

Neural Networks

Carlos Carvalho, Mladen Kolar and Robert McCulloch

10/21/2015

BASED ON YOUR
INTERNET HISTORY,
YOU MIGHT BE DUMB
ENOUGH TO ENJOY
EXTREME SPORTS.



Dilbert.com DilbertCartoonist@gmail.com

CLICK HERE TO BUY A
TICKET TO BASE JUMP
FROM THE INTERNA-
TIONAL SPACE STATION.



2-2-13 ©2013 Scott Adams, Inc./Dist. by Universal Uclick

I THINK
THE INTER-
NET IS
TRYING TO
KILL ME.



WE
CALL IT
"MACHINE
LEARNING."



Neural networks

Our learning algorithms so far:

Training data: $(x_i, y_i)_{i=1}^n \rightarrow \text{Machine Learning} \rightarrow y = \hat{f}(x)$

All of the procedures directly work on input features.

What if the input features are not informative?

Neural networks

Feature engineering — handcrafting transformations

Training data: $(x_i, y_i)_{i=1}^n \rightarrow \Phi \rightarrow (\Phi_x(x_i), \Phi_y(y_i))_{i=1}^n$

Here Φ is designed by a human.

$(\Phi_x(x_i), \Phi_y(y_i))_{i=1}^n \rightarrow \text{Machine Learning} \rightarrow \Phi_y(y) = \hat{f}(\Phi_x(x))$

This process is expensive and time consuming.

Example: Handwritten Digit Recognition

$$P(y=2 | \text{2}, b)$$

$$P(y=9 | \text{9}, b)$$

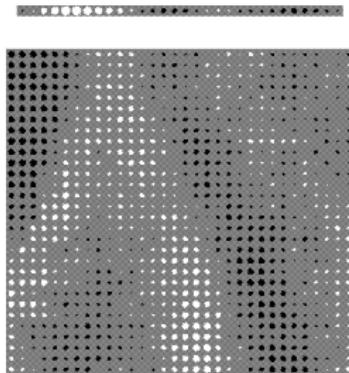
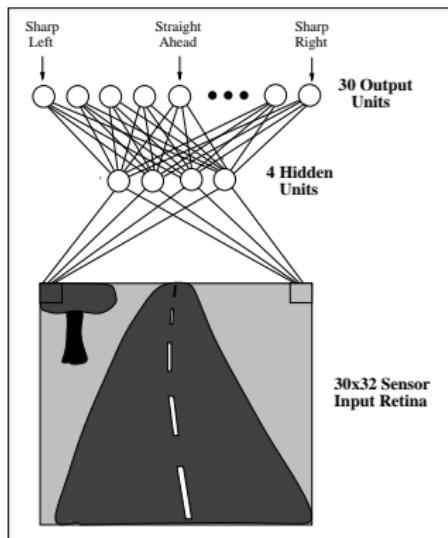
How to represent image?

How informative is each pixel?

Logistic regression trained on pixel values gives ~90% accuracy.

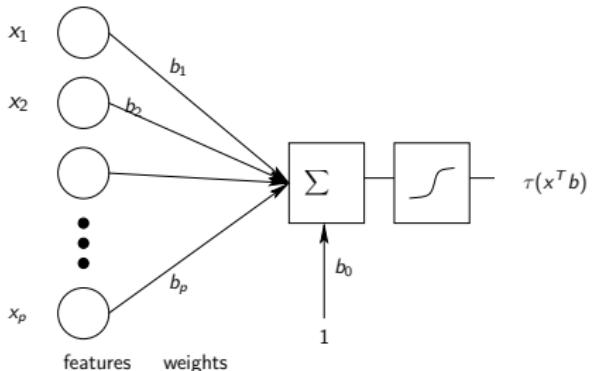
Example: ALVINN

Autonomous Land Vehicle In a Neural Network

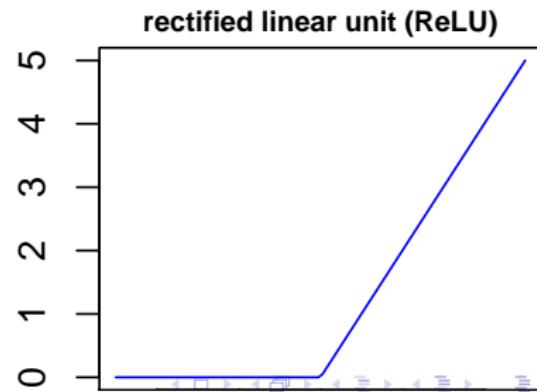
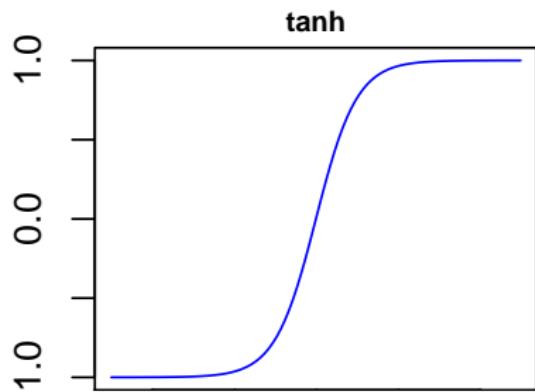


Video: <http://watson.latech.edu/book/intelligence/intelligenceOverview5b4.html>

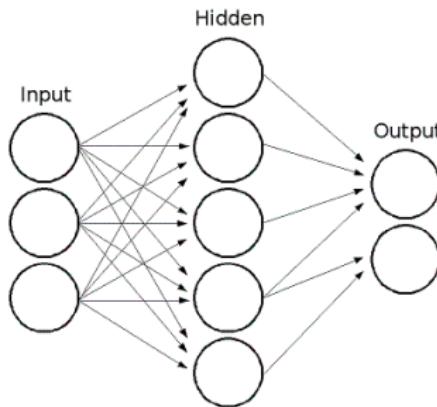
Model of a neuron



Other nonlinear activations



Multilayer Perceptron



2 Layers of Neurons

- ▶ 1st layer takes input x
- ▶ 2nd layer takes output of 1st layer
- ▶ The last layer is the output

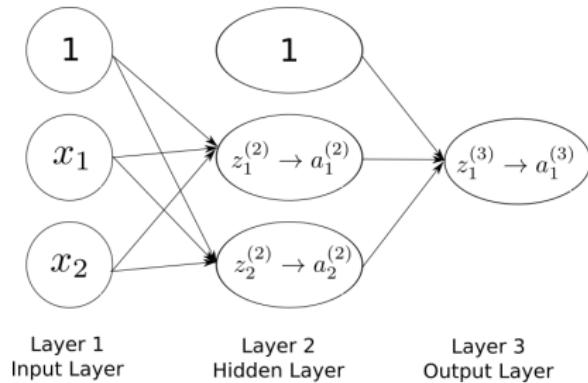
Multilayer Perceptron

The activities of the neurons in each layer are a non-linear function of the activities in the layer below.

Can approximate arbitrary functions

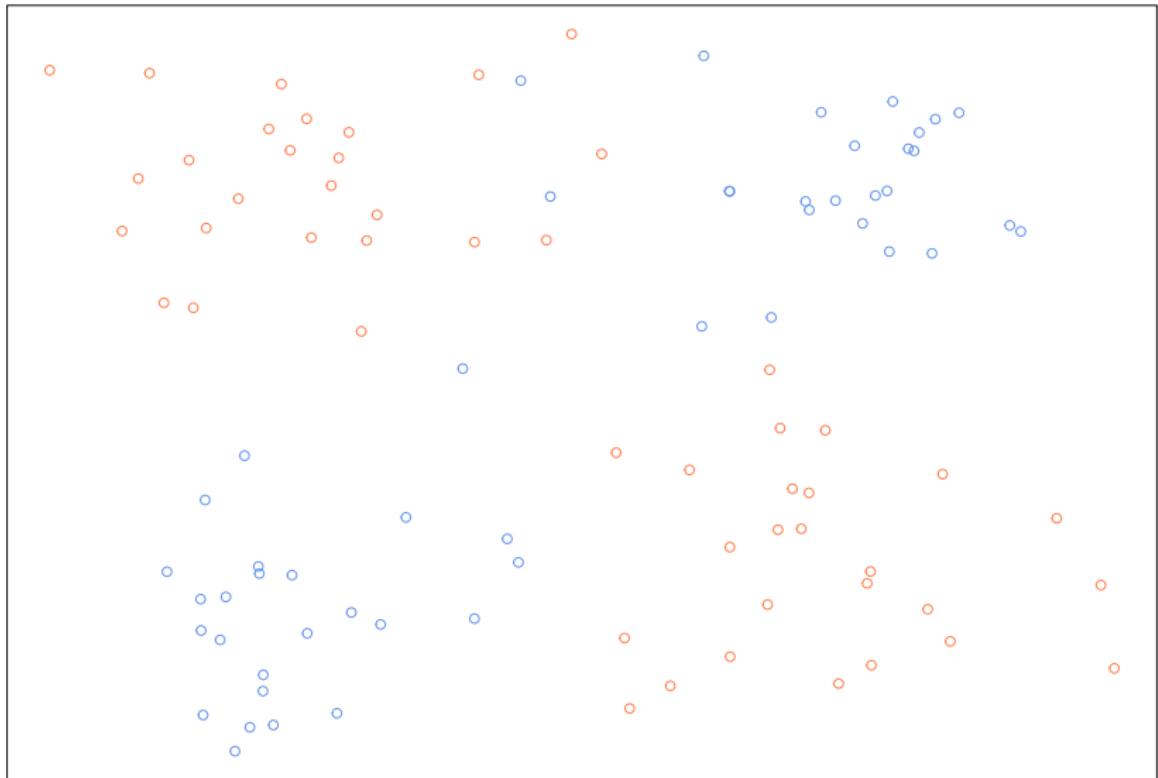
- ▶ Provided hidden layer is large enough
- ▶ “fat” 2-layer network

1 hidden layer details

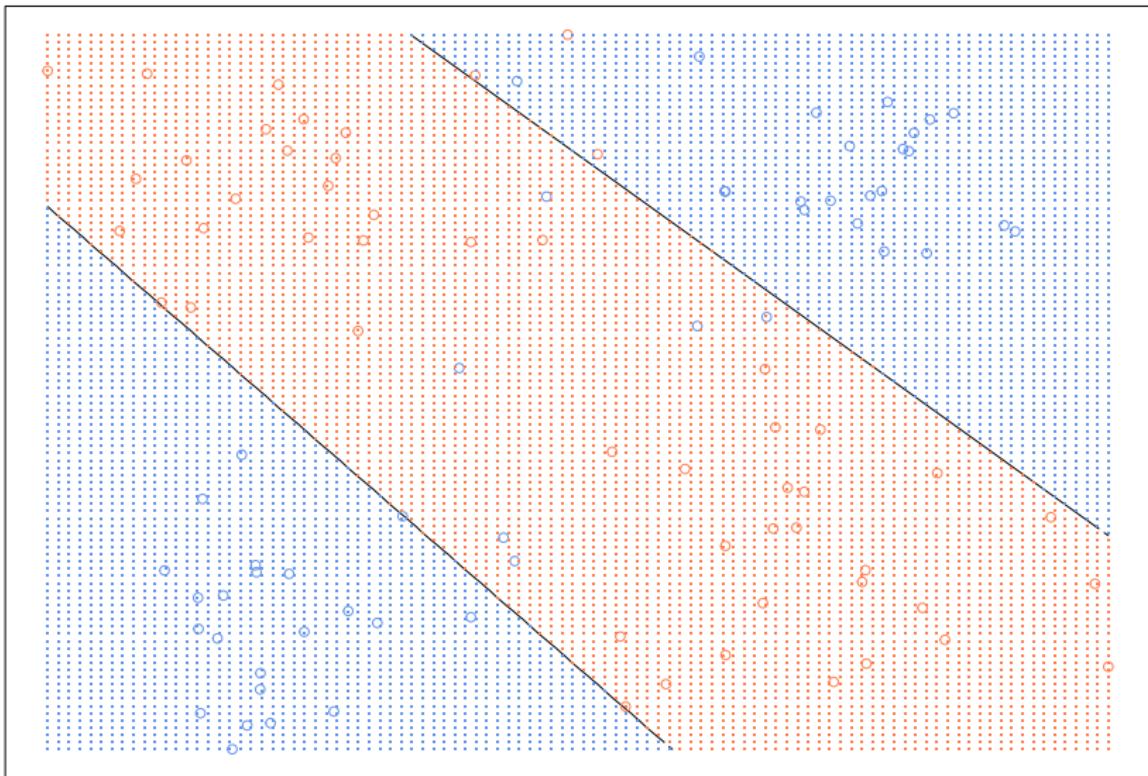


$$\begin{aligned} z_1^{(2)} &= b_{10}^{(1)}x_0 + b_{11}^{(1)}x_1 + b_{12}^{(1)}x_2 &\longrightarrow a_1^{(2)} &= g(z_1^{(2)}) \\ z_2^{(2)} &= b_{20}^{(1)}x_0 + b_{21}^{(1)}x_1 + b_{22}^{(1)}x_2 &\longrightarrow a_1^{(2)} &= g(z_1^{(2)}) \\ z_1^{(3)} &= b_{10}^{(2)}a_0^{(2)} + b_{11}^{(2)}a_1^{(2)} + b_{12}^{(2)}a_2^{(2)} &\longrightarrow a_1^{(3)} &= g(z_1^{(3)}) \end{aligned}$$

Example: Simulated XOR



neural network -- 1 hidden layer with 2 neurons



Weights in hidden layer

```
h2o.biases(model, vector_id = 1)
```

```
##          C1
## 1 16.524170
## 2 2.476905
```

```
h2o.weights(model, matrix_id = 1)
```

```
##          x.1      x.2
## 1 14.903464 12.896398
## 2 -1.927662 -1.998819
```

Feature transformation

```
tmp.df = as.h2o(data.frame(x.1=c(-1, -1, 1, 1),
                            x.2=c(-1, 1, -1, 1)))

trans.features = h2o.deepfeatures(model, tmp.df, layer = 1)
as.matrix( h2o.cbind(tmp.df, trans.features) )

##      x.1 x.2   DF.L1.C1 DF.L1.C2
## [1,] -1 -1 -0.9999884      -1
## [2,] -1  1 -0.9864362       1
## [3,]  1 -1 -0.9877517       1
## [4,]  1  1  0.7584453       1
```

Weights in output layer

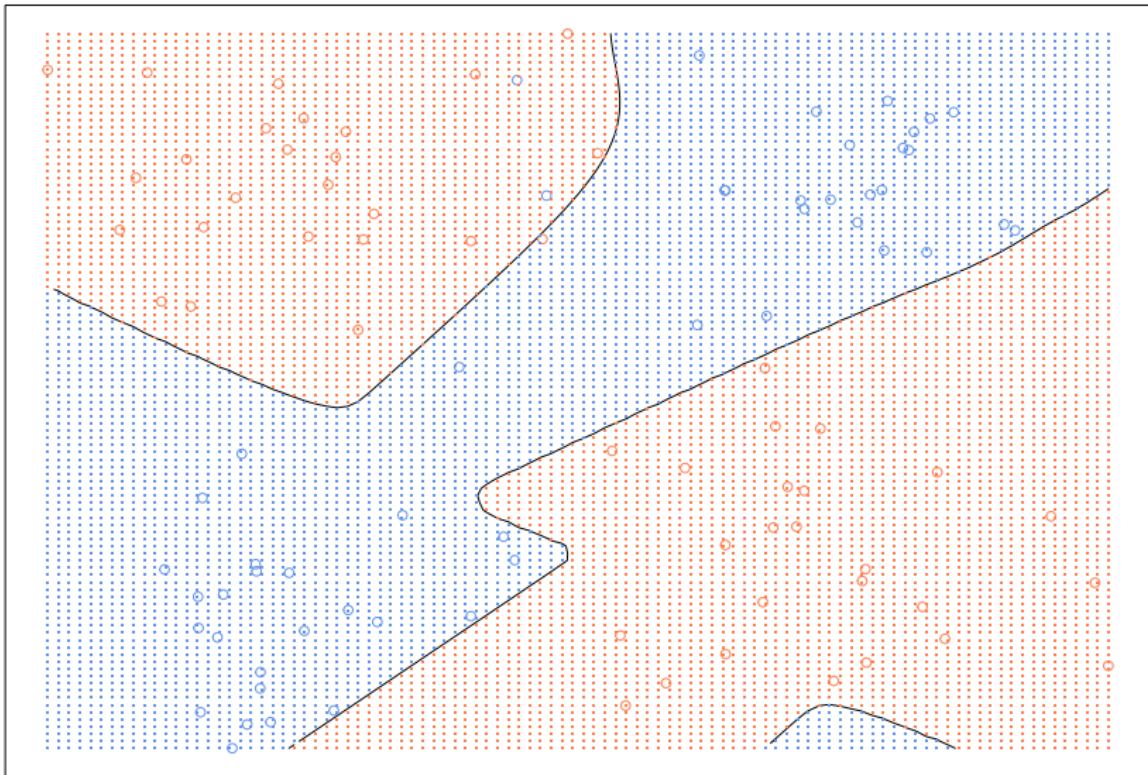
```
h2o.biases(model, vector_id = 2)
```

```
##          C1
## 1  2.978030
## 2 -2.376198
```

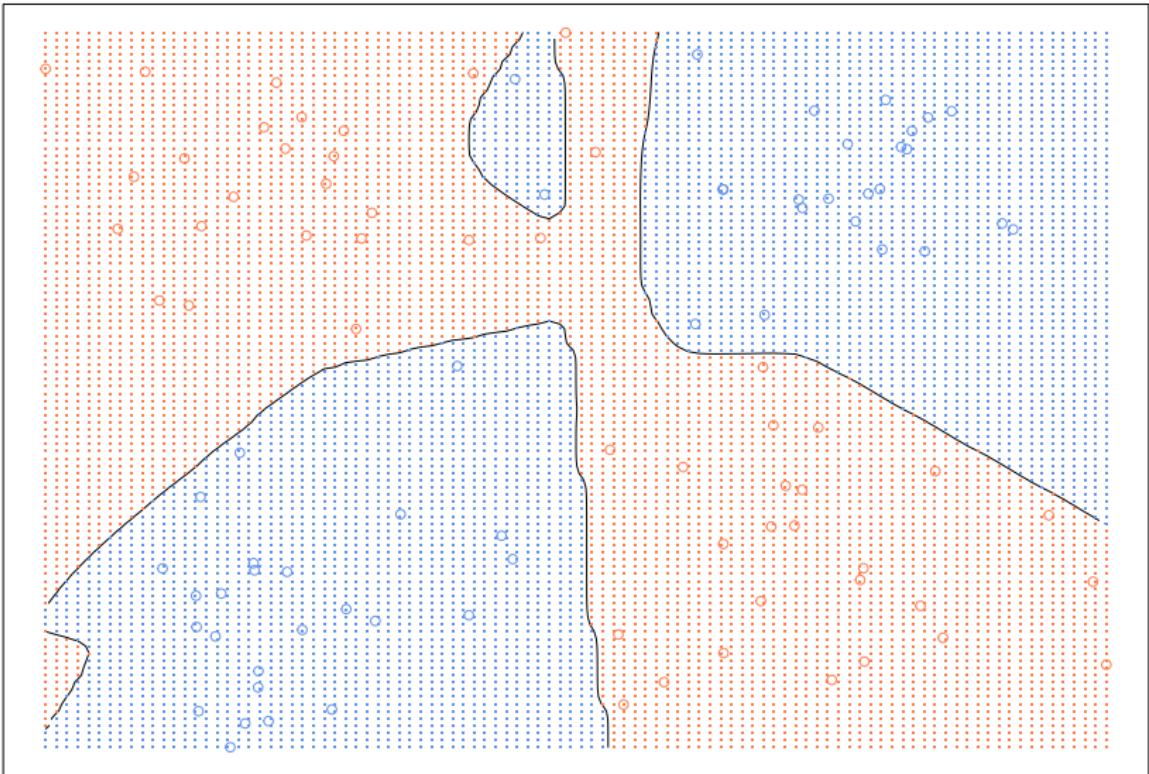
```
h2o.weights(model, matrix_id = 2)
```

```
##          C1          C2
## 1 -1.339904 -1.936511
## 2 -4.520841  2.660101
```

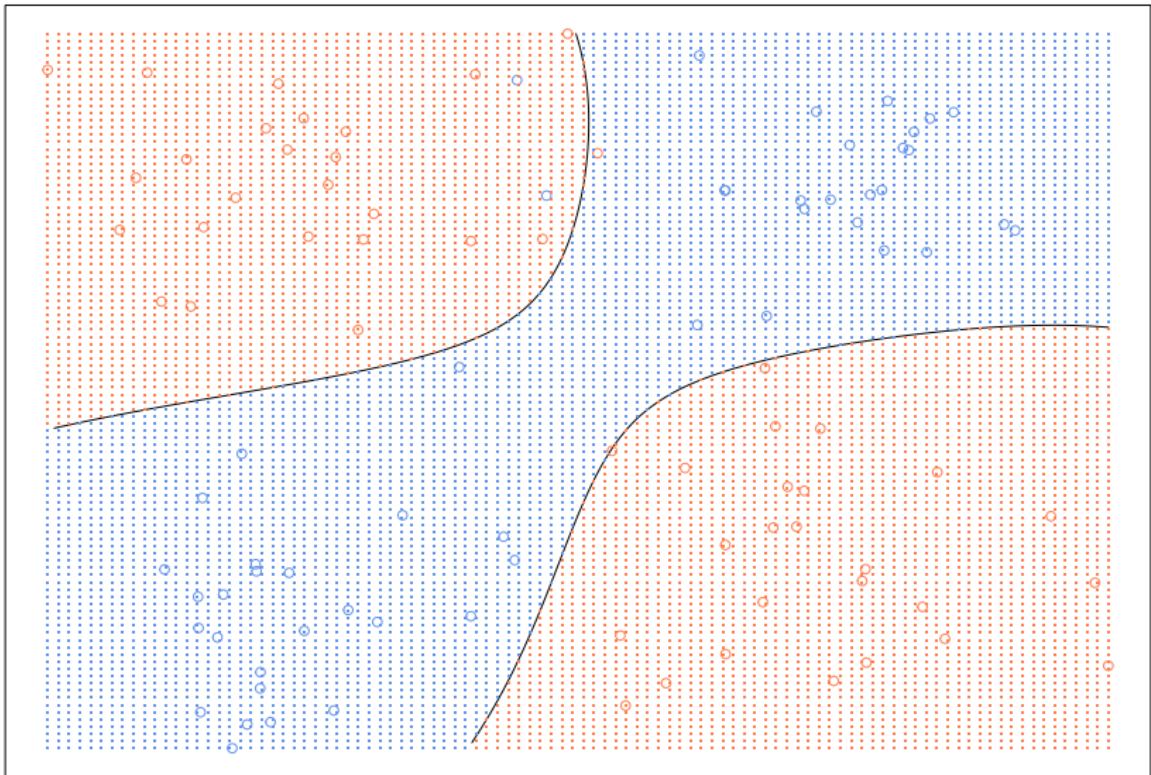
neural network -- 1 hidden layer with 5 neurons



neural network -- 1 hidden layer with 10 neurons



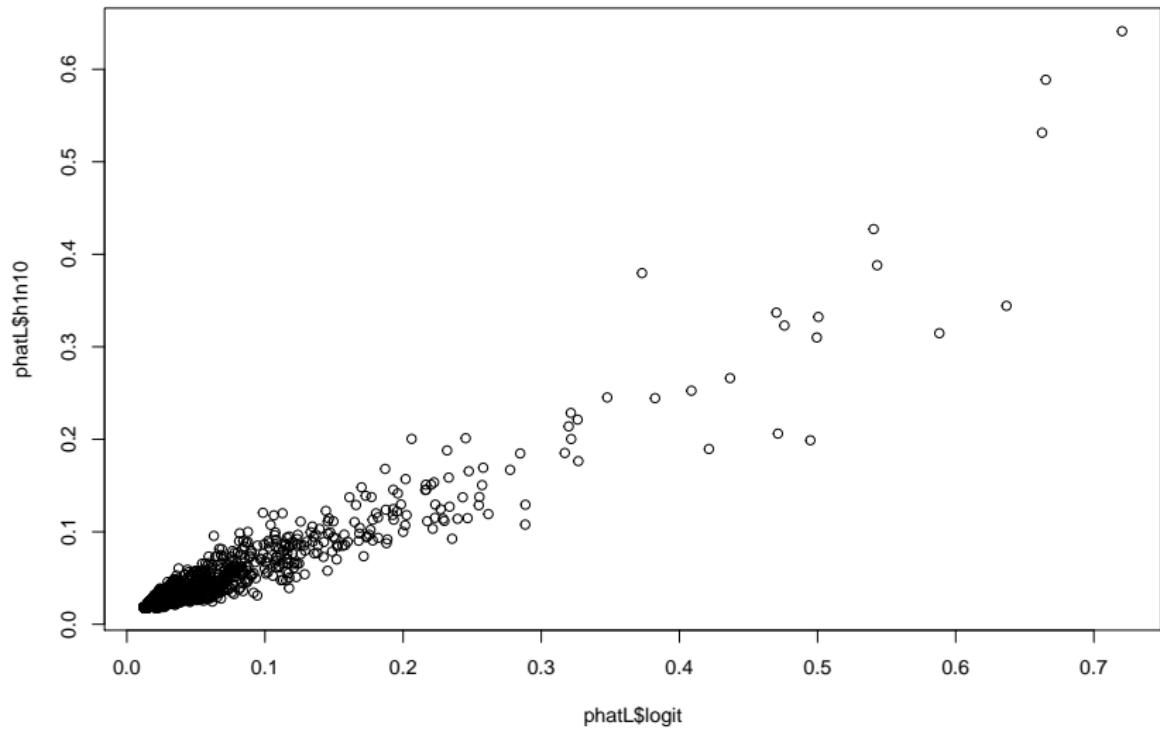
neural network -- 1 hidden layer with 10 neurons with regularization

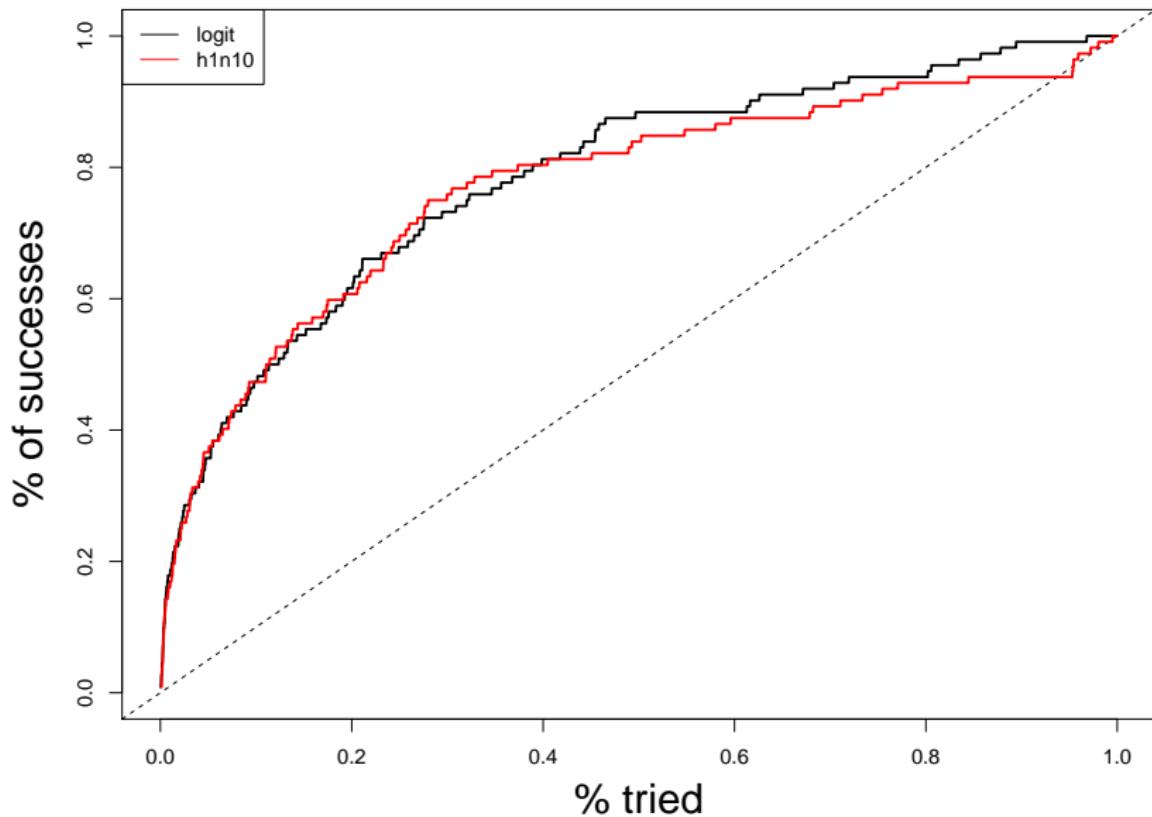


Example: Tabloid data

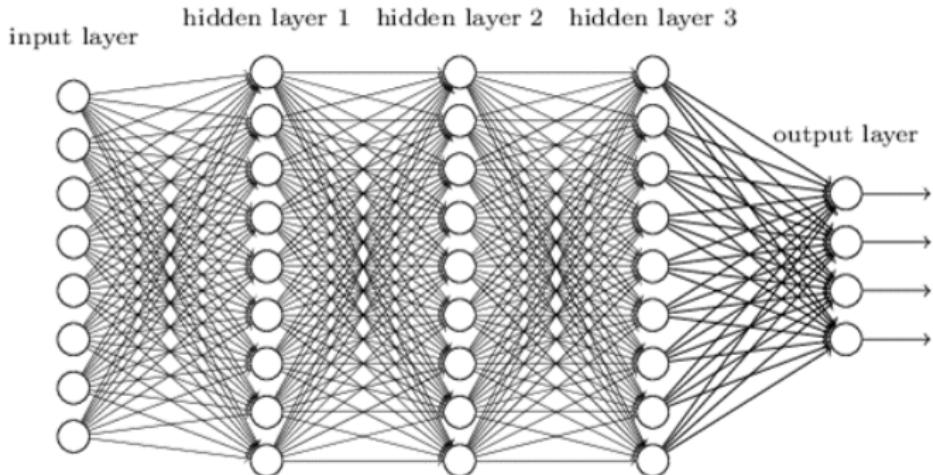
See *tabloid.h2o.R*

For example of how to use **nnet** package, see *tabloid.nnet.R*



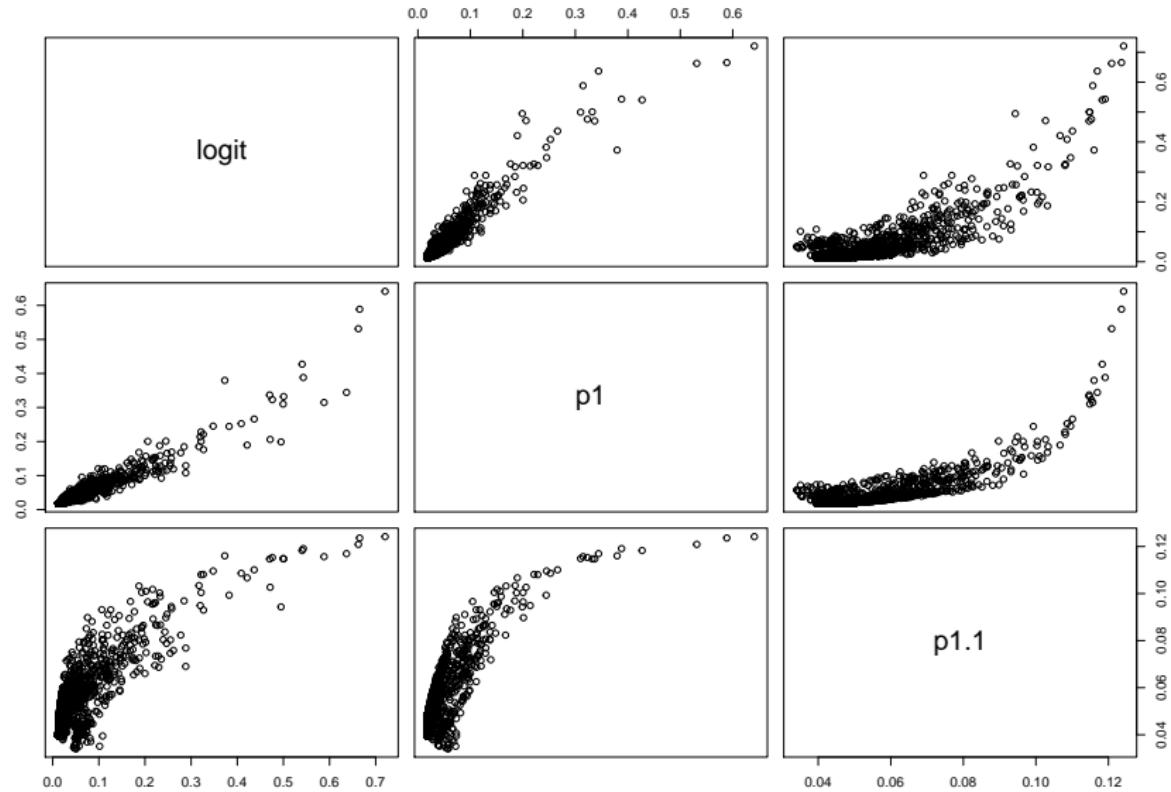


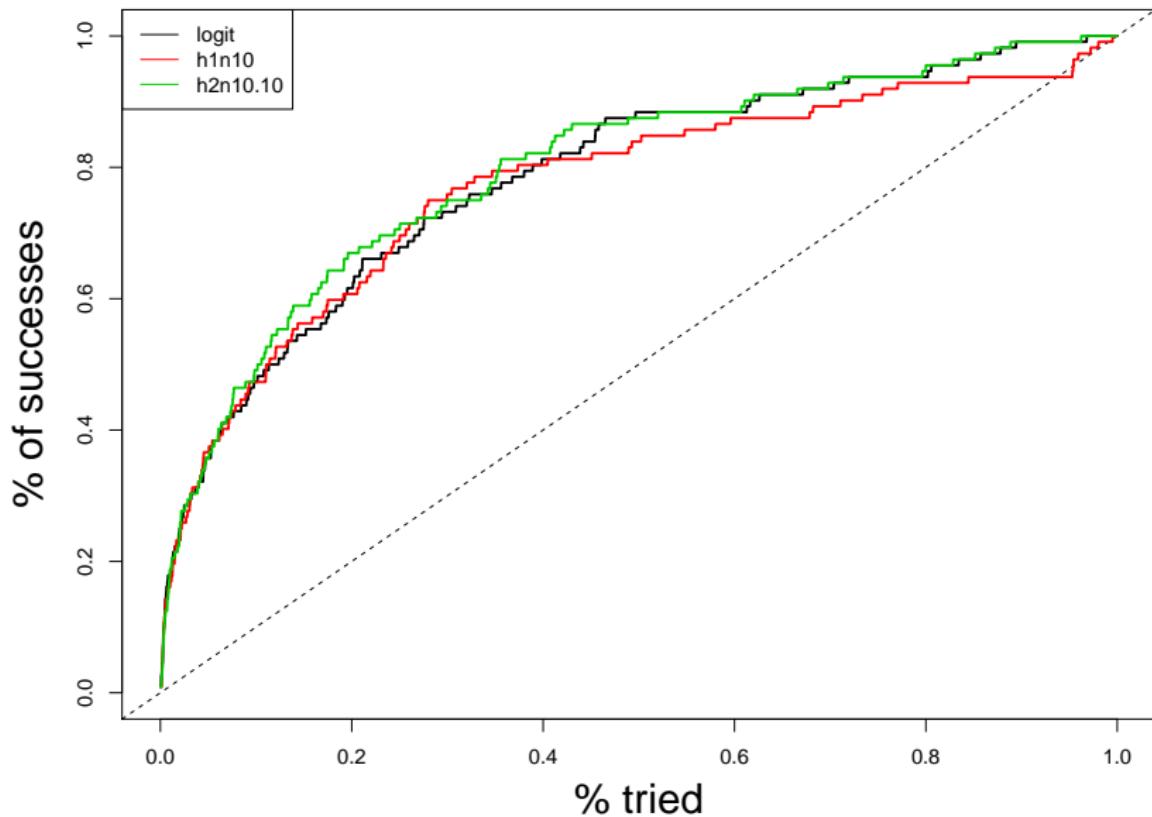
Deep neural network



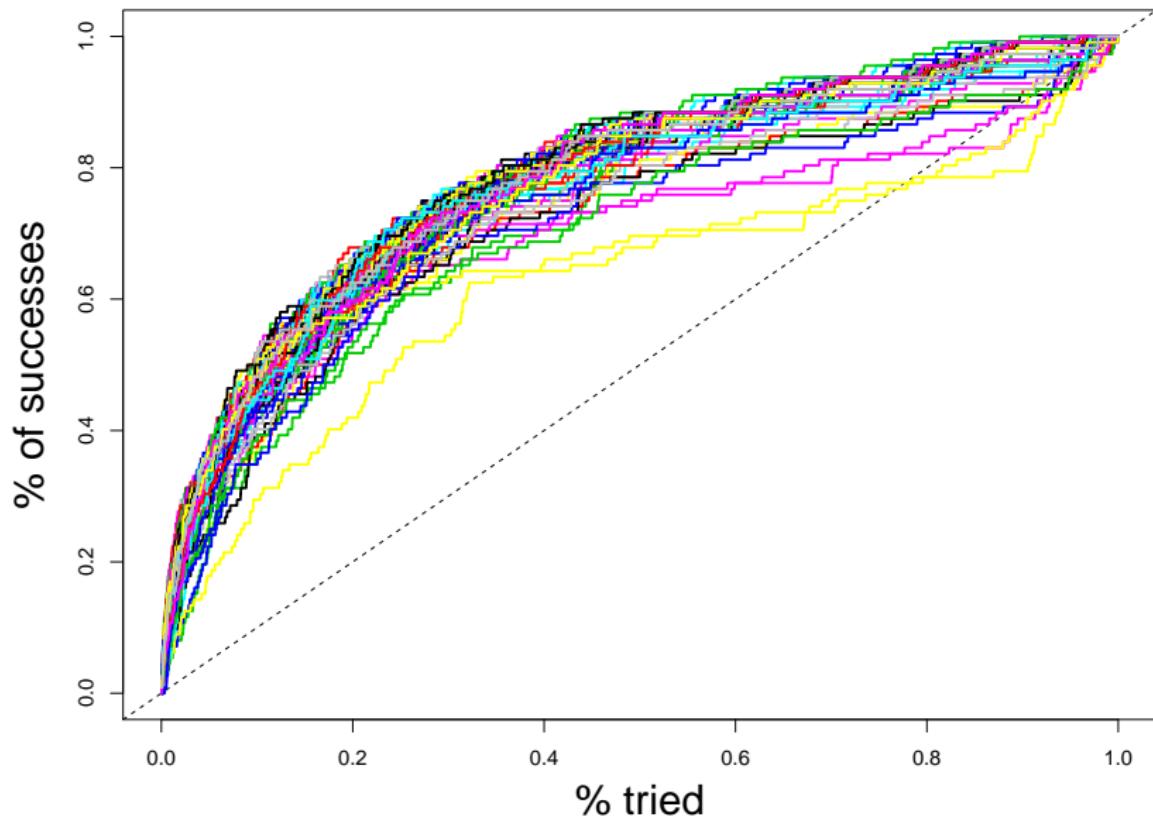
If there is more than one hidden layer, networks are called “deep” neural networks.

Tabloid again





Grid search



Fitting neural networks

Gradient descent + chain rule + lot of tricks

- ▶ We will not provide details
- ▶ The procedure is called backpropagation

Difficult to train because there are many local minima

- ▶ Train multiple nets with different initial weights
- ▶ Initialize weights near zero
- ▶ Therefore, initial networks near-linear
- ▶ Increasingly non-linear functions possible as training progresses

Fitting neural networks

Adaptive Learning Rate

- ▶ Automatically set learning rate for each neuron based on its training history
- ▶ ADADELTA:
<http://www.matthewzeiler.com/pubs/googleTR2012/googleTR2012.pdf>

Momentum

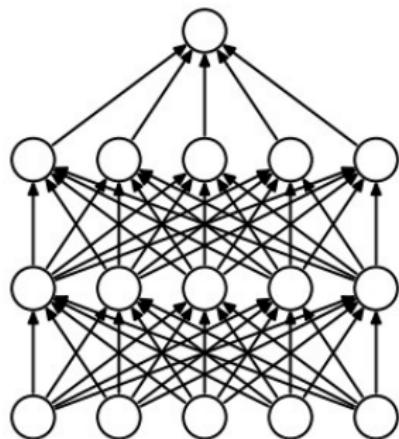
- ▶ $b^{t+1} = b^t - \eta \cdot \nabla J(b) + \alpha(b^t - b^{t-1})$
- ▶ α is the momentum parameter
- ▶ helps avoiding stuck in a local optimum

Regularization

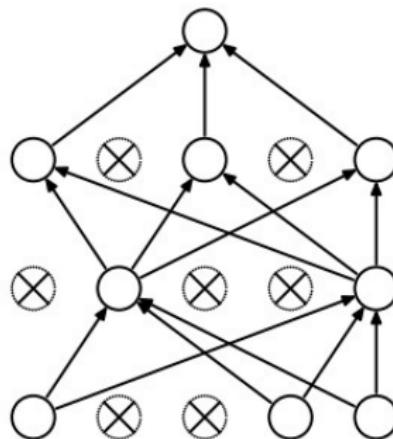
- ▶ L1 penalty on the parameters
- ▶ L2 penalty on the parameters (weight decay parameter)
- ▶ Early stopping

Fitting neural networks

Dropout:



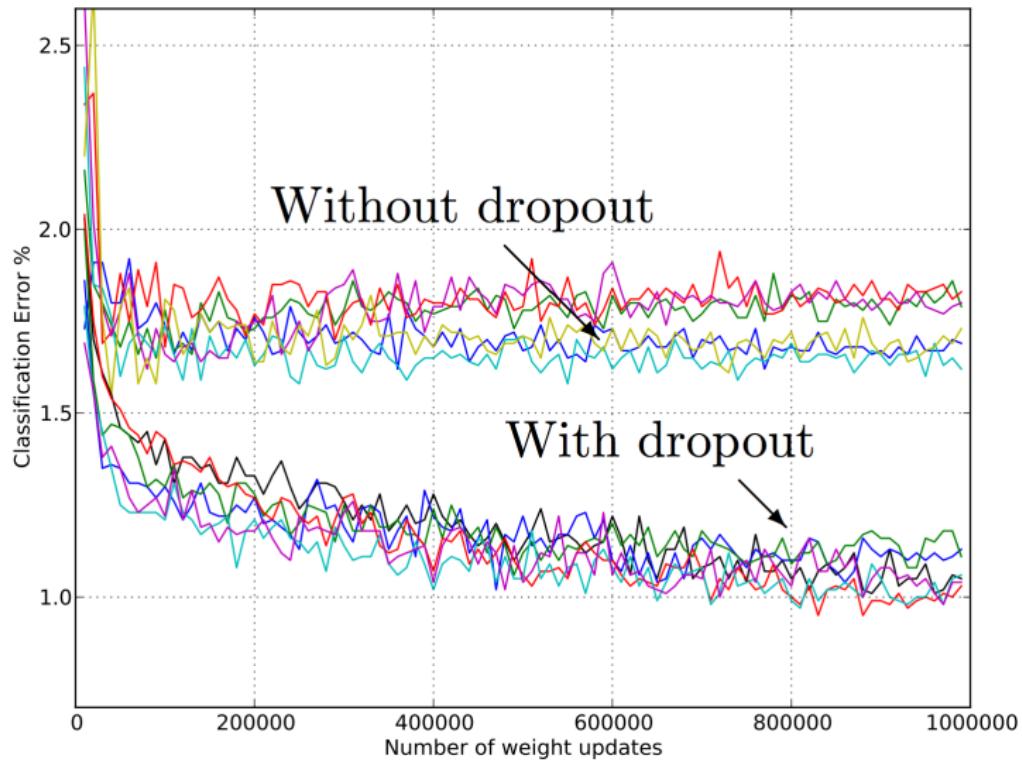
(a) Standard Neural Net



(b) After applying dropout.

<https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf>

Fitting neural networks



Fitting neural networks: Tips from H_2O

- ▶ more layers for more complex functions (more non-linearity)
- ▶ more neurons per layer to fit finer structure in data
- ▶ add regularization (`max_w2=50` or `L1=1e-5`)
- ▶ do a grid search do get a feel for parameters
- ▶ try “Tanh,” then “Rectifier”
- ▶ try dropout (input 20%, hidden 50%)

Example: MNIST

Famous data set in machine learning community

- ▶ <http://yann.lecun.com/exdb/mnist/>

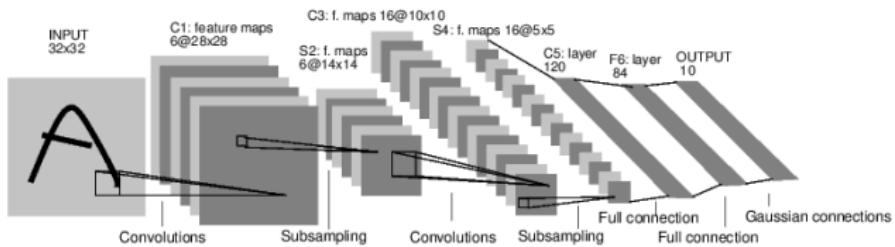
Even today: Kaggle competition

- ▶ <https://www.kaggle.com/c/digit-recognizer>

Online demo

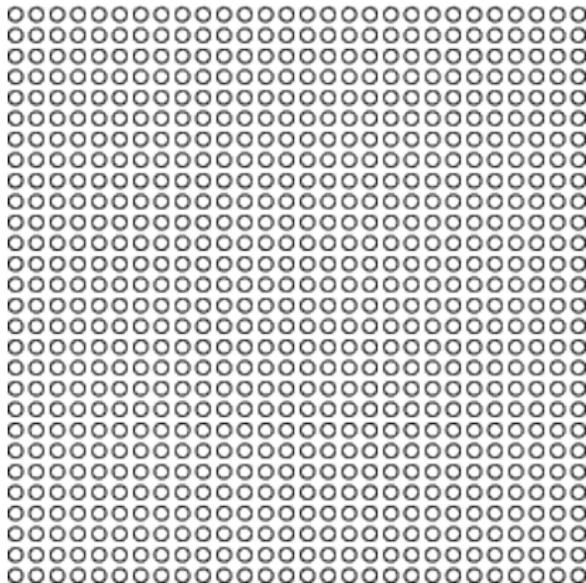
- ▶ <http://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html>

LeNet5: convolutional neural network

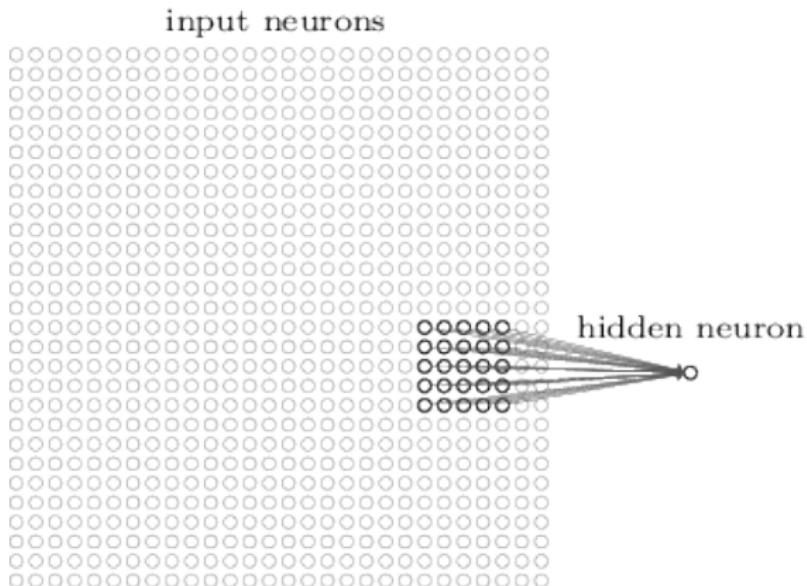


See <http://yann.lecun.com/exdb/lenet/>

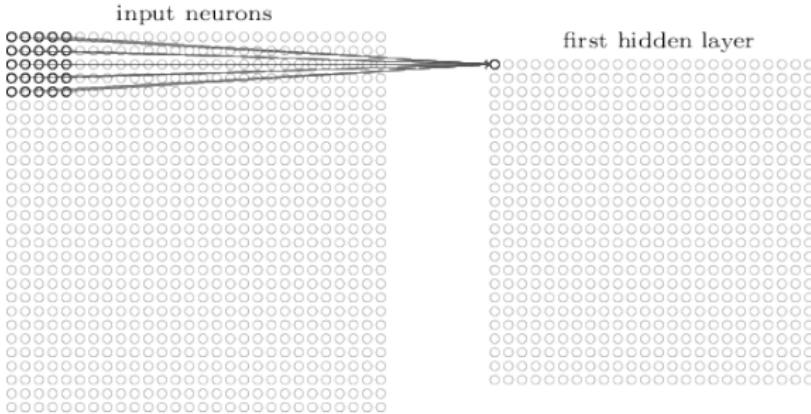
input neurons



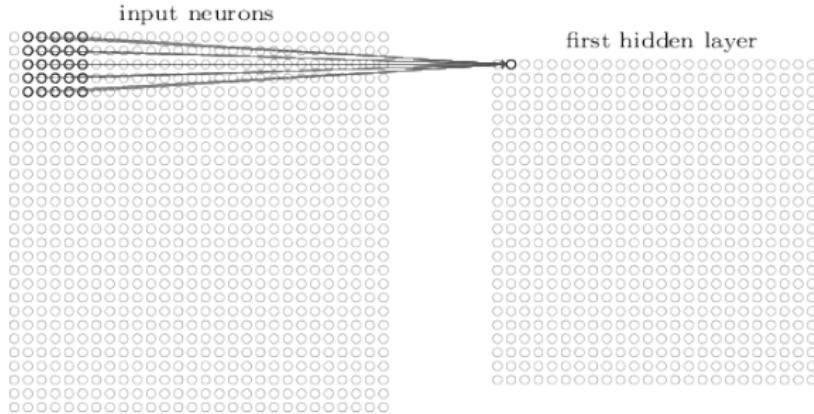
From: <http://neuralnetworksanddeeplearning.com/chap6.html>



From: <http://neuralnetworksanddeeplearning.com/chap6.html>

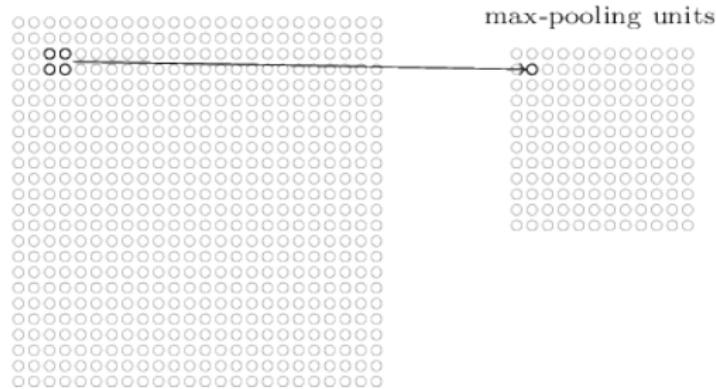


From: <http://neuralnetworksanddeeplearning.com/chap6.html>



From: <http://neuralnetworksanddeeplearning.com/chap6.html>

hidden neurons (output from feature map)

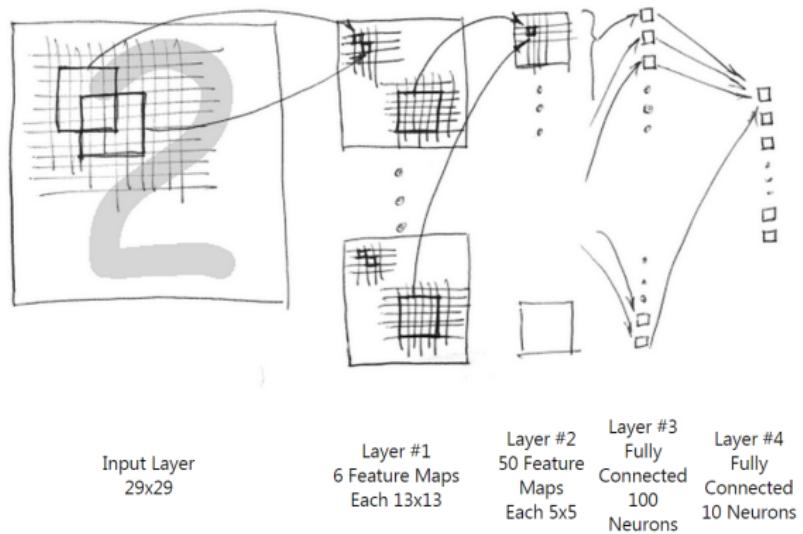


From: <http://neuralnetworksanddeeplearning.com/chap6.html>

Mistakes made by LeNet5



A simpler architecture



From: <http://www.codeproject.com/Articles/16650/Neural-Network-for-Recognition-of-Handwritten-Digi>

Practical consideration

Standard trick — expand the set of examples

- ▶ small distortions, scaling, rotation, ...

What else needs to be done to make system useful?

Advantages and disadvantages

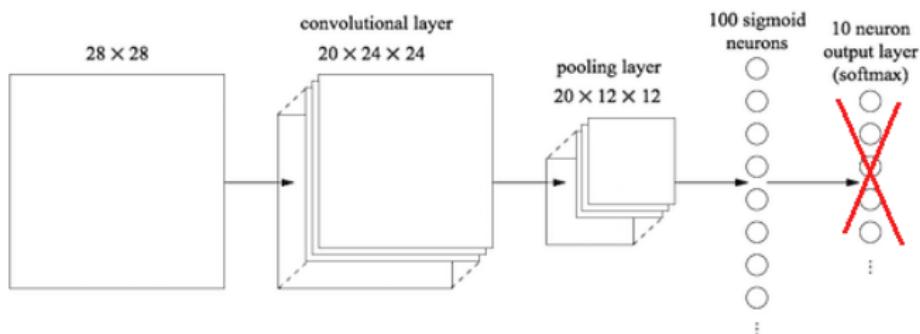
Pros:

- ▶ Tolerance to noise
- ▶ Able to capture complex signals
- ▶ In some applications lead to the state-of-the-art performance
- ▶ Fast at test time

Cons:

- ▶ Very hard/impossible to interpret (black box method)
- ▶ Can easily overfit
- ▶ Need a large amount of data to train
- ▶ Slow to train

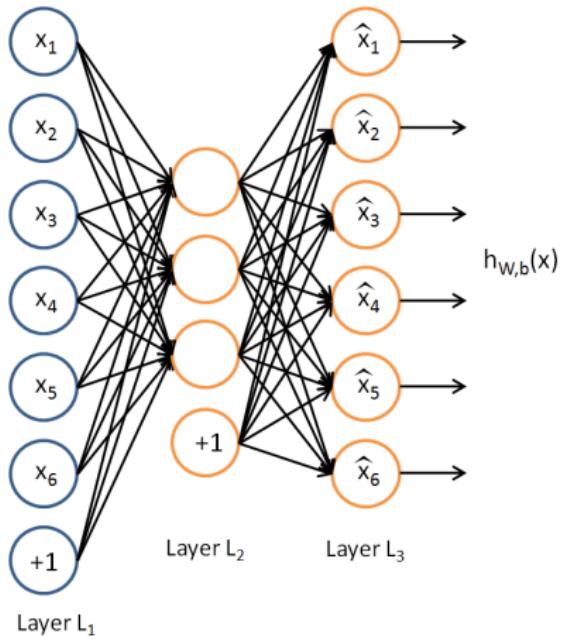
Learning representation



Use the output of the last layer as a representation of your data.

Fit a model with this representation.

Autoencoder



Network trained to reproduce its input at the output layer.

Usually tie the weights that go into and out of the hidden layer.

Autoencoder

Loss function

- ▶ For real valued inputs, try to find weights such that

$$\frac{1}{2} \sum_k (x_k - \hat{x}_k)^2$$

is minimized

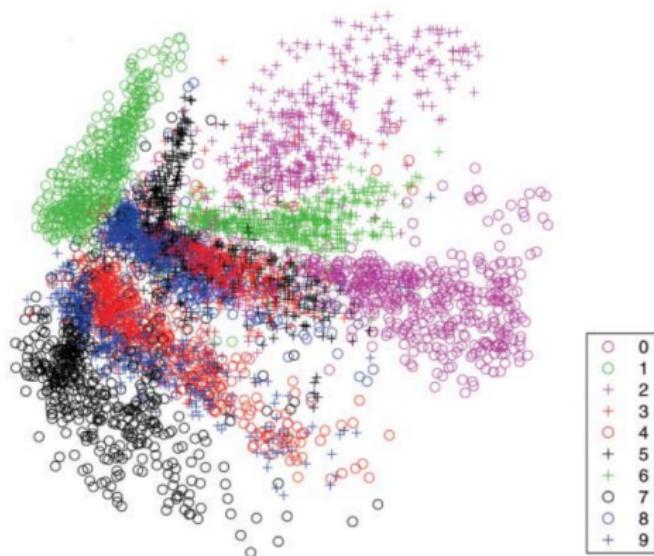
- ▶ For binary input cross entropy is used, which is similar to deviance

Fitting autoencoder

- ▶ Same tricks as before
- ▶ Greedy learning of stacked autoencoders
- ▶ <https://www.cs.toronto.edu/~hinton/science.pdf>

Autoencoder: Why are they useful?

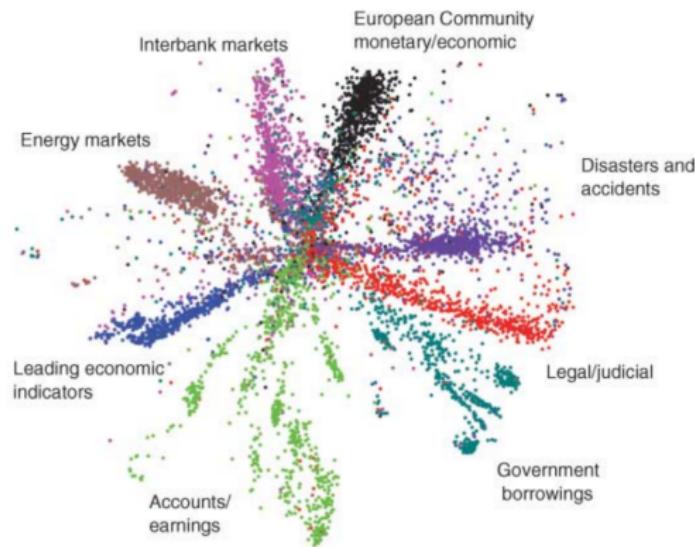
Learning compressed representation of the input distribution
(dimensionality reduction)



Autoencoder structure: 784 — 1000 — 500 — 250 — 2

Autoencoder: Why are they useful?

Information retrieval: 804,414 newswire stories

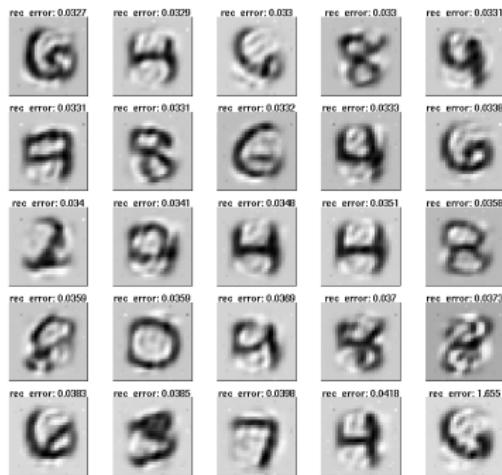
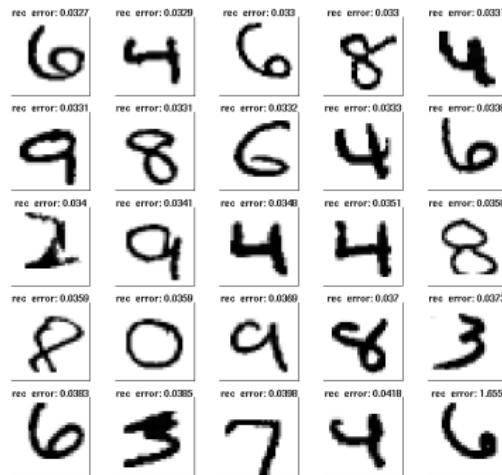


Autoencoder structure: 2000 — 500 — 250 — 125 — 2

<https://www.cs.toronto.edu/~hinton/science.pdf>

Autoencoder: Why are they useful?

- ▶ unsupervised pretraining of weights (many unlabeled images, but only few labeled)
- ▶ anomaly detection



Some success stories

- ▶ Google voice transcription

<http://googleresearch.blogspot.com/2015/08/the-neural-networks-behind-google-voice.html>

- ▶ Google voice search

<http://googleresearch.blogspot.com/2015/09/google-voice-search-faster-and-more.html>

- ▶ Google translate app

<http://googleresearch.blogspot.com/2015/07/how-google-translate-squeezes-deep.html>

Some success stories

- ▶ Facebook face recognition

<http://www.technologyreview.com/news/525586/facebook-creates-software-that-matches-faces-almost-as-well-as-you-do>

- ▶ Paypal fraud detection

<http://www.slideshare.net/0xdata/paypal-fraud-detection-with-deep-learning-in-h2o-presentationh2oworld2014>

Additional resources

- ▶ Free online book by Michael Nielsen
<http://neuralnetworksanddeeplearning.com/>
(explains backpropagation well)
- ▶ <http://deeplearning.net/tutorial/>
Excellent tutorial using Theano library in Python