S&DS 265 / 565

Introductory Machine Learning

Autoencoders

Thursday, November 18

Yale

Home stretch...

- Assignment 6 posted
- Assignment 7 (neural nets) after break
- Two more topics
- One more quiz
- Final exam, Dec 21 at 7pm

Home stretch...

11	Nov 9, 11	Introduction to neural networks	Minimal neural network Regression examples	Thu: Assn 5 in Assn6 out	Nov 9: Topic models Nov 11: Neural networks
12	Nov 16, 18	Deep neural networks	Tensorflow playground Autoencoder examples	Tue: Quiz 3	Nov 16: Neural networks (continued) Notes on backpropagation Nov 18: Autoencoders
13	Nov 19-28	No class, Thanksgiving break			
14	Nov 30, Dec 2	Reinforcement learning		Tue: Assn 6 in; Assn 7 out	
15	Dec 7, 9	Societal issues for machine learning		Tue: Quiz 4 Thu: Assn 7 in	

For today

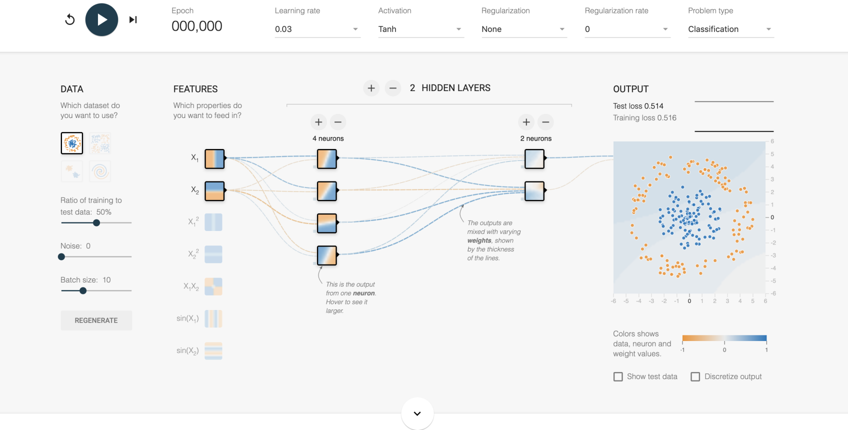
- Variants of autoencoders
- Illustration on MNIST
- Jump starting assn7

Minimal neural network: Recall

- First discussed a logistic regression model
- Then a simple 2-layer network, backprop calcs
- Toy data: 3-class spirals (TF playground and today)
- Your job: Extend starter code

These types of networks are sometimes called *multilayer perceptrons*

Interactive visualizations



http://playground.tensorflow.org

Classification

For classification we use softmax to compute probabilities

$$(p_1, p_2, p_3) = \frac{1}{e^{f_1} + e^{f_2} + e^{f_3}} \left(e^{f_1}, e^{f_2}, e^{f_3} \right)$$

The loss function is

$$\mathcal{L} = -\log P(y \mid x) = \log \left(e^{f_1} + e^{f_2} + e^{f_3} \right) - f_y$$

So, we have

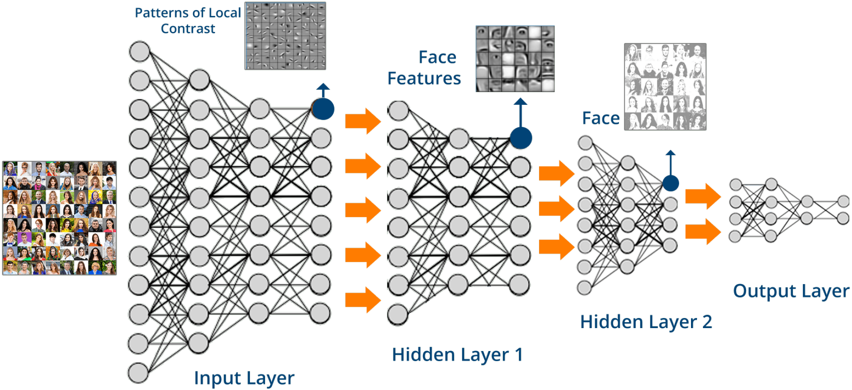
$$\frac{\partial \mathcal{L}}{\partial f_k} = p_k - \mathbb{1}(y = k)$$

Further reading: http://neuralnetworksanddeeplearning.com/ Disclaimer, I haven't "vetted" this online book.

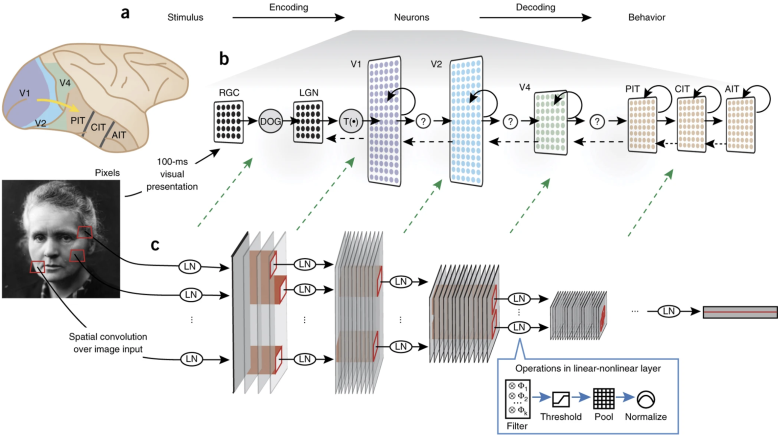
Examples

Let’s go to the spiral data notebook

Deep neural networks



Deep neural networks

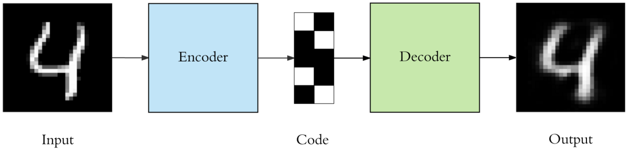


Using goal-driven deep learning models to understand sensory cortex, Yamins and DiCarlo, 2016, https://www.nature.com/articles/nn.4244

Autoencoders

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

Autoencoders

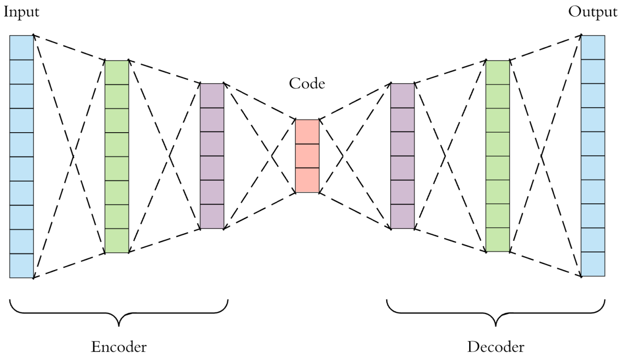


https://github.com/ardendertat/Applied-Deep-Learning-with-Keras

Important aspects

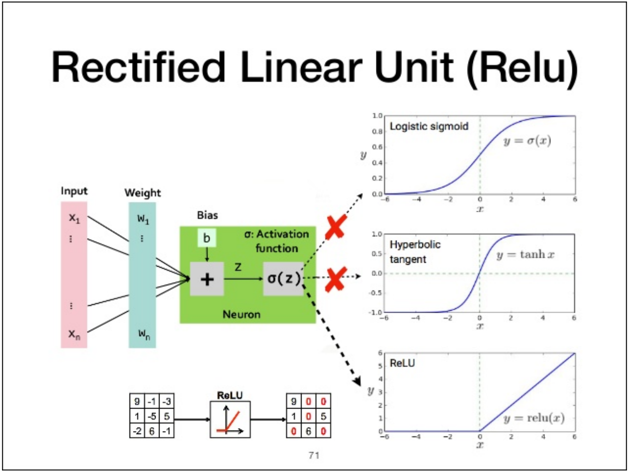
- Unsupervised: No labels used, discovers useful features of input
- Compression: Code reduces dimension of data
- Lossy: Input won't be reconstructed exactly
- Trained: The compression algorithm is learned for specific data

Deep architecture



https://github.com/ardendertat/Applied-Deep-Learning-with-Keras

Activation functions



Simple autoencoder example

Encoder network of the form

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

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

$$\widehat{x} = \mathrm{ReLU}(\widetilde{W}h + \widetilde{b})$$

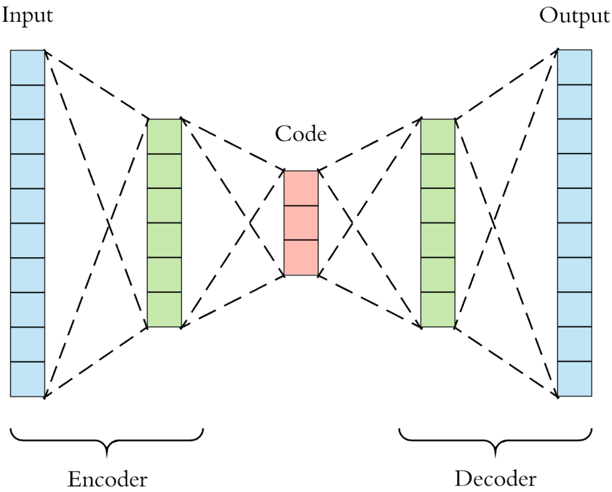
where $\widetilde{W} \in \mathbb{R}^{D \times H}$ and $\widetilde{b} \in \mathbb{R}^D$.

Objective function:

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

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

Simple 2-layer architecture



Simple example – code of dimension K

Encoder network of the form

$$h = \mathrm{ReLU}(W_1 x + b_1)$$
$$c = \mathrm{ReLU}(W_2 h + b_2)$$

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

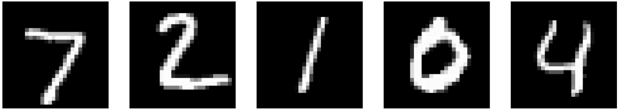
$$\widehat{h} = \mathrm{ReLU}(\widetilde{W}_1 c + \widetilde{b}_1)$$
$$\widehat{x} = \mathrm{Sigmoid}(\widetilde{W}_2 \widehat{h} + \widetilde{b}_2)$$

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

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

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.
DATA
ANSWERS

Implementation using Keras

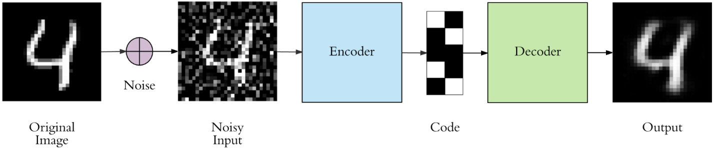
```
input_size = 784
hidden_size = 128
code_size = 32

input_img = Input(shape=(input_size,))
hidden_1 = Dense(hidden_size, activation='relu')(input_img)
code = Dense(code_size, activation='relu')(hidden_1)
hidden_2 = Dense(hidden_size, activation='relu')(code)
output_img = Dense(input_size, activation='sigmoid')(hidden_2)

autoencoder = Model(input_img, output_img)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
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

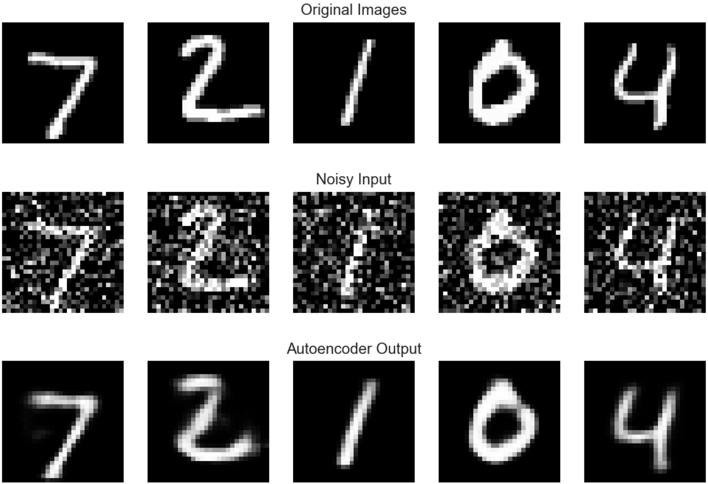
"Adam: A method for stochastic optimization," D. Kingma and J. Ba, `https://arxiv.org/abs/1412.6980`

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

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