ENGR-E 533

Deep Learning Systems

Module 02

Deep Neural Network Basics

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

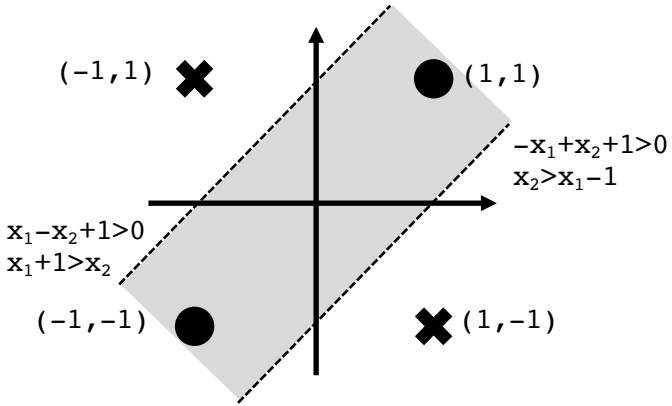
INDIANA UNIVERSITY

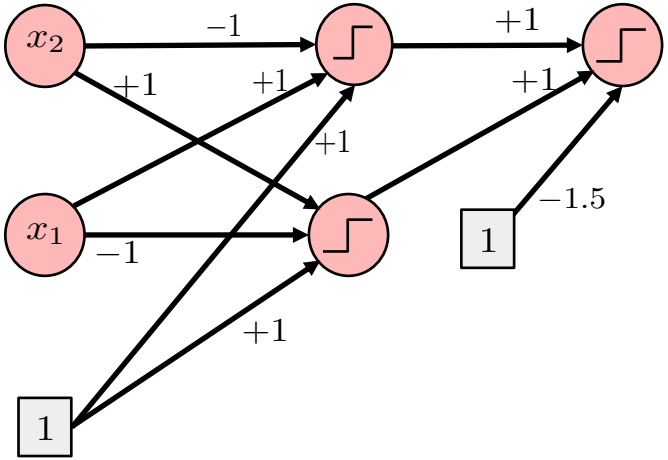
SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

Advantage of Deeper Networks

- Parity function

- NN for the XOR problem





- How may parameters are there?

Advantage of Deeper Networks

- Parity function

- A parity function

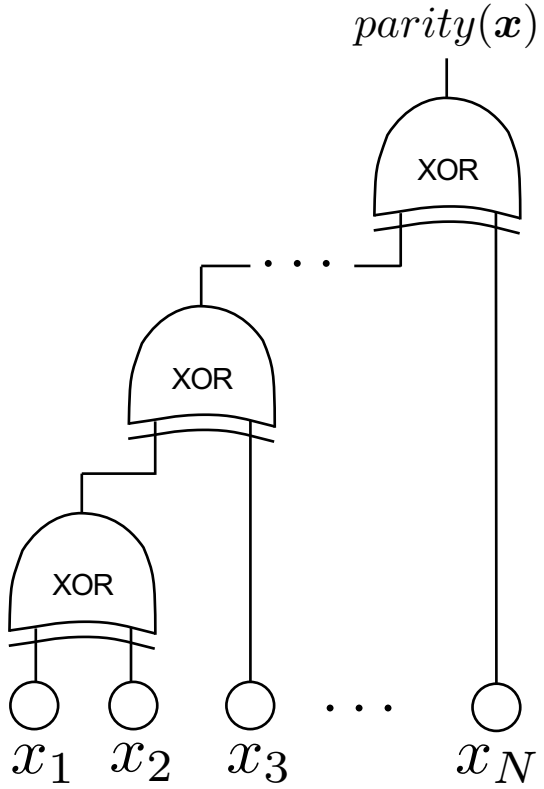
$$parity(\boldsymbol{x}) = \begin{cases} +1 & \text{if the number of } +1 \text{ is odd} \\ -1 & \text{otherwise} \end{cases}$$

  - For a vector of bipolar binaries

- For example,

$$parity([+1, -1, -1, +1, +1, -1]) = +1$$

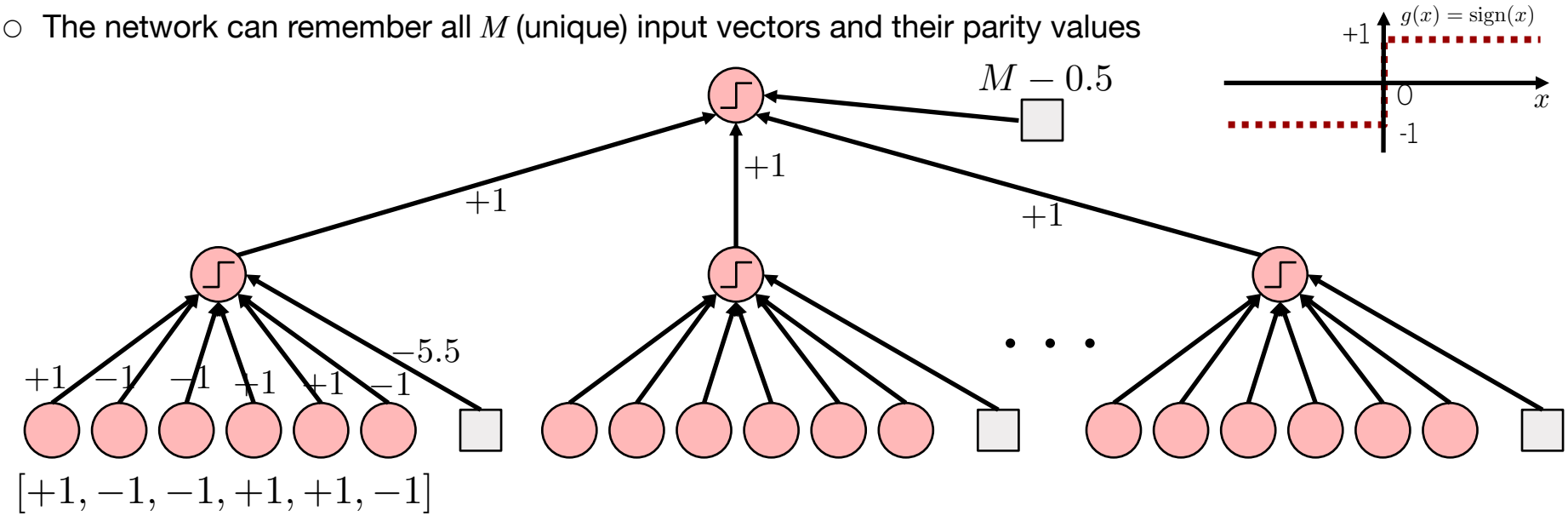
- A network that does this job?
- How many layers?
  - N
- How many weights?
  - 9N



Advantage of Deeper Networks

- Parity function

- A shallower version
- The network can remember all M (unique) input vectors and their parity values

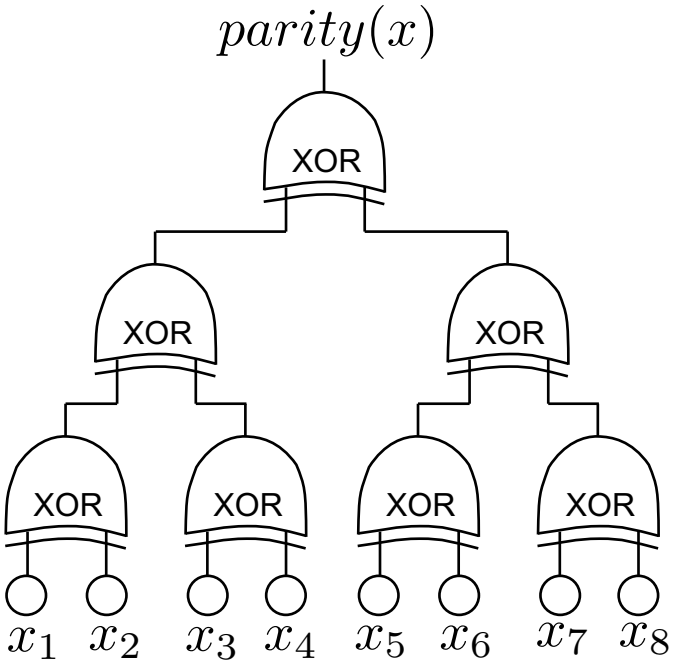


- How many weights do you need?
  - *O(NM)* weights

Advantage of Deeper Networks

- Parity function

- If we reuse the subnetwork carefully
  - Depth: *O(log N)*
  - Weights: *O(N)*



Advantage of Deeper Networks

- Summary

- A lot of possible choices for the neural network structure
- In theory (universal approximation theorem)
  - A shallow neural network with one hidden layer can approximate any mapping function
  - Once we're allowed to use as many hidden units as possible
- In practice
  - This shallow and wide network will be with too many weights to estimate
- A Deep Neural Network (DNN)
  - Has more than one hidden layers
  - Is efficient in its structure to learn a very complicated mapping function
- Question:
  - Then, can we just come up with a neural network with lots of layers?
- Answer:
  - Life is not that simple!

Feature Learning for a Shallow Network

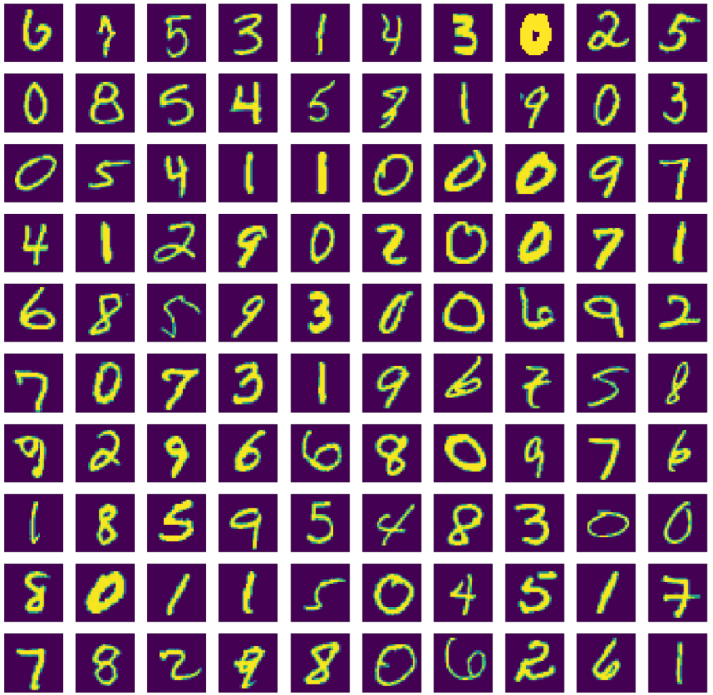
- Overview

- Feature extraction is important
  - But difficult
  - Usually involves unsupervised learning
- Supervised learning
  - Is easy if you start from a good set of features
  - Otherwise you'll need some nonlinearity
- A shallow neural network takes care of both tasks at the same time
  - The first layer learns features
  - The last layer learns the regressor
- Universal Approximation Theorem
  - If there are many many hidden units, a shallow neural network can approximate any (nonlinear) function
  - e.g. Sinusoidal functions
  - But how many?

Feature Learning for a Shallow Network

- MNIST

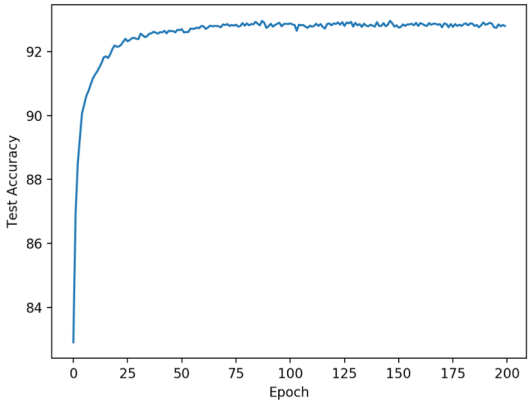
- MNIST Hand Written Digit Recognition
  - Considered as the “Hello, World!” problem in deep learning
  - 28X28=784 pixels per image
  - 55,000 training images
  - 10,000 testing images

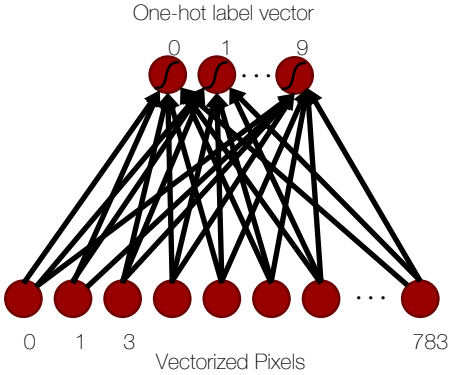


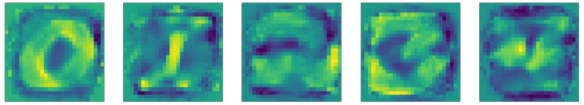
Feature Learning for a Shallow Network

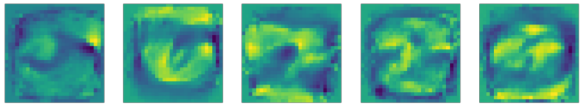
- Softmax regression

- A softmax classification with no feature extraction
  - i.e. Raw pixel intensities are the features

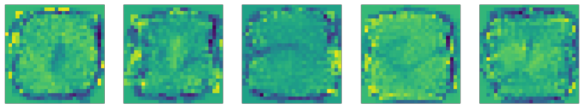


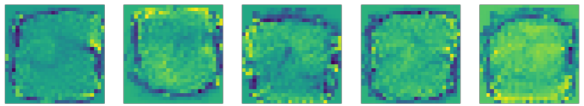






Weights at 20th epoch





Weights at 200th epoch

- It's not bad, but how about some feature extraction?
  - YALT last time

Feature Learning for a Shallow Network

- AEs for feature learning

- Let's extract some features, and then do the softmax
- Two cases: naïve AE versus sparse AE
- The AE objective function

$$\arg\min_{\boldsymbol{W},\boldsymbol{H}} \mathcal{D}(\boldsymbol{X}||\boldsymbol{W}^{\dagger}\boldsymbol{H}) + \lambda g(\boldsymbol{H}), \quad \text{where} \boldsymbol{H} = \sigma(\boldsymbol{W}\boldsymbol{X})$$

  - With the sparsity constraint

$$\hat{\alpha} = \frac{1}{T}\sum_{j=0}^{T-1} \boldsymbol{H}_{i,j}$$

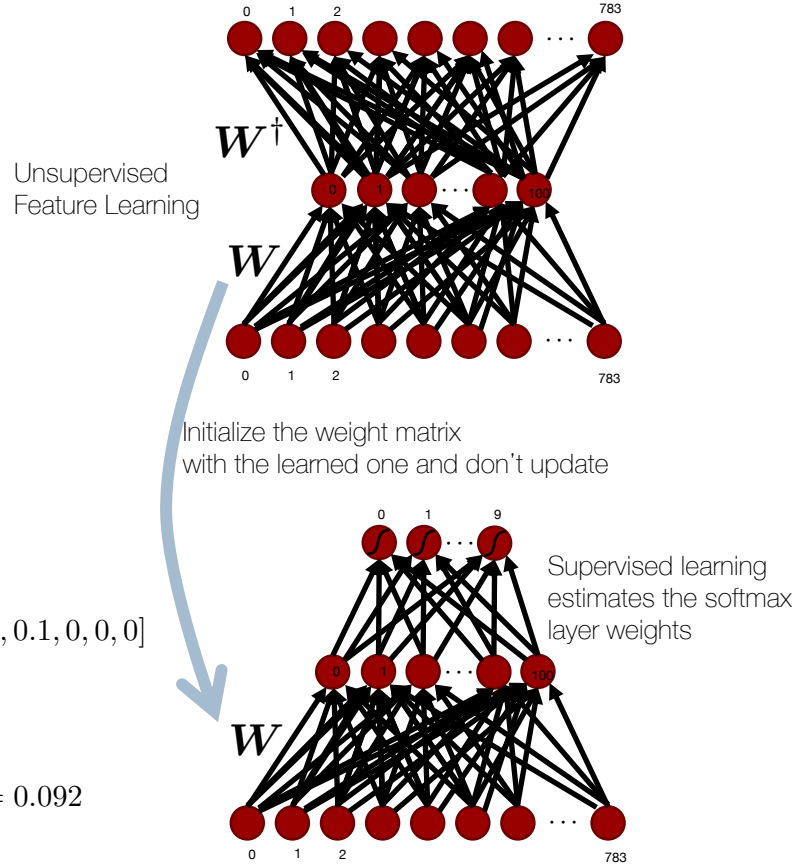
$$g(\boldsymbol{H}_{i,:}) = KL(\alpha||\hat{\alpha}) = -\alpha \log\left(\frac{\alpha}{\hat{\alpha}}\right) - (1-\alpha)\log\left(\frac{1-\alpha}{1-\hat{\alpha}}\right)$$

$$g(\boldsymbol{H}) = \sum_i g(\boldsymbol{H}_{i,:})$$

- For example, $x = [1,0,1,0,0,0,0,0,0,0]$ $\hat{\alpha} = 0.2$ If $\alpha = 0.2$ $g(x) = 0$

  $x = [0.8,0,0.9,0,0.2,0,0.1,0,0,0]$ $\hat{\alpha} = 0.2$

  - A less sparse one $x = [0.8,1,0.9,1,0.2,0,0.1,0,0,0]$ $\hat{\alpha} = 0.4$ $g(x) = 0.092$

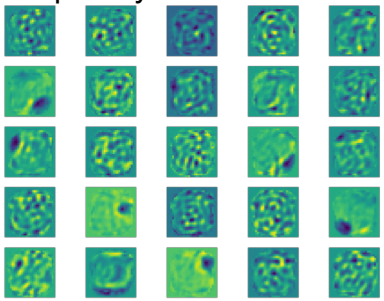


Feature Learning for a Shallow Network

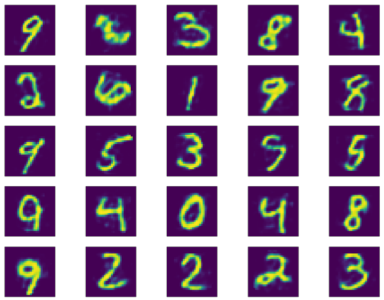
- AEs for feature learning

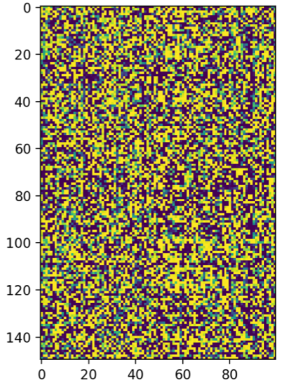
- Case A: Naïve AE
  - No sparsity constraint

Weights



Recons.



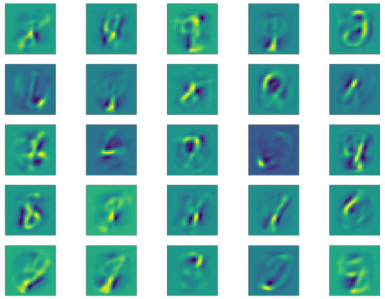


Hidden unit activations

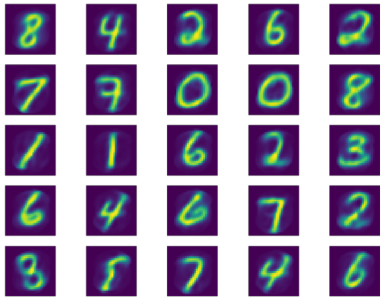
H

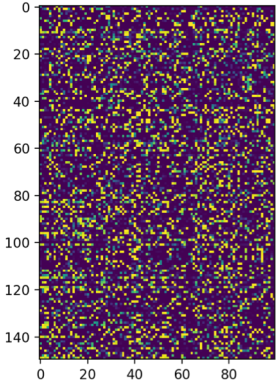
- Case B: Sparse AE
  - $\lambda = 0.1,\ \alpha = 0.2$

Weights



Recons.





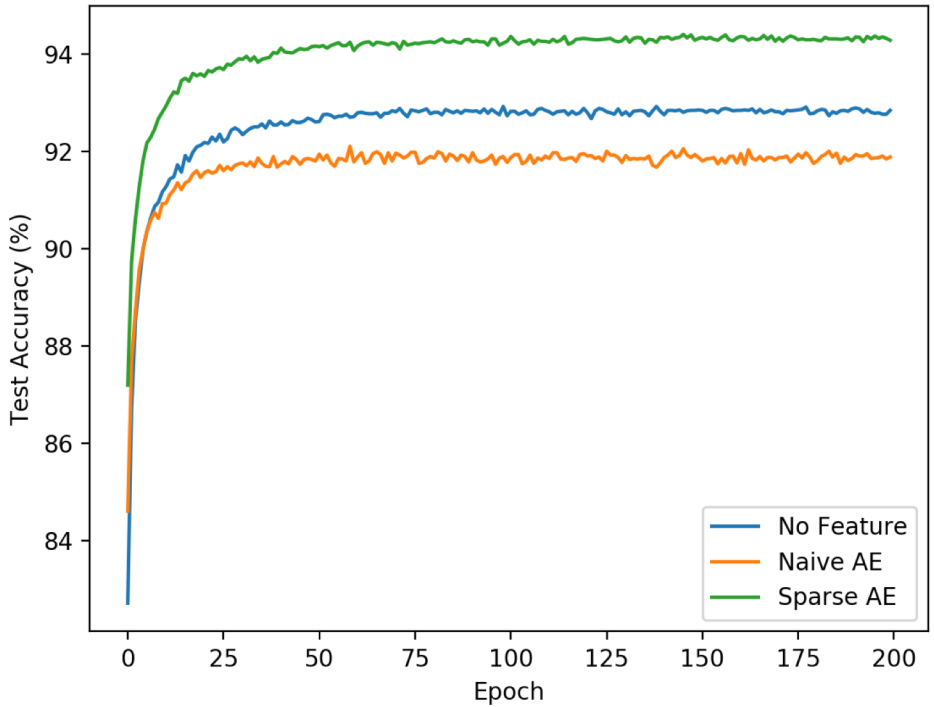
Hidden unit activations

H

Feature Learning for a Shallow Network

- AEs for feature learning

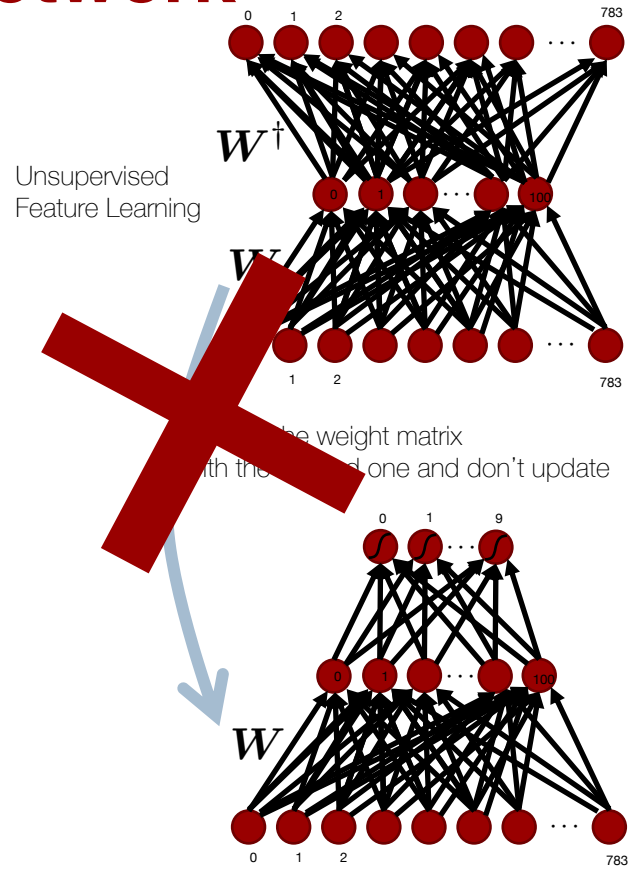
- A direct softmax classification on the raw pixel intensities (blue line)
  - Works pretty well
  - No features
- If your features are NOT good enough (naïve AE)
  - They degrade the performance
- If your features are good enough (sparse AE)
  - They improve the performance



Feature Learning for a Shallow Network

- Feature extraction + softmax?

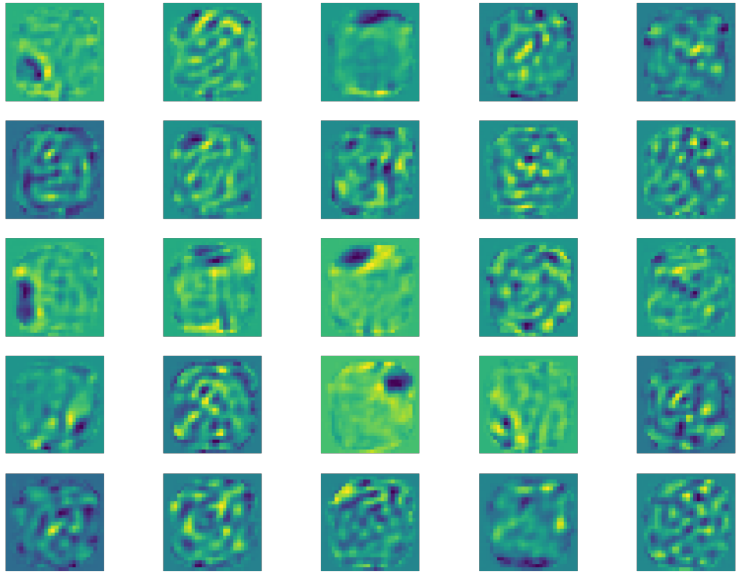
- Big question: why bother learning the features?
  - Feature learning can have a lot of different choices
    - e.g. regularization, constraints, ranks, error functions, activation functions, etc.
    - Could be unpredictable until you go ahead and test them out
  - Supervised learning has a lot of options, too
    - e.g. choice of the kernel functions, generative models, hyperparameters, etc.
- We can learn a neural network with one hidden layer
  - Randomly initialize both weight matrices in the two layers
  - Estimate them using backpropagation
  - Does it work better than the previous methods with feature engineering?
    - See the graph in the next page



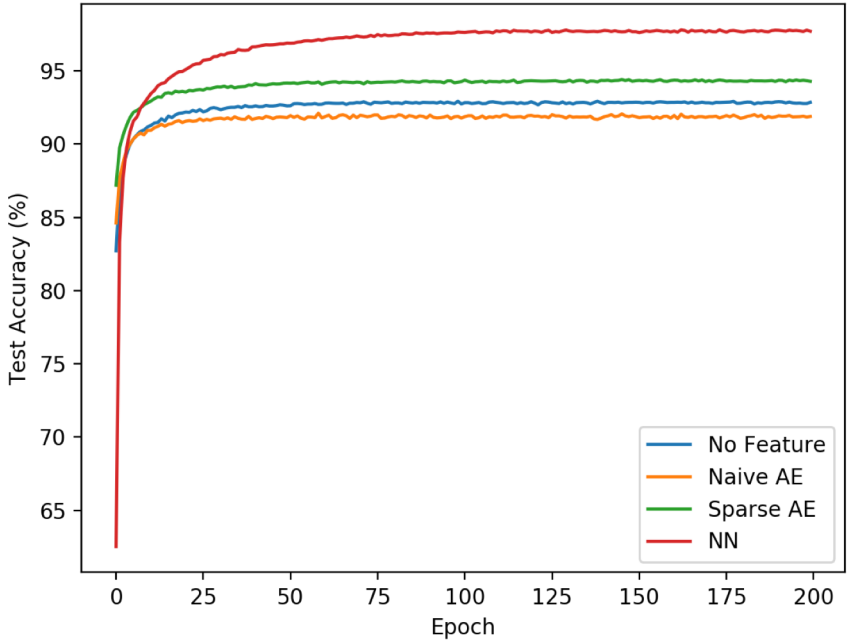
Backpropagation on a Shallow Network

- Feature learning is harmonized with NN training

- The first layer works as the feature learning process



The First Layer Weights



Backpropagation on a Shallow Network

- Going under the last layer

- i=0
  1. Initialize parameters with small random numbers
  2. Calculate the output using all training samples (i.e. input and target pairs)

$$\boldsymbol{Z}^{(1)} \leftarrow \boldsymbol{W}^{(1)}\boldsymbol{X}^{(1)} + \boldsymbol{b}^{(1)} \qquad \boldsymbol{X}^{(2)} \leftarrow \sigma(\boldsymbol{Z}^{(1)})$$

$$\boldsymbol{Z}^{(2)} \leftarrow \boldsymbol{W}^{(2)}\boldsymbol{X}^{(2)} + \boldsymbol{b}^{(2)} \qquad \hat{\boldsymbol{Y}} \leftarrow g(\boldsymbol{Z}^{(2)})$$

Softmax for classification

  3. Calculate the error (cost)

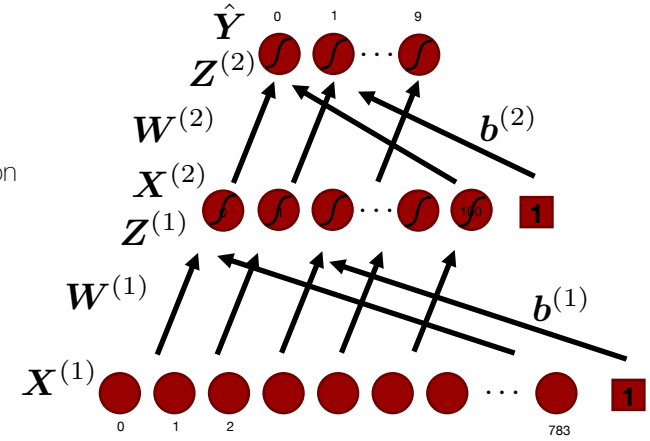
$$\mathcal{E} = \sum_t \mathcal{D}(\boldsymbol{Y}_{:,t} || \hat{\boldsymbol{Y}}_{:,t})$$

Cross entropy

  4. Update the parameters

$$\boldsymbol{W}^{(2)} \leftarrow \boldsymbol{W}^{(2)} - \rho\frac{\partial \mathcal{E}}{\partial \boldsymbol{W}^{(2)}} \qquad \boldsymbol{W}^{(1)} \leftarrow \boldsymbol{W}^{(1)} - \rho\frac{\partial \mathcal{E}}{\partial \boldsymbol{W}^{(1)}}$$

$$\boldsymbol{b}^{(2)} \leftarrow \boldsymbol{b}^{(2)} - \rho\frac{\partial \mathcal{E}}{\partial \boldsymbol{b}^{(2)}} \qquad \boldsymbol{b}^{(1)} \leftarrow \boldsymbol{b}^{(1)} - \rho\frac{\partial \mathcal{E}}{\partial \boldsymbol{b}^{(1)}}$$



- i>0
  - Repeat 2-4

Backpropagation on a Shallow Network

- Going under the last layer

$$
\begin{aligned}
\boldsymbol{Z}^{(1)} &\leftarrow \boldsymbol{W}^{(1)}\boldsymbol{X}^{(1)} + \boldsymbol{b}^{(1)} \\
\boldsymbol{X}^{(2)} &\leftarrow \sigma(\boldsymbol{Z}^{(1)}) \\
\boldsymbol{Z}^{(2)} &\leftarrow \boldsymbol{W}^{(2)}\boldsymbol{X}^{(2)} + \boldsymbol{b}^{(2)} \\
\hat{\boldsymbol{Y}} &\leftarrow g(\boldsymbol{Z}^{(2)})
\end{aligned}
$$

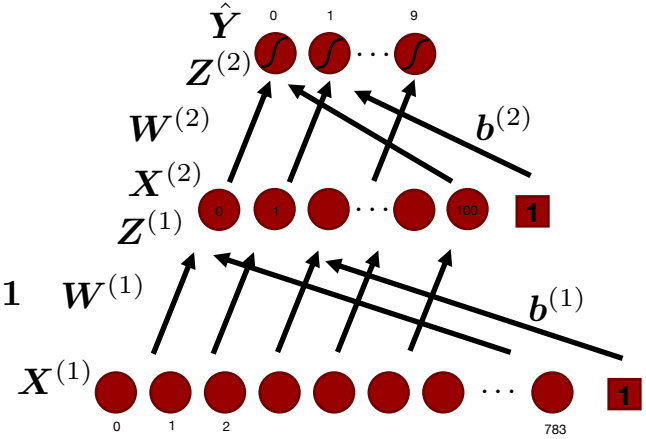
- For $\boldsymbol{W}^{(2)}$ and $\boldsymbol{b}^{(2)}$
  - YALT

$$
\frac{\partial \mathcal{E}}{\partial \boldsymbol{W}^{(2)}} = \underbrace{\frac{\partial \mathcal{E}}{\partial \hat{\boldsymbol{Y}}}\frac{\partial \hat{\boldsymbol{Y}}}{\partial \boldsymbol{Z}^{(2)}}}_{\text{BP Error } \delta^{(2)}}\frac{\partial \boldsymbol{Z}^{(2)}}{\partial \boldsymbol{W}^{(2)}} = \underbrace{(\hat{\boldsymbol{Y}} - \boldsymbol{Y})}_{\text{BP Error } \delta^{(2)}}\boldsymbol{X}^{(2)\top} \qquad \frac{\partial \mathcal{E}}{\partial \boldsymbol{b}^{(2)}} = (\hat{\boldsymbol{Y}} - \boldsymbol{Y})\mathbf{1}
$$

  - Backpropagation Error $\delta^{(l)}$: the derivative of the cost function w.r.t. the **input** to the (l)-th hidden layer **units**

- For $\boldsymbol{W}^{(1)}$ and $\boldsymbol{b}^{(1)}$

$$
\begin{aligned}
\frac{\partial \mathcal{E}}{\partial \boldsymbol{W}^{(1)}} &= \underbrace{\frac{\partial \mathcal{E}}{\partial \hat{\boldsymbol{Y}}}\frac{\partial \hat{\boldsymbol{Y}}}{\partial \boldsymbol{Z}^{(2)}}}_{\text{BP Error } \delta^{(2)}}\underbrace{\frac{\partial \boldsymbol{Z}^{(2)}}{\partial \boldsymbol{X}^{(2)}}\frac{\partial \boldsymbol{X}^{(2)}}{\partial \boldsymbol{Z}^{(1)}}}_{\text{BP Error } \delta^{(1)}}\frac{\partial \boldsymbol{Z}^{(1)}}{\partial \boldsymbol{W}^{(1)}} \\
&= \underbrace{\left(\underbrace{\left(\boldsymbol{W}^{(2)\top}\underbrace{(\hat{\boldsymbol{Y}} - \boldsymbol{Y})}_{\text{BP Error } \delta^{(2)}}\right) \odot \sigma'(\boldsymbol{Z}^{(1)})}_{\text{BP Error } \delta^{(1)}}\right)}\boldsymbol{X}^{(1)\top}
\end{aligned}
\qquad \frac{\partial \mathcal{E}}{\partial \boldsymbol{b}^{(1)}} = \boldsymbol{\delta}^{(1)}\mathbf{1}
$$



  - The gradient for $\boldsymbol{W}^{(l)}$ is [BP Error at (l)] X [Input from the previous layer]
  - $\delta^{(l)}$: [BP Error at $(l+1)$] X [Weights at $(l+1)$] X [Derivative of the activation at (l)]