S&DS 265 / 565

Introductory Machine Learning

Autoencoders

November 17

Yale

Home stretch...

- Assignment 5 due Dec 1
- Assignment 6 (last!): neural nets and reinforcement learning
- Quiz 5 posted today after class; 48 hours, 20 minutes
- Final exam, Dec 19 at 7pm, Davies Aud

Home stretch...

12	Nov 15, 17	Deep neural networks	Tensorflow playground Autoencoder examples	Nov 15: Neural networks (continued) Nov 17: Autoencoders	ISL Section 10.7 Notes on backpropagation	Thu: Quiz 5
13	Nov 22, 24	No class, Thanksgiving break				
14	Nov 29, Dec 1	Reinforcement learning	Q-learning	Nov 29: Reinforcement learning Dec 1: Deep reinforcement learning		Thu: Assn 5 in Assn 6 out
15	Dec 6, 8	Societal issues for machine learning		Dec 6: Societal issues Dec 8: Course wrap up		Thu: Quiz 6
16	Dec 15					Thu: Assn 6 in
17	Mon, Dec 19, 7pm, Davies Aud	Final exam			Registrar: Final exam schedule Practice final	

For today

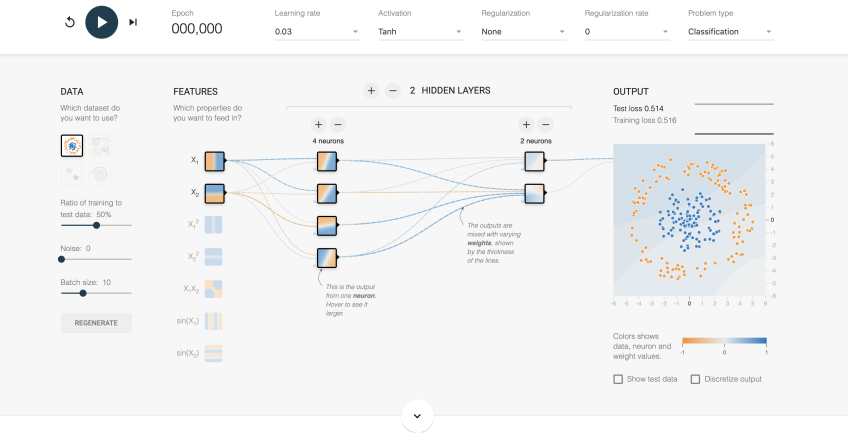
- Variants of autoencoders
- Illustration on MNIST

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: From batch to SGD

These types of networks are sometimes called *multilayer perceptrons*

Interactive visualizations



http://playground.tensorflow.org

Backprop calcs

We’ll go through some of the backpropagation calculations. It’s not essential that you can repeat all of these (but try!)

Start simple

Loss is

$$\mathcal{L} = \frac{1}{2}(y - f)^2$$

The change in loss due to making a small change in output f is

$$\frac{\partial \mathcal{L}}{\partial f} = (f - y)$$

We now send this backward through the network

Start simple

Loss is

$$\mathcal{L} = \frac{1}{2}(y - f)^2$$

Now suppose that $f = ab$:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial a} &= \frac{\partial \mathcal{L}}{\partial f}\frac{\partial f}{\partial a} \\ &= \frac{\partial \mathcal{L}}{\partial f} \cdot b \\ &= (f - y) \cdot b\end{aligned}$$

Start simple

Loss is

$$\mathcal{L} = \frac{1}{2}(y - f)^2$$

Now suppose that $f = ab$:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial b} &= \frac{\partial f}{\partial b}\frac{\partial \mathcal{L}}{\partial f} \\ &= a \cdot \frac{\partial \mathcal{L}}{\partial f} \\ &= a \cdot (f - y)\end{aligned}$$

Fancy verison

We need a matrix version of this. If $A = BC$, then

$$\frac{\partial \mathcal{L}}{\partial B} = \frac{\partial \mathcal{L}}{\partial A} \, C^T$$

$$\frac{\partial \mathcal{L}}{\partial C} = B^T \, \frac{\partial \mathcal{L}}{\partial A}$$

Check that the dimensions match up!

Example

So if $f = Wx + b$ then

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial W} &= \frac{\partial \mathcal{L}}{\partial f}\ x^T \\ &= (f - y)\ x^T \end{aligned}$$

Example

So if $f = Wx + b$ then

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial b} &= \frac{\partial \mathcal{L}}{\partial f} \\ &= (f - y) \end{aligned}$$

Two layers

Now add a layer:

$$f = W_2 h + b_2$$
$$h = W_1 x + b_1$$

Then we have

$$\frac{\partial \mathcal{L}}{\partial W_2} = \frac{\partial \mathcal{L}}{\partial f} \; h^T$$
$$= (f - y) \; h^T$$

$$\frac{\partial \mathcal{L}}{\partial h} = W_2^T \; \frac{\partial \mathcal{L}}{\partial f}$$
$$= W_2^T \; (f - y)$$

Two layers

Now send this back (backpropagate) to the first layer:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial W_1} &= \frac{\partial \mathcal{L}}{\partial h}\, x^T \\ &= W_2^T\, \frac{\partial \mathcal{L}}{\partial f}\, x^T \\ &= W_2^T\, (f - y)\, x^T\end{aligned}$$

Adding a nonlinearity

Remember, this just gives a linear model! Need a nonlinearity:

$$h = \varphi(W_1 x + b_1)$$

$$f = W_1 h + b_2$$

Adding a nonlinearity

If $\varphi(u) = relu(u) = \max(u, 0)$ then this just becomes

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial W_1} &= \mathbb{1}(h > 0)\, \frac{\partial \mathcal{L}}{\partial h}\, x^T \\
&= \mathbb{1}(h > 0)\, W_2^T\, \frac{\partial \mathcal{L}}{\partial f}\, x^T \\
&= \mathbb{1}(h > 0)\, W_2^T\, (f - y)\, x^T
\end{aligned}$$

where

$$\mathbb{1}(u) = \begin{cases} 1 & u > 0 \\ 0 & \text{otherwise} \end{cases}$$

See notes on backpropagation for details

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

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

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

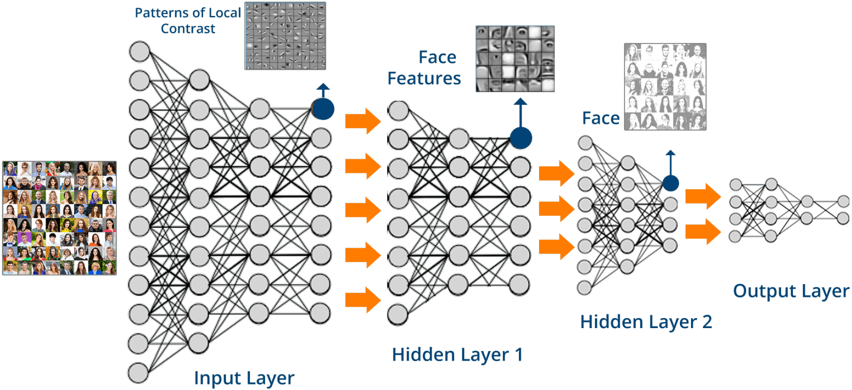
$$\mathcal{L} = -\log P(y \mid x) = \log \left(e^{f_1} + e^{f_2} + e^{f_3} \right) - f_y$$

So, we have

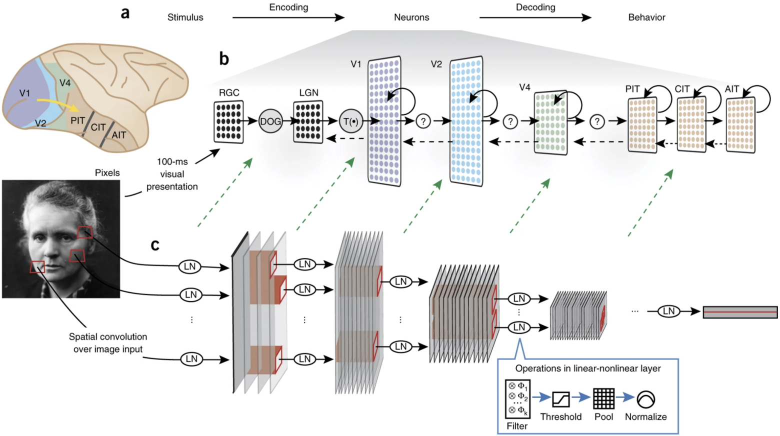
$$\frac{\partial \mathcal{L}}{\partial f_k} = p_k - \mathbb{1}(y = k)$$

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

Deep neural networks



Deep neural networks

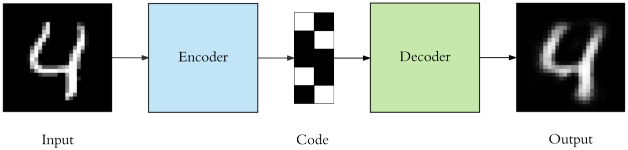


Using goal-driven deep learning models to understand sensory cortex, Yamins and DiCarlo, 2016, `https://www.nature.com/articles/nn.4244`

Autoencoders

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

Autoencoders

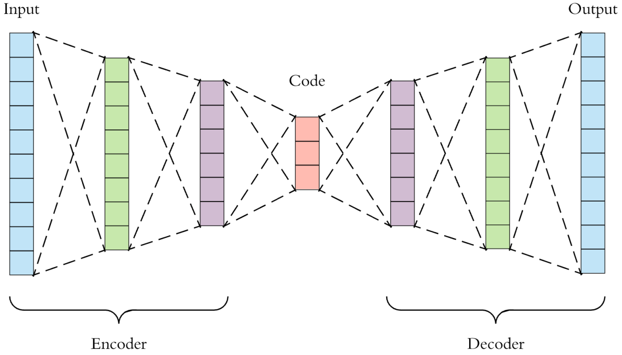


https://github.com/ardendertat/Applied-Deep-Learning-with-Keras

Important aspects

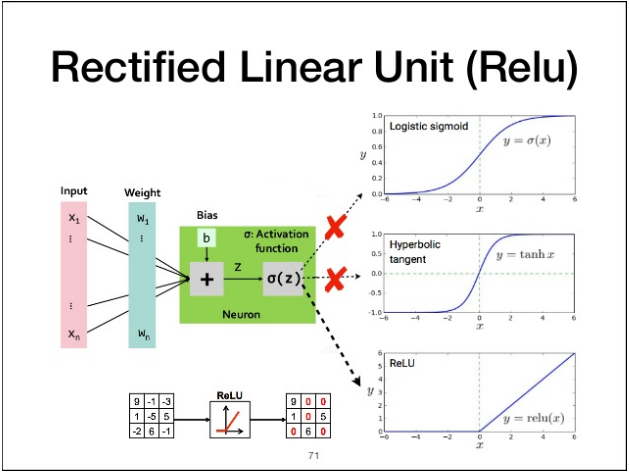
- Unsupervised: No labels used, discovers useful features of input
- Compression: Code reduces dimension of data
- Lossy: Input won't be reconstructed exactly
- Trained: The compression algorithm is learned for specific data

Deep architecture



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

Activation functions



Simple autoencoder example: Single hidden layer

Encoder network of the form

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

decoder network is

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

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 autoencoder example: Single hidden layer

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

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

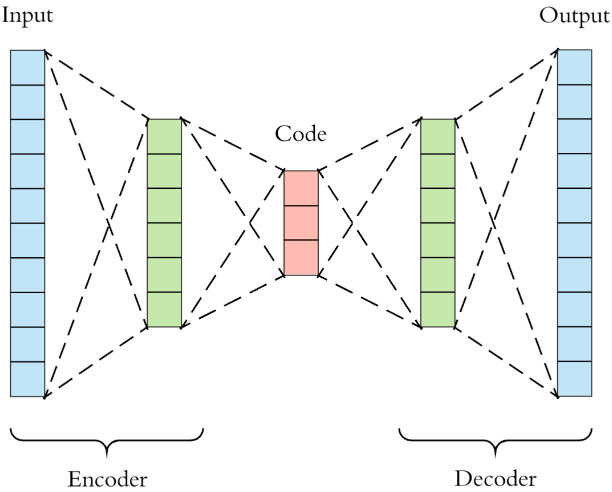
where $\widetilde{W} \in \mathbb{R}^{D \times H}$ and $\widetilde{b} \in \mathbb{R}^{D}$.

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.

2-layer architecture



2-layer architecture: Code of dimension K

Encoder network of the form

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

Decoder network of the form

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

Objective function: binary cross-entropy

$$
\mathcal{L} = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{d} \left(x_{ij} \log \widehat{x}_{ij} + (1 - x_{ij}) \log(1 - \widehat{x}_{ij}) \right)
$$

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

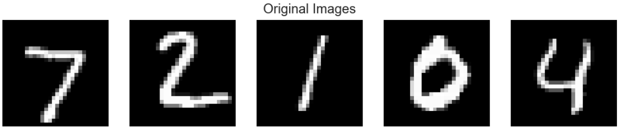
$$\widehat{h} = \mathrm{ReLU}(\widetilde{W}_1 c + \widetilde{b}_1)$$
$$\widehat{x} = \mathrm{Sigmoid}(\widetilde{W}_2 \widehat{h} + \widetilde{b}_2)$$

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

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{d}\left(x_{ij}\log\widehat{x}_{ij} + (1 - x_{ij})\log(1 - \widehat{x}_{ij})\right)$$

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

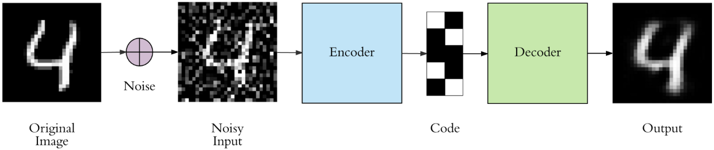
```
input_size = 784
hidden_size = 128
code_size = 32

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

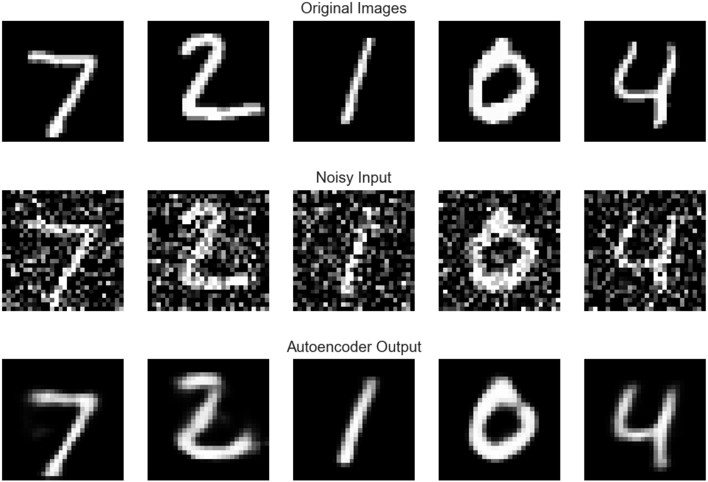
"Adam: A method for stochastic optimization," D. Kingma and J. Ba, `https://arxiv.org/abs/1412.6980`

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`

Comparison

Let's consider autoencoders in comparison to other methods:

- PCA
- Bayesian inference
- Topic models
- ...Variational autoencoders (IML)

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