Model Compression Techniques & VAE

Gopi Kishan
GCDSL, IISc

Overview

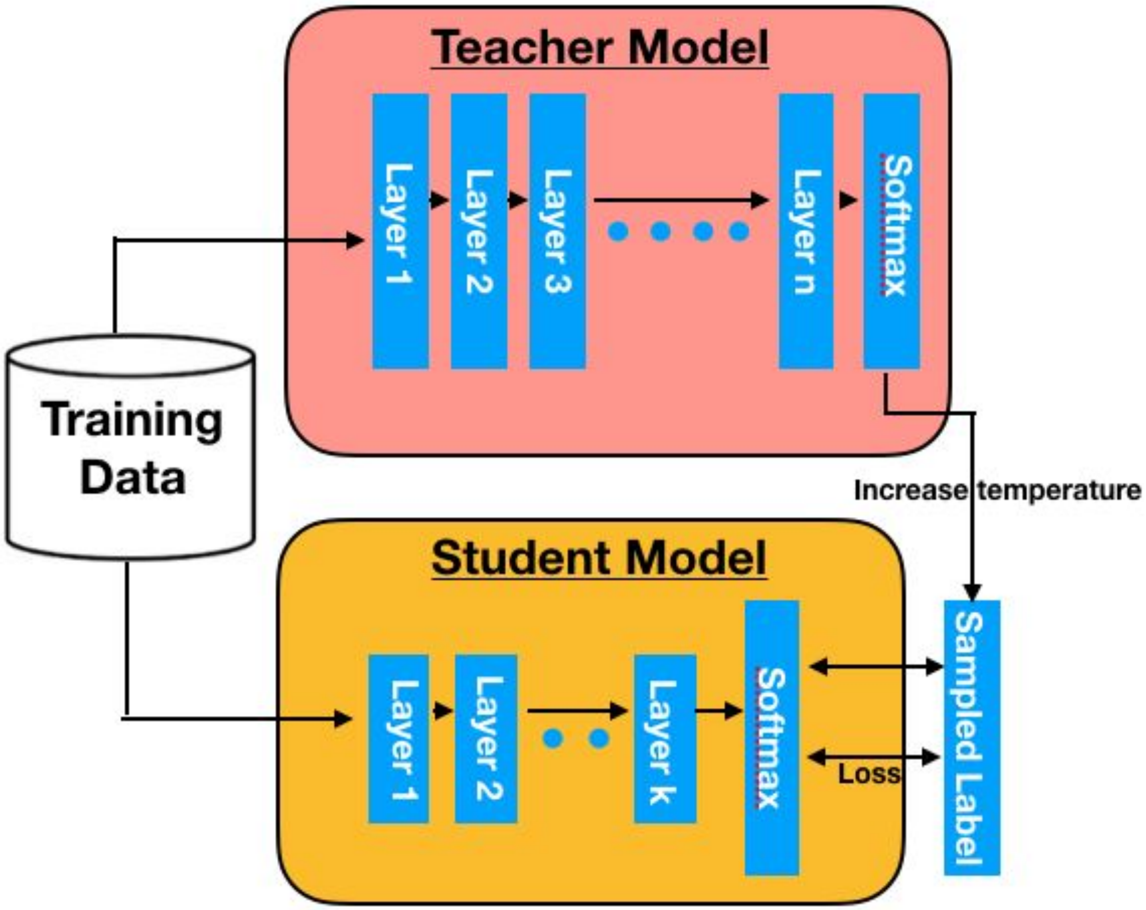
1. Knowledge Distillation (KD)
2. Auto Encoders
3. Variational AutoEncoders (VAE)
4. Exploring other prior of VAE
5. Other Compression techniques
6. Micronet Challenge hosted at NeurIPS 2019

Distilling the Knowledge in a Neural Network

Neural networks typically produce class probabilities by using a “softmax” output layer that converts the logit, zi , computed for each class into a probability, qi , by comparing zi with the other logits.

$$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)}$$

where T is a temperature that is normally set to 1. Using a higher value for T produces a softer probability distribution over classes.


Teacher Model
Layer 1
Layer 2
Layer 3
Layer n
Softmax
Training Data
Increase temperature
Student Model
Layer 1
Layer 2
Layer k
Softmax
Loss
Sampled Label

Matching logits is a special case of distillation

The cross-entropy gradient, dC/dzi is given by :

$$\frac{\partial C}{\partial z_i} = \frac{1}{T}\left(q_i - p_i\right) = \frac{1}{T}\left(\frac{e^{z_i/T}}{\sum_j e^{z_j/T}} - \frac{e^{v_i/T}}{\sum_j e^{v_j/T}}\right)$$

If the temperature is high compared with the magnitude of the logits, we can approximate:

$$\frac{\partial C}{\partial z_i} \approx \frac{1}{T}\left(\frac{1 + z_i/T}{N + \sum_j z_j/T} - \frac{1 + v_i/T}{N + \sum_j v_j/T}\right)$$

If we now assume that the logits have been zero-meaned separa P tely for each transfer case so that $\sum_j z_j = \sum_j v_j = 0$

$$\frac{\partial C}{\partial z_i} \approx \frac{1}{NT^2}\left(z_i - v_i\right)$$

Knowledge Distillation on MNIST

Teacher Model

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 28, 28, 1)]	0
Conv1 (Conv2D)	(None, 26, 26, 32)	320
Conv2 (Conv2D)	(None, 24, 24, 64)	18496
MaxPool (MaxPooling2D)	(None, 12, 12, 64)	0
Dropout1 (Dropout)	(None, 12, 12, 64)	0
Flat (Flatten)	(None, 9216)	0
FC1 (Dense)	(None, 128)	1179776
Dropout2 (Dropout)	(None, 128)	0
logits (Dense)	(None, 10)	1290
Softmax (Activation)	(None, 10)	0

Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0

Student Model

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	(None, 28, 28, 1)	0
flatten (Flatten)	(None, 784)	0
FC1 (Dense)	(None, 128)	100480
FC2 (Dense)	(None, 128)	16512
logits (Dense)	(None, 10)	1290
Softmax (Activation)	(None, 10)	0

Total params: 118,282
Trainable params: 118,282
Non-trainable params: 0

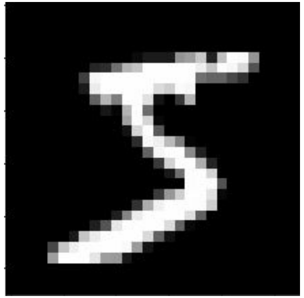
Prediction over MNIST

All the models are trained for 10 epoches with batch size = 64

Models	Test Accuracy
Teacher Model	0.9924
Student Model	0.9871
Student Model without teacher guidance	0.9632

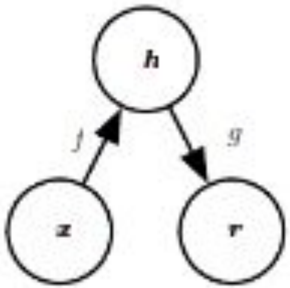
Probability distribution over number at Temperature value = 4

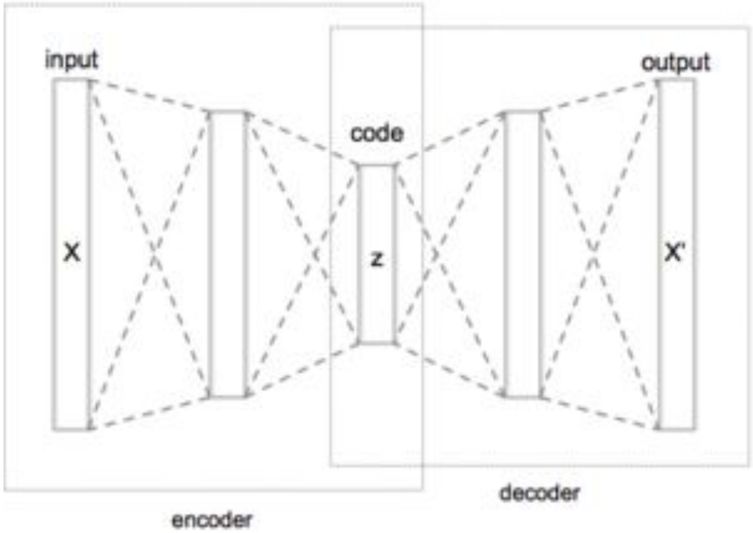
```
Unsoftened probabilities:  [2.9399757e-23 7.7865502e-24 3.3027068e-24
6.2618165e-16 1.4026793e-27
 7.1093457e-12 2.4744464e-21 3.6832551e-24 9.0432768e-20 5.3667499e-19]

Prediction based on unsoftened probability for:  5
```



```
Softened probabilities:  [5.1947024e-09 3.7265848e-09 3.0074072e-09
3.5289850e-07 4.3173209e-10
 3.6427730e-06 1.5734186e-08 3.0905281e-09 3.8686210e-08 6.0381318e-08]

Prediction based on Softened probability for:  5
```

AutoEncoders





The learning process is described simply as minimizing a loss function

$$L(\boldsymbol{x}, g(f(\boldsymbol{x})))$$

Where L is a loss function penalizing g(f(x)) for being dissimilar from x, such as the mean squared error is given by :

$$\min_{\mathbf{w}_1, \mathbf{w}_2} \|\mathbf{x}_i - g(f(\mathbf{x}_i; \mathbf{w}_1); \mathbf{w}_2)\|_2^2$$

Results on MNIST

Model description

Layer (type)	Output Shape	Param #
input_14 (InputLayer)	(None, 28, 28, 1)	0
conv2d_22 (Conv2D)	(None, 28, 28, 16)	160
max_pooling2d_10 (MaxPooling	(None, 14, 14, 16)	0
conv2d_23 (Conv2D)	(None, 14, 14, 8)	1160
max_pooling2d_11 (MaxPooling	(None, 7, 7, 8)	0
conv2d_24 (Conv2D)	(None, 7, 7, 8)	584
max_pooling2d_12 (MaxPooling	(None, 4, 4, 8)	0
conv2d_25 (Conv2D)	(None, 4, 4, 8)	584
up_sampling2d_10 (UpSampling	(None, 8, 8, 8)	0
conv2d_26 (Conv2D)	(None, 8, 8, 8)	584
up_sampling2d_11 (UpSampling	(None, 16, 16, 8)	0
conv2d_27 (Conv2D)	(None, 14, 14, 16)	1168
up_sampling2d_12 (UpSampling	(None, 28, 28, 16)	0
conv2d_28 (Conv2D)	(None, 28, 28, 1)	145

Total params: 4,385
Trainable params: 4,385
Non-trainable params: 0

Layer (type)	Output Shape	Param #
input_7 (InputLayer)	(None, 784)	0
dense_5 (Dense)	(None, 128)	100480
dense_6 (Dense)	(None, 784)	101136

Total params: 201,616
Trainable params: 201,616
Non-trainable params: 0

Loss = Cross_Entropy + alpha * MSE(Encoded_S - Endoded_T)*temp2**

At training temperature = 4 and alpha = 0.5

Models	Result
Teacher Model	
DNN	
DNN with teacher guidance	

MNIST Image sharpening with temp term

Recall

$$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)}$$

Original Image		
AE reconstructed Image		
AE reconstruction using temp = 0.6		

Maths review

- Bayes' Theorem

$$\underbrace{P(A|B)}_{\text{posterior}} = \frac{P(B|A)\,\overbrace{P(A)}^{\text{prior}}}{P(B)}$$

- KL Divergence

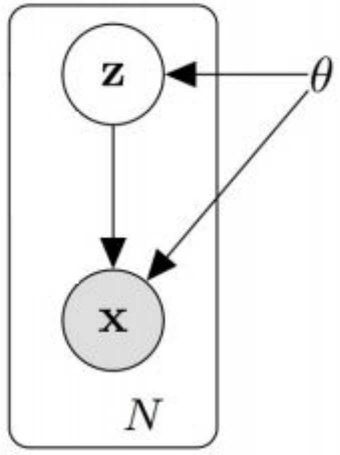
$$D_{KL}(Q||P) = \mathbb{E}_{x\sim Q}[\log \frac{Q(x)}{P(x)}]$$

Variational Autoencoder

Variational autoencoders (VAEs) are a deep learning technique for learning latent representations.

Let's define some notions:

1. X : data that we want to model
2. z: latent variable
3. P(X) : probability distribution of the data
4. P(z) : probability distribution of latent variable
5. P(X|z) : distribution of generating data given latent variable

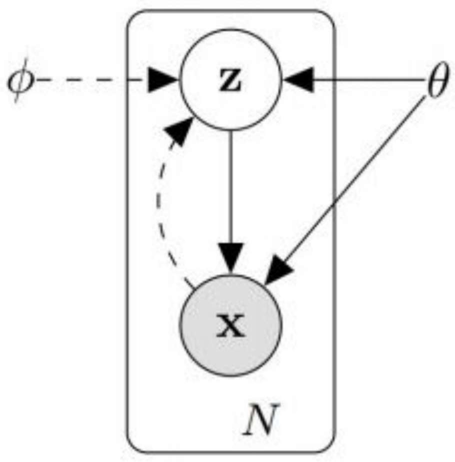


Goal: To compute the distribution P(z|X)

Alright, now let’s say we want to infer P(z|X) using Q(z|X). The KL divergence then formulated as follows:

$$D_{KL}[Q(z|X)\|P(z|X)] = \sum_z Q(z|X) \log \frac{Q(z|X)}{P(z|X)}$$

$$= E\left[\log \frac{Q(z|X)}{P(z|X)}\right] \quad = E[\log Q(z|X) - \log P(z|X)]$$



Applying Bayes’ rule :

$$D_{KL}[Q(z|X)\|P(z|X)] = E\left[\log Q(z|X) - \log \frac{P(X|z)P(z)}{P(X)}\right]$$

$$D_{KL}[Q(z|X)\|P(z|X)] - \log P(X) = E[\log Q(z|X) - \log P(X|z) - \log P(z)]$$

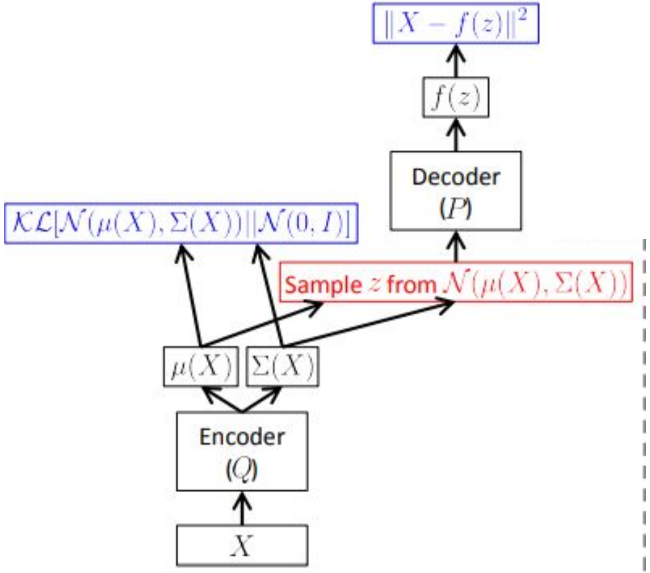
$$\log P(X) - D_{KL}[Q(z|X)\|P(z|X)] \quad = E[\log P(X|z)] - E[\log Q(z|X) - \log P(z)]$$

And this is it, the VAE objective function:

$$\log P(X) - D_{KL}[Q(z|X) \| P(z|X)] = E[\log P(X|z)] - D_{KL}[Q(z|X) \| P(z)]$$

At this point, what do we have? Let's enumerate:

1. Q(z|X) that project our data X into latent variable space
2. z, the latent variable
3. P(X|z) that generate data given latent variable.

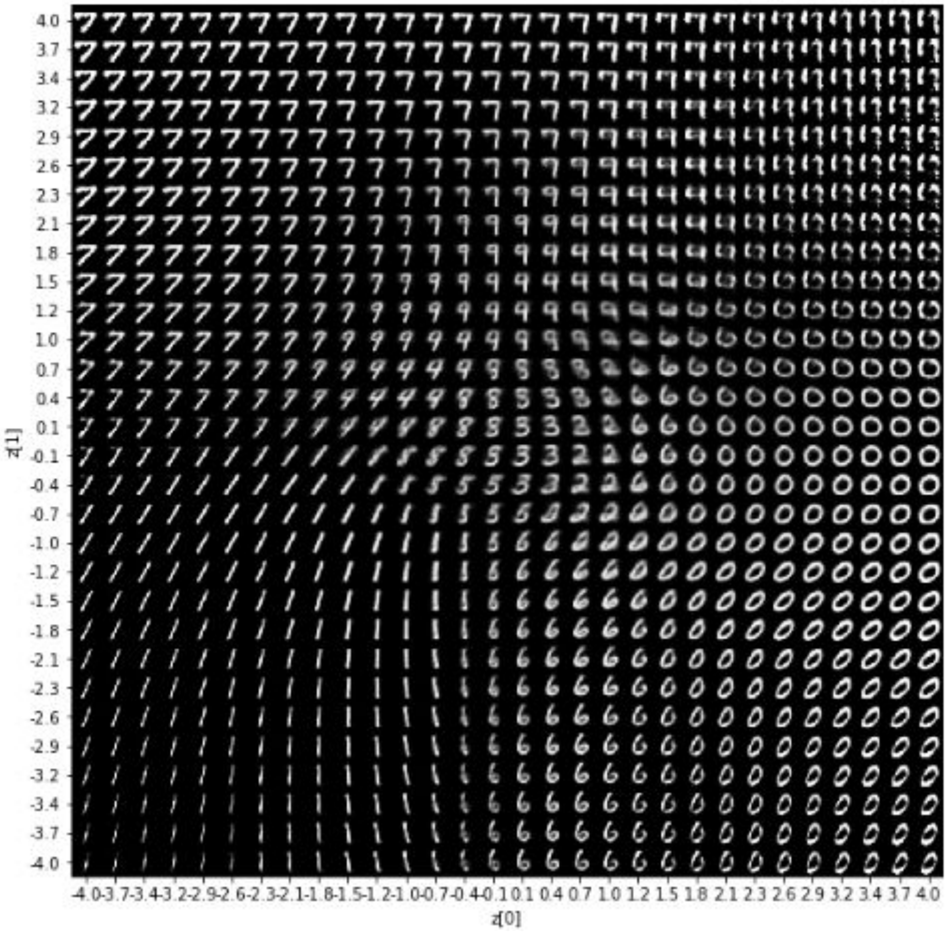


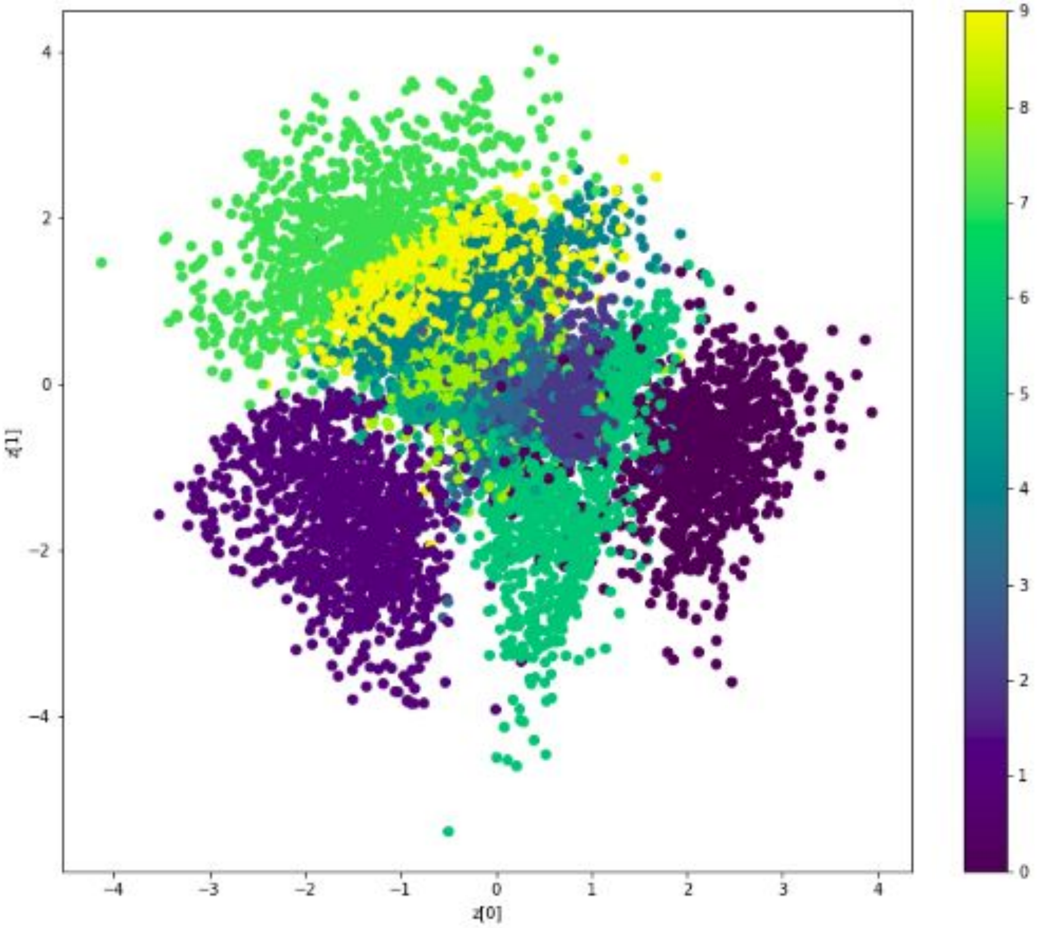
Result on MNIST

```
Layer (type)                 Output Shape              Param #
=================================================================
encoder_input (InputLayer)   (None, 784)               0

dense_1 (Dense)              (None, 512)               401920

z_mean (Dense)               (None, 2)                 1026

z_log_var (Dense)            (None, 2)                 1026

z (Lambda)                   (None, 2)                 0

=================================================================
Total params: 403,972
Trainable params: 403,972
Non-trainable params: 0
```

z = z_mean + sqrt(var) * epsilon

Where, epsilon = N(0,1)





4
2
0
−2
−4
z[1]
−4
−3
−2
−1
0
1
2
3
4
z[0]
9
8
7
6
5
4
3
2
1
0

Exploring other priors in VAE

Current prior : bernoulli distribution and Gaussian distribution.

Motivation : Image pixel values follow multinomial distribution. So none of the above priors are suitable.

$$\underbrace{P(A|B)}_{\text{posterior}} = \frac{P(B|A)\overbrace{P(A)}^{\text{prior}}}{P(B)}$$

Proposal : Dirichlet Distribution

Dirichlet Distribution

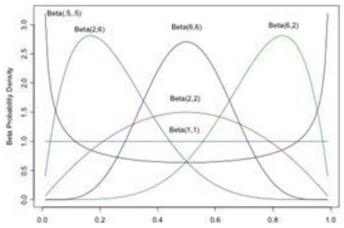
Dirichlet distribution is a multivariate distribution with parameters α=[α1,α2,...,αK], with the following probability density function

$$p(x;\alpha) = \frac{\Gamma(\sum_{k=1}^{K}\alpha_k)}{\prod_{k=1}^{K}\Gamma(\alpha_k)}\prod_{k=1}^{K} x_k^{\alpha_k - 1}$$

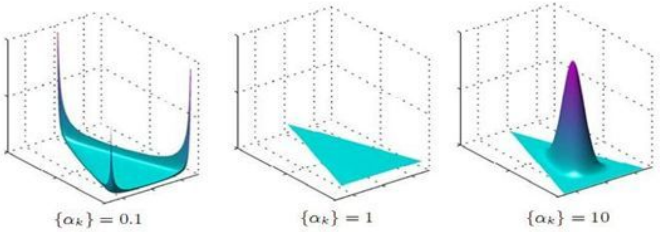
Beta distribution $$f(\theta;\ \alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\theta^{\alpha-1}(1-\theta)^{\beta-1}$$

Dirichlet distribution $$f(\theta_1,\dots,\theta_k\,;\alpha_1,\dots,\alpha_k) = \frac{\Gamma(\alpha_1+\dots+\alpha_k)}{\Gamma(\alpha_1)\dots\Gamma(\alpha_k)}\theta_1^{\alpha_1}\dots\theta_k^{\alpha_k}$$

It is a multivariate form of Beta distribution and conjugate prior to multinomial distribution.



Beta



Dirichlet

Dirichlet VAE

Instead of using the standard Gaussian distribution, we use the Dirichlet distribution which is a conjugate prior distribution of the multinomial distribution.

$$\mathbf{z} \sim p(\mathbf{z}) = \text{Dirichlet}(\boldsymbol{\alpha}), \ \mathbf{x} \sim p_\theta(\mathbf{x}|\mathbf{z})$$

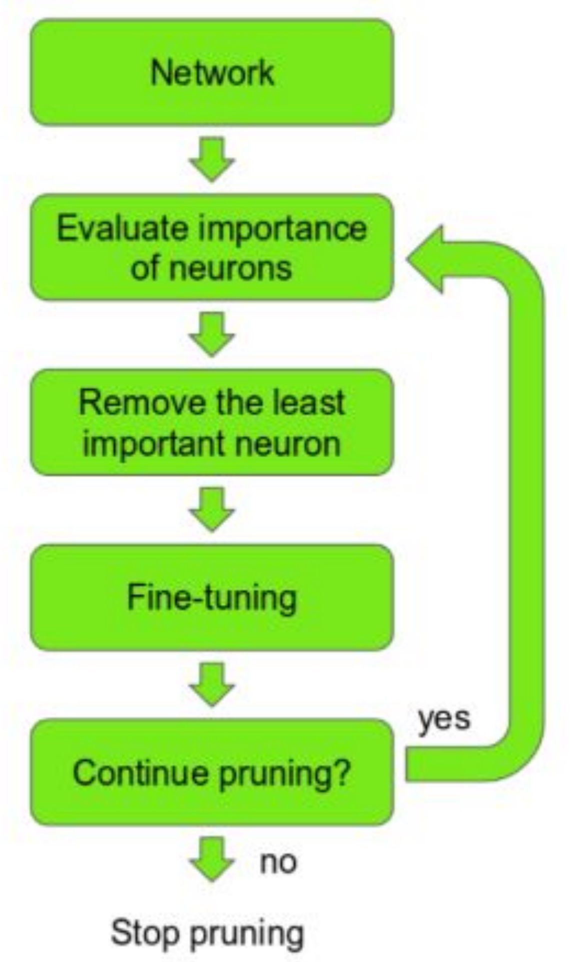
A paper with the same idea was submitted in ICML 2019, but unfortunately was not accepted. It proposes DirVAE advantages as :

1. DirVAE models the latent representation result with the best log-likelihood compared to the baselines
2. DirVAE produces more interpretable latent values with no collapsing issues which the baseline models suffer from.
3. Better performance on MNIST and OMNIGLOT compared to the baseline VAEs

Other Compression techniques :

1. Pruning

Deciding importance of neuron :
The ranking can be done according to the **L1/L2 mean of neuron weights**, their mean activations, the number of times a neuron wasn't zero on some validation set, and other creative methods .



2. Quantisation

Key Idea: The value distribution of neural network weight is of small range, which is very close to 0.
With such value range in (−1,1), quantizing floating point is mapping FP32 to INT8 using

$$x_{float} = x_{scale} \times (x_{quantized} - x_{zero\_point})$$

$$x_{float} \in [x_{float}^{min}, x_{float}^{max}]$$

$$x_{scale} = \frac{x_{float}^{max} - x_{float}^{min}}{x_{quantized}^{max} - x_{quantized}^{min}}$$

$$x_{zero\_point} = x_{quantized}^{max} - x_{float}^{max} \div x_{scale}$$

$$x_{quantized} = x_{float} \div x_{scale} + x_{zero\_point}$$

where , x_float denotes the FP32 weight,
x_quantized denotes the quantized INT8 weight, and
x_scale is the mapping fractor (scaling factor).

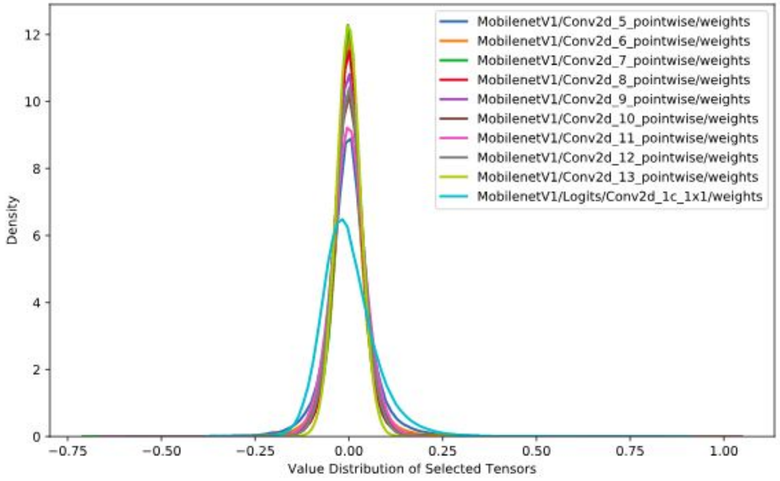


Figure 8 shows weight distribution of 10 layers (layers that have most value points) of MobileNetV1.

3. Weight matrix factorization using SVD

Singular Value Decomposition of any matrix is $\mathbf{A}_{nxp} = \mathbf{U}_{nxn} \mathbf{S}_{nxp} \mathbf{V}^{\mathbf{T}}_{pxp}$

where the columns of U are the left singular vectors,
S has singular values and is diagonal,
and V^T has rows that are the right singular vectors.

$\mathbf{U}^{\mathrm{T}}\mathbf{U} = \mathbf{I}_{\mathrm{nxn}}$

$\mathbf{V}^{\mathrm{T}}\mathbf{V} = \mathbf{I}_{\mathrm{pxp}}$ (i.e. U and V are orthogonal)

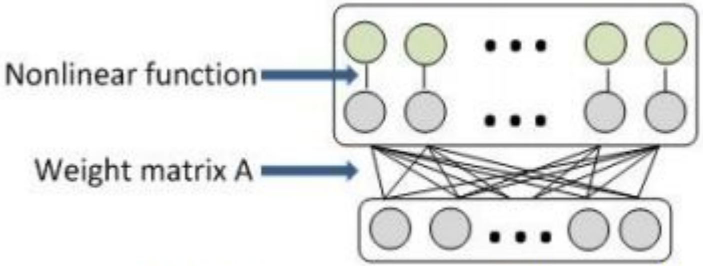
For a weight matrix A, if we apply SVD on it, we get

$$A_{m\times n} = U_{m\times n}\Sigma_{n\times n}V^T_{n\times n}$$

where ∑ is a diagonal matrix with A's singular values on the diagonal in the decreasing order.

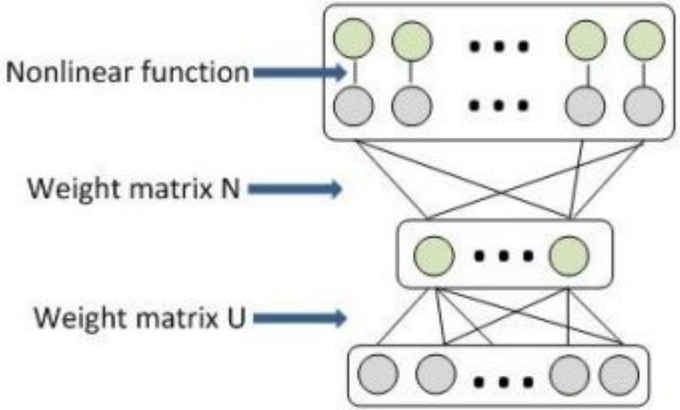
$$A_{m\times n} = U_{m\times k}\Sigma_{k\times k}V^T_{k\times k} = U_{m\times k}N_{k\times k}$$

In this way we decompose matrix A into two smaller matrices U and N. For one single layer in a DNN model, we replace it with two layers, while the first one has no nonlinear function, and the second one does.



(a) One layer in original DNN model

$$A_{m\times n} = U_{m\times n}\Sigma_{n\times n}V^{T}_{n\times n}$$



b) Two corresponding layers in new DNN model

$$A_{m\times n} = U_{m\times k}\Sigma_{k\times k}V^{T}_{k\times k} = U_{m\times k}N_{k\times k}$$

Note : The number of parameters changes from mn to (m+n)k. We reduce the model size significantly if k is much smaller than n.

Illustration :

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} = \begin{bmatrix} u_{11} & \cdots & u_{1n} \\ \vdots & \ddots & \vdots \\ u_{m1} & \cdots & u_{mn} \end{bmatrix} \cdot \begin{bmatrix} \epsilon_{11} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & \epsilon_{kk} & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \epsilon_{nn} \end{bmatrix} \cdot \begin{bmatrix} v_{11} & \cdots & v_{1n} \\ \vdots & \ddots & \vdots \\ v_{n1} & \cdots & v_{nn} \end{bmatrix}$$

$$\approx \begin{bmatrix} u_{11} & \cdots & u_{1n} \\ \vdots & \ddots & \vdots \\ u_{m1} & \cdots & u_{mn} \end{bmatrix} \cdot \begin{bmatrix} \epsilon_{11} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & \epsilon_{kk} & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 \end{bmatrix} \cdot \begin{bmatrix} v_{11} & \cdots & v_{1n} \\ \vdots & \ddots & \vdots \\ v_{n1} & \cdots & v_{nn} \end{bmatrix}$$

$$= \begin{bmatrix} u_{11} & \cdots & u_{1k} \\ \vdots & \ddots & \vdots \\ u_{m1} & \cdots & u_{mk} \end{bmatrix} \cdot \begin{bmatrix} \epsilon_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \epsilon_{kk} \end{bmatrix} \cdot \begin{bmatrix} v_{11} & \cdots & v_{1k} \\ \vdots & \ddots & \vdots \\ v_{k1} & \cdots & v_{kk} \end{bmatrix}$$

$$= \begin{bmatrix} u_{11} & \cdots & u_{1k} \\ \vdots & \ddots & \vdots \\ u_{m1} & \cdots & u_{mk} \end{bmatrix} \cdot \begin{bmatrix} n_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & n_{kk} \end{bmatrix}$$

What Next?

MicroNet Challenge Hosted at NeurIPS 2019

Competition Start: June 1st, 2019.

Submission Deadline: Midnight Pacific Time, September 30th, 2019.

Scoring

Two factors will be taken into account when scoring an entry:

1. **Parameter Storage**: A 16-bit parameter counts as one parameter. If quantization is performed, a parameter of less than 16-bits will be counted as a fraction of one parameter.

2. **Math Operations**: The mean number of arithmetic operations per example required to perform inference on the test set. Multiplies and additions count separately.

BaseLine : For ImageNet and CIFAR-100, parameter storage, and compute requirements will be normalized relative to MobileNetV2 with width 1.4 (6.9M parameters, 1170M math operations).

References

- G. Hinton, O. Vinyals and J. Dean. *Distilling the Knowledge in a Neural Network.* In NIPS 2014
- Carl Doersch. *Tutorial on Variational Autoencoders.* stat.ML 2016
- Song Han, Huizi Mao, William and J. Dally. *Deep Compression: Compressing deep neural networks with pruning, trained quantization and huffman coding.* In ICLR 2016.
- J. Xue, J. Li, and Y. Gong. *Restructuring of Deep Neural Network Acoustic Models with Singular Value Decomposition.*
- W. Joo, W. Lee, S. Park & Il-Chul Moon. *Dirichlet Variational Autoencoder.*

Thank You