Discrete Flows: Invertible Generative Models of Discrete Data

CARL EDWARDS (CNE2)

Outline

- Introduction and Motivation
- Background
- Building Blocks
- Models
  - Discrete Autoregressive Flows
  - Discrete Bipartite Flows
- Training
- Results
- Takeaway

Introduction and Motivation

- Normalizing flows have shown strong results in modeling continuous domains, but they haven't been explored in the discrete setting.
- Discrete flows don't require determinant-Jacobian computations
- This can be used for tasks such as language modeling, addition, and Potts models
- **Key Idea:** Can flows be used on discrete distributions?
- Paper introduces two approaches:
  - Discrete autoregressive flows
  - Discrete bipartite flows

Background

- There have not been advances like normalizing flows for discrete distributions
  - Work focuses either on latent-variable models (e.g. “Generating sentences from a continuous space”)
  - Or models assume a fixed ordering of the data (e.g. the Transformer, RNNs)
- There is older work using bidirectional models such as Markov random fields, but they require either approximate inference or approximate sampling
  - Bidirectional models such as BERT have shown increased performance over single directions.
- There has been some work on non-autoregressive discrete models, but they do not maintain an exact density like this work does.

Background – Normalizing Flows

- There are many normalizing flow methods for continuous distributions.
  - They require a transformation that is invertible and has a computationally efficient Jacobian determinant calculation
  - They can generally be divided into two categories:
    - Autoregressive flows
    - Bipartite flows

Background – Autoregressive Flows

- Models that are both autoregressive and flows.
- Examples are Inverse Autoregressive Flows and Masked Autoregressive flows

Given a base distribution $\mathbf{x} \sim p(\mathbf{x})$ in D dimensions

Transform x into y:

$$\boldsymbol{\mu}_d, \boldsymbol{\sigma}_d = f(\mathbf{y}_1, \ldots, \mathbf{y}_{d-1})$$

$$\mathbf{y}_d = \boldsymbol{\mu}_d + \boldsymbol{\sigma}_d \cdot \mathbf{x}_d \qquad \text{for } d \text{ in } 1, \ldots, D.$$

To compute the inverse: (note this can be parallelized)

$$\mathbf{x}_d = \boldsymbol{\sigma}_d^{-1}(\mathbf{y}_d - \boldsymbol{\mu}_d) \qquad \text{for } d \text{ in } 1, \ldots, D.$$

Background – Bipartite Flows

- Uses a “bipartite” factorization where some variables are constant and the others are transformed
- Both forward and inverse computations are fast, but they are less flexible than autoregressive flows
- An example is RealNVP

$$\mathbf{y}_{1:d} = \mathbf{x}_{1:d}$$
$$\mathbf{y}_{d+1:D} = \boldsymbol{\mu} + \boldsymbol{\sigma} \cdot \mathbf{x}_{(d+1):D},$$

Building Blocks

Discrete Change of Variables

Suppose that *x* is a discrete random variable and $\mathbf{y} = f(\mathbf{x})$

Then the change of variables is

$$p(\mathbf{y} = y) = \sum_{x \in f^{-1}(y)} p(\mathbf{x} = x),$$

- Note: f^{-1} is the preimage of *f*

If *f* is invertible, this simplifies:

$$p(\mathbf{y} = y) = p(\mathbf{x} = f^{-1}(y))$$

Compare this to the continuous version:

$$p(\mathbf{y}) = p(f^{-1}(\mathbf{y})) \det \left| \frac{\mathrm{d}\,\mathbf{x}}{\mathrm{d}\,\mathbf{y}} \right|$$

This makes intuitive sense: discrete distributions have no volume, so there is no need to correct for the change in volume (which is what the determinant does)

Discrete Flow Transformations - XOR

- We will first consider the binary case: XOR
- Given a binary vector *x*

$$\mathbf{y}_d = \boldsymbol{\mu}_d \oplus \mathbf{x}_d, \qquad \text{for } d \text{ in } 1, \ldots, D,$$

- This has inverse:

$$\mathbf{x}_d = \boldsymbol{\mu}_d \oplus \mathbf{y}_d.$$

Discrete Flow Transformations – XOR Example

- Given D = 2 with p(x) defined:

	$\mathbf{x}_2 = 0$	$\mathbf{x}_2 = 1$
$\mathbf{x}_1 = 0$	0.63	0.07
$\mathbf{x}_1 = 1$	0.03	0.27

- We cannot model the distribution as $p(x_1)p(x_2)$ (which would be an independence assumption)
- Instead set the following flow: $f(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1, \mathbf{x}_1 \oplus \mathbf{x}_2)$ $p(x_1) = [0.7, 0.3]$, $p(x_2) = [0.9, 0.1]$
- Why is $p(x_2)$ that? Call $p(x_2) = p(y_2)$. Then $p(y_2=0) = p(x_1= x_2) = 0.63 + 0.27 = 0.9$
- Essentially the flow “relabels” the data so that it is better modelled by the base

Discrete Flow Transformations – Extension to Categorical Data

- To extend the XOR to categorical data, the authors introduce the "Modulo location-scale transform", where a modulo integer space is used

- Given a *D*-dimensional vector *x* where each element has *K* values,

$$\mathbf{y}_d = (\boldsymbol{\mu}_d + \boldsymbol{\sigma}_d \cdot \mathbf{x}_d) \bmod K$$

- μ_d and σ_d are autoregressive functions of *y*. Note that σ cannot be zero (just 1,…,*K*-1) just like in continuous case

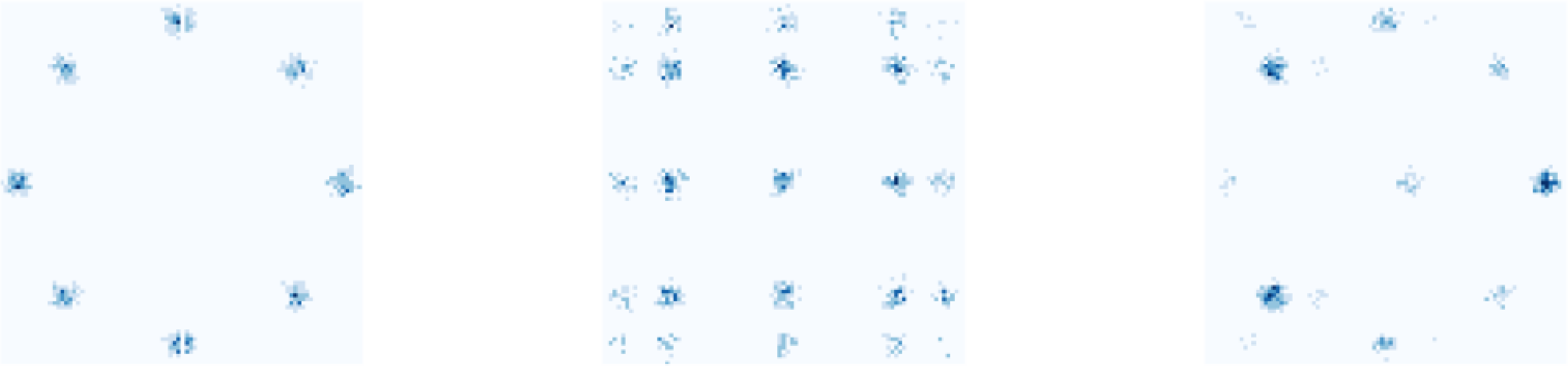
- To be invertible, σ and *K* must be coprime (only sharing divisor 1). There are three easy fixes:
  - Set *K* to be prime
  - Mask noninvertible (non-coprime to *K*) values of σ
  - Set σ = 1

- Note that when *K* = 2 and σ = 1 is XOR

Discrete Flow Transformations –
Modulo location-scale transform Example

- This example shows (a) the data modeled (which is discretized into bins), (b) an attempt to factorize the assumed base distribution of the data, and (c) a single discrete flow

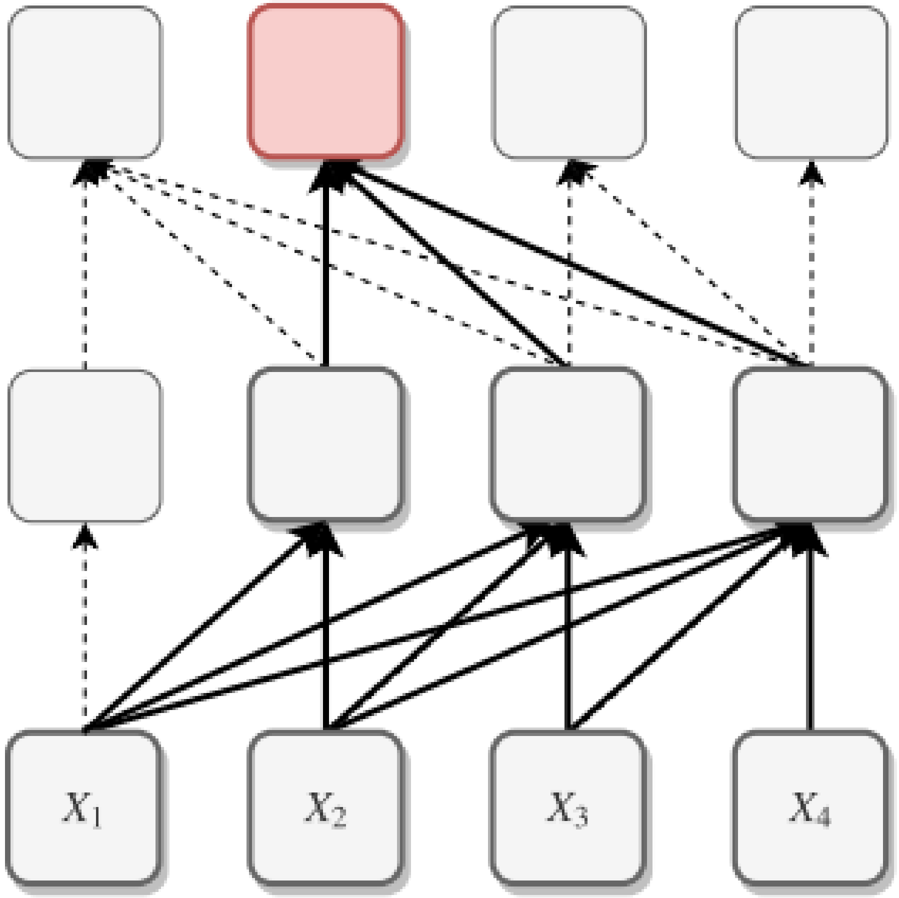
- Even just a single layer of flow is much better at modeling the data.



(a) Data　　(b) Factorized Base　　(c) 1 Flow

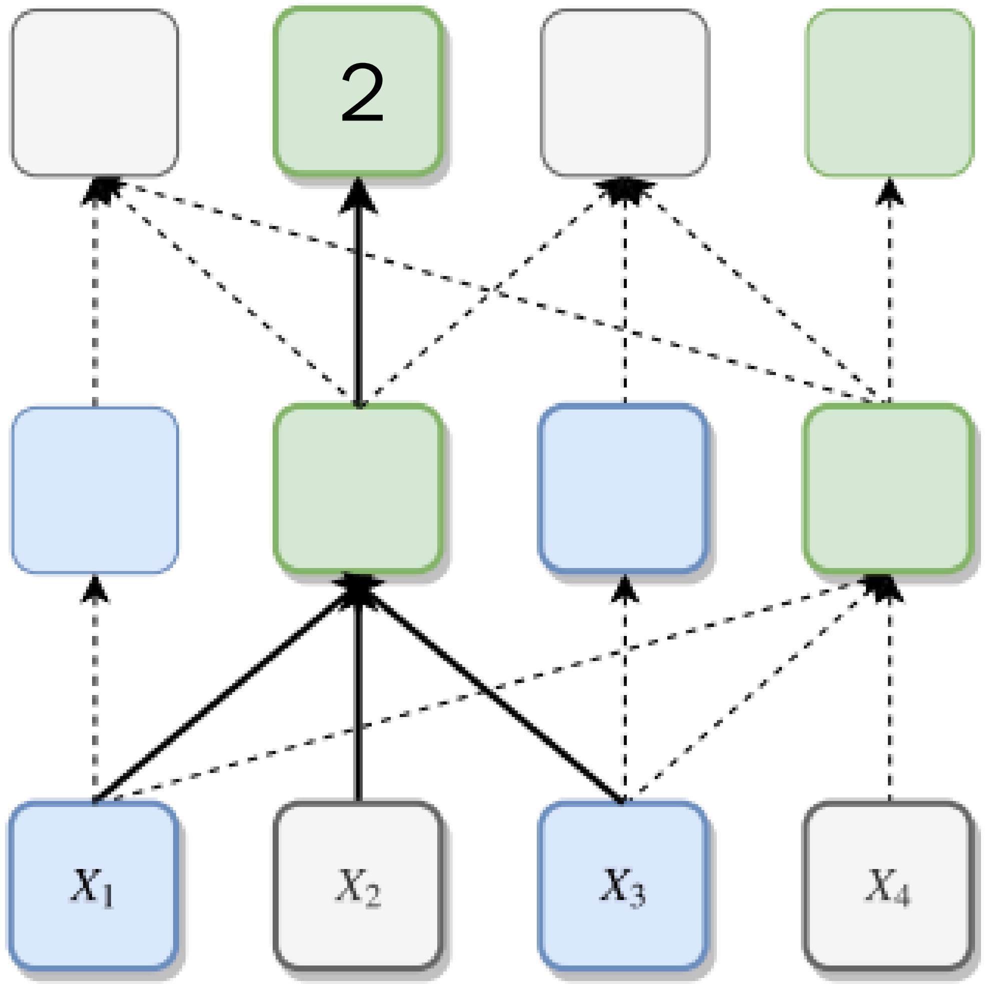
Discrete Autoregressive Flows

- Can stack multiple levels of autoregression
- Solid lines show receptive fields of the red block
- Dashed lines show other connections



Discrete Bipartite Flows

- Receptive field of 2 is only $x_{1:3}$
- Blue and green are each a set of transformed variables
- White blocks are not transformed



Training

- Like other normalizing flows, directly optimize the maximum likelihood
- The maximum likelihood of a discrete model is $\log p(\mathbf{y}) = \log p(f^{-1}(\mathbf{y}))$, (discrete change of variables)
  - The parameters of *f* and of the base distribution *p* can be optimized
  - Note: notation for *p* is often less clear than earlier papers

Training – Gradient Tricks

- μ and σ both produce discrete values. To train *f*, they must be backpropagated through.
  - Address this issue using the straight-through gradient estimator
  - Essentially, on the backward pass of the network pretend the discrete function is the identity function and just pass the gradients through.
- To compute μ and σ , produce *K* logits for each. The value is chosen using argmax:

$$\boldsymbol{\mu}_d = \text{one\_hot}(\text{argmax}(\theta_d))$$

- This isn't differentiable! To fix this, we can instead use the following differentiable function:

$$\frac{d\boldsymbol{\mu}_d}{d\theta_d} \approx \frac{d}{d\theta_d}\text{softmax}\left(\frac{\theta_d}{\tau}\right)$$

- This approaches an argmax as τ approaches 0. Experimentally, they use $\tau = 0.1$

Results

Experimental Settings

- For discrete autoregressive flows, use an autoregressive Categorical base distribution
- For discrete bipartite flows, use a factorized Categorial distribution
- Use $\sigma = 1$ for all experiments except character-level language modeling

Full-rank Discrete Distribution

- How well can the discrete flows fit full-rank discrete distributions?
- Sample true probabilities for *K* classes in *D* dimensions using Dirichlet distribution with $\alpha = 1$
- Transformer with 64 hidden units is used as a base model and for flow parameters
- Compute "nats" for negative log likelihood, indicating natural logarithm is used

	Autoregressive Base	Autoregressive Flow	Factorized Base	Bipartite Flow
$D = 2, K = 2$	**0.9**	**0.9**	1.3	**1.0**
$D = 5, K = 5$	7.7	**7.6**	8.0	**7.9**
$D = 5, K = 10$	10.7	**10.3**	11.5	**10.7**
$D = 10, K = 5$	15.9	**15.7**	16.6	**16.0**

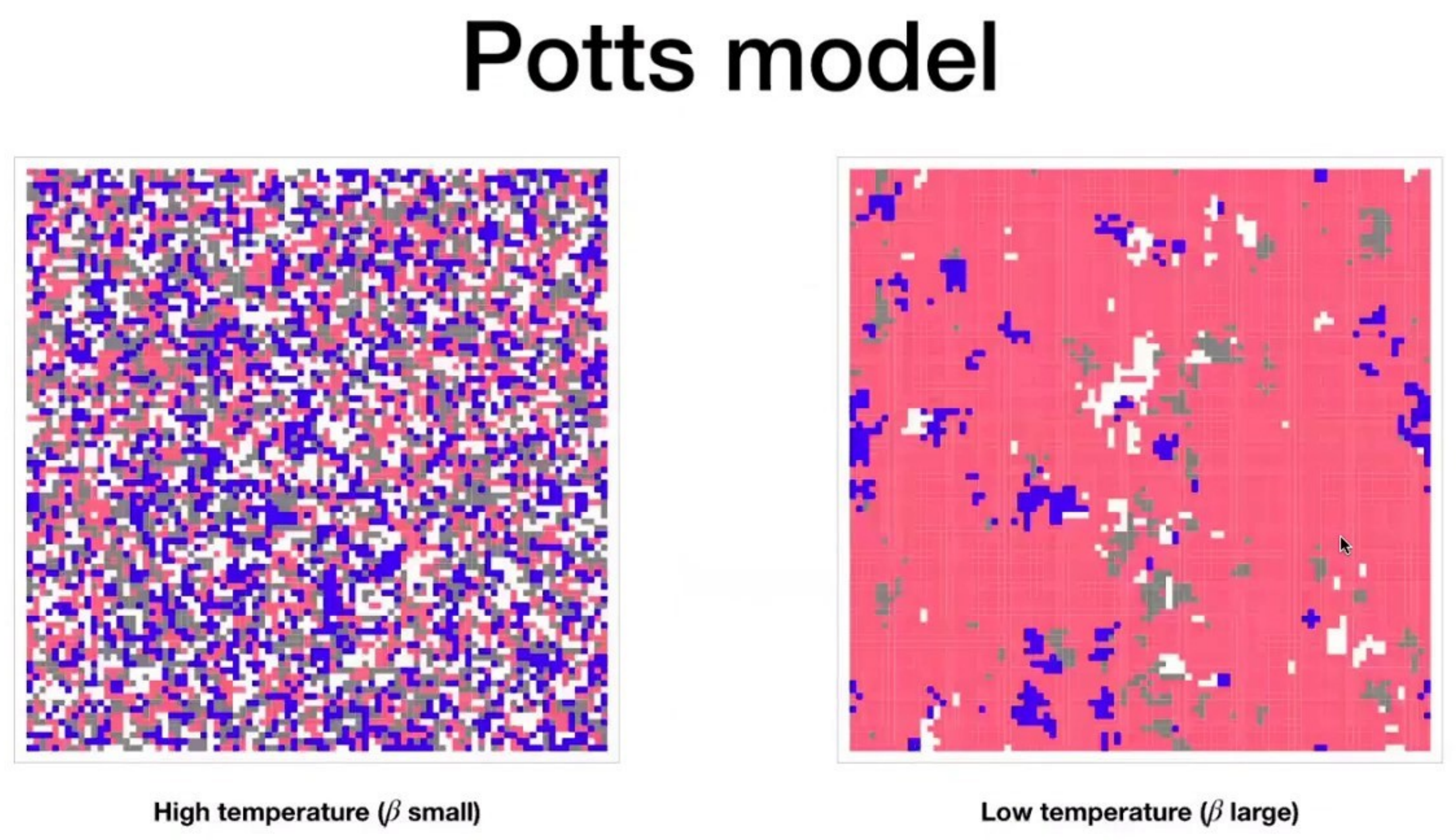
Negative Log Likelihood

Addition

- Base-10 addition using D=10 and D=20 digits.
- Addition is a right-to-left task, which disadvantages the base autoregressive model.
- Use an LSTM with 256 hidden units for D=10 and 512 for D=20 as a base.
- For D=10, the autoregressive base (left to right) achieves **4.0 nats** (negative log likelihood). The autoregressive flow achieves **0.2 nats.**
- A bipartite model achieves **4.0**, **3.17**, and **2.58** nats for 1, 2, and 4 flows.
- For D=20, the autoregressive base achieves **12.2 nats** (negative log likelihood). The autoregressive flow achieves **4.8 nats.**
- A bipartite model achieves **12.2**, **8.8**, **7.6**, and **5.08** nats for 1, 2, 4, and 8 flows.

Potts Model

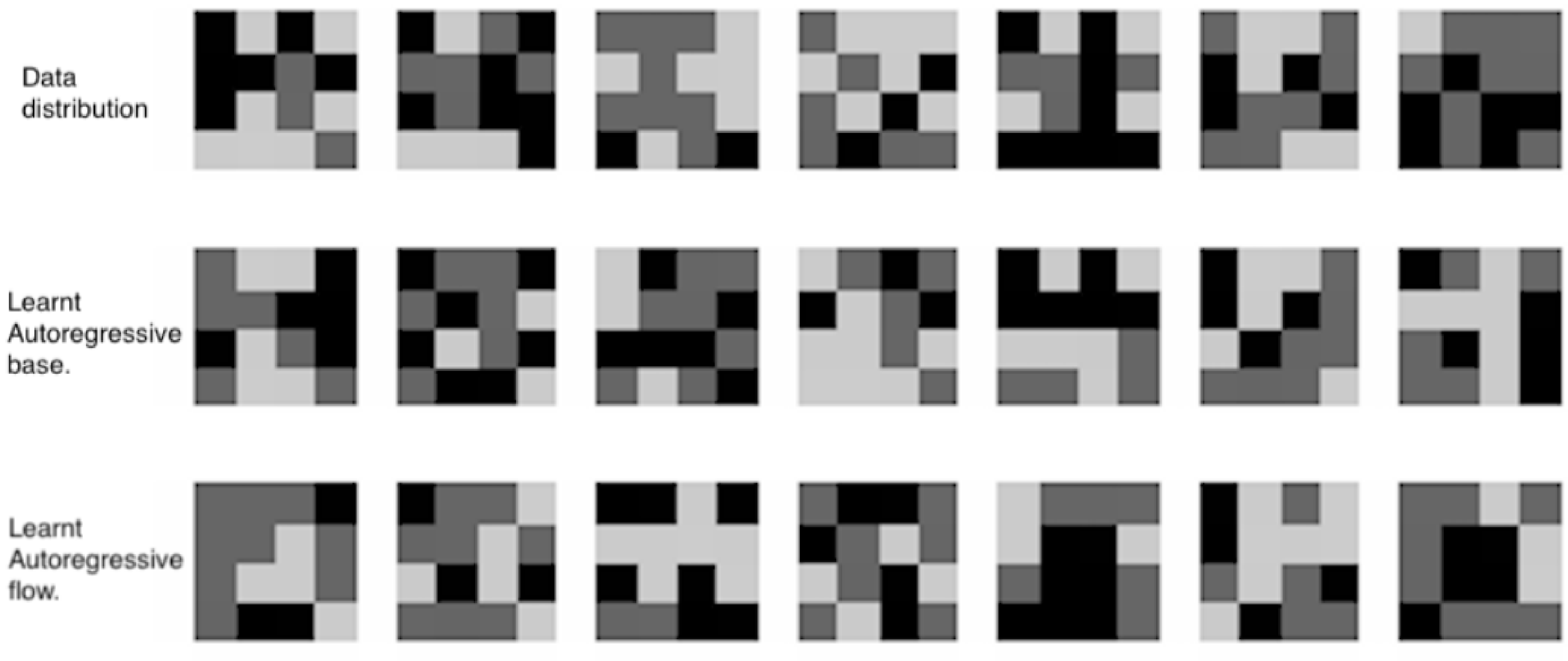
- Can the discrete flows be applied to models with untractable sampling and likelihood?
- Sample from the Potts model, which is a 2D Markov random field
- Samples are a *D* x *D* matrix, where the coupling between elements is *J*
- Data sampled using 500 steps of Metropolis-Hastings (MCMC)



Example of Potts Model [2]. $\beta = J$

Potts Model Results

	AR Base	AR Flow
number of states = 3		
$D = 9, J = 0.1$	9.27	**9.124**
$D = 9, J = 0.5$	3.79	**3.79**
$D = 16, J = 0.1$	16.66	**11.23**
$D = 16, J = 0.5$	6.30	**5.62**
number of states = 4		
$D = 9, J = 0.1$	11.64	**10.45**
$D = 9, J = 0.5$	5.87	**5.56**
number of states = 5		
$D = 9, J = 0.1$	13.58	**10.25**
$D = 9, J = 0.5$	7.94	**7.07**



Potts Model Samples

- 3 states, 4x4, $J = 0.1$
- Samples are indistinguishable from ground truth

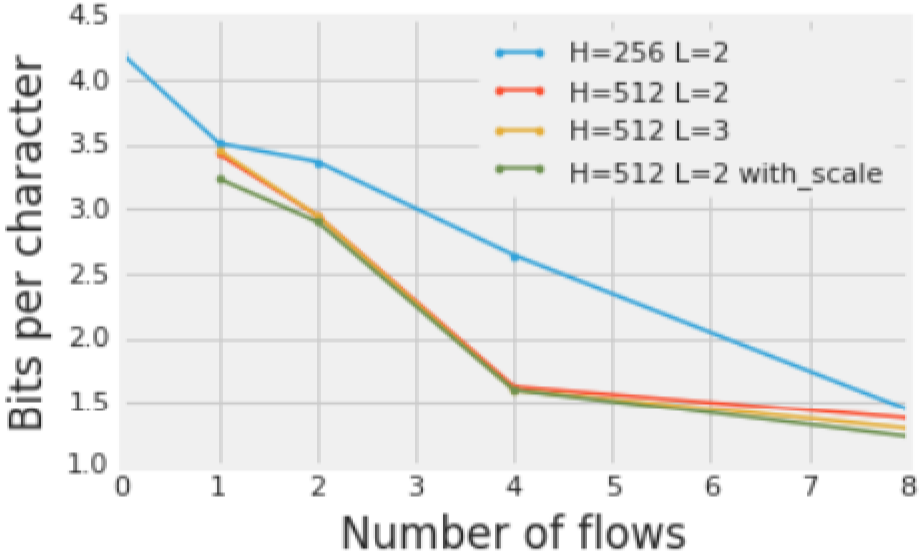
Character-Level Penn Treebank

- Goal is to model the Penn treebank, which has K = 51 characters.
- Data is split into sentences. In this work, sequence length is restricted to 288 (which is not explained)
- Compare to only other nonautoregressive language model [3], a VAE-based generative model which learns a normalizing flow in the latent space

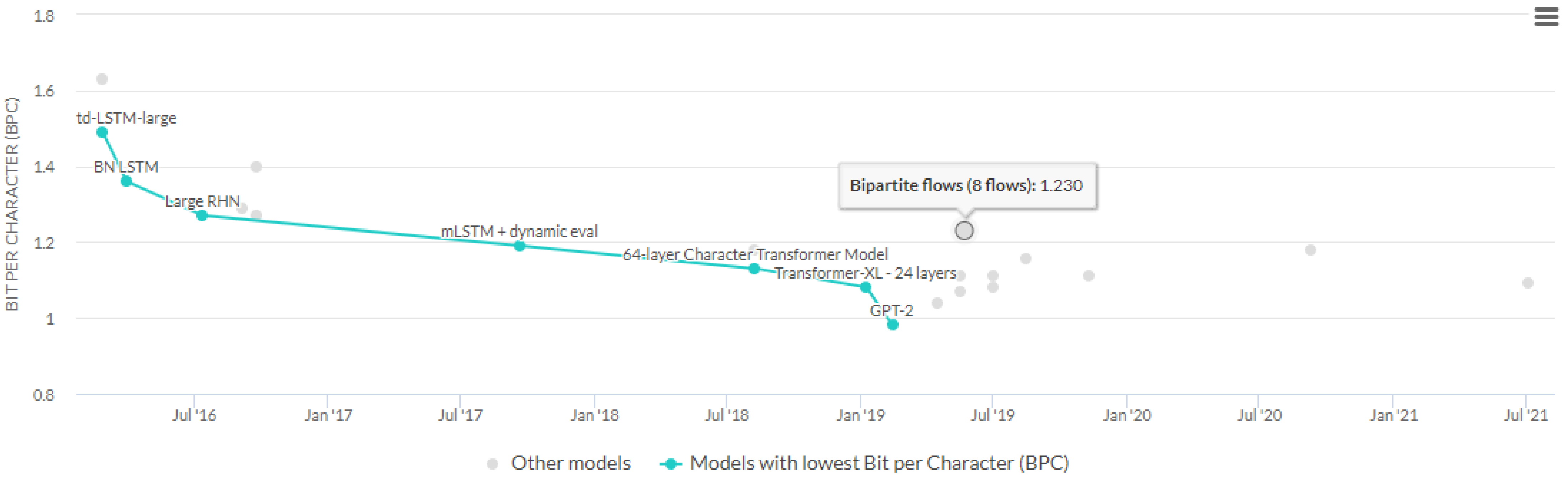
	Test NLL (bpc)	Generation
3-layer LSTM (Merity et al., 2018)	**1.18**[3]	3.8 min
Ziegler and Rush (2019) (AF/SCF)	1.46	-
Ziegler and Rush (2019) (IAF/SCF)	1.63	-
Bipartite flow	1.38	**0.17 sec**

Character level text8

- A larger text dataset (100M characters as opposed to 5M) which is intended for testing text compression algorithms
- Bipartite flows can generate much faster than autoregressive models



	bpc	Gen.
LSTM (Coojimans+2016)	1.43	19.8s
64-layer Transformer (Al-Rfou+2018)	**1.13**	35.5s
Bipartite flow (4 flows, w/ σ)	1.60	**0.15s**
Bipartite flow (8 flows, w/o σ)	1.29	**0.16s**
Bipartite flow (8 flows, w/ σ)	1.23	**0.16s**



Text8 – All paper results from [4]

Takeaway

- Motivation: Normalizing flows generally are only used for continuous distributions
- This can be extended to discrete distributions using a different change of variables formulation (with a Jacobian determinant!)
- Discrete autoregressive flows enable bidirectionality
- Discrete bipartite flows enable quick generation
- Future work:
  - Can inverse autoregressive flows be made discrete?
  - How to scale to many more classes? The straight-through estimator might not work on word sequences with 1000s of vocabulary tokens
  - Can other invertible discrete cryptographic functions be applied?

References

- [1] Tran, Dustin, et al. "Discrete flows: Invertible generative models of discrete data." *Advances in Neural Information Processing Systems* 32 (2019): 14719-14728.
- [2] ACM Youtube: “Session 6B - Efficient sampling and counting algorithms for the Potts model”
- [3] Ziegler, Zachary, and Alexander Rush. "Latent normalizing flows for discrete sequences." *International Conference on Machine Learning*. PMLR, 2019.
- [4] https://paperswithcode.com/sota/language-modelling-on-text8