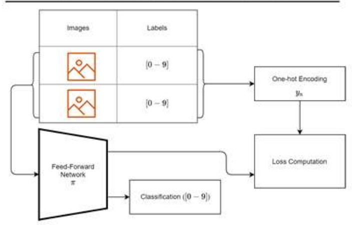
Semi-supervised Learning using Variational Autoencoders

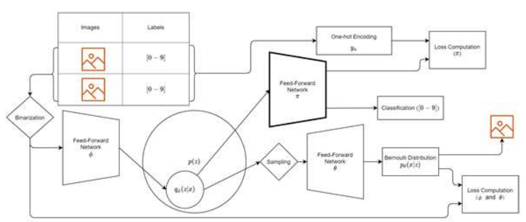
a P14 project in 02456 Deep Learning, autumn 2020
by Uffe Poul Hansen s203264 & Axel Bregnsbo s152672

The Models

Standard Feed-forward classification network



Regular VAE with classifier on latent representation (M1)



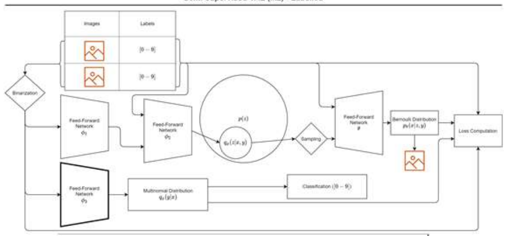
Loss Computation - ϕ, θ:

$$-\mathcal{L} = E_{q_\phi(z|x)}[\log p_\theta(x|z)] - KL[q_\phi(z|x)||p(z)]$$

Loss Computation - π:

$$\sum_{n=0}^{9} y_n \log \pi_n$$

Semi-supervised VAE (M2) - Labelled

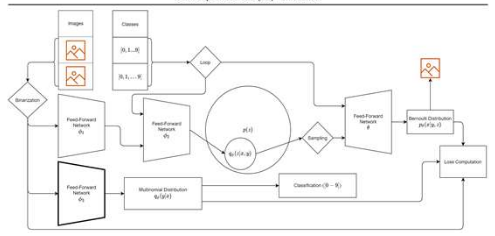


Loss computation:

$$-\mathcal{L}(x,y) = E_{q_\phi(z|x,y)}[\log p_\theta(x|y,z) + \log p_\theta(y) + \log p(z) - \log q_\phi(z|x,y)]$$

$$\mathcal{J}^\alpha = \mathcal{L} + \alpha * E_{p(x,y)}[-\log q_\phi(y|x)]$$

Semi-supervised VAE (M2) - Unlabelled



Loss computation:

$$-\mathcal{L}(x,y) = E_{q_\phi(z|x,y)}[\log p_\theta(x|y,z) + \log p_\theta(y) + \log p(z) - \log q_\phi(z|x,y)]$$

$$-\mathcal{U} = \sum_y q_\phi(y|x)(-\mathcal{L}(x,y)) + \mathcal{H}(q_\phi(y|x))$$

Introduction

Based on the article by Kingma et al. [1] we are presenting two different classifiers based on VAE's, where the VAE part enables us to train the model on non-labelled data. As we both train on labelled/supervised and on unlabeled/unsupervised, it is called semi-supervised learning. These VAE classifiers are then compared to a classical FF model which can only be trained on labelled data.

M1: VAE + classifier that is using the much smaller latent space instead of the full image
M2: VAE where the continuous z latent space is augmented with a discreet y class variable. The network generating the p(y|z) posterior is used as classifier.
FF: a reference classifier used for comparison. It has the same number of weights as the parts of M1 and M2 that are used for classification.

The M2 model has the interesting property that is enables style transfer: with given input x, one can hold z and vary y, and get digit pictures of other numbers, that have a style like the reference picture, see lower right corner.

[1]: Kingma et al. (2014). Semi-supervised Learning with Deep Generative Models, arxiv.org/abs/1406.5298v2

Summary

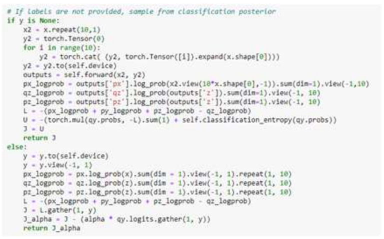
With just 100 labelled training data, and 40k unlabeled, we achieved 94% accuracy on the MNIST test set using the M2 VAE classifier, and thus substantially beating an FF classifier having the same size of weights, which could only achieve 73% accuracy.

The initial functional M2 was only giving 78% accuracy with 100/40k training, not much better than 73% from FF. Two changes were made giving a large improvement:
- Changing the alpha coefficient, which weights the labelled classifier loss. It was initially 0.1, which is the value used Kingma in [1], and we changed it to 0.5
- Making a forward pass with both labelled and unlabeled training data, before doing a back prop & coefficient update, instead of having a separate backward pass for labelled and for unlabeled.

Our focus was getting M1 & M2 to work and therefore only did limited experimentation with the hyper parameters of the internal neural networks. These are all using FFNN, ReLU & BatchNorm1d.

We fell into quite some PyTorch and model-building bumps during our journey. Especially M2 was tricky to get to work. Some of the pain points:
- Wrongly having ReLU activation function on the network calculating the posterior my,sigma
- M2 training ending in NaN after a few epochs due to sub-parts of the loss function having wrong sign
- Applying softmax() along wrong dimension
- M1 encoder weights were not frozen during classifier training.
- Getting too good classifier results because, because a fresh set of labelled training set were wrongly introduced when doing an incremental training.

Heart of the M2



```python
# If labels are not provided, sample from classification posterior
if y is None:
    x2 = x.repeat(10,1)
    y2 = torch.Tensor(0)
    for i in range(10):
        y2 = torch.cat( (y2, torch.Tensor([i]).expand(x.shape[0])))
    y2 = y2.to(self.device)
    outputs = self.forward(x2, y2)
    px_logprob = outputs['px'].log_prob(x2.view(10*x.shape[0],-1)).sum(dim=1).view(-1,10)
    qz_logprob = outputs['qz'].log_prob(outputs['z']).sum(dim=1).view(-1, 10)
    pz_logprob = outputs['pz'].log_prob(outputs['z']).sum(dim=1).view(-1, 10)
    L = -(px_logprob + py_logprob + pz_logprob - qz_logprob)
    U = -(torch.mul(qy.probs, -L).sum(1) + self.classification_entropy(qy.probs))
    J = U
    return J
else:
    y = y.to(self.device)
    y = y.view(-1, 1)
    px_logprob = px.log_prob(x).sum(dim = 1).view(-1, 1).repeat(1, 10)
    qz_logprob = qz.log_prob(z).sum(dim = 1).view(-1, 1).repeat(1, 10)
    pz_logprob = pz.log_prob(z).sum(dim = 1).view(-1, 1).repeat(1, 10)
    L = -(px_logprob + py_logprob + pz_logprob - qz_logprob)
    J = L.gather(1, y)
    J_alpha = J - (alpha * qy.logits.gather(1, y))
    return J_alpha
```

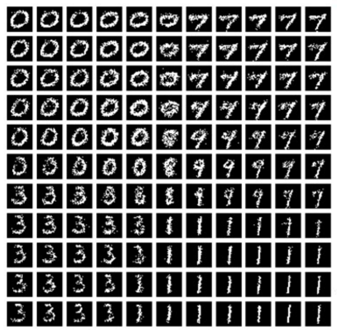
Python implementation of the M2 loss calculation.

Model Parameters

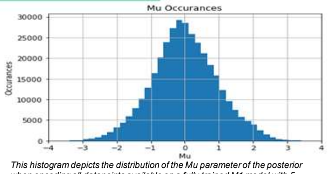
Model	Structure	Classification Parameters	Additional Parameters
FF NN	Classifier: 784 – 500 – 250 - 10	520250	0
M1	Encoder: 784 – 450 – 250 – 100 – 100 Classifier: 100 – 100 – 80 – 10 Decoder: 50 – 128 – 256 – 512 - 784	520090	572544
M2	Encoder 1: 784 – 200 Encoder 2: 210 – 200 – 10 Classifier: 784 – 500 – 250 – 10 Decoder: 15 – 200 – 400 – 600	520250	996800

Modelling of the various network components. All sub networks are FCFF, using ReLU activation function in the hidden layers. Sigmoid is used as activation function in the output layer of FF and M1 classifier.

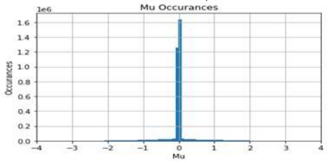
Inside the VAE



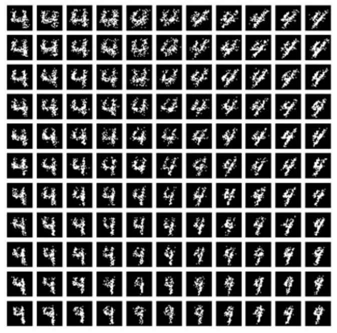
This illustration shows renditions of various points in latent space of an M1 model. The depicted model uses 5-dimensional latent space. Three dimensions are locked to 0, while the remaining two dimensions are traversed from -5 to 5.



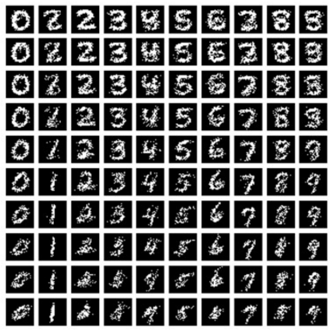
This histogram depicts the distribution of the Mu parameter of the posterior when encoding all datapoints available on a fully trained M1 model with 5 latent dimensions. The regularization term attempts to force these towards 0, while the reconstruction term demands a certain spread.



Same as above, but with 50 latent dimensions. We can see that the increased number of dimensions allows a tighter gathering around 0.

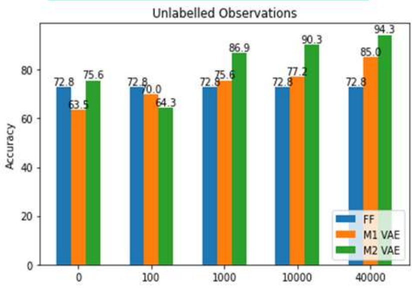


This illustration shows renditions of various points in latent space of an M2 model. The depicted model uses 5-dimensional latent space. Three dimensions are locked to 0, while the remaining two dimensions are traversed from -5 to 5. For each coordinate, the digit "4" is rendered.



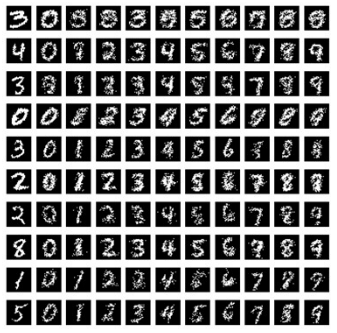
This illustration shows renditions of various points in latent space of an M2 model. The depicted model uses 5-dimensional latent space. A 5-dimensional diagonal is traversed from -5 to 5. For each point, all digits are rendered.

Classifier Results



Learning from unlabelled data. On the graph above, we show the achived accuracies when training each of the models on 100 labelled samples and increasing numbers of unlabelled samples. Naturally, the standard FFNN does not support semisupervised learning, and is not trained on the unlabelled data, so the result is constant. The complexity of the variational model penalizes the network when compared to a standard FFNN when identical data is available, but the ability to improve the model from unlabelled data quickly pays off. The M2 VAE performing better with no unlabelled datapoints than with 100 unlabelled datapoints is a curiosity we have yet to explain.

Style Transfer



This illustration shows attempts to copy the style from a provided digit. Posterior distribution is derived from provided image (left most column, and each digit is recreated using a sample from the posterior.