STRUCTURED DATA REPORT
YOUR CLASSIFIER IS SECRETLY AN ENERGY BASED MODEL AND YOU SHOULD TREAT IT LIKE ONE

Joël Garde
March 2021

1 Overview

The article proposes to re-interpret the classifier loss as part of a *Energy Based Model* (EMB). By doing so, they obtain a state-of-the art hybrid model, capable of of both classification and generation. Additionally, the model presents desirable qualities: good calibration, out-of-distribution detection (OOD), and adversarial robustness. This bridges the gap between discriminative and generative models.

notations. I will adopt some vocabulary from Lecun's tutorial. (`http://yann.lecun.com/exdb/publis/pdf/lecun-06.pdf`). I will use $x[y]$ to denote the y^{th} value of the vector x.

problem considered. The paper considers a *multi-class*, *single-output* classification setting. Inputs are noted $x \in \mathbb{R}^D$, outputs are $y \in \{1, 2, \dots, K\}$. The model is parametrized by $\theta : f_\theta \in \mathbb{R}^D \to \mathbb{R}^K$ is a function associating to each $x \in \mathbb{R}^D$ a set of *logits*. In the EMB framework, logits define a negative energy:

$$f_\theta(x)[y] = -E_\theta(x, y) \tag{1}$$

The *inference procedure* consists in finding the y minimizing the energy $E(x, y)$. In our context, this can be done by complete enumeration over $\{1, \dots, K\}$.

$$\begin{aligned} \text{prediction}(x, \theta) &= \text{argmin}_{y \in \{1,2,\dots,K\}} E_\theta(x, y) \\ &= \text{argmax}_{y \in \{1,2,\dots,K\}} f_\theta(x)[y] \end{aligned} \tag{2}$$

For given class x, the chosen class is the one with the maximum logit.

discussion. This is not a fully general structured output model. The output space $\mathcal{Y}$ is restricted to $\{-1, 1\}^n$ for a n classes problem.

2 Energy Based Model

A remainder on EMB, an overview of Lecun's tutorial, and a developpement interpreted in the specific context of our paper.

2.1 Motivation

An energy based model defines an *energy* for each configuration of its inputs variable. Here, the inputs are $x \in \mathbb{R}^D$ and $y \in \{1, 2, \dots, K\}$. Our model defines this energy as seen in equation 1.

Lecun defines 4 tasks answered by enery based systems:

- *Prediction*, classification, and decision-making.

  "Which value of Y is most compatible with this X?"

- *Ranking*:

  "Is Y1 or Y2 more compatible with this X?"

- *Detection*:

  "Is this value of Y compatible with X?"

- *Conditional density estimation*:

  "What is the conditional probability distribution over Y given X?"

Classifier are interested in the *prediction* task. Generative model in *conditional density estimation*. We will also see how the author introduce the *ranking* and *Detection* tasks through calibration and Out-of-Density detection (OOD).

2.1.1 COMMENT.

The problem the authors tinker with is the gap between hybrids models and specific models. Let us adopt a classifier-centric viewpoint. In term of raw accuracy, a specialized model will perform better. Its others qualities such as calibration, OOD, robustness are developped pre, or post learning, using techniques such as data augmentation.

A hybrid model does all of it at once. Yet, they are not popular today, because their classifiying performance are not state-of-the-art. This work brings them up to part. As such, the generative part of the EBM is used indirectly, to boost the other qualities of the classifier.

2.2 PROBABILISTIC MODELING

Energies can be seen as unnormalized probabilities. In particular, Lecun defines how to go from a collection of energies to probability distribution using the *Gibbs Distribution*.

$$P(y \mid x) = \frac{e^{-\beta E(y,x)}}{\int_{y \in \mathcal{Y}} e^{-\beta E(y,x)}} \tag{3}$$

In the machine learning community, β is the inverse-temperature (by a physics analogy) and is commonly set to 1. What is more, $\mathcal{Y}$ is denumerable. The expression is then simplified to:

$$P(y \mid x) = \frac{e^{-E(y,x)}}{\sum_{y \in \mathcal{Y}} e^{-E(y,x)}} \tag{4}$$

We retrieve the function commonly known as the *softmax*. The denominator is called the *partition function*. A key point to understand the Joint Energy-based Model (JEM) proposed by the authors is to see that the partition function is not always tractable. $\sum_{y \in \mathcal{Y}} e^{-E(y,x)}$ can be computed, however, this is not simple for $P(x|y)$:

$$P(x \mid y) = \frac{e^{-\beta E(y,x)}}{\int_{x \in \mathcal{X}} e^{-\beta E(y,x)}} \tag{5}$$

Since our x are arbitrary vectors in $\mathbb{R}^D$, this integral might not be tractable. The *partition function* will be denoted $Z(\cdot)$. IE, $P_\theta(y \mid x) = e^{-E(y,x)}/Z(\theta, x)$.

comment. When dealing with structured output, the probalistic model shoves all the difficulties of dealing with the complex output space into the partition function. The problem is thus refraimed as how to overcome the unknown partition function.

2.3 Negative Log-Likelihood

Lecun provides a detailed explanation of the Negative-Log-Likelihood (NLL) loss in the context of EBM. It is extremely relevant, since the common *cross-entropy* loss used in machine learning is the composition of the *softmax* and the negative log-likelihood. The main point of this paper in finding the hidden JEM in a classifier can actually be seen as a refinement of Lecun's paragraph on the negative log-likelihood.

We will keep using the notations defined by Grathwohl, $\beta = 1$ and Z for the partition function. In this setting, the Negative Log Likelihood (NLL) is defined as:

$$\begin{aligned} L_\theta^{NLL}(x, y) &= E_\theta(x, y) + Z(\theta, x) \\ &= -f_\theta(x)[y] + log(\sum_{j=1}^{K} e^{-f_\theta(x)[y]}) \end{aligned} \tag{6}$$

One can directly recognize the cross-entropy here. It is for instance directly the formula in the pytorch doc. In this context $Z(\theta, x)$ is called the *free-energy*. Additionally, we note the log of the sum of exponential as the *LogSumExp* function. It will come handy to have it named later. The first term, $f_\theta(x)[y]$, pushes down the energy of the correct class. The second, $LogSumExp(f_\theta(x)[y])$ pushes up all the energy.

The negative log-likelihood is to be interpreted probabilistically. Assuming independent samples, we have the following likelihood $L = \prod_{x,y \in (X,Y)} p_\theta(y|x)$. We want to maximize it, which correspond to minimizing the negative log likelihood,

$$NLL = -\sum log\, p_\theta(y|x) \tag{7}$$

and using equation 3, we retrieve 6.

Surprisingly, Lecun does not develop further the Bayesian analogy. And this is exactly what the authors did. We need to use Bayes law:

$$p_\theta(y|x) \cdot p_\theta(x) = p_\theta(x, y) \tag{8}$$

go through the log, $log\, p_\theta(x|y) + log\, p_\theta(y) = log\, p_\theta(x, y)$. Using equations 7, and 6 we see that

$$E_\theta(x) = -Z(\theta, x) = -logsumexp(f_\theta(x)[y]) \tag{9}$$

And we have fully derived the EBM behind a classifier:

$$E_\theta(x, y) = -f_\theta(x)[y] \tag{10}$$
$$E_\theta(y) = E_\theta(x, y) - E_\theta(x) \tag{11}$$
$$E_\theta(x) = -Z(\theta, x) = -logsumexp(f_\theta(x)[y]) \tag{12}$$

Those three equations allow us to understand the magnificent schema in figure 1 proposed by the authors (where they conflated energy and probability).

3 Developing the Bayesian viewpoint

This a personal development to further the Bayesian analogy. In classical Bayesian setting, we do not optimize the likelihood directly, but impose a prior and optimize the posterior:

$$posterior \sim likelihood \times prior \tag{13}$$

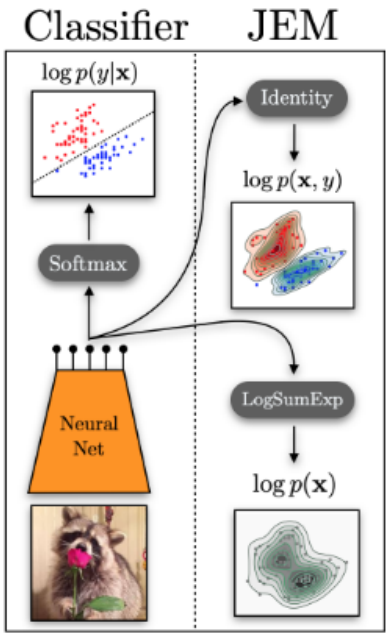

Figure 1: Visualization of our method, JEM, which defines a joint EBM from classifier architectures.

Figure 1: visualisation taken from the paper

$$\begin{aligned} p(\theta|x,y) &\sim p(x,y|\theta)\ p(\theta) \\ &\sim p(y|x,\theta)\ p(x|\theta)p(\theta) \\ logp_\theta(x,y) &\sim logp_\theta(y|x)\ + logp_\theta(x) + logp(\theta) \end{aligned} \tag{14}$$

We retrieve in this way the factorization proposed in section 4 of their paper. hidden behind the $\sim$, is $p(x,y)$ which is conveniently ignored since it does not depend on θ, and thus unnecessary for optimizing θ. And we see that to optimize the posterior log probability of our weights, we need to sum three components:

- $log\, p_\theta(y|x)$, the *likelihood*, that is readily optimized by cross-entropy.
- $log\, p_\theta(x)$, which is discussed below.
- $log\, p(\theta)$, the *prior* on weights. Gaussian prior have been shown to be equivalent to a $L2$ regularization, also known as weight decay and commonly used in deep-learning architectures.

Choosing to factorize as $p(y|x,\theta)\ p(x|\theta)$ instead of $p(x|y,\theta)\ p(y|\theta)$ is a differentiating input from the authors. The major benefit is that $logp_\theta(y|x)$ is computed exactly using cross-entropy. we only have to approximate $p(x|\theta)$.

The only term not already optimized for in the classical classifier architecture is $logp_\theta(x)$. It can be seen as a *generative* term since it tries to optimize the probability of the inputs given the weights. We can understand better why those are called *hybrids* models: they optimize both the generative and the discriminative term together, resulting in a posterior optimization.

The big question is then: How to take in account this parameter?

4 Optimization

with some inputs from https://arxiv.org/pdf/1904.09770.pdf.

The authors optimization scheme achieve the same objective as described above, athough they do not expose it this way.

How to optimize the generative term $logp_\theta(x)$ is not obvious. Indeed, if we have $E_\theta(x) = -logsumexp(f_\theta(x)[y])$ which is tractable, the partition term $Z(\theta, y)$ is not: we cannot integrate over the input space.

> The primary difficulty in training EBMs comes from effectively estimating and sampling the partition function.

[Du, Mordach](`https://arxiv.org/pdf/1903.08689.pdf`)

To overcome this stumbling block ,we need to reach for the probabilistic toolbox, and rewrite the derivative of the log-likelihood:

$$log_{p_\theta}(x) = -E_\theta(x) - \int_x e^{-E_\theta(x)} \tag{15}$$

$$\begin{aligned}
\frac{\partial logp_\theta(x)}{\partial \theta} &= -\frac{\partial}{\partial \theta} E_\theta(x) - \frac{\partial}{\partial \theta} \int_x e^{-E_\theta(x)} && (16)\\
&= -\frac{\partial}{\partial \theta} E_\theta(x) - \frac{1}{Z(\theta, y)} \int_x (e^{-E_\theta(x)} \frac{\partial}{\partial \theta} - E_\theta(x)) && (17)\\
&= -\frac{\partial}{\partial \theta} E_\theta(x) + \int_x p_\theta(x) \frac{\partial}{\partial \theta} E_\theta(x) && (18)\\
&= -\frac{\partial}{\partial \theta} E_\theta(x) + \mathbb{E}_{p_\theta(x')}[\frac{\partial}{\partial \theta} E_\theta(x')] && (19)
\end{aligned}$$

> Intuitively, this gradient decreases energy of the positive data samples x^+, while increasing the energy of the negative samples x^- from the model p_θ.

[Du, Mordach]

We managed to hide the Z partition function inside an expectation. While we can't sample directly from $p_\theta(x)$ (since we put the intractable Z here), this expectation can be approximated using Markov Chains Monte Carlo (MCMC) methods. Here, the authors have chosen to use Stochastic Gradient Langevin Dynamics (SGLD), which sample according to the following kernel:

$$x_0 \sim p_0(x), \qquad x_{i+1} = x_i - \frac{\alpha}{2} \frac{\partial E_\theta(x_i)}{\partial x_i} + \epsilon, \qquad \epsilon \sim \mathcal{N}(\theta, \alpha) \tag{20}$$

All in all, the training procedure in figure 2 is as follows. Step 3 is the classic forward pass. Step 4 is the initialization of the Markov Chain using the replay buffer B. Instead of generation new samples at each step, a single chain is maintained by storing past samples in the replay buffer B (at step 11). This is called Persistent Contrastive Divergence (PCD) [tielman]. Finally steps 8,9 combine the two losses.

We have the following expressions for the loss (with $\hat{x} \sim p_\theta(x)$) :

Algorithm 1 JEM training: Given network f_θ, SGLD step-size α, SGLD noise σ, replay buffer B, SGLD steps η, reinitialization frequency ρ

while not converged **do**
  Sample $\mathbf{x}$ and y from dataset
  $L_{\text{clf}}(\theta) = \text{xent}(f_\theta(\mathbf{x}), y)$
  Sample $\widehat{\mathbf{x}}_0 \sim B$ with probability $1-\rho$, else $\widehat{\mathbf{x}}_0 \sim \mathcal{U}(-1, 1)$ ▷ Initialize SGLD
  for $t \in [1, 2, \ldots, \eta]$ **do** ▷ SGLD
    $\widehat{\mathbf{x}}_t = \widehat{\mathbf{x}}_{t-1} + \alpha \cdot \frac{\partial \text{LogSumExp}_{y'}(f_\theta(\widehat{\mathbf{x}}_{t-1})[y'])}{\partial \widehat{\mathbf{x}}_{t-1}} + \sigma \cdot \mathcal{N}(0, I)$
  end for
  $L_{\text{gen}}(\theta) = \text{LogSumExp}_{y'}(f(\mathbf{x})[y']) - \text{LogSumExp}_{y'}(f(\widehat{\mathbf{x}}_t)[y'])$ ▷ Surrogate for Eq 2
  $L(\theta) = L_{\text{clf}}(\theta) + L_{\text{gen}}(\theta)$
  Obtain gradients $\frac{\partial L(\theta)}{\partial \theta}$ for training
  Add $\widehat{\mathbf{x}}_t$ to B
end while

Variable	Values
initial learning rate	.0001
learning epochs	150
learning rate decay	.3
learning rate decay epochs	50, 100
SGLD steps η	20
Buffer-size	10000
reinitialization frequency ρ	.05
SGLD step-size α	1
SGLD noise σ	.01

Figure 2: algorithm taken from the paper

$$L(\theta) = L_{clf}(\theta) \quad + \quad L_{disc}(\theta) \tag{21}$$

$$\begin{aligned} L(\theta) = & -f_\theta(x)[y] + LogSumExp_y(f_\theta(x)[y]) \\ & + \quad LogSumExp_y(f_\theta(x)[y]) - LogSumExp_y(f_\theta(\hat{x})[y])(\theta) \end{aligned} \tag{22}$$

$$L(\theta) = E_\theta(x, y) - E_\theta(x) \quad + \quad -E_\theta(x) + E_\theta(\hat{x}) \tag{23}$$

$$L(\theta) = E_\theta(x, y) - 2E_\theta(x) + E_\theta(\hat{x}) \tag{24}$$

making Z disappear. To make Z go away, one can form a ratio. If you have two samples from the same distribution, with $p(a) = \frac{e^{-E(a)}}{Z}$, $p(b) = \frac{e^{-E(b)}}{Z}$, the ratio $\frac{p(a)}{p(b)}$ is free of the partition function Z. In Markov Chains, this ratio appears in the *acceptance ratio*. Another alternative is to form this ratio explicitely, in a *constrastive loss*. This is typically the loss used in Generative Adversarial Networks (GAN).

comment. The implementation and usage of SGLD can be discussed. They add a lot of hyperparameters (step size, mixing time ...). The implementation is also biased because the sampler is run for a limited amount of time. In addition, Markovs Chains methods are also unstable and supervising the learning process gets complicated. This is acknowledge by the authors, and it is even the motivation for their continuation paper, No MCMC for me: Amortized sampling for fast and stable training of energy-based models (ICLR 2021).

5 Applications

hybrid modeling. The authors report competitive performances as both a classifier and a generative model on standard image datasets (CIFAR, SVHN). This is the first goal, bridging the gap between hybrids and specific models.

calibration. Calibration is a desirable property of classifiers. A calibrated classifier predictive confidence, $p_\theta(y|x)$ is equal to the expected classification rate. A good calibration

can help a Humain in the Loop (HITL) decide when to intervene. However, there is no explaination as to why their hybrid model is better calibrated.

OOD detection. Since the model is explicitly trained to model the input density $p_\theta(x)$, but also as a classifier, $p_\theta(y|x)$ authors found good results using thoses mesures to detect OOD. They also derive a new mesure, on probability *masses*. Instead of looking at the point density, it is sensitive to the density of the area around it. This new mesure needs the generative side of the model.

Robustness. Generative models are also more robust to adversarial attack. The hindsight is that adversarial exemples, although close to real exemples in a given metric, may not be in term of the model defined density. By running the Monte Carlo scheme on the adversarial exemple, it is brought back to an area of high density where the model can classify it correctly.

6 CONCLUSION.

The derived EBM architecture is simple as it is directly the common classifier architecture. However, the complexity was transfered to the training procedure and the Monte Carlo sampling scheme. The experiments were restricted to image classification datasets, where it is easier to evaluate the quality of the generation. It remains interesting to see the hybrid model as classifier with an added plug-in generative loss, and results are promizing. Yet, it lacks some explaination as for the reason behind its performances compared to other EMB.

7 TESTING.

I use the provided code at `https://github.com/wgrathwohl/JEM`. The code is rather straightforward and easy to grasp. we have for instance: `https://github.com/wgrathwohl/JEM/blob/master/train_wrn_ebm.py#L228` where the SGLD is performed, and lines 312 to 341 where the loss is assembled. Looking at the github issues proved instructive, most notably this comment by wgrathwohl (one of the authors) on a diverging loss:

> I will be the first to admit that EBM training in this way is a nightmare and requires pretty consistent baby-sitting. At the moment these models are basically where GANs were in like 2014. Not easy to train. Requires a lot of hand-tuning. The main point of this paper was to demonstrate the utility of these models if they can be trained. There have been a number of improvements which can stabilize EBM training.

The classifier architecture used is a WideResNet (depth 28, width 10). The trainning for the CIFAR10 is quite expensive. Reported times are in the order of 80 hours using a GPU. Using a public Google colab instance, I can train a WideResNet(4,2) at approximately 3 iteration per second, or 3 minutes per epochs. I stopped the training after 5 epochs at an accurracy of approximately 30%.

In an effort to obtain an acceptably performant model in a resonable time, I lower the number of SGLD iterations to 10, and set the learning rate to 0.01 from 0.0001 and doubled the batch size. However, I ran into the instability error and the loss diverged. As expected, the training of such models is quite tedious. Restarting the process, I was able to train a model up to 40% accuracy in 5 epochs. figures 3 and 4 present generated samples at epochs 0 and 5. We can see some structure is emerging, but it is not yet human-identifiable. Given the difficulty to self-train a model, I used the pre-trained model for further visualisations. 5 shows generated class conditional samples. Generating class conditional samples works by running the SGLD scheme from noise.

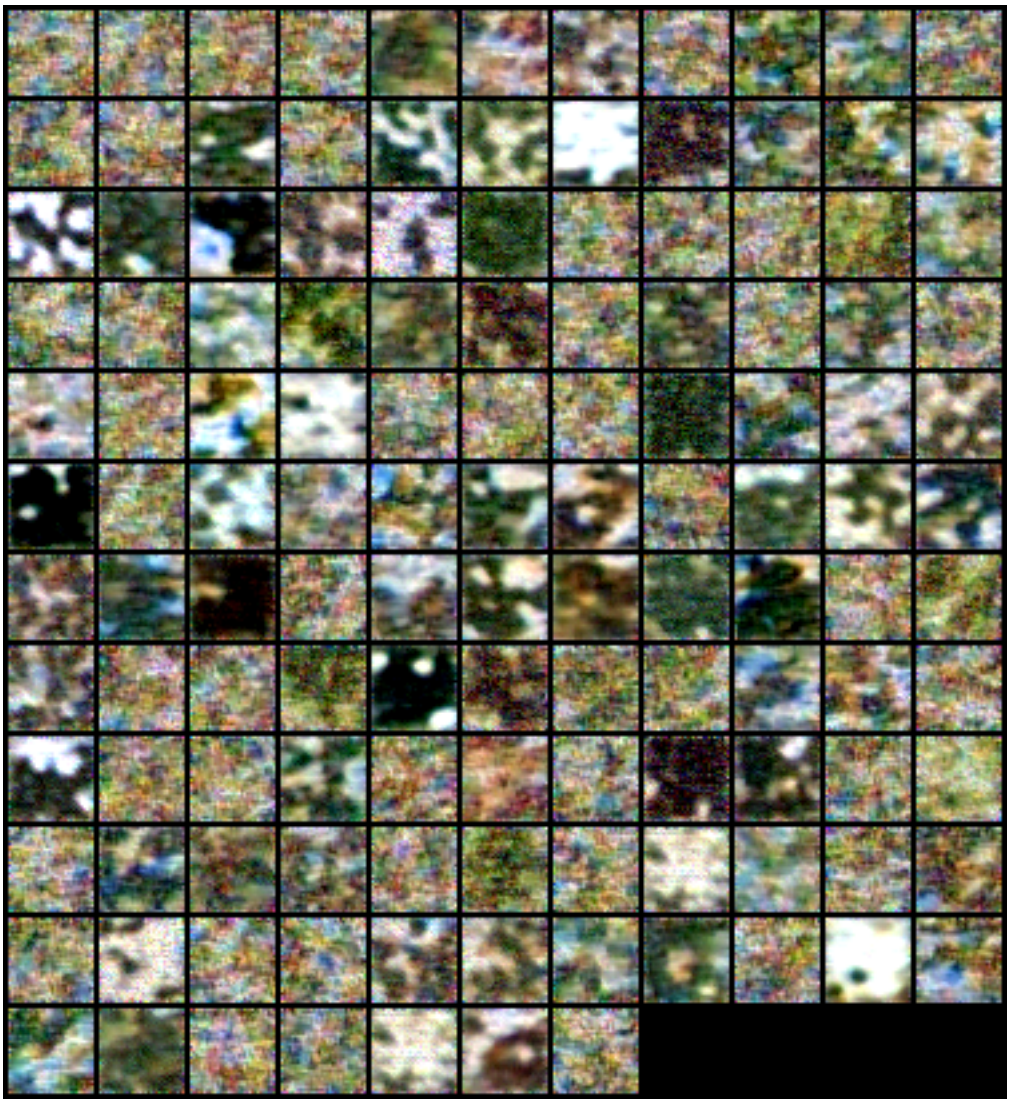

Figure 3: generated batch at epoch 0

Figure 4: generated batch at epoch 1

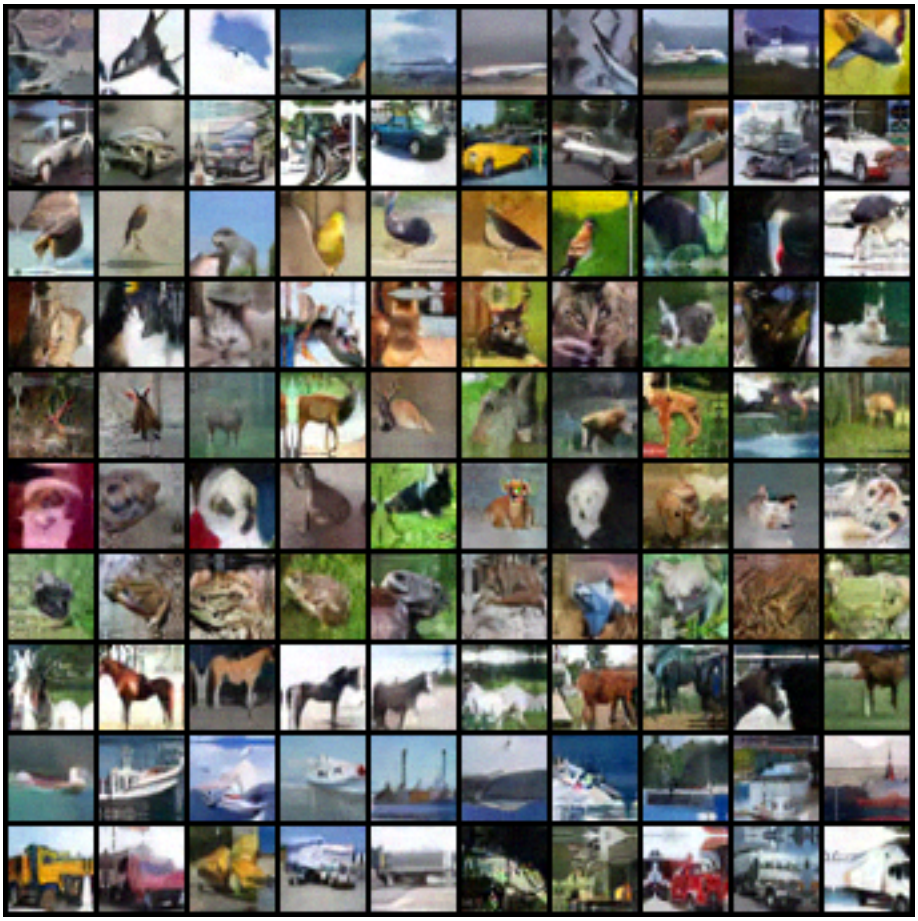

Figure 5: class conditional samples (1 class per line)