# Your Classifier is Secretly an Energy Based Model and You Should Treat it Like One

January 29, 2020

# What is an Energy Based Model (EBM)?

# What is an Energy Based Model (EBM)?

Any pdf $p_\theta(x)$ can be written down as:

$$p_\theta(\mathbf{x}) = \frac{\exp\left(-E_\theta(\mathbf{x})\right)}{Z_\theta} \tag{1}$$

# What is an Energy Based Model (EBM)?

Any pdf $p_\theta(x)$ can be written down as:

$$p_\theta(\mathbf{x}) = \frac{\exp\left(-E_\theta(\mathbf{x})\right)}{Z_\theta} \tag{1}$$

where $Z_\theta$ is a normalizing constant:

$$Z_\theta = \int_{\boldsymbol{x}} \exp\left(-E_\theta(\boldsymbol{x})\right) d\boldsymbol{x}$$

# What is an Energy Based Model (EBM)?

Any pdf $p_\theta(x)$ can be written down as:

$$p_\theta(\mathbf{x}) = \frac{\exp\left(-E_\theta(\mathbf{x})\right)}{Z_\theta} \tag{1}$$

where $Z_\theta$ is a normalizing constant:

$$Z_\theta = \int_{\boldsymbol{x}} \exp\left(-E_\theta(\boldsymbol{x})\right) d\boldsymbol{x}$$

- This parametrization allows us to forget about $p_\theta(x)$ and work with $E_\theta(x)$

# What is an Energy Based Model (EBM)?

Any pdf $p_\theta(x)$ can be written down as:

$$p_\theta(\mathbf{x}) = \frac{\exp\left(-E_\theta(\mathbf{x})\right)}{Z_\theta} \tag{1}$$

where $Z_\theta$ is a normalizing constant:

$$Z_\theta = \int_{\boldsymbol{x}} \exp\left(-E_\theta(\boldsymbol{x})\right) d\boldsymbol{x}$$

- This parametrization allows us to forget about $p_\theta(x)$ and work with $E_\theta(x)$
- Function $E_\theta(x)$ is an *energy function* (hence the name — EBM)

# What is an Energy Based Model (EBM)?

Any pdf $p_\theta(x)$ can be written down as:

$$p_\theta(\mathbf{x}) = \frac{\exp\left(-E_\theta(\mathbf{x})\right)}{Z_\theta} \tag{1}$$

where $Z_\theta$ is a normalizing constant:

$$Z_\theta = \int_{\boldsymbol{x}} \exp\left(-E_\theta(\boldsymbol{x})\right) d\boldsymbol{x}$$

- This parametrization allows us to forget about $p_\theta(x)$ and work with $E_\theta(x)$
- Function $E_\theta(x)$ is an *energy function* (hence the name — EBM)
- Energy functions are *easier* to work with: you do not have any restrictions on $E_\theta(x)$ ($p_\theta(x)$ is nonnegative and integrates to 1).

# What is an Energy Based Model (EBM)?

Any pdf $p_\theta(x)$ can be written down as:

$$p_\theta(\mathbf{x}) = \frac{\exp\left(-E_\theta(\mathbf{x})\right)}{Z_\theta} \tag{1}$$

where $Z_\theta$ is a normalizing constant:

$$Z_\theta = \int_{\boldsymbol{x}} \exp\left(-E_\theta(\boldsymbol{x})\right) d\boldsymbol{x}$$

- This parametrization allows us to forget about $p_\theta(x)$ and work with $E_\theta(x)$
- Function $E_\theta(x)$ is an *energy function* (hence the name — EBM)
- Energy functions are *easier* to work with: you do not have any restrictions on $E_\theta(x)$ ($p_\theta(x)$ is nonnegative and integrates to 1).
- Energy functions are *harder* to work with: $Z_\theta$ is hard or impossible to compute, so we can't optimize EBM for $\theta$

# How to train EBMs

# How to train EBMs

- We perform standard MLE: maximize $\log p_\theta(x_i)$ at our points $x_i$.

# How to train EBMs

- We perform standard MLE: maximize $\log p_\theta(x_i)$ at our points $x_i$.
- By optimizing $\log p_\theta(x)$ we also optimize $E_\theta(x)$.

# How to train EBMs

- We perform standard MLE: maximize $\log p_\theta(x_i)$ at our points $x_i$.
- By optimizing $\log p_\theta(x)$ we also optimize $E_\theta(x)$.
- We can't compute $\log p_\theta(x)$, but we can (approximately) compute its log-derivative via:

$$\frac{\partial \log p_\theta(\mathbf{x})}{\partial \theta} = \mathbb{E}_{p_\theta(\mathbf{x}')}\left[\frac{\partial E_\theta(\mathbf{x}')}{\partial \theta}\right] - \frac{\partial E_\theta(\mathbf{x})}{\partial \theta} \tag{2}$$

# How to train EBMs

- We perform standard MLE: maximize $\log p_\theta(x_i)$ at our points $x_i$.
- By optimizing $\log p_\theta(x)$ we also optimize $E_\theta(x)$.
- We can't compute $\log p_\theta(x)$, but we can (approximately) compute its log-derivative via:

$$\frac{\partial \log p_\theta(\mathbf{x})}{\partial \theta} = \mathbb{E}_{p_\theta(\mathbf{x}')}\left[\frac{\partial E_\theta(\mathbf{x}')}{\partial \theta}\right] - \frac{\partial E_\theta(\mathbf{x})}{\partial \theta} \tag{2}$$

- Proof: follows directly from the definition.

# How to train EBMs

- We perform standard MLE: maximize $\log p_\theta(x_i)$ at our points $x_i$.
- By optimizing $\log p_\theta(x)$ we also optimize $E_\theta(x)$.
- We can't compute $\log p_\theta(x)$, but we can (approximately) compute its log-derivative via:

$$\frac{\partial \log p_\theta(\mathbf{x})}{\partial \theta} = \mathbb{E}_{p_\theta(\mathbf{x}')}\left[\frac{\partial E_\theta(\mathbf{x}')}{\partial \theta}\right] - \frac{\partial E_\theta(\mathbf{x})}{\partial \theta} \tag{2}$$

- Proof: follows directly from the definition.
- But a new problem arises: how to compute expectation?

# How to train EBMs

- We perform standard MLE: maximize $\log p_\theta(x_i)$ at our points $x_i$.
- By optimizing $\log p_\theta(x)$ we also optimize $E_\theta(x)$.
- We can't compute $\log p_\theta(x)$, but we can (approximately) compute its log-derivative via:

$$\frac{\partial \log p_\theta(\mathbf{x})}{\partial \theta} = \mathbb{E}_{p_\theta(\mathbf{x}')}\left[\frac{\partial E_\theta(\mathbf{x}')}{\partial \theta}\right] - \frac{\partial E_\theta(\mathbf{x})}{\partial \theta} \tag{2}$$

- Proof: follows directly from the definition.
- But a new problem arises: how to compute expectation?
- Answer: we can't do that directly, so let's use SGLD sampling.

# Stochastic Gradient Langevin Dynamics (SGLD)

Motivation:

# Stochastic Gradient Langevin Dynamics (SGLD)

Motivation:

- Imagine we have a "magic" function $p_\theta(x)$ that takes an input $x$ and tells us its density.

# Stochastic Gradient Langevin Dynamics (SGLD)

Motivation:

- Imagine we have a "magic" function $p_\theta(x)$ that takes an input $x$ and tells us its density.
- Unfortunately, we cannot sample new $x$-s from it directly, i.e. $p_\theta(x)$ does not support such operations.

# Stochastic Gradient Langevin Dynamics (SGLD)

Motivation:

- Imagine we have a "magic" function $p_\theta(x)$ that takes an input $x$ and tells us its density.
- Unfortunately, we cannot sample new $x$-s from it directly, i.e. $p_\theta(x)$ does not support such operations.
- But we really want to sample from it! What should we do?

# Stochastic Gradient Langevin Dynamics (SGLD)

Motivation:

- Imagine we have a "magic" function $p_\theta(x)$ that takes an input $x$ and tells us its density.
- Unfortunately, we cannot sample new $x$-s from it directly, i.e. $p_\theta(x)$ does not support such operations.
- But we really want to sample from it! What should we do?
- SGLD is a method to sample from such functions.

# Stochastic Gradient Langevin Dynamics (SGLD)

SGLD in short:

# Stochastic Gradient Langevin Dynamics (SGLD)

SGLD in short:

1. Start from any random point $x_0$.

# Stochastic Gradient Langevin Dynamics (SGLD)

SGLD in short:

1. Start from any random point $x_0$.
2. Compute the gradient of $\nabla \log p_\theta(x_0)$.

# Stochastic Gradient Langevin Dynamics (SGLD)

SGLD in short:

1. Start from any random point $x_0$.
2. Compute the gradient of $\nabla \log p_\theta(x_0)$.
3. Perform a small step in that direction.

# Stochastic Gradient Langevin Dynamics (SGLD)

SGLD in short:

1. Start from any random point $x_0$.
2. Compute the gradient of $\nabla \log p_\theta(x_0)$.
3. Perform a small step in that direction.
4. Add a little bit of noise.

# Stochastic Gradient Langevin Dynamics (SGLD)

SGLD in short:

1. Start from any random point $x_0$.
2. Compute the gradient of $\nabla \log p_\theta(x_0)$.
3. Perform a small step in that direction.
4. Add a little bit of noise.
5. Obtain $x_{t+1}$.

# Stochastic Gradient Langevin Dynamics (SGLD)

SGLD in short:

1. Start from any random point $x_0$.
2. Compute the gradient of $\nabla \log p_\theta(x_0)$.
3. Perform a small step in that direction.
4. Add a little bit of noise.
5. Obtain $x_{t+1}$.

In other words, run the following recurrence a lot of times:

$$\mathbf{x}_0 \sim p_0(\mathbf{x}), \quad \mathbf{x}_{i+1} = \mathbf{x}_i - \frac{\alpha}{2} \frac{\partial E_\theta(\mathbf{x}_i)}{\partial \mathbf{x}_i} + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \alpha) \tag{3}$$

# Stochastic Gradient Langevin Dynamics (SGLD)

SGLD in short:

1. Start from any random point $x_0$.
2. Compute the gradient of $\nabla \log p_\theta(x_0)$.
3. Perform a small step in that direction.
4. Add a little bit of noise.
5. Obtain $x_{t+1}$.

In other words, run the following recurrence a lot of times:

$$\mathbf{x}_0 \sim p_0(\mathbf{x}), \quad \mathbf{x}_{i+1} = \mathbf{x}_i - \frac{\alpha}{2} \frac{\partial E_\theta(\mathbf{x}_i)}{\partial \mathbf{x}_i} + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \alpha) \tag{3}$$

Justification: if your model and your choice of $\alpha$ are "good" (in some sense), then after infinite amount of steps your samples will be "good" (in some sense).

# Ok, how to treat a classifier as an EBM?

# Ok, how to treat a classifier as an EBM?

- Remember that *any* function can be used as $E_\theta(x)$

# Ok, how to treat a classifier as an EBM?

- Remember that *any* function can be used as $E_\theta(x)$
- Let $f_\theta(x)$ be our neural network.

# Ok, how to treat a classifier as an EBM?

- Remember that *any* function can be used as $E_\theta(x)$
- Let $f_\theta(x)$ be our neural network.
- Let $f_\theta(x)[y]$ be the $y$-th logit (i.e. logit value for class $y$).

# Ok, how to treat a classifier as an EBM?

- Remember that *any* function can be used as $E_\theta(x)$
- Let $f_\theta(x)$ be our neural network.
- Let $f_\theta(x)[y]$ be the $y$-th logit (i.e. logit value for class $y$).
- Let our examples be $s = (x, y)$

# Ok, how to treat a classifier as an EBM?

- Remember that *any* function can be used as $E_\theta(x)$
- Let $f_\theta(x)$ be our neural network.
- Let $f_\theta(x)[y]$ be the $y$-th logit (i.e. logit value for class $y$).
- Let our examples be $s = (x, y)$
- We define energy $E_\theta(s) = E_\theta(x, y) = -f_\theta(x)[y]$.

# Ok, how to treat a classifier as an EBM?

- Remember that *any* function can be used as $E_\theta(x)$
- Let $f_\theta(x)$ be our neural network.
- Let $f_\theta(x)[y]$ be the $y$-th logit (i.e. logit value for class $y$).
- Let our examples be $s = (x, y)$
- We define energy $E_\theta(s) = E_\theta(x, y) = -f_\theta(x)[y]$.
- Using $E_\theta(x, y)$ we can define $p_\theta(x)$:

$$p_\theta(\mathbf{x}) = \sum_y p_\theta(\mathbf{x}, y) = \frac{1}{Z_\theta} \sum_y \exp\left(f_\theta(\mathbf{x})[y]\right) \tag{4}$$

# Ok, how to treat a classifier as an EBM?

- Remember that *any* function can be used as $E_\theta(x)$
- Let $f_\theta(x)$ be our neural network.
- Let $f_\theta(x)[y]$ be the $y$-th logit (i.e. logit value for class $y$).
- Let our examples be $s = (x, y)$
- We define energy $E_\theta(s) = E_\theta(x, y) = -f_\theta(x)[y]$.
- Using $E_\theta(x, y)$ we can define $p_\theta(x)$:

$$p_\theta(\mathbf{x}) = \sum_y p_\theta(\mathbf{x}, y) = \frac{1}{Z_\theta} \sum_y \exp\left(f_\theta(\mathbf{x})[y]\right) \tag{4}$$

- Using $p_\theta(x)$ we can define $E_\theta(x)$:

$$E_\theta(\mathbf{x}) = -\log \sum_y \exp\left(f_\theta(\mathbf{x})[y]\right) \tag{5}$$

# Ok, how to treat a classifier as an EBM?

- Remember that *any* function can be used as $E_\theta(x)$
- Let $f_\theta(x)$ be our neural network.
- Let $f_\theta(x)[y]$ be the $y$-th logit (i.e. logit value for class $y$).
- Let our examples be $s = (x, y)$
- We define energy $E_\theta(s) = E_\theta(x, y) = -f_\theta(x)[y]$.
- Using $E_\theta(x, y)$ we can define $p_\theta(x)$:

$$p_\theta(\mathbf{x}) = \sum_y p_\theta(\mathbf{x}, y) = \frac{1}{Z_\theta} \sum_y \exp\left(f_\theta(\mathbf{x})[y]\right) \tag{4}$$

- Using $p_\theta(x)$ we can define $E_\theta(x)$:

$$E_\theta(\mathbf{x}) = -\log \sum_y \exp\left(f_\theta(\mathbf{x})[y]\right) \tag{5}$$

- Higher the logits — lower the energy of $x$.

# Ok, how to treat a classifier as an EBM?

- Remember that *any* function can be used as $E_\theta(x)$
- Let $f_\theta(x)$ be our neural network.
- Let $f_\theta(x)[y]$ be the $y$-th logit (i.e. logit value for class $y$).
- Let our examples be $s = (x, y)$
- We define energy $E_\theta(s) = E_\theta(x, y) = -f_\theta(x)[y]$.
- Using $E_\theta(x, y)$ we can define $p_\theta(x)$:

$$p_\theta(\mathbf{x}) = \sum_y p_\theta(\mathbf{x}, y) = \frac{1}{Z_\theta} \sum_y \exp\left(f_\theta(\mathbf{x})[y]\right) \tag{4}$$

- Using $p_\theta(x)$ we can define $E_\theta(x)$:

$$E_\theta(\mathbf{x}) = -\log \sum_y \exp\left(f_\theta(\mathbf{x})[y]\right) \tag{5}$$

- Higher the logits — lower the energy of $x$.
- Our final loss looks like:

$$\mathcal{L}(x) = \log p(x, y) = \underbrace{\log p(y|x)}_{\text{classification loss}} + \underbrace{\log p(x)}_{\text{generative loss}} \tag{6}$$

# Results and discussion

# Results and discussion

- Disclaimer: results are really good.

# Results and discussion

- Disclaimer: results are really good.
- Authors built a model on SVHN/CIFAR10/CIFAR100 which:

# Results and discussion

- Disclaimer: results are really good.
- Authors built a model on SVHN/CIFAR10/CIFAR100 which:
  - has good classification accuracy

# Results and discussion

- Disclaimer: results are really good.
- Authors built a model on SVHN/CIFAR10/CIFAR100 which:
  - has good classification accuracy
  - has good IS/FID scores (i.e. has good samples)

# Results and discussion

- Disclaimer: results are really good.
- Authors built a model on SVHN/CIFAR10/CIFAR100 which:
  - has good classification accuracy
  - has good IS/FID scores (i.e. has good samples)
  - has much better calibrated predictions (i.e. is not overconfident unlike traditional classifiers)

# Results and discussion

- Disclaimer: results are really good.
- Authors built a model on SVHN/CIFAR10/CIFAR100 which:
  - has good classification accuracy
  - has good IS/FID scores (i.e. has good samples)
  - has much better calibrated predictions (i.e. is not overconfident unlike traditional classifiers)
  - has good out-of-distribution predictions (i.e. detecting when we feed OOD-samples into it)

# Results and discussion

- Disclaimer: results are really good.
- Authors built a model on SVHN/CIFAR10/CIFAR100 which:
  - has good classification accuracy
  - has good IS/FID scores (i.e. has good samples)
  - has much better calibrated predictions (i.e. is not overconfident unlike traditional classifiers)
  - has good out-of-distribution predictions (i.e. detecting when we feed OOD-samples into it)
  - is robust to adversarial attacks

# Results and discussion

- Disclaimer: results are really good.
- Authors built a model on SVHN/CIFAR10/CIFAR100 which:
  - has good classification accuracy
  - has good IS/FID scores (i.e. has good samples)
  - has much better calibrated predictions (i.e. is not overconfident unlike traditional classifiers)
  - has good out-of-distribution predictions (i.e. detecting when we feed OOD-samples into it)
  - is robust to adversarial attacks
  - ...and all of these at the same time!

# Results and discussion

- Disclaimer: results are really good.
- Authors built a model on SVHN/CIFAR10/CIFAR100 which:
  - has good classification accuracy
  - has good IS/FID scores (i.e. has good samples)
  - has much better calibrated predictions (i.e. is not overconfident unlike traditional classifiers)
  - has good out-of-distribution predictions (i.e. detecting when we feed OOD-samples into it)
  - is robust to adversarial attacks
  - ...and all of these at the same time!
- A huge limitation is that because of SGLD the training is unstable and the model is hard to scale.

# Results and discussion

- Disclaimer: results are really good.
- Authors built a model on SVHN/CIFAR10/CIFAR100 which:
  - has good classification accuracy
  - has good IS/FID scores (i.e. has good samples)
  - has much better calibrated predictions (i.e. is not overconfident unlike traditional classifiers)
  - has good out-of-distribution predictions (i.e. detecting when we feed OOD-samples into it)
  - is robust to adversarial attacks
  - ...and all of these at the same time!
- A huge limitation is that because of SGLD the training is unstable and the model is hard to scale.
- A big limitation: we cannot compute the loss value!!!

# Results and discussion

- Disclaimer: results are really good.
- Authors built a model on SVHN/CIFAR10/CIFAR100 which:
  - has good classification accuracy
  - has good IS/FID scores (i.e. has good samples)
  - has much better calibrated predictions (i.e. is not overconfident unlike traditional classifiers)
  - has good out-of-distribution predictions (i.e. detecting when we feed OOD-samples into it)
  - is robust to adversarial attacks
  - ...and all of these at the same time!
- A huge limitation is that because of SGLD the training is unstable and the model is hard to scale.
- A big limitation: we cannot compute the loss value!!!
- An approach was coined Joint Energy based Model (JEM) since we model $(x, y)$ simultaneously.