Bayesian neural networks

ISTA 410 / INFO 510: Bayesian Modeling and Inference

U. of Arizona School of Information

December 1, 2021

Outline

Last time:

- Approximate computational methods: EM, ADVI, OPVI

Today:

- Brief overview of Bayesian neural networks

Quick NN overview

The basic idea

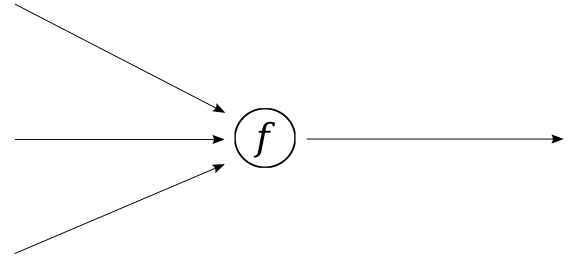
A neural network is a way to build a mathematical function out of smaller components.

- Create a simple statistical model called a *neuron*
- Organize many neurons into a structure where data is processed through several *layers* of neurons
- Use the output of the final layer to predict a target variable

To understand a network, first let's understand the building block: an individual neuron.

A single neuron

A single neuron is a mathematical function that takes a vector of inputs, multiplies them by weights, and then applies a function:



$$out = f(b + w_0x_0 + w_1x_1 + \ldots + w_nx_n)$$

Activation function

The function f is called the *activation function*.

Basic idea:

- f measures how much the neuron sees "what it is looking for"; i.e. the neuron is sensitive to a certain combination of inputs
- f should be an increasing function of its input
- f should be nonlinear

Example activation functions

A few common choices of activation function:

- Logistic sigmoid

$$f(x) = \frac{1}{1 + e^{-x}}$$

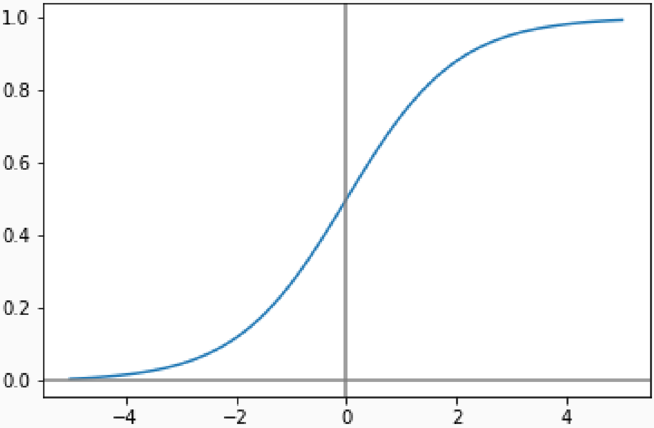
- Hyperbolic tangent

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- Rectified linear unit (ReLU)

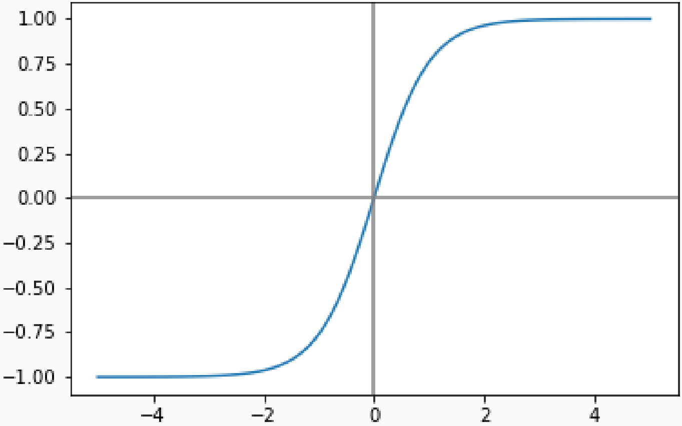
$$f(x) = \max(x, 0)$$

Example activation function: sigmoid



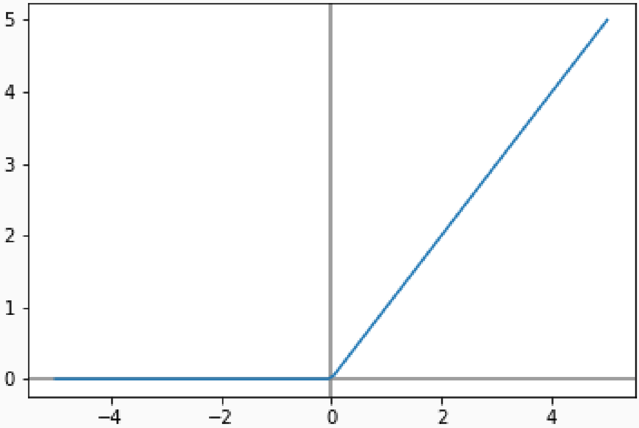
Sigmoid activation function

Example activation functions



Tanh activation function

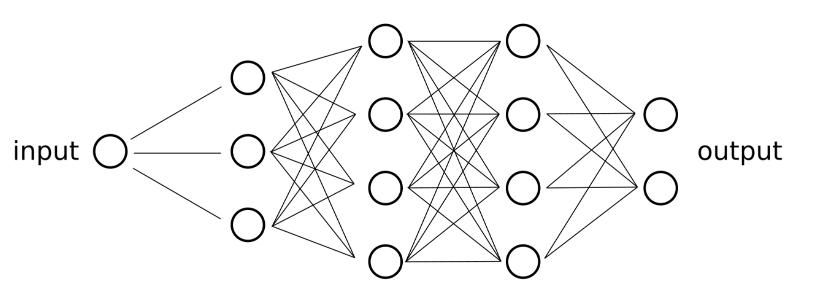
Example activation functions



ReLU activation function

Organization into layers

A *dense feed-forward neural network* is organized into layers:



Structurally, the activation from in each layer ℓ_i is

$$\boldsymbol{A}_{\ell_i} = f(\boldsymbol{W}\boldsymbol{a}_{\ell_{i-1}})$$

with the activation f applied row-wise to the data/activation matrix.

So what do the neurons actually do?

What the individual neurons are actually up to:

- The activation is an increasing function of its input
- The input is a dot product between that neuron's weights and the incoming activation vector
- Dot products, geometrically, measure the degree to which two vectors point in the same direction
- So the weights learned by each neuron can be thought of as a prototype for the combination of features it is "looking for"
- In later layers, these combinations are combinations of previous activations, so features of increasing complexity are selected for

So: neural networks are feature extraction engines with some conventional estimation model placed on the end

Training a neural network

What does training mean?

Conventionally, a neural network is just treated a really complicated curve fit:

- By composing the functions together, you could in theory write down a formula
- That formula would contain all the weights as parameters
- The weights can be adjusted using gradient descent

Backpropagation allows for relatively efficient estimation of the gradients without explicitly writing the estimation function

- Feed an instance through the network and make a prediction
- Calculate the value of the loss function
- Feed the loss backwards through the network, calculating gradients along the way

So we never have to expand out the entire formula for the network.

Bayesian NN

The previous framework is not, however, Bayesian:

- No probability distributions for model weights
- No regularizing priors
- No uncertainty quantification for estimated weights
- MLE only

To make this Bayesian, we think of each weight as an unknown random variable with a probability distribution

Implementation details in PyMC3

Layer weights

The parameters to be estimated in a dense feedforward NN are:

- Matrices of weights, one per layer, Normal priors
- Shape determined by size of layer and size of previous layer/input
- Final layer size determined by number of classes
- At each layer, multiply the weight matrix by the input and pass to the activation function

Activation functions

Some of the common activation functions are implemented in PyMC/Theano:

- `theano.tensor.nnet.sigmoid` – logistic sigmoid
- `theano.tensor.nnet.tanh` – hyperbolic tangent
- `theano.tensor.nnet.relu` – ReLU

The `pm.Data` class

The `pm.Data` class:

- Relatively recent addition to PyMC3
- Creates a variable for storing observed values
- Allows for substituting different values after model fitting
    - Substitute in testing data for evaluation
    - Substitute in new observations for prediction

The `pm.Data` class

Inside a model context:

```
nn_input = pm.Data('nn_input', X)
nn_output = pm.Data('nn_output', y)

...

species = pm.Categorical('species',
                         p=prob_output,
                         observed=nn_output)
```

Packaging this up

Useful to define a function that sets up the model and returns it:

```
def construct_nn(X, y, layer_width):
    with pm.Model() as neural_network:
        nn_input = pm.Data('nn_input', X)
        nn_output = pm.Data('nn_output', y)

        # Layer weights
        weights_1 = pm.Normal('weights_1', mu=0, sigma=1,
                        shape=(X.shape[1], layer_width))
        weights_2 = pm.Normal('weights_2', mu=0, sigma=1,
                        shape=(layer_width, layer_width))
        weights_out = pm.Normal('weights_out', mu=0, sigma=1,
                                  shape=(layer_width, 3))

        # Hidden layer activations
        # Dot each output matrix with the weight matrix
        layer_1_output = pm.math.tanh(tt.dot(nn_input, weights_1))
        layer_2_output = pm.math.tanh(tt.dot(layer_1_output, weights_2))

        prob_output = pm.Deterministic('species_probs',
         tt.nnet.softmax(tt.dot(layer_2_output, weights_out)))

        # Categorical likelihood is to Bernoulli what Multinomial is to Binomial
        species = pm.Categorical('species', p=prob_output, observed=nn_output)

    return neural_network
```

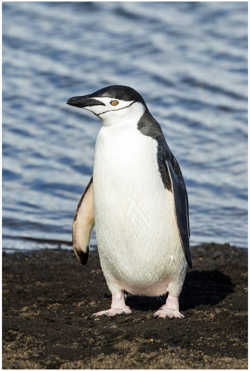
Using the network

Once we have defined this function:

- Construct the network, passing layer size parameters and training data
- Fit to training data using ADVI or MCMC
- Make predictions:
    - Substitute new data
    - Sample from the posterior predictive distribution and average predictions

Example

We can start by testing this idea on an easy classification problem from a standard data set.

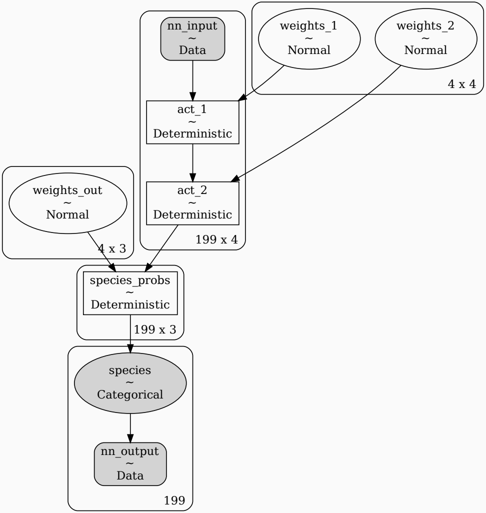




Palmer penguins data set

Steps

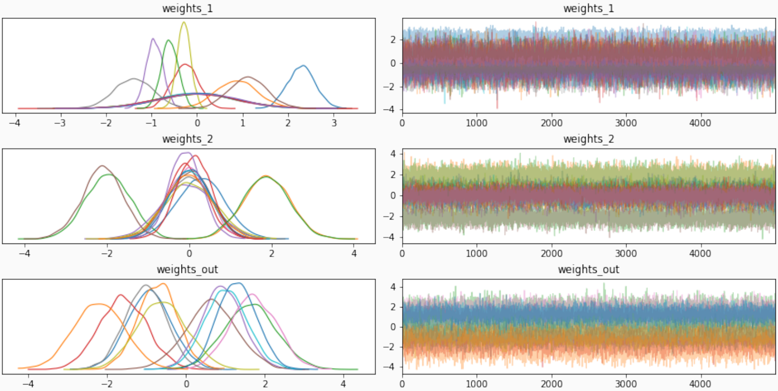
- Standardize the inputs
- Construct a NN with 5 neurons per layer
- Fit using ADVI
- Check the parameters

Plate diagram



After fitting

Inspect the traceplot for the weights:



Evaluating the model

To test the model, sub in new data and sample from the posterior predictive:

```
pm.set_data(new_data={"nn_input": testX,
                      "nn_output": testy},
            model=nn)
ppc = pm.sample_posterior_predictive(trace,
            samples=500,
            model=nn,
            var_names=['species', 'species_probs'])
```

Evaluating the model

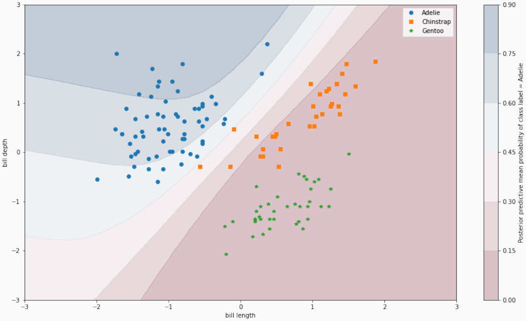
Treat each sample in the predictive sample as a "vote:"

```
>>> from scipy.stats import mode
>>> sum(mode(ppc['species'], axis=0).mode[0,:] == testy) / len(testy)
0.9925373134328358
```

- 99% accuracy on the testing set
- (granted, this is a pretty easy data set)

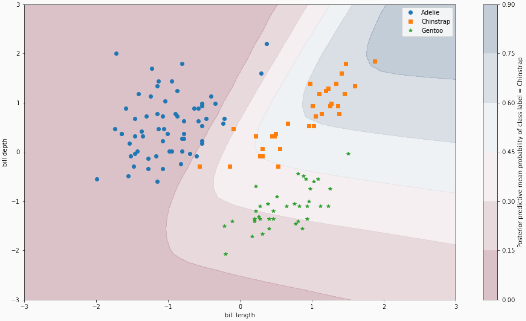
Visualizing the decision function

By predicting on a grid of values, we can also visualize the learned prediction function:



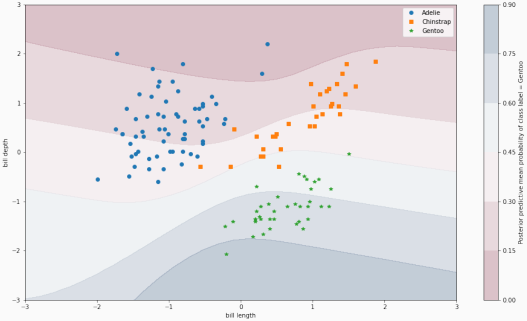
Visualizing the decision function

By predicting on a grid of values, we can also visualize the learned prediction function:



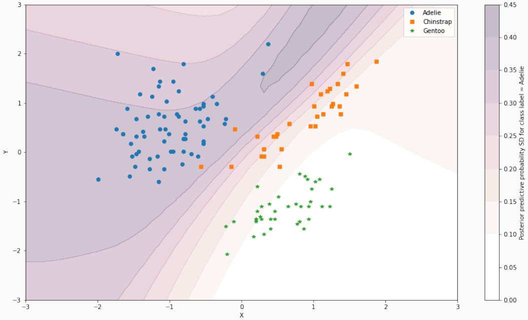
Visualizing the decision function

By predicting on a grid of values, we can also visualize the learned prediction function:



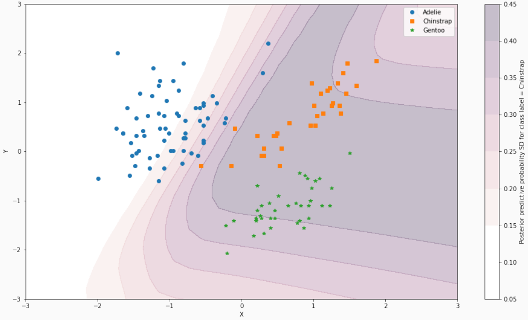
Visualizing the prediction uncertainty

Because our network is Bayesian, we also get estimates of the uncertainty in prediction



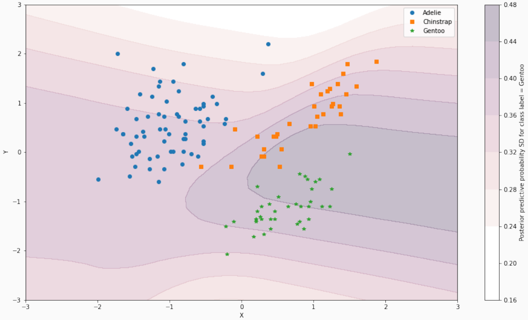
Visualizing the prediction uncertainty

Because our network is Bayesian, we also get estimates of the uncertainty in prediction



Visualizing the prediction uncertainty

Because our network is Bayesian, we also get estimates of the uncertainty in prediction



Hierarchical neural networks

What else does Bayes have to offer

The previous example showed the possibility of using PyMC3 and Bayesian inference to fit a neural network

- Didn't introduce much structure that isn't seen in conventional neural networks
- Uncertainty estimates are nice for stats nerds, but do ML prediction wizards care?
- TensorFlow machine go brrr

Adding hierarchical structure

One contribution of Bayesian modeling to the broader statistical modeling landscape: hierarchical/multilevel structure

- Have data partitioned into a number of groups, with some similarity across the groups
- Too much difference between the groups for pooled inferences to be adequately representative
- Similarity across groups necessary to get precise estimates
    - Particularly if data in individual groups is limited

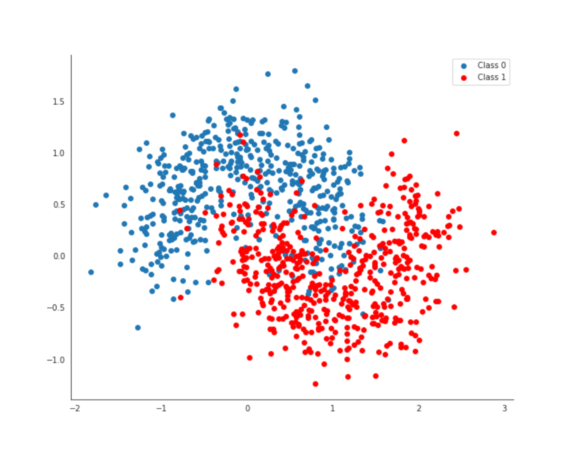
Example

This example is due to Thomas Wiecki.

- Have a standard “two moons” classification problem
- In the simple form, a BNN learns the decision boundary quite nicely
- But, data is split across many groups, each of which is transformed by a rotation
- Within each group, not enough data to accurately estimate the decision boundary

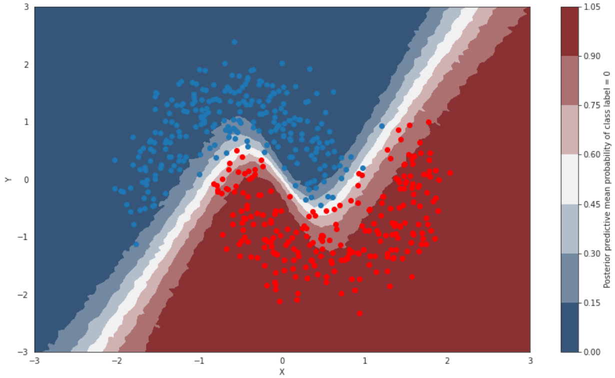
Untransformed moons

Without any rotations:

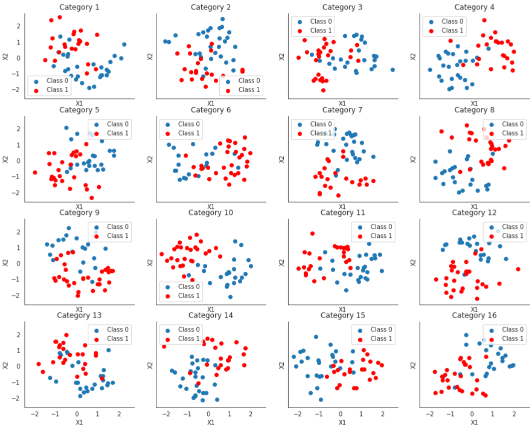


Untransformed moons

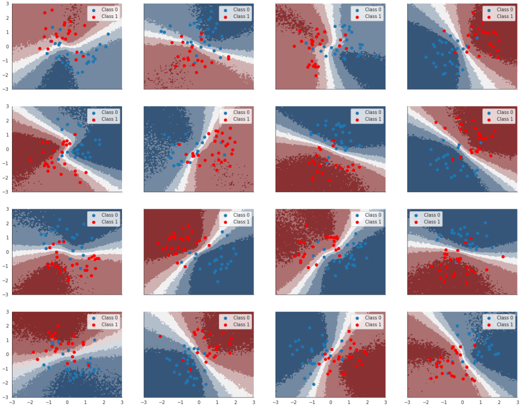
Without no rotations and abundant data, the network learns well:



16 categories



16 categories

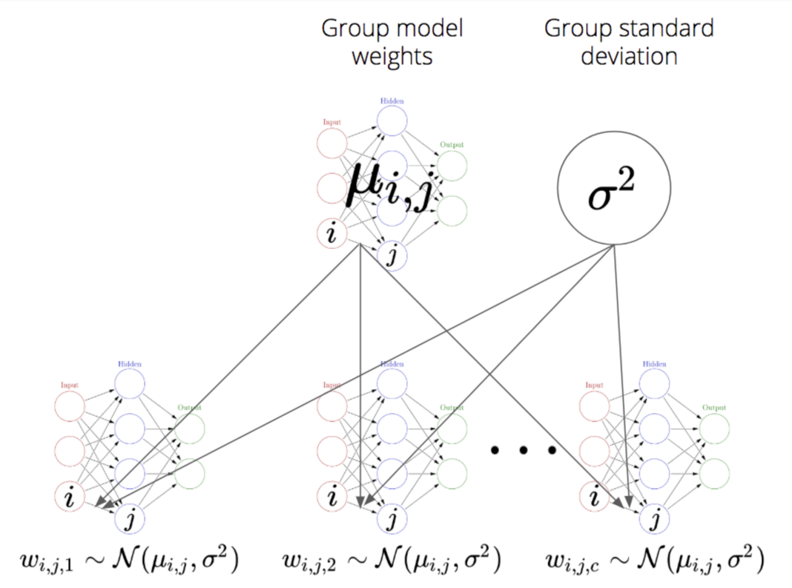


Actually about 80% accuracy on this!

Hierarchical structure

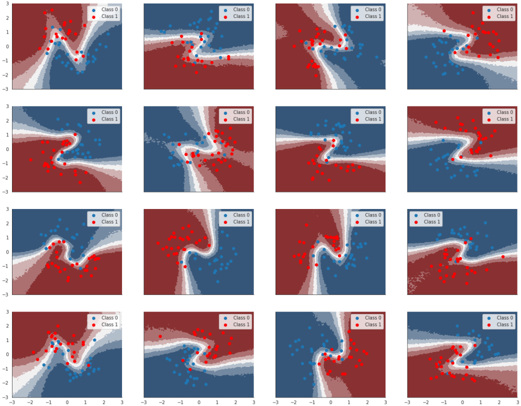
We can try to address this by adding a multilevel structure:

- Each individual weight $(w_{\ell,i,j})_n$ (layer ℓ, neuron i, component j, in network n!) is drawn from a common prior, shared across the networks
- Each prior has a single common mean and SD
- 16 different networks, but the weights are not fully independent
- The hope is that the patterns that are similar across the categories can be learned


Group model weights
Group standard deviation
Hidden
Input
Output
$\mu_{i,j}$
σ^2
i
j
$w_{i,j,1} \sim \mathcal{N}(\mu_{i,j}, \sigma^2)$
$w_{i,j,2} \sim \mathcal{N}(\mu_{i,j}, \sigma^2)$
$w_{i,j,c} \sim \mathcal{N}(\mu_{i,j}, \sigma^2)$

Hierarchical structure

After fitting using MCMC, about 90% accuracy:



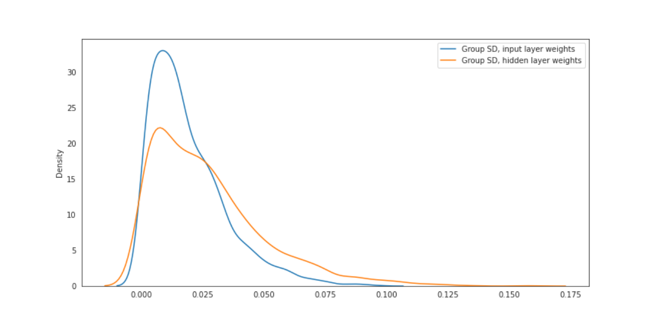
One last insight

Let's look inside the hierarchical structure a bit:

- Which parameters are being pooled?
- Which are being estimated very differently for different clusters?
- To examine this, look at posterior estimate for the group-level SDs
  - If group SD is small, all networks have similar values for these weights
  - If group SD is large, weights vary across networks

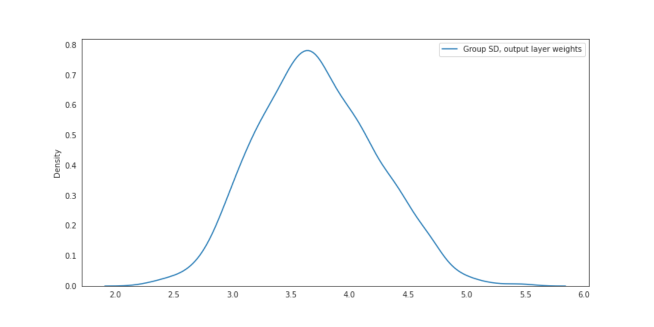
One last insight

Weights for first 2 layers:



One last insight

Weights for first 2 layers:



Summary

Today:

- Brief intro to Bayesian neural networks

Next week:

- Dirichlet processes
- Particle filters