ENGR-E 533 "Deep Learning Systems"
Lecture 04: The First Layer

Minje Kim

Department of Intelligent Systems Engineering

Email: minje@indiana.edu

Website: http://minjekim.com

Research Group: http://saige.sice.indiana.edu

Meeting Request: http://doodle.com/minje

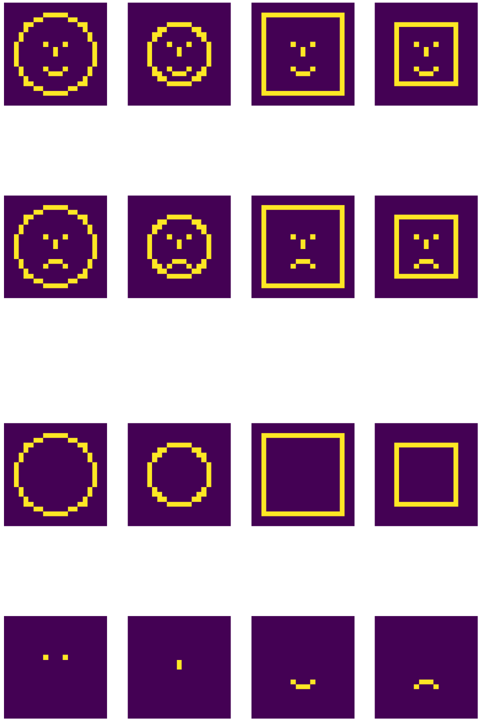
INDIANA UNIVERSITY

SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

Unsupervised Feature Learning

- What do we do in the first layer?

- Before deep learning, a common practice was to extract features and then learn a supervised model
  - Source separation
    - Convert the time domain audio signals into matrices by using Short-Time Fourier Transform (STFT) and then learn dictionaries
  - Object recognition
    - Extract a bunch of different features (e.g. HoG, SIFT, etc) and then build a classifier
  - Sentiment analysis
    - Preprocess the text, learn topics, and then build a classifier
  - Speech recognition
    - Extract Mel-Frequency Cepstrum Coefficients (MFCC) and then learn Hidden Markov Models
- YALT
  - In our baby facial expression recognition problem we manually extracted the features first
  - And then built a softmax classifier



Unsupervised Feature Learning
- What do we do in the first layer?

- Today we will learn how to systematically learn those features from raw data
  - We call this procedure **unsupervised feature learning**
  - Don’t worry, we’re not getting away from neural networks
- Why unsupervised?
  - Supervision here means human intervention to solve the problem
    - e.g. When you train a classifier, you need a bunch of pairs: a data sample (facial image) and its label (happy or sad)
    - You let the model know how you think of the problem
    - Mathematically, this can be done by either learning the mapping function between the data sample and its label
      $\boldsymbol{y} = \mathcal{F}(\boldsymbol{x})$ $\hat{\boldsymbol{y}} = \mathcal{G}(\boldsymbol{x} \; ; \mathbb{W})$
    - Or, finding the parameters that maximize the *a posteriori* probability
      $P(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\Theta})$
  - Unsupervised learning
    - You don’t have the human intervention, or the labels
    - Then, what are we learning?
    - A model that best describes the data

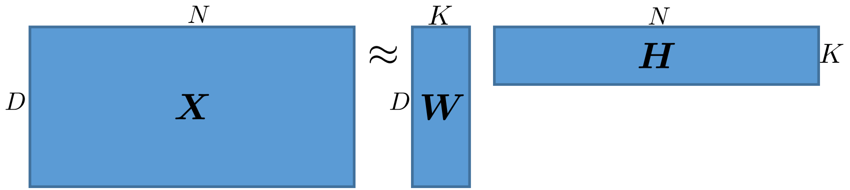
Unsupervised Feature Learning
- What do we do in the first layer?

- Modeling the data
  - If we use a probabilistic distribution it's to find out the maximum likelihood solution

    $P(\boldsymbol{x}; \boldsymbol{\Theta})$
  - Often, this can be viewed as a latent variable analysis with K latent variables, too

The data matrix → $\boldsymbol{X} \approx \boldsymbol{W}\boldsymbol{H}$ ← The activation of the latent variable

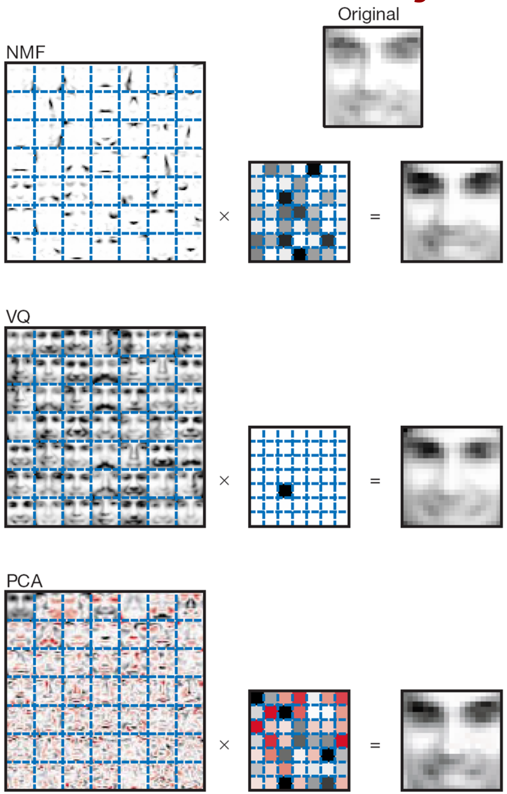
The representative vectors of the latent variable



- For example
  - If you find the eigenvectors of $\boldsymbol{X}\boldsymbol{X}^\top$, that will correspond to $\boldsymbol{W}$
    - $\boldsymbol{W}^\top\boldsymbol{W} = \boldsymbol{I}$ and, therefore, $\boldsymbol{W}^\top\boldsymbol{X} = \boldsymbol{H}$. Also, rows of $\boldsymbol{H}$ are ordered in their variances
    - Principal Component Analysis
    - $\boldsymbol{H}$ is a set of lower dimensional features that can still describe the original distribution
  - If $\boldsymbol{X}_{:,n} \sim \sum_k \boldsymbol{H}_{k,n} \mathcal{N}(\boldsymbol{W}_{:,k}, \boldsymbol{\Sigma}_{:,:,k})$ and $\boldsymbol{H}_{:,n}$ is a one hot vector
    - Gaussian Mixture Model (a.k.a. Vector Quantization)
  - If you find $\boldsymbol{W}$ and $\boldsymbol{H}$ that minimize the approximation error, but are nonnegative at the same time
    - Nonnegative Matrix Factorization
    - $\boldsymbol{H}$ gives you the parts-based representation

Unsupervised Feature Learning

- What do we do in the first layer?



NMF estimates parts-based representations, something like Lego blocks.
Reconstruction is a linear combination of them, but subtraction is not allowed.

VQ finds a bunch of cluster means.
Reconstruction is choosing the most similar mean.

PCA finds the holistic eigenfaces.
From the important one down to the subtle ones.

Lee&Seung Nature 1999, Lee&Seung NIPS 2001

Unsupervised Feature Learning

- What do we do in the first layer?

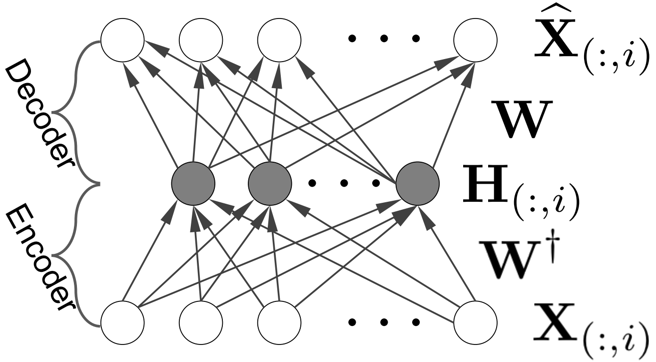
- There are so many different ways to solve the linear approximation $\boldsymbol{X} \approx \boldsymbol{W}\boldsymbol{H}$
- So, eventually it's all about how to constrain the problem

$$\arg\min_{\boldsymbol{W},\boldsymbol{H}} \mathcal{D}(\boldsymbol{X}||\boldsymbol{W}\boldsymbol{H}) + \lambda_{\boldsymbol{W}} f(\boldsymbol{W}) + \lambda_{\boldsymbol{H}} g(\boldsymbol{H})$$

  - For clustering, you want a super sparse column vectors in $\boldsymbol{H}$ so that the data sample is one of the cluster means, deviated accordingly
  - For PCA, you want the after-projection-samples are with maximal variances
  - For NMF, you constrain $\boldsymbol{W}, \boldsymbol{H}$ to be nonnegative (element-wise)
  - ...
- It is convenient to assume that there exists some kind of pseudo inverse of $\boldsymbol{W}$
  - For PCA, $\boldsymbol{W}^\dagger = \boldsymbol{W}^\top$
  - For NMF, $\boldsymbol{W}^\dagger = \boldsymbol{W}^\top$ depending on the choice of the error function $\boldsymbol{H} \leftarrow \boldsymbol{H} \odot \frac{\boldsymbol{W}^\top \boldsymbol{X}}{\boldsymbol{W}^\top \boldsymbol{W}\boldsymbol{H}}$
- If you assume this pseudo inverse of the basis vectors, you can think of the projection as a way to convert your data into features $\boldsymbol{W}^\dagger \boldsymbol{X} \approx \boldsymbol{W}^\dagger \boldsymbol{W}\boldsymbol{H} \approx \boldsymbol{H}$

The Autoencoders

- A unified neural network-based representation of unsupervised learning



$$\arg\min_{\boldsymbol{W},\boldsymbol{W}^{\dagger},\boldsymbol{H}} \mathcal{D}(\boldsymbol{X}||\boldsymbol{W}\boldsymbol{W}^{\dagger}\boldsymbol{X}) + \lambda_{\boldsymbol{W}} f(\boldsymbol{W}) + \lambda_{\boldsymbol{W}^{\dagger}} f(\boldsymbol{W}^{\dagger}) + \lambda_{\boldsymbol{H}} g(\boldsymbol{H})$$

- Once the objective function is setup in this way, you can estimate the parameters using SGD
  - TF or PT will take care of this part using automatic gradient calculation

The Autoencoders

- A unified neural network-based representation of unsupervised learning

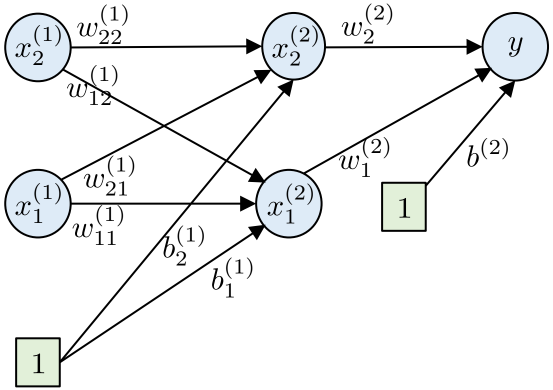
- Demo
  - Sparse coding
  - NMF

Nonlinearity in Neural Networks

- Stacked linear models form another linear model

- So far I haven't said anything about nonlinearity in the lower layers (at least officially)
- Why do we need it?
  - Because stacked linear models form yet another linear model
- Stacked linear models

$$
\begin{aligned}
x_1^{(2)} &= w_{11}^{(1)} x_1^{(1)} + w_{12}^{(1)} x_2^{(1)} + b_1^{(1)} \\
x_2^{(2)} &= w_{21}^{(1)} x_1^{(1)} + w_{22}^{(1)} x_2^{(1)} + b_2^{(1)} \\
y &= w_1^{(2)} x_1^{(2)} + w_2^{(2)} x_2^{(2)} + b^{(2)}
\end{aligned}
$$



- Form another linear model

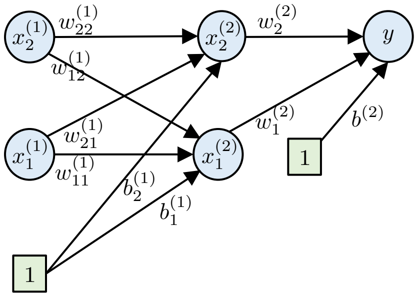
$$
\begin{aligned}
y &= w_1^{(2)} w_{11}^{(1)} x_1^{(1)} + w_1^{(2)} w_{12}^{(1)} x_2^{(1)} + w_1^{(2)} b_1^{(1)} \\
&\quad + w_2^{(2)} w_{21}^{(1)} x_1^{(1)} + w_2^{(2)} w_{22}^{(1)} x_2^{(1)} + w_2^{(2)} b_2^{(1)} + b^{(2)} \\
&= \left(w_1^{(2)} w_{11}^{(1)} + w_2^{(2)} w_{21}^{(1)}\right) x_1^{(1)} + \left(w_1^{(2)} w_{12}^{(1)} + w_2^{(2)} w_{22}^{(1)}\right) x_2^{(1)} + w_1^{(2)} b_1^{(1)} + w_2^{(2)} b_2^{(1)} + b^{(2)}
\end{aligned}
$$

  - Or $\quad y = \boldsymbol{w}^{(2)}(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}) + b^{(2)}$

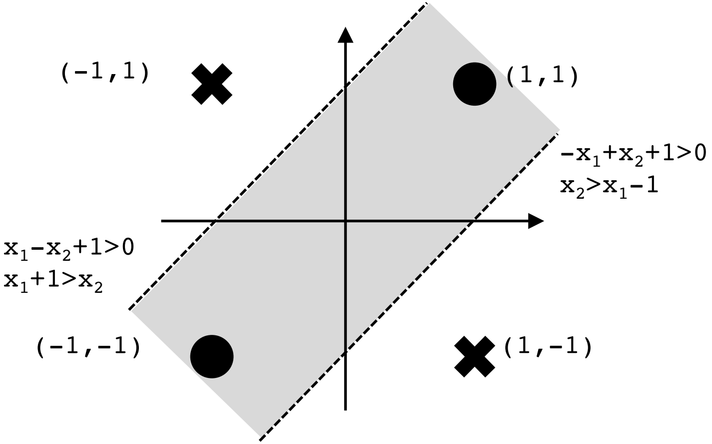
$$
= \boldsymbol{w}^{(2)}\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{w}^{(2)}\boldsymbol{b}^{(1)} + b^{(2)}
$$

Nonlinearity in Neural Networks

- Stacked linear models form another linear model



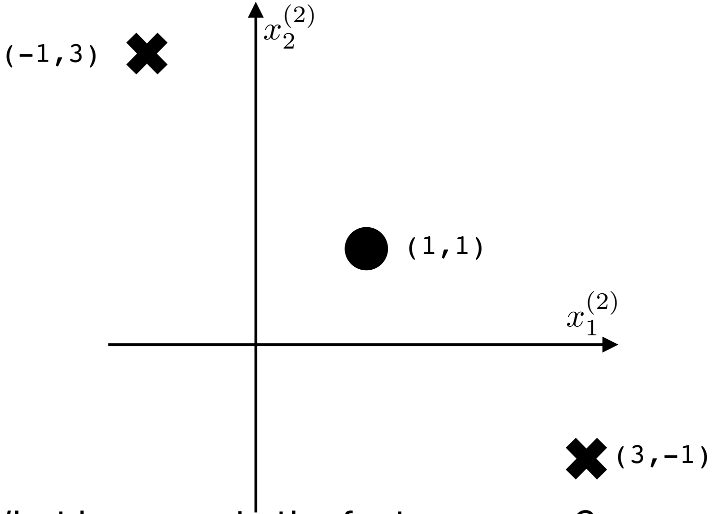
- What can we do then?
- The XOR problem
  - A nonlinear problem that needs two hyperplanes



- How do you transform the data into features?
  - It must have something to do with hyperplanes

$$\boldsymbol{W}^{(1)} = \begin{bmatrix} +1 & -1 & 1 \\ -1 & +1 & 1 \end{bmatrix}$$

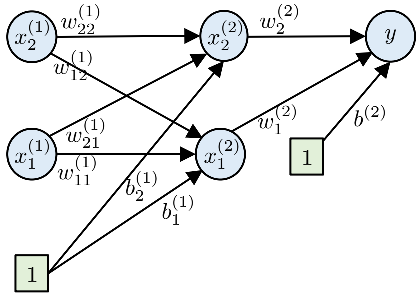
- The new feature space



- What happens in the feature space?
  - If it were not for the help from the nonlinearity
  - Still NOT linearly solvable!

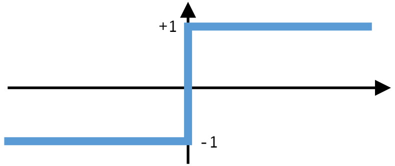
Nonlinearity in Neural Networks

- Adding nonlinearity



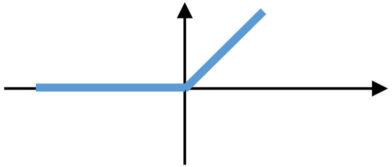
- With a sign function

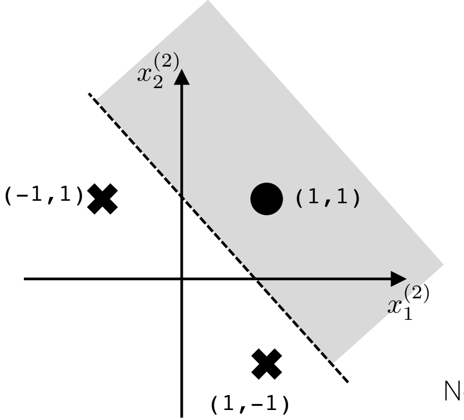
$$x_i^{(2)} = \mathrm{sign}(\boldsymbol{W}_{i,:}^{(1)}\boldsymbol{x} + b_i^{(1)})$$

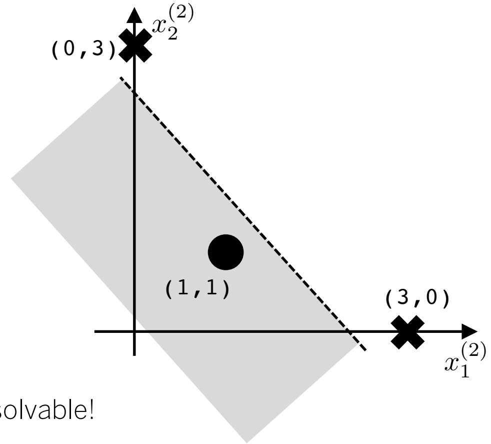


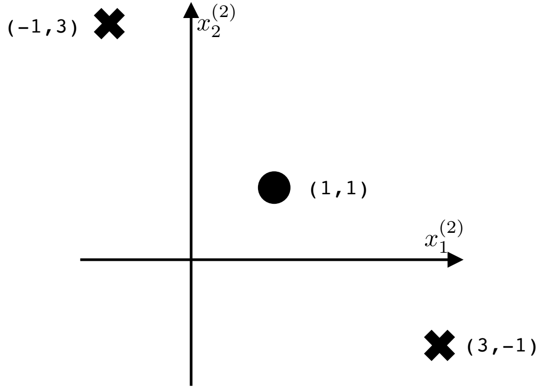
- With a Rectified Linear Unit (ReLU)

$$x_i^{(2)} = \max(\boldsymbol{W}_{i,:}^{(1)}\boldsymbol{x} + b_i^{(1)}, 0)$$







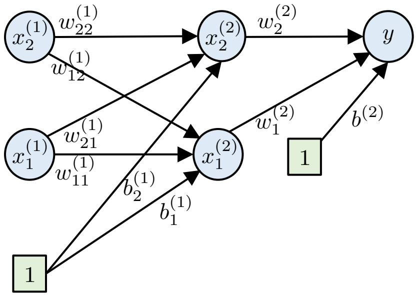


Now they are linearly solvable!

New hyperplane corresponds to the last layer weights

Nonlinearity for AEs?

- What does the XOR example mean for unsupervised learning?



- You can replace the final layer binary output with the input vector
  - Classifier turns into an AE
- Suppose you can estimate these weights of the XOR network by using BP
- Can you also estimate these weights with the AE setup?
- You just saw that nonlinearity can help producing better features
- AE might benefit from it to

$$\arg\min_{\boldsymbol{W},\boldsymbol{W}^\dagger,\boldsymbol{H}} \mathcal{D}(\boldsymbol{X}||\boldsymbol{W}\sigma(\boldsymbol{W}^\dagger\boldsymbol{X})) + \lambda_{\boldsymbol{W}} f(\boldsymbol{W}) + \lambda_{\boldsymbol{W}^\dagger} f(\boldsymbol{W}^\dagger) + \lambda_{\boldsymbol{H}} g(\boldsymbol{H})$$

- But it's not clear if the encoder weights are going to be actually create useful features for the following classification
- We will cover this part in the next time

The Pipeline for Supervised Learning

- What people have done so far before deep learning

- Feature engineering
  - Tries to come up with a set of features that best describe the data and the problem
  - There can be many different ways (as you've seen)
  - You never know the quality until you actually test out those features for your problem
- Supervised learning (classification/regression)
  - Tries to come up with the best model to best predict the output variable
  - Even if you use a very nice set of features, nonlinearity can be still involved (e.g. kernel methods for SVM)
  - If you don't like the performance, you never know what to blame
    - It could be because of either bad feature engineering or classifier
- A holistic approach
  - In a multilayer perceptron, you have the first layer dedicated to the feature extraction
  - And the last layer as your supervised learning part
  - What if you just optimize the weights of both layers all together?
    - Then, you can skip the feature engineering part
  - Does this work? When does this approach not work? Why?

Thank You!