Using Synthetic Data to Train Neural Networks is Model-Based Reasoning

Tuan Anh Le*, Atılım Güneş Baydin*, Robert Zinkov† and Frank Wood*
*Department of Engineering Science
University of Oxford, Parks Road, OX1 3PJ Oxford, UK
Email: {tuananh, gunes, fwood}@robots.ox.ac.uk
†School of Informatics and Computing
Indiana University, 919 E 10th Street, Bloomington, IN 47408, USA
Email: zinkov@iu.edu

***Abstract*—We draw a formal connection between using synthetic training data to optimize neural network parameters and approximate, Bayesian, model-based reasoning. In particular, training a neural network using synthetic data can be viewed as learning a proposal distribution generator for approximate inference in the synthetic-data generative model. We demonstrate this connection in a recognition task where we develop a novel Captcha-breaking architecture and train it using synthetic data, demonstrating both state-of-the-art performance and a way of computing task-specific posterior uncertainty. Using a neural network trained this way, we also demonstrate successful breaking of real-world Captchas currently used by Facebook and Wikipedia. Reasoning from these empirical results and drawing connections with Bayesian modeling, we discuss the robustness of synthetic data results and suggest important considerations for ensuring good neural network generalization when training with synthetic data.**

I. Introduction

Neural networks are powerful regressors [1]. Training a neural network for regression means finding values for its free parameters using supervised learning techniques. This generally requires a large amount of labeled training data. Generally the harder the task, the larger the neural network, and the more training data required.

When labeled training data are scarce, one must either generate and use synthetic data to train, or resort to unsupervised generative modeling and generally slow test-time inference since it must be run afresh for new data. The deep learning community has reported remarkable results taking the former approach, either in the limited form of data augmentation [2, 3], where a dataset is artificially enlarged using label-preserving transformations, or training models solely on synthetic data, such as the groundbreaking work on text recognition in the wild [4, 5, 6], which was achieved by training a neural network to recognize text using synthetically generated realistic renders. Goodfellow et al. [7] addressed recognition of house numbers in Google Street View images in a supervised fashion, also solving reCaptcha [8] images using synthetic data to train a recognition network from image to latent text. That the authors were Google employees meant that they had access to the true reCaptcha generative model and thus could generate millions of labeled instances for use in a standard supervised-learning pipeline. More recently, Stark et al. [9] also used synthetic data for Captcha-solving and Wang et al. [10] for font identification.

A contribution of this paper is to point out that this kind of use of synthetic data to train a neural network under a standard loss is, in fact, equivalent to training an artifact to do amortized approximate inference, in the sense of Gershman and Goodman [11], for the generative model corresponding to the synthetic data generator. This relationship forms the basis of our recent work on inference compilation for probabilistic programming [12] and is also noted by both Paige and Wood [13] and Papamakarios and Murray [14], where approximate inference guided by neural proposals is the goal rather than training neural networks using synthetic data. A consequence of this is that there is no need to ever reuse training data, as “infinite” labeled training data can be generated at training time from the generative model. Another contribution we make is a suggestion for how to take advantage of this framework by running a neural network more than once at test time to compute task-specific uncertainties of interest.

These contributions can also be seen as a reminder and guidance to the neural network community as it continues to move towards tackling unsupervised inference and problems in which labeled training data are difficult or impossible to obtain. Towards this end, we examine experimental findings that highlight problems that are likely to arise when using synthetic data to train neural networks. We discuss these problems in terms of the brittleness demonstrated to exist for deep neural networks, for example by Szegedy et al. [15], who showed that perceptually indistinguishable variations in neural network input can lead to profound changes in output. We also discuss model misspecification in the Bayesian sense [16].

The paper structure is as follows. In Section II, we develop a probabilistic synthetic data generative model and suggest a single, flexible neural network architecture for Captcha-breaking. In Section III, we train each such model independently using training data derived from running the synthetic data generator with parameters set to produce the corresponding style. These neural networks are shown to produce extremely good breaking performance, both in terms of accuracy and speed, well beyond standard computer vision pipeline results and comparable to recent deep learning results. We then discuss and demonstrate the brittleness of these regressors. We demonstrate improved robustness by focusing on and improving the generative model. In Section IV, we illustrate the connection of the demonstrated brittleness with Bayesian model mismatch. We end by explaining how the learned neural network can be used to perform sample-based approximate inference.

II. Captcha-breaking

Assuming no access to the true Captcha [21] generating system and a paucity of labeled training data, how does one go about breaking Captchas? A hint appears in the probabilistic programming community's approach to procedural graphics [22] where a generative model for Captchas is proposed and then general purpose Markov chain Monte Carlo (MCMC) Bayesian inference is used to computationally inefficiently invert the said model. We will make the argument that this is, effectively, the same as generating synthetic training data in the manner of Jaderberg et al. [4, 5] to train a neural network that regresses to the latent Captcha variables. In either case, developing a flexible, well-calibrated synthetic training data generator is our first concern.

A. Generating synthetic training data

Our synthetic data generative model for Captcha specifies joint densities $p_s(x, y)$, parameterized by style s, that describe how to generate both the latent random variable x and the corresponding Captcha image y. Referring to the first row of Table I, style s pertains to different schemes (e.g., Baidu, eBay, Wikipedia, Facebook) involving distinct character ranges, fonts, kerning, deformations, and noise. Note that in the following equations we omit the style subscript while keeping in mind that there is a separate unique model for each style. The latent structured random variable $x = \{L, \epsilon_{1:K}, i_{1:L}\}$ includes L, the number of letters, $\epsilon_{1:K}$, a multidimensional structured parameter set controlling Captcha-rendering parameters such as kerning and various style-specific deformations, and $i_{i:L}$, letter identities. Given these, we use a custom stochastic Captcha renderer $\mathcal{R}$ to generate each Captcha image y, this renderer and its fidelity being the primary component of the synthetic data generation effort. The corresponding per-style synthetic data generator corresponds to the model

$$x \sim p(x) \tag{1}$$

$$y|x \sim \mathcal{R}(x) \ , \tag{2}$$

where $p(x)$ is a style-specific prior distribution over the latent variables including the character identities. For each different style shown in Table I, we use different settings of the prior parameters to drive the Captcha renderer. In particular, the model places style-specific uniform distributions over different intervals for L, $\epsilon_{1:K}$, and $i_{1:L}$. This is the mechanism for generating synthetic training data $\{(x^{(n)}, y^{(n)})\}$. Note that $p(y|x)$ cannot be evaluated for a given y, rather only sampled.

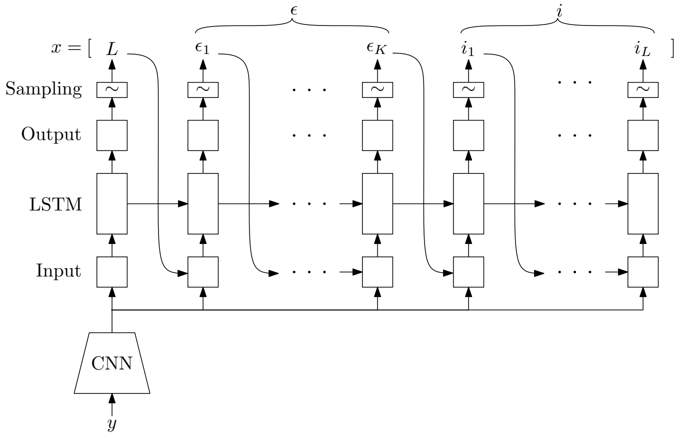


Fig. 1. Neural network architecture mapping the Captcha image y to the latent variables x of interest.

B. Neural network architecture

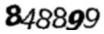
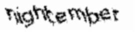
Our Captcha-breaking neural network is designed taking into account architectures that have been shown to perform well on image inputs and variable-length output sequences [23, 24]. Specifically, we choose a combination of convolutional neural networks (CNNs) and recurrent neural networks.

The core of our neural architecture (Figure 1) is a long short-term memory (LSTM) network [25], the output of which at each time step is passed through output layers corresponding one-to-one to the components of the latent variable x in the generative model (i.e., number of letters L, rendering parameters $\epsilon_{1:K}$, and letter identities $i_{1:L}$) that constitute the inputs to the Captcha renderer. Since the latent variable x has $T = 1 + K + L$ components, where K is style-specific and L is instance-specific, the LSTM is run for T time steps, and we represent by $x_{1:T}$ the components of the latent x at each time step. The output layers are fully-connected layers followed by a softmax function, distinct for each latent variable, that parameterize a discrete probability distribution. Since the LSTM has a fixed-dimensional output, these output layers allow us to match the dimensions of the discrete distributions for the corresponding latent variables.

A CNN is used to embed the Captcha image y into a fixed-dimensional embedding vector CNN(y). At each time step, the LSTM input is constructed as the concatenation of the image embedding CNN(y), the value of the latent variable x_{t-1} of the previous time step, and a label vector $\{0, 1\}^D$ corresponding to each x_t. During training, all $x_{1:T}$ are provided to the network in a way similar to that used by Reed and de Freitas [26], using the actual values that generated the synthetic image y. At test time, the values of x_t are sampled from the corresponding discrete probability distribution.

We denote the combined set of parameters of the overall architecture θ and its forward propagation function η, so given an input y, the output of the softmax layer at time step t corresponding to x_t is $\eta_{\theta,t}(y)$. In the running example of Figure 1, $x_1 = L$, $x_{2:(2+K-1)} = \epsilon_{1:K}$, and $x_{(2+K):(2+K+L-1)} = i_{1:L}$.

TABLE I
SYNTHETIC CAPTCHA BREAKING RESULTS. RR: RECOGNITION RATE, BT: BREAKING TIME.

Style		Baidu (2011)	Baidu (2013)	eBay	Yahoo	reCaptcha	Wikipedia	Facebook
								
Our method	RR	99.8%	99.9%	99.2%	98.4%	96.4%	93.6%	91.0%
	BT	72 ms	67 ms	122 ms	106 ms	78 ms	90 ms	90 ms
Bursztein et al. [17]	RR	38.68%	55.22%	51.39%	5.33%	22.67%	28.29%	
	BT	3.94 s	1.9 s	2.31 s	7.95 s	4.59 s		
Starostenko et al. [18]	RR				91.5%	54.6%		
	BT					< 0.5 s		
Gao et al. [19]	RR	34%			55%	34%		
Gao et al. [20]	RR		51%		36%			
	BT		7.58 s		14.72 s			
Goodfellow et al. [7]	RR					99.8%		
Stark et al. [9]	RR					90%		

C. Loss

By design, the softmax outputs determine the parameters for the discrete probability distributions of the Captcha generator parameters. The loss we minimize during training is the negative sum of the log of the softmax outputs

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{n=1}^{N}\left[-\sum_{t=1}^{T}\log\left(\left[\eta_{\theta,t}(y^{(n)})\right]_{x_t^{(n)}}\right)\right] , \tag{3}$$

where we use the notation $[z]_i$ to denote the ith element of z. This is a standard loss used in training neural networks for classification. The connection with Bayesian modeling in which we interpret softmax outputs as probabilities of discrete random variables in a joint importance sampling proposal distribution is explored in more detail in Section IV-B.

III. EXPERIMENTS

We wrote synthetic data generative models for seven different Captcha styles, covering the types frequently found in the Captcha breaking literature [18, 17, 20, 19]. For each of these, we trained a neural architecture consisting of (1) a CNN with six convolutions (3×3, with successively 64, 64, 64, 128, 128, 128 filters), max-pooling (2×2, step size 2) after the second, fifth, and sixth convolutions, and two final fully-connected layers of 1024 units; (2) a stack of two LSTMs of 512 hidden units each; and (3) fully-connected layers of appropriate dimension mapping the LSTM output to the corresponding softmax dimension of each latent variable. ReLU activations were used after the convolutions and the fully-connected layers overall.

We empirically verified that supplying the image embedding CNN(y) to the LSTM at every time step makes the training progress faster in our setup where we train the CNN from scratch together with the rest of the components, compared with the alternative of using CNN(y) only once and pretraining CNN weights on an image recognition database as in Vinyals et al. [23] and Karpathy and Fei-Fei [24].

The networks were implemented in Torch [27] and trained with Adam [28] optimization, with initial learning rate $\alpha = 0.0001$, hyperparameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, using minibatches of size 128. The generative models were implemented in the Anglican probabilistic programming language [29]. The two are coupled in our inference compilation [12] framework.[1]

A. Initial results

As can be seen in Table I, this architecture, and our method for training it using synthetic data, outperforms nearly all state-of-the-art Captcha breakers in terms of both accuracy and recognition times with the exception of Goodfellow et al. [7], which used data drawn from the true reCaptcha generator. The row labeled “our method” shows breaking results and speeds for our neural network trained using synthetic data to decode unlabeled Captchas from the same Captcha generator. The Goodfellow et al. [7] and Stark et al. [9] rows show the most directly comparable results, namely, using deep neural networks to break unlabeled Captchas training on synthetic data. The additional rows show breaking results for more traditional segment-and-classify computer vision image processing pipelines. These, in contrast to the others, do not have access to the true Captcha generator but instead report test results on real-world Captchas gathered in the wild. If robust, $> 90\%$ accuracies would seem to confirm that Captcha, from a computer security perspective [30, 31], is indeed broken.

While the capabilities of deep neural networks are impressive, it should be noted that these kinds of results, on occasion, can be somewhat misleading [15]. In particular, one should note the assumption that, up to this point in this paper and in the referenced results from the deep learning literature, the training procedure of the Captcha-breaking network has access

[1] https://probprog.github.io/inference-compilation/

to data from the true generative process. Indeed, samples from the true generative process are superior even to hand-labeled training instances gathered in the wild. Any simulated data, required when we do not have access to the true generative model, must come from an approximation to the true generative process, a model per se. Whether or not networks trained using such approximate data are robust in the sense of working well on real data in the wild becomes the real question. To put it another way, is Captcha really broken if we do not have access to the true generative model—or a legion of human labelers and a pile of cash?

B. Robustness of results

So, what happens to these state-of-the-art models if the test data is subtly different to the generated synthetic data? Or, what happens if you attempt to transfer learning from one Captcha style to another? Our exploration of these questions forms the inspiration and basis for the rest of the paper.

To start, we tried to use our trained models on real Captchas from Wikipedia and Facebook, which we identified as two major web services that still make use of textual Captchas,[2] collecting and hand-labeling test sets of 500 images each. We found that the trained Wikipedia and Facebook models achieving $> 90\%$ recognition with synthetic data yielded practically zero breaking rates with real data. We then tried using a model trained on one Captcha style to break another style and found that it nearly always failed as well. We found that this was only partially caused by the non-overlapping latent variable domains (e.g., the distinct character ranges) for renderers of different styles. For instance, one might expect the reCaptcha breaker to work on the visually similar Yahoo Captchas, but we found that this was not the case.

To investigate, we performed experiments where we constructed test Captchas that the trained networks cannot recognize despite being perceptually indistinguishable from Captchas from the original generative model. We found that we could more-or-less arbitrarily degrade test performance by shifting the test data in either of two ways away from the original synthetic data (Figure 2, left). In the first (Figure 2, middle), we corrupted the image by subtle additive noise which shifts each Captcha a small, imperceptible Euclidean distance from its original position. This causes our Captcha breaking networks to exhibit the kind of brittleness well known to be a problem for deep neural network classifiers [15]. In the second (Figure 2, right), by changing the generative model of the test data relative to the training data, even in ways that are arguably below human ability to perceive, we were also able to cause test performance to degrade. This is the kind of model misspecification that has been discussed in the Bayesian inference literature [16].

Inspired by the success of Jaderberg et al. [4], we attacked these problems by improving our synthetic training data generation. In particular, we developed a substantially more flexible generative model using the elastic displacement fields introduced by Simard et al. [2], effectively forcing the neural network to generalize over a greater variation than that exhibited by ground-truth labeled test data from the wild. These improved generative models have been observed to be robust to the subtle modifications that we report in Figure 2. The results we obtained are encouraging, achieving 81% and 42% recognition rates on real Wikipedia and Facebook Captchas respectively. In both cases our robust results, arrived at by improving the quality of the synthetic data generator, have performance comparable (in the case of Wikipedia, superior) to traditional vision pipelines, and are significantly higher than the 1% recognition threshold suggested to deem a deployed Captcha system broken [30].

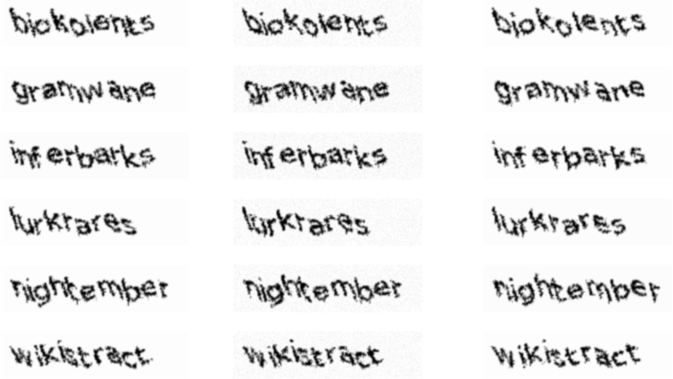


Fig. 2. Synthetic data from the Wikipedia generative model (left) are recognized correctly whereas even perceptually subtle changes such as adding per-pixel white noise with $\sigma = 5$ (middle) and $\epsilon_{kerning}$ modified by just one pixel (right) result in severely degraded recognition rates. The overall recognition rates for the test groups from which these samples are taken are 93.6% (left), 24.0% (middle) and 65.2% (right). Note that the middle and right columns do get recognized correctly with the robust Wikipedia model.

IV. Discussion and connections to model-based Bayesian reasoning

In order to explore some of the factors that cause the brittleness of the neural network performance that we have just reported, we draw a connection between Bayesian model mismatch and out-of-sample generalization failure of neural network and other regressors when tested on data that is different to that used for training.

As a prerequisite to this, we review importance sampling [32], the approximate probabilistic inference algorithm that most naturally corresponds to the kind of inference our trained neural networks allow us to do. Given a joint distribution $p(x, y)$ and a user-specified proposal distribution $q(x|y)$, importance sampling allows us to approximate the posterior distribution $p(x|y)$ and expectations of arbitrary functions f

[2] Facebook Captchas appear as a measure for preventing flood-posting and when links to particular Facebook pages are followed. Wikipedia Captchas appear on the account creation page. We note that textual reCaptchas, as of version 2.0, have been replaced with tasks such as image recognition [31], making them unlikely to encounter and collect.

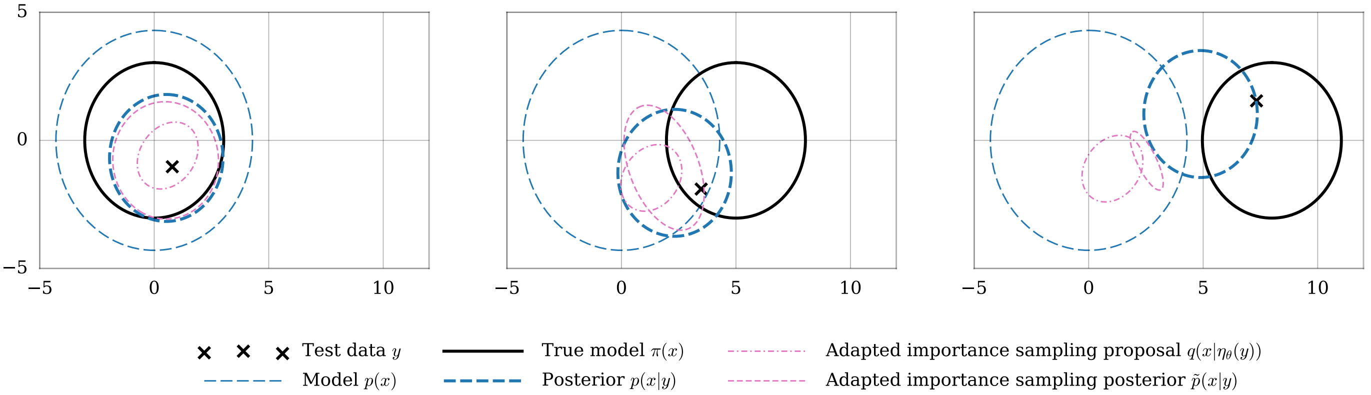


Fig. 3. Illustration of model mismatch. Left: The model encompasses the true data distribution; Middle: the model partially matches the true data distribution; Right: the model is completely mismatched to the true data distribution.

under it

$$p(x|y) \approx \sum_{m=1}^{M} W_m \delta(x - x^{(m)}) \tag{4}$$

$$\mathbb{E}_{p(x|y)}[f] \approx \sum_{m=1}^{M} W_m f(x^{(m)}) \ . \tag{5}$$

This is done by generating M weighted samples $\{(w_m, x^{(m)})\}_{m=1}^{M}$

$$x^{(m)} \sim q(x|y) \qquad m = 1, \dots, M \tag{6}$$

$$w_m = p(x^{(m)}, y)/q(x^{(m)}|y) \qquad m = 1, \dots, M \tag{7}$$

$$W_m = w_m / \sum_j w_j \qquad m = 1, \dots, M \ . \tag{8}$$

Note that importance sampling is generally inefficient unless the proposal distribution is well-matched to the target distribution in the sense that it "overlaps" the target, and is extremely efficient if it matches exactly.

A. Bayesian model misspecification

We illustrate the effects of mismatch between synthetic and real data in terms of Bayesian model misspecification using a simpler experiment (Figure 3), highlighting conceptually what we believe to be happening. Let $\pi(x, y)$ be the true data generating distribution and $p(x, y)$ a model, where

$$\pi(x) = \mathcal{N}(x|\mu_\pi, \Sigma_\pi) \tag{9}$$

$$\pi(y|x) = \mathcal{N}(y|x, \Sigma) \tag{10}$$

$$p(x) = \mathcal{N}(x|\mu_p, \Sigma_p) \tag{11}$$

$$p(y|x) = \mathcal{N}(y|x, \Sigma) \ . \tag{12}$$

We will use the mismatch between the distributions $\pi(x)$ and $p(x)$ as an illustrative proxy to the mismatch of the joint distributions $\pi(x, y)$ and $p(x, y)$.

The marginal $p(x)$ of the model distribution $p(x, y)$ is shown in Figure 3 as a thin blue dashed ellipse which covers 99% of its probability mass. We draw a data point y from this model by first drawing x from $p(x)$ and then drawing y from $\pi(y|x)$ where $\Sigma_p = 2I$, $\mu_p = [0, 0]^\mathsf{T}$ and $\Sigma = I$.

The marginal $\pi(x)$ of the true data generating distribution $\pi(x, y)$ is shown in Figure 3 as a black solid ellipse. A typical data point y is drawn by first drawing x from $\pi(x)$ and then drawing y from $\pi(y|x)$ where $\Sigma_\pi = I$ and μ_π is $[0, 0]^\mathsf{T}$, $[5, 0]^\mathsf{T}$ and $[8, 0]^\mathsf{T}$ from left to right.

Such a model has a posterior

$$p(x|y) = \mathcal{N}(x|\mu_{\text{post}}, \Sigma_{\text{post}}) \tag{13}$$

$$\Sigma_{\text{post}} = (\Sigma_p^{-1} + \Sigma^{-1})^{-1} \tag{14}$$

$$\mu_{\text{post}} = \Sigma_{\text{post}}(\Sigma_p^{-1}\mu_p + \Sigma^{-1}y) \ , \tag{15}$$

which is shown in Figure 3 as a thick blue dashed ellipse.

Using a procedure similar to the one described in Section II, we generate training data $\{(x^{(n)}, y^{(n)})\}$ from the model $p(x, y)$ and use it to train a neural network mapping from y to importance sampling proposal parameters $(\mu_q, \Sigma_q) := \eta_\theta(y)$. The resulting proposals generated from such a proposal distribution $q(x|\eta_\theta(y)) := \mathcal{N}(x|\mu_q, \Sigma_q)$ are shown in Figure 3 as magenta dash-dotted ellipses. Remember that μ_q and Σ_q are functions of y computed by the trained neural network regressor.

If we then draw $M = 1000$ samples from this proposal distribution by repeatedly running the trained neural network forward and weight the resulting samples according to the importance sampling scheme in the beginning of Section IV, we arrive at approximations to the model-based posterior mean and covariance:

$$\tilde{\mu}_M \approx \mathbb{E}_{p(x|y)}[x] \tag{16}$$

$$\tilde{\Sigma}_M \approx \mathbb{E}_{p(x|y)}\left[\left(x - \mathbb{E}_{p(x|y)}[x]\right)\left(x - \mathbb{E}_{p(x|y)}[x]\right)^\mathsf{T}\right] . \tag{17}$$

The distribution $\tilde{p}(x|y) := \mathcal{N}(x|\tilde{\mu}_M, \tilde{\Sigma}_M)$ is shown in Figure 3 as a magenta dashed ellipse.

Now consider the three scenarios in Figure 3, in which the difference between the true data generating distribution, illustrated by its marginal $\pi(x)$, and the model $p(x)$ is progressively increased from left to right. As the true data generating distribution $\pi(x)$ moves further away from our model $p(x)$ we see that we get, for a fixed computational budget of $M = 1000$ samples, progressively worse estimates $\tilde{p}(x|y)$ of $p(x|y)$ (Figure 3, middle and right). What is happening here is that the neural network, at training time, learns to invert the model $p(x, y)$ from samples drawn from it. In Figure 3 (left), when the model overlaps the true data generative process, the neural network sees examples of x and y pairs that are representative of the true data generating mechanism and then, given sufficient capacity in terms of neural architecture and training time (remembering that we have access in this setting to infinite training data), can almost certainly learn a mapping that solves the task of predicting x given y. If the model is slightly misspecified then the number of training examples in the domain of the true model might be small and as such we might not expect good generalization performance. When there is high model misspecification (Figure 3, right) the neural network will simply never see training examples that look like the true data, and, as such, will produce mostly spurious regression results leading to unhelpful proposal distributions.

This experiment graphically illustrates the kinds of problems that can arise from model misspecification. What it indicates is that if we are going to use synthetic data to train a neural network regressor we should ensure that our synthetic data generator is ideally as close as possible to the true data generation process and that mismatch from the true data in terms of broadness (e.g., the Gaussian example in Figure 3 (left), in which μ_π and μ_p match but Σ_π and Σ_p do not) is more tolerable and in fact preferable to a perceptually indistinguishably miscalibrated model (e.g. the phenomenon illustrated in Figure 2 and described in Section III-B). We conjecture that the latter is what caused the brittleness we discovered in our trained neural networks and illustrate in Figure 2.

This intuition guided our decision to broaden our synthetic data generator by adding the displacement fields of Simard et al. [2] in Section III-B, leading to significant improvements to robustness evidenced by the improved real-data results we obtained. This, we believe, accounts for the fact that our Captcha generator is not likely to capture all details of the true generative model such as subtle font differences.

B. Inference

A corollary to the Bayesian inference interpretation of training a neural network on synthetic data is that the resulting neural network can be used for approximate inference in the probabilistic model $p(x, y)$ corresponding to the synthetic training data generator.

Let the importance sampling proposal distribution be factorized as $q(x|y) = \prod_{t=1}^{T} q_t(x_t|x_{1:t-1}, y)$. If we consider the individual time-dependent softmax layers of the Captcha-solving neural network to be probabilities of a proposal distribution $q_t(x_t|x_{1:t-1}, y)$, we can adopt an alternative way of writing our loss in (3) as

$$\begin{aligned}
\mathcal{L}(\theta) &= \frac{1}{N}\sum_{n=1}^{N}\left[-\sum_{t=1}^{T}\log\left(\left[\eta_{\theta,t}(y^{(n)})\right]_{x_t^{(n)}}\right)\right] \\
&= \frac{1}{N}\sum_{n=1}^{N}\left[-\sum_{t=1}^{T}\log q_t(x_t^{(n)}|x_{1:t-1}^{(n)}, y)\right] \\
&= \frac{1}{N}\sum_{n=1}^{N}\left[-\log\left(\prod_{t=1}^{T} q_t(x_t^{(n)}|x_{1:t-1}^{(n)}, y)\right)\right] \\
&= \frac{1}{N}\sum_{n=1}^{N}\left[-\log q(x^{(n)}|\eta_\theta(y^{(n)}))\right] \ . \qquad (18)
\end{aligned}$$

The loss in (18) can be viewed as a Monte Carlo approximation of an expectation over a function under the joint distribution $p(x, y)$ of the synthetic data, which, following Paige and Wood [13], can be shown to be the Kullback-Leibler divergence between the proposal and the posterior averaged over all possible datasets

$$\begin{aligned}
&\mathbb{E}_{p(x,y)}[-\log q(x|\eta_\theta(y))] \\
&= \int_{\mathcal{Y}}\int_{\mathcal{X}} p(x, y)(-\log q(x|\eta_\theta(y)))\,\mathrm{d}x\mathrm{d}y \\
&= \int_{\mathcal{Y}} p(y)\int_{\mathcal{X}} p(x|y)\log\frac{p(x|y)}{q(x|\eta_\theta(y)))}\,\mathrm{d}x\mathrm{d}y + \text{const.} \\
&= \mathbb{E}_{p(y)}[D_{\mathrm{KL}}\left(p(x|y)\ ||\ q(x|\eta_\theta(y))\right)] + \text{const.} \qquad (19)
\end{aligned}$$

Hence, minimizing (18) is also known as importance sampling proposal adaptation.

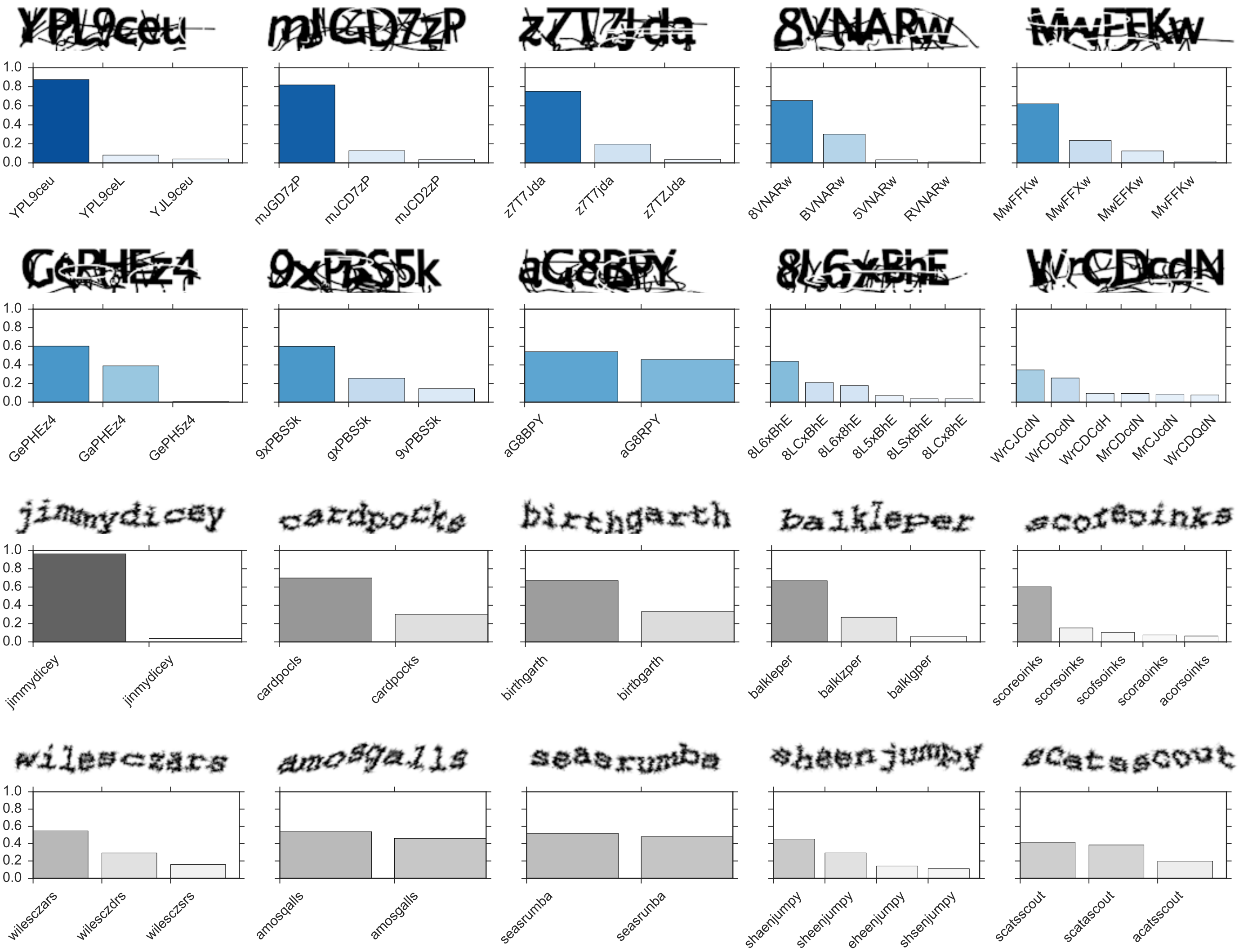
Running a neural network trained using synthetic data and this common loss on an input y actually produces efficient proposal distribution parameters $\eta_\theta(y)$. By running the neural network M times given the same input and subsequently weighting the sampled x values according to (7), we obtain an approximate posterior distribution (Figure 4). We note that, in the case of Captchas, we must use a likelihood based on approximate Bayesian computation (ABC) [33] instead of the intractable $p(y|x)$ in order to calculate the weight in (7).

Accounting for uncertainty is a principal benefit of model-based inference and is particularly useful when there is actual ambiguity in y as in Figure 4.

V. Conclusion

What is remarkable about the natural scene text recognition results of Jaderberg et al. [4, 5] is that they show generalization from synthetic data, to the degree that one could argue that their result is actually a generative modeling triumph. Our results showing improved robustness of Wikipedia- and Facebook-style Captcha-breaking stem likewise from focusing on the synthetic data generative model. In addition to being usefully prescriptive, our point that training neural networks using synthetic data is equivalent to performing proposal adaptation for importance sampling inference in the

Fig. 4. Posteriors of real Facebook and Wikipedia Captchas. Conditioning on each Captcha, we show an approximate posterior produced by a set of weighted importance sampling particles $\{(w_m, x^{(m)})\}_{m=1}^{M=100}$.



synthetic data generative model sets an empirical cornerstone for future theory that quantifies and bounds the impact of model mismatch on neural network and approximate inference performance.

Acknowledgments

Tuan Anh Le is supported by EPSRC DTA and Google (project code DF6700) studentships. Atılım Güneş Baydin and Frank Wood are supported under DARPA PPAML through the U.S. AFRL under Cooperative Agreement FA8750-14-2-0006, Sub Award number 61160290-111668. Robert Zinkov is supported under DARPA grant FA8750-14-2-0007.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," in *Proceedings of the Seventh International Conference on Document Analysis and Recognition - Volume 2*, ser. ICDAR '03. Washington, DC: IEEE Computer Society, 2003, pp. 958–962.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[4] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, "Synthetic data and artificial neural networks for natural scene text recognition," *arXiv preprint arXiv:1406.2227*, 2014.

[5] ——, "Reading text in the wild with convolutional neural networks," *International Journal of Computer Vision*, vol. 116, no. 1, pp. 1–20, 2016.

[6] A. Gupta, A. Vedaldi, and A. Zisserman, "Synthetic Data for Text Localisation in Natural Images," in *Proceedings*

of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2315–2324.

[7] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet, "Multi-digit number recognition from street view imagery using deep convolutional neural networks," *arXiv preprint arXiv:1312.6082*, 2013.

[8] L. von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum, "reCAPTCHA: Human-based character recognition via web security measures," *Science*, vol. 321, no. 5895, pp. 1465–1468, 2008.

[9] F. Stark, C. Hazırbaş, R. Triebel, and D. Cremers, "Captcha recognition with active deep learning," in *GCPR Workshop on New Challenges in Neural Computation*, Aachen, Germany, 2015.

[10] Z. Wang, J. Yang, H. Jin, E. Shechtman, A. Agarwala, J. Brandt, and T. S. Huang, "Deepfont: Identify your font from an image," in *Proceedings of the 23rd ACM international conference on Multimedia*. ACM, 2015, pp. 451–459.

[11] S. J. Gershman and N. D. Goodman, "Amortized inference in probabilistic reasoning," in *Proceedings of the 36th Annual Conference of the Cognitive Science Society*, 2014.

[12] T. A. Le, A. G. Baydin, and F. Wood, "Inference compilation and universal probabilistic programming," in *20th International Conference on Artificial Intelligence and Statistics, April 20–22, 2017, Fort Lauderdale, US*, 2017.

[13] B. Paige and F. Wood, "Inference networks for sequential Monte Carlo in graphical models," in *Proceedings of the 33rd International Conference on Machine Learning*, ser. JMLR, vol. 48, 2016.

[14] G. Papamakarios and I. Murray, "Fast ϵ-free inference of simulation models with Bayesian conditional density estimation," *arXiv preprint arXiv:1605.06376*, 2016.

[15] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[16] A. Gelman and C. R. Shalizi, "Philosophy and the practice of Bayesian statistics," *British Journal of Mathematical and Statistical Psychology*, vol. 66, no. 1, pp. 8–38, 2013.

[17] E. Bursztein, J. Aigrain, A. Moscicki, and J. C. Mitchell, "The end is nigh: generic solving of text-based CAPTCHAs," in *8th USENIX Workshop on Offensive Technologies (WOOT 14)*, 2014.

[18] O. Starostenko, C. Cruz-Perez, F. Uceda-Ponga, and V. Alarcon-Aquino, "Breaking text-based CAPTCHAs with variable word and character orientation," *Pattern Recognition*, vol. 48, no. 4, pp. 1101–1112, 2015.

[19] H. Gao, W. Wang, Y. Fan, J. Qi, and X. Liu, "The robustness of "connecting characters together" CAPTCHAs," *Journal of Information Science and Engineering*, vol. 30, no. 2, pp. 347–369, 2014.

[20] H. Gao, W. Wang, J. Qi, X. Wang, X. Liu, and J. Yan, "The robustness of hollow CAPTCHAs," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*. ACM, 2013, pp. 1075–1086.

[21] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford, "CAPTCHA: Using hard AI problems for security," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2003, pp. 294–311.

[22] V. Mansinghka, T. D. Kulkarni, Y. N. Perov, and J. Tenenbaum, "Approximate Bayesian image interpretation using generative probabilistic graphics programs," in *Advances in Neural Information Processing Systems*, 2013, pp. 1520–1528.

[23] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3156–3164.

[24] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3128–3137.

[25] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[26] S. Reed and N. de Freitas, "Neural programmer-interpreters," in *International Conference on Learning Representations (ICLR)*, 2016.

[27] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," in *BigLearn, NIPS Workshop*, 2011.

[28] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR), San Diego, US*, 2015.

[29] F. Wood, J. W. van de Meent, and V. Mansinghka, "A new approach to probabilistic programming inference," in *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, 2014, pp. 1024–1032.

[30] E. Bursztein, M. Martin, and J. Mitchell, "Text-based CAPTCHA strengths and weaknesses," in *Proceedings of the 18th ACM Conference on Computer and communications security*. ACM, 2011, pp. 125–138.

[31] S. Sivakorn, I. Polakis, and A. D. Keromytis, "I am robot: (deep) learning to break semantic image captchas," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2016, pp. 388–403.

[32] A. Doucet and A. M. Johansen, "A tutorial on particle filtering and smoothing: Fifteen years later," *Handbook of Nonlinear Filtering*, vol. 12, no. 656-704, p. 3, 2009.

[33] R. D. Wilkinson, "Approximate Bayesian computation (ABC) gives exact results under the assumption of model error," *Statistical Applications in Genetics and Molecular Biology*, vol. 12, no. 2, pp. 129–141, 2013.