Data Generation as Sequential Decision Making

Authored by:

Doina Precup
Philip Bachman

Abstract

We connect a broad class of generative models through their shared reliance on sequential decision making. Motivated by this view, we develop extensions to an existing model, and then explore the idea further in the context of data imputation – perhaps the simplest setting in which to investigate the relation between unconditional and conditional generative modelling. We formulate data imputation as an MDP and develop models capable of representing effective policies for it. We construct the models using neural networks and train them using a form of guided policy search. Our models generate predictions through an iterative process of feedback and refinement. We show that this approach can learn effective policies for imputation problems of varying difficulty and across multiple datasets.

1 Paper Body

Directed generative models are naturally interpreted as specifying sequential procedures for generating data. We traditionally think of this process as sampling, but one could also view it as making sequences of decisions for how to set the variables at each node in a model, conditioned on the settings of its parents, thereby generating data from the model. The large body of existing work on reinforcement learning provides powerful tools for addressing such sequential decision making problems. We encourage the use of these tools to understand and improve the extended processes currently driving advances in generative modelling. We show how sequential decision making can be applied to general prediction tasks by developing models which construct predictions by iteratively refining a working hypothesis under guidance from exogenous input and endogenous feedback. We begin this paper by reinterpreting several recent generative models as sequential decision making processes, and then show how changes inspired by this point of view can improve the performance of the LSTM-based model introduced in [3]. Next, we explore the connections between directed generative models and reinforcement learning more fully by developing an approach to training policies for sequential data imputation. We base our approach on formulating imputation as a finitehorizon Markov Decision Process

which one can also interpret as a deep, directed graphical model. We propose two policy representations for the imputation MDP. One extends the model in [3] by inserting an explicit feedback loop into the generative process, and the other addresses the MDP more directly. We train our models/policies using techniques motivated by guided policy pearch [9, 10, 11, 8]. We examine their qualitative and quantitative performance across imputation problems covering a range of difficulties (i.e. different amounts of data to impute and different ?missingness mechanisms?), and across multiple datasets. Given the relative paucity of existing approaches to the general imputation problem, we compare our models to each other and to two simple baselines. We also test how our policies perform when they use fewer/more steps to refine their predictions. As imputation encompasses both classification and standard (i.e. unconditional) generative modelling, our work suggests that further study of models for the general imputation problem is worthwhile. The performance of our models suggests that sequential stochastic construction of predictions, guided by both input and feedback, should prove useful for a wide range of problems. Training these models can be challenging, but lessons from reinforcement learning may bring some relief. 1

2

Directed Generative Models as Sequential Decision Processes

Directed generative models have grown in popularity relative to their undirected counter-parts [6, 14, 12, 4, 5, 16, 15] (etc.). Reasons include: the development of efficient methods for training them, the ease of sampling from them, and the tractability of bounds on their log-likelihoods. Growth in available computing power compounds these benefits. One can interpret the (ancestral) sampling process in a directed model as repeatedly setting subsets of the latent variables to particular values, in a sequence of decisions conditioned on preceding decisions. Each subsequent decision restricts the set of potential outcomes for the overall sequence. Intuitively, these models encode stochastic procedures for constructing plausible observations. This section formally explores this perspective. 2.1

Deep AutoRegressive Networks

The deep autoregressive networks investigated in [4] define distributions of the following form: T X Y p(x) = p(x—z)p(z), with p(z) = p0 (z0) pt (zt —z0 , ..., zt?1) (1) z

t=1

in which x indicates a generated observation and z0 , ..., zT represent latent variables in the model. The distribution p(x—z) may be factored similarly to p(z). The form of p(z) in Eqn. 1 can represent arbitrary distributions over the latent variables, and the work work in [4] mainly concerned approaches to parameterizing the conditionals pt (zt —z0 , ..., zt?1) that restricted representational power in exchange for computational tractability. To appreciate the generality of Eqn. 1, consider using zt that are univariate, multivariate, structured, etc. One can interpret any model based on this sequential factorization of p(z) as a non-stationary policy pt (zt —st) for selecting each action zt in a state st , with each st determined by all zt0 for t0 ¡ t, and train it using some

form of policy search. 2.2

Generalized Guided Policy Search

We adopt a broader interpretation of guided policy search than one might initially take from, e.g., [9, 10, 11, 8]. We provide a review of guided policy search in the supplementary material. Our expanded definition of guided policy search includes any optimization of the general form:

minimize E E E [‘(?, iq , ip)] + ? div (q(? —iq , ip), p(? —ip)) (2) p,q

iq ?Iq ip ?Ip (?—iq) ? ?q(? —iq ,ip)

in which p indicates the primary policy, q indicates the guide policy, Iq indicates a distribution over information available only to q, Ip indicates a distribution over information available to both p and q, ‘(?, iq , ip) computes the cost of trajectory ? in the context of iq /ip , and div(q(? —iq , ip), p(? —ip)) measures dissimilarity between the trajectory distributions generated by p/q. As ? ¿ 0 goes to infinity, Eqn. 2 enforces the constraint p(? —ip) = q(? —iq , ip), ??, ip , iq . Terms for controlling, e.g., the entropy of p/q can also be added. The power of the objective in Eq. 2 stems from two main points: the guide policy q can use information iq that is unavailable to the primary policy p, and the primary policy need only be trained to minimize the dissimilarity term div(q(? —iq , ip), p(? —ip)). For example, a directed model structured as in Eqn. 1 can be interpreted as specifying a policy for a finite-horizon MDP whose terminal state distribution encodes p(x). In this MDP, the state at time 1 ? t ? T +1 is determined by {z0 , ..., zt?1 }. The policy picks an action zt ? Zt at time 1 ? t ? T , and picks an action x ? X at time t = T + 1. I.e., the policy can be written as pt (zt —z0 , ..., zt?1) for 1 ? t ? T , and as p(x—z0 , ..., zT) for t = T + 1. The initial state z0 ? Z0 is drawn from p0 (z0). Executing the policy for a single trial produces a trajectory ? , {z0 , ..., zT , x}, and the distribution over xs from these trajectories is just p(x) in the corresponding directed generative model. The authors of [4] train deep autoregressive networks by maximizing a variational lower bound on the training set log-likelihood. To do this, they introduce a variational distribution q which provides q0 (z0 —x?) and qt (zt —z0 , ..., zt?1 , x?) for 1 ? t ? T , with the final step q(x—z0 , ..., zT , x?) given by a Dirac-delta at x? . Given these definitions, the training in [4] can be interpreted as guided policy search for the MDP described in the previous paragraph. Specifically, the variational distribution q provides a guide policy q(? —x?) over trajectories ? , {z0 , ..., zT , x? }: T Y q(? —x?) , q(x—z0 , ..., zT , x?)q0 (z0 —x?) qt (zt —z0 , ..., zt?1 , x?) (3) t=1

2

The primary policy p generates trajectories distributed according to: T Y p(?) , p(x—z0 , ..., zT)p0 (z0) pt (zt —z0 , ..., zt?1)

(4)

t=1

which does not depend on x? . In this case, x? corresponds to the guide-only information iq ? Iq in Eqn. 2. We now rewrite the variational optimization as:

minimize ? E E ? [‘(?, x?)] + KL(q(? —x?) —— p(?)) (5) p,q

x ?DX

? ?q(? —x)
?

where ‘(?, x) , 0 and DX indicates the target distribution for the terminal state of the primary policy p.1 When expanded, the KL term in Eqn. 5 becomes: KL(q(? —x?) —— p(?)) = (6) # ” T q0 (z0 —x?) X qt (zt —z0 , ..., zt?1 , x?) E ? log + ? log p(x? —z0 , ..., zT) log p0 (z0) p (z —z , ..., z) ? ?q(? —x) t t 0 t?1 t=1 Thus, the variational approach used in [4] for training directed generative models can be interpreted as a form of generalized guided policy search. As the form in Eqn. 1 can represent any finite directed generative model, the preceding derivation extends to all models we discuss in this paper.2 2.3

Time-reversible Stochastic Processes

One can simplify Eqn. 1 by assuming suitable forms for X and Z0 , ..., ZT . E.g., the authors of [16] proposed a model in which Zt ? X for all t and p0 (x0) was Gaussian. We can write their model as: p(xT) =

X
pT (xT —xT ?1)p0 (x0)
x0 ,...,xT ?1
TY ?1
pt (xt —xt?1)
(7)
t=1

where p(xT) indicates the terminal state distribution of the non-stationary, finite-horizon Markov process determined by {p0 (x0), p1 (x1 —x0), ..., pT (xT —xT ?1)}. Note that, throughout this paper, we (ab)use sums over latent variables and trajectories which could/should be written as integrals. The authors of [16] observed that, for any reasonably smooth target distribution DX and sufficiently large T , one can define a ?reverse-time? stochastic process qt (xt?1 —xt) with simple, time-invariant dynamics that transforms q(xT) , DX into the Gaussian distribution p0 (x0). This q is given by: q0 (x0) =

X
q1 (x0 —x1)DX (xT)
x1 ,...,xT
T Y
qt (xt?1 —xt) ? p0 (x0)
(8)
t=2

Next, we define q(?) as the distribution over trajectories ? , {x0 , ..., xT } generated by the reversetime process determined by {q1 (x0 —x1), ..., qT (xT ?1 —xT), DX (xT)}. We define p(?) as the distribution over trajectories generated by the ?forward-time? process in Eqn. 7. The training in [16] is equivalent to guided policy search using guide trajectories sampled from q, i.e. it uses the objective: ” # T ?1 q1 (x0 —x1) X DX (xT) qt+1 (xt —xt+1) minimize E log + + log (9) log p,q p0 (x0) pt (xt —xt?1) pT (xT —xT ?1) ? ?q(?) t=1 which corresponds to minimizing KL(q —— p). If the log-densities in Eqn. 9 are tractable, then this minimization can be done using

basic Monte-Carlo.h If, as in [16], the reverse-time processi q is not PT trained, then Eqn. 9 simplifies to: minimizep Eq(?) ? log p0 (x0) ? t=1 log pt (xt —xt?1) . This trick for generating guide trajectories exhibiting a particular distribution over terminal states xT ? i.e. running dynamics backwards in time starting from xT ? DX ? may prove useful in settings other than those considered in [16]. E.g., the LapGAN model in [1] learns to approximately invert a fixed (and information destroying) reverse-time process. The supplementary material expands on the content of this subsection, including a derivation of Eqn. 9 as a bound on Ex?DX [? log p(x)]. We could pull the ? log p(x? —z0 , ..., zT) term from the KL and put it in the cost '(?, x?), but we prefer the ?path-wise KL? formulation for its elegance. We abuse notation using KL(?(x = x?) —— p(x)) , ? log p(x?). 2 This also includes all generative models implemented and executed on an actual computer. 1

3

2.4

Learning Generative Stochastic Processes with LSTMs

The authors of [3] introduced a model for sequentially-deep generative processes. We interpret their model as a primary policy p which generates trajectories ? , {z0 , ..., zT , x} with distribution: p(?) , p(x—s? (?¡x))p0 (z0)

T Y

pt (zt), with ?¡x , {z0 , ..., zT }

(10)

t=1

in which ?¡x indicates a latent trajectory and s? (?¡x) indicates a state trajectory {s0 , ..., sT } computed recursively from ?¡x using the update st ? f? (st?1 , zt) for t ? 1. The initial state s0 is given by a trainable constant. Each state st , [ht ; vt] represents the joint hidden/visible state ht /vt of an LSTM and f? (state, input) computes a standard LSTM update.3 The authors of [3] defined all pt (zt) as isotropic Gaussians and defined the output distribution p(x—s? (?¡x)) as p(x—cT), where PT cT , c0 + t=1 ?? (vt). Here, c0 is a trainable constant and ?? (vt) is, e.g., an affine transform of vt . Intuitively, ?? (vt) transforms vt into a refinement of the ?working hypothesis? ct?1 , which gets updated to ct = ct?1 + ?? (vt). p is governed by parameters ? which affect f? , ?? , s0 , and c0 . The supplementary material provides pseudo-code and an illustration for this model. To train p, the authors of [3] introduced a guide policy q with trajectory distribution: q(? —x?) , q(x—s? (?¡x), x?)q0 (z0 —x?)

T Y

qt (zt —? st , x?), with ?¡x , {z0 , ..., zT }

(11)

t=1

in which s? (?¡x) indicates a state trajectory {? s0 , ..., s?T } computed recursively from ?¡x using the guide policy?s state update s?t ? f? (? st?1 , g? (s? (?¡t), x?)). In this update s?t?1 is the previous guide state and g? (s? (?¡t), x?) is a deterministic function of x? and the partial (primary) state

trajectory s? (?¡t) , {s0 , ..., st?1 }, which is computed recursively from ?¡t , {z0 , ..., zt?1 } using the state update st ? f? (st?1 , zt). The output distribution q(x—s? (?¡x), x?) is defined as a Dirac-delta at x? .4 Each qt (zt —? st , x?) is a diagonal Gaussian distribution with means and log-variances given by an affine function L? (? vt) of v?t . q0 (z0) is defined as identical to p0 (z0). q is governed by parameters ? which affect the state updates f? (? st?1 , g? (s? (?¡t), x?)) and the step distributions qt (zt —? st , x?). ? g? (s? (?¡t), x) corresponds to the ?read? operation of the encoder network in [3]. Using our definitions for p/q, the training objective in [3] is given by: " T # X qt (zt —? st , x?) ? minimize ? E E log ? log p(x —s(?¡x)) p,q x ?DX ? ?q(? —x?) pt (zt) t=1

(12)

which can be written more succinctly as Ex? ?DX KL(q(? —x?) —— p(?)). This objective upper-bounds P Ex? ?DX [? log p(x?)], where p(x) , ?¡x p(x—s? (?¡x))p(?¡x). 2.5

Extending the LSTM-based Generative Model

QT We propose changing p in Eqn. 10 to: p(?) , p(x—s? (?¡x))p0 (z0) t=1 pt (zt —st?1). We define pt (zt —st?1) as a diagonal Gaussian distribution with means and log-variances given by an affine function L? (vt?1) of vt?1 (remember that st , [ht ; vt]), and we define p0 (z0) as an isotropic Gaussian. We set s0 using s0 ? f? (z0), where f? is a trainable function (e.g. a neural network). Intuitively, our changes make the model more like a typical policy by conditioning its ?action? zt on its state st?1 , and upgrade the model to an infinite mixture by placing a distribution over its initial state s0 . We also consider using ct , L? (ht), which transforms the hidden part of the LSTM state st directly into an observation. This makes ht a working memory in which to construct an observation. The supplementary material provides pseudo-code and an illustration for this model. We train this model by optimizing the objective: # " T qt (zt —? st , x?) q0 (z0 —x?) X ? + log ? log p(x —s(?¡x)) minimize ? E E log p,q x ?DX ? ?q(? —x?) p0 (z0) pt (zt —st?1) t=1

(13)

3 For those unfamiliar with LSTMs, a good introduction can be found in [2]. We use LSTMs including input gates, forget gates, output gates, and peephole connections for all tests presented in this chapter. 4 It may be useful to relax this assumption.

4

where we now have to deal with pt (zt —st?1), p0 (z0), and q0 (z0 —x?), which could be treated as constants in the model from [3]. We define q0 (z0 —x?) as a diagonal Gaussian distribution whose means and log-variances are given by a trainable function g? (x?). When trained for the binarized MNIST benchmark used in [3], our extended model scored a negative log-likelihood of 85.5 on the test set.5 For comparison, the score reported in [3] was 87.4.6 After finetuning the variational distribution (i.e. q) on the test set, our model?s score improved to 84.8, which is quite strong considering it is an upper bound. For comparison, see the best upper bound reported for this benchmark in [15], which

was 85.1. When the model used the alternate cT , L? (hT), the raw/finetuned test scores were 85.9/85.3. Fig. 1 shows samples from the model. Model/test code is available at http://github.com/Philip-Bachman/ Sequential-Generation.

3

Figure 1: The left block shows ?(ct) for t ? {1, 3, 5, 9, 16}, for a policy p with ct , c0 + Pt 0 t0 =1 L? (vt). The right block is analogous, for a model using ct , L? (ht).

Developing Models for Sequential Imputation

The goal of imputation is to estimate p(xu —xk), where x , [xu ; xk] indicates a complete observation with known values xk and missing values xu . We define a mask m ? M as a (disjoint) partition of x into xu /xk . By expanding xu to include all of x, one recovers standard generative modelling. By shrinking xu to include a single element of x, one recovers standard classification/regression. Given distribution DM over m ? M and distribution DX over x ? X , the objective for imputation is:

minimize E E ? log p(xu —xk) (14) p

x?DX m?DM

We now describe a finite-horizon MDP for which guided policy search minimizes a bound on the objective in Eqn. 14. The MDP is defined by mask distribution DM , complete observation distribution DX , and the state spaces {Z0 , ..., ZT } associated with each of T steps. Together, DM and DX define a joint distribution over initial states and rewards in the MDP. For the trial determined by x ? DX and m ? DM , the initial state z0 ? p(z0 —xk) is selected by the policy p based on the known values xk . The cost ‘(?, xu , xk) suffered by trajectory ? , {z0 , ..., zT } in the context (x, m) is given by ? log p(xu —?, xk), i.e. the negative log-likelihood of p guessing the missing values xu after following trajectory ? , while seeing the known values xk . QT We consider a policy p with trajectory distribution p(? —xk) , p(z0 —xk) t=1 p(zt —z0 , ..., zt?1 , xk), where xk is determined by x/m for the current trial and p can?t observe the missing values xu . With these definitions, we can find an approximately optimal imputation policy by solving:

minimize E E E ? log p(xu —?, xk) (15) p

x?DX m?DM ? ?p(? —xk)

I.e. the expected negative log-likelihood of making a correct imputation on any given trial. This is a valid, but loose, upper bound on the imputation objective in Eq. 14 (from Jensen?s inequality). We can tighten the bound by introducing a guide policy (i.e. a variational distribution). As with the unconditional generative models in Sec. 2, we train p to imitate a guide policy q shaped by additional information (here it?s xu). This q generates trajectories with distribution q(? —xu , xk) , QT q(z0 —xu , xk) t=1 q(zt —z0 , ..., zt?1 , xu , xk). Given this p and q, guided policy search solves:

u minimize E E E [? log q(x —?, iq , ip)] + KL(q(? —iq , ip) —— p(? —ip)) (16) p,q

x?DX m?DM

? ?q(? —iq ,ip)

where we define iq , xu , ip , xk , and q(xu —?, iq , ip) , p(xu —?, ip). 5 6

Data splits from: http://www.cs.toronto.edu/?larocheh/public/datasets/binarized_mnist The model in [3] significantly improves its score to 80.97 when using an image-specific architecture.

5

3.1

A Direct Representation for Sequential Imputation Policies

We define an imputation trajectory as c? , {c0 , ..., cT }, where each partial imputation ct ? X is computed from a partial step trajectory ?¡t , {z1 , ..., zt }. A partial imputation ct?1 encodes the policy?s guess for the missing values xu immediately prior to selecting step zt , and cT gives the policy?s final guess. At each step of iterative refinement, the policy selects a zt based on ct?1 and the known values xk , and then updates its guesses to ct based on ct?1 and zt . By iteratively refining its guesses based on feedback from earlier guesses and the known values, the policy can construct complexly structured distributions over its final guess cT after just a few steps. This happens naturally, without any post-hoc MRFs/CRFs (as in many approaches to structured prediction), and without sampling values in cT one at a time (as required by existing NADE-type models [7]). This property of our approach should prove useful for many tasks. We consider two ways of updating the guesses in ct , mirroring those described in Sec. 2. The first way sets ct ? ct?1 + ?? (zt), where ?? (zt) is a trainable function. We set c0 , [cu0 ; ck0] using a trainable bias. The second way sets ct ? ?? (zt). We indicate models using the first type of update with the suffix -add, and models using the second type of update with -jump. Our primary policy p? selects zt at each step 1 ? t ? T using p? (zt —ct?1 , xk), which we restrict to be a diagonal Gaussian. This is a simple, stationary policy. Together, the step selector p? (zt —ct?1 , xk) and the imputation constructor ?? (zt) fully determine the behaviour of the primary policy. The supplementary material provides pseudo-code and an illustration for this model. We construct a guide policy q similarly to p. The guide policy shares the imputation constructor ?? (zt) with the primary policy. The guide policy incorporates additional information x , [xu ; xk], i.e. the complete observation for which the primary policy must reconstruct some missing values. The guide policy chooses steps using q? (zt —ct?1 , x), which we restrict to be a diagonal Gaussian. We train the primary/guide policy components ?? , p? , and q? simultaneously on the objective:

u u u k k minimize E E E [? log q(x —cT)] + KL(q(? —x , x) —— p(? —x)) (17) ?,?

x?DX m?DM

? ?q? (? —xu ,xk)

where q(xu —cuT) , p(xu —cuT). We train our models using Monte-Carlo roll-outs of q, and stochastic backpropagation as in [6, 14]. Full implementations and test code are available from http:// github.com/Philip-Bachman/Sequential-Generation. 3.2

Representing Sequential Imputation Policies using LSTMs

To make it useful for imputation, which requires conditioning on the exogenous information xk , we modify the LSTM-based model from Sec. 2.5 to include

a ?read? operation in its primary policy p. We incorporate a read operation by spreading p over two LSTMs, pr and pw , which respectively ?read? and ?write? an imputation trajectory c? , {c0 , ..., cT }. Conveniently, the guide policy q for this model takes the same form as the primary policy?s reader pr . This model also includes an ?infinite mixture? initialization step, as used in Sec. 2.5, but modified to incorporate conditioning on x and m. The supplementary material provides pseudo-code and an illustration for this model. Following the infinite mixture initialization step, a single full step of execution for p involves several k substeps: first p updates the reader state using srt ? f?r (srt?1 , ??r (ct?1 , sw t?1 , x)), then p selects a r w w w step zt ? p? (zt —vt), then p updates the writer state using st ? f? (st?1 , zt), and finally p updates r,w r,w its guesses by setting ct ? ct?1 + ??w (vtw) (or ct ? ??w (hw , [hr,w t ; vt] t)). In these updates, st r,w refer to the states of the (r)reader and (w)writer LSTMs. The LSTM updates f? and the read/write operations ??r,w are governed by the policy parameters ?. We train p to imitate trajectories sampled from a guide policy q. The guide policy shares the primary policy?s writer updates f?w and write operation ??w , but has its own reader updates f?q and read operation ??q . At each step, the guide policy: updates the guide state sqt ? f?q (sqt?1 , ??q (ct?1 , sw t?1 , x)), w w then selects zt ? q? (zt —vtq), then updates the writer state sw ? f (s , z), and finally updates t t?1 t ? its guesses ct ? ct?1 + ??w (vtw) (or ct ? ??w (hw)). As in Sec. 3.1, the guide policy?s read opt eration ??q gets to see the complete observation x, while the primary policy only gets to see the known values xk . We restrict the step distributions p? /q? to be diagonal Gaussians whose means and log-variances are affine functions of vtr /vtq . The training objective has the same form as Eq. 17. 6

350 300 250 200

Imputation NLL vs. Available Information

88

TM-orc TM-hon VAE-imp GPSI-add GPSI-jump LSTM-add LSTM-jump

86 84 82

Imputation NLL vs. Available Information

98

GPSI-add GPSI-jump LSTM-add LSTM-jump

94 92 90

78 88

76

86

74 100

84

72 0.60

0.65

GPSI-add GPSI-jump

80

150

50 0.55

The Effect of Increased Refinement Steps

96
0.70 0.75 0.80 Mask Probability
0.85
0.90
0.95
70 0.55
(a)
0.60
0.65
0.70 0.75 0.80 Mask Probability
0.85
0.90
0.95
(b)
82 0
2
4
6 8 10 Refinement Steps
12
14
16
(c)

Figure 2: (a) Comparing the performance of our imputation models against several baselines, using MNIST digits. The x-axis indicates the % of pixels which were dropped completely at random, and the scores are normalized by the number of imputed pixels. (b) A closer view of results from (a), just for our models. (c) The effect of increased iterative refinement steps for our GPSI models.

4

Experiments

We tested the performance of our sequential imputation models on three datasets: MNIST (28x28), SVHN (cropped, 32x32) [13], and TFD (48x48) [17]. We converted images to grayscale and shift/scaled them to be in the range [0...1] prior to training/testing. We measured the imputation log-likelihood log q(xu —cuT) using the true missing values xu and the models? guesses given by ?(cuT). We report negative log-likelihoods, so lower scores are better in all of our tests. We refer to variants of the model from Sec. 3.1 as GPSI-add and GPSI-jump, and to variants of the model from Sec. 3.2 as LSTM-add and LSTM-jump. Except where noted, the GPSI models used 6 refinement steps and the LSTM models used 16.7 We tested imputation under two types of data masking: missing completely at random (MCAR) and missing at random (MAR). In MCAR, we masked pixels uniformly at random from the source images, and indicate removal of d% of the pixels by MCAR-d. In MAR, we masked square regions, with the occlusions located uniformly at random within the borders of the source image. We indicate occlusion of a d ? d square by

MAR-d. On MNIST, we tested MCAR-d for d ? {50, 60, 70, 80, 90}. MCAR-100 corresponds to unconditional generation. On TFD and SVHN we tested MCAR-80. On MNIST, we tested MAR-d for d ? {14, 16}. On TFD we tested MAR-25 and on SVHN we tested MAR-17. For test trials we sampled masks from the same distribution used in training, and we sampled complete observations from a held-out test set. Fig. 2 and Tab. 1 present quantitative results from these tests. Fig. 2(c) shows the behavior of our GPSI models when we allowed them fewer/more refinement steps.

LSTM-add LSTM-jump GPSI-add GPSI-jump VAE-imp
MNIST MAR-14 MAR-16 170 167 172 169 177 175 183 177 374 394
TFD MCAR-80 MAR-25 1381 1377 ? ? 1390 1380 1394 1384 1416 1399
SVHN MCAR-80 MAR-17 525 568 ? ? 531 569 540 572 567 624

Table 1: Imputation performance in various settings. Details of the tests are provided in the main text. Lower scores are better. Due to time constraints, we did not test LSTM-jump on TFD or SVHN. These scores are normalized for the number of imputed pixels. We tested our models against three baselines. The baselines were ?variational auto-encoder imputation?, honest template matching, and oracular template matching. VAE imputation ran multiple steps of VAE reconstruction, with the known values held fixed and the missing values re-estimated with each reconstruction step.8 After 16 refinement steps, we scored the VAE based on its best 7 GPSI stands for ?Guided Policy Search Imputer?. The tag ?-add? refers to additive guess updates, and ?-jump? refers to updates that fully replace the guesses. 8 We discuss some deficiencies of VAE imputation in the supplementary material.

7

(a)

(b)

(c)

Figure 3: This figure illustrates the policies learned by our models. (a): models trained for (MNIST, MAR-16). From top?bottom the models are: GPSI-add, GPSI-jump, LSTM-add, LSTM-jump. (b): models trained for (TFD, MAR-25), with models in the same order as (a) ? but without LSTMjump. (c): models trained for (SVHN, MAR-17), with models arranged as for (b). guesses. Honest template matching guessed the missing values based on the training image which best matched the test image?s known values. Oracular template matching was like honest template matching, but matched directly on the missing values. Our models significantly outperformed the baselines. In general, the LSTM-based models outperformed the more direct GPSI models. We evaluated the log-likelihood of imputations produced by our models using the lower bounds provided by the variational objectives with respect to which they were trained. Evaluating the template-based imputations was straightforward. For VAE imputation, we used the expected log-likelihood of the imputations sampled from multiple runs of the 16-step imputation process. This provides a valid, but loose, lower bound on their log-likelihood. As shown in Fig. 3, the imputations produced by our models appear promising. The imputations are generally of high quality, and the models are capable of capturing strongly

multi-modal reconstruction distributions (see subfigure (a)). The behavior of GPSI models changed intriguingly when we swapped the imputation constructor. Using the -jump imputation constructor, the imputation policy learned by the direct model was rather inscrutable. Fig. 2(c) shows that additive guess updates extracted more value from using more refinement steps. When trained on the binarized MNIST benchmark discussed in Sec. 2.5, i.e. with binarized images and subject to MCAR-100, the LSTMadd model produced raw/fine-tuned scores of 86.2/85.7. The LSTM-jump model scored 87.1/86.3. Anecdotally, on this task, these ?closed-loop? models seemed more prone to overfitting than the ?open-loop? models in Sec. 2.5. The supplementary material provides further qualitative results.

5

Discussion

We presented a point of view which links methods for training directed generative models with policy search in reinforcement learning. We showed how our perspective can guide improvements to existing models. The importance of these connections will only grow as generative models rapidly increase in structural complexity and effective decision depth. We introduced the notion of imputation as a natural generalization of standard, unconditional generative modelling. Depending on the relation between the data-to-generate and the available information, imputation spans from full unconditional generative modelling to classification/regression. We showed how to successfully train sequential imputation policies comprising millions of parameters using an approach based on guided policy search [9]. Our approach outperforms the baselines quantitatively and appears qualitatively promising. Incorporating, e.g., the local read/write mechanisms from [3] should provide further improvements. 8

2 References

[1] Emily L Denton, Soumith Chintala, Arthur Szlam, and Robert Fergus. Deep generative models using a laplacian pyramid of adversarial networks. arXiv:1506.05751 [cs.CV], 2015. [2] Alex Graves. Generating sequences with recurrent neural networks. arXiv:1308.0850 [cs.NE], 2013. [3] Karol Gregor, Ivo Danihelka, Alex Graves, and Daan Wierstra. Draw: A recurrent neural network for image generation. In International Conference on Machine Learning (ICML), 2015. [4] Karol Gregor, Ivo Danihelka, Andriy Mnih, Charles Blundell, and Daan Wierstra. Deep autoregressive networks. In International Conference on Machine Learning (ICML), 2014. [5] Diederik P Kingma, Danilo J Rezende, Shakir Mohamed, and Max Welling. Semi-supervised learning with deep generative models. In Advances in Neural Information Processing Systems (NIPS), 2014. [6] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In International Conference on Learning Representations (ICLR), 2014. [7] Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. In International Conference on Machine Learning (ICML), 2011. [8] Sergey Levine and Pieter Abbeel. Learning neural network policies

with guided policy search under unknown dynamics. In Advances in Neural Information Processing Systems (NIPS), 2014. [9] Sergey Levine and Vladlen Koltun. Guided policy search. In International Conference on Machine Learning (ICML), 2013. [10] Sergey Levine and Vladlen Koltun. Variational policy search via trajectory optimization. In Advances in Neural Information Processing Systems (NIPS), 2013. [11] Sergey Levine and Vladlen Koltun. Learning complex neural network policies with trajectory optimization. In International Conference on Machine Learning (ICML), 2014. [12] Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. In International Conference on Machine Learning (ICML), 2014. [13] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. NIPS Workshop on Deep Learning and Unsupervised Feature Learning, 2011. [14] Danilo Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In International Conference on Machine Learning (ICML), 2014. [15] Danilo J Rezende and Shakir Mohamed. Variational inference with normalizing flows. In International Conference on Machine Learning (ICML), 2015. [16] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In International Conference on Machine Learning (ICML), 2015. [17] Joshua Susskind, Adam Anderson, and Geoffrey E Hinton. The toronto face database. 2010.

9