

Probabilistic Programming

Marius Popescu

popescunmarius@gmail.com

2019 - 2020

Project 2

Bayesian Neural Networks

Bayesian Neural Networks



Bayesian learning for neural networks

Advisers: [Geoffrey Hinton](#)

Authors: [Radford M. Neal](#)

Publication:

· Doctoral Dissertation

Bayesian learning for neural networks

University of Toronto Toronto, Ont., Canada, Canada ©1995

ISBN:0-612-02676-0

1995 Doctoral Dissertation

 [Bibliometrics](#)

- Citation Count: 56
- Downloads (cumulative): n/a
- Downloads (12 Months): n/a
- Downloads (6 Weeks): n/a

Lecture Notes in
Statistics

118

Radford M. Neal

**Bayesian Learning
for Neural Networks**



<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.446.9306&rep=rep1&type=pdf>

Bayesian Neural Networks

A Bayesian neural network is a neural network with a prior distribution on its weights

Define the prior on the weights and biases \mathbf{w} to be the standard normal:

$$p(\mathbf{w}) = \text{Normal}(\mathbf{0}, \mathbf{I})$$

Define the likelihood for a data point as:

$$p(y|\mathbf{x}, \mathbf{w}) = \text{Bernoulli}(NN(\mathbf{x}, \mathbf{w}))$$

Where $\mathbf{x} \in \mathbb{R}^d$ is the feature vector, $y \in \{0,1\}$ is the output (binary classification), and $NN(\mathbf{x}, \mathbf{w})$ is the function (deterministic) implemented by the neural network whose weights and biases form the latent variables \mathbf{w} . The output of the neural network must be a value between 0 and 1 (for example the last layer has only one sigmoid neuron).

Or:

$$p(y|\mathbf{x}, \mathbf{w}) = \text{Categorical}(NN(\mathbf{x}, \mathbf{w}))$$

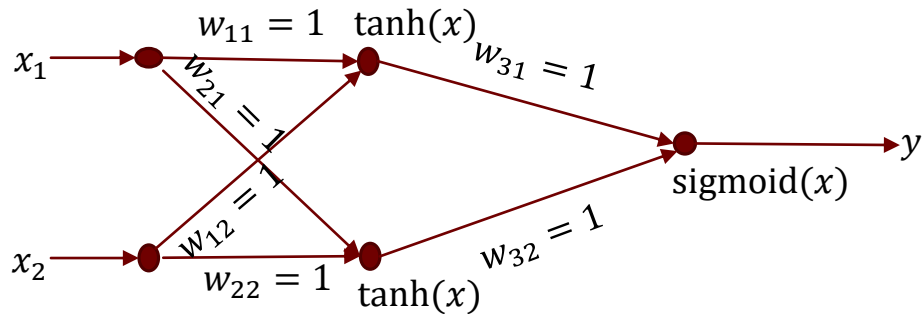
Where $\mathbf{x} \in \mathbb{R}^d$ is the feature vector, $y \in \{1,2, \dots k\}$ is the output (multiclass classification), and $NN(\mathbf{x}, \mathbf{w})$ is the function (deterministic) implemented by the neural network whose weights and biases form the latent variables \mathbf{w} . The outputs of the neural network must sum to 1 (for example the last layer is a softmax layer).

The Task

- Start with a network architecture
- Training: using the training data (observed) infer the posterior distribution of \mathbf{w}
- Testing: using this posterior distribution predict the labels of testing data
- Compare to “classic” neural networks results on the same problem
- Do this for both binary classification and multiclass classification

Sanity Check

Take a simple neural network with fixed weights



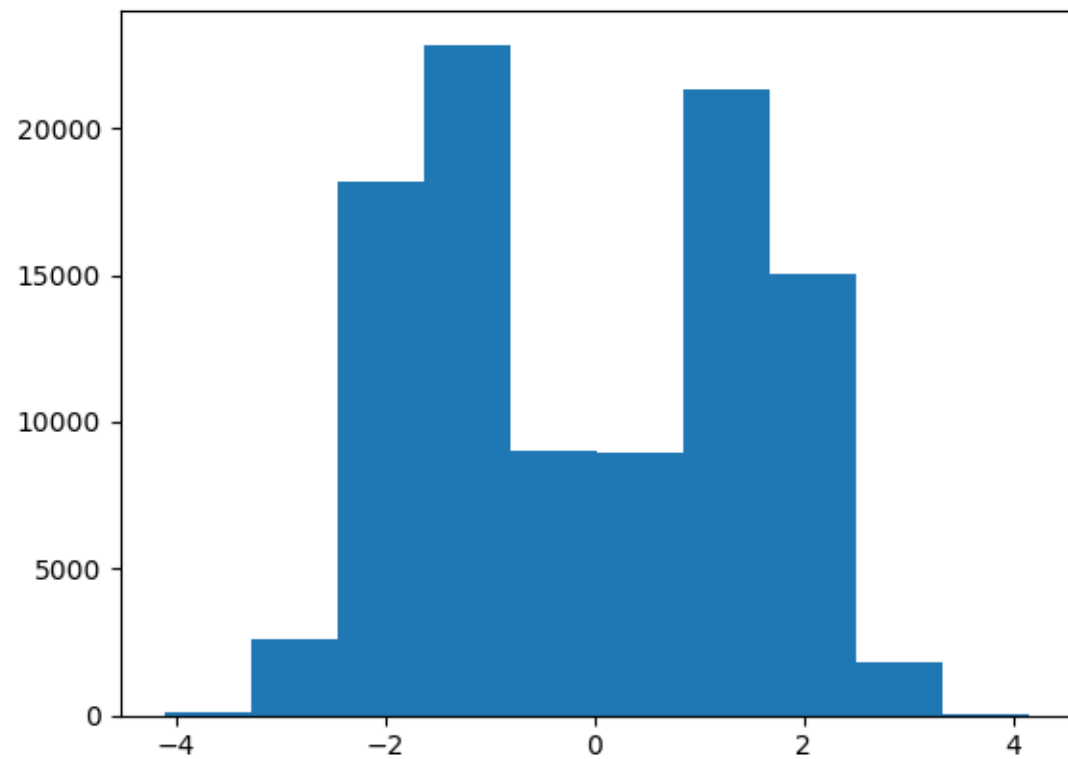
Generate labeled data (100 training, 100 testing) by feeding random input vectors into this network:

```
X = np.random.randn(100, 2)
Y = np.tanh(X[:, 0] + X[:, 1])
Y = 1. / (1. + np.exp(-(Y + Y)))
Y = Y > 0.5
```

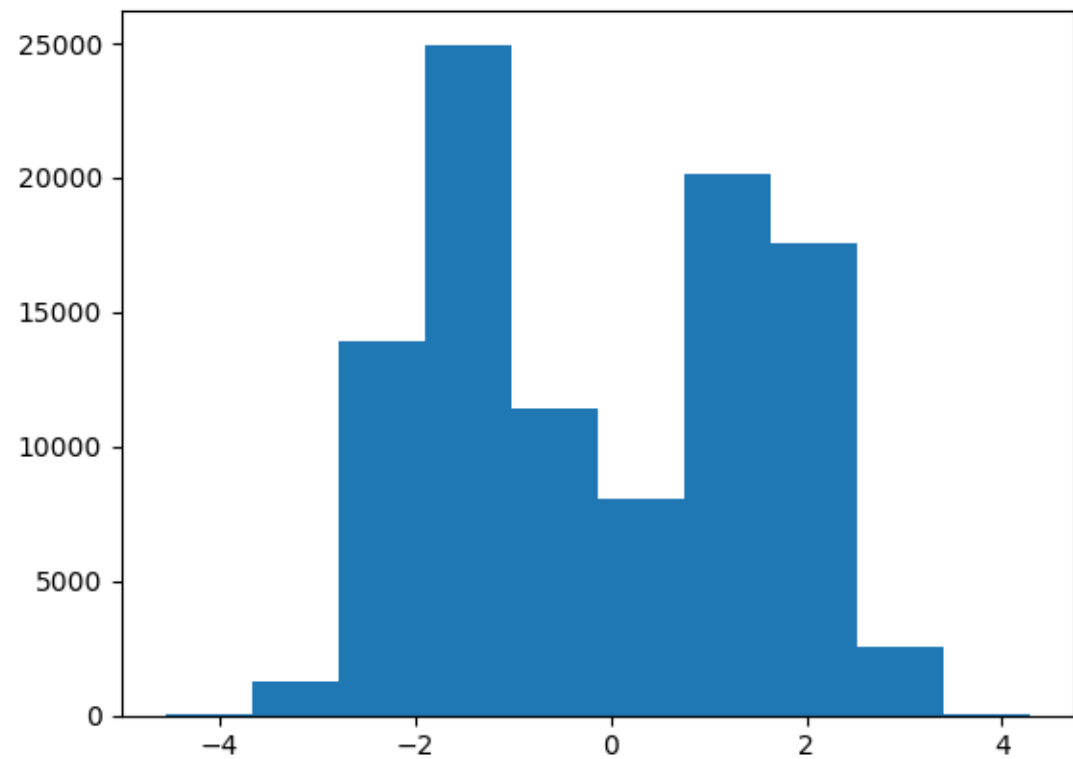
- “Train” a Bayesian neural network with the same architecture
- Analyse the posterior of weights
- Test it

Sanity Check

Posterior of w_{11}

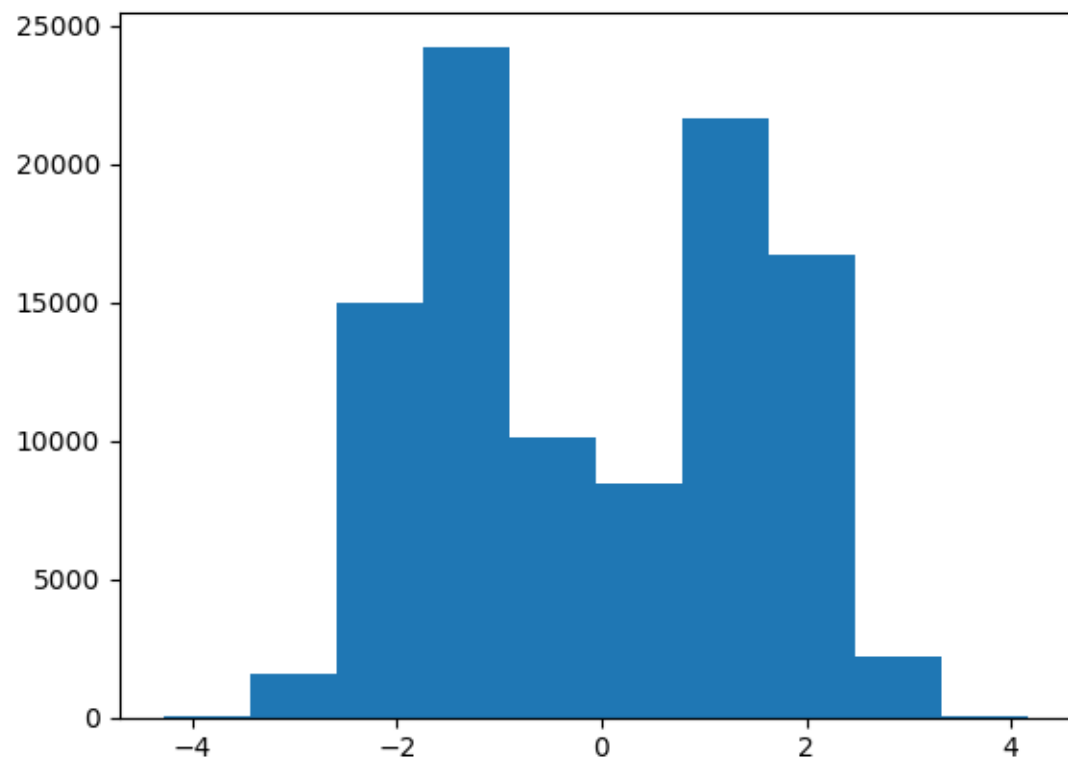


Posterior of w_{12}

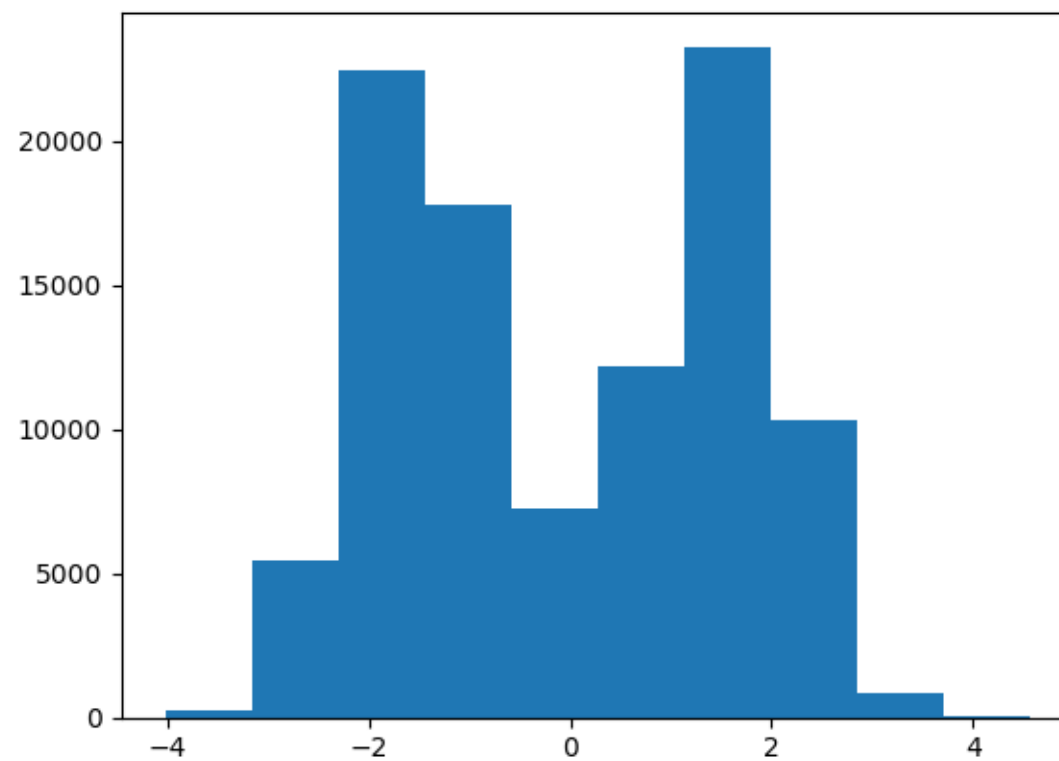


Sanity Check

Posterior of w_{21}

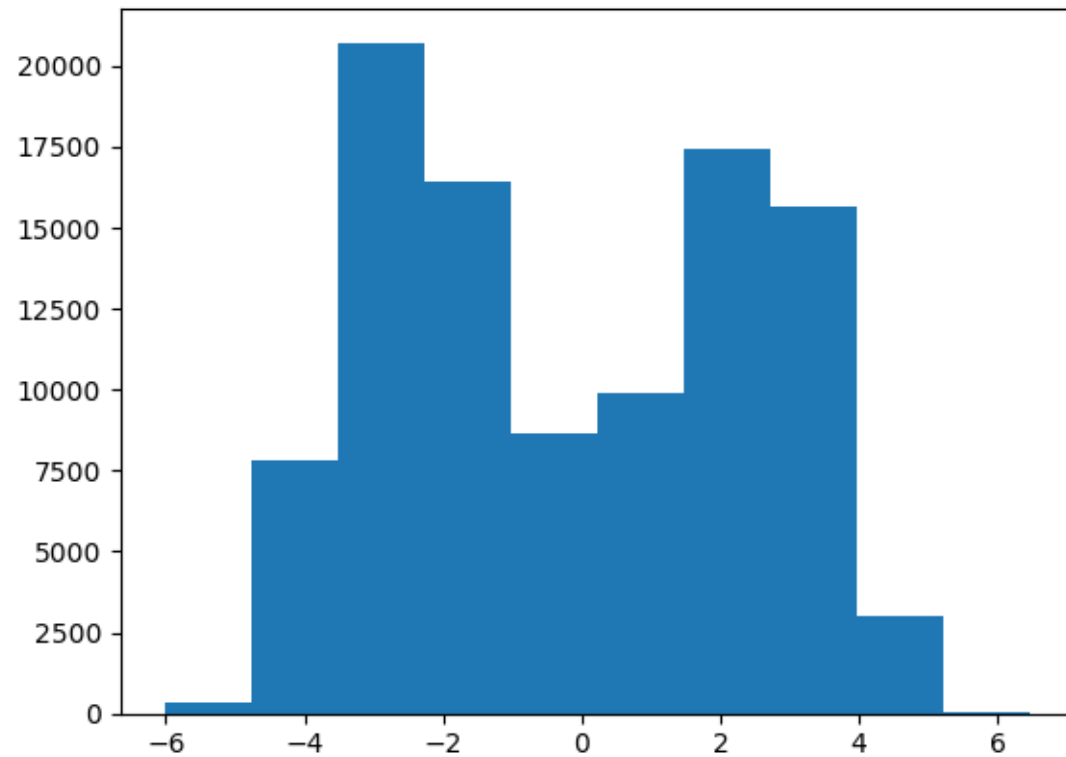


Posterior of w_{22}

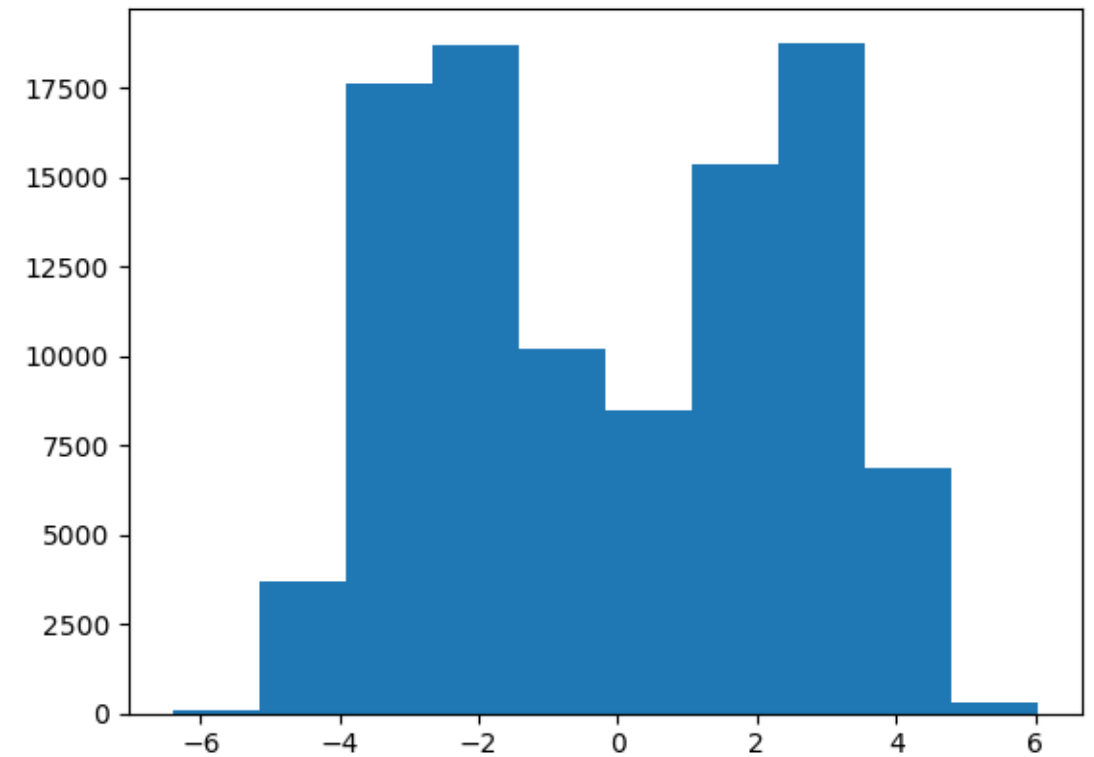


Sanity Check

Posterior of w_{31}



Posterior of w_{32}



Sanity Check

```
{C:\My\ProbabilisticProgrammingCourse\PyMC\11} - Far 3.0.5225 x64 Administrator
-----98%-----] 1966667 of 2000000 complete in 41:06 PM
-----98%-----] 1968991 of 2000000 complete in 433.1 se
-----98%-----] 1971313 of 2000000 complete in 433.6 se
-----98%-----] 1973610 of 2000000 complete in 434.1 se
-----98%-----] 1975933 of 2000000 complete in 434.6 se
-----98%-----] 1978259 of 2000000 complete in 435.1 se
-----99%-----] 1980585 of 2000000 complete in 435.6 se
-----99%-----] 1982913 of 2000000 complete in 436.1 se
-----99%-----] 1985239 of 2000000 complete in 436.6 se
-----99%-----] 1987565 of 2000000 complete in 437.1 se
-----99%-----] 1989886 of 2000000 complete in 437.6 se
-----99%-----] 1992214 of 2000000 complete in 438.1 se
-----99%-----] 1994539 of 2000000 complete in 438.6 se
-----99%-----] 1996867 of 2000000 complete in 439.1 se
-----99%-----] 1999016 of 2000000 complete in 439.6 se
-----100%-----] 2000000 of 2000000 complete in 439.8 se
C
One sample of weights (w11 w12 w21 w22 w31 w32):
1.9341008360887364 1.8406646861829108 -1.7122120349091845 -1.9078024233806719 1.
3749158404689492 -2.761102320866148
Accuracy on training dataset: 1.0
Accuracy on testing dataset: 0.99
C:\...ilisticProgrammingCourse\PyMC\11>
```

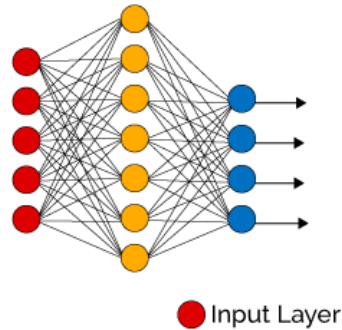
```
{C:\My\ProbabilisticProgrammingCourse\PyMC\11} - Far 3.0.5225 x64 Administrator
-----98%-----] 1965447 of 2000000 complete in 12:28 PM
-----98%-----] 1967809 of 2000000 complete in 422.6 se
-----98%-----] 1970179 of 2000000 complete in 423.1 se
-----98%-----] 1972541 of 2000000 complete in 423.6 se
-----98%-----] 1974900 of 2000000 complete in 424.1 se
-----98%-----] 1977247 of 2000000 complete in 424.6 se
-----98%-----] 1979616 of 2000000 complete in 425.1 se
-----99%-----] 1981980 of 2000000 complete in 425.6 se
-----99%-----] 1984345 of 2000000 complete in 426.1 se
-----99%-----] 1986708 of 2000000 complete in 426.6 se
-----99%-----] 1989074 of 2000000 complete in 427.1 se
-----99%-----] 1991437 of 2000000 complete in 427.6 se
-----99%-----] 1993797 of 2000000 complete in 428.1 se
-----99%-----] 1996159 of 2000000 complete in 428.6 se
-----99%-----] 1998520 of 2000000 complete in 429.1 se
-----100%-----] 2000000 of 2000000 complete in 429.4 se
C
One sample of weights (w11 w12 w21 w22 w31 w32):
-1.9476446732221109 -2.187090102085202 -1.0147916576441696 -0.8044554053301556 -
3.612231498456268 -1.1565146231491958
Accuracy on training dataset: 1.0
Accuracy on testing dataset: 1.0
C:\...ilisticProgrammingCourse\PyMC\11>
```

Comparison to “classic” neural networks

```
[C:\My\ProbabilisticProgrammingCourse\PyMC\11] - Far 3.0.5225 x64 Administrator 1:12 PM
0000 - val_loss: 0.1936 - val_acc: 0.9700
Epoch 9994/10000
100/100 [=====] - 0s 10us/step - loss: 0.1784 - acc: 1.0000 - val_loss: 0.1936 - val_acc: 0.9700
Epoch 9995/10000
100/100 [=====] - 0s 10us/step - loss: 0.1784 - acc: 1.0000 - val_loss: 0.1936 - val_acc: 0.9700
Epoch 9996/10000
100/100 [=====] - 0s 10us/step - loss: 0.1784 - acc: 1.0000 - val_loss: 0.1935 - val_acc: 0.9700
Epoch 9997/10000
100/100 [=====] - 0s 10us/step - loss: 0.1784 - acc: 1.0000 - val_loss: 0.1935 - val_acc: 0.9700
Epoch 9998/10000
100/100 [=====] - 0s 5us/step - loss: 0.1784 - acc: 1.0000 - val_loss: 0.1935 - val_acc: 0.9700
Epoch 9999/10000
100/100 [=====] - 0s 5us/step - loss: 0.1784 - acc: 1.0000 - val_loss: 0.1935 - val_acc: 0.9700
Epoch 10000/10000
100/100 [=====] - 0s 5us/step - loss: 0.1784 - acc: 1.0000 - val_loss: 0.1935 - val_acc: 0.9700
C:\...\ilisticProgrammingCourse\PyMC\11>
1Left 2Right 3View.. 4Edit.. 5Print 6MkLink 7Find 8Histy 9Video 10
```

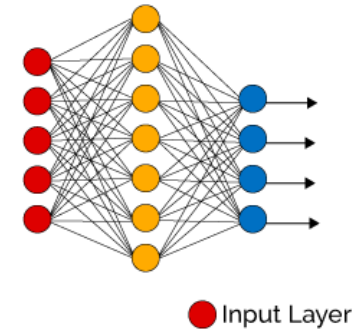
```
[C:\My\ProbabilisticProgrammingCourse\PyMC\11] - Far 3.0.5225 x64 Administrator 1:09 PM
700 - val_loss: 0.2605 - val_acc: 0.9300
Epoch 9994/10000
100/100 [=====] - 0s 10us/step - loss: 0.2462 - acc: 0.9700 - val_loss: 0.2605 - val_acc: 0.9300
Epoch 9995/10000
100/100 [=====] - 0s 10us/step - loss: 0.2461 - acc: 0.9700 - val_loss: 0.2604 - val_acc: 0.9300
Epoch 9996/10000
100/100 [=====] - 0s 10us/step - loss: 0.2461 - acc: 0.9700 - val_loss: 0.2604 - val_acc: 0.9300
Epoch 9997/10000
100/100 [=====] - 0s 10us/step - loss: 0.2461 - acc: 0.9700 - val_loss: 0.2604 - val_acc: 0.9300
Epoch 9998/10000
100/100 [=====] - 0s 5us/step - loss: 0.2461 - acc: 0.9700 - val_loss: 0.2604 - val_acc: 0.9300
Epoch 9999/10000
100/100 [=====] - 0s 5us/step - loss: 0.2460 - acc: 0.9700 - val_loss: 0.2603 - val_acc: 0.9300
Epoch 10000/10000
100/100 [=====] - 0s 5us/step - loss: 0.2460 - acc: 0.9700 - val_loss: 0.2603 - val_acc: 0.9300
C:\...\ilisticProgrammingCourse\PyMC\11>
1Left 2Right 3View.. 4Edit.. 5Print 6MkLink 7Find 8Histy 9Video 10
```

Overparametrization May Help Optimization : Folklore Experiment



Generate labeled data by feeding random input vectors into depth 2 net with hidden layer of size n

Still no theorem explaining this...



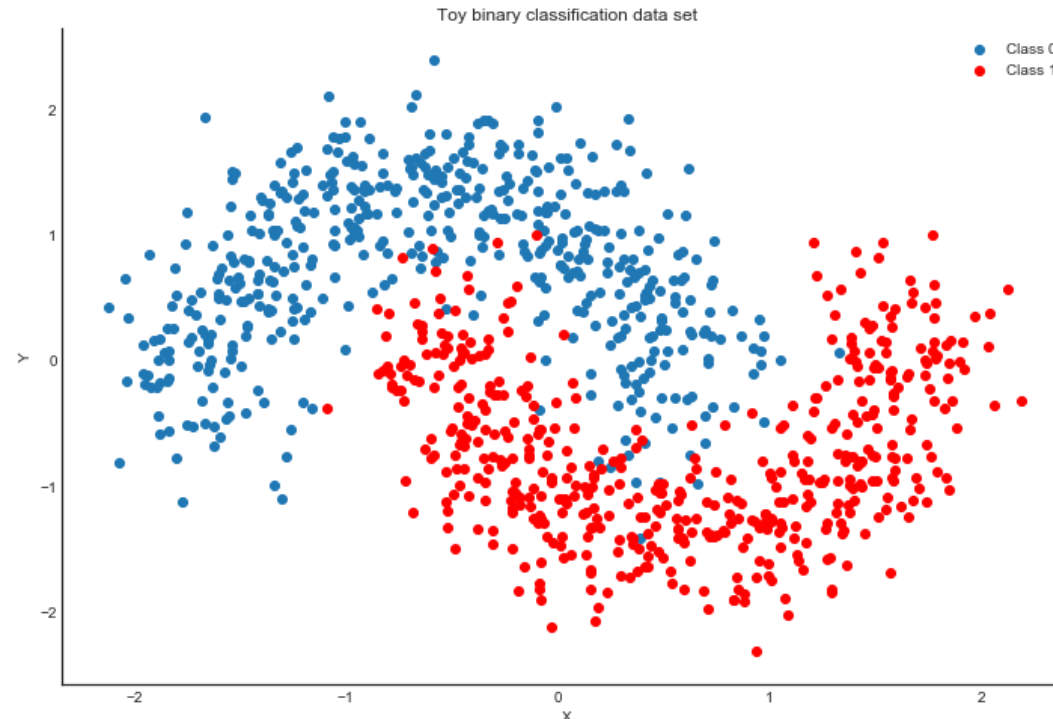
Difficult to train a new net using this labeled data with same # of hidden nodes

Much easier to train a new net with bigger hidden layer!

Other small datasets

“Two Moons” dataset: toy dataset for a simple binary classification problem that's not linearly separable.

```
sklearn.datasets.make_moons(noise=0.2, random_state=0, n_samples=1000)
```



Extras

- Extensions (up to 1.5 points):
 - Priors over architectures
 - Weight sharing (CNN, Siamese Networks)
 - for more ideas see chapter 2 from “Bayesian Learning for Neural Networks”
 - ...