

S&DS 265 / 565
Introductory Machine Learning

Autoencoders

Thursday, November 18

Yale

For today

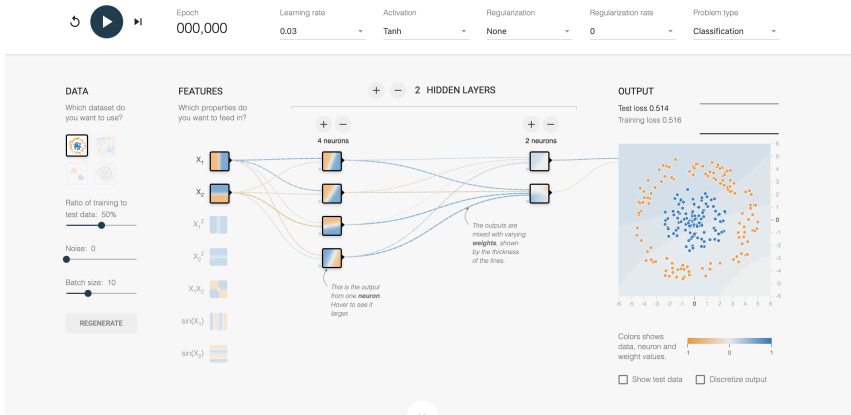
- Variants of autoencoders
- Illustration on MNIST
- Jump starting assn7

Minimal neural network: Recall

- Toy data: 3-class spirals
- First discussed a logistic regression model
- Then a simple 2-layer network, backprop calcs

These types of networks are sometimes called *multilayer perceptrons*

Interactive visualizations

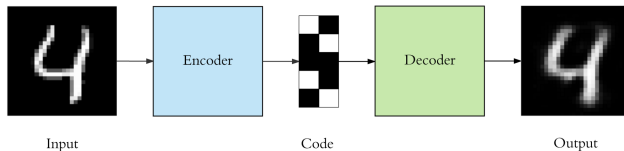


<http://playground.tensorflow.org>

Autoencoders

- Unsupervised learning methods
- Squeeze high dimensional data through a “bottleneck” of lower dimension
- Train to minimize reconstruction error

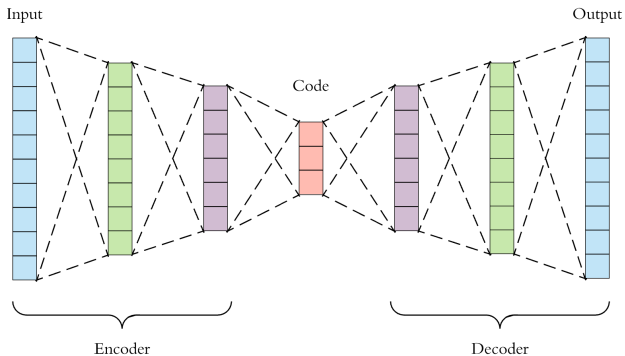
Autoencoders



Important aspects

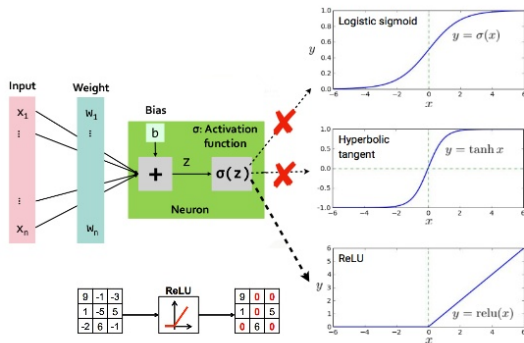
- Unsupervised: No labels used, discovers useful features of input
- Compression: Code reduces dimension of data
- Lossy: Input won't be reconstructed exactly
- Trained: The compression algorithm is learned for specific data

Deep architecture



Activation functions

Rectified Linear Unit (Relu)



71

Simple autoencoder example

Encoder network of the form

$$h = \text{ReLU}(Wx + b)$$

where $W \in \mathbb{R}^{H \times D}$ and $b \in \mathbb{R}^H$, decoder network is

$$\hat{x} = \text{ReLU}(\widetilde{W}h + \widetilde{b})$$

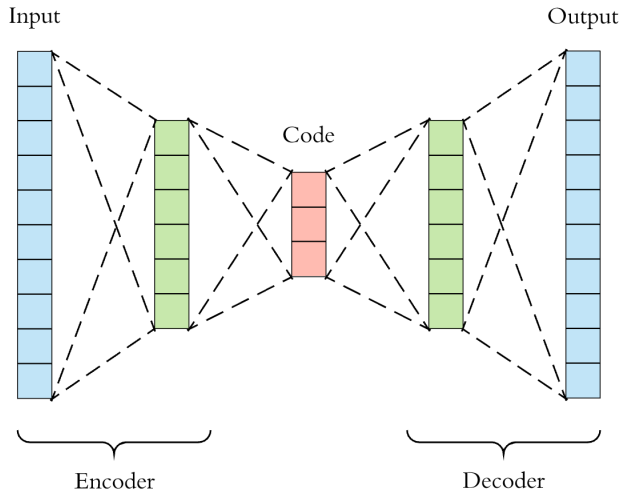
where $\widetilde{W} \in \mathbb{R}^{D \times H}$ and $\widetilde{b} \in \mathbb{R}^D$.

Objective function:

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^n \|x_i - \text{ReLU}(\widetilde{W} \text{ReLU}(Wx_i + b) + \widetilde{b})\|^2.$$

$\text{ReLU}(x) = \max(0, x)$, applied component-wise.

Simple 2-layer architecture



Simple example – code of dimension K

Encoder network of the form

$$h = \text{ReLU}(W_1 x + b_1)$$

$$c = \text{ReLU}(W_2 h + b_2)$$

where $W_1 \in \mathbb{R}^{H \times D}$ and $b_1 \in \mathbb{R}^H$, and $W_2 \in \mathbb{R}^{K \times H}$ and $b_2 \in \mathbb{R}^K$

Decoder network of the form

$$\hat{h} = \text{ReLU}(\widetilde{W}_1 c + \widetilde{b}_1)$$

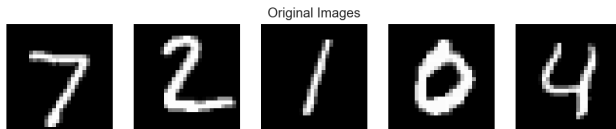
$$\hat{x} = \text{Sigmoid}(\widetilde{W}_2 \hat{h} + \widetilde{b}_2)$$

where $\widetilde{W}_1 \in \mathbb{R}^{H \times K}$ and $\widetilde{b}_1 \in \mathbb{R}^H$, and $\widetilde{W}_2 \in \mathbb{R}^{D \times H}$ and $\widetilde{b}_2 \in \mathbb{R}^D$

Objective function: binary cross-entropy

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^n (x_i \log \hat{x}_i + (1 - x_i) \log(1 - \hat{x}_i))$$

MNIST data



$$28 \times 28 = 784 = D$$

It cuts both ways...



numpy



keras

THIS IS YOUR MACHINE LEARNING SYSTEM?

YUP! YOU POUR THE DATA INTO THIS BIG
PILE OF LINEAR ALGEBRA, THEN COLLECT
THE ANSWERS ON THE OTHER SIDE.

WHAT IF THE ANSWERS ARE WRONG?

JUST STIR THE PILE UNTIL
THEY START LOOKING RIGHT.



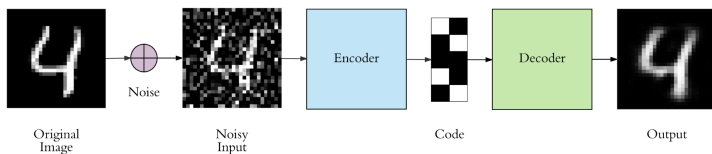
Implementation using Keras

```
1  input_size = 784
2  hidden_size = 128
3  code_size = 32
4
5  input_img = Input(shape=(input_size,))
6  hidden_1 = Dense(hidden_size, activation='relu')(input_img)
7  code = Dense(code_size, activation='relu')(hidden_1)
8  hidden_2 = Dense(hidden_size, activation='relu')(code)
9  output_img = Dense(input_size, activation='sigmoid')(hidden_2)
10
11 autoencoder = Model(input_img, output_img)
12 autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
13 autoencoder.fit(x_train, x_train, epochs=5)
```


Adam optimizer

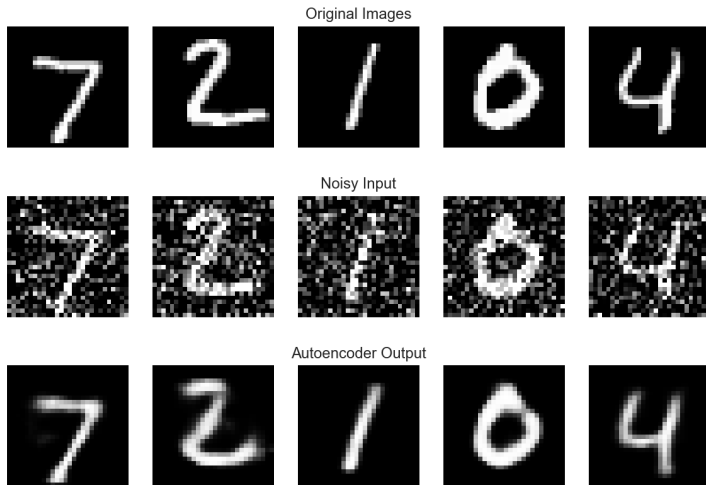
- Variant of stochastic gradient descent where separate learning rate (step size) is maintained for each network weight (parameter)
- Each step size adapted as learning progresses based on moments of the derivatives

Variant: Denoising autoencoder



- Feed in noisy data
- Train to match to denoised data

Example result on MNIST



Variant: Sparse autoencoder

- Add penalty to encourage sparsity
- Forces autoencoder to discover interesting structure in data
- Same principle as sparse coding

In Keras this is easy to do:

```
code = Dense(code_size, activation='relu',  
             activity_regularizer=l1(10e-6))(input_img)
```

Sample code

Let's take a look at the starter code `autoencoder-demo.ipynb`

Summary: What did we learn today?

- Autoencoders compress the input and then reconstruct it
- Bottleneck forces extraction of useful features
- Will overfit and “memorize” the data
- Overfitting mitigated by denoising autoencoders