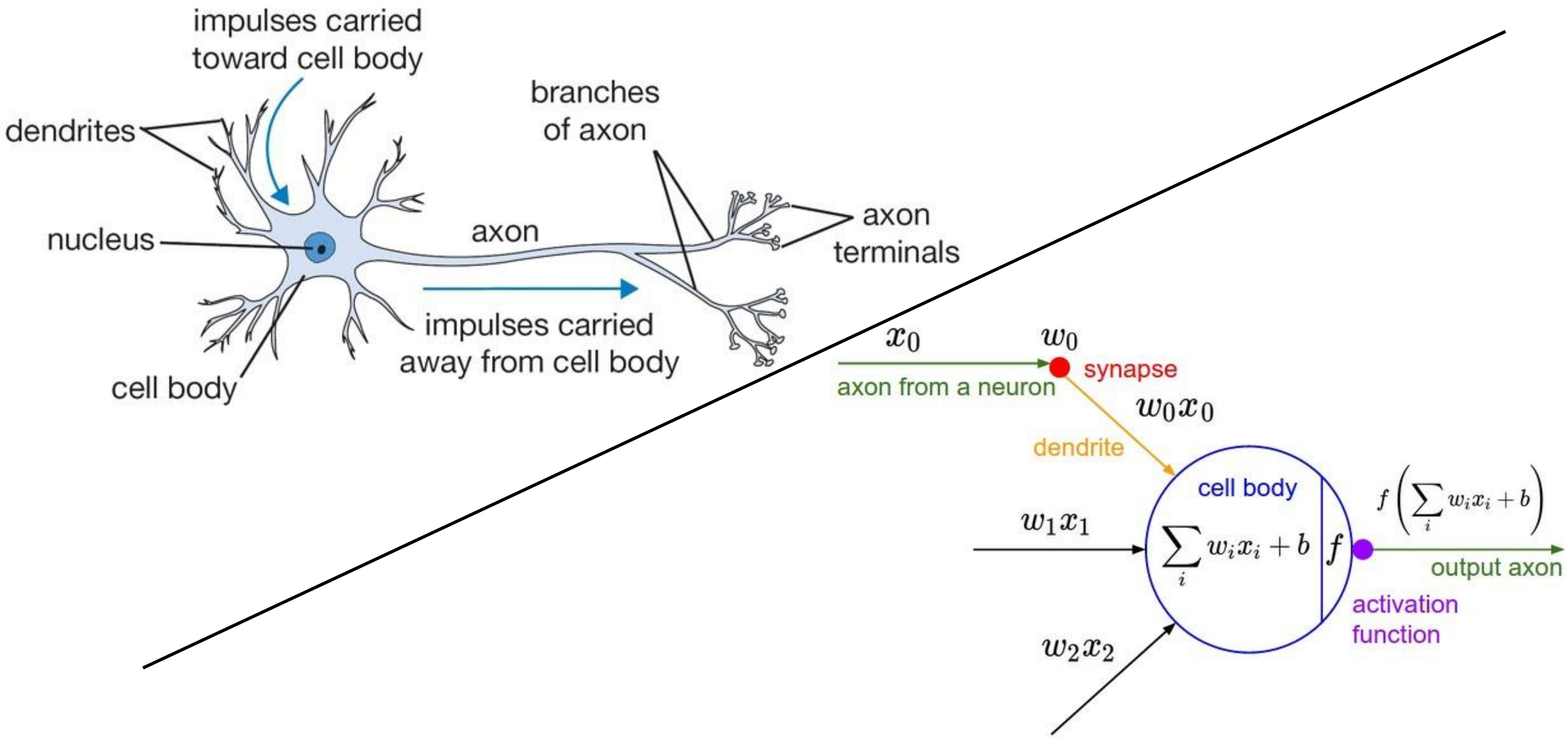
Recap of generative vs discriminative



Basic Concepts: Neural Networks

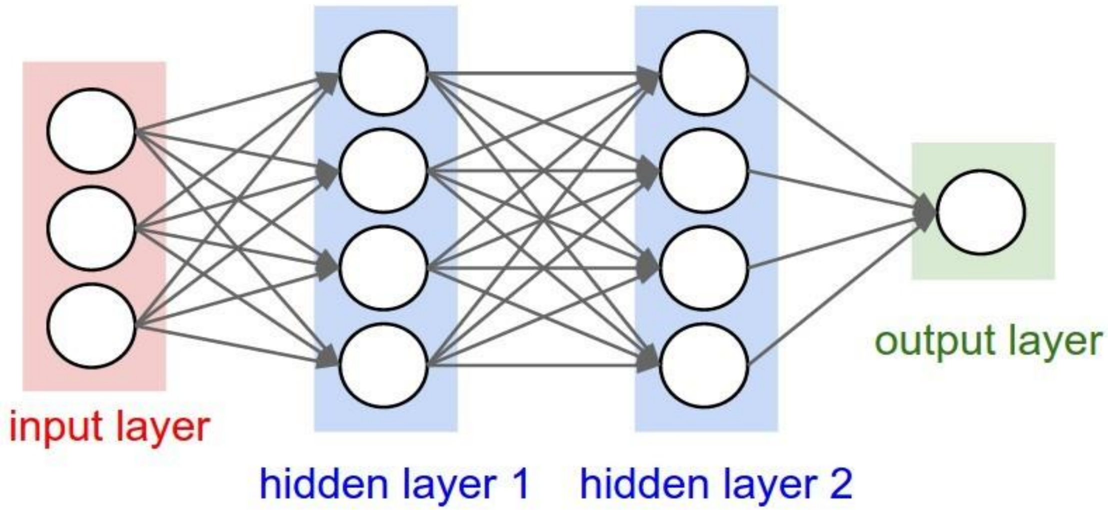
Neural Networks – inspiration

- Made up of artificial neurons



Neural Networks – score function

- Made up of artificial neurons
  - Linear function (dot product) followed by a nonlinear activation function
- Example a Multi Layer Perceptron



Basic NN building block

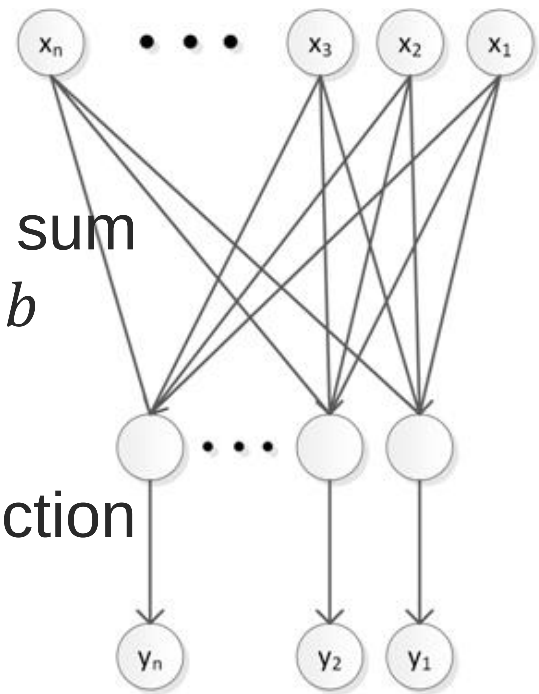
- Weighted sum followed by an activation function

Input

Weighted sum
$Wx + b$

Activation function

Output



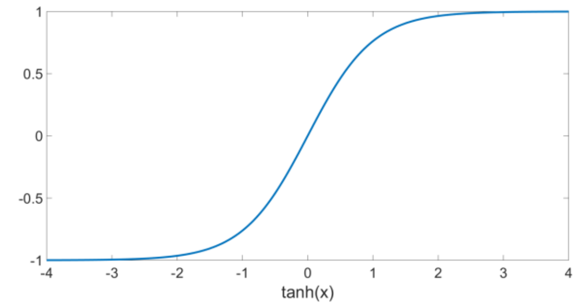
This part of the neural network is very similar to another predictive model we studied. Which one?

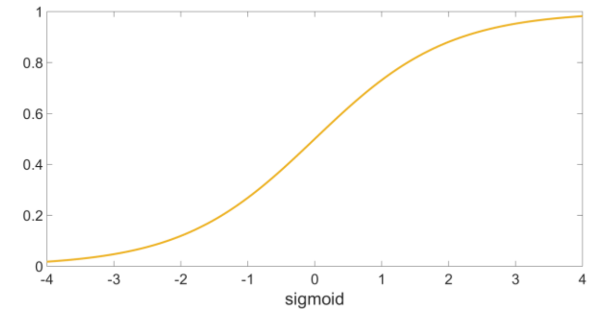
Linear classifier

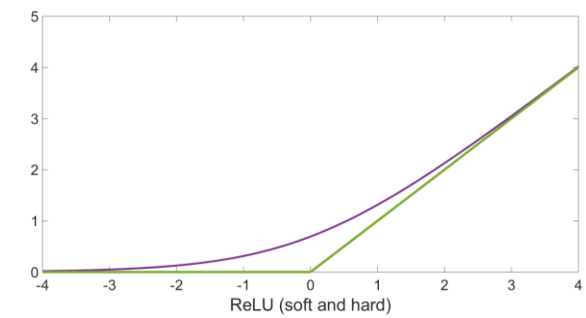
$$y = f(Wx + b)$$

Neural Networks – activation function

- $f(x) = \tanh(x)$
- Sigmoid - $f(x) = (1 + e^{-x})^{-1}$
- Linear – $f(x) = ax + b$
- ReLU $f(x) = \max(0, x) \sim \log(1 + \exp(x))$
  - Rectifier Linear Units
  - Faster training - no gradient vanishing
  - Induces sparsity







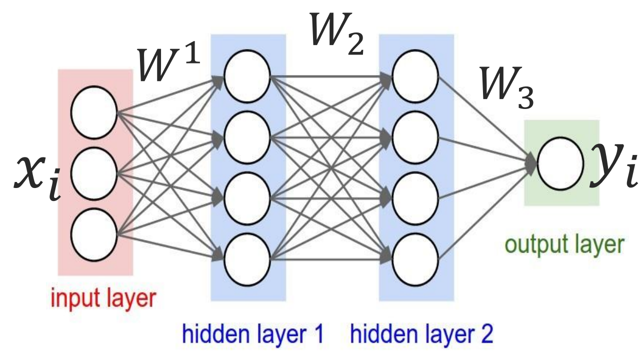
Multi-Layer Feedforward Network

Activation functions (individual layers)

$$f_{1;W_1}(x) = \sigma(W_1 x + b_1)$$

$$f_{2;W_2}(x) = \sigma(W_2 x + b_2)$$

$$f_{3;W_3}(x) = \sigma(W_3 x + b_3)$$



Score function

$$y_i = f(x_i) = f_{3;W_3}(f_{2;W_2}(f_{1;W_1}(x_i)))$$

Loss function (e.g., Euclidean loss)

$$L_i = (f(x_i) - y_i)^2 = (f_{3;W_3}(f_{2;W_2}(f_{1;W_1}(x_i))))^2$$

Neural Networks inference and learning

- Inference (Testing)
  - Use the score function ($y = f(\boldsymbol{x}; W)$)
  - Have a trained model (parameters W)
- Learning model parameters (Training)
  - Loss function (L)
  - Gradient
  - Optimization

Gradient descent algorithm for MLP

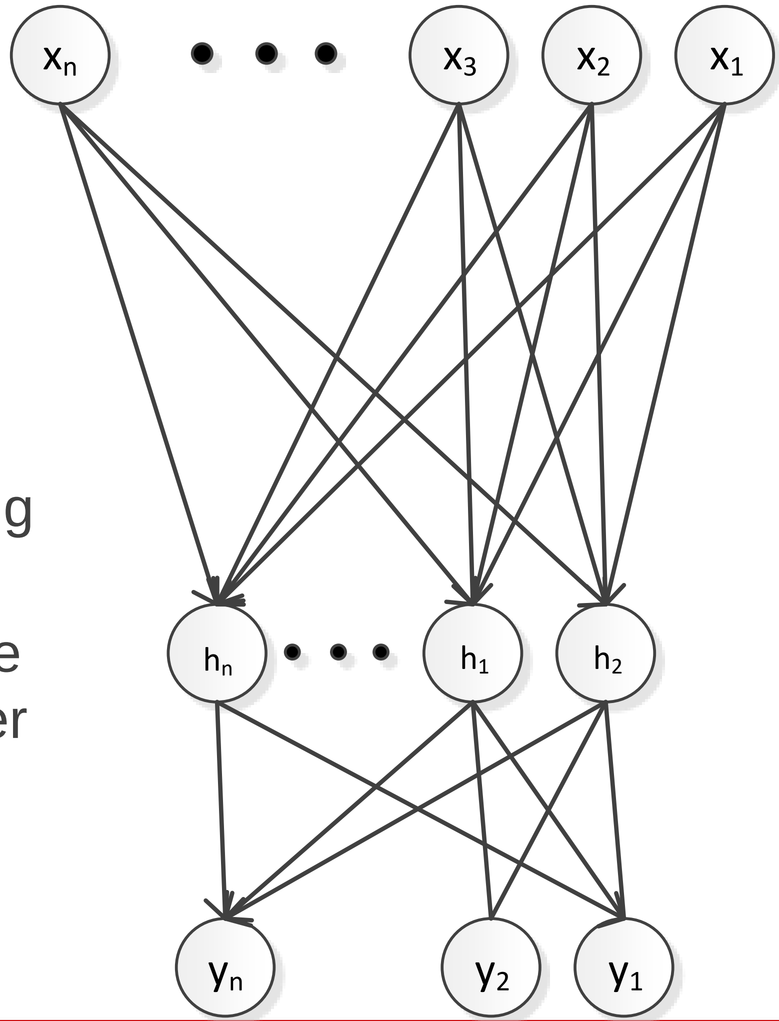
- All layers are differentiable
- Start from random weight values
- Iteratively adjust weights in the direction that minimises the error

```
while not converged:
    # compute gradients
    weights_grad = compute_gradient(loss_fun, data, weights)
    # perform parameter update
    weights += - step_size * weights_grad
```

Training the model efficiently

- Backpropagation - propagate the error backward
  - An efficient model of gradient descent, nothing more nothing less

- Forward propagate from input to output through all the layers keeping track of intermediate results
- Compute error at the final layer, use this to compute error at hidden layer (continue to input)



Backpropagation Algorithm (efficient gradient)

Forward pass

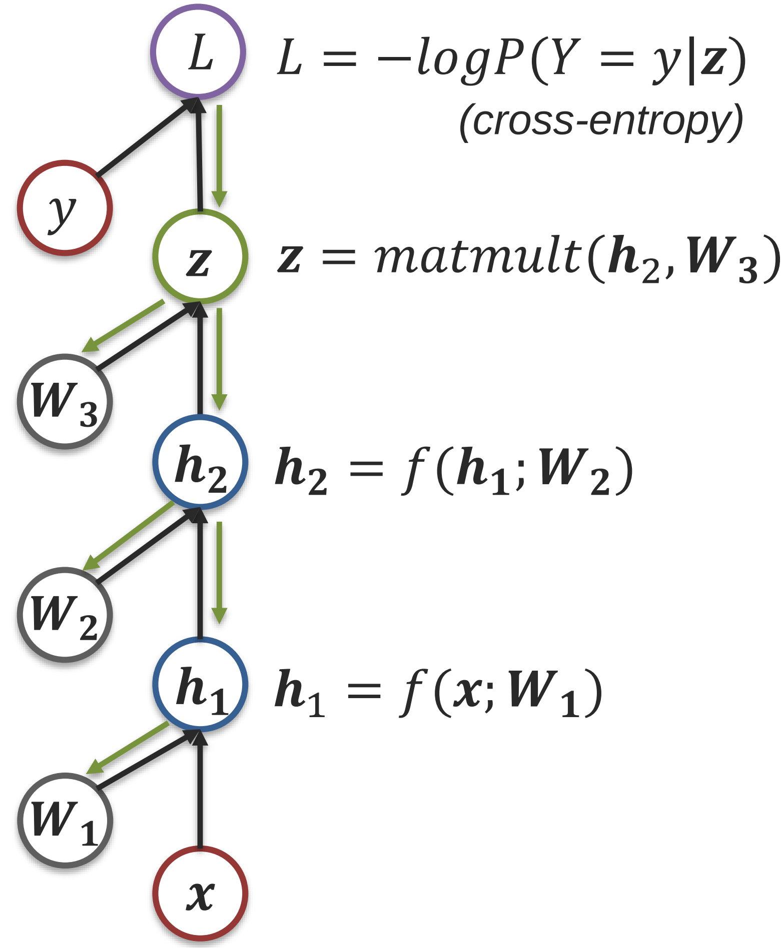
- Following the graph topology, compute value of each unit

Backpropagation pass

- Initialize output gradient = 1
- Compute “local” Jacobian matrix using values from forward pass
- Use the chain rule:

Gradient = “local” Jacobian x “backprop” gradient

- Why is this rule important?



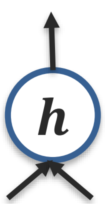
Computational Graph: Multi-layer Feedforward Network

Computational unit:

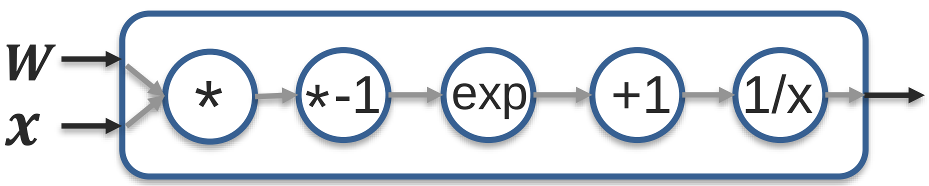
$$\boldsymbol{h} = f(\boldsymbol{x}; \boldsymbol{W})$$

- Multiple input
- One output
- Vector/tensor

- Sigmoid unit:

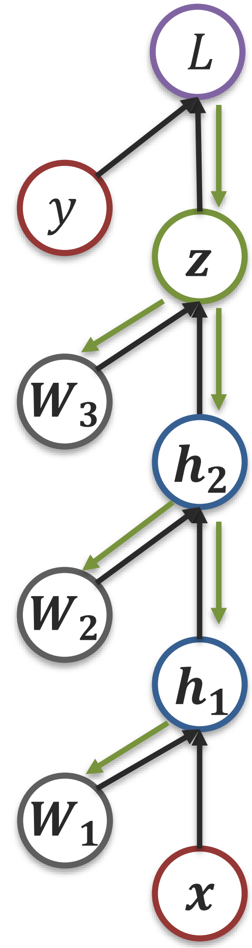


$$h_j = (1 + e^{-W_j \boldsymbol{x}})^{-1}$$



Differentiable "unit" function!

(or close approximation to compute "local Jacobian)



$L = -logP(Y = y|\boldsymbol{z})$
(cross-entropy)

$\boldsymbol{z} = matmult(\boldsymbol{h}_2, \boldsymbol{W}_3)$

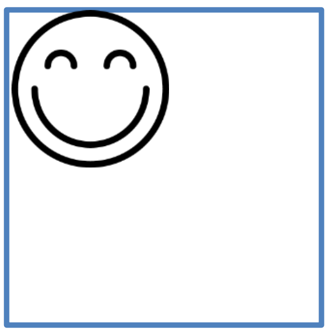
$\boldsymbol{h}_2 = f(\boldsymbol{h}_1; \boldsymbol{W}_2)$

$\boldsymbol{h}_1 = f(\boldsymbol{x}; \boldsymbol{W}_1)$

Convolutional Neural Network

A Shortcoming of MLP

2 Data Points – detect which head is up!
Easily modeled using one neuron.
What is the best neuron to model this?

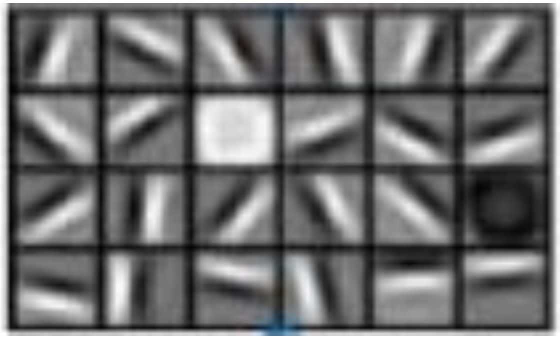


This head may or may not be up – what happened?

Solution: instead of modeling the entire image, model the important region.

Why not just use an MLP for images?

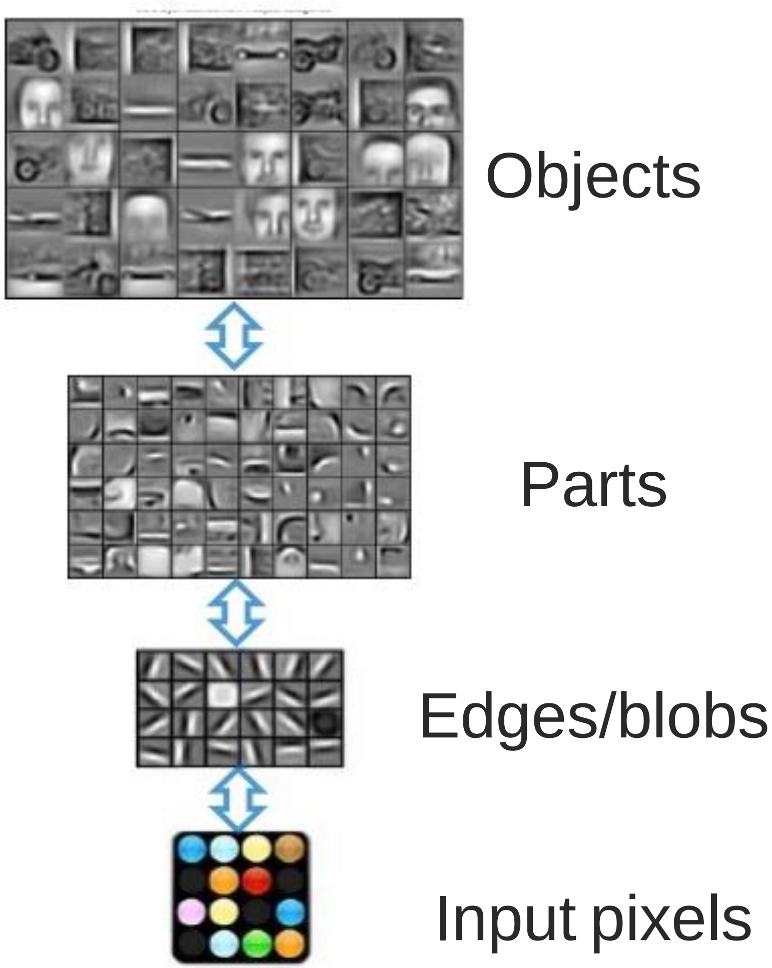
- MLP connects each pixel in an image to each neuron
- Does not exploit redundancy in image structure
  - Detecting edges, blobs
  - Don’t need to treat the top left of image differently from the center



- Too many parameters
  - For a small 200×200 pixel RGB image the first matrix would have $120000 \times n$ parameters for the first layer alone

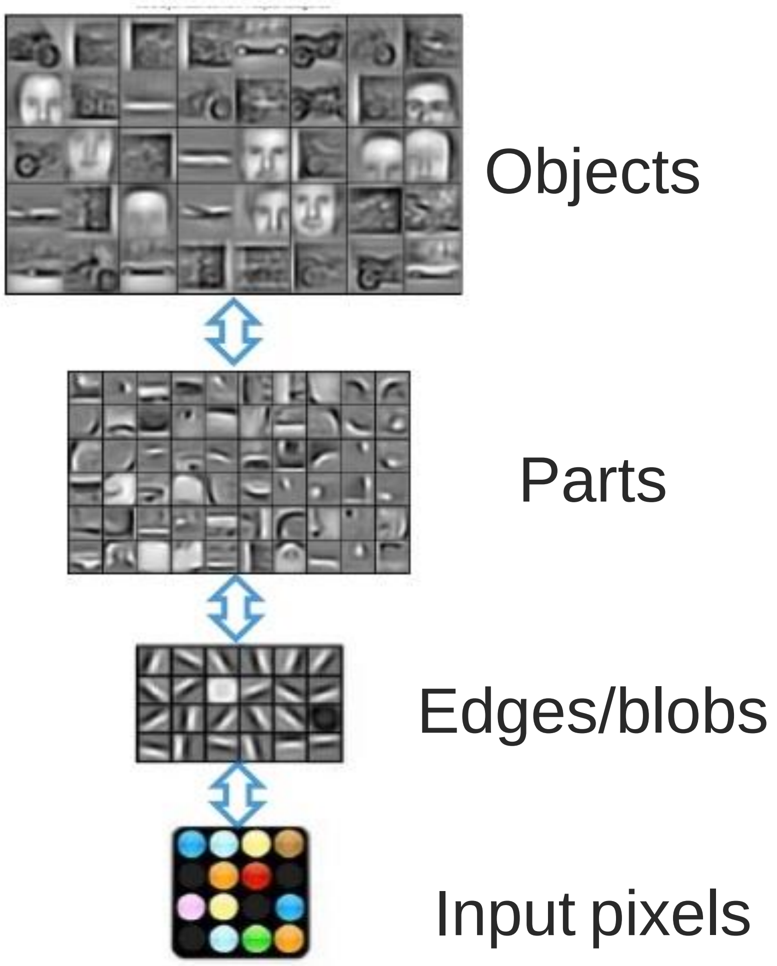
Feature hierarchy intuition

- Each layer extracts features from output of the previous layer
- Features learn to be tailored to the problem (at least that's the idea)



Building blocks

- Function $\mathbf{y} = f(\boldsymbol{x})$
  - Differentiable (or locally differentiable)
  - Non-linear
- Desired
  - Efficient
  - Most often mapping from a vector to a vector



Fully connected layer

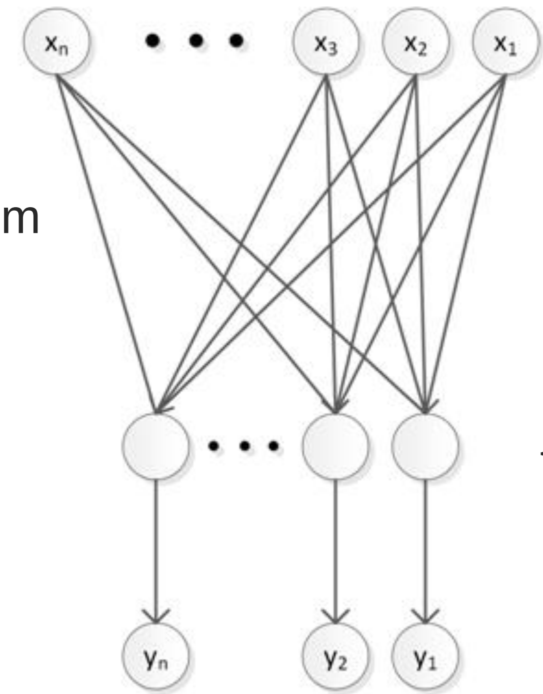
- Weighted sum followed by an activation function (saw this before)

Input

Weighted sum

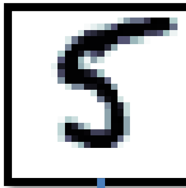
Activation

Output



$$y_i = f(\boldsymbol{w}_i^T \boldsymbol{x} + b_i)$$

$$\begin{cases} f(x) = \tanh(x) \\ f(x) = \max(0, x) \\ f(x) = (1 + e^{-x})^{-1} \\ f(x) = \mathrm{softmax}(x) \end{cases}$$



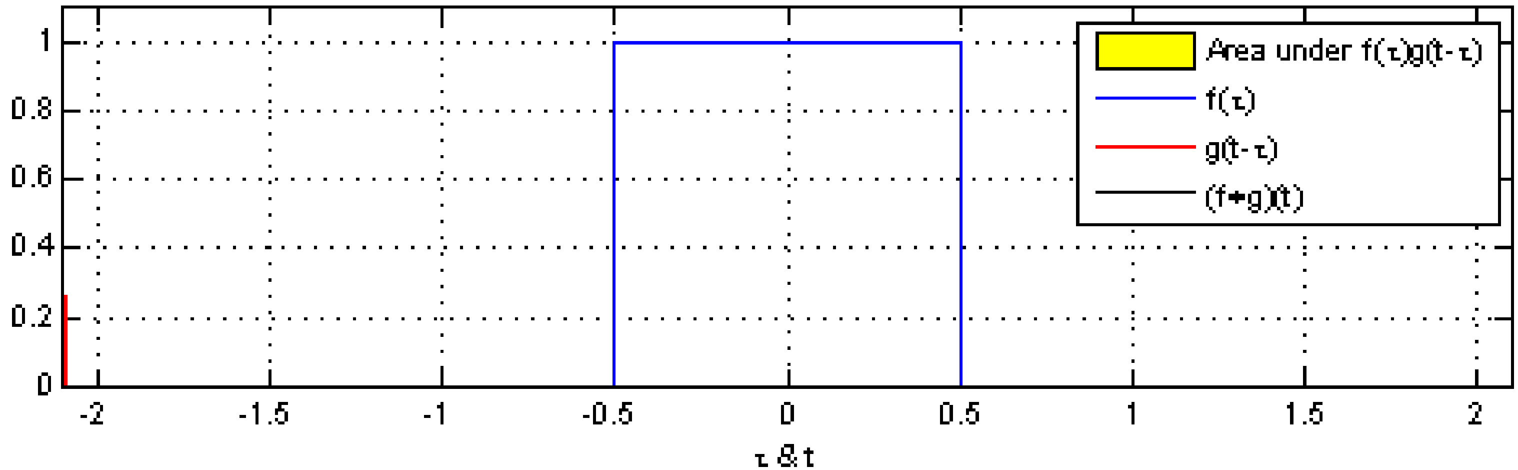
$$(\boldsymbol{w}_1, \dots, \boldsymbol{w}_5, \boldsymbol{w}_6, \dots, \boldsymbol{w}_9)$$

$$(0.1, \dots, 0.8, 0.3, \dots, 0.2)$$

1D convolution

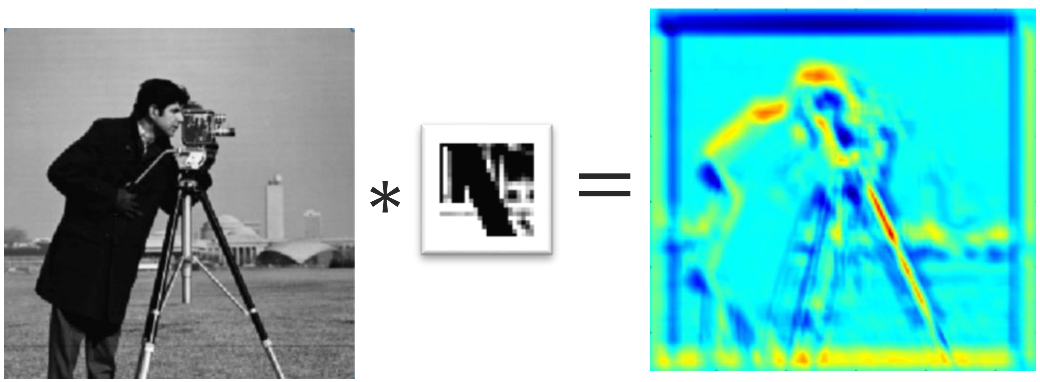
$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)\, g(t - \tau)\, d\tau$$

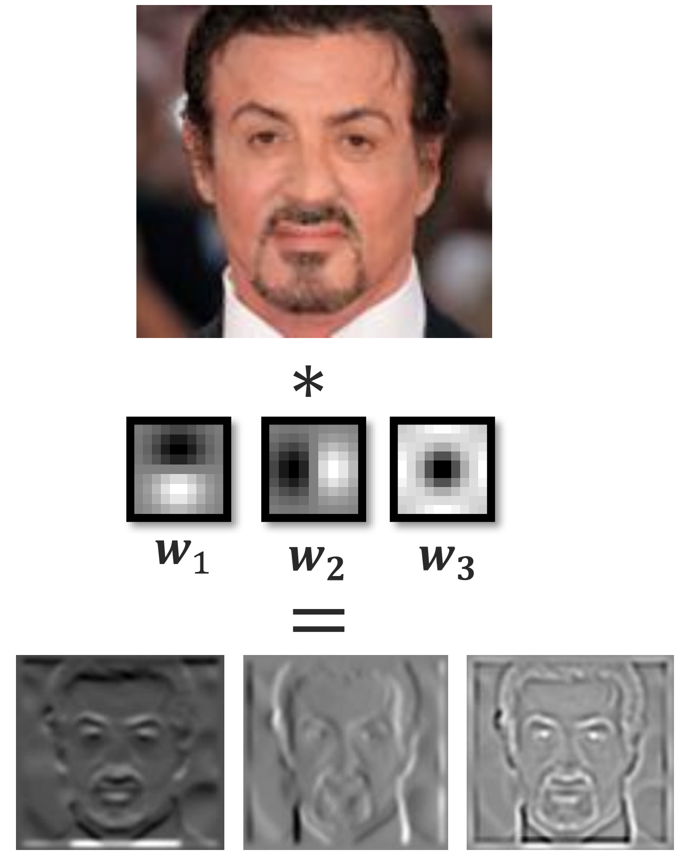
- Intuition
- Correlation between signals



2D convolution

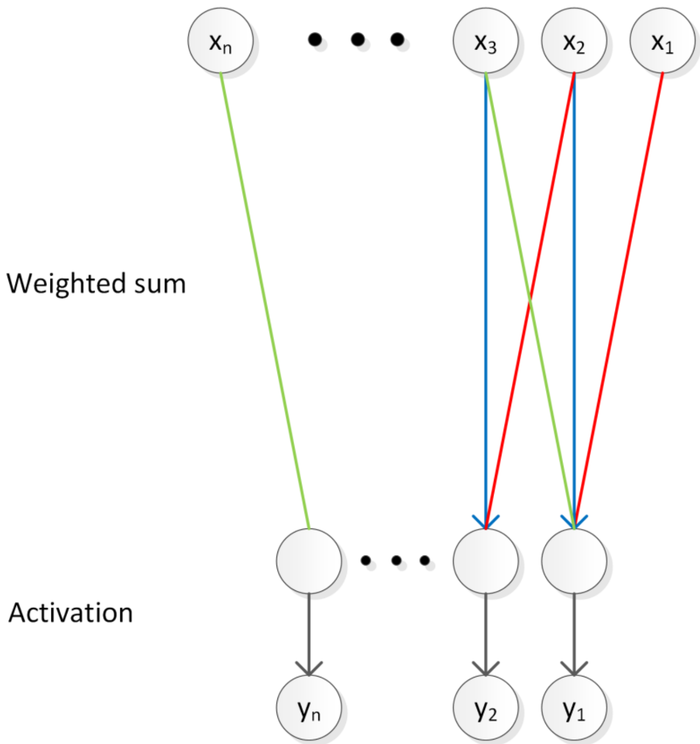
- Intuition
  - Correlation between signals
- Can be done in multichannel images with multichannel kernels

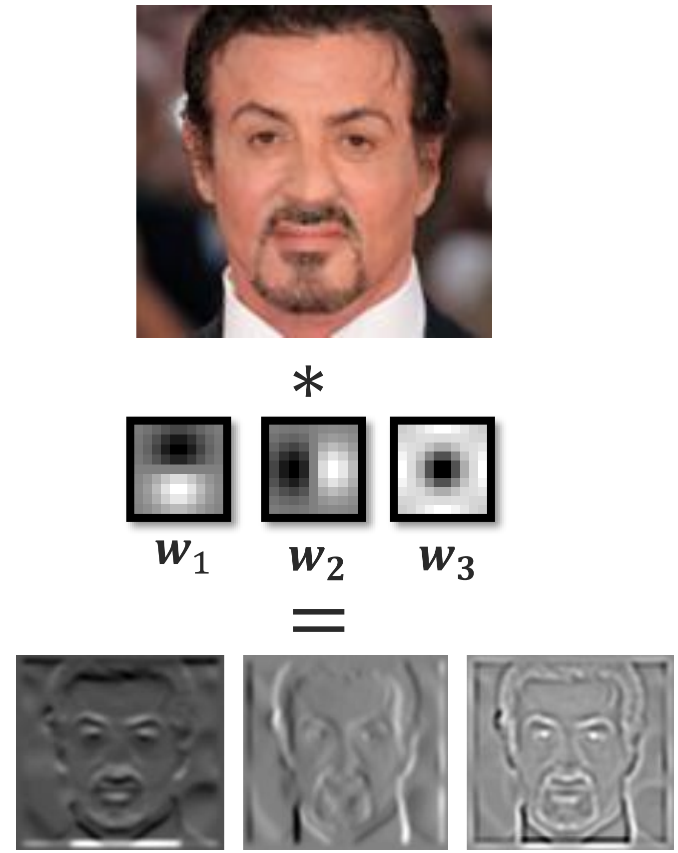




Weight sharing (convolutional) layer

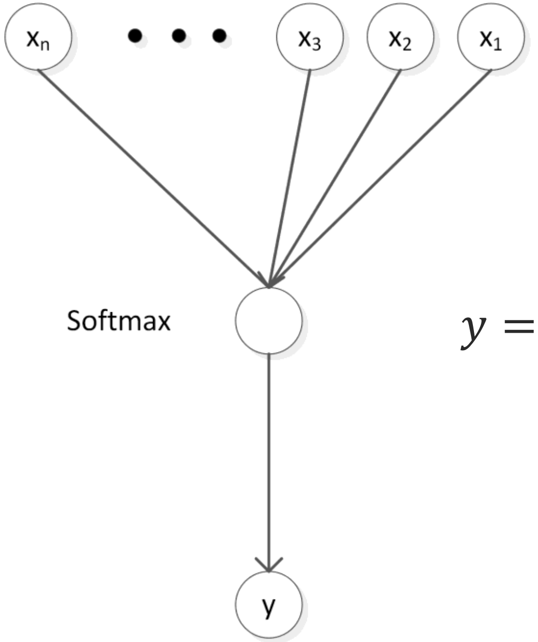
- Same colour indicates same (shared) weight
- Used to implement convolution



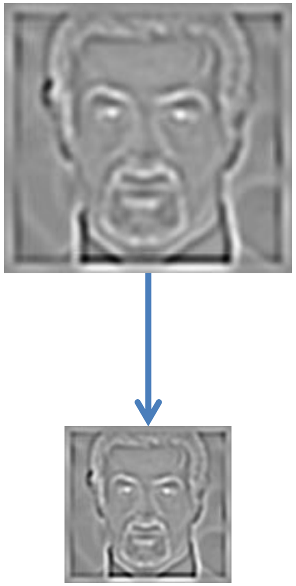


Max pooling layer

- Pick the maximum value from input using a smooth and differentiable approximation
- Used for sub-sampling

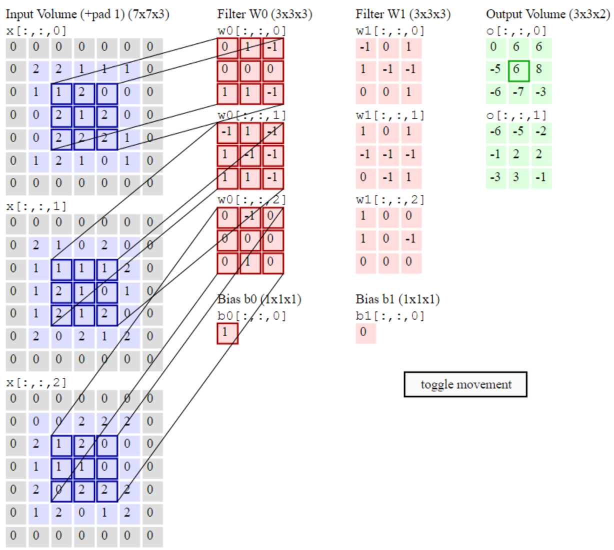


$$y = \frac{\sum_{i=1}^{n} x_i e^{\alpha x_i}}{\sum_{i=1}^{n} e^{\alpha x_i}}$$



Sample CNN convolution

- Great animated visualization of 2D convolution
- http://cs231n.github.io/convolutional-networks/



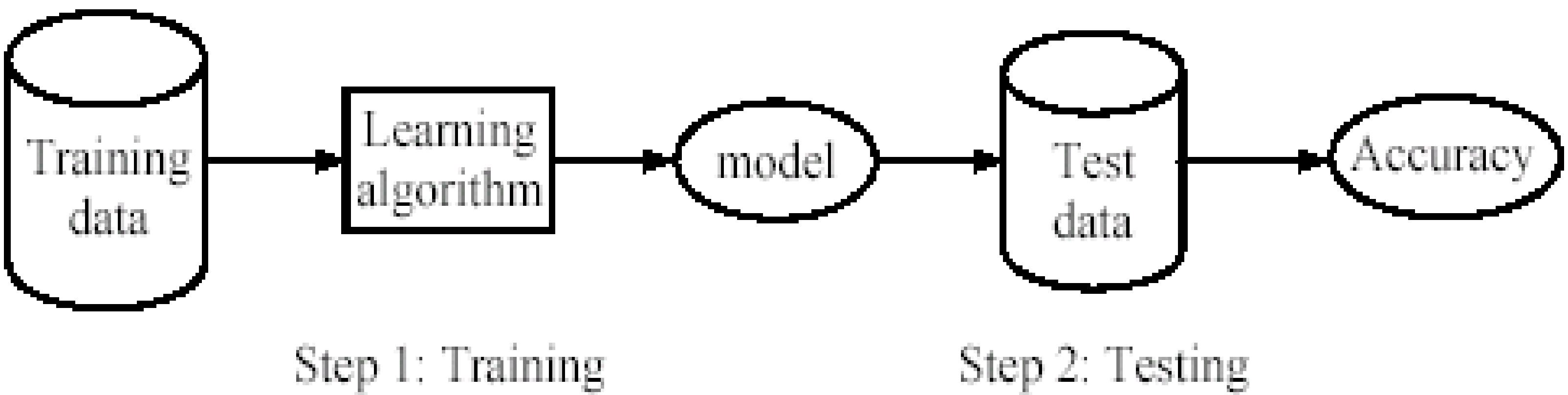
Machine Learning: Evaluation Methods

Supervised learning process: two steps

Learning (training): Learn a model using the training data

Testing: Test the model using unseen test data to assess the model accuracy

$$Accuracy = \frac{\text{Number of correct classifications}}{\text{Total number of test cases}},$$



Evaluation methods

- Holdout set: The available data set *D* is divided into two disjoint subsets,
  - the *training set* D_{train} (for learning a model)
  - the *test set* D_{test} (for testing the model)
- Important: training set should not be used in testing and the test set should not be used in learning.
  - Unseen test set provides a unbiased estimate of accuracy.
- The test set is also called the holdout set. (the examples in the original data set *D* are all labeled with classes.)
- This method is mainly used when the data set *D* is large.
- **Unless building person specific models the training and test sets should not contain the same person**

Evaluation methods (cont…)

- n-fold cross-validation: The available data is partitioned into *n* equal-size disjoint subsets.
- Use each subset as the test set and combine the rest *n*-1 subsets as the training set to learn a classifier.
- The procedure is run *n* times, which give *n* accuracies.
- The final estimated accuracy of learning is the average of the *n* accuracies.
- 10-fold and 5-fold cross-validations are commonly used.
- This method is used when the available data is not large.

Evaluation methods (cont…)

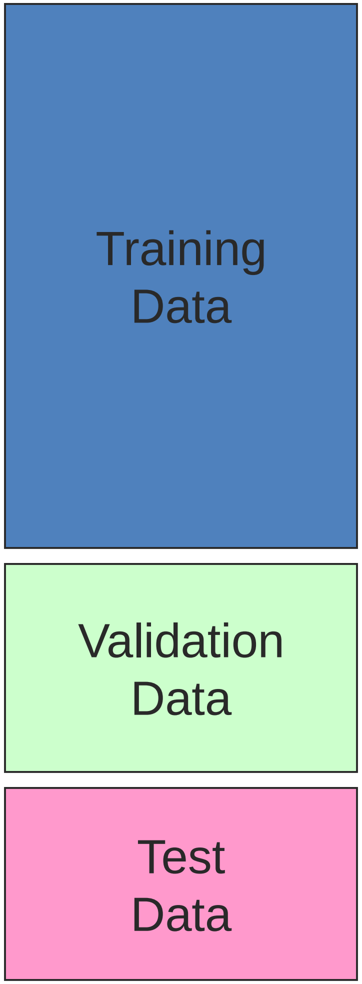
- Leave-one-out cross-validation: This method is used when the data set is very small.
- It is a special case of cross-validation
- Each fold of the cross validation has only a single test example and all the rest of the data is used in training.
- If the original data has *m* examples, this is *m*-fold cross-validation

Hyperparameters

- How do we determine C or γ for SVM training?
- Parameters that we do not learn through optimization are called hyper-parameters
- Need a way to find optimal values for our task
  - For some approaches rules of thumb exist
- Need an analytical way to do it
- Common ways
  - Grid search
  - Random search (not as bad as it sounds)

Training and Validation

- Data: labeled instances, e.g. emails marked spam/ham
  - Training set
  - Validation set
  - Test set
- Training
  - Estimate parameters on training set
  - Tune hyperparameters on validation/development set
  - Report results on test set
  - Anything short of this yields over-optimistic claims
- Evaluation
  - Many different metrics
  - Ideally, the criteria used to train the classifier should be closely related to those used to evaluate the classifier
- Statistical issues
  - Want a classifier which does well on *test* data
  - Overfitting: fitting the training data very closely, but not generalizing well
  - Error bars: want realistic (conservative) estimates of accuracy



Take home

- 1. Never touch test data during training/validation
- 2. Never touch test data during training/validation
- 3. Never touch test data during training/validation

Machine Learning: Measuring Error

Measuring Error

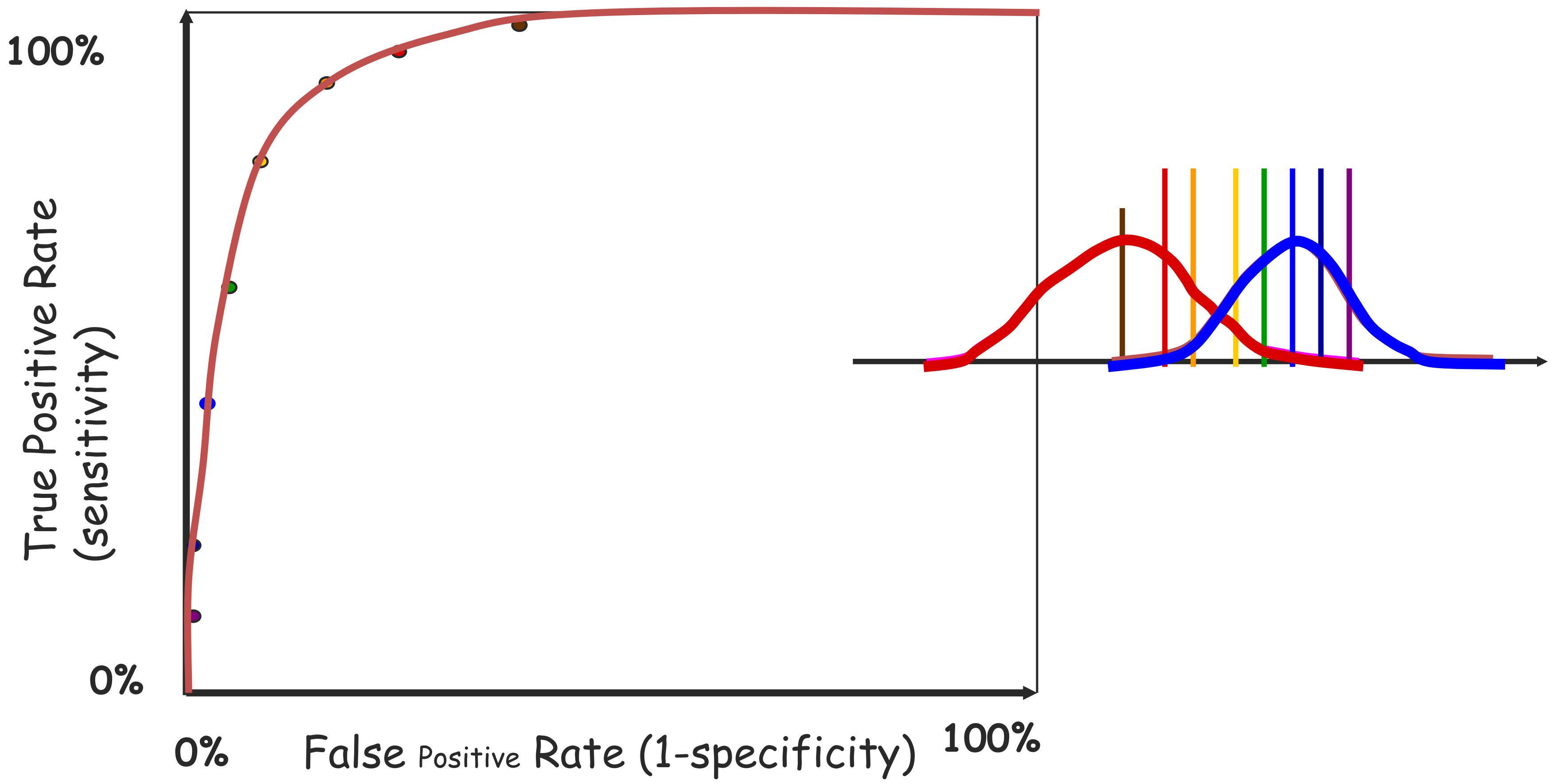
	Predicted class	
True Class	Yes	No
Yes	TP: True Positive	FN: False Negative
No	FP: False Positive	TN: True Negative

- Error rate = # of errors / # of instances = (FN+FP) / N
- Recall = # of found positives / # of positives
  = TP / (TP+FN) = sensitivity = hit rate
- Precision = # of found positives / # of found
  = TP / (TP+FP)
- Specificity = TN / (TN+FP)
- False alarm rate = FP / (FP+TN) = 1 - Specificity

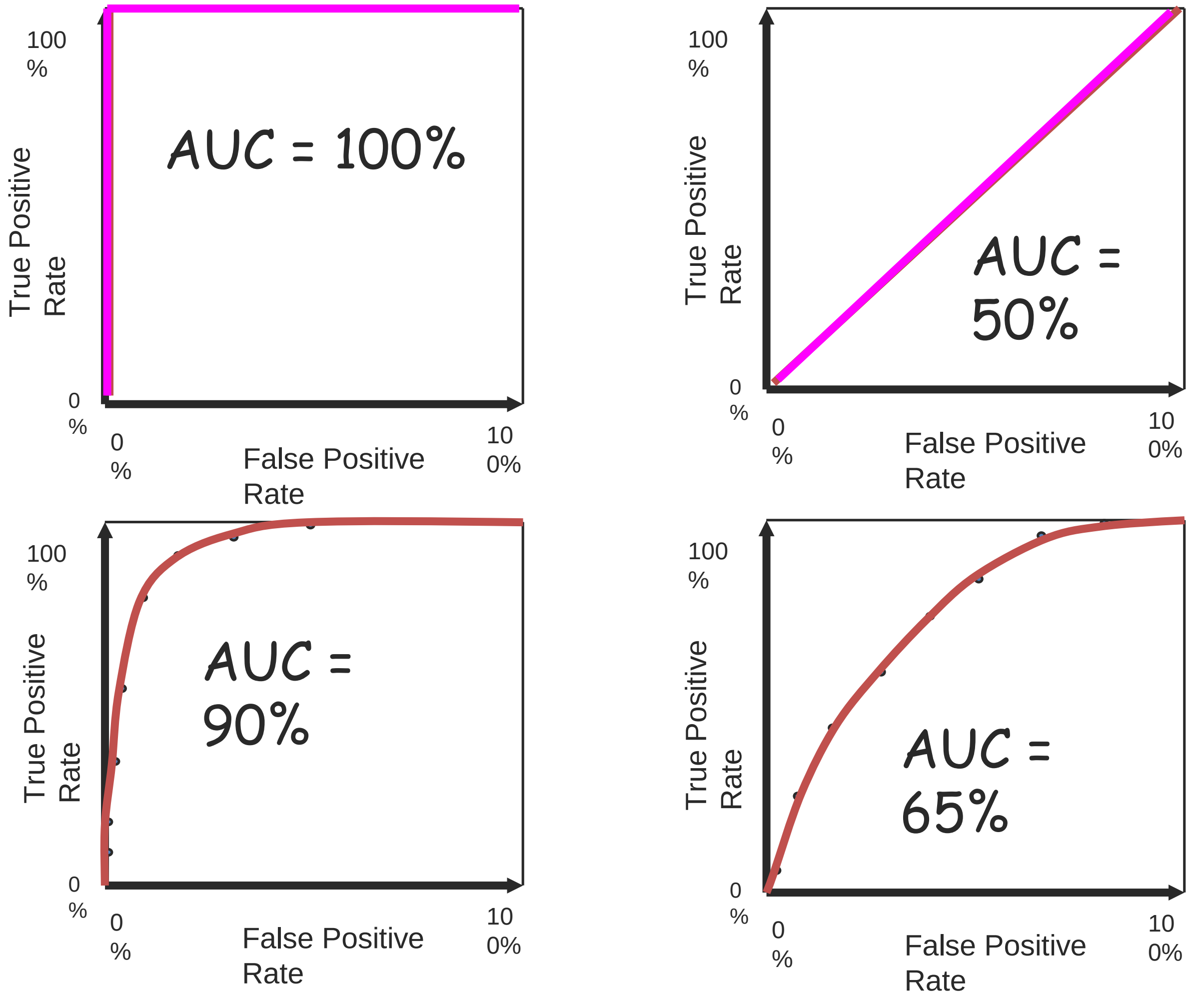
F_1-value (also called F_1-score)

- It is hard to compare two classifiers using two measures. F_1 score combines precision and recall into one measure
  - $F_1 = \frac{2 \cdot p \cdot r}{p + r}$
  - F_1 - score is the harmonic mean of precision and recall
  - $F_1 = \frac{2}{\frac{1}{p} + \frac{1}{r}}$
- The harmonic mean of two numbers tends to be closer to the smaller of the two
- Preferred over accuracy when data is unbalanced
  - Why?

Receiver Operating Characteristic (ROC) Curve



AUC for ROC curves



Evaluation of regression

- Root Mean Square Error
  - $\sqrt{\sum_i (y_i - x_i)^2}$
  - Not easily interpretable

- Correlation – trend prediction in a way
  - Nice interpretation: 0 – no relationship, 1 – perfect relationship
  - $\rho = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{(n-1)\sigma_x \sigma_y}$

- Concordance Correlation Coefficient (CCC)
  - A method to combine both
  - $\rho_c = \frac{2\rho\sigma_x\sigma_y}{\sigma_x^2 + \sigma_y^2 + (\mu_x - \mu_y)^2}$, ρ – correlation coefficient
  - Has nice interpretability as well

Take home

- Error measure selection is not straightforward
  - Pick the right one for your problem
  - F1, AUC, Accuracy, RMSE, CCC
- Make sure the same measure is used for validation and testing
  - Otherwise you might be learning suboptimal models
- Wrong error measure can hide both bad and good results