# Pyro

Building on top of the VAE:
Recipes for missing and sequential data

UBER AI Labs

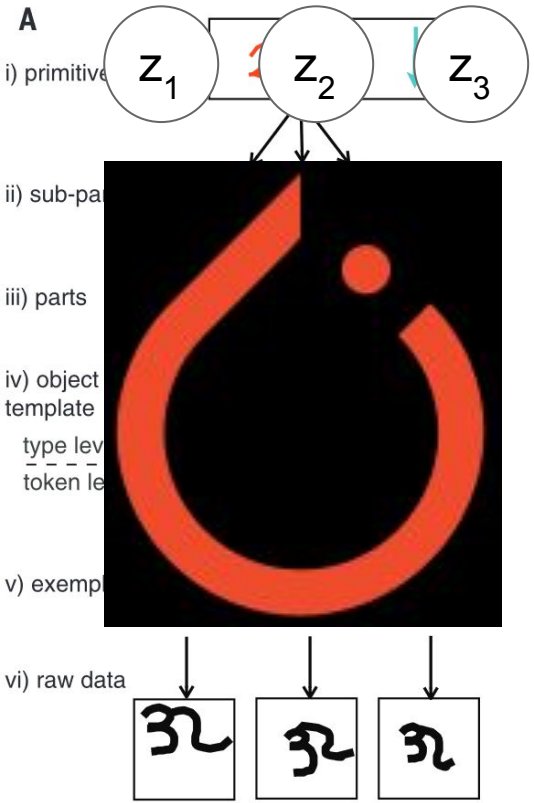
# In this tutorial:

1. Introduction to deep generative models beyond the VAE
2. Case study 1: Extending the VAE model by adding a new latent variable, and troubleshooting SVI in the extended model
3. Case study 2: Extending the VAE model by repeating it sequentially, and building up a new guide with sequential dependence structure

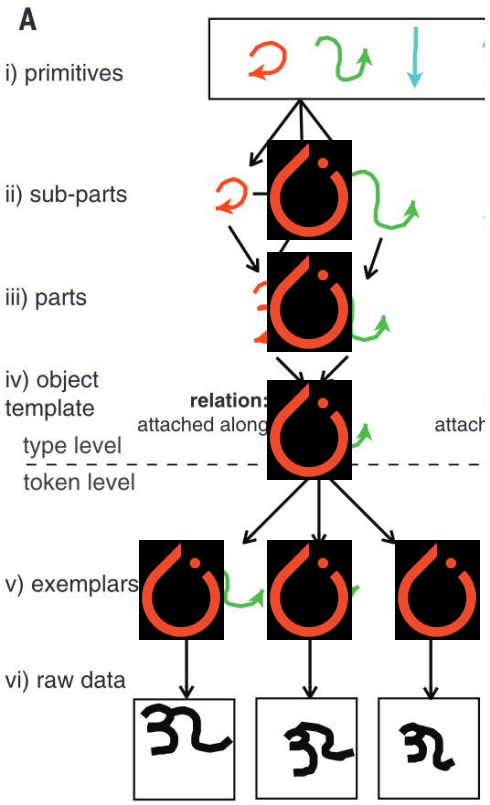
Based on the following Pyro tutorials:

- The Semi-supervised VAE: http://pyro.ai/examples/ss-vae.html
- Deep Markov Model: http://pyro.ai/examples/dmm.html

# Decoder network: the simplest deep generative model

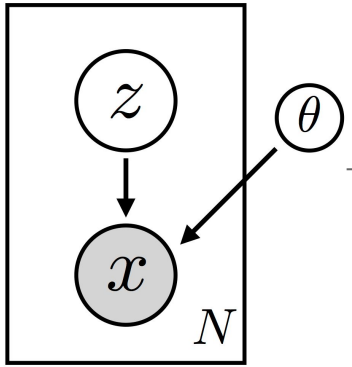


# Deep generative models beyond decoder networks



# Revisiting the VAE results

```
def model():
    pyro.module("decoder", decoder)
    z = pyro.sample("z", Normal(zeros(10), ones(10)))
    return pyro.sample("x", Bernoulli(decoder(z)))
```



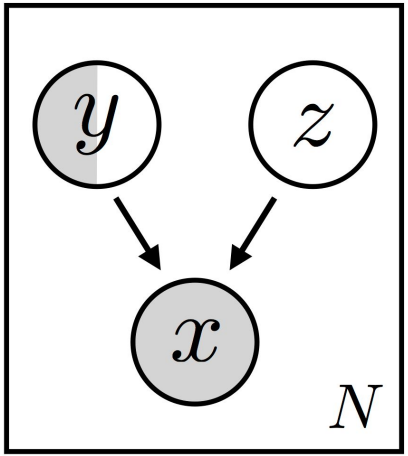

# Separating “class” and “style”

To allow the decoder latent variable z to explain only within-class variability, we add a second latent variable y that should only explain class:

# Separating “class” and “style”

To allow the decoder latent variable z to explain only within-class variability, we add a second latent variable y that should only explain class:

```
def model():
    pyro.module("decoder", decoder)
    z = pyro.sample("z", Normal(zeros(20), ones(20)))

    y = pyro.sample("y", Categorical(ones(10)))

    return pyro.sample("x", Bernoulli(decoder([z, y])))
```






*Semi-supervised Learning with Deep Generative Models*,
Diederik P Kingma, Danilo Jimenez Rezende, Shakir Mohamed, Max Welling

# Forcing the model to learn that “y” is “class”

To encourage y to explain class, we observe labels for some images:

```
def model(y=None):
    pyro.module("decoder", decoder)
    z = pyro.sample("z", Normal(zeros(20), ones(20)))

    y = pyro.sample("y", Categorical(ones(10)), obs=y)

    return pyro.sample("x", Bernoulli(decoder([z, y])))
```

# SSVAE: guide

With our new model in hand, how should we extend the guide from the VAE?

```
def guide(x):
    pyro.module("encoder", encoder)
    loc_z, scale_z = encoder(x)
    return pyro.sample("z", dist.Normal(loc_z, scale_z))
```

Can we exploit the class labels that we're already using to learn the model?

# SSVAE: guide

If we were always given a class label y, we could use that to predict style:

```
def guide(x, y):
    ...
    pyro.module("encoder_z", encoder_z)
    loc_z, scale_z = encoder_z([x, y])
    z = pyro.sample("z", Normal(loc_z, scale_z))
    return (z, y)
```

What about the case when no label is provided?

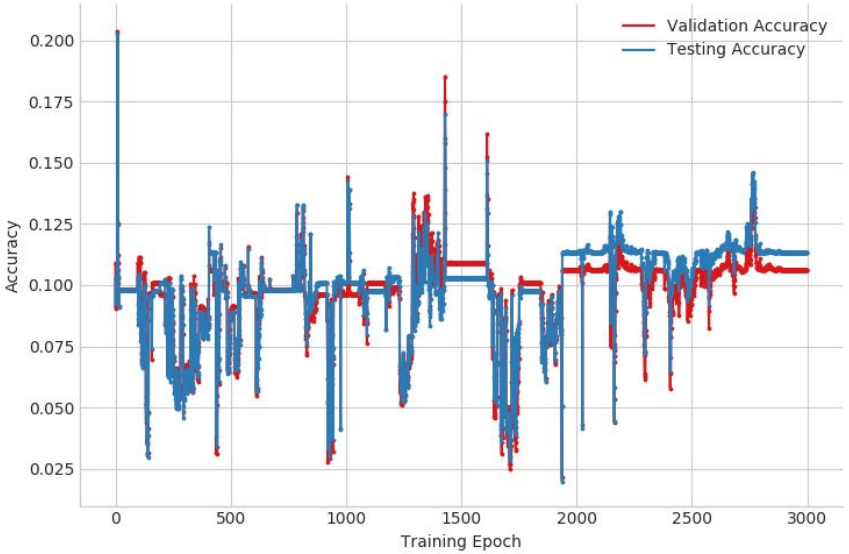
# SSVAE: guide

The guide guesses (or is given) the class y first, then uses y to estimate z:

```
def guide(x, y=None):
    ...
    if y is None:
        pyro.module("encoder_y", encoder_y)
        y = pyro.sample("y", Categorical(encoder_y(x)))
    pyro.module("encoder_z", encoder_z)
    loc_z, scale_z = encoder_z([x, y])
    z = pyro.sample("z", Normal(loc_z, scale_z))
    return (z, y)
```

# SSVAE: Inference (attempt #1)

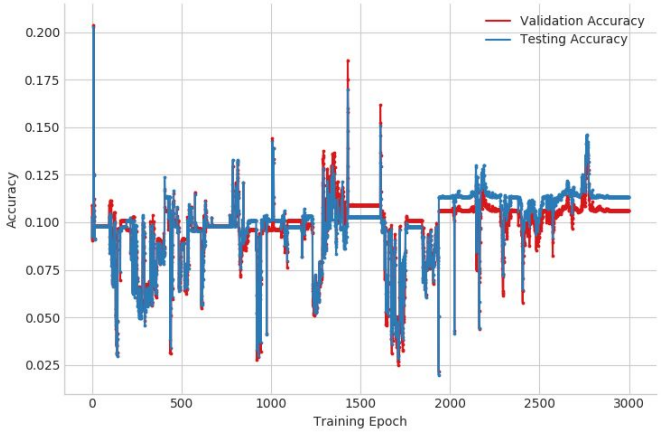
```
svi = pyro.infer.SVI(model=conditioned_model,
                     guide=guide,
                     optim=pyro.optim.Adam({"lr": 0.001}),
                     loss=pyro.infer.Trace_ELBO())

for batch_x, batch_y in batches:
    svi.step(batch_x, y=batch_y)
```




UBER

# SSVAE: Inference (attempt #1)

Why are the classification results so awful and volatile?




One possibility: the gradients it's getting are too noisy to follow

# Unobserved y: trying all possible labels in parallel

We can tell `SVI` to enumerate all possible labels instead of only guessing one:

```
@pyro.infer.config_enumerate(default="parallel")
def guide(x, y=None):
    ...
```

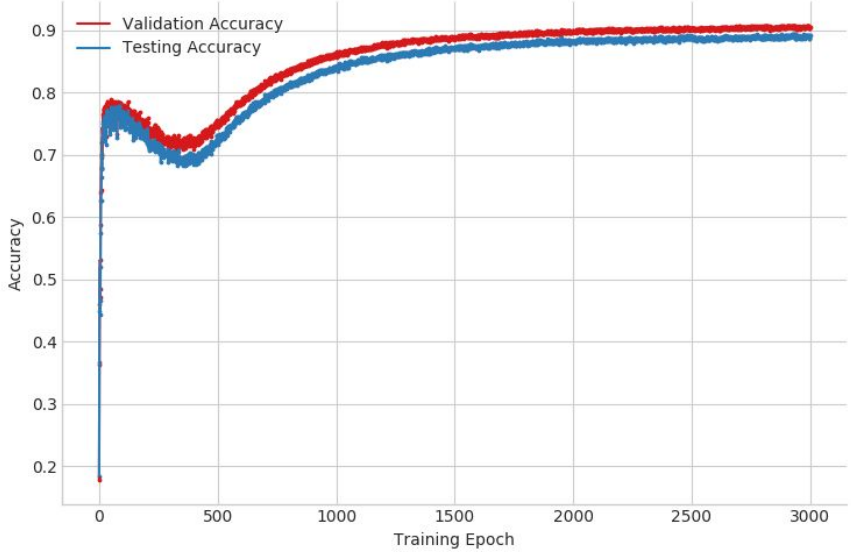
We use the enumeration-aware `TraceEnum_ELBO` to efficiently compute the loss:

```
svi = pyro.infer.SVI(..., loss=pyro.infer.TraceEnum_ELBO())
```

# SSVAE: Inference (attempt #2)

```
@pyro.infer.config_enumerate(default="parallel")
def guide(x, y=None):
    ...
svi = pyro.infer.SVI(..., loss=pyro.infer.TraceEnum_ELBO())
```

```
for batch_x, batch_y in batches:
    svi.step(batch_x, y=batch_y)
```

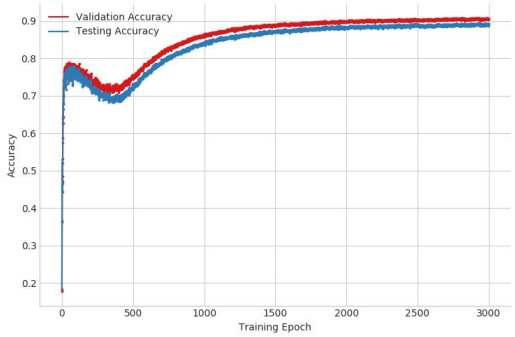



UBER

# SSVAE: Inference (attempt #2)

Classifier accuracy is much improved, but still very low for MNIST

```
def guide(x, y=None):
    ...
    if y is None:
        y = pyro.sample("y", Categorical(encoder_y(x)))
    ...
```




The classifier is never trained on the ground-truth labels!

# Observed y: training the “class” encoder with labels

We build another model that only contains the encoder:

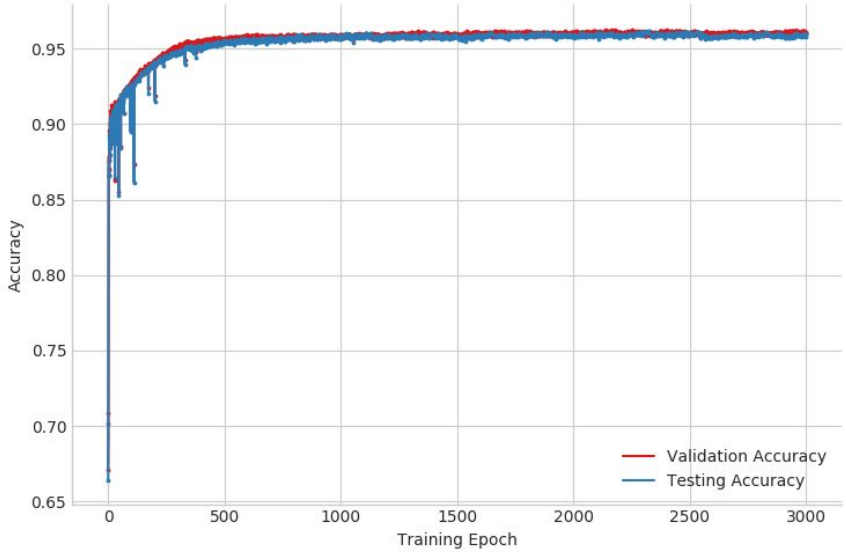
```
def model_classify(x, y=None):
    pyro.module("encoder_y", encoder_y)
    with pyro.iarange("data"):
        alpha = encoder_y(x)
        return pyro.sample("y", Categorical(alpha), obs=y)
```

Because there are no latent variables, we learn it with an empty guide:

```
guide = lambda x, y: None
svi_aux = pyro.infer.SVI(model_classify, guide, optim=Adam(), loss=Trace_ELBO())
```

# SSVAE: Inference (attempt #3)

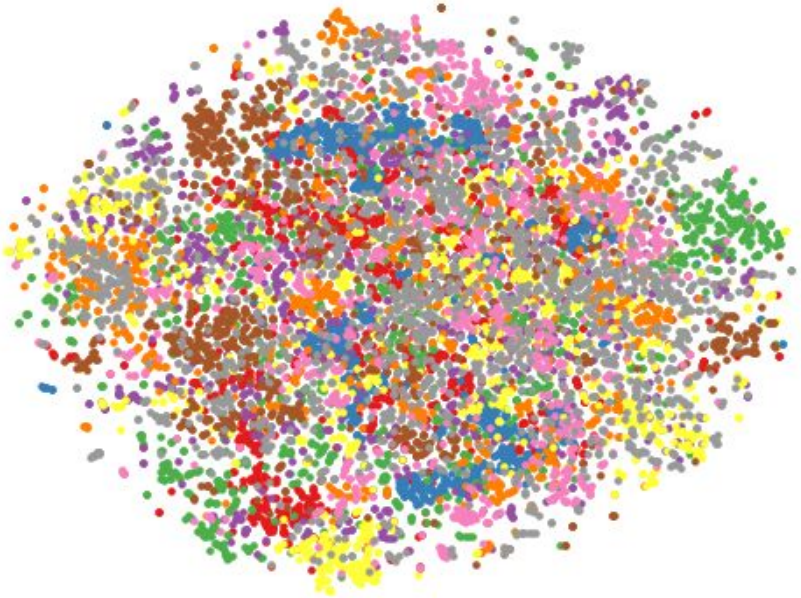
```
svi = pyro.infer.SVI(model=conditioned_model,
                     guide=pyro.infer.config_enumerate(guide, default="parallel"),
                     ...)


svi_aux = pyro.infer.SVI(model_classify, ...)

for batch_x, batch_y in batches:
    svi.step(batch_x, y=batch_y)
    if batch_y is not None:
        svi_aux.step(batch_x, batch_y)
```

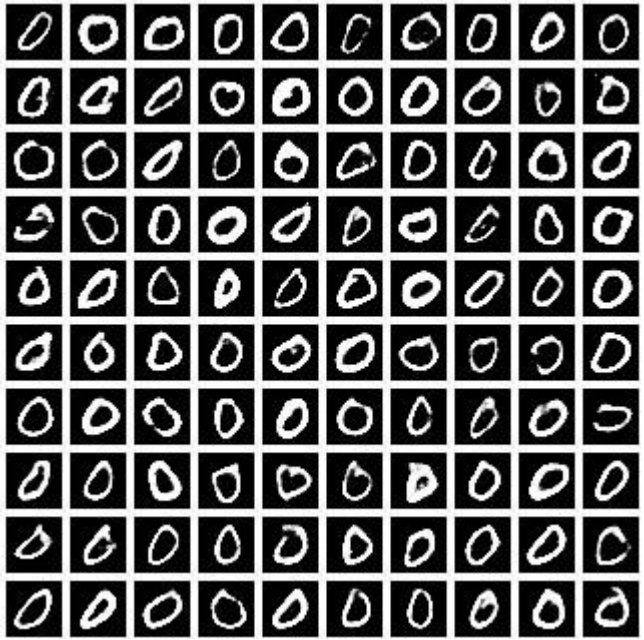



U B E R

# SSVAE: results

| Supervised data | First variant | Second variant | Third variant | Baseline classifier |
|---|---|---|---|---|
| 100 | 0.2007(0.0 353) | 0.2254(0.0 346) | 0.9319(0.0 060) | 0.7712(0.0159) |
| 600 | 0.1791(0.0 244) | 0.6939(0.0 345) | 0.9437(0.0 070) | 0.8716(0.0064) |
| 1000 | 0.2006(0.0 295) | 0.7562(0.0 235) | 0.9487(0.0 038) | 0.8863(0.0025) |
| 3000 | 0.1982(0.0 522) | 0.8932(0.0 159) | 0.9582(0.0 012) | 0.9108(0.0015) |

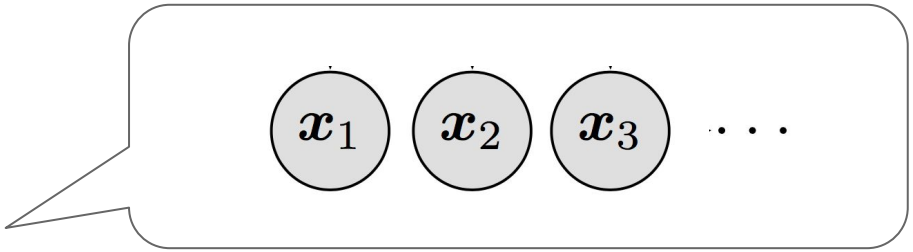




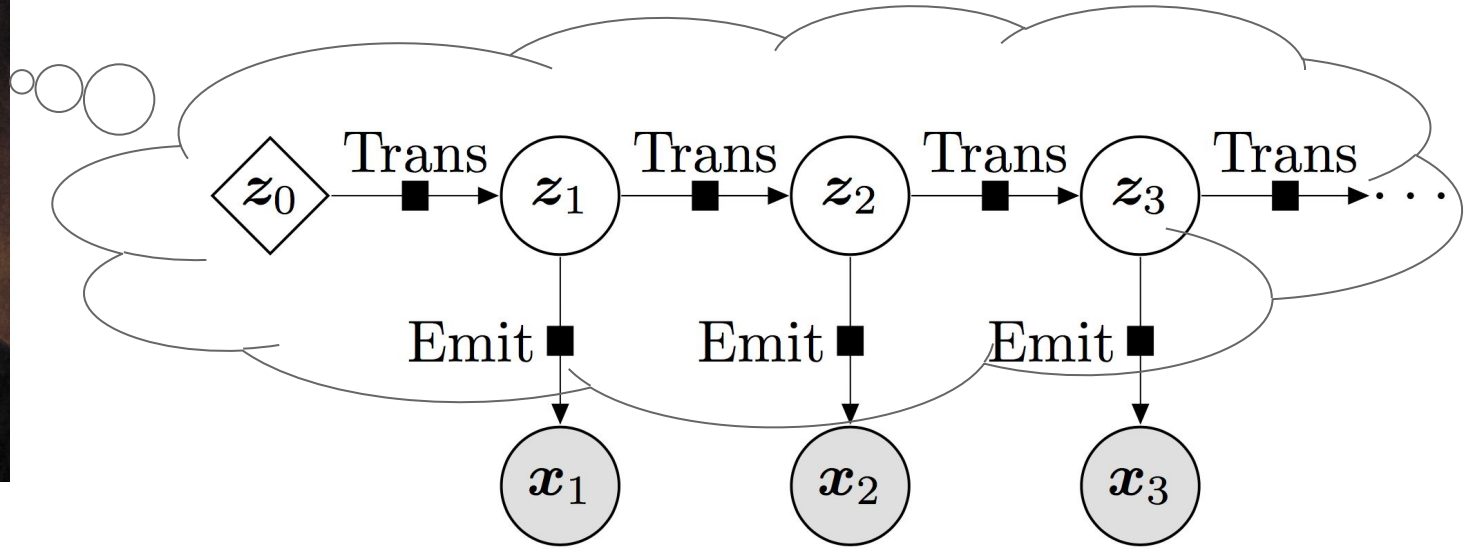
UBER

See our SSVAE tutorial for complete results

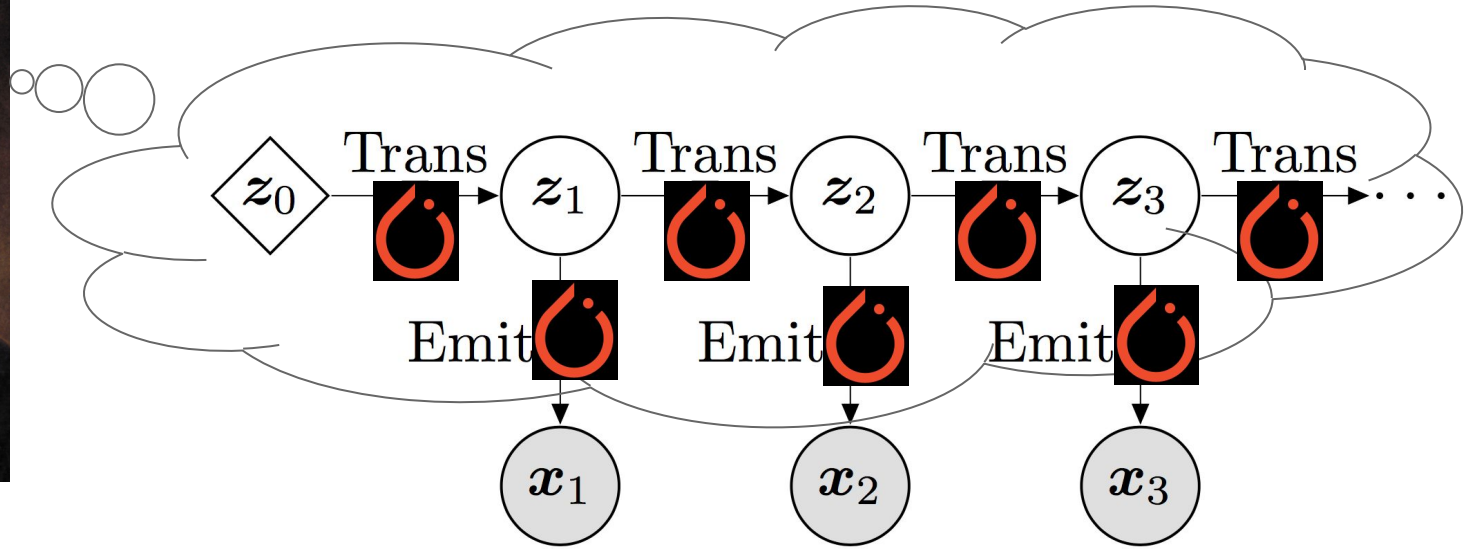
# Handling sequential data: generating MIDI music



# State Space Models



# Deep Markov Models



# VAE model, again...

Consider the decoder network model from the VAE example:

```
def vae_model():
    pyro.module("decoder", decoder)
    z = pyro.sample("z", Normal(0., 1.))
    return pyro.sample("x", Bernoulli(decoder(z)))
```

The graph suggests computing the parameters of p(z) from previous z values

# Deep Markov Models: model

We pass the prior parameters to the decoder as arguments…

```
def vae_model(z_loc, z_scale):
    ...

    z = pyro.sample("z", Normal(z_loc, z_scale))

    return z, pyro.sample("x", Normal(*decoder(z)))
```

# Deep Markov Models: model

...and compute each decoder's prior p(z | … ) from the previous z:

```
def dmm_model():
    pyro.module("transition", transition)

    z = pyro.sample("z_init", Normal(0., 1.))

    for t in range(T):
        z_loc, z_scale = transition(z)
        z, x = vae_model(z_loc, z_scale, t=t)
```

# Deep Markov Models: guide

Recall the guide we used for the VAE example:

```
def vae_guide(x):
    pyro.module("encoder", encoder)
    z_loc, z_scale = encoder(x)
    return pyro.sample("z", Normal(z_loc, z_scale))
```

How should this be expanded to capture dependence across time?

# Deep Markov Models: guide

We could provide each step with the output of the previous step...

```
def vae_guide(x, z_prev):
    ...

    z_loc, z_scale = encoder(x, z_prev)

    return pyro.sample("z", Normal(z_loc, z_scale))
```

# Deep Markov Models: guide

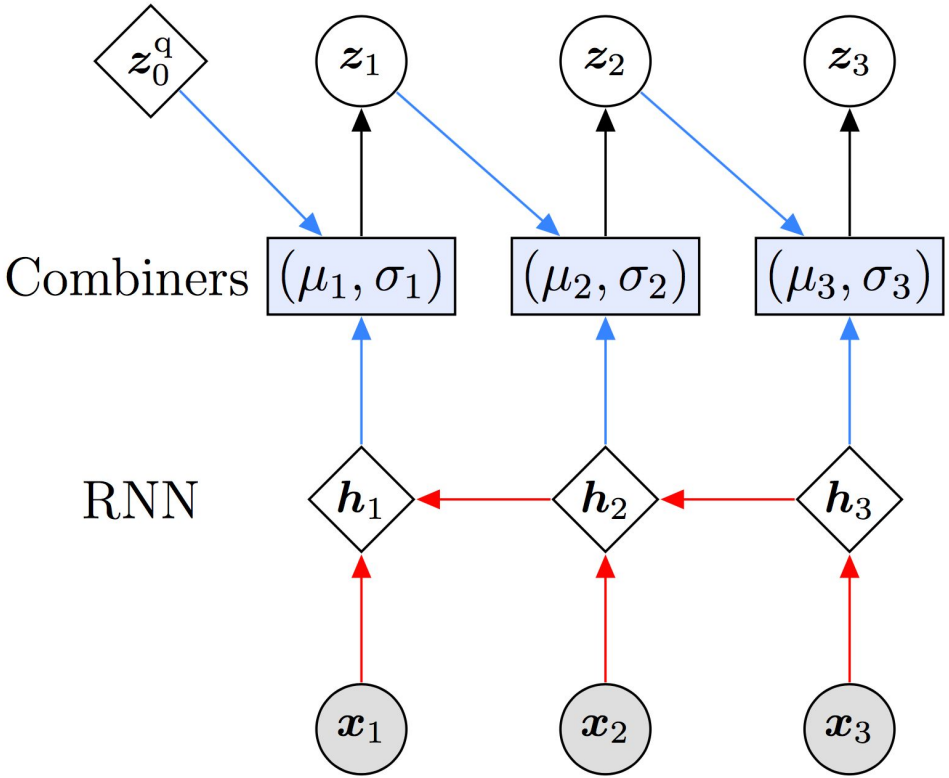
...and recall the structure of the model:

```
def dmm_model():
    ...
    for t in range(T):
        z_loc, z_scale = transition(z)
        z, x = vae_model(z_loc, z_scale, t=t)
```

This suggests calling the VAE guide in a loop of some sort:

```
def dmm_guide(xs):
    ...
    for t in ...:
        ...
        z = vae_guide(xs[t], z, t)
```

# Deep Markov Models: guide



# Deep Markov Models: guide

We pass a constant-size feature h summarizing all future observations…

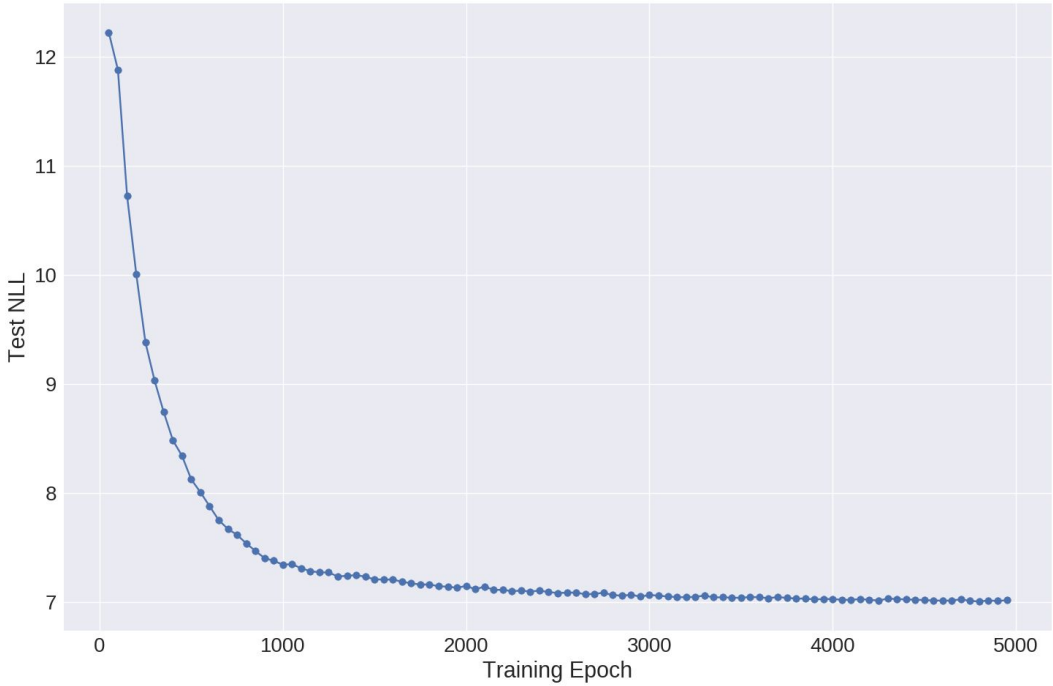
```
def vae_guide(h, z):
    ...

    z_loc, z_scale = encoder(h, z)

    return pyro.sample("z", Normal(z_loc, z_scale))
```

# Deep Markov Models: guide

...and learn the right features with a recurrent neural network:

```
def dmm_guide(xs):
    ...
    z = pyro.sample("z_init", Normal(...))

    hs = reversed(rnn(reversed(xs)))

    for t in range(T):
        ...
        z = vae_guide(hs[t], z, t)
```

# Deep Markov Models: results



# Recap

1. Introduction to deep generative models beyond the VAE
2. Case study 1: Extended the VAE model by adding a new latent variable, and did troubleshooting of SVI in the extended model
3. Case study 2: Extended the VAE model by repeating it sequentially, and built up a new guide with sequential dependence structure

# pyro.ai

Eli Bingham

JP Chen

Martin Jankowiak

Theo Karaletsos

Fritz Obermeyer

Neeraj Pradhan

Rohit Singh

Paul Szerlip

Noah Goodman

Special thanks to

Paul Horsfall
Du Phan
Soumith Chintala
Adam Paszke
Dustin Tran

# Would you like to know more?

## Pyro tutorials web page: http://pyro.ai/examples/index.html

More details on the math and implementation of stochastic variational inference in Pyro:

http://pyro.ai/examples/svi_part_ii.html and http://pyro.ai/examples/svi_part_iii.html

Detailed description of tensor and distribution shapes and broadcasting in Pyro:

http://pyro.ai/examples/tensor_shapes.html

Walkthrough of Pyro implementation of Attend-Infer-Repeat, a generative model for scenes:

http://pyro.ai/examples/air.html