# Machine Learning for Optical Communication

Team 6: Alex Wang, Matthew Luzenski

# Credits

We divided the work evenly for this project

All algorithms were researched together

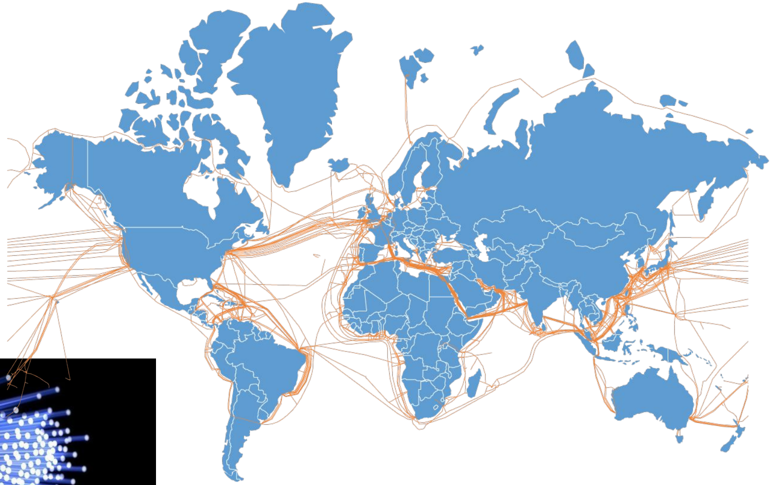
- Maximum Likelihood Estimator implemented by Matthew
  - Coded in Matlab
- Neural Network implemented by Alex
  - Coded in Python using TensorFlow for hardware acceleration
- Support Vector Machines were designed, tested, and analyzed together.
  - Coded in Matlab
  - Basic structure - coded together
  - Soft-margin regularizer and desynchronization - Alex
  - RBF kernel and channel noise - Matthew

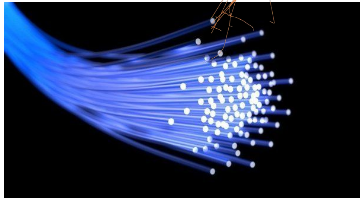
# Outline

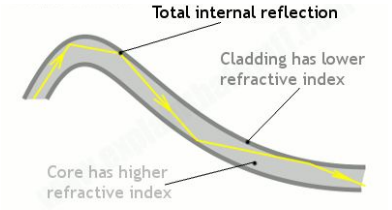
- Background
  - Dispersive optical fibers
  - Feature extraction: Binary (NRZ) and 4-bit (4-PAM)
- Maximum Likelihood Estimator
  - Approximating probability distributions
- Convolutional Neural Network
  - Hidden layer architecture
  - Learned features
- Support Vector Machine
  - Hyperplane hinge loss
  - Hard margin vs. soft margin
  - RBF kernel

# Background

# Problem Statement



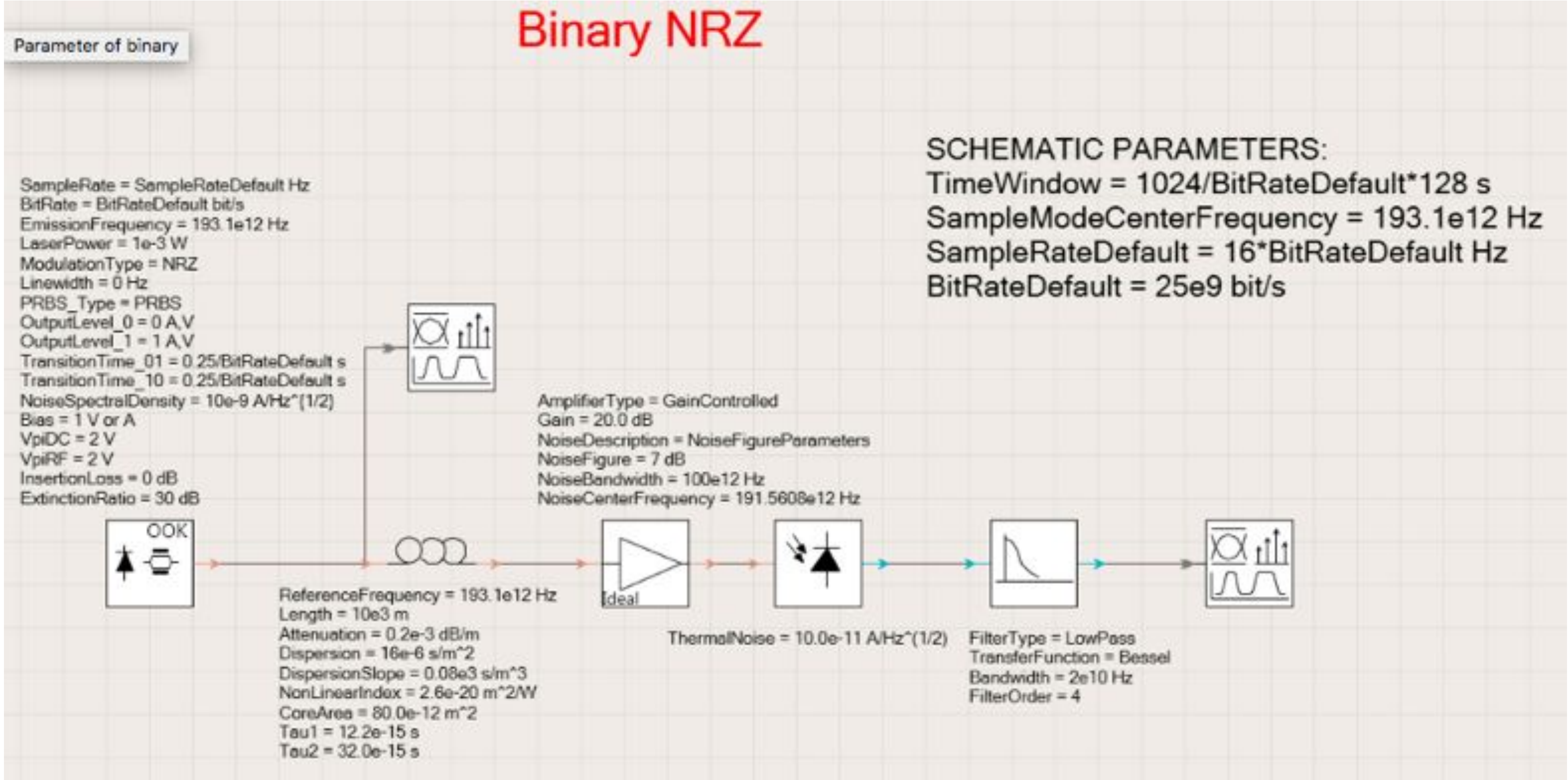




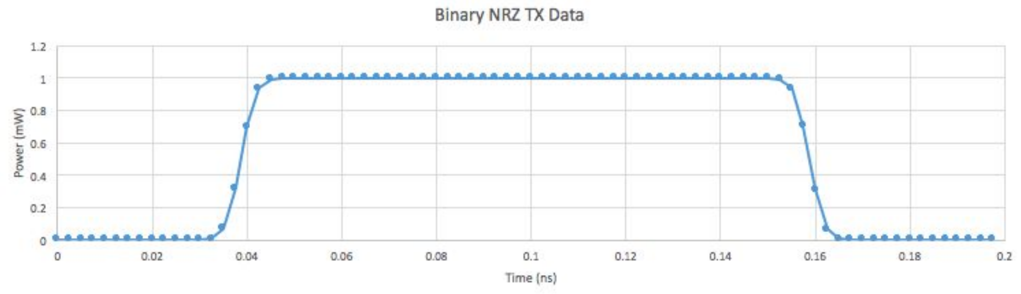

- Fiber-optic communications becoming more prevalent
  - Light confined by total internal reflection
  - High speed, high bandwidth, low loss
  - Issue: signal distortion
- Waveguide dispersion
  - Redistribution of light due to waveguide geometry
- Material dispersion
  - Refractive index and absorption varies with wavelength

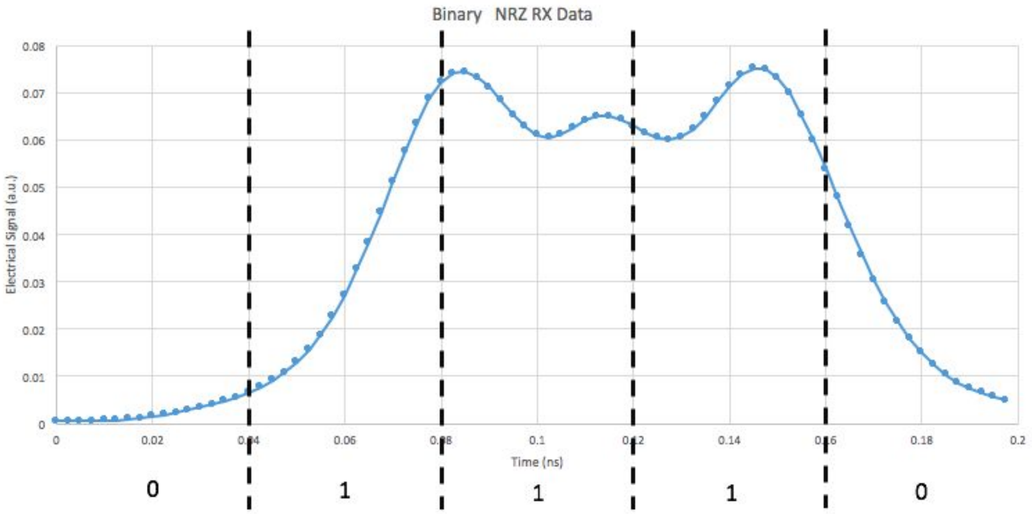
from images.google.com

# Model Simulation - VPIphotonics



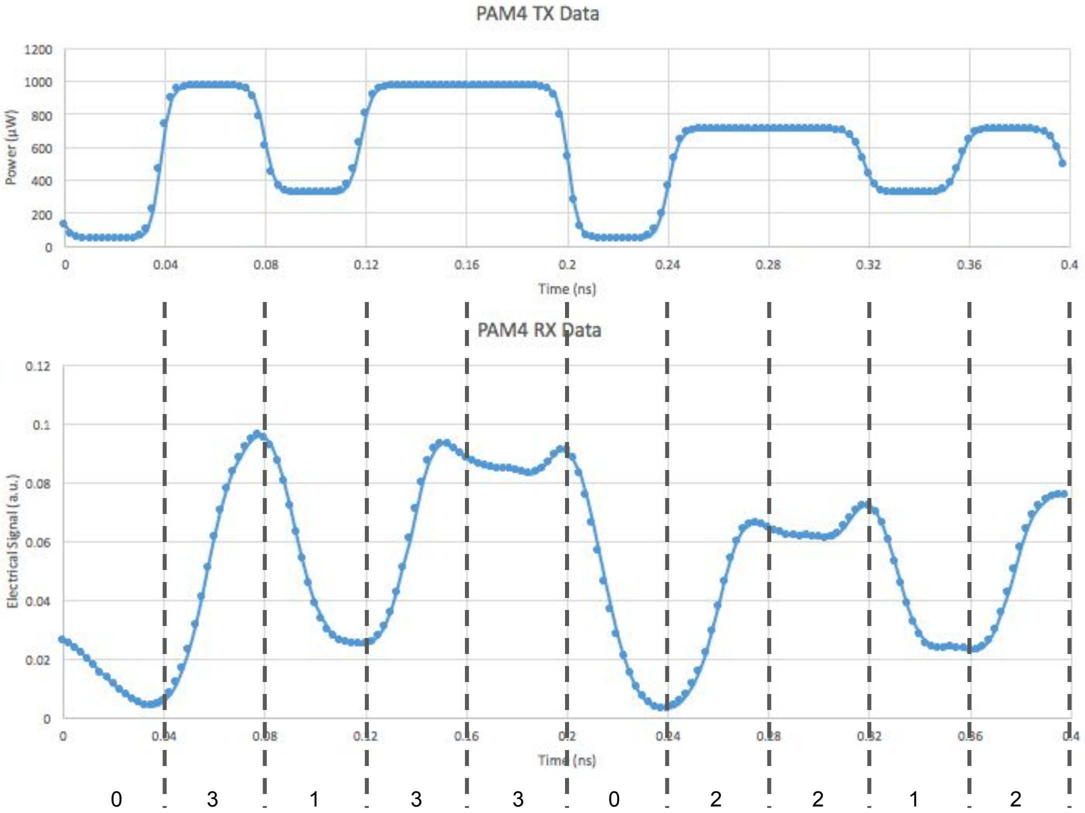
# Feature Extraction - Binary (NRZ)




- Input light power is modulated
  - Two levels: 0 and 1mW
- 16 samples per symbol
- Pre-processing:
  - Threshold TX data at 0.5 for labels
  - The majority label within each clock cycle becomes the representative label for all 16 samples

# Feature Extraction - 4-PAM




- Four power levels
  - ~40μW, ~350μW, ~750μW, ~970μW
- 16 samples per symbol
- Pre-processing:
  - Threshold TX data around each power level
  - Grey coding:
    - 0 -> 00
    - 1 -> 01
    - 2 -> 11
    - 3 -> 10

# Maximum Likelihood Estimator

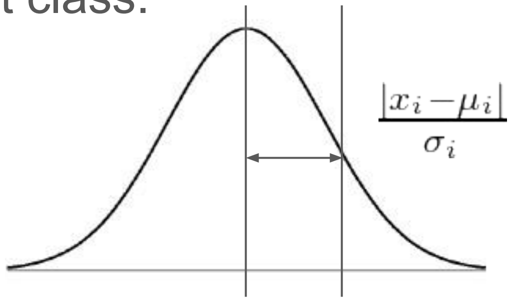
# Approximating Probability Distributions

Bayes' theorem: $\mathrm{P}(\theta \mid x_1, x_2, \ldots, x_n) = \frac{f(x_1, x_2, \ldots, x_n \mid \theta)\,\mathrm{P}(\theta)}{\mathrm{P}(x_1, x_2, \ldots, x_n)}$ $n = 16$

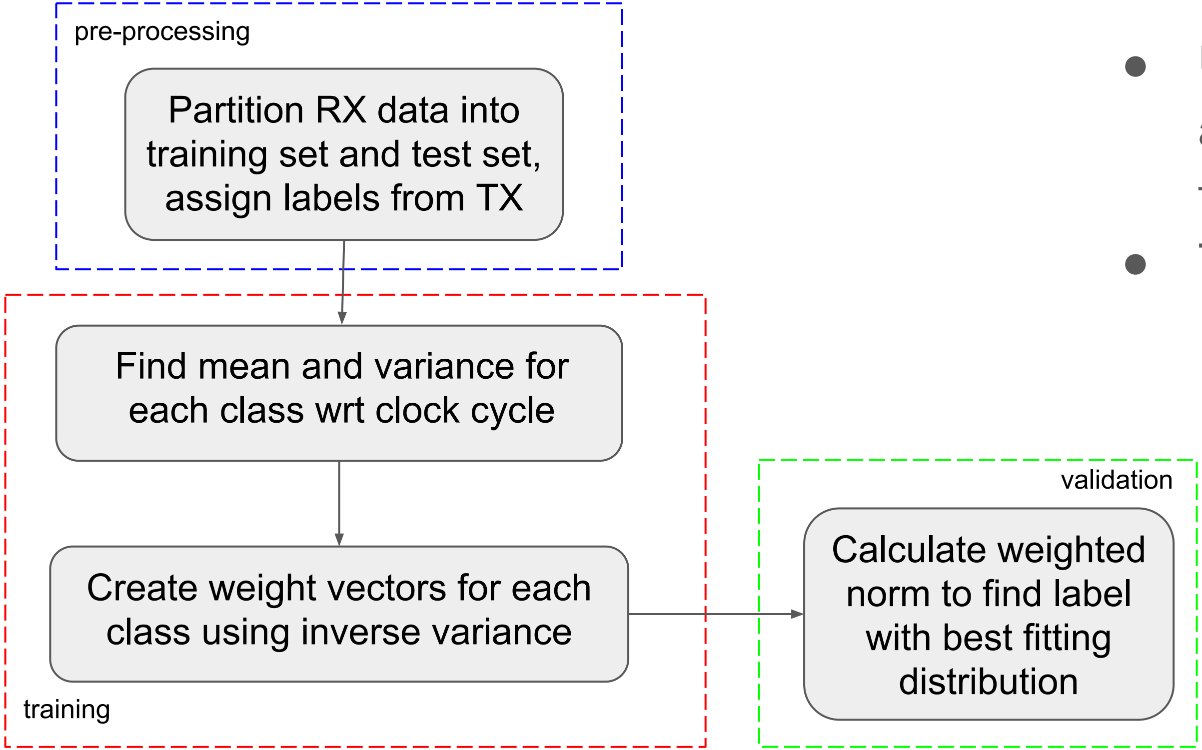
where $x_i$ are samples in a set of data, and $\theta$ is the classification for the set. We wish to make the maximum a posteriori estimate, assuming that each class is equally likely.

We assume each time sample within a clock cycle has a Gaussian distribution. An input sample is scaled by the variance to find the deviation from the mean of each class, and the prediction is given by the nearest class.

$$x_i \sim N(\mu_i, \sigma_i^2)$$

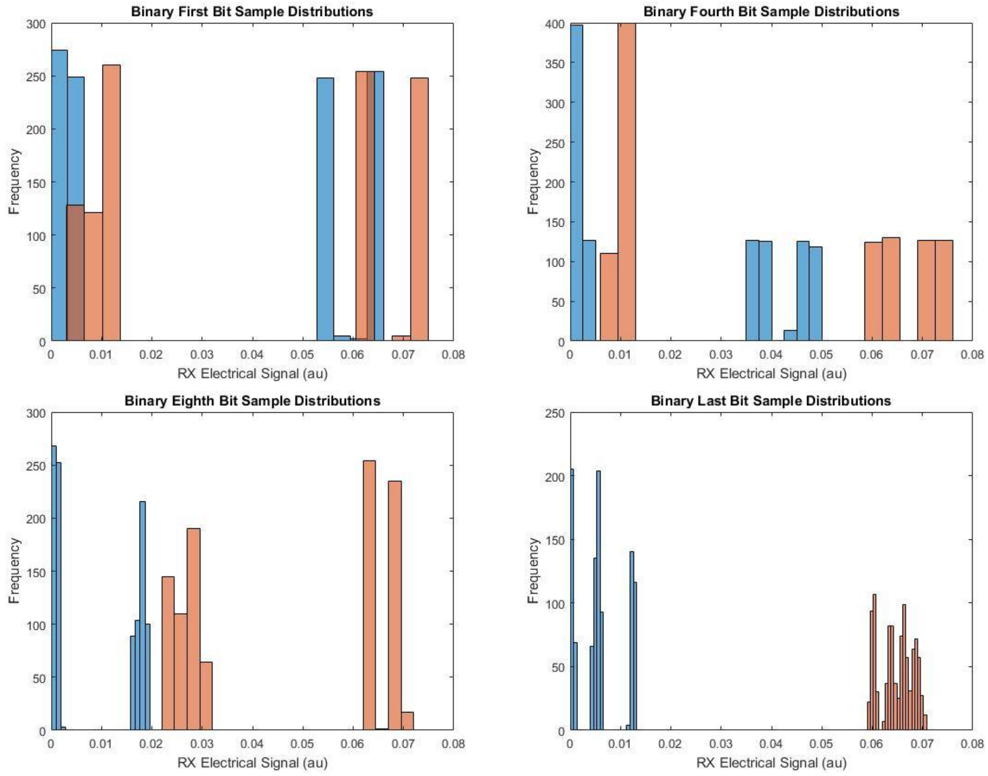


# Program Flow - Maximum Likelihood



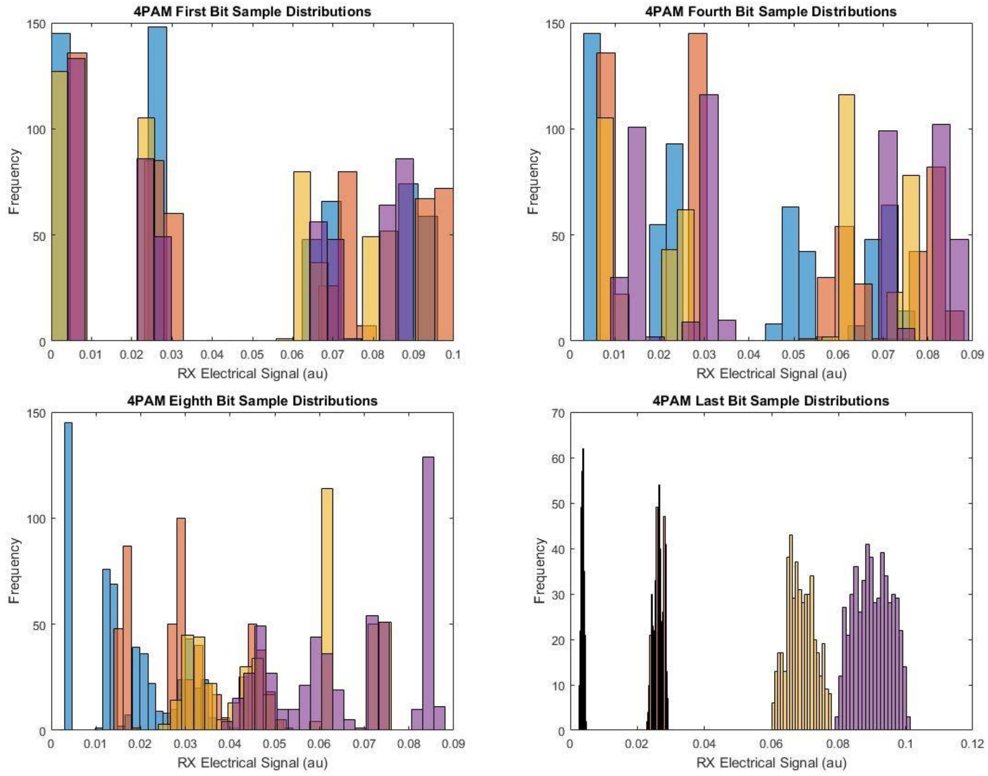

- Using the training set, approximate a distribution for each of the 16 samples
- To label the test data:
  - Calculate the weighted norm from the test vector to each class's mean vector
  - Use inverse variances as weights
  - The label is decided by the lowest norm

# Histograms (Binary)




- Blue: label 0
- Orange: label 1
- The distributions of the signal values for the two classes are overlapping early in the clock cycle
- The last sample in the clock cycle is the most distinguishable

# Histograms (4-PAM)



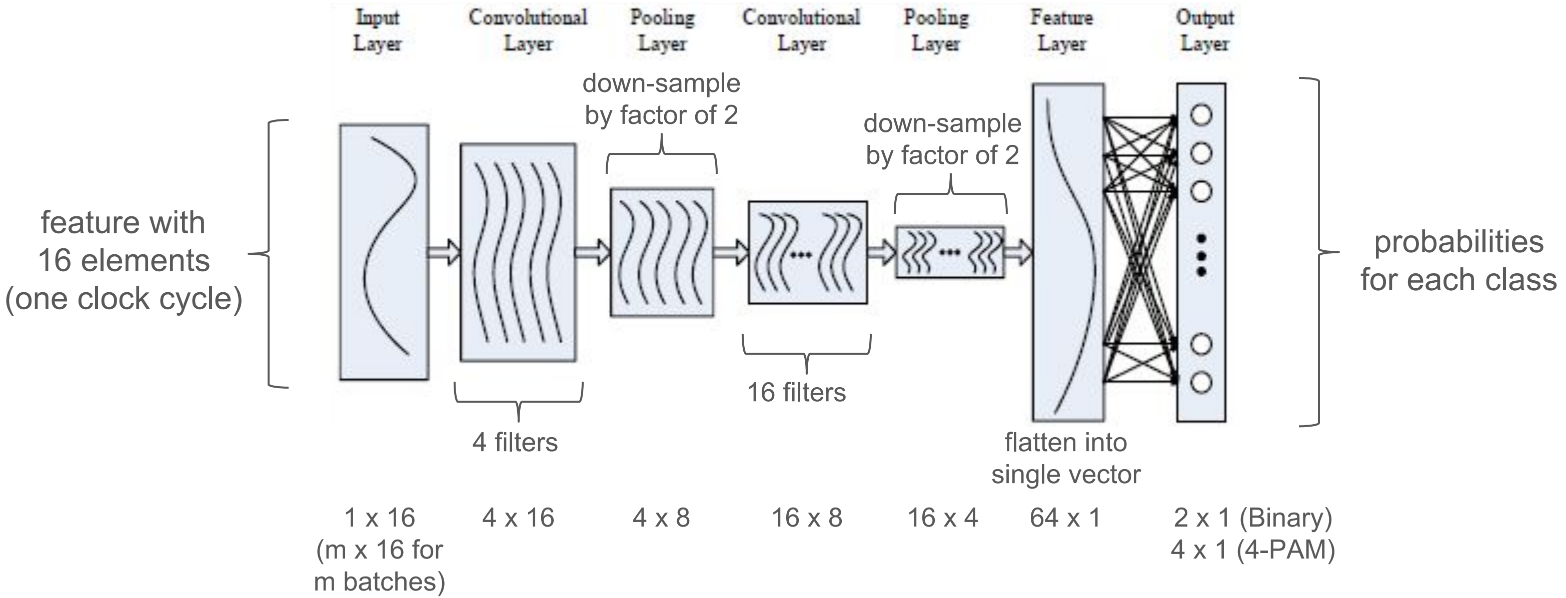

- Blue: label 0
- Red: label 1
- Yellow: label 2
- Purple: label 3
- Similar to the binary case, the last sample in the clock cycle is the most distinct

Results: 0 misclassifications for both Binary and 4-PAM

# Convolutional Neural Network

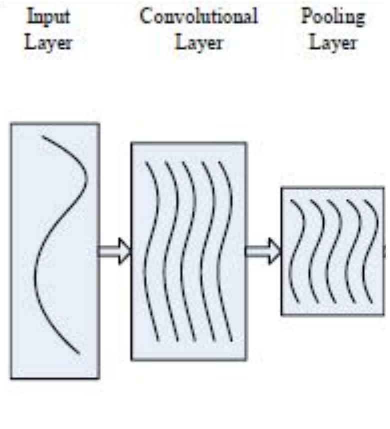
# Neural Network Architecture

Typically used for classifying 2D images, we can implement a convolutional neural net for 1D waveforms.

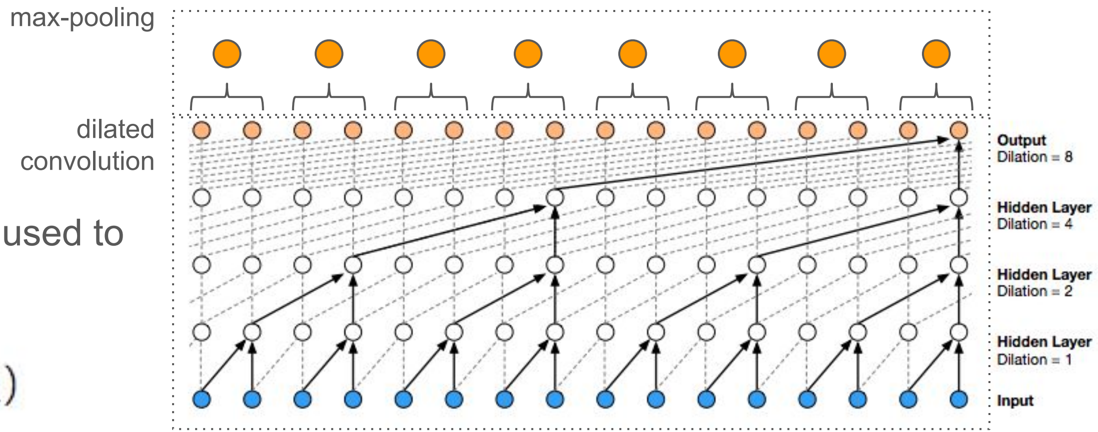



B. Zhao, et al., **Waveforms Classification based on Convolutional Neural Networks.** *IEEE (2017); DOI:10.1109.*

# Convolutions and Pooling




- Max-pooling: the largest node in each window is selected for the next layer
- The outputs can be considered as learned feature embeddings



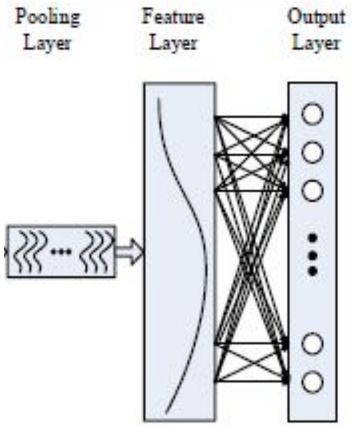

- Dilated causal convolutions are used to respect the ordering of the data

$$p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t \mid x_1, \ldots, x_{t-1})$$

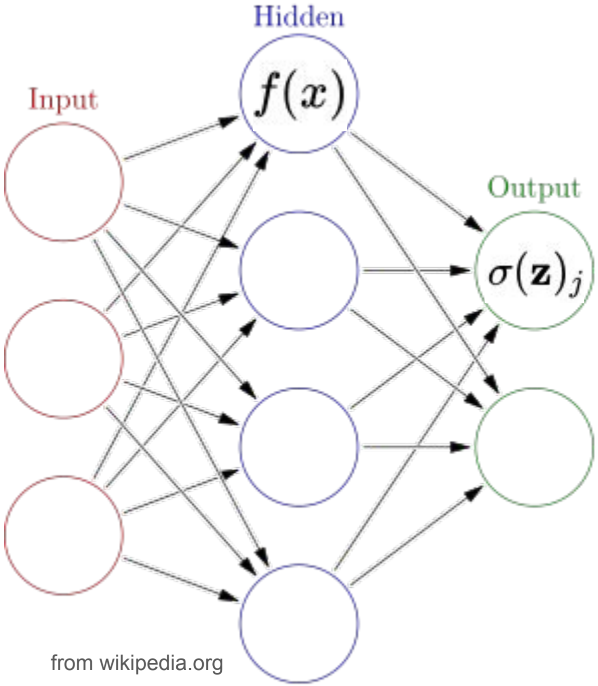
- High receptive field, low memory cost

A. van den Oord, et al., **WaveNet: A Generative Model for Raw Audio.** *Google DeepMind (2016); arXiv:1609.03499v2.*

# Dense Layers




- Each connection multiplies the input by a weight
- A node is calculated by summing over all incoming connections




from wikipedia.org

- An additional relu function is applied to the hidden layer

$$f(x) = x^+ = \max(0, x)$$

- Softmax is used to normalize the output layer

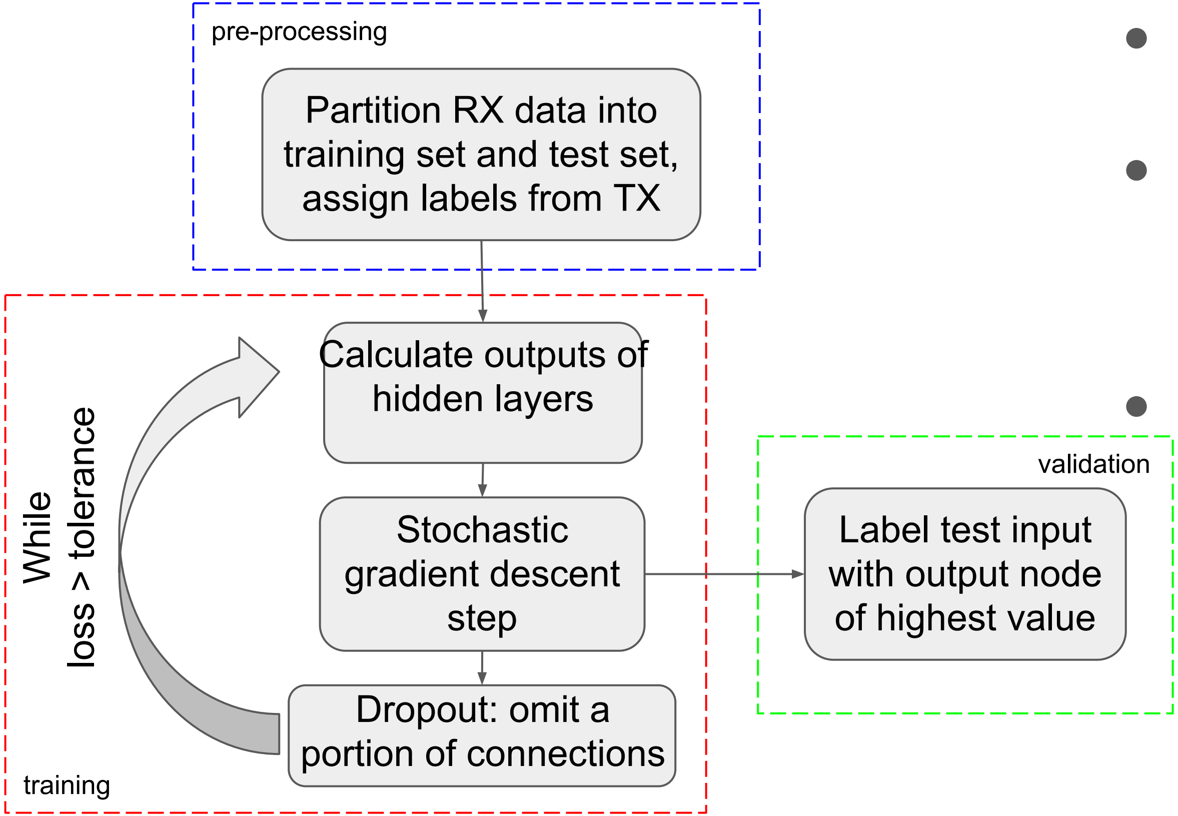
$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

- Cross-entropy is used as a loss function

$$H(p, q) = \sum_{x_i} p(x_i) \log \frac{1}{q(x_i)}$$

where p and q are two probability distributions

# Program Flow - CNN

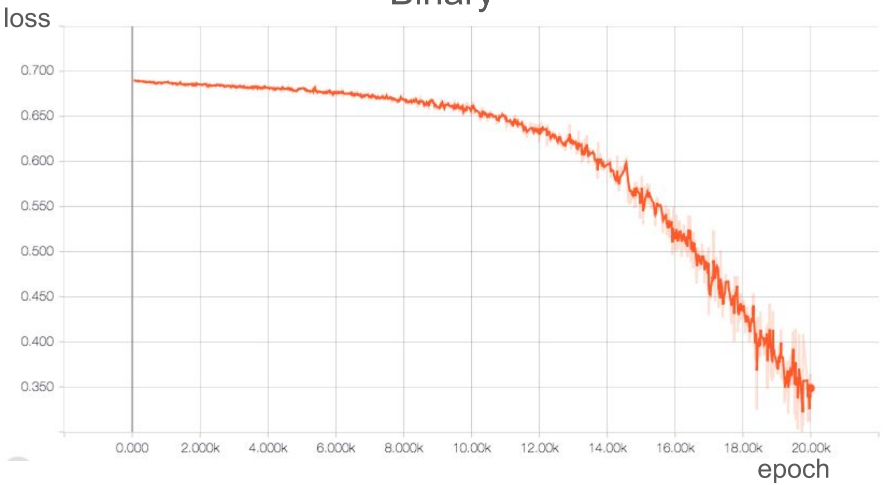



- Inputs are fed to the neural net in batches
- Training is done via gradient descent
  - Gradients automatically calculated by TensorFlow
- Values being optimized:
  - Weights of the dilated convolutional filter
  - Weights of the dense layer connections

# Classifier Performance

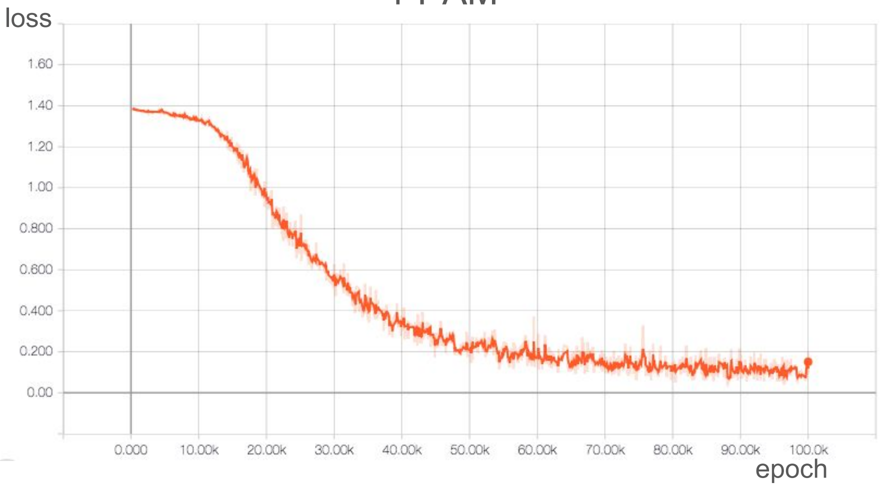
Loss vs. Training Epochs

Binary



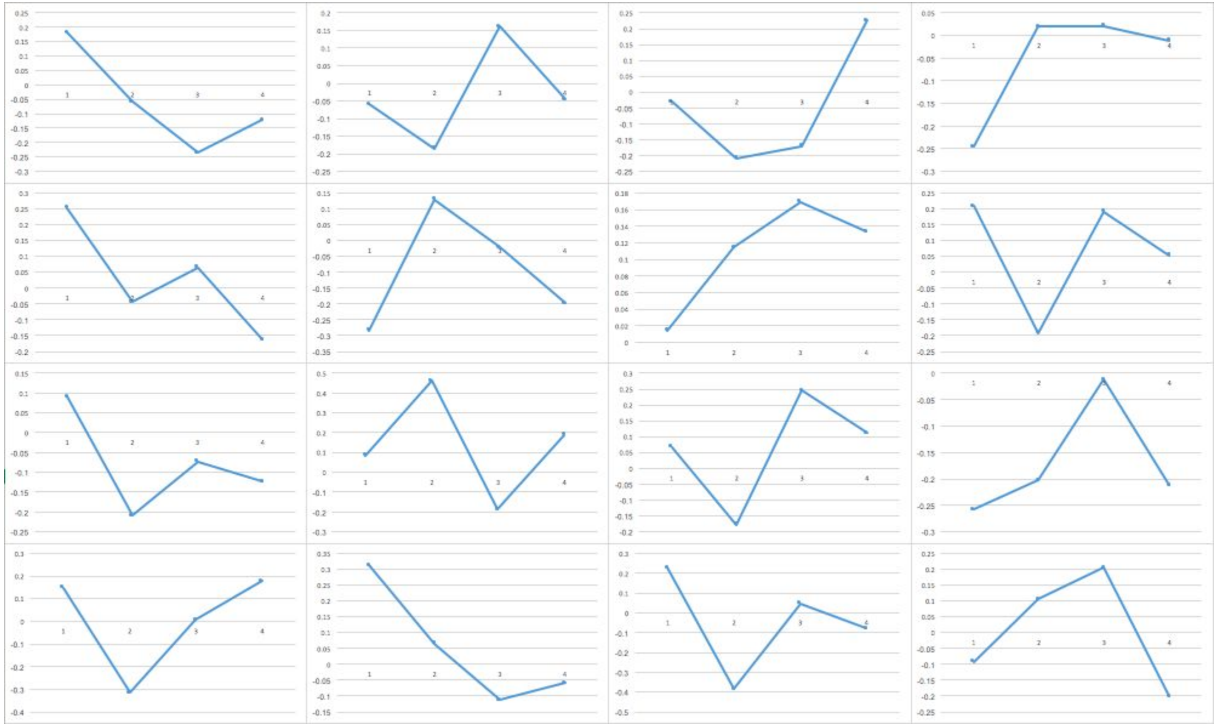

trains in 20,000 epochs
~1 minute
0 misclassifications
loss = 0.35

4-PAM




trains in 100,000 epochs
~10 minutes
0 misclassifications
loss = 0.10

# Learned Features



- After training, the kernels from the 2nd dilated convolutional layer are displayed here
- These are obtained by plotting the outputs of the 2nd pooling layer using an input delta function
- Each graph can be interpreted as a low-resolution waveform shape that the neural network learns to look for

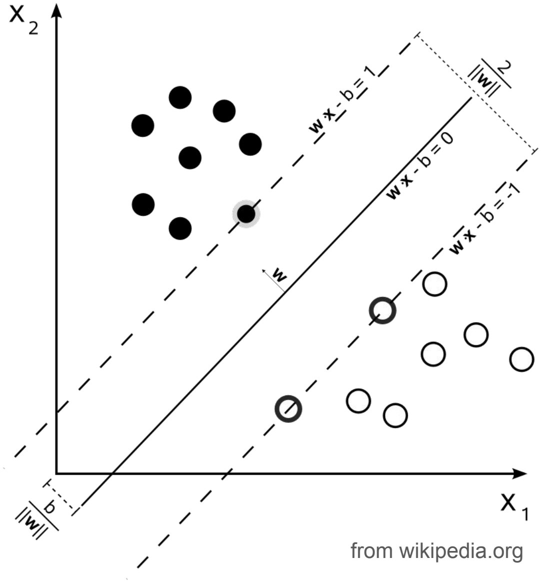
# Support Vector Machine

# Hyperplane Hinge Loss

We wish to find the maximum-margin hyperplane: $\vec{w} \cdot \vec{x} - b = 0$ $\quad w, x_i \in \mathcal{R}^{16}$

For the binary case, we have two classes.

$\vec{w} \cdot \vec{x}_i - b \geq 1$, if $y_i = 1$ $\qquad$ $\vec{w} \cdot \vec{x}_i - b \leq -1$, if $y_i = -1$



from wikipedia.org

To train the SVM, we minimize the hinge loss:

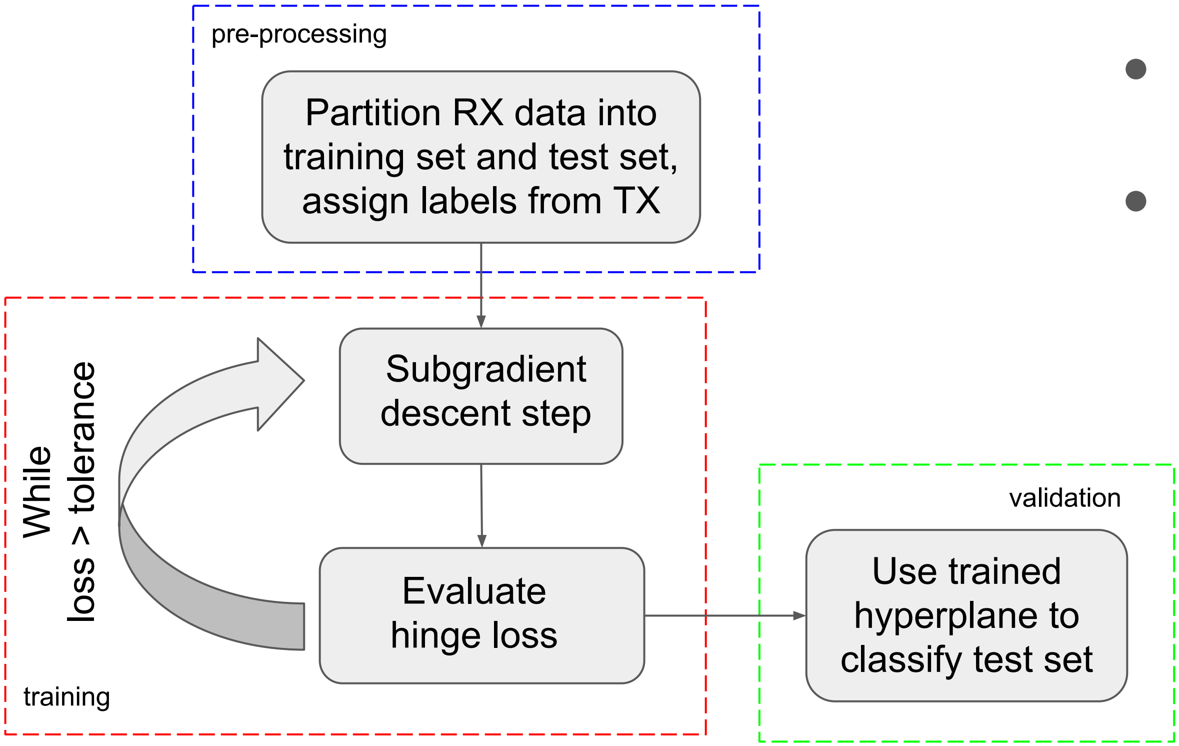
$$\left[\frac{1}{n}\sum_{i=1}^{n} \max\left(0, 1 - y_i(\vec{w} \cdot \vec{x}_i - b)\right)\right]$$

Subgradient descent:

$$\text{if } 1 - y_i(w \cdot x_i - b) > 0: \quad w_m = w_{m-1} + \tfrac{\mu}{n}\textstyle\sum_{i=1}^{n} y_i x_i$$
$$b_m = b_{m-1} - \tfrac{\mu}{n}\textstyle\sum_{i=1}^{n} y_i$$

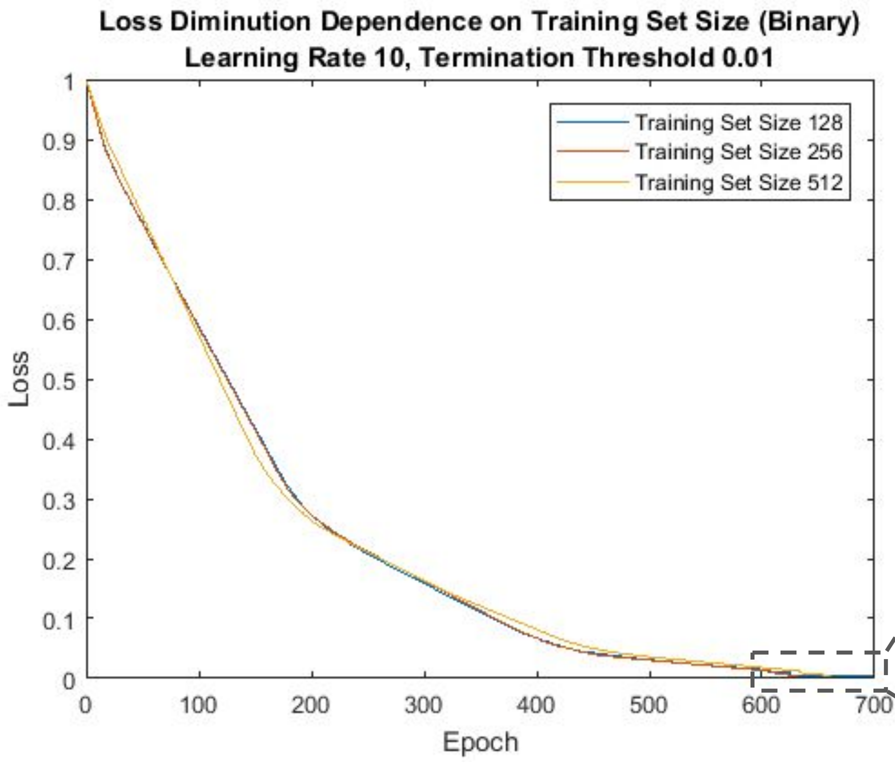
$\mu \stackrel{\Delta}{=} learning\ rate$ $\qquad$ else: $\quad w_m = w_{m-1} \qquad b_m = b_{m-1}$

# Program Flow - SVM

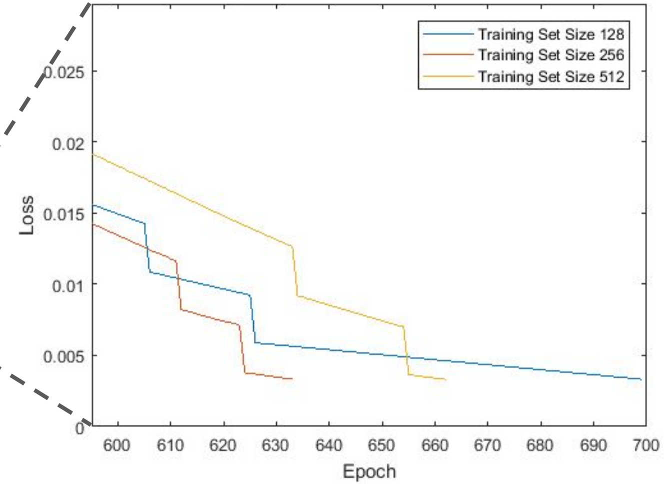



- Single SVM for binary classification
- For 4-PAM, we train two SVMs
  - First hyperplane separates lower and upper power levels
  - Second hyperplane separates between levels close to mean and far from mean

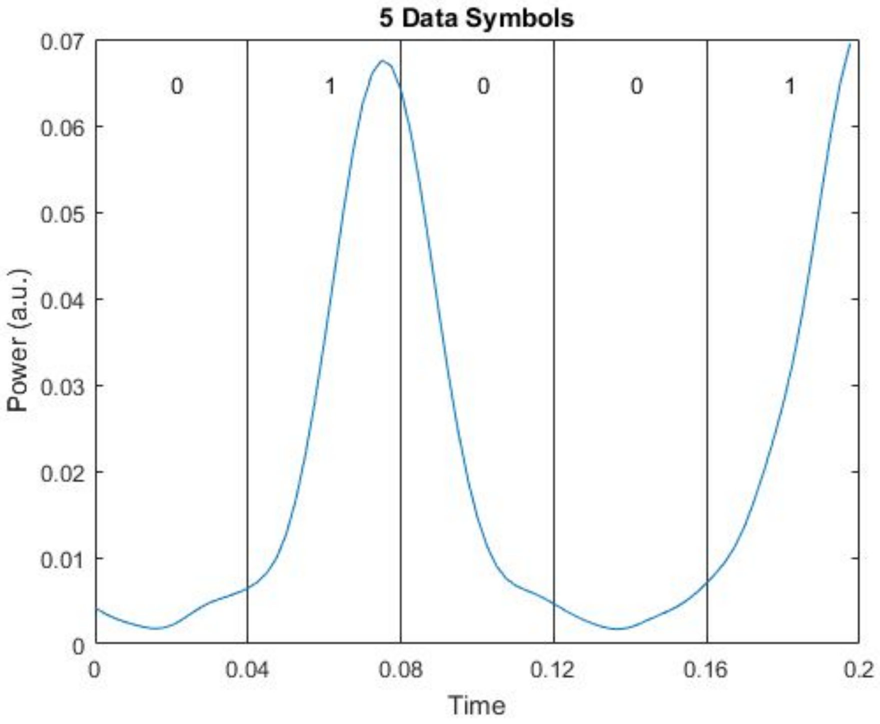
# Hard-margin SVM Results (Binary)

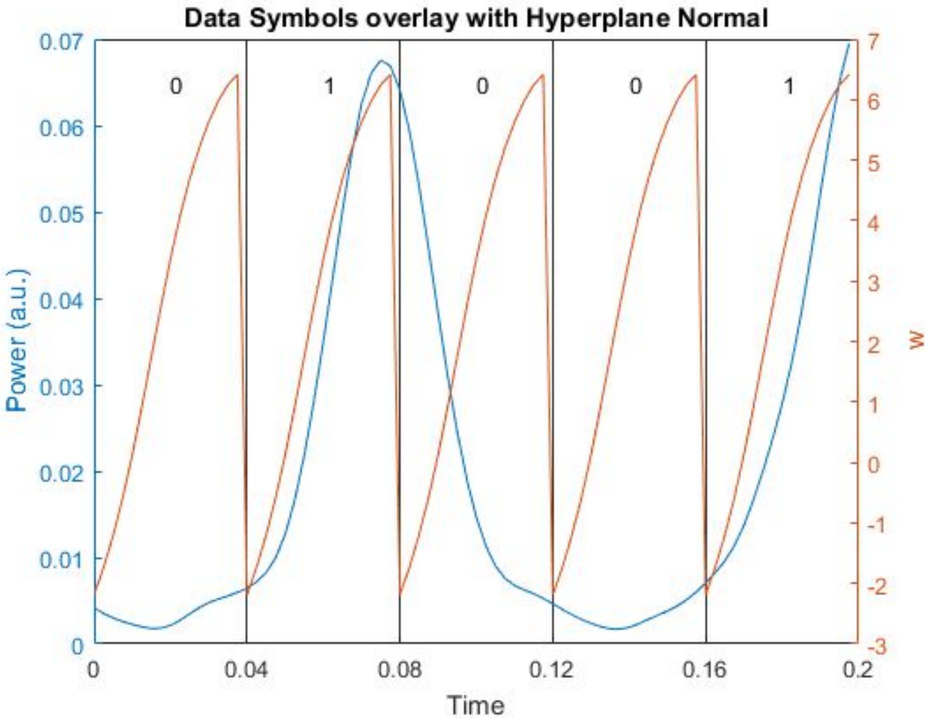



- Loss converges quickly without oscillations
- Running on the test set results in zero misclassifications

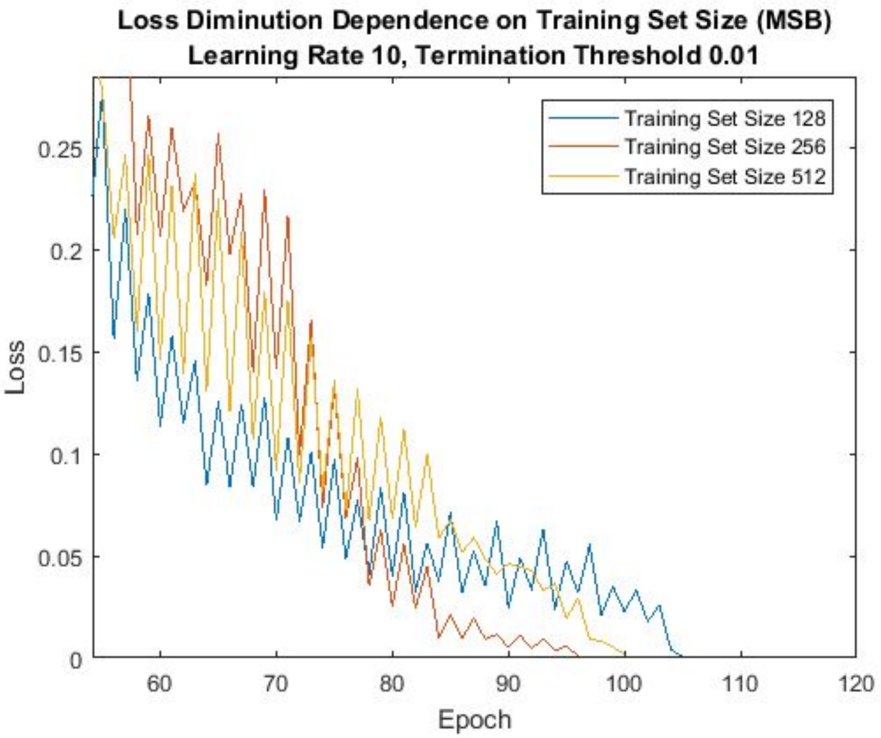


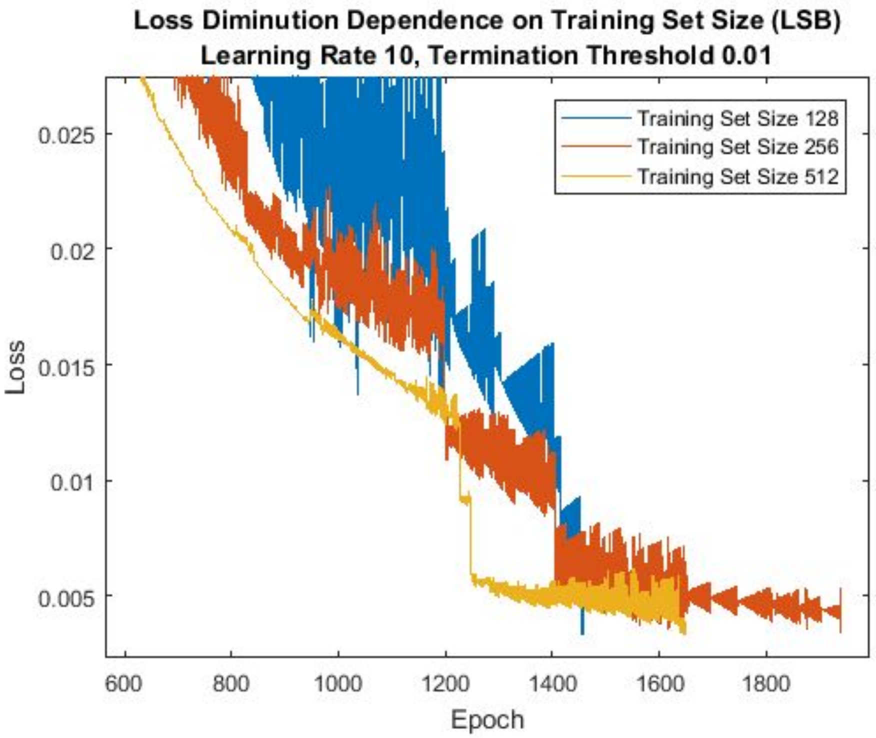
# Optimized Hyperplane



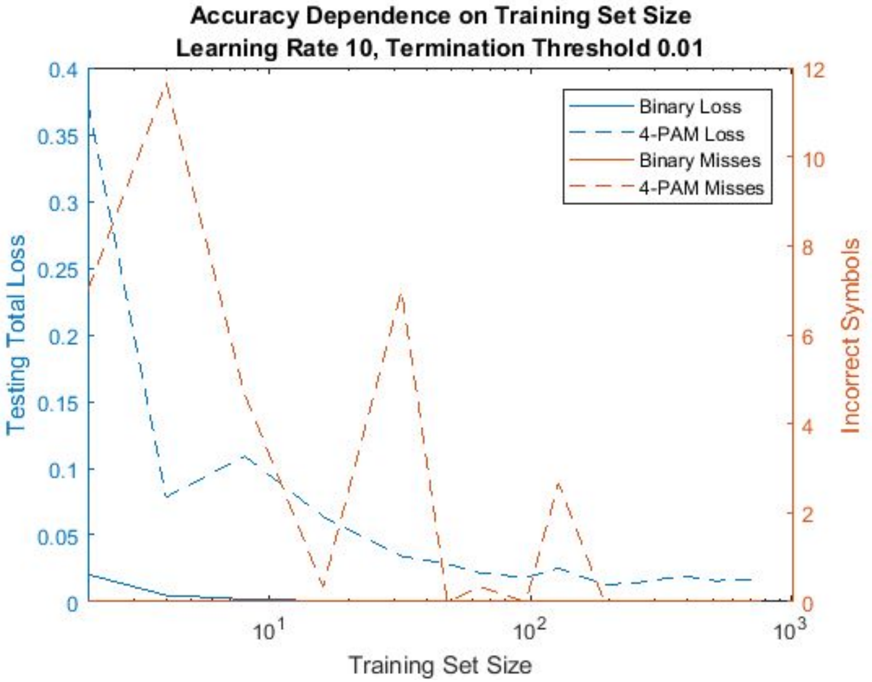



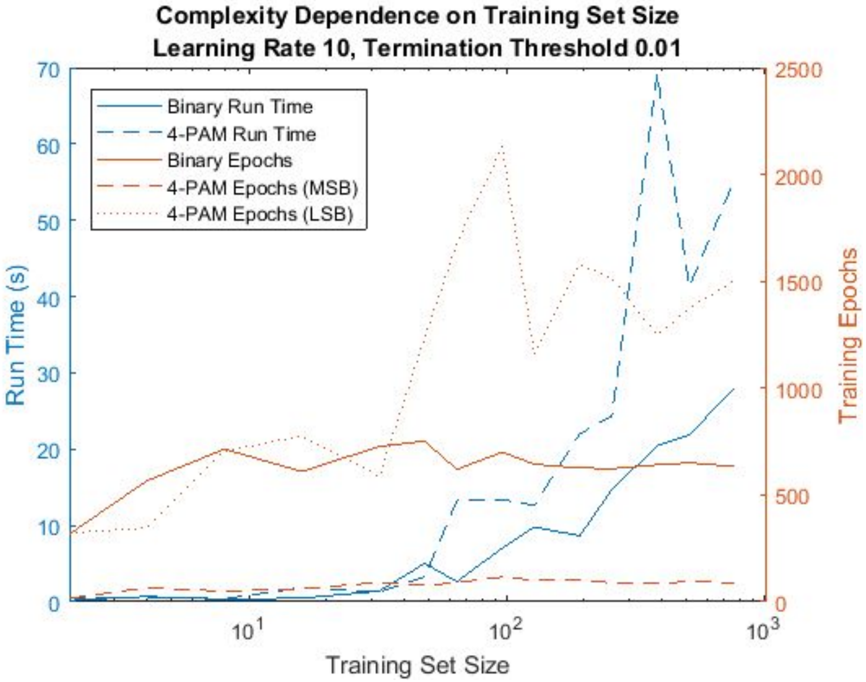
# Hard-margin SVM Results (4-PAM)



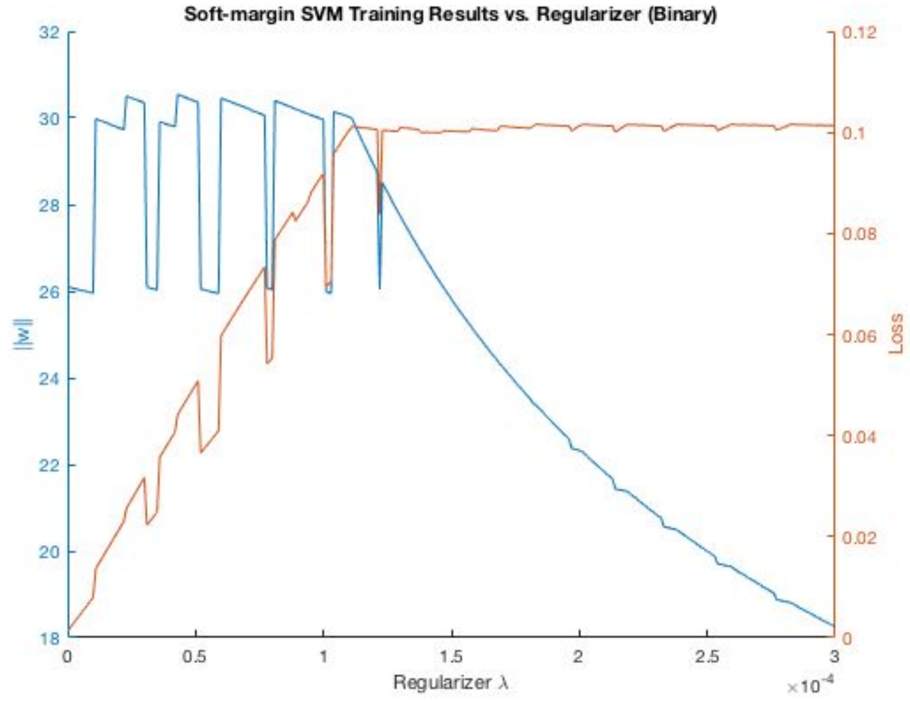



# Hard-margin SVM Accuracy



# Soft-margin SVM



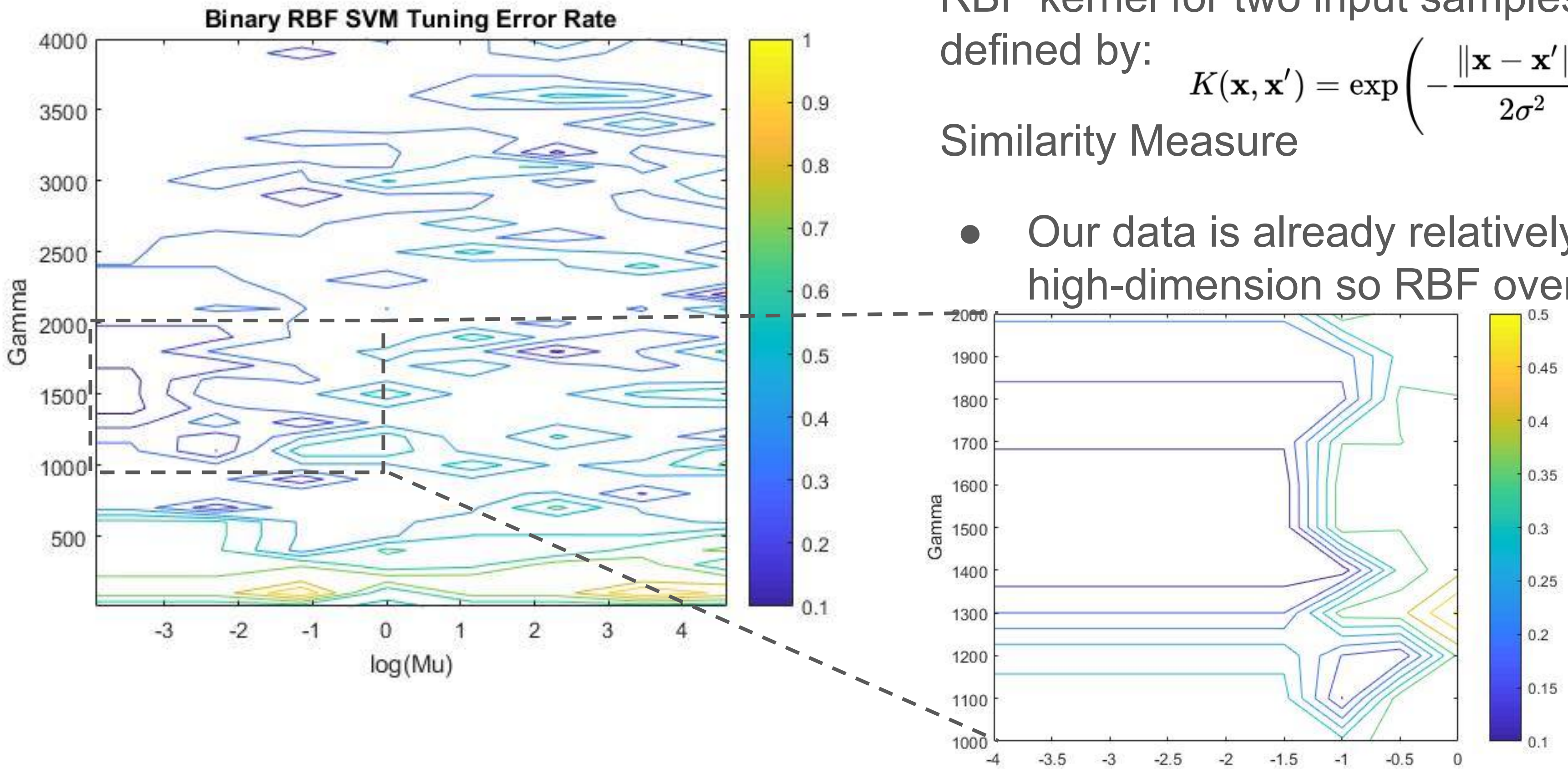

Introduce another term to the loss function:

$$\left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0, 1 - y_i(\vec{w}\cdot\vec{x}_i - b)\right)\right] + \lambda\|\vec{w}\|^2$$

- More points are allowed to violate the margin
- Once the tolerance is reached, ||w|| begins to reduce (margin grows wider)

μ = 10, tolerance = 0.1,
training size = 128 symbols

# Radial Basis Function Kernel



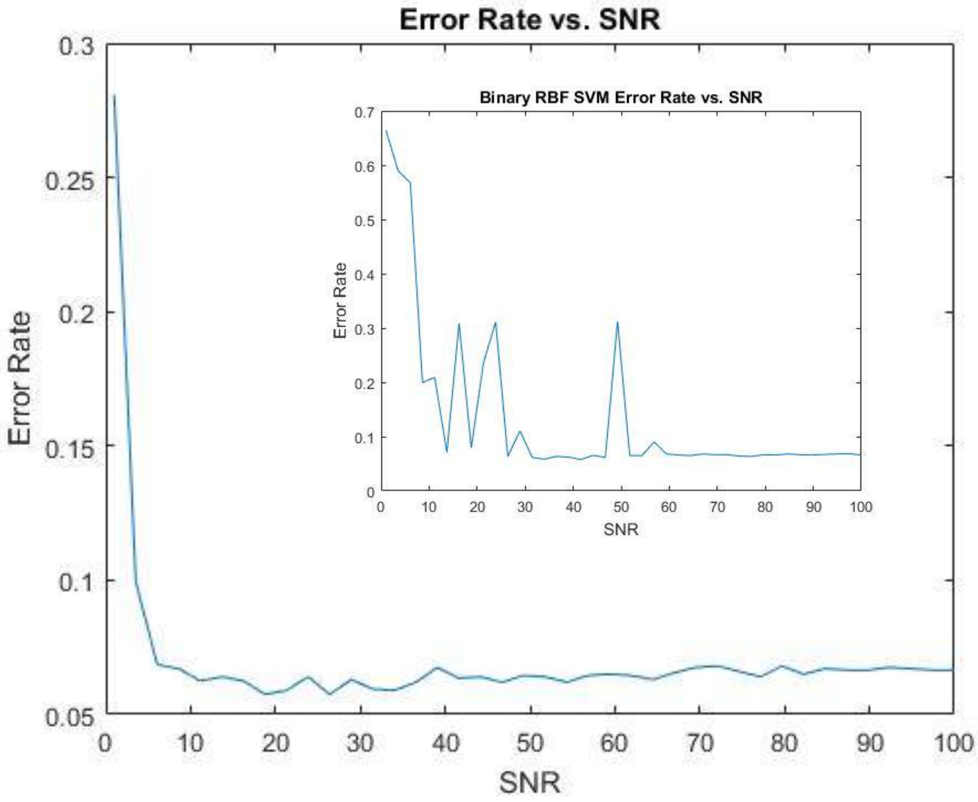

RBF kernel for two input samples defined by:

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)$$

Similarity Measure

- Our data is already relatively high-dimension so RBF overfits

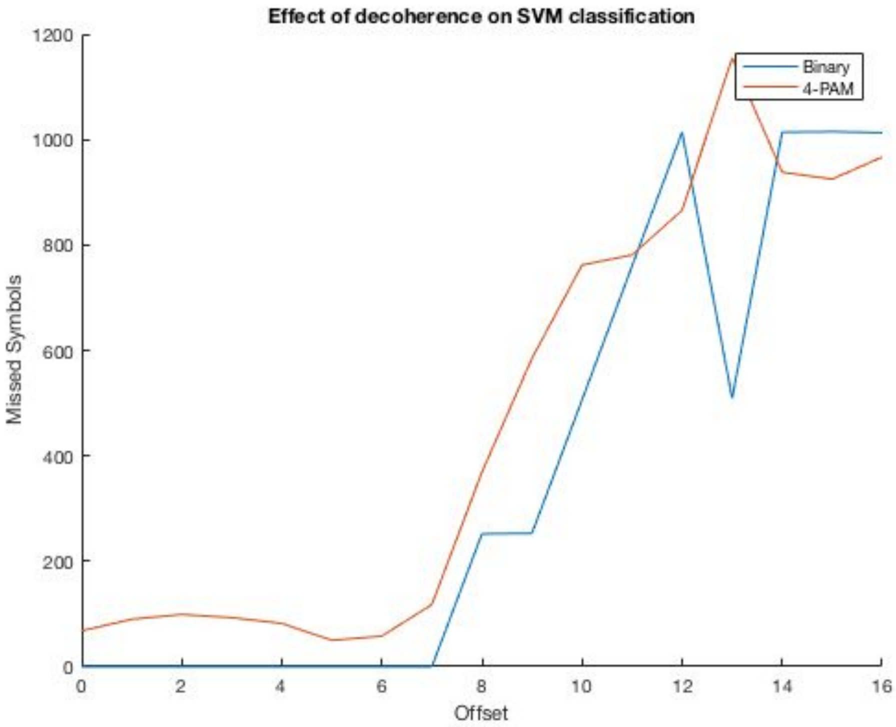
# RBF SVM on a Noisy Binary Channel




- RBF SVM tested on data added with white gaussian noise
- Misclassifications decrease as SNR increases
- High error rate for certain high SNR demonstrates RBF overfitting
- Error Rate 6.91% with no noise

μ = .01, γ = 1500
training size = 64 symbols

# Effect of Signal Desynchronization




- The trained SVM can classify inputs even when the RX data is received late
- Accuracy begins to decrease when input vectors lag by 7 samples

μ = 10, tolerance = 0.01,
training size = 128 symbols

# Conclusions

- Maximum Likelihood Estimator
  - Assumes a uniform prior distribution and is not adaptive
- Convolutional Neural Network
  - Binary data can be perfectly decoded after ~1 minute of training
  - 4-PAM data requires ~10 minutes of training
  - Difficult to get 100% accuracy without a large training set (we used ½ of the data)
- Support Vector Machines
  - Binary data can be perfectly decoded after training for 10 seconds on only 10 samples
  - 4-PAM data requires 30 seconds on a training set on the order of 100 samples
  - MSB classifier trains much faster than LSB
  - SVM is more resilient to non-ideal scenarios

# References

D. Wang, et al., **Nonlinear Decision Boundary Created by a Machine Learning-based Classifier to Mitigate Nonlinear Phase Noise.** *Ecoc (2015); ID:0720.*

B. Zhao, et al., **Waveforms Classification based on Convolutional Neural Networks.** *IEEE (2017); DOI:10.1109.*

D. J. Sebald, et al., **Support Vector Machines and the Multiple Hypothesis Test Problem.** *Trans. Signal Process., vol. 49, no. 11, p. 2865 (2001).*

A. van den Oord, et al., **WaveNet: A Generative Model for Raw Audio.** *Google DeepMind (2016); arXiv:1609.03499v2.*

Questions?