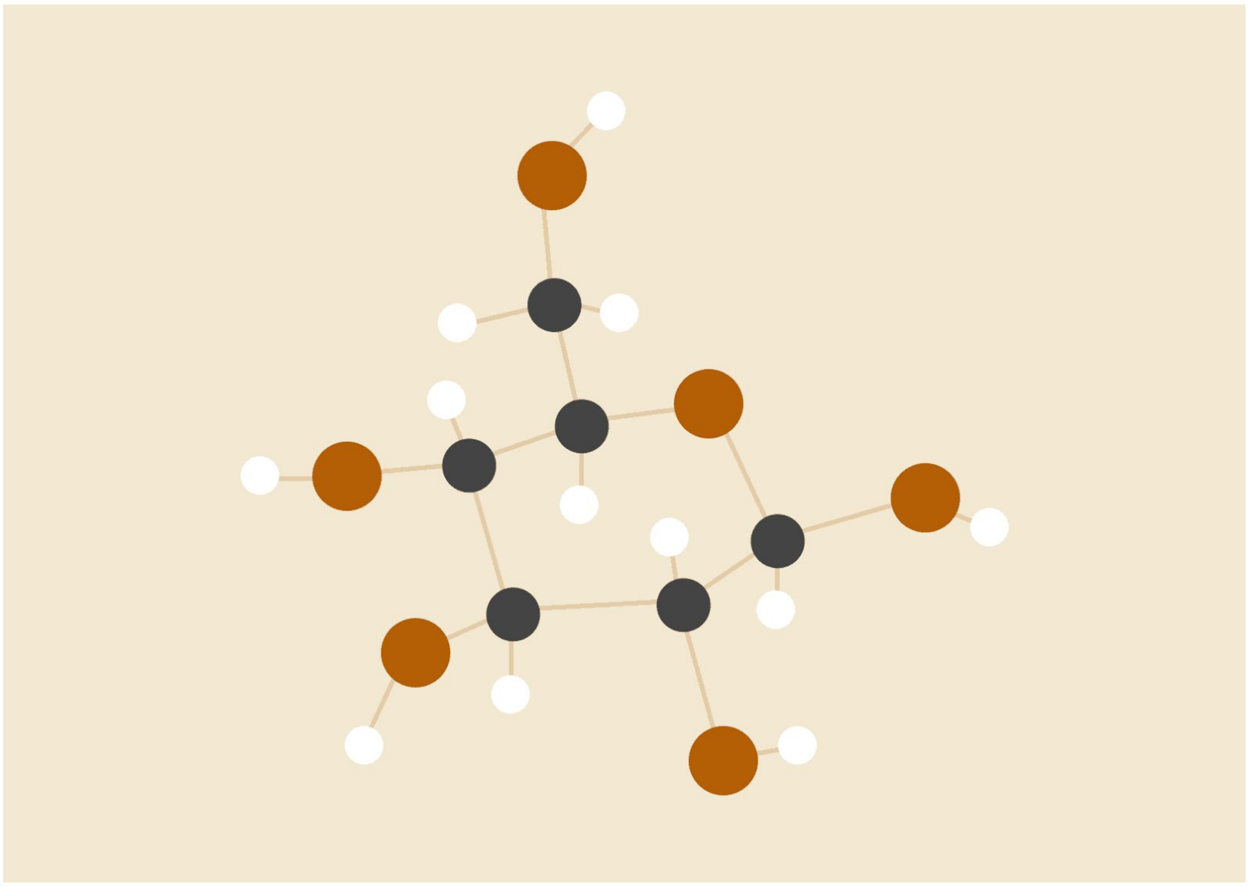
HOMEWORK 4 REPORT

Autoencoders for Image Classification

Computer Vision and Image Processing CSE 573



Jayant Solanki

UBIT Name: jayantso
Person Number: 50246821
Fall 2017 CSE

Question/Answer

Question 1: How does an autoencoder detect errors?

Answer: Autoencoder detects the error using the Loss Function:

$$E = \underbrace{\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K}\left(x_{kn} - \hat{x}_{kn}\right)^2}_{\text{mean squared error}} + \lambda * \underbrace{\Omega_{weights}}_{\substack{L_2 \\ \text{regularization}}} + \beta * \underbrace{\Omega_{sparsity}}_{\substack{\text{sparsity} \\ \text{regularization}}},$$

Figure 1.1 (courtesy MATLAB)

In above picture **x** is the original input feature vector. **x^** is the reconstructed output.

Question 2: The network starts out with 28x28 = 784 inputs. Why do subsequent layers have fewer nodes?

Answer: Subsequent layers have fewer nodes so as to force the encoder to represent the original input features into lower number of features/nodes. The main purpose of the autoencoder is to reduce the dimensions of the input features so that that reduced features can be use to non linearly model the target . Dimension reduction serves its purpose by capturing the important features and discarding the non-necessary features.

Question 3: Why are autoencoders trained one hidden layer at a time?

Answer: Training all the hidden layers of an autoencoders directly is usually difficult, however, those layers can be trained as a stack of single layer autoencoders. First step is to train the first hidden layer to reconstruct the input data, and then train the second hidden layer to reconstruct the states of the first hidden layer, and this goes on till the last layer. This is also called greedy approach. This approach trains the parameters of each layer individually while freezing parameters for the remainder layers of the **autoencoders**. For the better results, at the completion of each stage of training, fine-tuning using backpropagation can be used to improve the results by tuning the parameters of all layers are changed at the same time ad global minima can be reached.

Question 4: How were the features in Figure 3 obtained? Compare the method of

identifying features here with the method of HW1. What are a few pros and cons of these methods?

Answer: If the input features are totally random then the feature learning in the hidden layer/s will be near to impossible. If some of the input features are correlated, such as in the example data, such as the curls and the stroke patterns from the digit images, then autoencoder ends up learning those correlated and those discovered correlation ends up getting shown as the features of the hidden layer which is the low dimensional representation of the original inputs. Hence the images in the Figure 3 depicts those learned correlated data.

The general step for obtaining the output is depicted below in Figure 4.1: (courtesy slideshare)

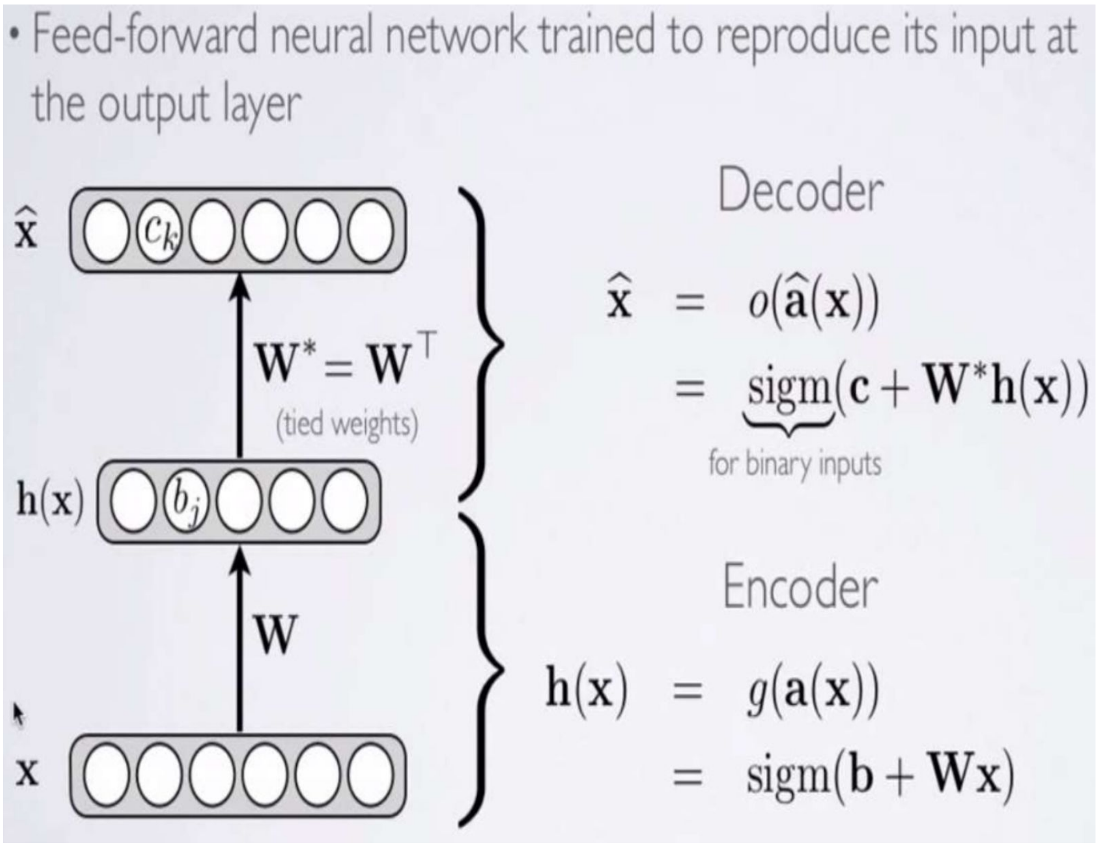

• Feed-forward neural network trained to reproduce its input at the output layer
$\hat{\mathbf{x}}$
c_k
$\mathbf{W}^* = \mathbf{W}^\top$
(tied weights)
$\mathbf{h}(\mathbf{x})$
b_j
$\mathbf{W}$
$\mathbf{x}$
Decoder
$\hat{\mathbf{x}} = o(\hat{\mathbf{a}}(\mathbf{x})) = \underbrace{\text{sigm}}_{\text{for binary inputs}}(\mathbf{c} + \mathbf{W}^*\mathbf{h}(\mathbf{x}))$
Encoder
$\mathbf{h}(\mathbf{x}) = g(\mathbf{a}(\mathbf{x})) = \text{sigm}(\mathbf{b} + \mathbf{W}\mathbf{x})$

Figure 4.1 (courtesy slideshare)

In above picture, **sigm** is sigmoid activation function. **W** is the weights, **b** is the bias, **x** is the original input feature vector. **x**^ is the reconstructed output.

The method followed in matlab for Figure 3 depiction is :

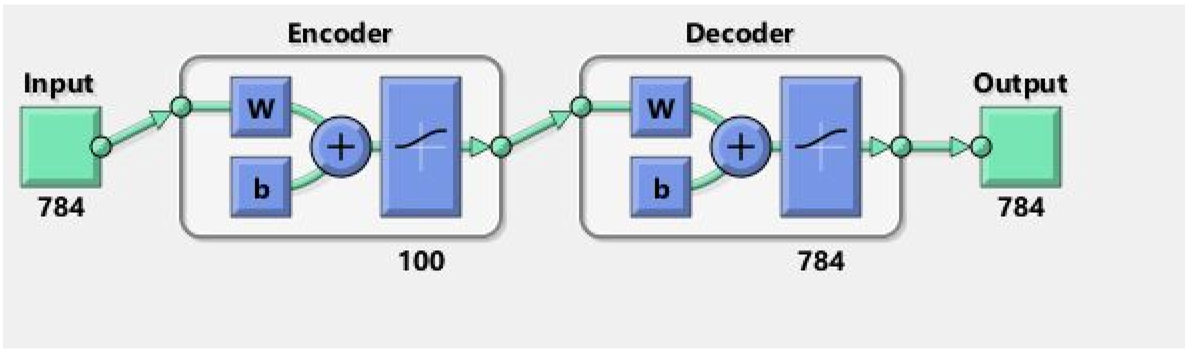


Figure 4.2 (courtesy MATLAB)

Autoencoder (HW4)	Bag-of-Words (HW1)
It is an unsupervised learning approach	It is also an unsupervised learning approach.
Relies on the presence of the correlated features such as curls and strokes.	Relies on the detection of edges using the Gaussian filters
Pixels' va;ues are standardised using the zero mean and unit standard deviation and then used directly or after the image resizing.	Pixels' values are transformed using the Gaussian filters and are used as the feature inputs.
Fails miserable in the absence of correlated features	Fails miserably in the absence of the edges.
Features are detected using the sigmoid transformation function on the weighted pixel inputs.	Visual words or features are detected using SIFT descriptors or SPM.
Using backpropagation on the transformed values the features are detected accurately.	Visual vocabulary is created using the K-means clustering using the minimised sum of Euclidean distances
Autoencoder captures irrelevant features representation along with the relevant features representation.	Feature representation capture is totally depended upon the types of the filters used on the image.
Autoencoders involve too much fiddling	One has to look for the neighbourhood

with the hyperparameters tuning, model validation and large set data processing.	edge cases, matrix dimensions of the filter map, hyperparameters tuning such as size of clusters, size of random pixels selection alpha, etc
If wrong image is provided to the autoencoder, it gets up learning wrong features which obfusticate the original learned features.	No such problem is faced
No prior data knowledge is required	No prior data knowledge is required
Costly to train and slow	Its fast.

Question 5: What does the function plotconfusion **do?**

Answer: From the Matlab documentation for the plotconfusion function: plotconfusion(targets, outputs) returns a confusion matrix plot for the target and output data in targets and outputs, respectively.

On the confusion matrix plot, the rows correspond to the predicted class **(Output Class)**, and the columns show the true class **(Target Class)**. The diagonal cells show for how many (and what percentage) of the examples the trained network correctly estimates the classes of observations. That is, it shows what percentage of the true and predicted classes match. The off diagonal cells show where the classifier has made mistakes. The column on the far right of the plot shows the accuracy for each predicted class, while the row at the bottom of the plot shows the accuracy for each true class. The cell in the bottom right of the plot shows the overall accuracy.

Question 6: What activation function is used in the hidden layers of the MATLAB tutorial?

Answer: From the Matlab documentation for the trainAutoencoder function: Logistic sigmoid Function 'logsig' is used as the default activation function for the hidden layers;

'logsig' => $f(z) = 1/(1+e^{-z})$

Question 7: In training deep networks, the ReLU activation function is generally preferred to the sigmoid activation function. Why might this be the case?

Answer: Overall ReLU gives the best train accuracy & validation accuracy.

Advantages of using ReLU are:

1. ReLUs are much simpler computationally. The forward and backward passes through an ReLU are both just a simple if statement whereas sigmoid activation requires computing an exponent. This advantage is huge when dealing with big networks with many neurons, and can significantly reduce both training and evaluation times.
2. Does not saturate (in +ve region) whereas Sigmoid activations are easier to saturate.
3. Generally models with relu neurons converge much faster than neurons with other activation functions.

Question 8: The MATLAB demo uses a random number generator to initialize the network. Why is it not a good idea to initialize a network with all zeros? How about all ones, or some other constant value? (Hint: Consider what the gradients from backpropagation will look like.)

Answer: Reasons for not initialising the network's initial weights to zero or any constants or ones are:

1. First, neural networks tend to get stuck in local minima, so it's a good idea to give them many different starting values. We can't do that if they all set to zeros.
2. Second, if the neurons start with the same weights, then all the neurons will follow the same gradient, and will always end up doing the same thing as one another during the backpropagation.
3. In each iteration of backpropagation, code will update the weights by multiplying the existing weight by a delta determined by backpropagation. If the initial weight value is 0, multiplying it by any value for delta won't change the weights.
4. Initialising them with random numbers is done to break the symmetry.
5. With different weights there's a better chance for reaching to the lowest (*or lower*) point (minima).

Question 9: Give pros and cons for both stochastic and batch gradient descent. In general, which one is faster to train in terms of number of epochs? Which one is faster in terms of number of iterations?

Answer:

Stochastic gradient descent:

Pros

1. Stochastic gradient descent (SGD) computes the gradient using a single sample of the data set.
2. SGD works better than batch gradient descent for error manifolds that have lots of local maxima/minima.
3. SGD is computationally a whole lot faster often useful where large data set cannot be loaded in RAM.
4. SGD usually does not get stuck in local minima because it is stochastic.

Cons

1. Many times SGD may result in low final accuracy .
2. Can have much more oscillation in the individual components of gradient, especially when dataset has high variance.

Batch gradient descent:

Pros

1. Batch gradient descent computes the gradient using the whole input dataset.
2. This is great for convex, or relatively smooth error manifolds.
3. Gradient move towards an optimum solution, either local or global.
4. Batch gradient descent, given an annealed learning rate, will eventually find the minimum located in its basin of attraction. (courtesy stackoverflow).

Cons

1. In practice nobody uses batch gradient descent
2. Batch gradient Descent has a problem of getting stuck in Local Minima. It has to be run exponential times in order to find global minima.

Overall SGD is faster in terms of number of iterations and BGD is faster in terms of epoch as it will reach minima in lower number of epochs.

Question 10: Try playing around with some of the parameters specified in the tutorial. Perhaps the sparsity parameters or the number of nodes; or number of layers. Report the impact of slightly modifying the parameters. Is the tutorial

presentation robust or fragile with respect to parameter settings?

Answer:

Report is tabulated below:

Sr. No	SparsityRegularization	SparsityProportion	hiddenSize1	hiddenSize2	Overall Accuracy(%)
1	4	0.1	100	50	98.5
2	8	0.1	100	50	98.7
3	12	0.1	100	50	98.4
4	4	0.1	100	50	99.3

Tutorial is robust as it is quickly converging to the global minima in lower number of epochs and producing the highest accuracy at given values of hyper parameters and hidden layers and nodes.

Code Files

DATASHEET

1.

SOFTWARE/HARDWARE USED

1. MATLAB R2017a on personal system.
2. Intel i7 8th Gen, 16GB Ram, Sublime Text 3 IDE.

REFERENCES

1. MATLAB Documentation
2. Hw34pdf
3. Stackoverflow.com
4. quora.com

=================================END===================================