Machine Learning: Lecture II

Michael Kagan

SLAC

CERN OpenLab Summer School

July 10, 2018

Lecture Topics

- Recap of last time
  - What is Machine Learning
  - Linear Regression
  - Logistic Regression
  - Over fitting and Regularization
  - Training procedures
  - Gradient descent

- This Lecture
  - Neural Networks
  - Decision Trees and Ensemble Methods
  - Unsupervised Learning
    - Dimensionality reduction
    - Clustering
  - No Free Lunch and some Practical Advice

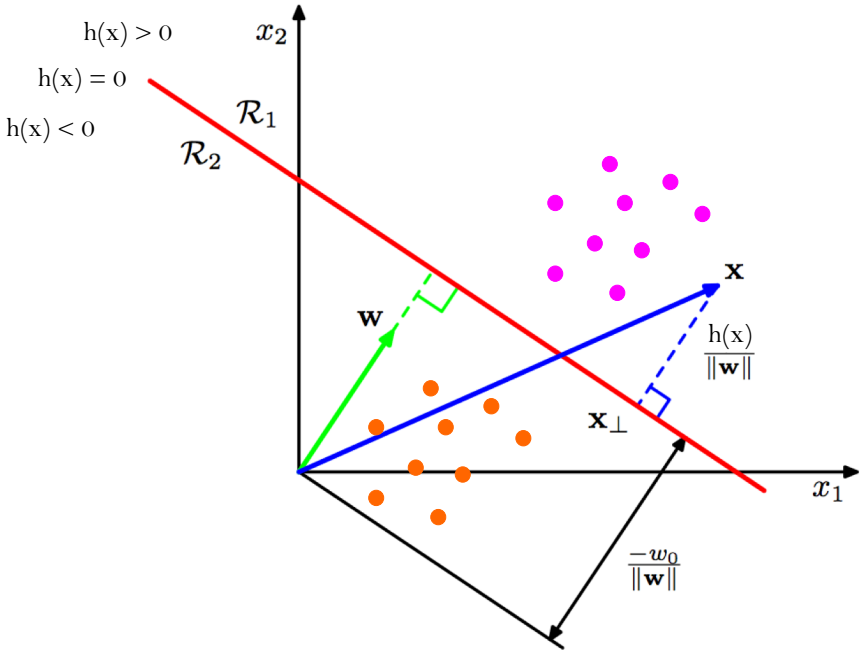
Neural Networks

Reminder of Logistic Regression

- Input output pairs $\{x_i, y_i\}$, with
  - $\mathbf{x}_i \in \mathbb{R}^m$
  - $y_i \in \{0,1\}$

- Linear decision boundary

$$h(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \mathbf{x}$$



[Bishop]

Reminder of Logistic Regression

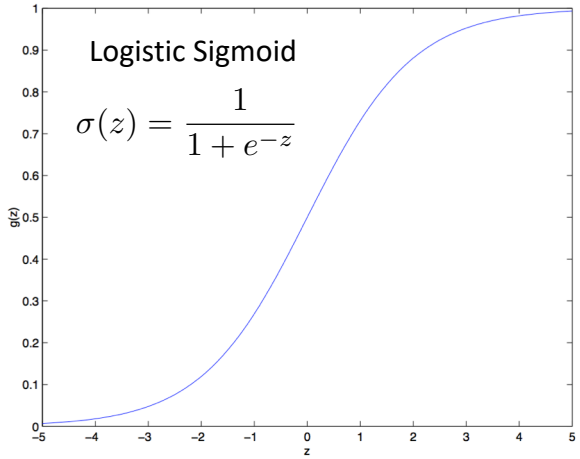
- Input output pairs $\{x_i, y_i\}$, with
  - $\mathbf{x}_i \in \mathbb{R}^m$
  - $y_i \in \{0,1\}$

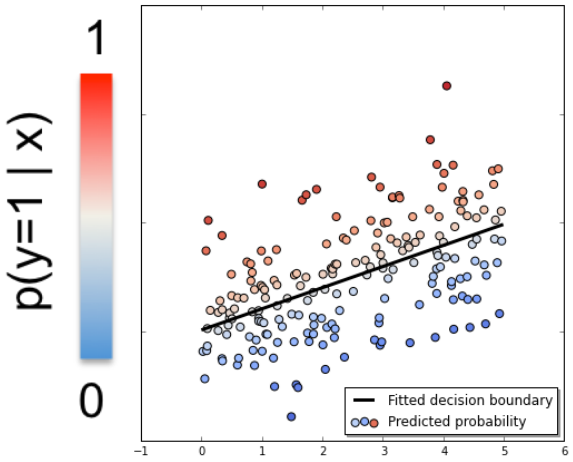
- Linear decision boundary

$$h(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \mathbf{x}$$

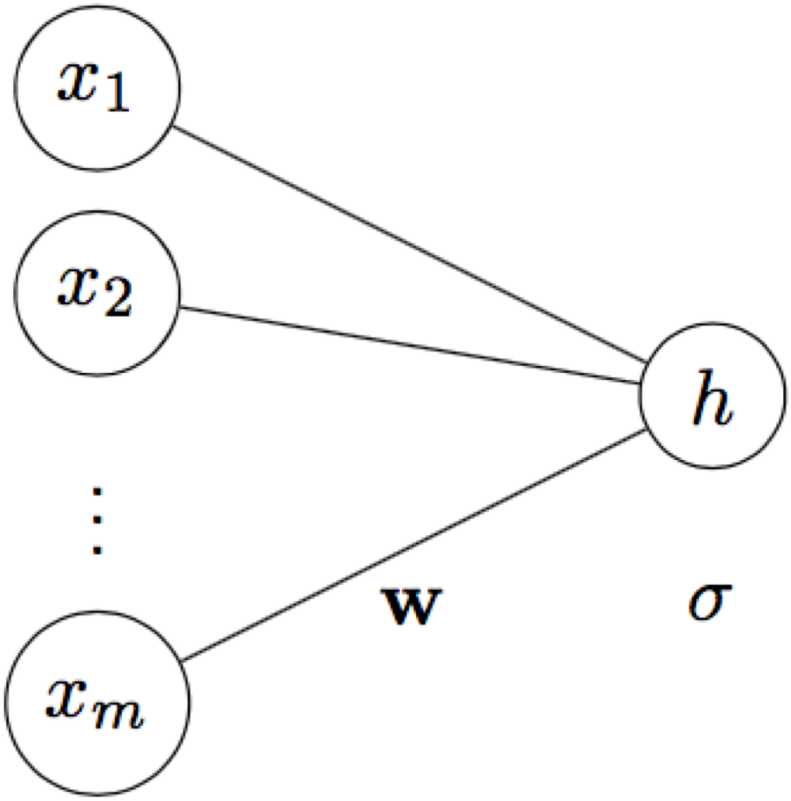
- Distance from decision boundary is converted to class probability using logistic sigmoid function

$$p(y = 1|\mathbf{x}) = \sigma(h(\mathbf{x}, \mathbf{w})) = \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}}$$





Logistic Regression



$$p(y = 1|\mathbf{x}) = \sigma(h(\mathbf{x}, \mathbf{w}))$$
$$= \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}}$$

Adding non-linearity

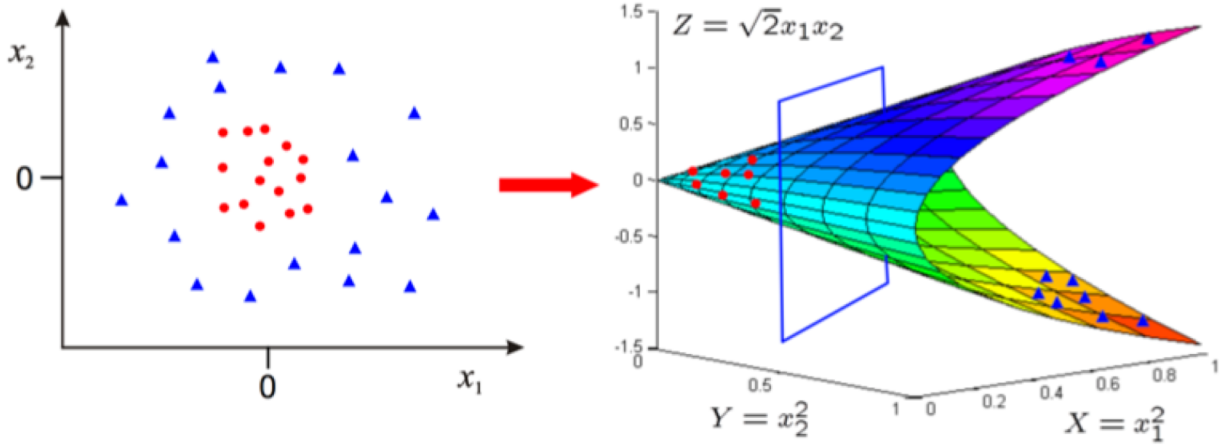
- What if we want a non-linear decision boundary?

Adding non-linearity

- What if we want a non-linear decision boundary?
  - Choose basis functions, e.g: $\phi(x) \sim \{x^2, \sin(x), \log(x), \ldots\}$

$$p(y = 1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \phi(\mathbf{x})}}$$

$$\Phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{pmatrix} \quad \mathbb{R}^2 \rightarrow \mathbb{R}^3$$



Adding non-linearity

- What if we want a non-linear decision boundary?
  - Choose basis functions, e.g: $\phi(x) \sim \{x^2, \sin(x), \log(x), \ldots\}$

$$p(y = 1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \phi(\mathbf{x})}}$$

- What if we don't know what basis functions we want?

Adding non-linearity

- What if we want a non-linear decision boundary?
  - Choose basis functions, e.g: $\phi(x) \sim \{x^2, \sin(x), \log(x), \ldots\}$

$$p(y = 1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \phi(\mathbf{x})}}$$

- What if we don't know what basis functions we want?
- Learn the basis functions directly from data

$$\phi(\mathbf{x}; \mathbf{u}) \qquad \mathbb{R}^m \rightarrow \mathbb{R}^d$$

  - Where $\mathbf{u}$ is a set of parameters for the transformation

Adding non-linearity

- What if we want a non-linear decision boundary?
  - Choose basis functions, e.g: $\phi(x) \sim \{x^2, \sin(x), \log(x), \ldots\}$

$$p(y = 1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \phi(\mathbf{x})}}$$

- What if we don't know what basis functions we want?
- Learn the basis functions directly from data

$$\phi(\mathbf{x}; \mathbf{u}) \qquad \mathbb{R}^m \rightarrow \mathbb{R}^d$$

  - Where **u** is a set of parameters for the transformation
  - Combines basis selection and learning
  - Several different approaches, focus here on neural networks
  - Complicates the optimization

Neural Networks

- Define the basis functions $j = \{1 \ldots d\}$

$$\phi_j(\mathbf{x}; \mathbf{u}) = \sigma(\mathbf{u}_j^T \mathbf{x})$$

Neural Networks

- Define the basis functions $j = \{1 \ldots d\}$

$$\phi_j(\mathbf{x}; \mathbf{u}) = \sigma(\mathbf{u}_j^T \mathbf{x})$$

- Put all $\mathbf{u}_j \in \mathbb{R}^{1xm}$ vectors into matrix $\mathbf{U}$

$$\phi(\mathbf{x}; \mathbf{U}) = \sigma(\mathbf{U}\mathbf{x}) = \begin{bmatrix} \sigma(\mathbf{u}_1^T \mathbf{x}) \\ \sigma(\mathbf{u}_2^T \mathbf{x}) \\ \ldots \\ \sigma(\mathbf{u}_d^T \mathbf{x}) \end{bmatrix} \in \mathbb{R}^d$$

  - σ is a point-wise sigmoid acting on each vector element

Neural Networks

- Define the basis functions $j = \{1 \ldots d\}$

$$\phi_j(\mathbf{x}; \mathbf{u}) = \sigma(\mathbf{u}_j^T \mathbf{x})$$

- Put all $\mathbf{u}_j \in \mathbb{R}^{1xm}$ vectors into matrix $\mathbf{U}$

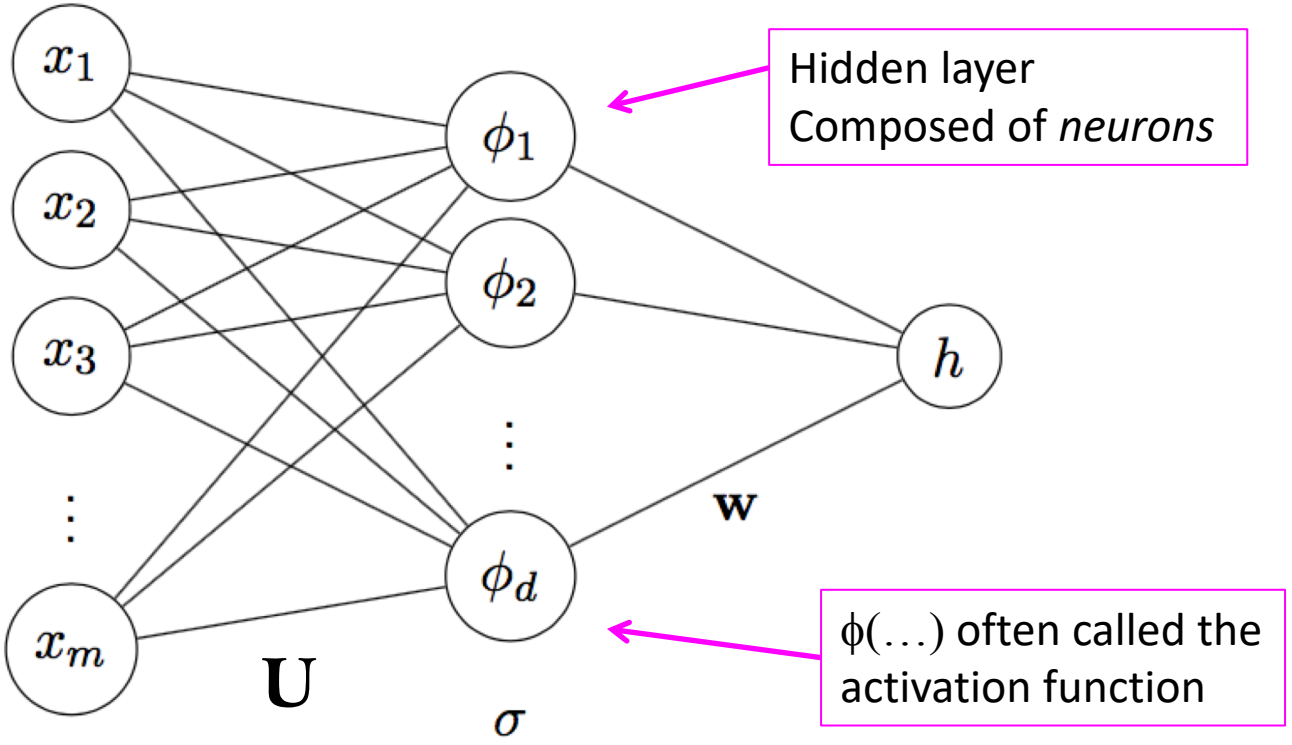
$$\phi(\mathbf{x}; \mathbf{U}) = \sigma(\mathbf{U}\mathbf{x}) = \begin{bmatrix} \sigma(\mathbf{u}_1^T \mathbf{x}) \\ \sigma(\mathbf{u}_2^T \mathbf{x}) \\ \ldots \\ \sigma(\mathbf{u}_d^T \mathbf{x}) \end{bmatrix} \in \mathbb{R}^d$$

  - σ is a point-wise sigmoid acting on each vector element

- Full model becomes

$$h(\mathbf{x}; \mathbf{w}, \mathbf{U}) = \mathbf{w}^T \phi(\mathbf{x}; \mathbf{U})$$

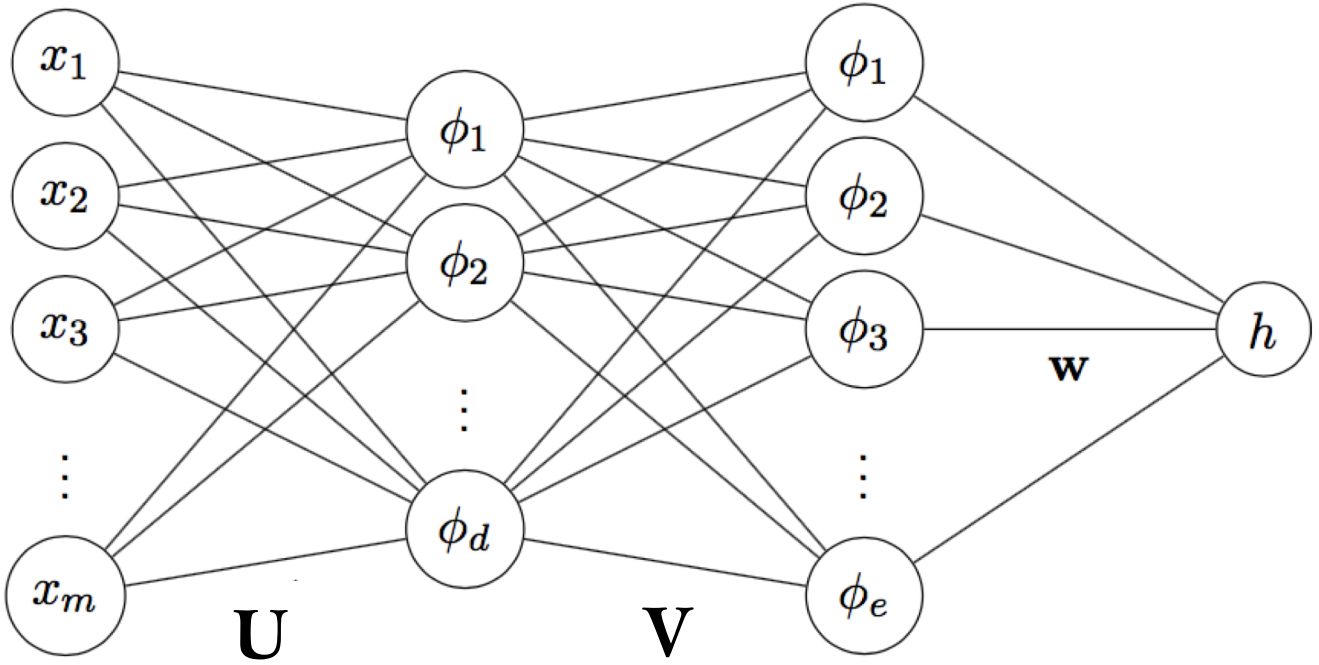
Feed Forward Neural Network



$$\phi(\mathbf{x}) = \sigma(\mathbf{U}\mathbf{x})$$

$$h(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$$

Multi-layer Neural Network



- Multilayer NN
  - Each layer adapts basis functions based on previous layer

Universal approximation theorem

- Feed-forward neural network with a single hidden layer containing a finite number of neurons can approximate continuous functions arbitrarily well on a compact space of $\mathbb{R}^n$

  - Only mild assumptions on non-linear activation function needed. Sigmoid functions work, as do others

Universal approximation theorem

- Feed-forward neural network with a single hidden layer containing a finite number of neurons can approximate continuous functions arbitrarily well on a compact space of $\mathbb{R}^n$

  - Only mild assumptions on non-linear activation function needed. Sigmoid functions work, as do others

- But no information on how many neurons needed, or how much data!

Universal approximation theorem

- Feed-forward neural network with a single hidden layer containing a finite number of neurons can approximate continuous functions arbitrarily well on a compact space of $\mathbb{R}^n$
  - Only mild assumptions on non-linear activation function needed. Sigmoid functions work, as do others
- But no information on how many neurons needed, or how much data!
- How to find the parameters, given a dataset, to perform this approximation?

Neural Network Optimization Problem

- Neural Network Model: $h(\mathbf{x}) = \mathbf{w}^T \sigma(\mathbf{U}\mathbf{x})$

- **Classification**: Cross-entropy loss function

$$p_i = p(y_i = 1|\mathbf{x}_i) = \sigma(h(\mathbf{x}_i))$$

$$L(\mathbf{w}, \mathbf{U}) = -\sum_i y_i \ln(p_i) + (1 - y_i) \ln(1 - p_i)$$

Neural Network Optimization Problem

- Neural Network Model: $h(\mathbf{x}) = \mathbf{w}^T \sigma(\mathbf{U}\mathbf{x})$

- **Classification**: Cross-entropy loss function

$$p_i = p(y_i = 1|\mathbf{x}_i) = \sigma(h(\mathbf{x}_i))$$

$$L(\mathbf{w}, \mathbf{U}) = -\sum_i y_i \ln(p_i) + (1 - y_i) \ln(1 - p_i)$$

- **Regression**: Square error loss function

$$L(\mathbf{w}, \mathbf{U}) = \frac{1}{2} \sum_i (y_i - h(\mathbf{x}_i))^2$$

Neural Network Optimization Problem

- Neural Network Model: $h(\mathbf{x}) = \mathbf{w}^T \sigma(\mathbf{U}\mathbf{x})$

- **Classification**: Cross-entropy loss function

$$p_i = p(y_i = 1|\mathbf{x}_i) = \sigma(h(\mathbf{x}_i))$$

$$L(\mathbf{w}, \mathbf{U}) = -\sum_i y_i \ln(p_i) + (1 - y_i) \ln(1 - p_i)$$

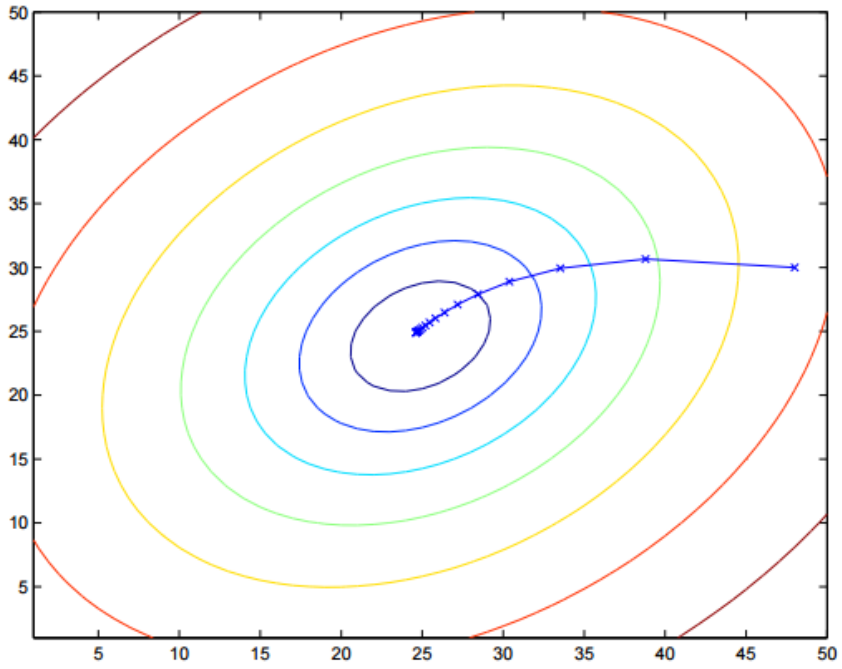
- **Regression**: Square error loss function

$$L(\mathbf{w}, \mathbf{U}) = \frac{1}{2} \sum_i (y_i - h(\mathbf{x}_i))^2$$

- Minimize loss with respect to weights **w**, **U**

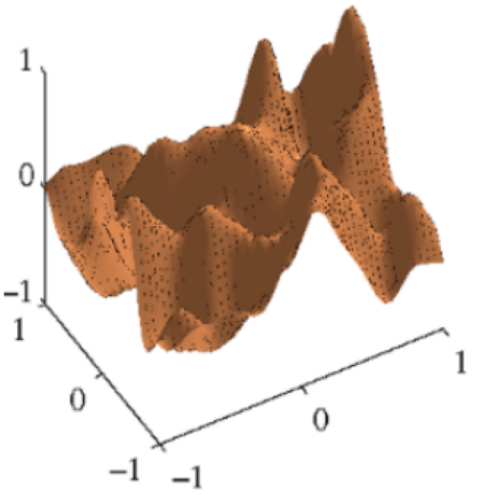
Gradient Descent

- Minimize loss by repeated gradient steps
  - Compute gradient w.r.t. parameters: $\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}}$
  - Update parameters: $\mathbf{w}' \leftarrow \mathbf{w} - \eta \frac{\partial L(\mathbf{w})}{\partial \mathbf{w}}$



Gradient Descent

- Minimize loss by repeated gradient steps
  - Compute gradient w.r.t. parameters: $\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}}$
  - Update parameters: $\mathbf{w}' \leftarrow \mathbf{w} - \eta \frac{\partial L(\mathbf{w})}{\partial \mathbf{w}}$
- Now we need gradients w.r.t. $\mathbf{w}$ and $\mathbf{U}$
- Gradients will depend on loss and network architecture
- Loss function is non-convex
  - Gradient descent may get stuck in non-optimal stationary point
  - Can be a major issue!
  - Variants of stochastic gradient descent can be helpful!



Chain Rule

$$L(\mathbf{w}, \mathbf{U}) = -\sum_i y_i \ln(\sigma(h(\mathbf{x}_i))) + (1 - y_i) \ln(1 - \sigma(h(\mathbf{x}_i)))$$

- Derivative of sigmoid: $\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$
- Chain rule to compute gradient w.r.t. $\mathbf{w}$

$$\frac{\partial L}{\partial \mathbf{w}} = \frac{\partial L}{\partial h} \frac{\partial h}{\partial \mathbf{w}} = \sum_i y_i (1 - \sigma(h(\mathbf{x}_i))) \sigma(\mathbf{U}\mathbf{x}) + (1 - y_i) \sigma(h(\mathbf{x})) \sigma(\mathbf{U}\mathbf{x}_i)$$

- Chain rule to compute gradient w.r.t. $\mathbf{u}_j$

$$\frac{\partial L}{\partial \mathbf{u}_j} = \frac{\partial L}{\partial h} \frac{\partial h}{\partial \sigma} \frac{\partial \sigma}{\partial \mathbf{u}_j} =$$

$$= \sum_i y_i (1 - \sigma(h(\mathbf{x}_i))) w_j \sigma(\mathbf{u}_j \mathbf{x}_i)(1 - \sigma(\mathbf{u}_j \mathbf{x}_i)) \mathbf{x}_i$$

$$+ (1 - y_i) \sigma(h(\mathbf{x}_i)) w_j \sigma(\mathbf{u}_j \mathbf{x}_i)(1 - \sigma(\mathbf{u}_j \mathbf{x}_i)) \mathbf{x}_i$$

Backpropagation

- Loss function composed of layers of nonlinearity

$$L\left(\phi^N\left(\ldots\phi^1(x)\right)\right)$$

Backpropagation

- Loss function composed of layers of nonlinearity

$$L\left(\phi^N\left(\ldots\phi^1(x)\right)\right)$$

- Forward step (f-prop)
  - Compute and save intermediate computations

$$\phi^N\left(\ldots\phi^1(x)\right)$$

Backpropagation

- Loss function composed of layers of nonlinearity

$$L\left(\phi^N\left(\ldots\phi^1(x)\right)\right)$$

- Forward step (f-prop)
  - Compute and save intermediate computations

$$\phi^N\left(\ldots\phi^1(x)\right)$$

- Backward step (b-prop)

$$\frac{\partial L}{\partial \phi^a} = \sum_j \frac{\partial \phi_j^{(a+1)}}{\partial \phi_j^a} \frac{\partial L}{\partial \phi_j^{(a+1)}}$$

Backpropagation

- Loss function composed of layers of nonlinearity

$$L\left(\phi^N\left(\ldots\phi^1(x)\right)\right)$$

- Forward step (f-prop)
  - Compute and save intermediate computations

$$\phi^N\left(\ldots\phi^1(x)\right)$$

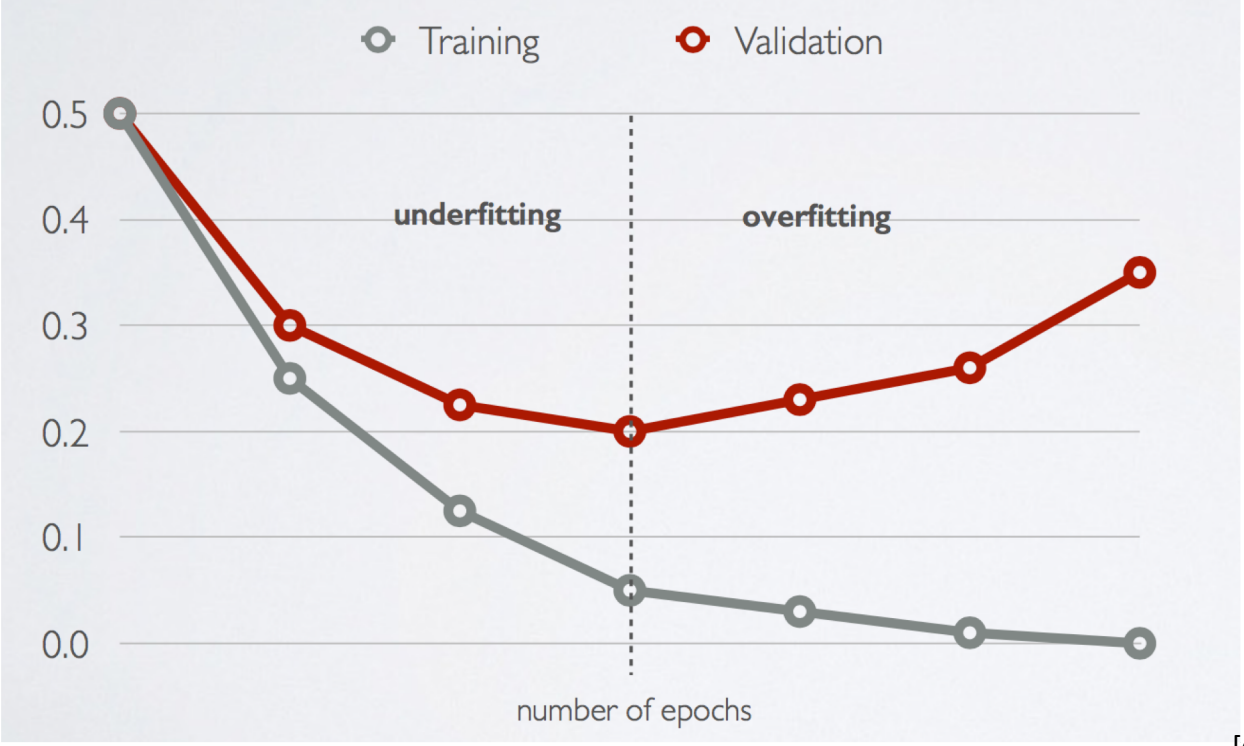
- Backward step (b-prop) $\dfrac{\partial L}{\partial \phi^a} = \sum_j \dfrac{\partial \phi_j^{(a+1)}}{\partial \phi_j^a} \dfrac{\partial L}{\partial \phi_j^{(a+1)}}$

- Compute parameter gradients $\dfrac{\partial L}{\partial \mathbf{w}^a} = \sum_j \dfrac{\partial \phi_j^a}{\partial \mathbf{w}^a} \dfrac{\partial L}{\partial \phi_j^a}$

Training

- Repeat gradient update of weights to reduce loss
  - Each iteration through dataset is called an epoch

- Use validation set to examine for overtraining, and determine when to stop training

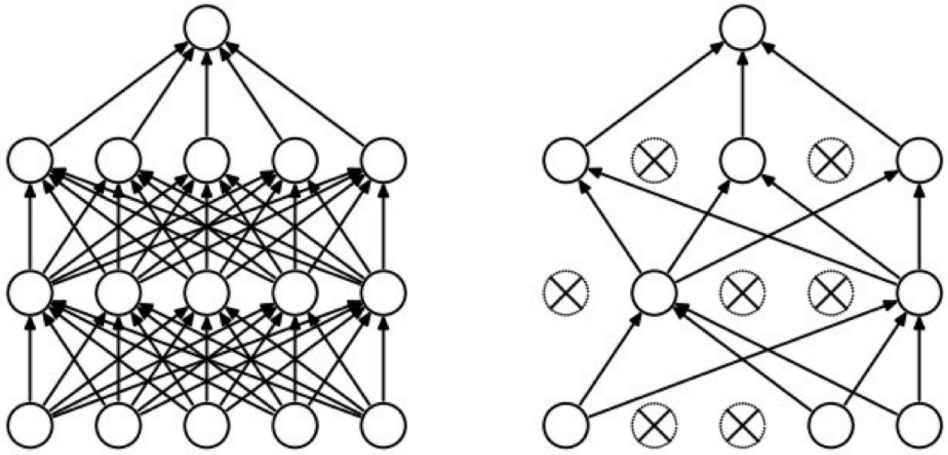


[graphic from H. Larochelle]

Regularization

- L2 regularization: add $\Omega(\mathbf{w}) = ||\mathbf{w}||^2$ to loss
  - Also called "weight decay"
  - Gaussian prior on weights, keep weights from getting too large and saturating activation function

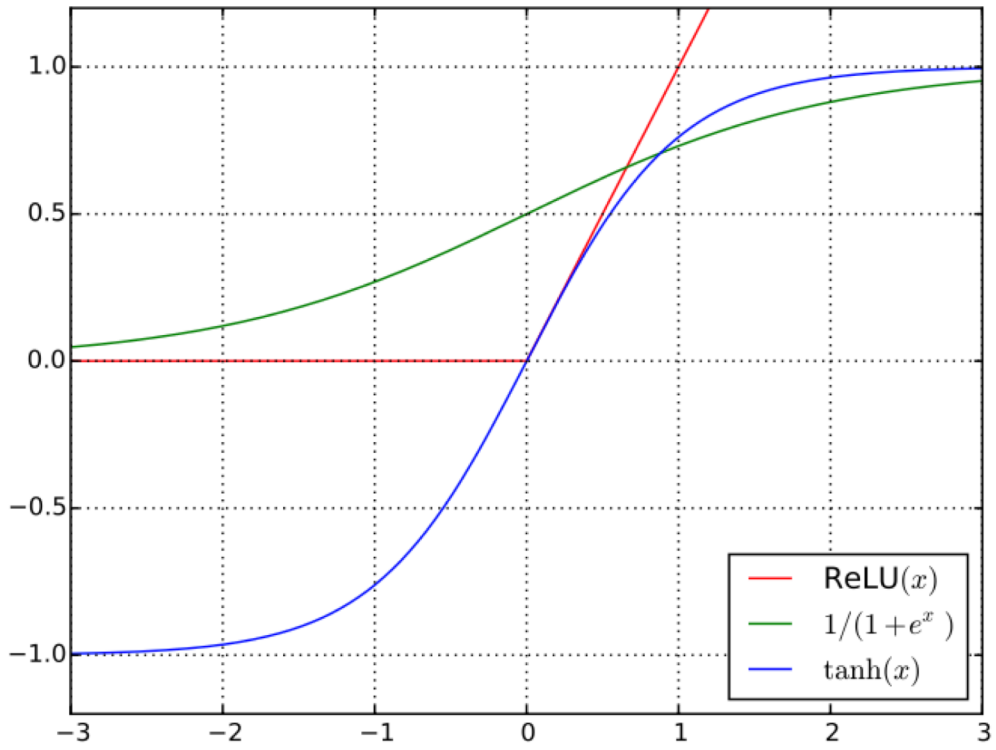
- Regularization inside network, example: **Dropout**
  - Randomly remove nodes during training
  - Avoid co-adaptation of nodes
  - Essentially a large model averaging procedure



(a) Standard Neural Net (b) After applying dropout.

arXiv:1207.0580

Activation Functions



- **Vanishing gradient problem**
  - Derivative of sigmoid:

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1-\sigma(x))$$

  - Nearly 0 when x is far from 0!
  - Gradient descent difficult!

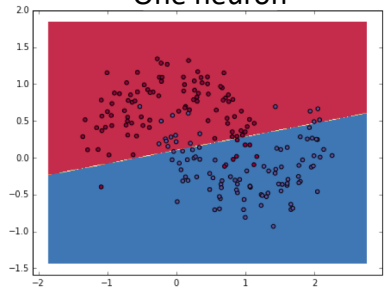
- **Rectified Linear Unit (ReLU)**
  - ReLU(x) = max{0, x}
  - Derivative is constant!

$$\frac{\partial \mathrm{Re}LU(x)}{\partial x} = \begin{cases} 1 & when\ x > 0 \\ 0 & otherwise \end{cases}$$

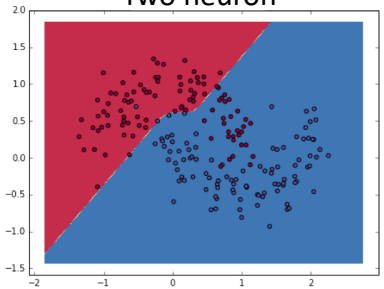
  - ReLU gradient doesn't vanish

Neural Network Decision Boundaries

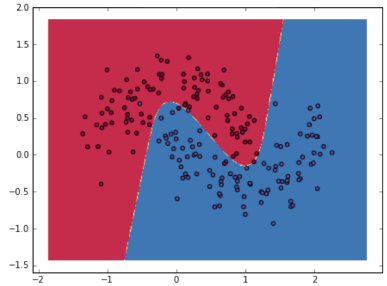
One neuron



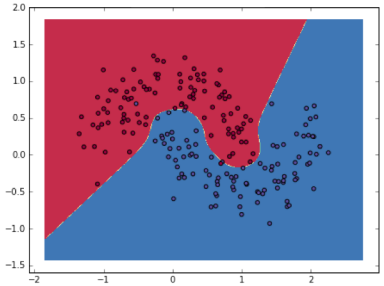
Two neuron



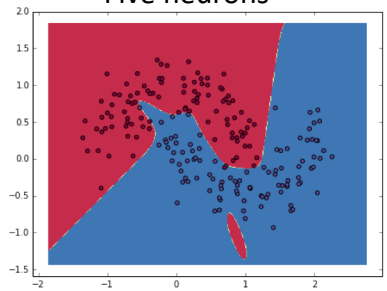
Three neurons



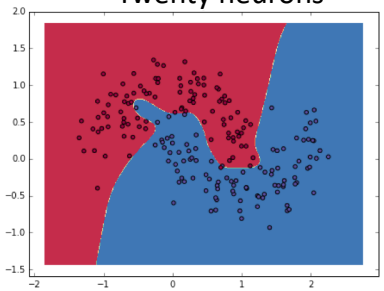
Four neurons



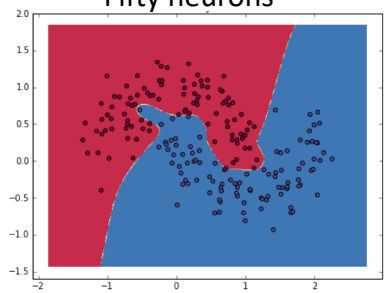
Five neurons

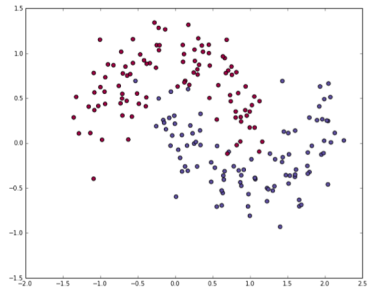


Twenty neurons



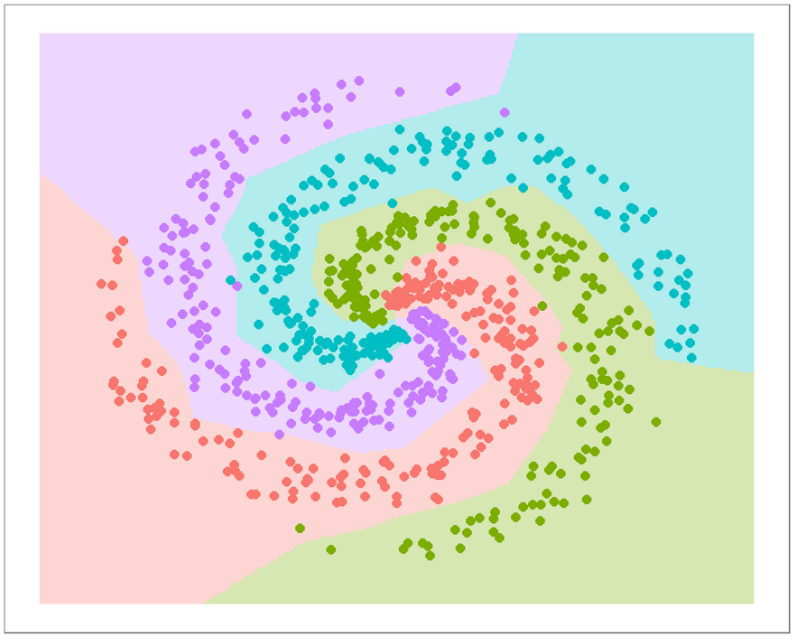
Fifty neurons





4-class classification
2-hidden layer NN
ReLU activations
L2 norm regularization

x_2



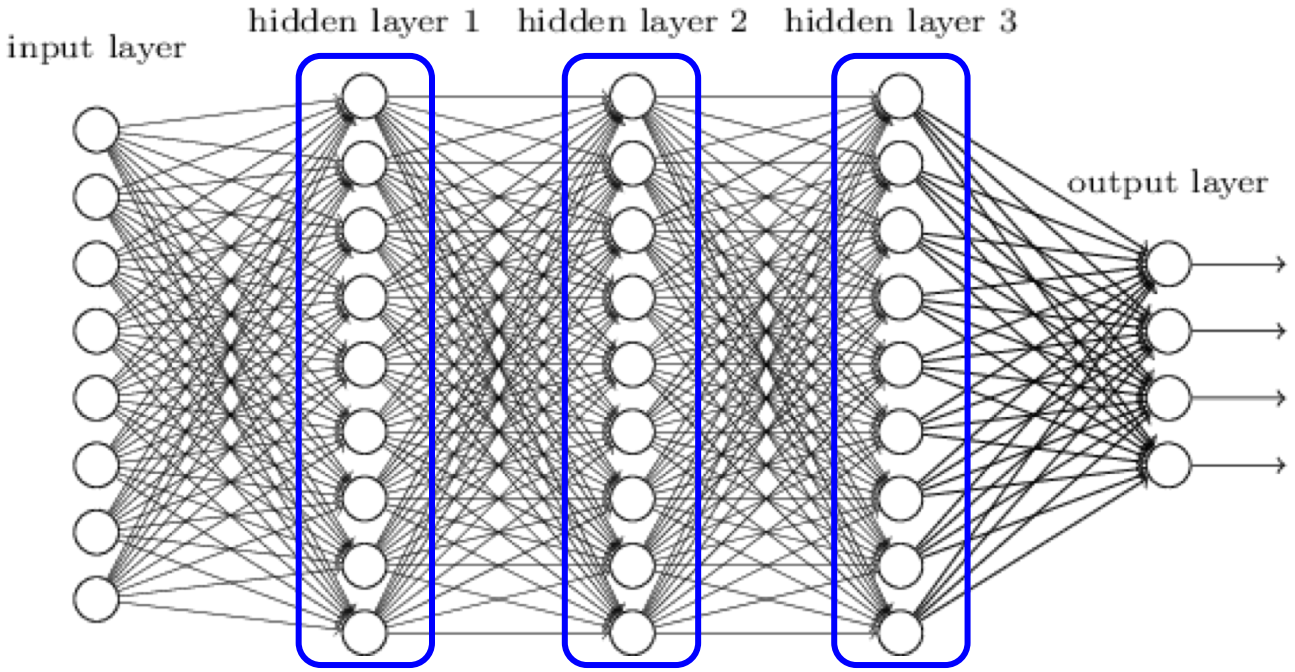
x_1

2-class classification
1-hidden layer NN
L2 norm regularization

http://www.wildml.com/2015/09/implementing-a-neural-network-from-scratch/

http://junma5.weebly.com/data-blog/build-your-own-neural-network-classifier-in-r

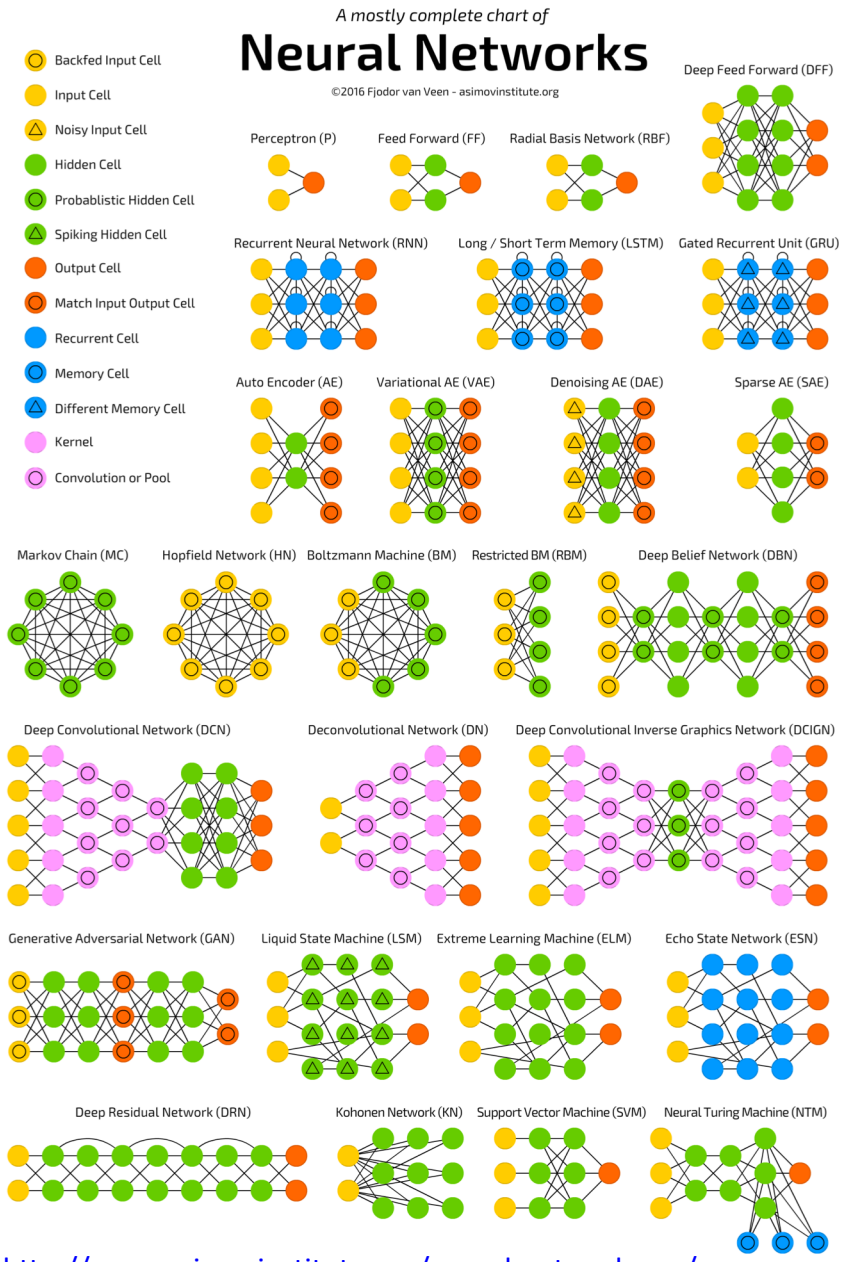
Deep Neural Networks



- As data complexity grows, need exponentially large number of neurons in a single-hidden-layer network to capture all the structure in the data
- Deep neural networks have many hidden layers
  - Factorize the learning of structure in the data across many layers
- Difficult to train, only recently possible with large datasets, fast computing (GPU) and new training procedures / network structures (like dropout)

Neural Network Architectures

- Structure of the networks, and the node connectivity can be adapted for problem at hand
- Moving inductive bias from feature engineering to machine learning (neural network) model design
  - Inductive bias:
    Knowledge about the problem
  - Feature engineering:
    Hand crafted variables
  - Model design:
    The data representation and the structure of the machine learning model / network

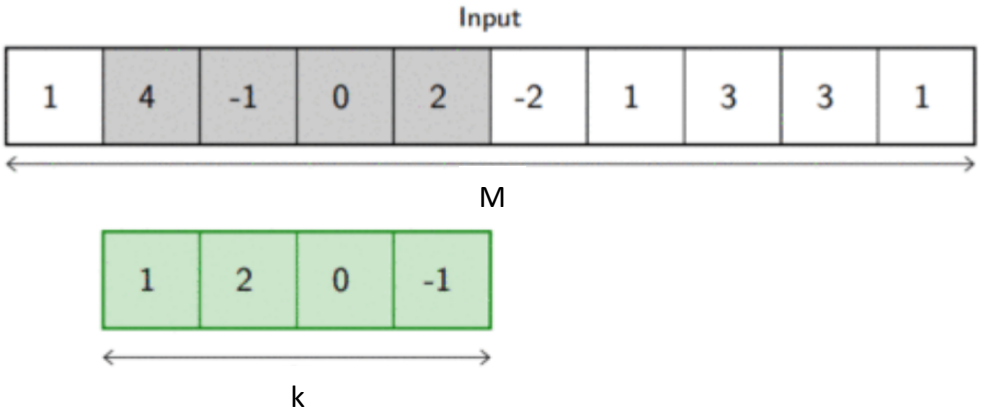


http://www.asimovinstitute.org/neural-network-zoo/

Convolutions

- **Convolutions**: $x \in \mathbb{R}^M$ and kernel $u \in \mathbb{R}^k$
  discrete convolution $x * u$ is vector of size M-k+1

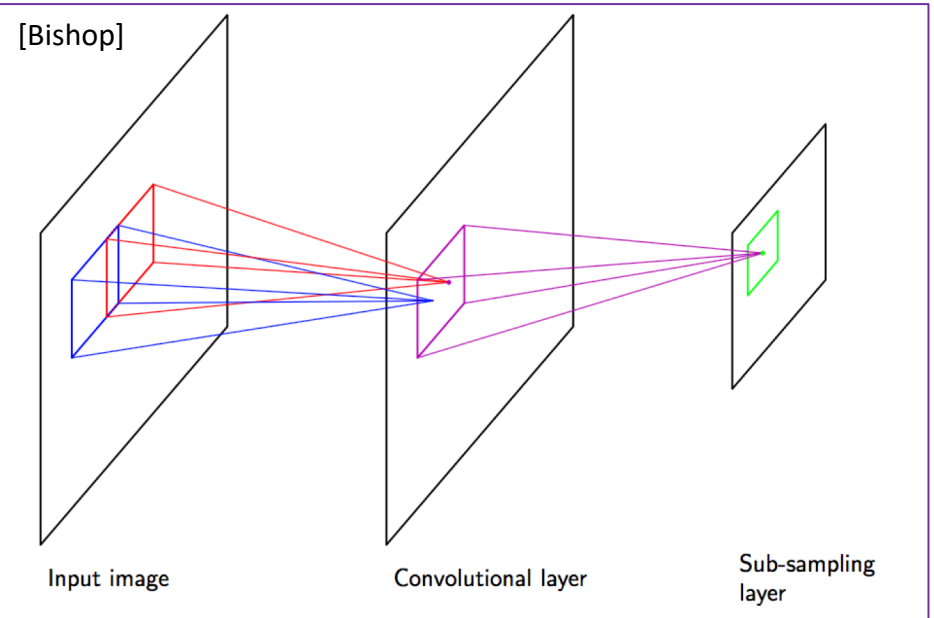
$$(x * \mathrm{u})_i = \sum_{b=0}^{k-1} x_{i+b} u_b$$

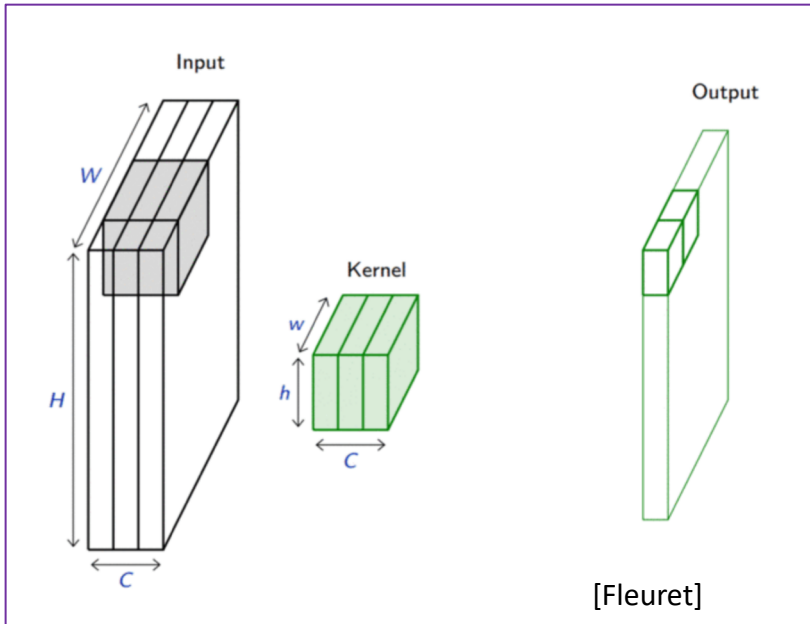


[Fleuret]

Convolutions

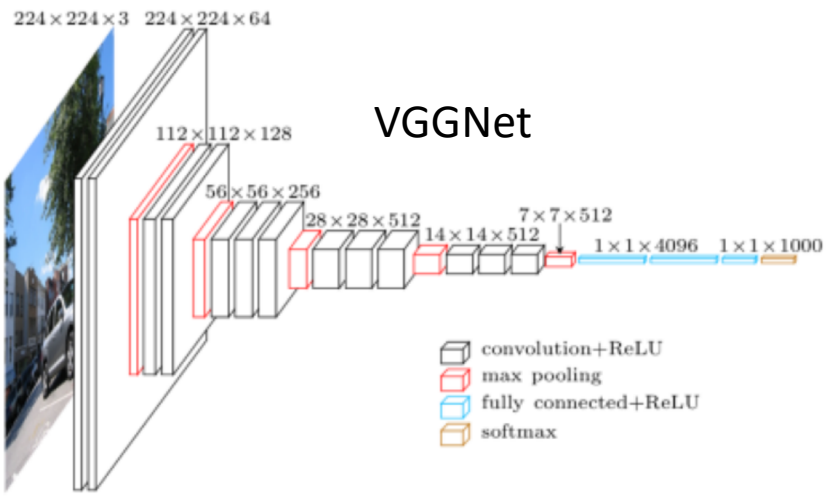
- Kernels are "scanned" across input, picking up local pattern learned by the weights
  - Shared weights of neurons, but each neuron only takes subset of inputs
  - Insensitive to translations of the features the kernel is activated by
  - "Tied weights" reduced total number of parameters



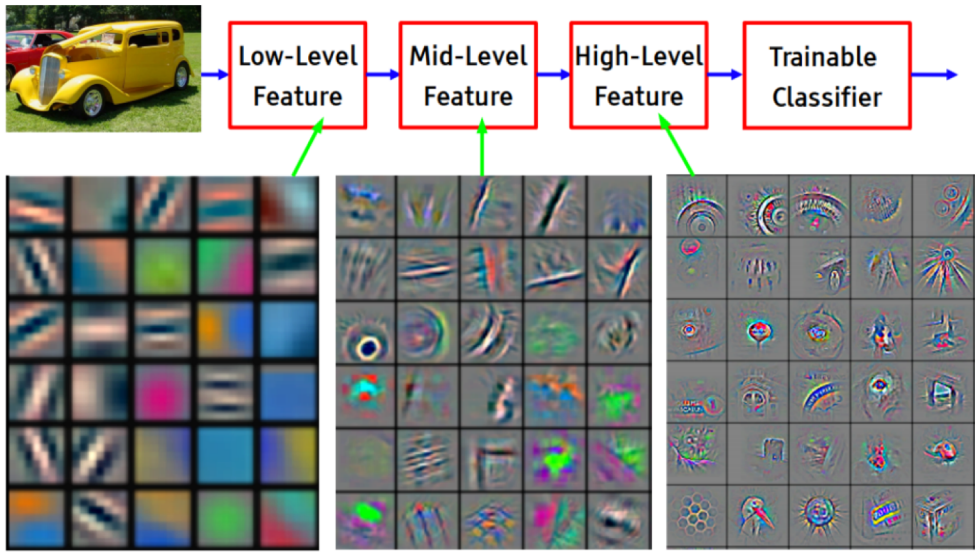


Convolutional Neural Networks

- Chain together with non-linearities and down-sampling (e.g. max-pooling)
- After processing with convolutions, use fully connected layers for classification
- Structure allows for capturing local structure in convolutions, and long range structure in later stage convolutions and in fully connected layers



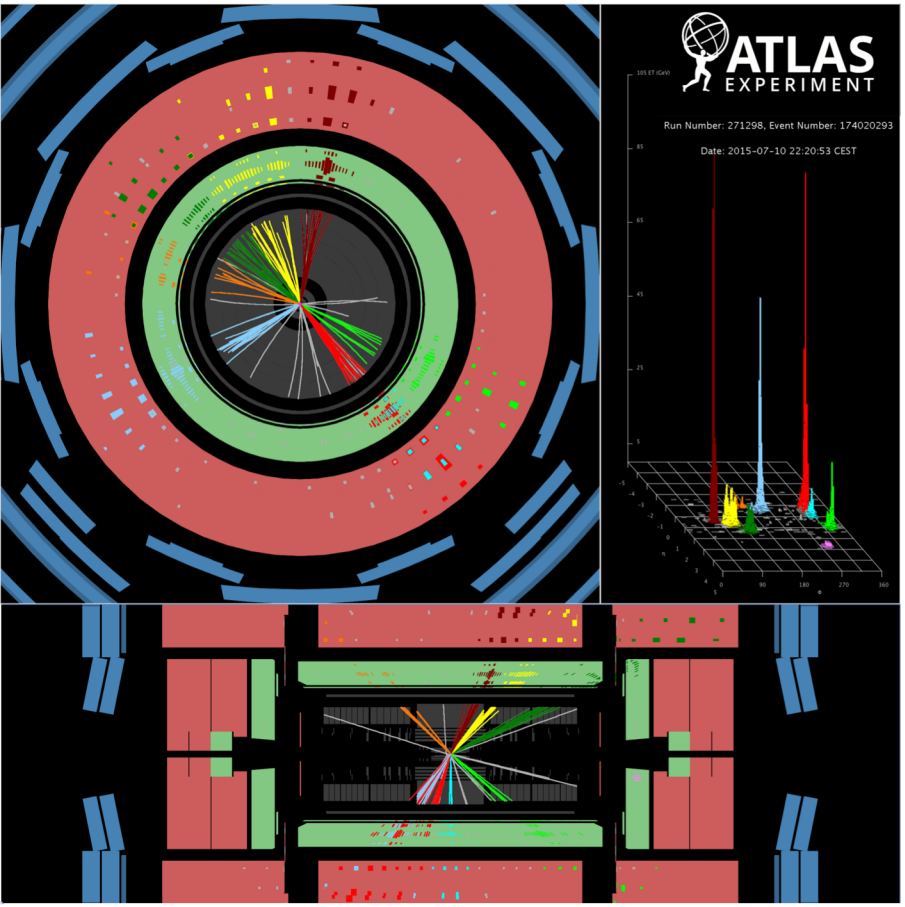
(Simonyan and Zisserman, 2014)



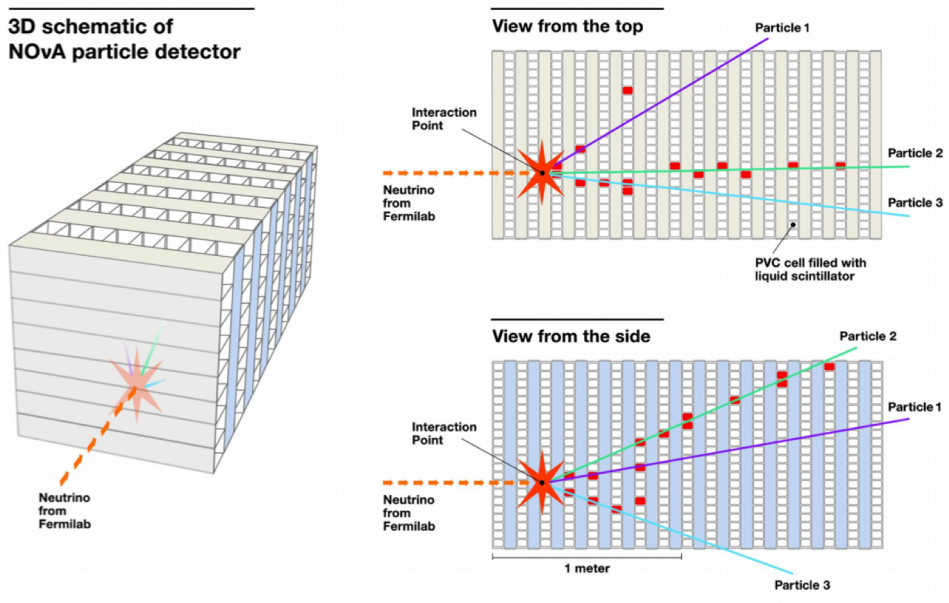
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Neural Networks in HEP

Jets at the LHC



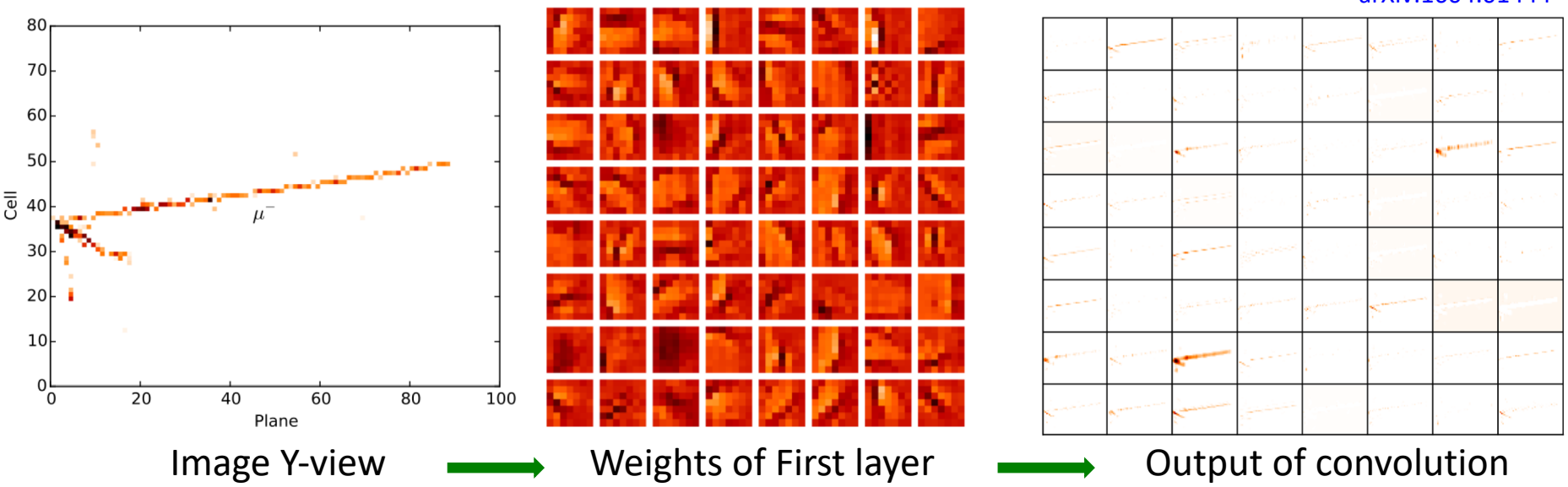
Neutrino identification
Example: NOνA



What do neural networks learn?

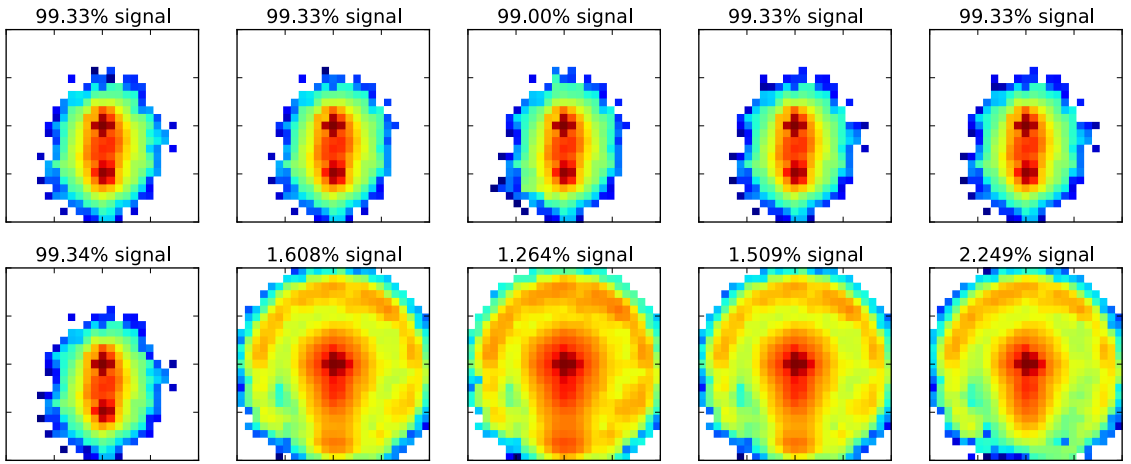
- Can visualize weights: neutrino decay classification

arXiv:1604.01444



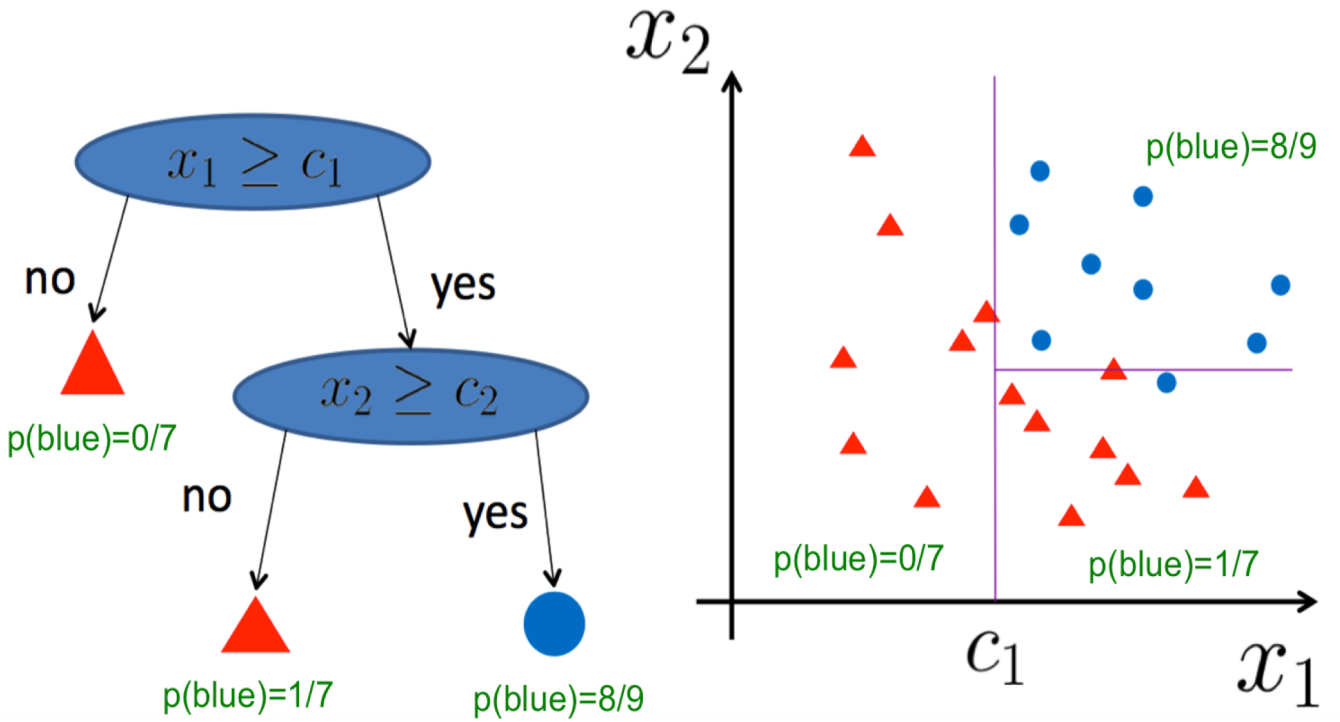
https://arxiv.org/abs/1511.05190

- Find inputs that most activate a neuron:
  - Separating boosted W-jets from quark/gluon jets



Decision Tree Models

Decision Trees



- Partition data based on a sequence of thresholds

- In a given partition, estimate the class probability from N_m examples in partition m and N_k of the examples in partition from class k:

$$p_{mk} = \frac{N_k}{N_m}$$

Single Decision Trees: Pros and Cons

- Pros:
  - Simple to understand, can visualize a tree
  - Requires little data preparation, and can use continuous and categorical inputs
- Cons:
  - Can create complex models that overfit data
  - Can be unstable to small variations in data
  - Training a tree is an NP-complete problem
    - Hard to find a global optimum of all data partitionings
    - Have to use heuristics like *greedy optimization* where locally optimal decisions are made
- We will discuss the ways to overcome these Cons, including early stopping of training, and ensembles

Greedy Training of a Decision Tree

- **Greedy Training**: instead of optimizing all splittings at the same time, optimize them one-by-one, then move onto next splitting

Greedy Training of a Decision Tree

- **Greedy Training**: instead of optimizing all splittings at the same time, optimize them one-by-one, then move onto next splitting

- Given N_m examples in a node, for a candidate splitting $\theta=(x_j , t_m)$ for feature x_j and threshold t_m

Greedy Training of a Decision Tree

- **Greedy Training**: instead of optimizing all splittings at the same time, optimize them one-by-one, then move onto next splitting

- Given N_m examples in a node, for a candidate splitting $\theta=(x_j, t_m)$ for feature x_j and threshold t_m

- If data partitioned into subsets Q_{left} and Q_{right}, compute:

$$G(Q,\theta) = \frac{n_{\text{left}}}{N_m} H(Q_{\text{left}}(\theta)) + \frac{n_{\text{right}}}{N_m} H(Q_{\text{right}}(\theta))$$

  - Where *H()* is an impurity function

Greedy Training of a Decision Tree

- **Greedy Training**: instead of optimizing all splittings at the same time, optimize them one-by-one, then move onto next splitting

- Given N_m examples in a node, for a candidate splitting $\theta=(x_j, t_m)$ for feature x_j and threshold t_m

- If data partitioned into subsets Q_{left} and Q_{right}, compute:

$$G(Q,\theta) = \frac{n_{\text{left}}}{N_m} H(Q_{\text{left}}(\theta)) + \frac{n_{\text{right}}}{N_m} H(Q_{\text{right}}(\theta))$$

  - Where *H()* is an impurity function

- Choose splitting θ using: $\theta^* = \arg\min_{\theta} G(Q,\theta)$

Impurity Functions

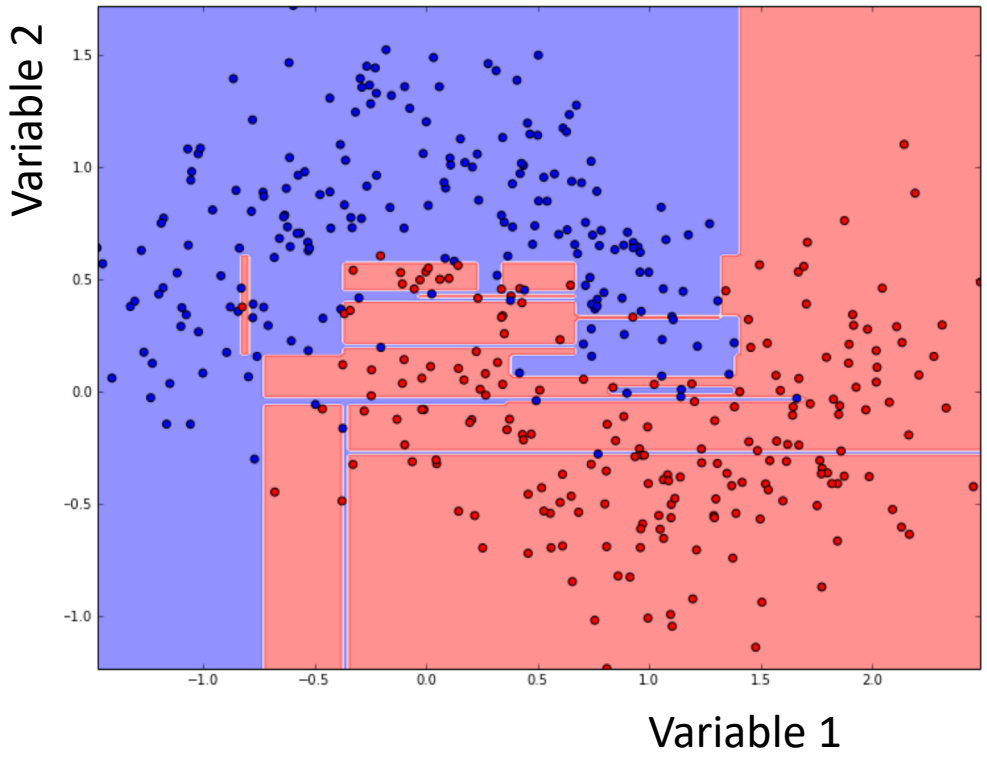
- Classification
  - Proportion of class k in node m: $p_{mk} = \frac{N_k}{N_m}$
  - Gini: $H(X_m) = \sum_k p_{mk}(1 - p_{mk})$
  - Cross entropy: $H(X_m) = -\sum_k p_{mk} \log(p_{mk})$
  - Miss-classification: $H(X_m) = 1 - \max_k(p_{mk})$
- Regression
  - Continuous target y, in region estimate: $c_m = \frac{1}{N_m} \sum_{i \in N_m} y_i$
  - Square error: $H(X_m) = \frac{1}{N_m} \sum_{i \in N_m} (y_i - c_m)^2$

When to stop splitting?

- In principle, can keep splitting until every event is properly classified…

When to stop splitting?

- In principle, can keep splitting until every event is properly classified…



[Rogozhnikov]

- Single decision trees can quickly overfit
- Especially when increasing the depth of the tree

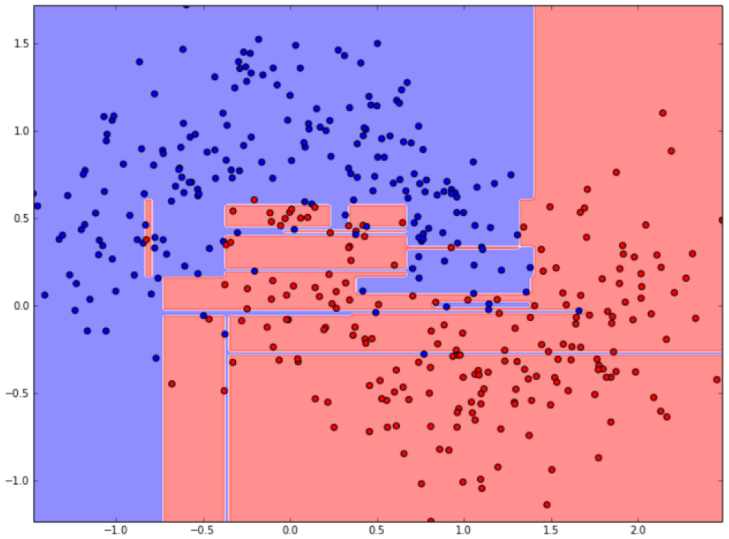
When to stop splitting?

- In principle, can keep splitting until every event is properly classified…

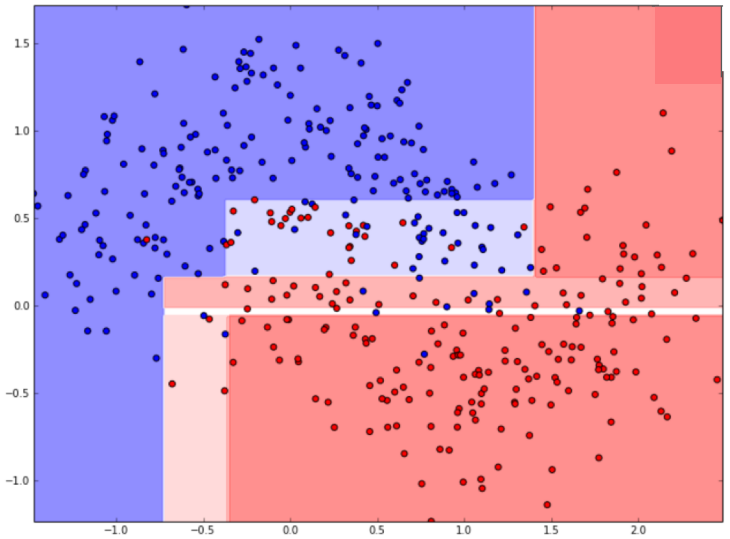
- Can stop splitting early. Many criteria:
  - Fixed tree depth
  - Information gain is not enough
  - Fix minimum samples needed in node
  - Fix minimum number of samples needed to split node

  - Combinations of these rules work as well

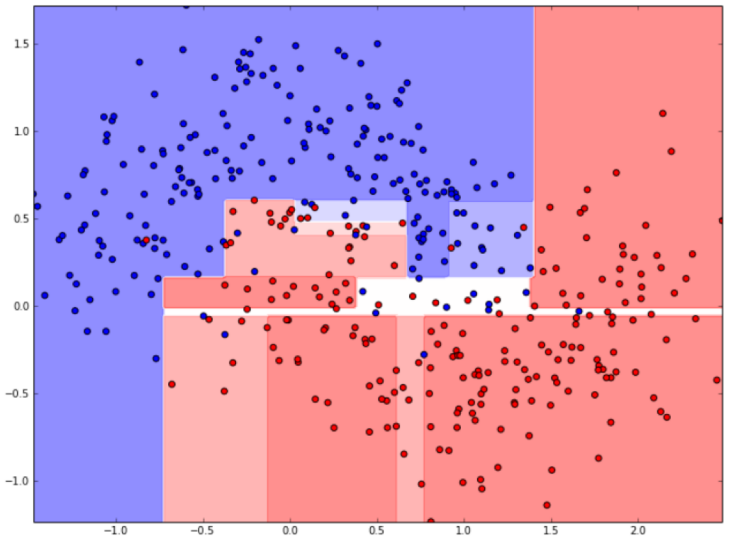
Mitigating Overfitting



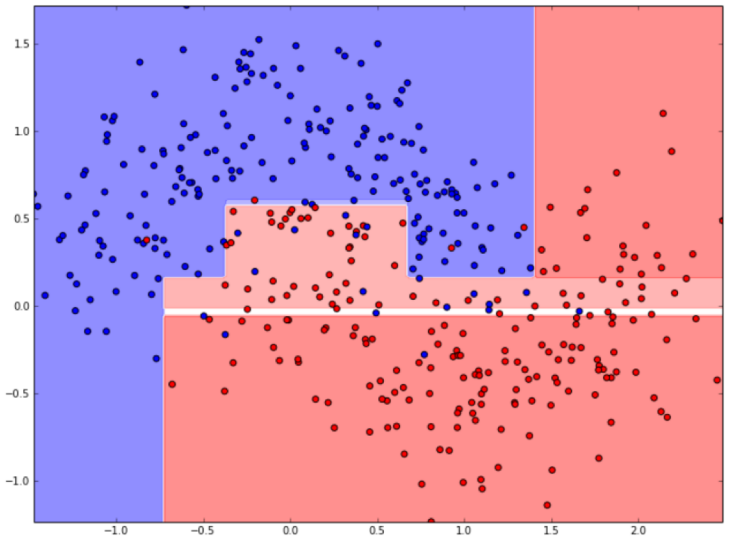
no pre-stopping



max_depth



min # of samples in leaf



maximal number of leaves

[Rogozhnikov]

Ensemble Methods

- Can we reduce the variance of a model without increasing the bias?

Ensemble Methods

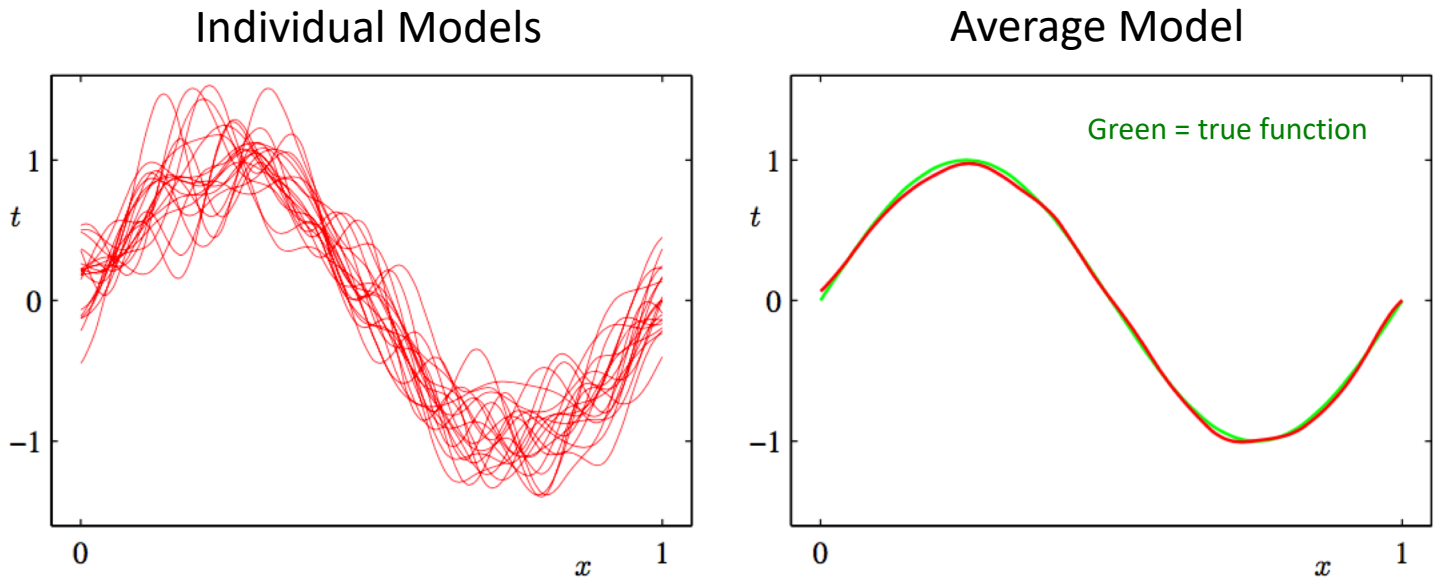
- Can we reduce the variance of a model without increasing the bias?

- Yes! By training several slightly different models and taking majority vote or average prediction

  – Bias does not largely increase because the average ensemble performance is equal to the average of its members

  – Variance decreases because a spurious pattern picked up by one model may not be picked up by other

Ensemble Methods



[Bishop]

- Combining several weak learners (only small correlation with target value) with high variance can be extremely powerful

- Can be used with decision trees to overcome their problems of overfitting!

Bagging and Boosting

- **Bootstrap Aggregating (Bagging)**:
  - Sample dataset D with replacement N-times, and train a separate model on each derived training set
  - Classify example with majority vote, or compute average output from each tree as model output

$$h(\mathbf{x}) = \frac{1}{N_{trees}} \sum_{i=1}^{N_{trees}} h_i(\mathbf{x})$$

- **Boosting**:
  - Train N models in sequence, giving more weight to examples not correctly classified by previous models
  - Take weighted vote to classify examples

$$h(\mathbf{x}) = \frac{\sum_{i=1}^{N_{trees}} \alpha_i h_i(\mathbf{x})}{\sum_{i=1}^{N_{trees}} \alpha_i}$$

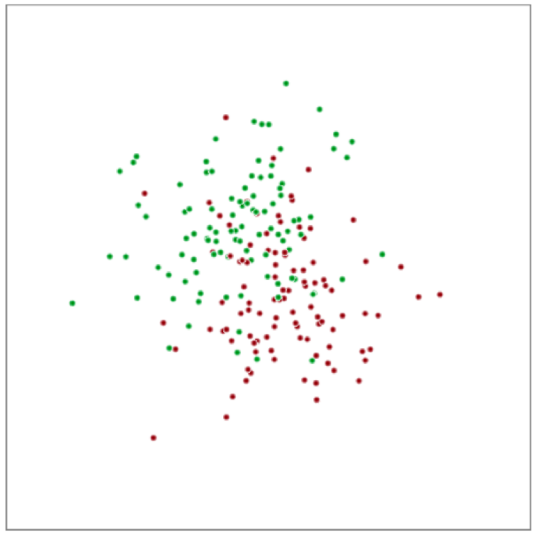
  - Boosting algorithms include:
    AdaBoost, Gradient boost, XGBoost

Random Forest

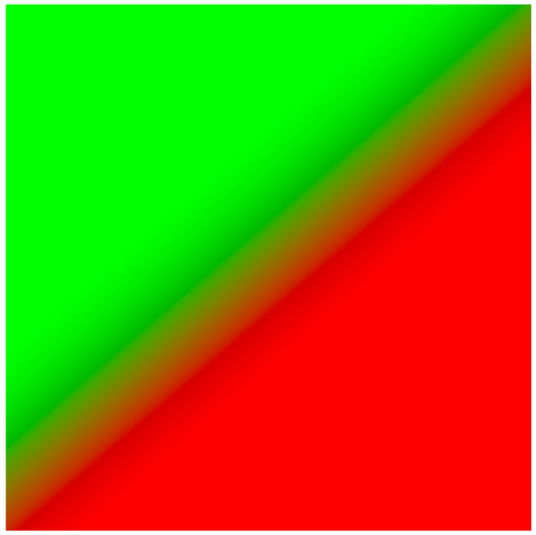
- One of the most commonly used algorithms in industry is the **Random Forest**
  - Use bagging to select random example subset
  - Train a tree, but only use random subset of features ($\sqrt{m}$ features) at each split. This increases the variance

Ensembles of Trees

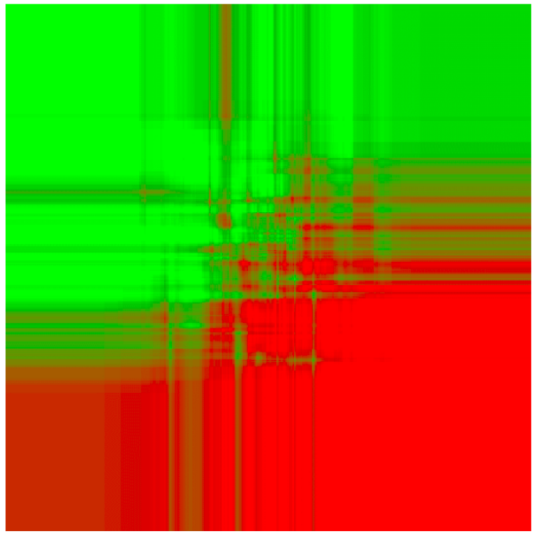
- Tree Ensembles tend to work well
  - Relatively simple
  - Relatively easy to train
  - Tend not to overfit (especially random forests)
  - Work with different feature types: continuous, categorical, etc.



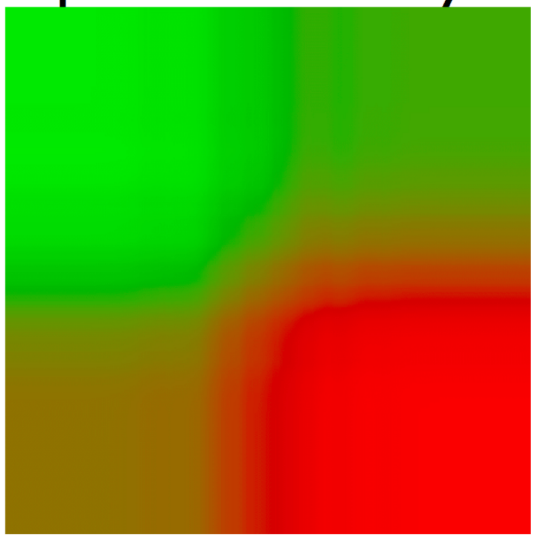
data



optimal boundary



50 trees



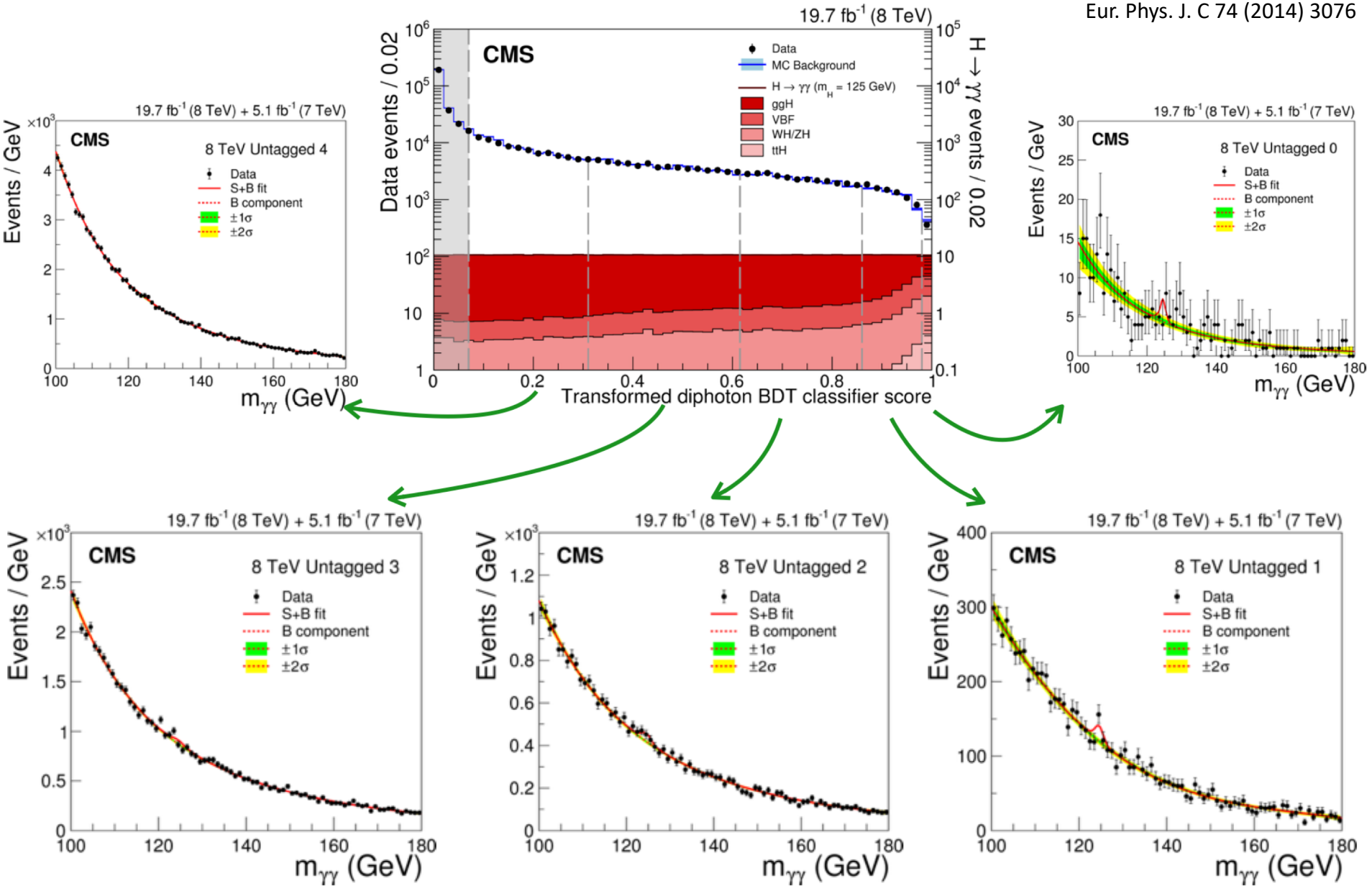
2000 trees

Random Forest

[Rogozhnikov]

CMS h→γγ (8 TeV) – Boosted decision tree

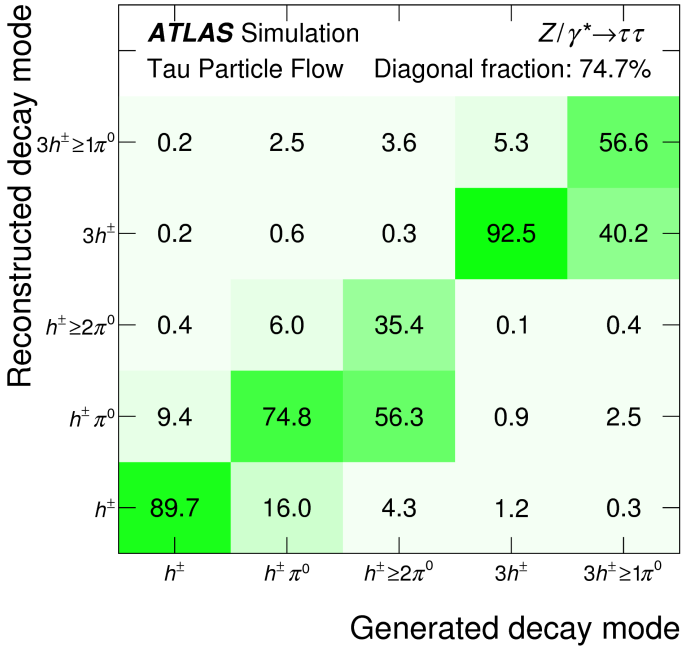
Eur. Phys. J. C 74 (2014) 3076

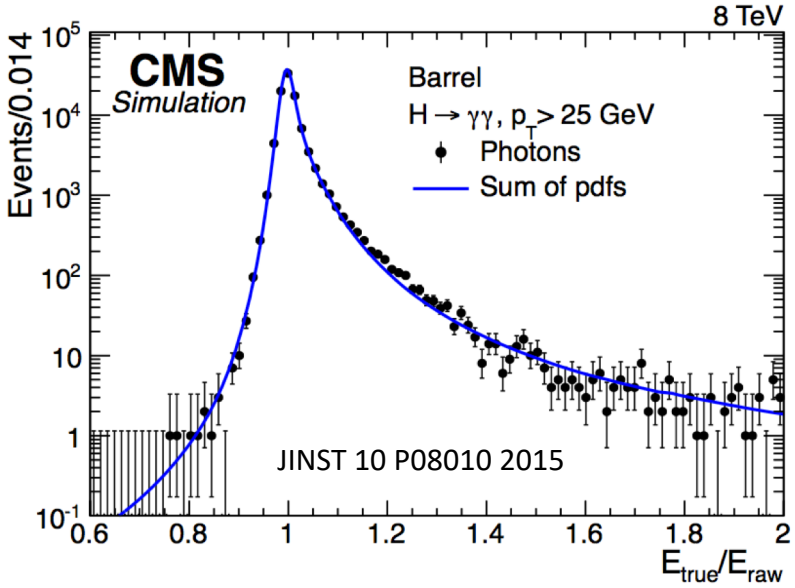


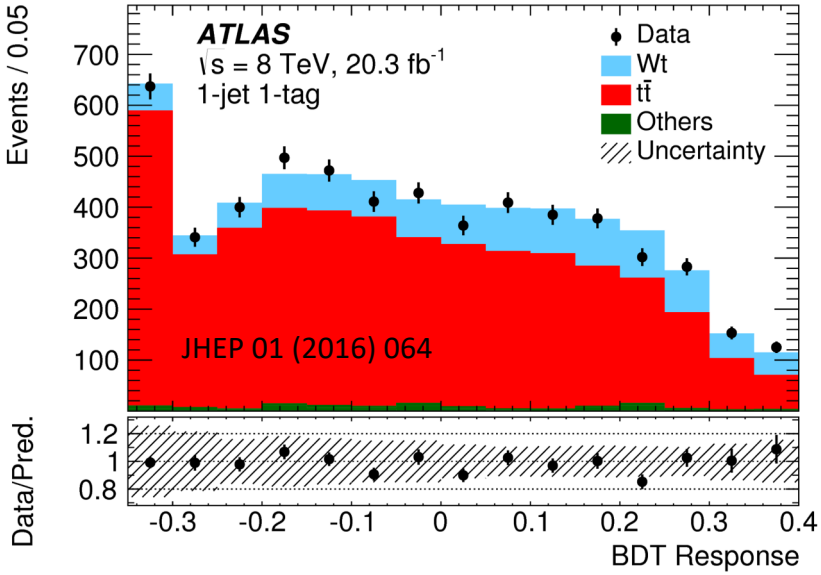
Decision Tree Ensembles in HEP

- Decision tree ensembles, especially with boosting, are used very widely in HEP!

https://arxiv.org/abs/1512.05955







Unsupervised Learning

- Learning without targets/labels, find structure in data

Dimensionality Reduction

- Find a low dimensional (less complex) representation of the data with a mapping $Z=h(X)$

Principle Components Analysis

- Given data $\{\mathbf{x}_i\}_{i=1\ldots N}$ can we find a directions in features space that explain most variation of data?

Principle Components Analysis

- Given data $\{\mathbf{x}_i\}_{i=1\ldots N}$ can we find a directions in features space that explain most variation of data?
- Data covariance: $S = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i - \overline{\boldsymbol{x}})(\boldsymbol{x}_i - \overline{\boldsymbol{x}})^T$

Principle Components Analysis

- Given data $\{\mathbf{x}_i\}_{i=1\ldots N}$ can we find a directions in features space that explain most variation of data?
- Data covariance: $S = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{x}_i - \bar{\boldsymbol{x}})^T$
- Let $\mathbf{u}_1$ be the projected direction, we can solve:

Variance of projected data

Unit length vector constraint

$$\mathbf{u}_1^* = \arg\max_{\mathbf{u}_1} \mathbf{u}_1^T \mathbf{S}\mathbf{u}_1 + \lambda(1 - \mathbf{u}_1^T\mathbf{u}_1)$$

$$\rightarrow \mathbf{S}\mathbf{u}_1 = \lambda\mathbf{u}_1$$

Principle Components Analysis

- Given data $\{\mathbf{x}_i\}_{i=1\ldots N}$ can we find a directions in features space that explain most variation of data?
- Data covariance: $S = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{x}_i - \bar{\boldsymbol{x}})^T$
- Let $\mathbf{u}_1$ be the projected direction, we can solve:

Variance of projected data

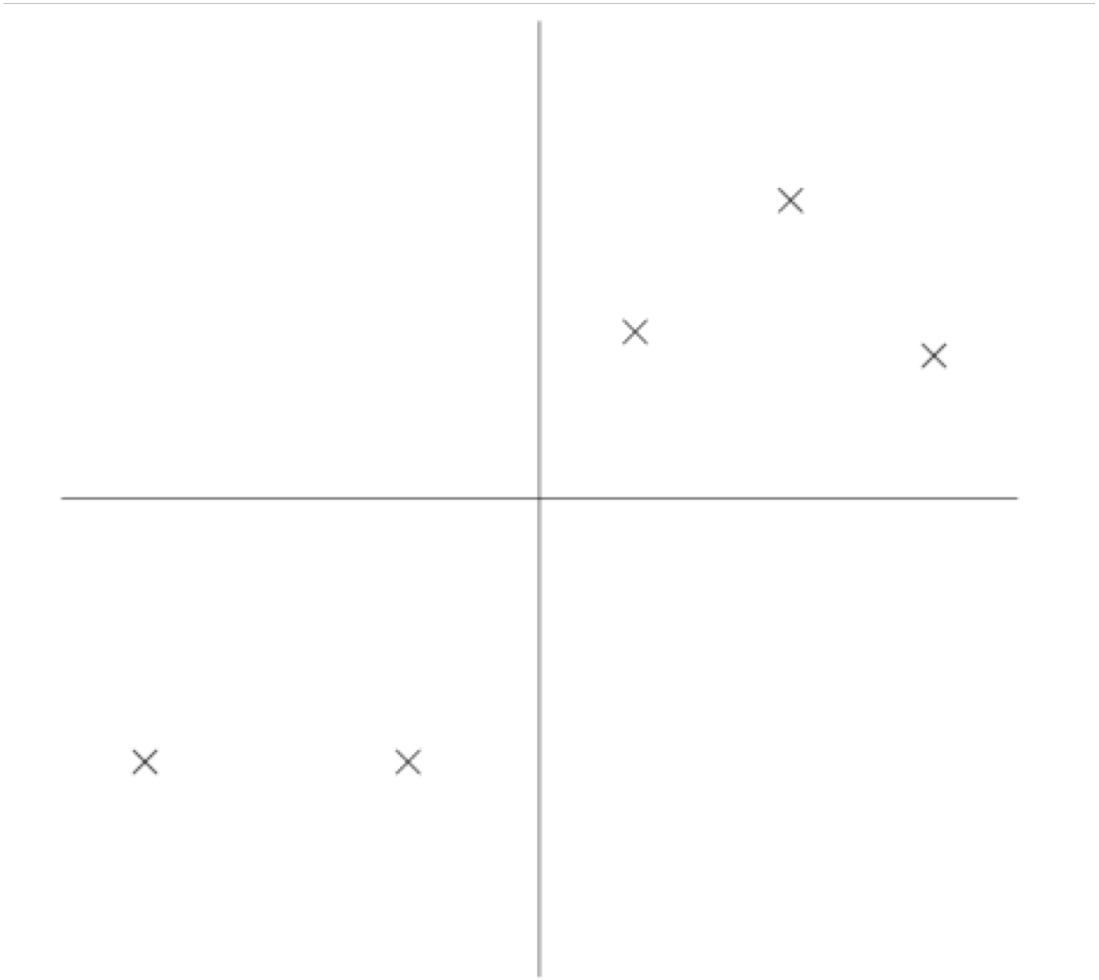
Unit length vector constraint

$$\mathbf{u}_1^* = \arg\max_{\mathbf{u}_1} \mathbf{u}_1^T \mathbf{S}\mathbf{u}_1 + \lambda(1 - \mathbf{u}_1^T\mathbf{u}_1)$$

$$\rightarrow \mathbf{S}\mathbf{u}_1 = \lambda\mathbf{u}_1$$

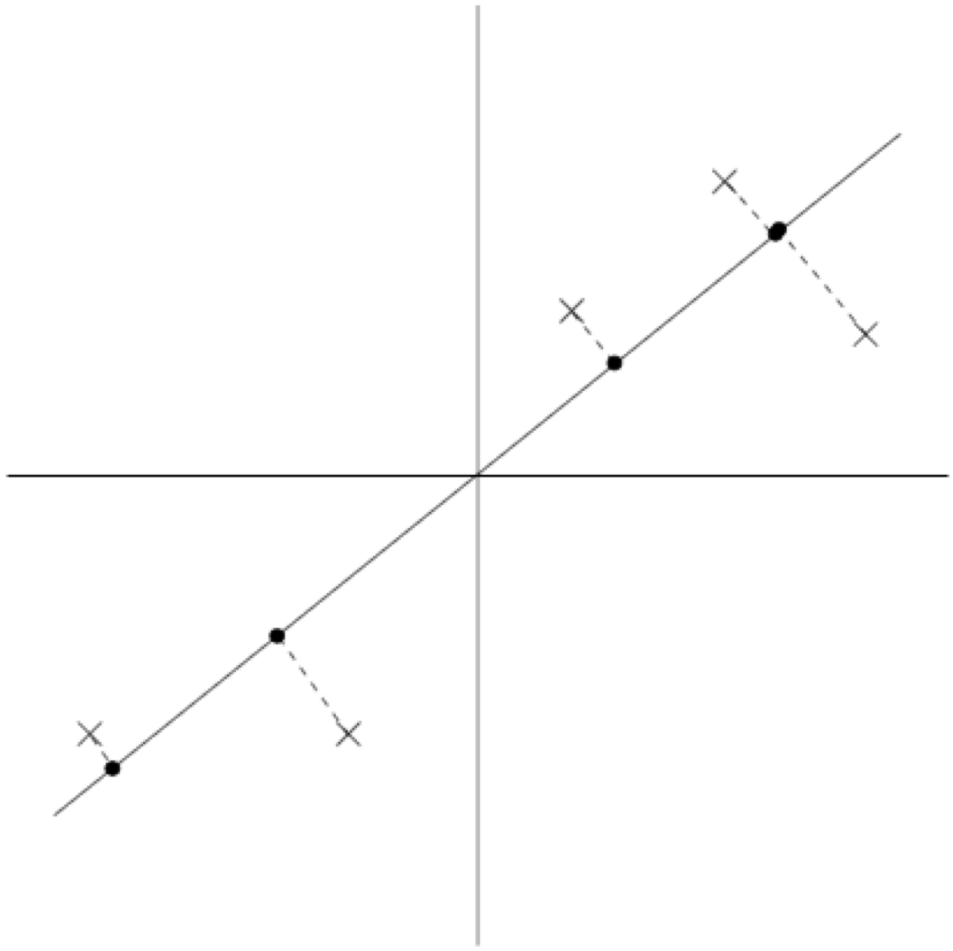
- *Principle components* are the eigenvectors of the data covariance matrix!
  - Eigenvalues are the variance explained by that component

PCA Example



[Ng]

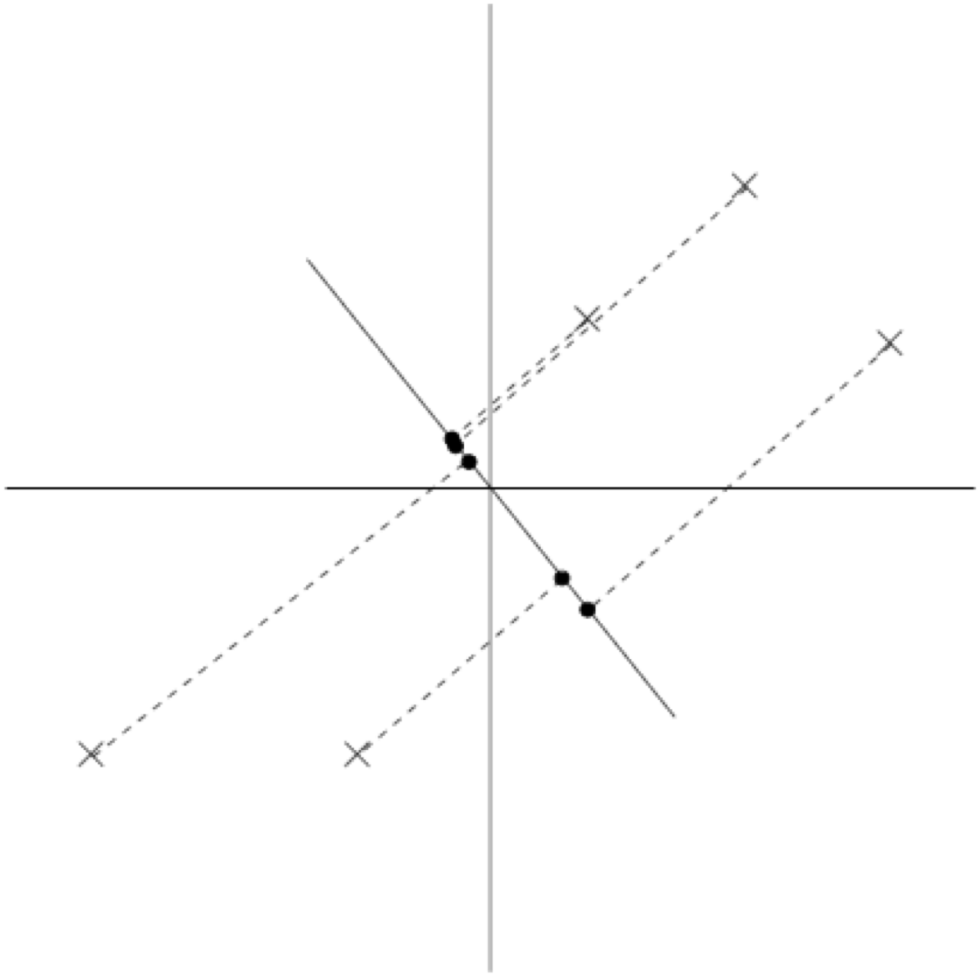
PCA Example



First principle component, projects on to this axis have large variance

[Ng]

PCA Example



Second principle component, projects have small variance

[Ng]

Clustering

Clustering

- Partition the data into groups $D=\{D_1 \cup D_2 \ \ldots \cup D_k\}$

- *What is a good clustering*?
  - One where examples within a cluster are more "similar" than to examples in other clusters

  - What does similar mean? Use distance metric, e.g.

$$d(\mathbf{x}, \mathbf{x}') = \sqrt{\sum_i (x_i - x'_i)^2}$$

K-means

- Data $\mathbf{x}_i \in \mathbb{R}^m$ which you want placed in K clusters

- Associate each example to a cluster by minimizing within-class variance

K-means

- Data $\mathbf{x}_i \in \mathbb{R}^m$ which you want placed in K clusters

- Associate each example to a cluster by minimizing within-class variance
  - Give each cluster S_k a prototype $\mu_k \in \mathbb{R}^m$ where $k=1\ldots K$

K-means

- Data $\mathbf{x}_i \in \mathbb{R}^m$ which you want placed in K clusters

- Associate each example to a cluster by minimizing within-class variance
  - Give each cluster S_k a prototype $\mu_k \in \mathbb{R}^m$ where $k=1\ldots K$

  - Assign each example to a cluster S_k

K-means

- Data $\mathbf{x}_i \in \mathbb{R}^m$ which you want placed in K clusters

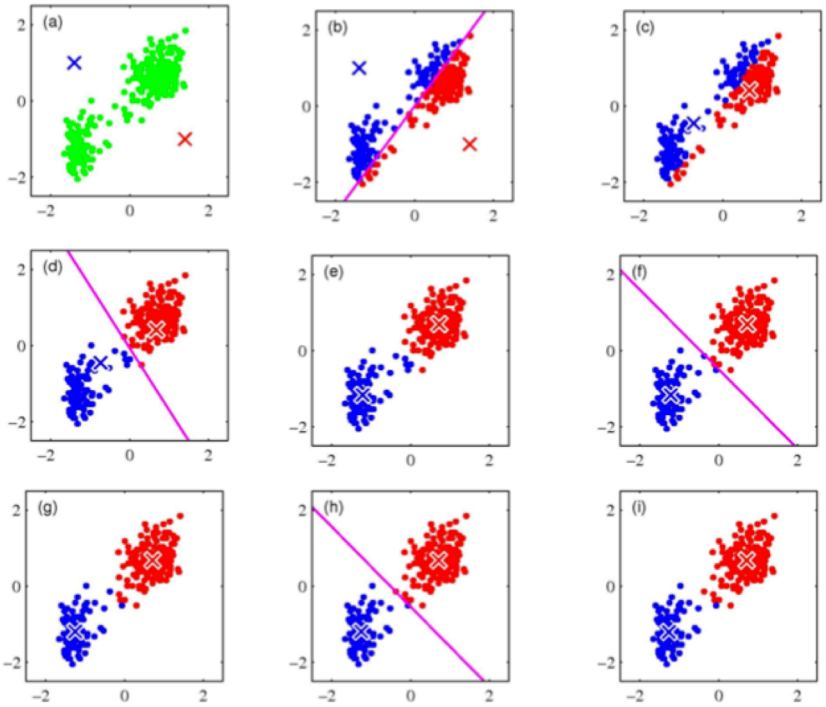
- Associate each example to a cluster by minimizing within-class variance
  - Give each cluster S_k a prototype $\mu_k \in \mathbb{R}^m$ where $k=1 \ldots K$
  - Assign each example to a cluster S_k
  - Find prototypes and assignments to minimize

$$L(S, \mu) = \sum_{k=1}^{K} \sum_{i \in S_k} \sqrt{(\mathbf{x}_i - \mu_k)^2}$$

    - This is an NP-hard problem, with many local minimum!

K-means algorithm

- Initialize the μ_k at random (typically using K-means++ initialization)
- **Repeat until convergence**:
  - Assign each example to closest prototype $\min_{k \in \{1 \ldots K\}} \sqrt{(\mathbf{x}_i - \mu_k)^2}$
  - Update prototypes $\mu_k = \frac{1}{n_k} \sum_{i \in S_k} \mathbf{x}_i$



[Bishop]

Practical Advice

What To Use? So Many Choices

- Once you know what you want to do…

  WHAT algorithm should you use?

  – Linear model
  – Nearest Neighbors
  – (Deep?) Neural network
  – Decision tree ensemble
  – Support vector machine
  – Gaussian processes
  – … and so many more …

No Free Lunch - Wolpert (1996)

- In the absence of prior knowledge, there is no a priori distinction between algorithms, no algorithm that will work best for every supervised learning problem
  - You can not say algorithm X will be better without knowing about the system
  - A model may work really well on one problem, and really poorly on another
  - This is why data scientists have to try lots of algorithms!
- But there are some empirical heuristics that have been observed…

Practical Advice – Empirical Analysis

- Test 179 classifiers (no deep neural networks) on 121 datasets
  http://jmlr.csail.mit.edu/papers/volume15/delgado14a/delgado14a.pdf

  – *The classifiers most likely to be the bests are the random forest (RF) versions, the best of which (…) achieves 94.1% of the maximum accuracy overcoming 90% in the 84.3% of the data sets*

From Kaggle

- For Structured data: "High level" features that have meaning
  – Winning algorithms have been lots of feature engineering + random forests, or more recently XGBoost (also a decision tree based algorithm)

- Unstructured data: "Low level" features, no individual meaning
  – Winning algorithms have been deep learning based, Convolutional NN for image classification, and Recurrent NN for text and speech

Conclusions

- Machine learning uses mathematical and statistical models learned from data to characterize patterns and relations between inputs, and use this for inference / prediction

- Machine learning provides a powerful toolkit to analyze data
  - Linear methods can help greatly in understanding data
  - Complex models like NN and decision trees can model intricate patterns
    - Care needed to train them and ensure they don't overfit
  - Unsupervised learning can provide powerful tools to understand data, even when no labels are available
  - Choosing a model for a given problem is difficult, but there may be some guidance in the literature
    - Keep in mind the bias-variance tradeoff when building an ML model

- Deep learning is an exciting frontier and powerful paradigm in ML research

Useful Python ML software

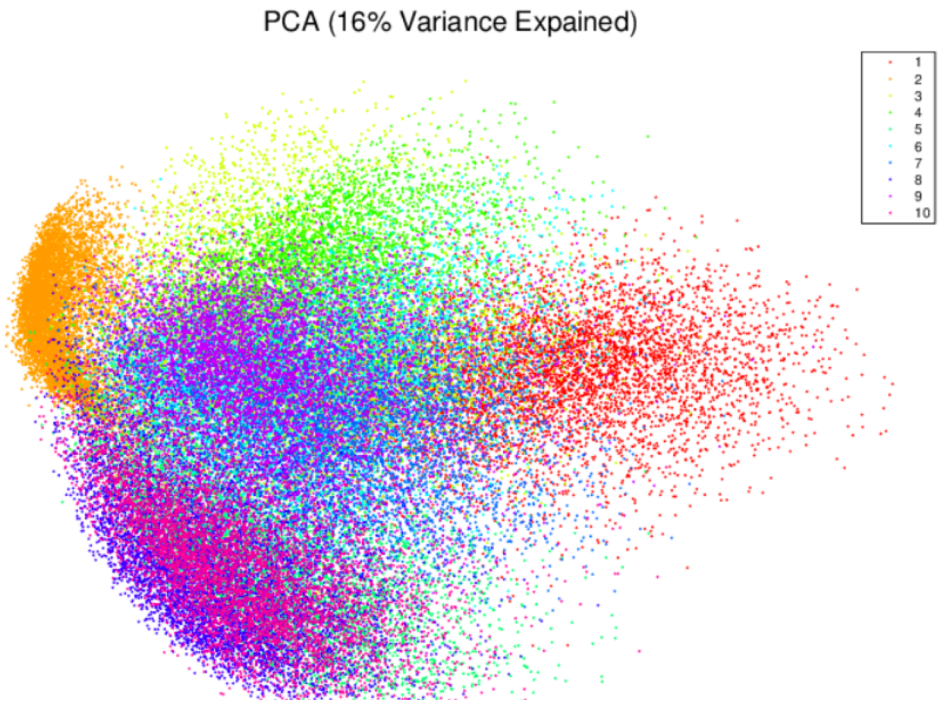
- Anaconda / Conda → easy to setup python ML / scientific computing environments
  - https://www.continuum.io/downloads
  - http://conda.pydata.org/docs/get-started.html
- Integrating ROOT / PyROOT into conda
  - https://nlesc.gitbooks.io/cern-root-conda-recipes/content/index.html
  - https://conda.anaconda.org/NLeSC
- Converting ROOT trees to python numpy arrays / panda dataframes
  - https://pypi.python.org/pypi/root_numpy/
  - https://github.com/ibab/root_pandas
- Scikit-learn → general ML library
  - http://scikit-learn.org/stable/
- Deep learning frameworks / auto-differentiation packages
  - https://www.tensorflow.org/
  - http://deeplearning.net/software/theano/
- High level deep learning package build on top of Theano / Tensorflow
  - https://keras.io/

References

- http://scikit-learn.org/
- [Bishop] Pattern Recognition and Machine Learning, Bishop (2006)
- [ESL] Elements of Statistical Learning (2nd Ed.) Hastie, Tibshirani & Friedman 2009
- [Murray] Introduction to machine learning, Murray
  - http://videolectures.net/bootcamp2010_murray_iml/
- [Ravikumar] What is Machine Learning, Ravikumar and Stone
  - http://www.cs.utexas.edu/sites/default/files/legacy_files/research/documents/MLSS-Intro.pdf
- [Parkes] CS181, Parkes and Rush, Harvard University
  - http://cs181.fas.harvard.edu
- [Ng] CS229, Ng, Stanford University
  - http://cs229.stanford.edu/
- [Rogozhnikov] Machine learning in high energy physics, Alex Rogozhnikov
  - https://indico.cern.ch/event/497368/
- [Fleuret] Francois Fleuret, EE559 Deep Learning, EPFL, 2018
  - https://documents.epfl.ch/users/f/fl/fleuret/www/dlc/

Example

- Classifying hand written digits
  - 10-class classification
  - Right plot shows projection of 10-class output onto 2 dimensions



Error Analysis

- Anti-spam classifier using logistic regression.
- How much did each component of the system help?
- Remove each component one at a time to see how it breaks

Component	Accuracy
Overall system	99.9%
Spelling correction	99.0
Sender host features	98.9%
Email header features	98.9%
Email text parser features	95%
Javascript parser	94.5%
Features from images	94.0%

Removing text parser caused largest drop in performance

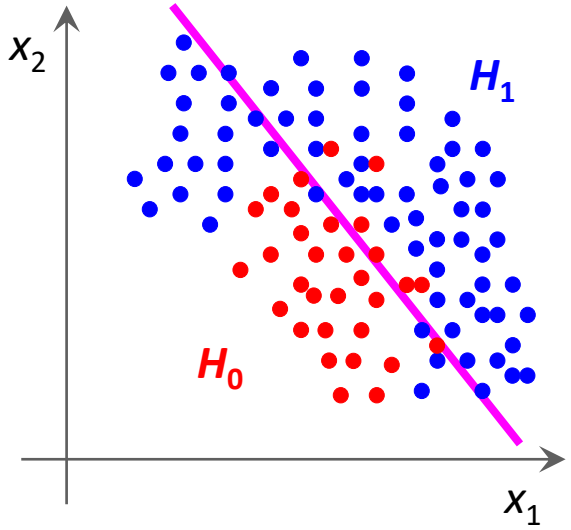
[baseline]

Ensemble Methods

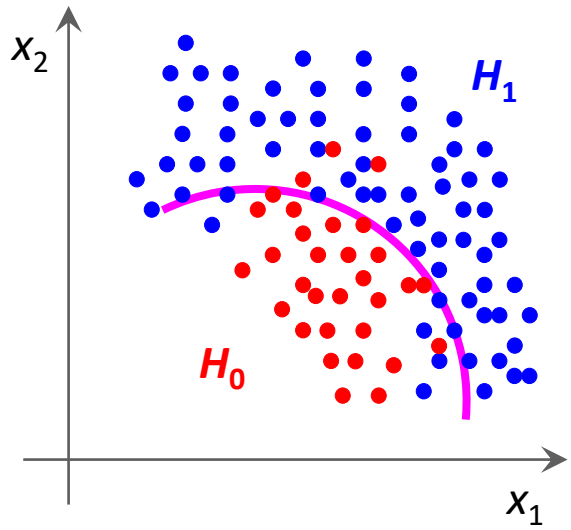
- Combine many decision trees, use the ensemble for prediction
- Averaging: $D(x) = \frac{1}{N_{tree}} \sum_{i=1}^{N_{tree}} d_i(x)$
  - **Random Forest**, averaging combined with:
    - **Bagging:** Only use a subset of events for each tree training
    - **Feature subsets**: Only use a subset of features for each tree
- Boosting (weighted voting): $D(x) = \sum_{i=1}^{N_{tree}} \alpha_i d_i(x)$
  - Weight computed such that events in current tree have higher weight misclassified in previous trees
  - Several boosting algorithms
    - AdaBoost
    - Gradient Boosting
    - XGBoost

Non-Linear Activations

- The activation function in the NN must be a non-linear function
  - If all the activations were linear, the network would be linear:
    $f(X) = W_n(W_{n-1} (\dots W_1 X)) = UX, \quad$ where $U = \Pi_i W_i$
- Linear functions can only correctly classify linearly separable data!
- For complex datasets, need nonlinearities to properly learn data structure



Linear Classifier



Non-linear Classifier

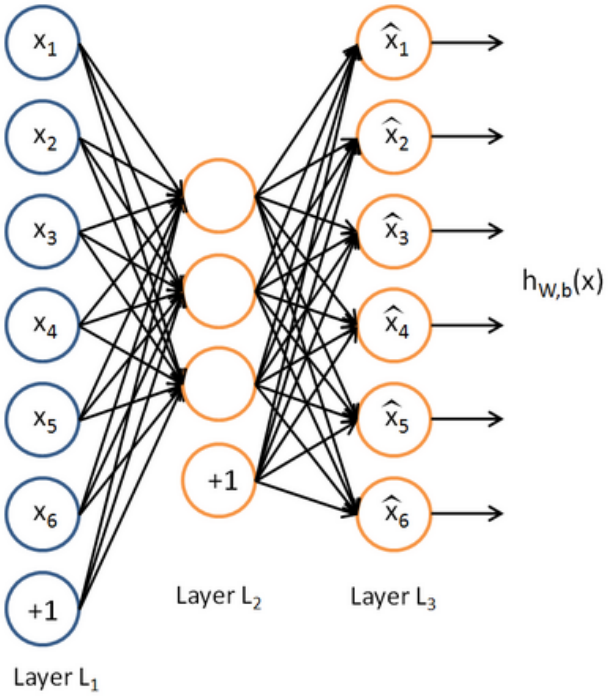
Neural Networks and Local Minima

- Large NN's difficult to train…trapping in local minimum?

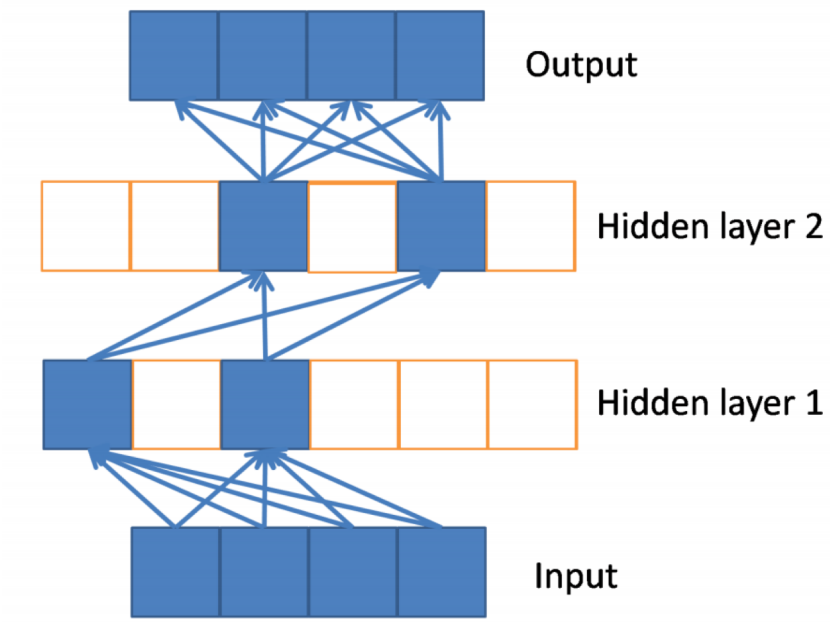
- Not in large neural networks *https://arxiv.org/abs/1412.0233*
  - Most local minima equivalent, and resonable
  - Global minima may represent overtraining
  - Most bad (high error) critical points are saddle points (different than small NN's)

Weight Initializations and Training Procedures

- Used to set weights to some small initial value
  - Creates an almost linear classifier
- Now initialize such that node outputs are normally distributed
- Pre-training with auto-encoder
  - Network reproduces the inputs
  - Hidden layer is a non-linear dimensionality reduction
  - Learn important features of the input
  - Not as common anymore, except in certain circumstances…
- Adversarial training, invented 2014
  - Will potential HEP applications later



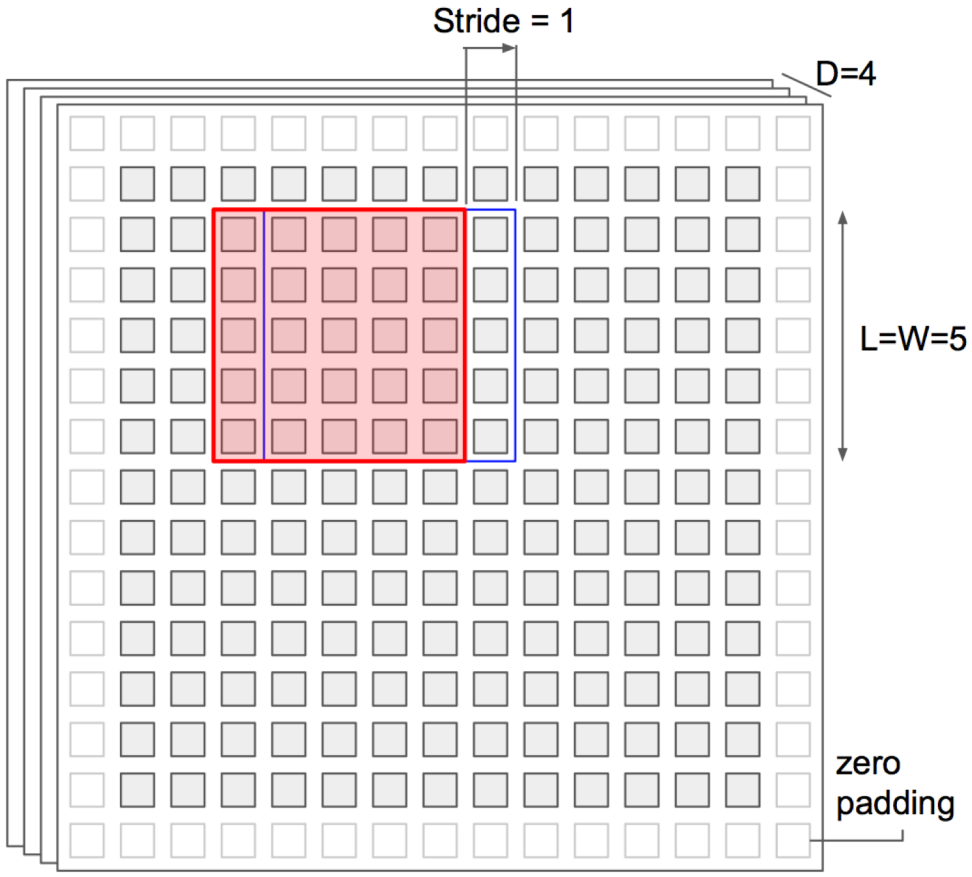
ReLU Networks



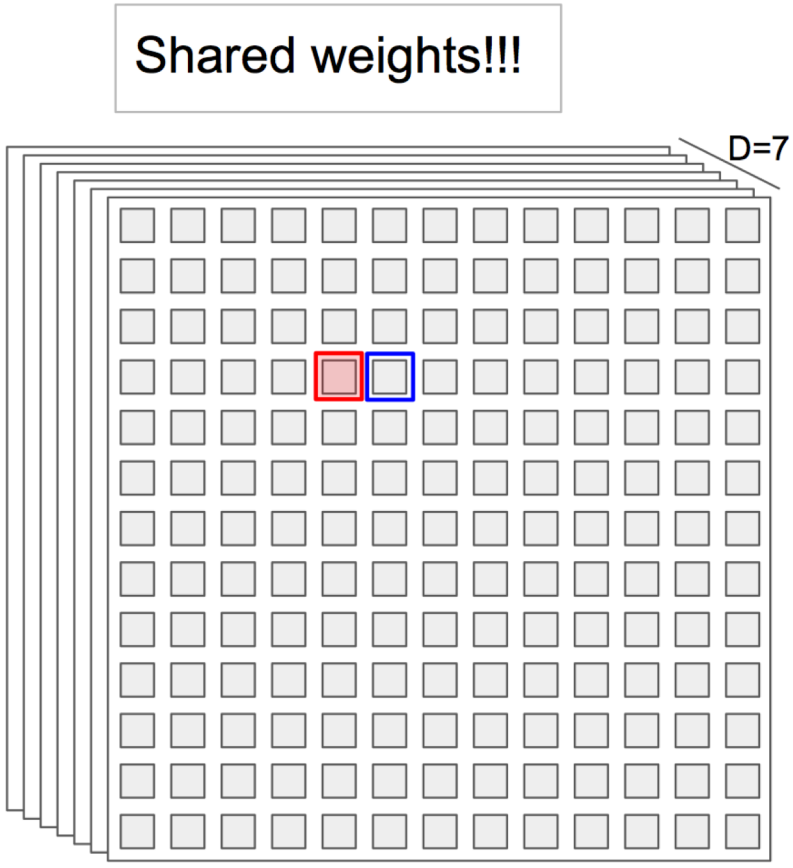
http://www.jmlr.org/proceedings/papers/v15/glorot11a/glorot11a.pdf

- Sparse propagation of activations and gradients in a network of rectifier units. The input selects a subset of active neurons and computation is linear in this subset.
- Model is "linear-by-parts", and can thus be seen as an exponential number of linear models that share parameters
- Non-linearity in model comes from path selection

Convolutions in 2D



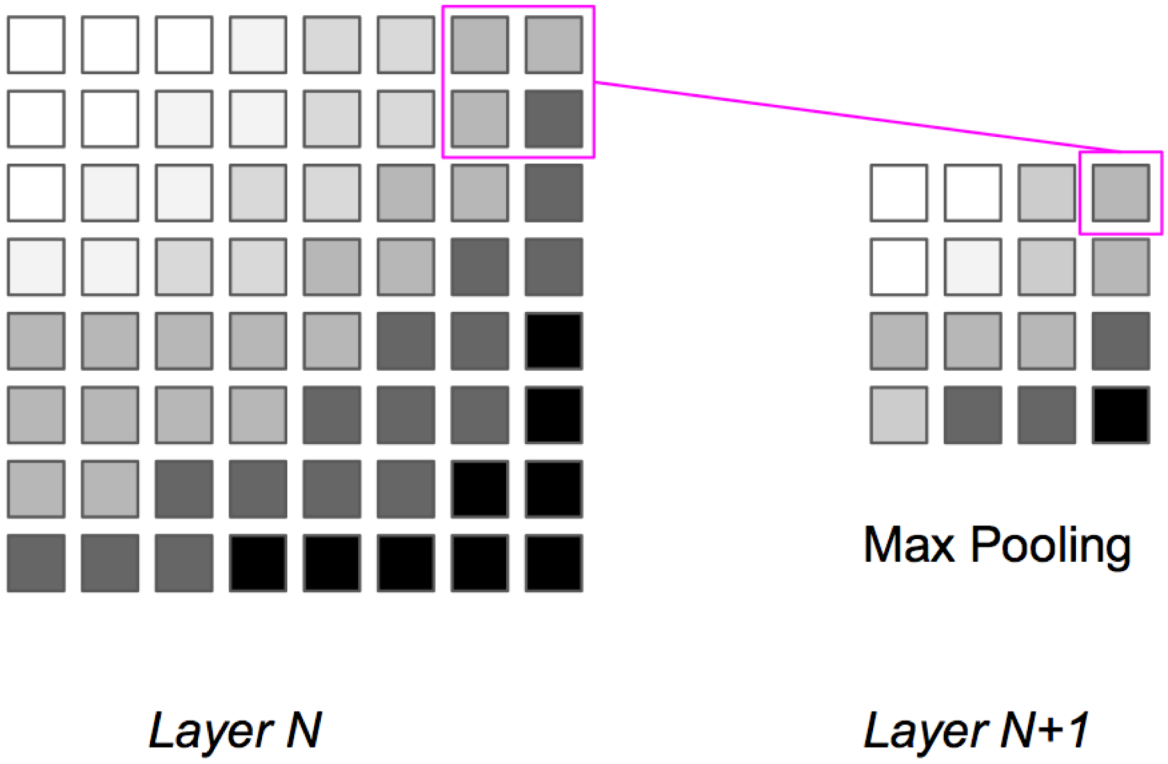
Input image



Convolved image

- Scan the filters over the 2D image, producing the convolved images

Max Pooling



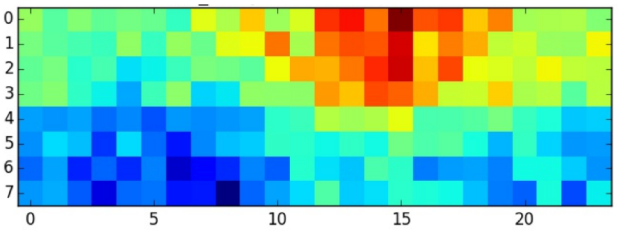
- Down-sample the input by taking MAX or average over a region of inputs
  - Keep only the most useful information

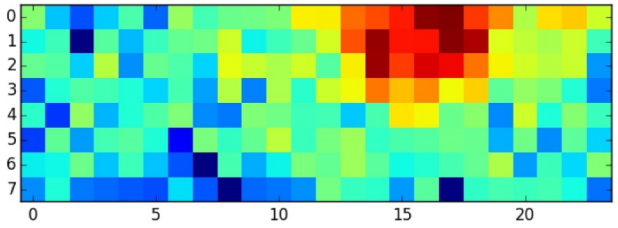
Daya Bay

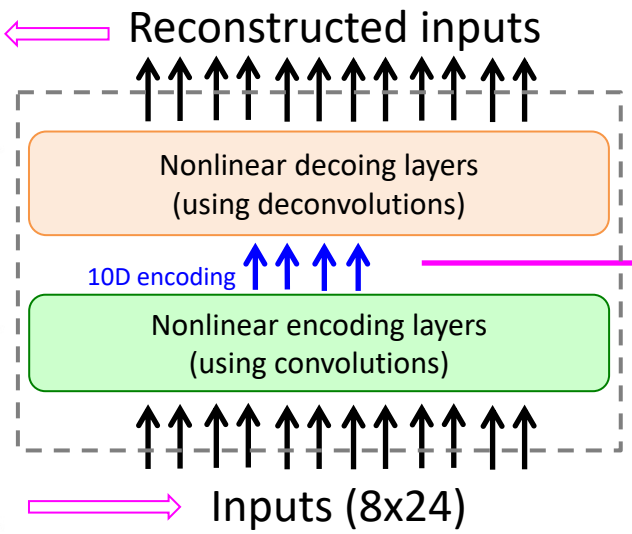
Daya Bay Neutrino Experiment

arXiv:1601.07621

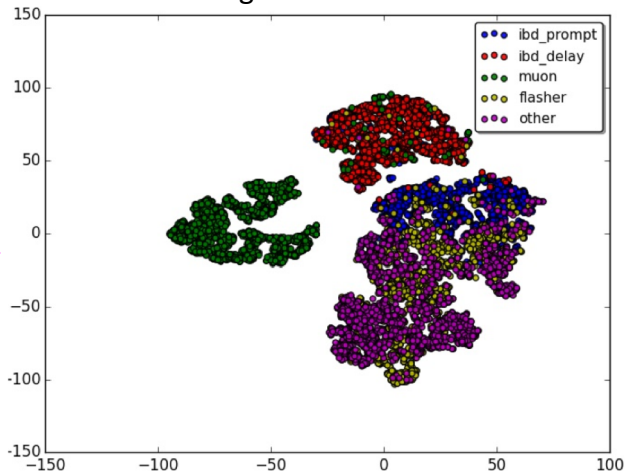
- Aim to reconstruct inverse β-decay interactions from scintillation light recorded in 8x24 PMT's
- Study discrimination power using CNN's
  - Supervised learning → observed excellent performance (97% accuracy)
  - Unsupervised learning: ML learns itself what is interesting!







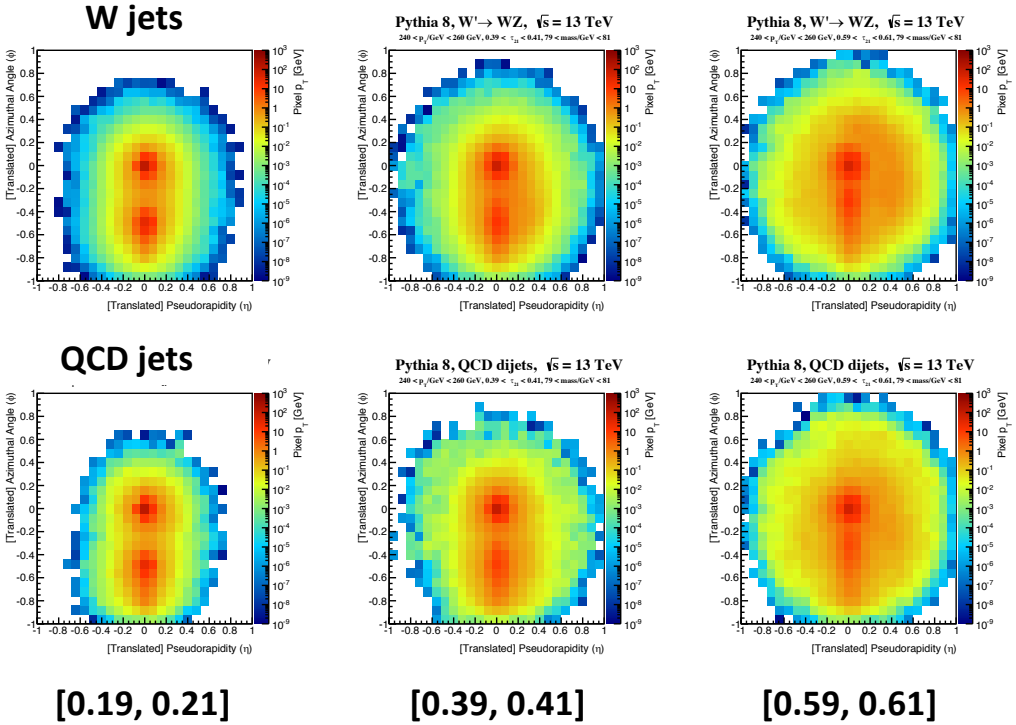
2D distant preserving representation of 10D encoding of events



Jet-Images

Restricted phase space

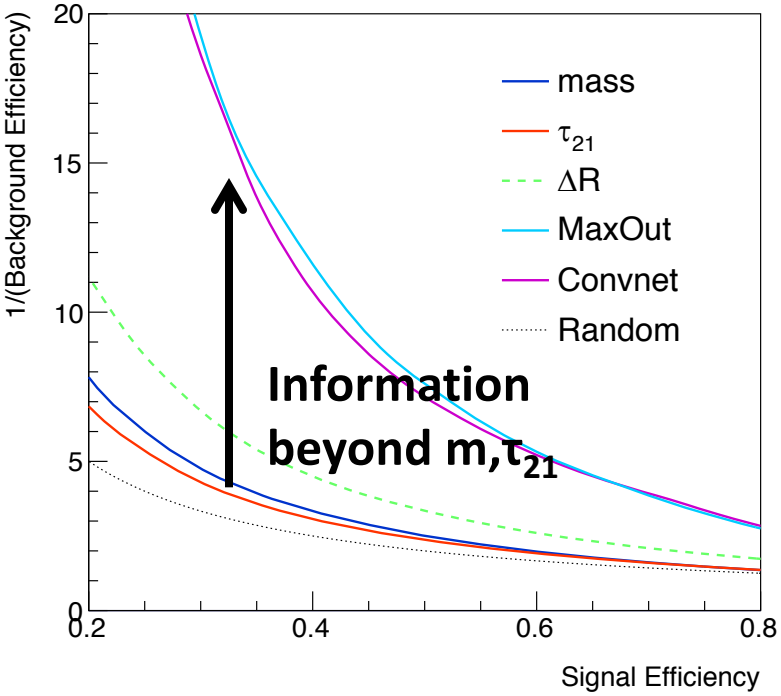
2-prong τ_{21} **1-prong**



[0.19, 0.21] **[0.39, 0.41]** **[0.59, 0.61]**

79 < m < 81 GeV

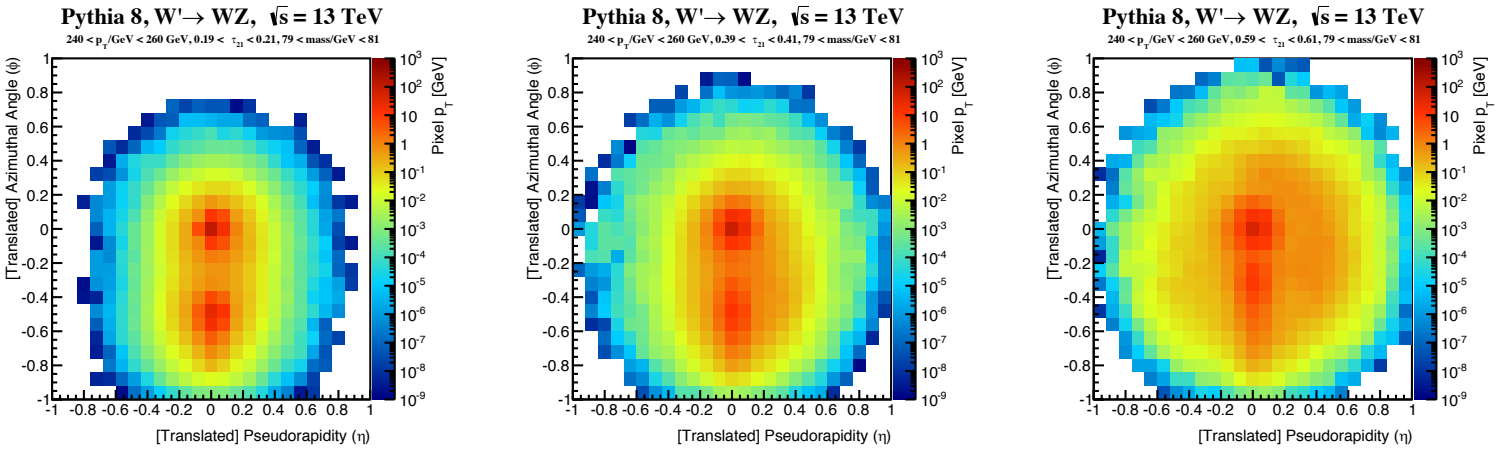
0.19 < τ_{21} <0.21

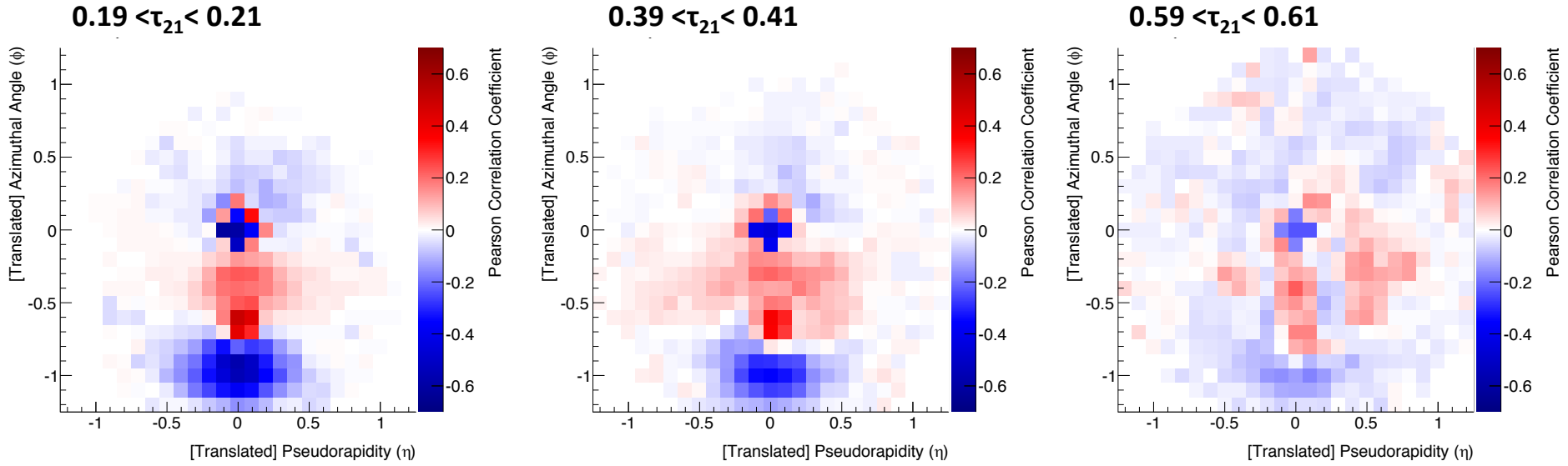


Restrict the phase space in very small mass and τ_{21} bins:

Improvement in discrimination from new, unique, information learned by the network

Deep correlation jet images





Spatial information indicative of radiation pattern for W and QCD: where in the image the network is looking for discriminating features

Recurrent Neural Networks

Recurrent Neural Networks

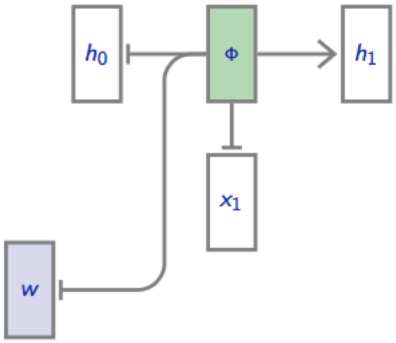
- What if our data doesn't have a fixed size? How do we process a variable length set of inputs

- More specifically, what if our data is sequence like?

$$x_i = \{x_i^0, x_i^1, \ldots, x_i^T\} = \{x_i^t\}_{t=0}^{T}$$

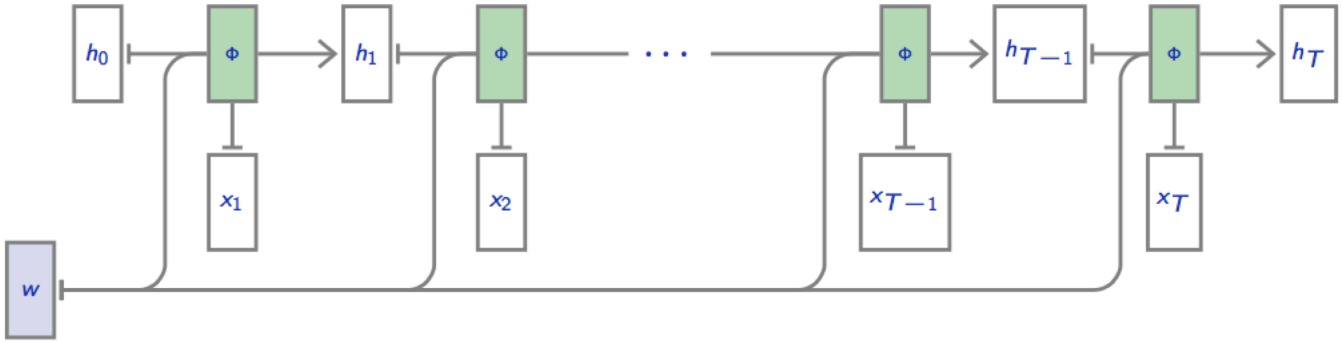
  - Natural language text
  - time-series data, like financial data
  - Ordered sets of particles, e.g. tracks in a jet

Recurrent Neural Networks



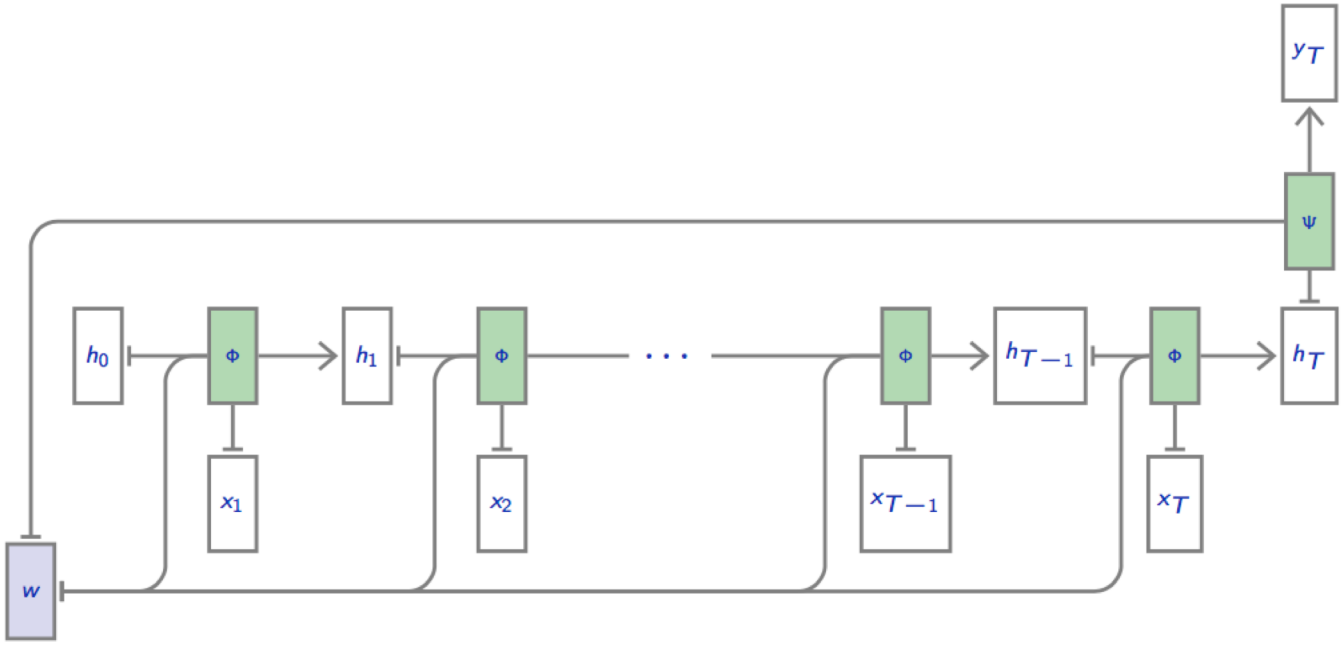
[Fleuret]

Recurrent Neural Networks



[Fleuret]

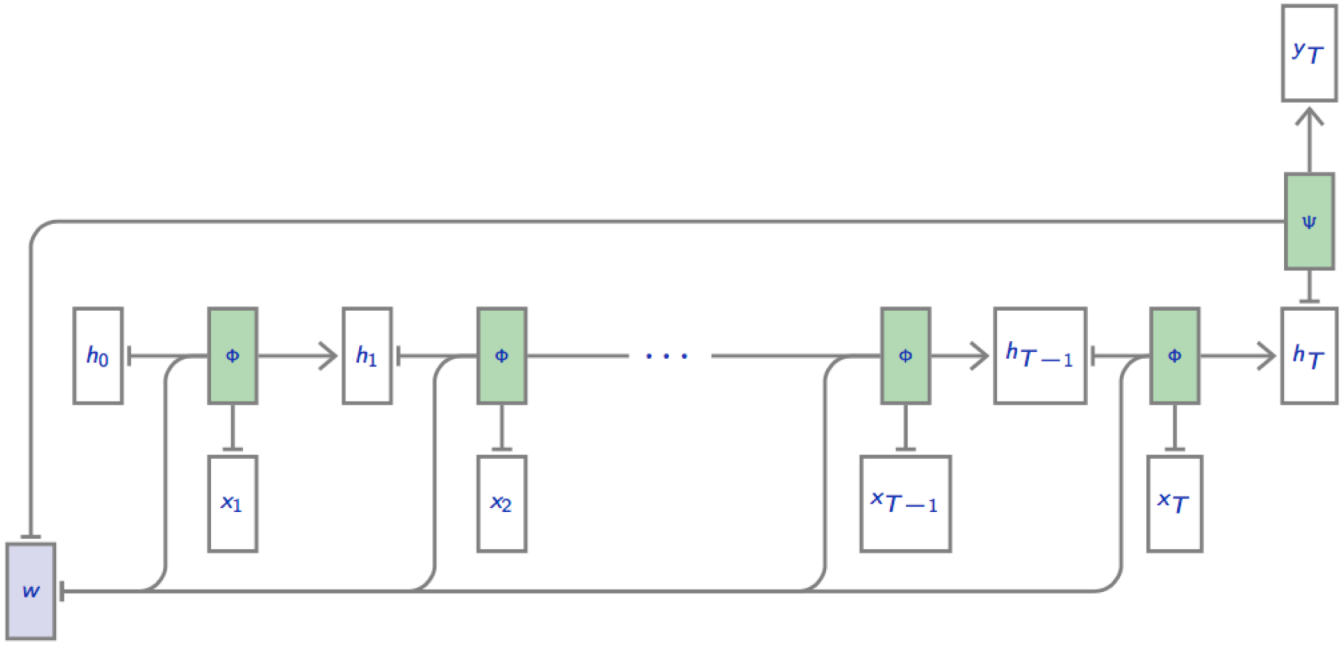
Recurrent Neural Networks



[Fleuret]

Recurrent Neural Networks

[0.98] → Positive Sentiment

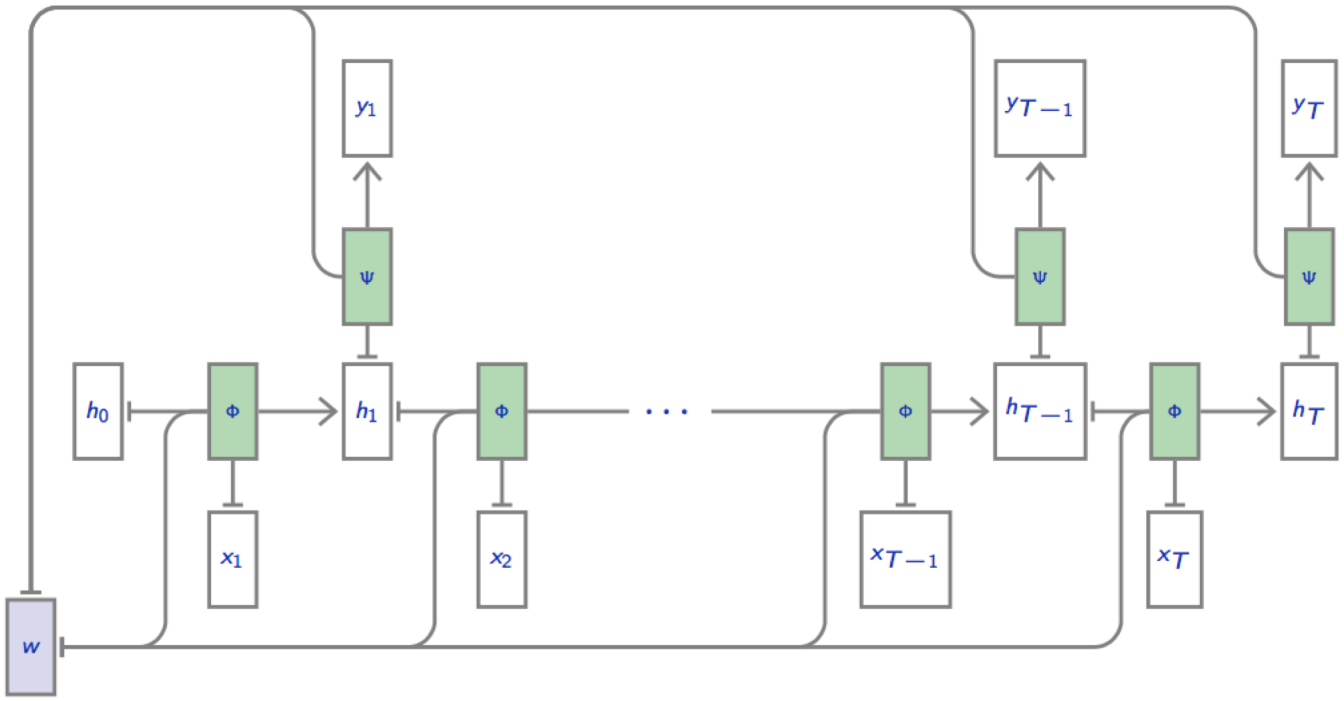


Sentiment Analysis

The movie was great

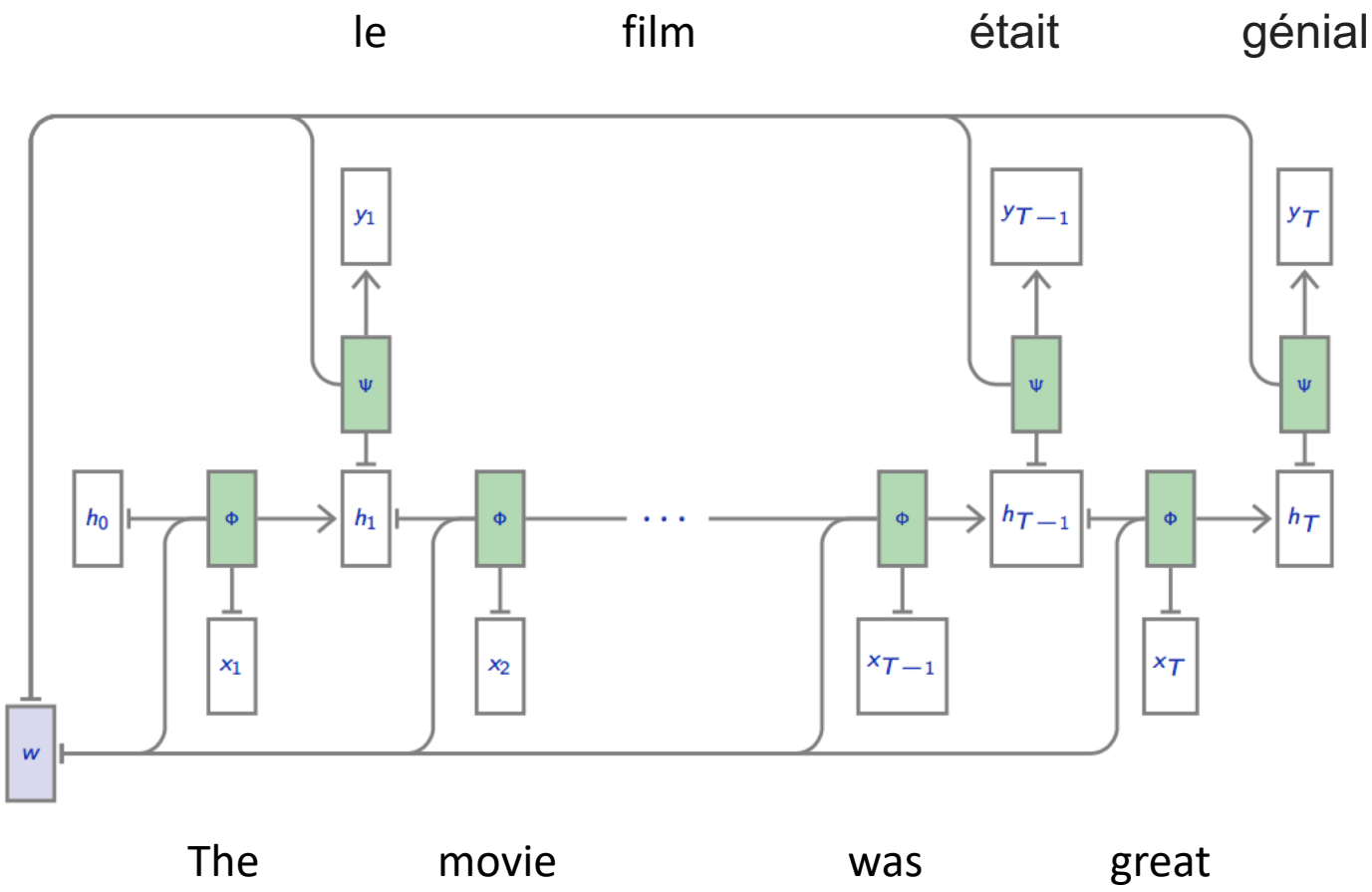
[Fleuret]

Recurrent Neural Networks



[Fleuret]

Recurrent Neural Networks



Translation

[Fleuret]

Recurrent Neural Networks

- In practice, a simple non-linearity is very hard to deal with
  - Hard to train
  - Hard to retain information across long sequences

- Utilize Gating
  - Long Short Term Memory (LSTM)
  - Gated Recurrent Unit (GRU

$$
\begin{aligned}
f_t &= \text{sigm}\left(W_{(\text{x f})}x_t + W_{(\text{h f})}h_{t-1} + b_{(\text{f})}\right) && \text{(forget gate)}\\
i_t &= \text{sigm}\left(W_{(\text{x i})}x_t + W_{(\text{h i})}h_{t-1} + b_{(\text{i})}\right) && \text{(input gate)}\\
g_t &= \tanh\left(W_{(\text{x c})}x_t + W_{(\text{h c})}h_{t-1} + b_{(\text{c})}\right) && \text{(full cell state update)}\\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t && \text{(cell state)}\\
o_t &= \text{sigm}\left(W_{(\text{x o})}x_t + W_{(\text{h o})}h_{t-1} + b_{(\text{o})}\right) && \text{(output gate)}\\
h_t &= o_t \odot \tanh(c_t) && \text{(output state)}
\end{aligned}
$$

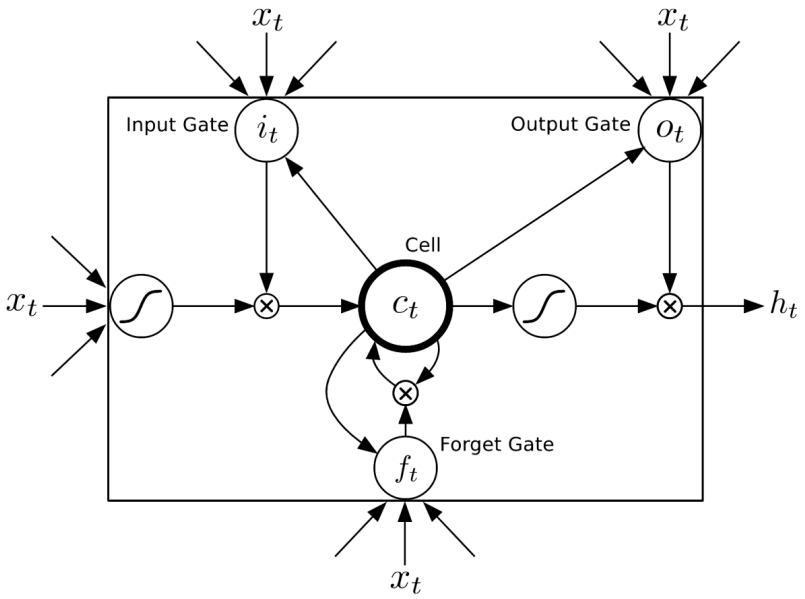
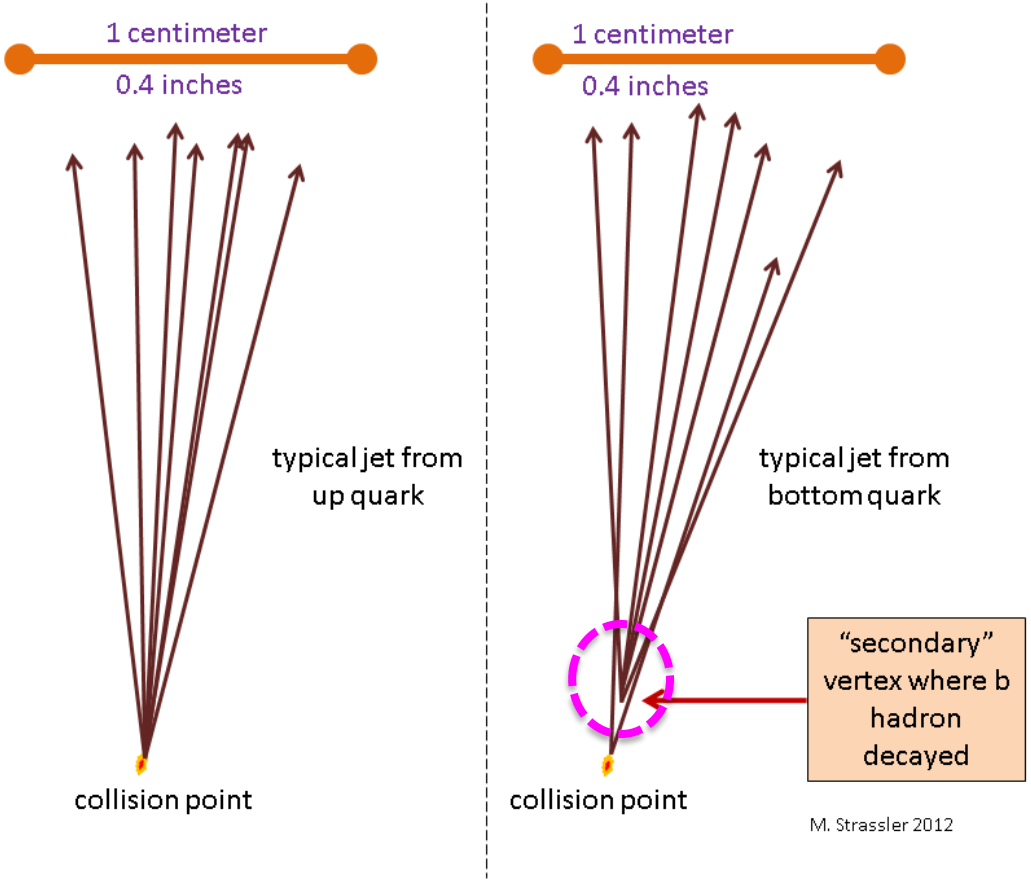
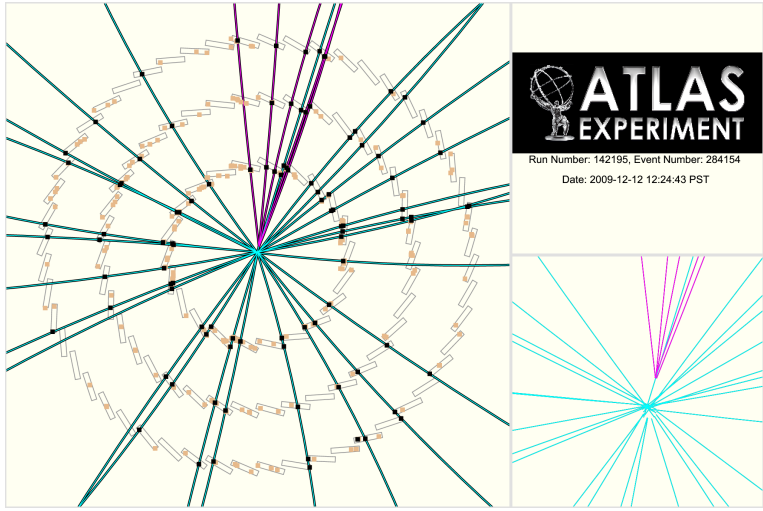
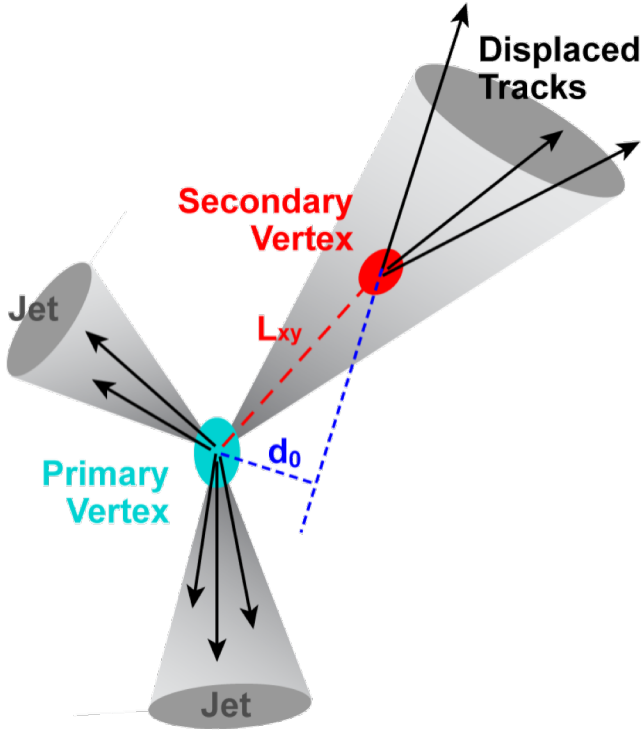


Figure 2: **Long Short-term Memory Cell**

Bottom Quark Decays

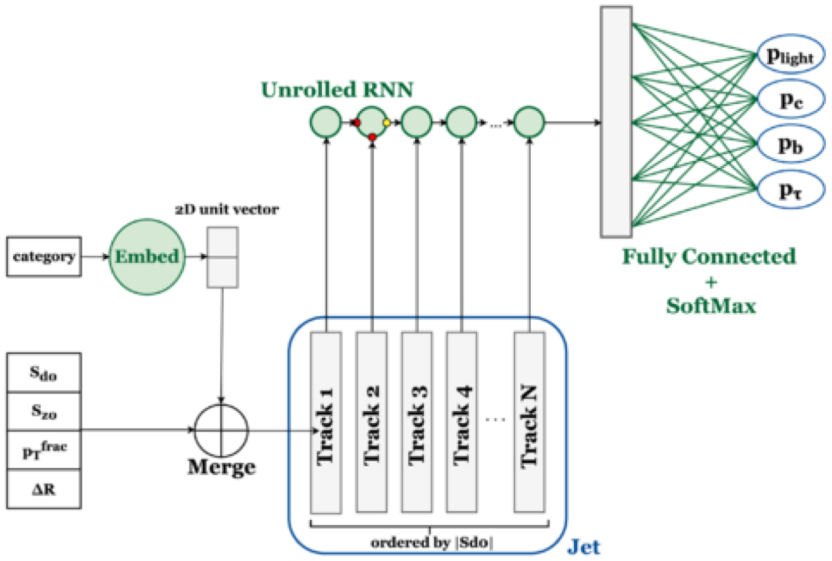


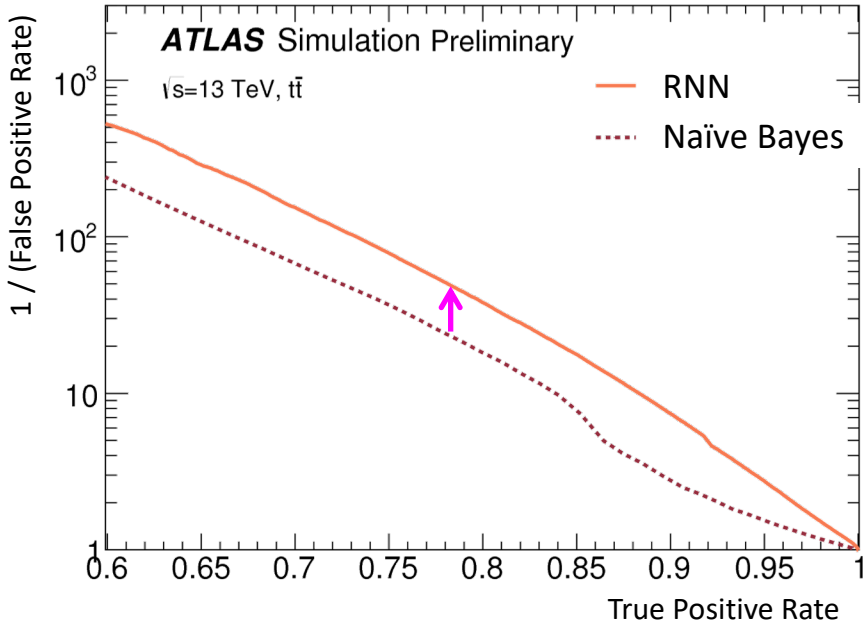


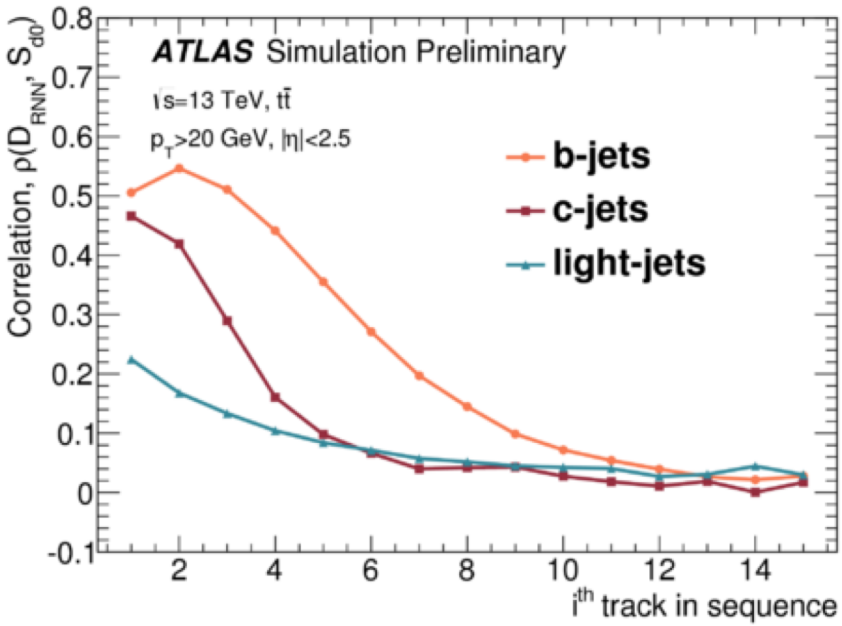


- Goal: Discriminate b-jets from non-b-jets
- **Track based** taggers: $p(jet\ flavor \mid tracks\ in\ jet)$
  - Dimensionality too high for easy density estimation
  - Often make naïve Bayes assumption that tracks independent!

RNN b-tagging







Fisher Discriminant

Fisher Discriminant

- Suppose our $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1\ldots N}$ is separated in two classes, we want a projection to maximize the separation between the two classes.

Fisher Discriminant

- Suppose our $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1\ldots N}$ is separated in two classes, we want a projection to maximize the separation between the two classes.

  – Want means ($\mathbf{m}_i$) of two classes (C_i) to be as far apart as possible → large *between-class* variation

$$\mathbf{S}_B = (\mathbf{m}_2 - \mathbf{m}_1)^T(\mathbf{m}_2 - \mathbf{m}_1)$$

Fisher Discriminant

- Suppose our $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1\ldots N}$ is separated in two classes, we want a projection to maximize the separation between the two classes.

  – Want means ($\mathbf{m}_i$) of two classes (C_i) to be as far apart as possible → large *between-class* variation

$$\mathbf{S}_B = (\mathbf{m}_2 - \mathbf{m}_1)^T(\mathbf{m}_2 - \mathbf{m}_1)$$

  – Want each class tightly clustered, as little overlap as possible → small *within-class* variation

$$\mathbf{S}_W = \sum_{i \in C_1} (\mathbf{x}_i - \mathbf{m}_1)^T(\mathbf{x}_i - \mathbf{m}_1) + \sum_{i \in C_2} (\mathbf{x}_i - \mathbf{m}_2)^T(\mathbf{x}_i - \mathbf{m}_2)$$

Fisher Discriminant

- Suppose our $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1\ldots N}$ is separated in two classes, we want a projection to maximize the separation between the two classes.

  – Want means ($\mathbf{m}_i$) of two classes (C_i) to be as far apart as possible → large *between-class* variation

$$\mathbf{S}_B = (\mathbf{m}_2 - \mathbf{m}_1)^T(\mathbf{m}_2 - \mathbf{m}_1)$$

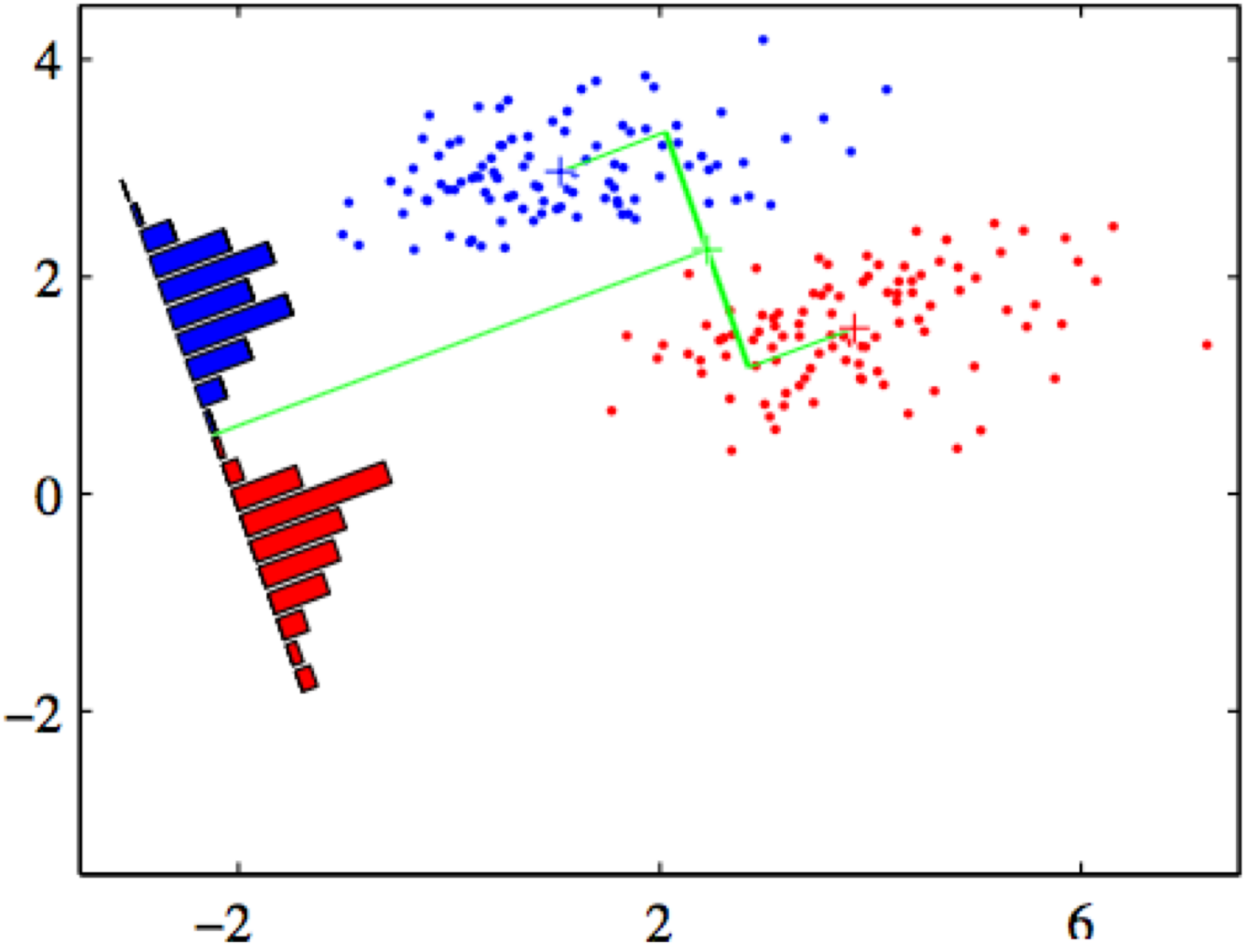
  – Want each class tightly clustered, as little overlap as possible → small *within-class* variation

$$\mathbf{S}_W = \sum_{i \in C_1} (\mathbf{x}_i - \mathbf{m}_1)^T(\mathbf{x}_i - \mathbf{m}_1) + \sum_{i \in C_2} (\mathbf{x}_i - \mathbf{m}_2)^T(\mathbf{x}_i - \mathbf{m}_2)$$

- Maximize Fisher criteria

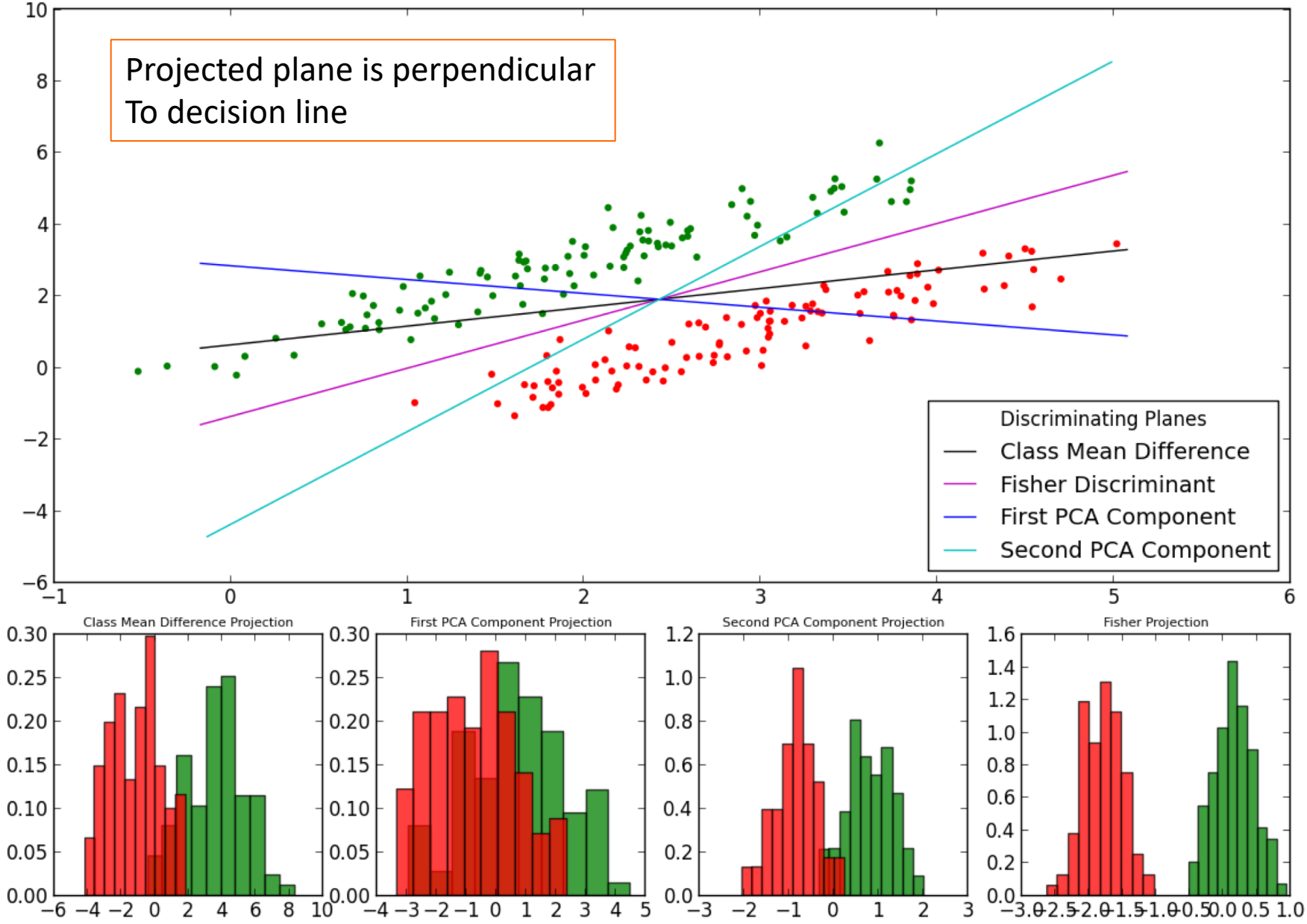
$$J(\mathbf{w}) = \frac{\mathbf{w}^T\mathbf{S}_B\mathbf{w}}{\mathbf{w}^T\mathbf{S}_W\mathbf{w}} \quad \rightarrow \quad \boxed{\mathbf{w} \propto \mathbf{S}_W(\mathbf{m}_2 - \mathbf{m}_1)}$$

Fisher Discriminant



[Bishop]

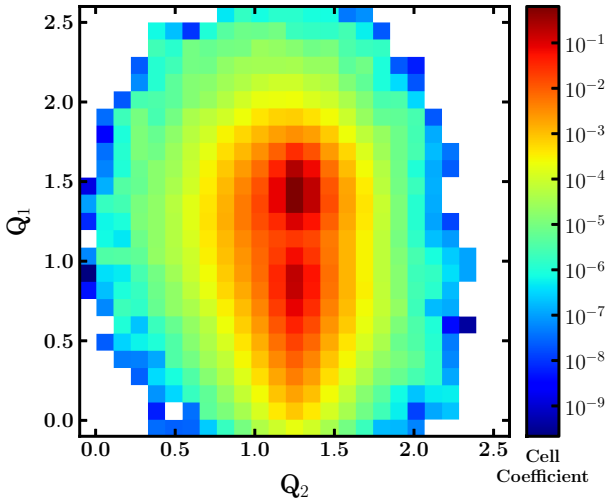
Comparing Techniques



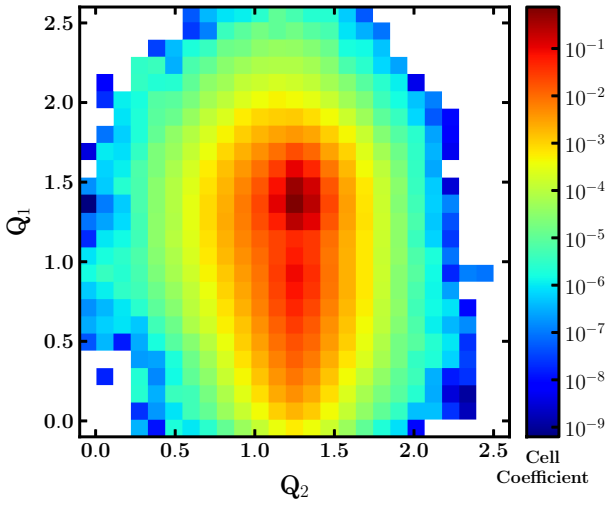
Fisher Discriminant

http://arxiv.org/abs/1407.5675

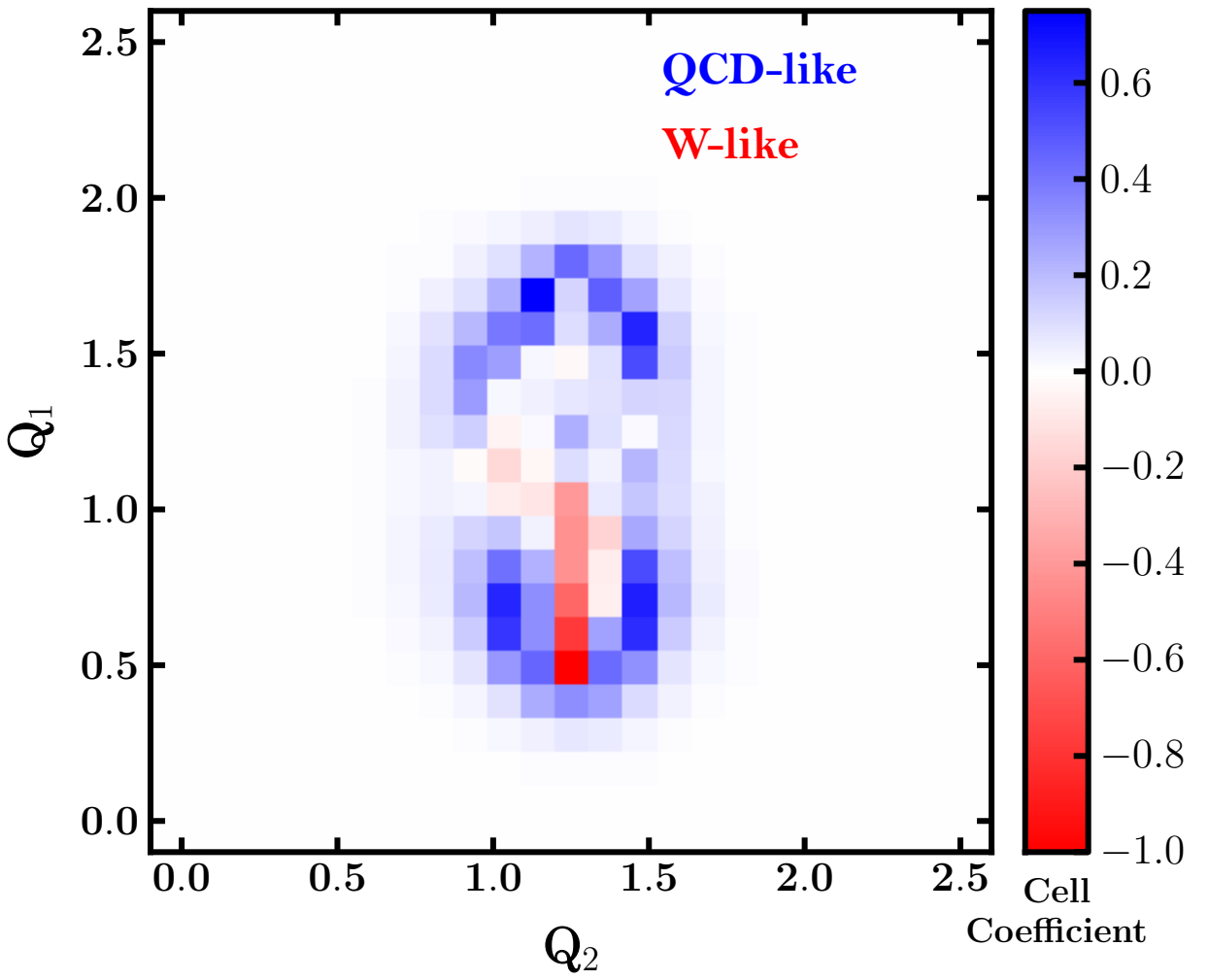
Average Boosted W jet



Average quark / gluon jet



Plotted weights of Fisher Discriminant



Hierarchical Agglomerative Clustering

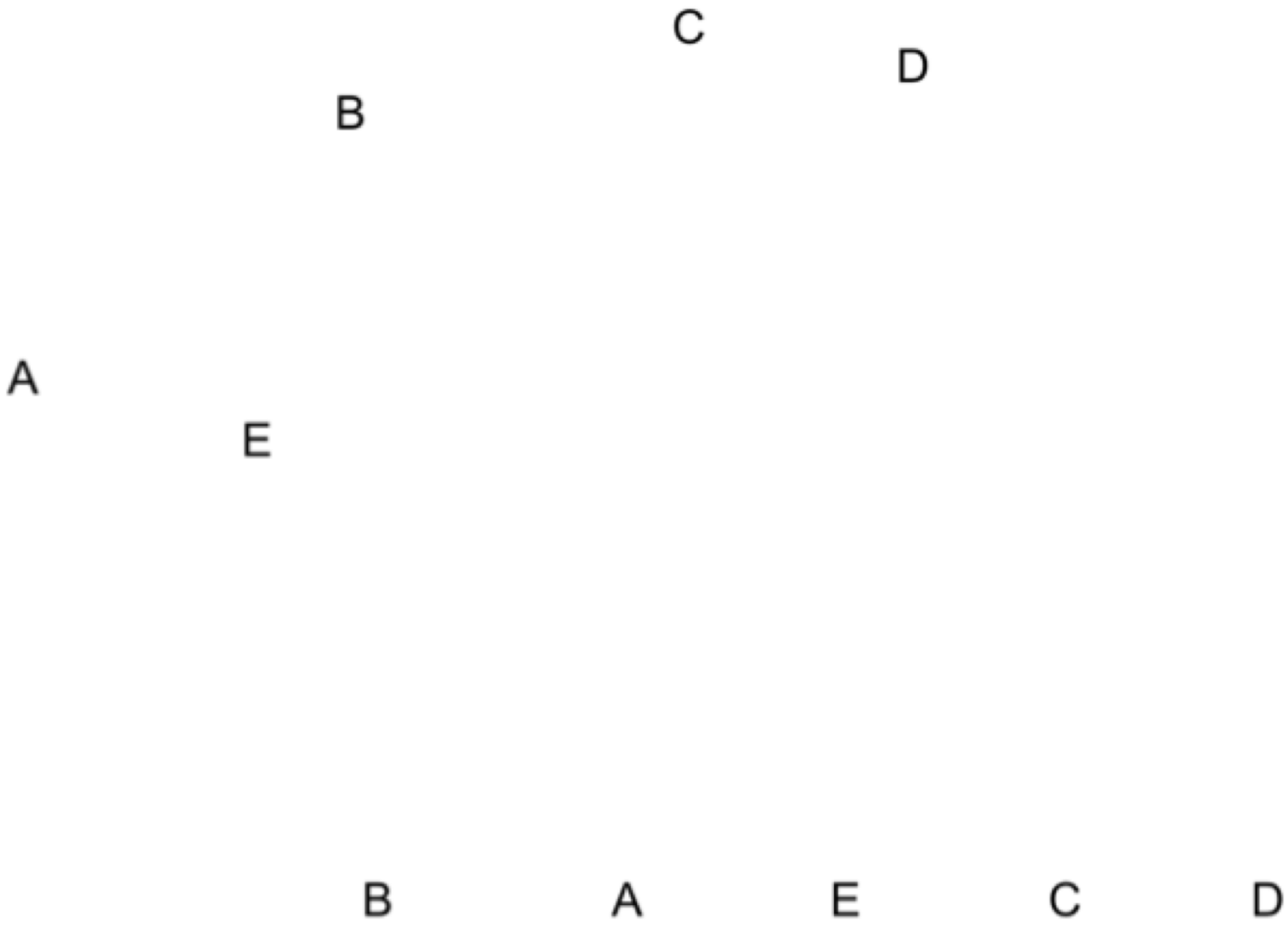
Hierarchical Agglomerative Clustering

- **Algorithm**
  - Start with each example $\mathbf{x}_i$ as its own cluster
  - Take pairwise distance between examples
  - Merge closest pair into a new cluster
  - Repeat until one cluster

- Doesn't require choice of number of clusters
- Clusters can have arbitrary shape
- Clusters have intrinsic hierarchy
- No random initialization

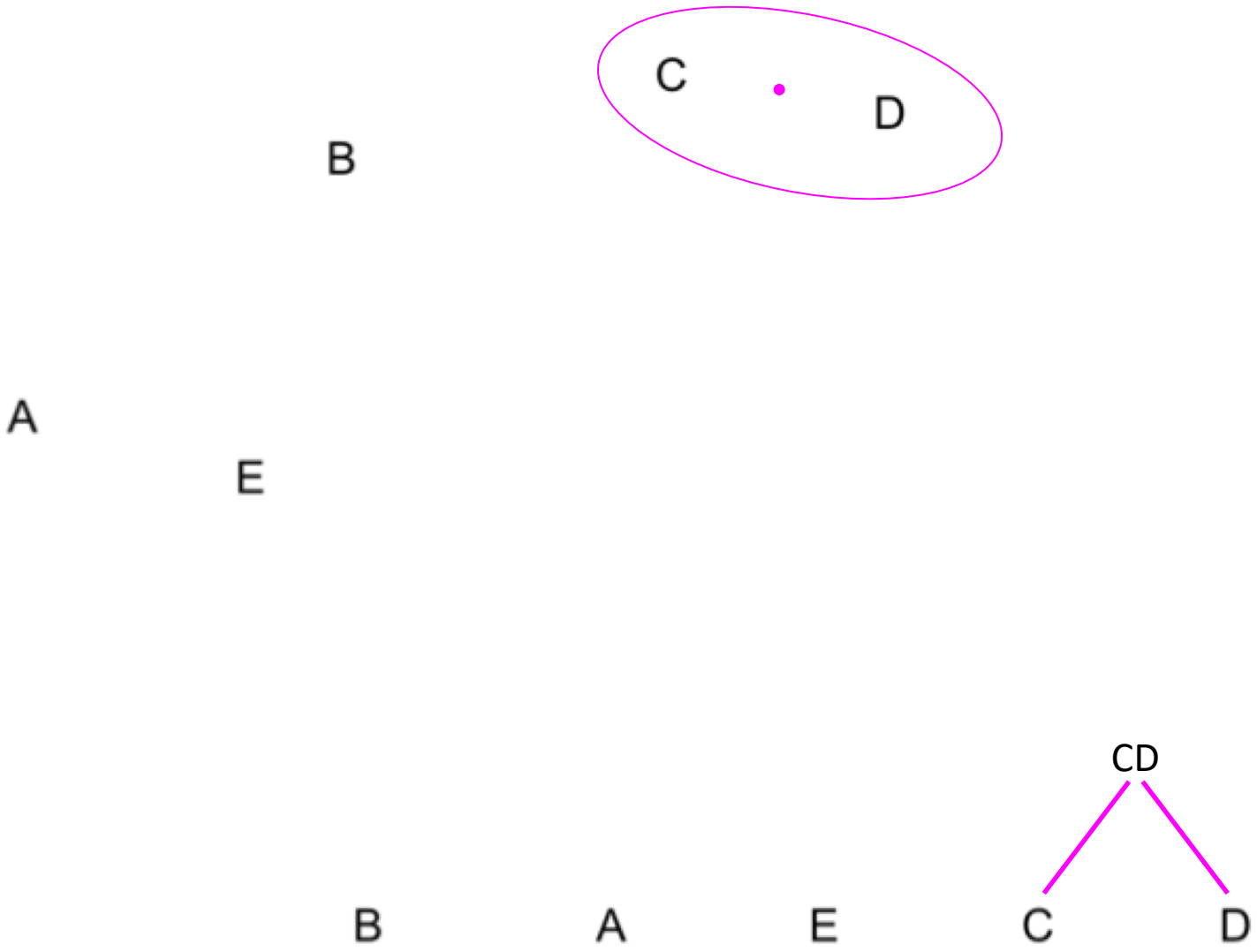
- What distance metric to use?
  - Here use Euclidean distance between cluster centroid (average of examples in cluster)

Hierarchical Agglomerative Clustering



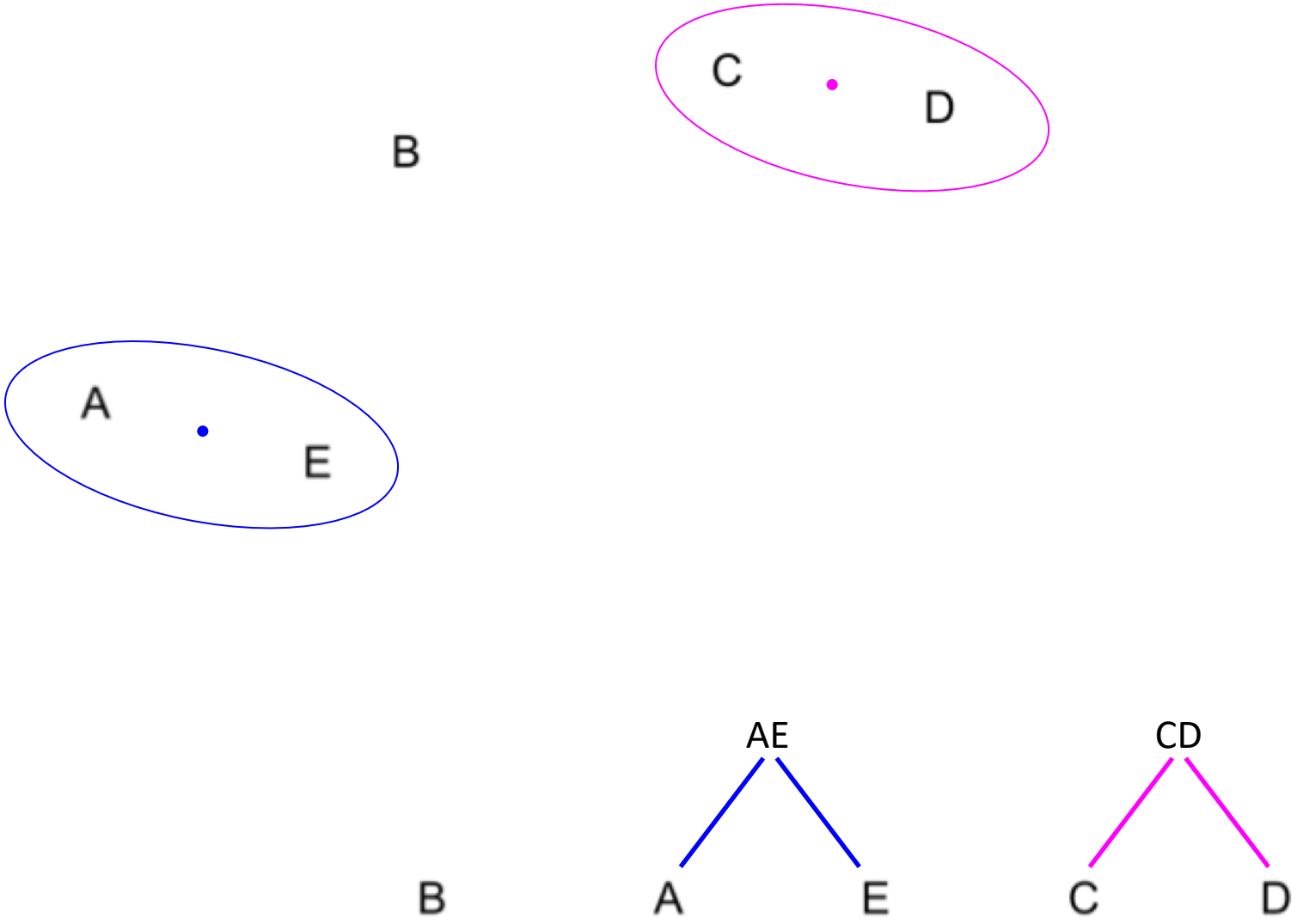
[Parkes]

Hierarchical Agglomerative Clustering



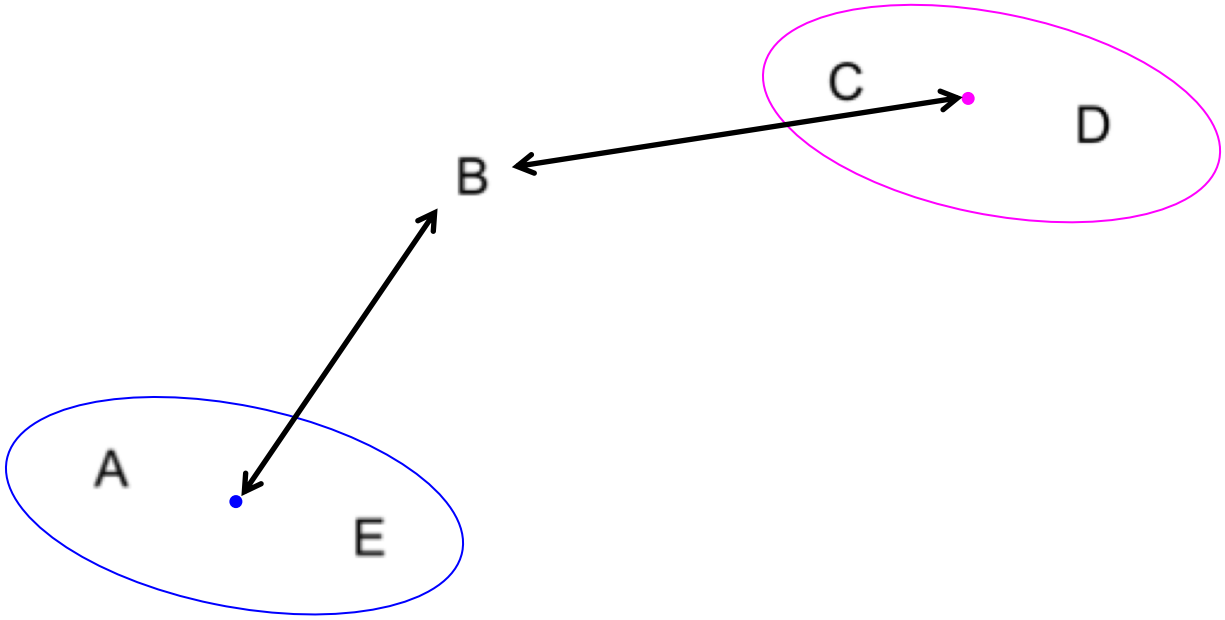
[Parkes]

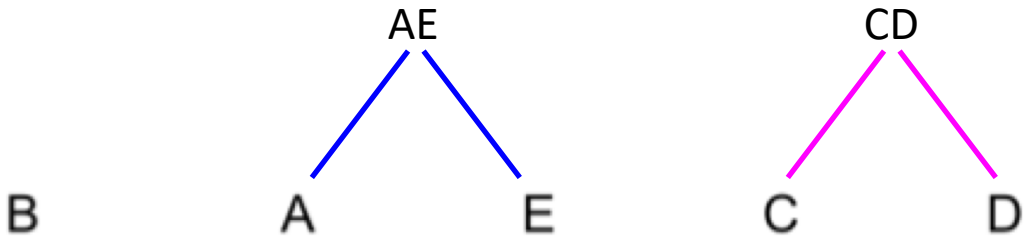
Hierarchical Agglomerative Clustering



[Parkes]

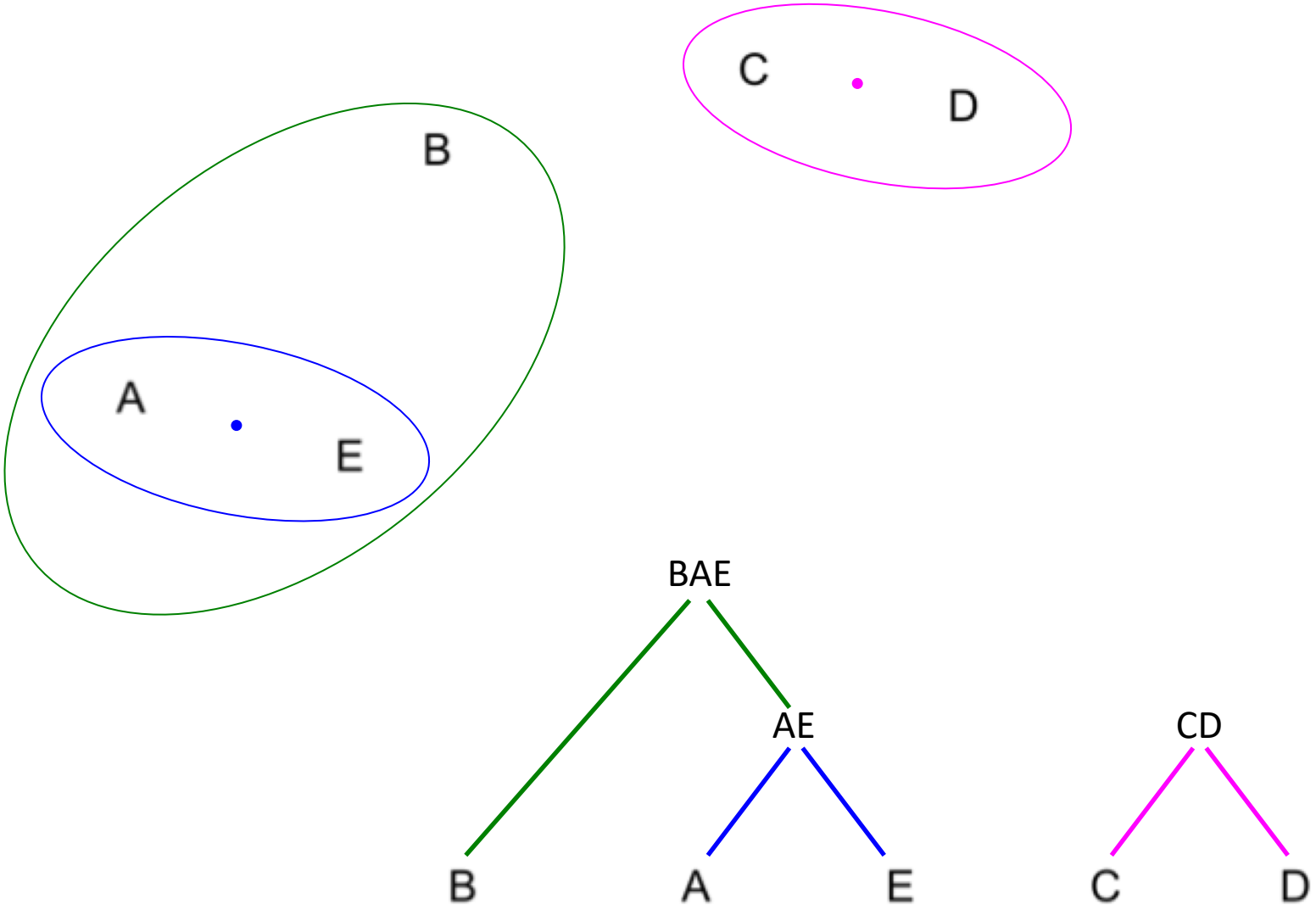
Hierarchical Agglomerative Clustering





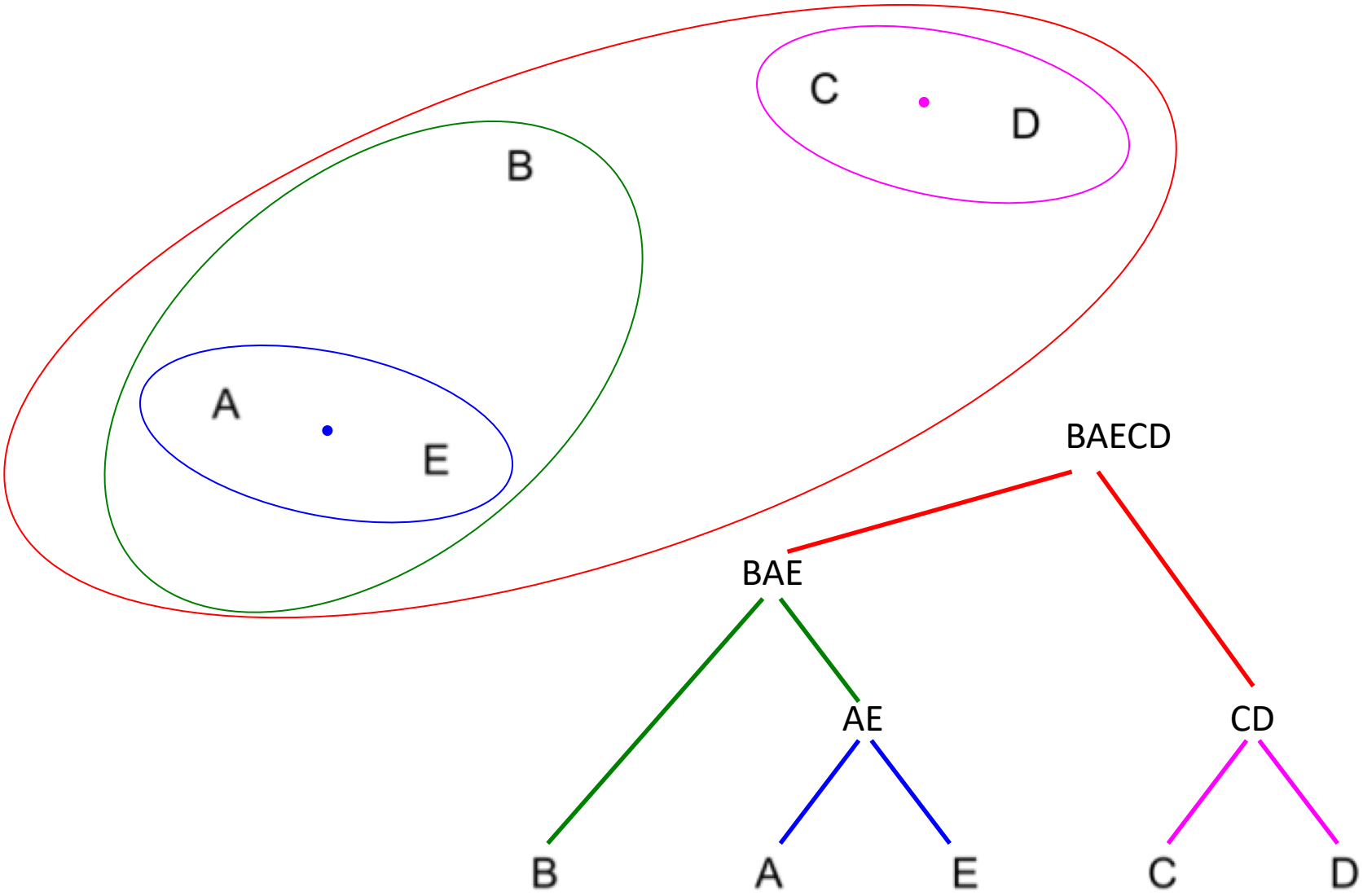
[Parkes]

Hierarchical Agglomerative Clustering



[Parkes]

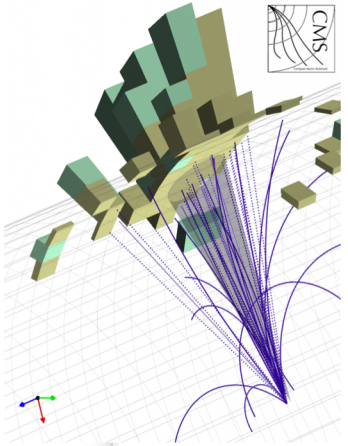
Hierarchical Agglomerative Clustering



[Parkes]

Jet Algorithms

- Sequential pairwise jet clustering algorithms are hierarchical clustering, and are a form of unsupervised learning



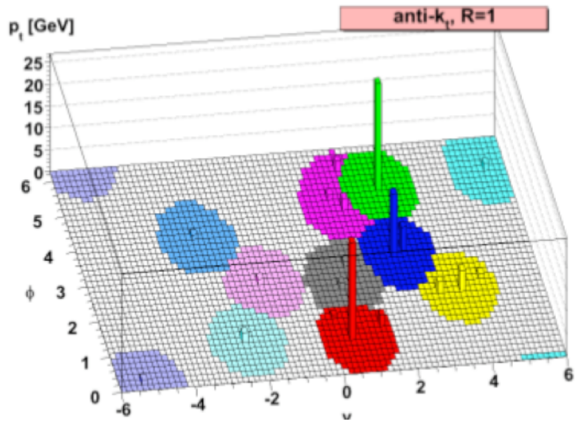
- Compute distance between pseudojets i and j

$$d_{ij} = \min\left(k_{\mathrm{T}i}^{2p}, k_{\mathrm{T}j}^{2p}\right)\frac{\Delta_{ij}}{D} \qquad \Delta_{ij}^2 = (y_i - y_j)^2 + (\phi_i - \phi_j)^2$$

- Distance between pseudojet and beam

$$d_{iB} = k_{\mathrm{T}i}^{2p}$$

- Find smallest distance between pseudojets d_{ij} or d_{iB}
  - Combine (sum 4-momentum) of two pseudojets if d_{ij} smallest
  - If d_{iB} is smallest, remove pseudojet i, call it a jet
  - Repeat until all pseudojets are jets



Debugging Learning Algorithms

More general advice

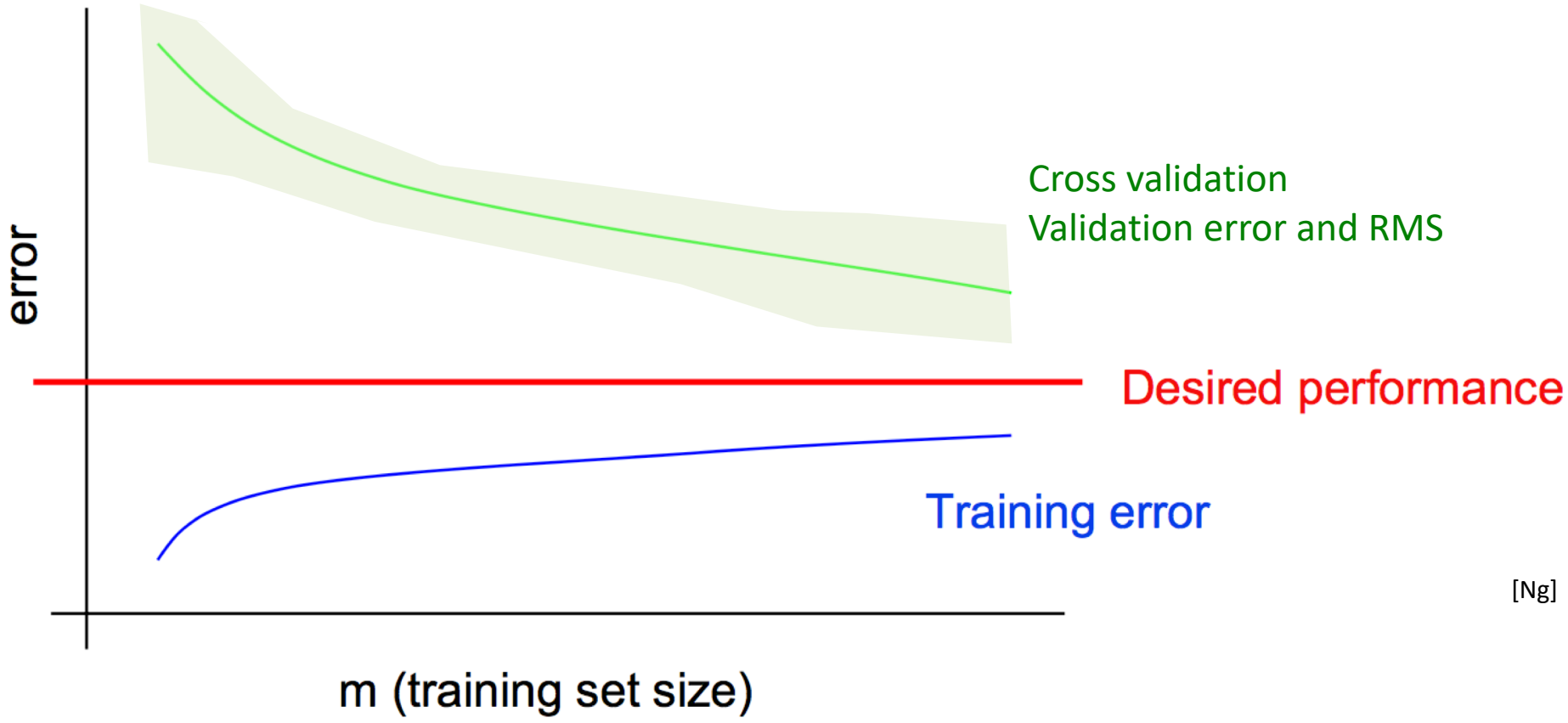
- You will likely need to try many algorithms…
  - Start with something simple!
  - Use more complex algorithms as needed
  - Use cross validation to check for overcomplexity / overtraining
- Check the literature
  - If you can cast your (HEP) problem as something in the ML / data science domain, there may be guidance on how to proceed
- Hyperparameters can be hard to tune
  - Use cross validation to compare models with different hyperparameter values!
- Use a training / validation / testing split of your data
  - Don't use training or validation set to determine final performance
  - And use cross validation as well!

Debugging Learning Algorithms

- Is my model working properly?
  - Where do I stand with respect to bias and variance?
  - Has my training converged?
  - Did I choose the right model / objective?
  - Where is the error in my algorithm coming from?

Section derived from [Ng]

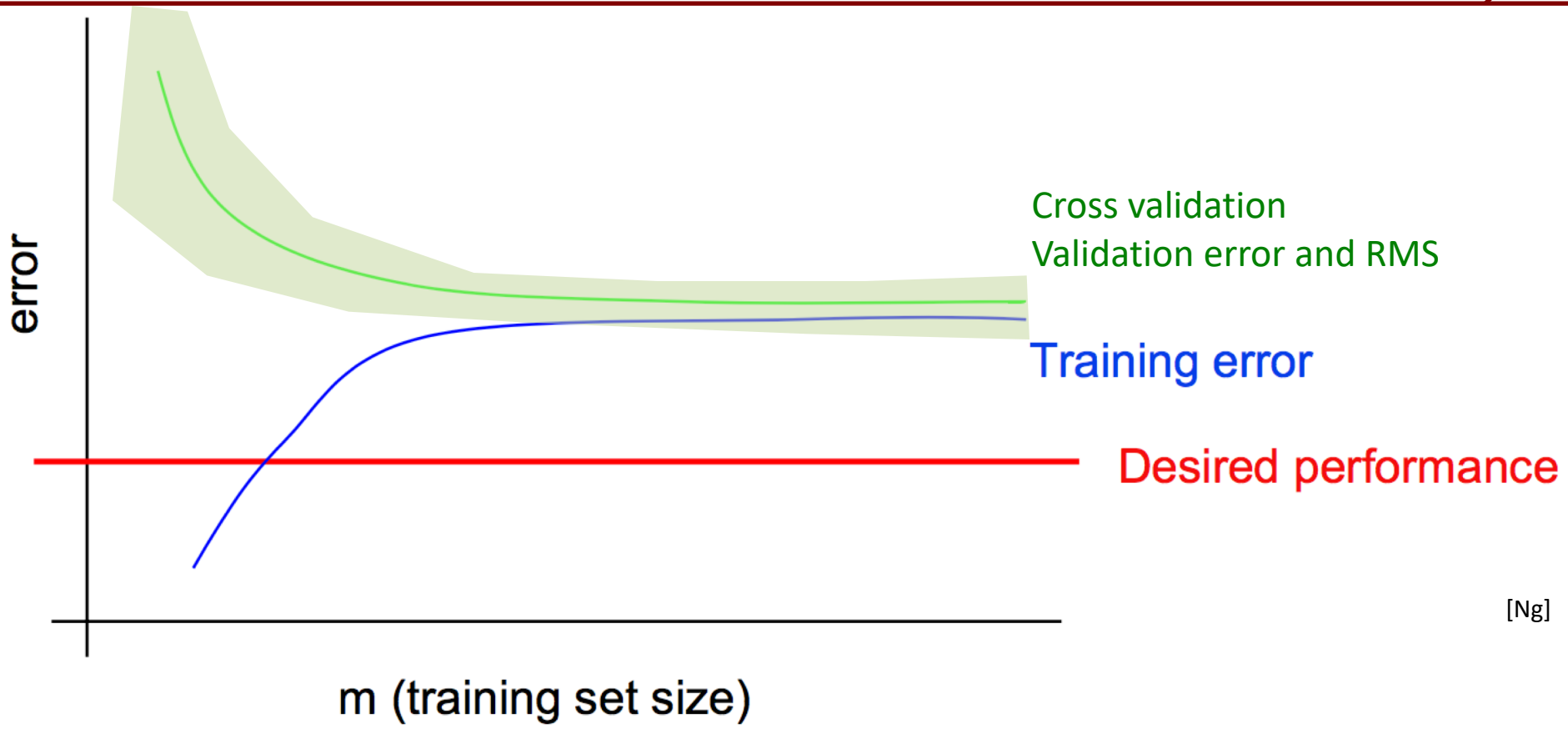
Typical learning curve for high variance



[Ng]

- Performance is not reaching desired level
- Error still decreasing with training set size
  - suggests to use more data in training
- Large gap between training and validtaion error
  - Some gap is expected (inherint bias towards training set)
- Better: Large Cross-validation RMS, large performance variation in trainings

Typical learning curve for high bias



[Ng]

- Training error is unacceptably high
- Small gap between training and validation error
- Cross validation RMS is small

Potential Fixes

- Fixes to try:
  - Get more training data — Fixes high variance
  - Try smaller feature set size — Fixes high variance
  - Try larger feature set size — Fixes high bias
  - Try different features — Fixes high bias

- Did the training converge?
  - Run gradient descent a few more iterations — Fixes optimization algorithm
    - or adjust learning rate
  - Try different optimization algorithm — Fixes optimization algorithm

- Is it the correct model / objective for the problem?
  - Try different regularization parameter value — Fixes optimization objective
  - Try different model — Fixes optimization objective

- You will often need to come up with your own diagnostics to understand what is happening to your algorithm

[Ng]