Papers Reviewed:

CS-GAN

MaskGAN

Realistic Evaluation of Deep Semi-Supervised Learning

CS & Mask GANs

The reasoning behind the use of GANs is motivated by the desire to improve *generation* ability of novel samples from the learned distribution. Since most deep text-generation happens autoregressively, to sample we simply condition on the previously generated tokens, instead of using teacher forcing which conditions on the actual token. By taking unconditional samples, we are basically pulling from the full language model.

We can generate via the standard RNN maximum likelihood trained net ala charRNN, but that has the problem that it suffers in off-manifold generation, because at sample time when the RNN is conditioned on a sequence that it itself generated, the sequence is likely to be not something the RNN has encountered before due to compounding sampling errors. Because the RNN is trained to maximize the quality of the on-manifold learned distribution, the off-manifold distribution is probably bad and generates unrealistic samples especially those of longer length.

So the core problem these models are trying to solve is to improve the quality of the off-manifold learned distribution.

There are a few ways to do this: Scheduled Sampling, where the teacher-fed versus generated-fed ratio is varied over time – or usage of adversarial training.

There are a few methods which use GANs for text generation, but they have to grapple with the core problem: the generator outputs a discrete set of tokens and so gradients can't be passed directly from the discriminator loss to the generator. There are a few ways of getting around this: recasting the problem in a continuous output space such as with Professor-Forcing, where there is a discriminator running on the hidden states of the generator, trying to tell if the model was input a true token or a generated one (which forces the dynamics of off-manifold inputs to resemble those of on-manifold inputs) and is of course differentiable.

The approach taken by both CS-GAN and MaskGAN is to add a reinforcement learning component to the GAN setup, wherein the previously generated tokens are the primary component of the state, the set of next tokens which could be generated is the action set, the output of the discriminator is a component of the reward, and the weights of the generator serve to parameterize the policy.

Improving the weights is therefore a policy-learning problem, and they are updated by policy gradients which serve to move the network in the direction of greater expected total reward.

Although these two networks have the same high-level architecture, they differ in the specifics.

CS-GAN

CS-GAN or "Category-Sequence GAN" has two goals – generating realistic sentences using the GAN framework and RL to avoid the discrete nature of text, and using category information when generating synthetic data. To solve the first part, the authors rely heavily on SeqGAN, which introduced the GAN+RL combo for text data. The novelty of the paper primarily lies in the second half, and how it can be used to improve the classifier performance.

The specifics of their architecture are as follows:

The sentence is tokenized at the character-level, which serves to reduce the action space, reduce policy gradient varience and so makes the markov rollouts more informative.

In total, there are three networks: the generator RNN, the discriminator CNN, and the classifier CNN (implemented as a separate head off the discriminator). The generator is trained by a policy gradient update based on how well it fools the discriminator and if the desired classification is reached by the classifier. The discriminator and classifier are trained as normal (with cross-entropy loss).

The generator is a standard LSTM RNN, at each timestep conditioning on the previously generated tokens, and the prior information which consists of some latent noise, and the category designation for the sequence. The fact that the category information is an input to the generator allows the class of the generated sentence to be controlled.

The parameter updates for the generator come from the blended rewards of the discriminator and generator, but those rewards are only created after the entire sequence is generated, that is there are no temporary rewards. To get around this, Monte Carlo rollouts are used to estimate the Q state-action value function, which is a component of the policy gradient. In Monte Carlo rollout, at the current state, for each possible action we simulate a few trajectories according to current policy in future timesteps as if we had selected this action. Then when the trajectory is over, a reward is received. The average reward over the trajectories for a given action is then an estimate of Q(s, a), and the policy gradient points in the direction of making higher reward actions in the given state more likely.

The discriminator and classifiers are CNNs, similar to SeqGAN. It's your standard example – each character embedding is concatenated together, various filters are applied across all the dimensions, then max pooling over time is used.

The loss of the classifier is the logistic loss over both generated sentences and real ones, based on the probability it assigns to each sentence being the correct class.

The discriminator has the usual discriminator loss about how well it's able to tell apart fake and real sentences.

Obviously training GANs is somewhat difficult- they pretrain the generator using maximum likelihood initially, pre-train the classifier and discriminator based on a set of generated samples. During actual training, the first character is generated uniformly at random, then the network is run based on that. The discriminator is only trained if it has less than 90% accuracy.

The datasets used are Amazon reviews, Yelp revies, Stanford sentiment tree bank, NEWS dataset, CrowdFlower Emotion dataset. These are subsampled to evaluate model generalization over small datasets.

The amazon, news, and emotion datasets are naturally short, maxing out at 150 characters, while Yelp is longer – they extracted 4 subdata sets with maximum review length pegged to 1000, 800, 500, 200 character resp. 15,000 training sentences for all the Yelp dataset.s

They test the language model using negative log likelihood on the Amazon-5000 dataset – not surprisingly CS-GAN does better than it’s more limited companions.

Feature correlation of classifier neurons on real to fake sentences.

Tested with data having a ratio from 0.25 to 1 of generated sentences to real ones when training classifier, found that as beta improved, classifier accuracy increased due to less overfitting. General idea = why not use higher beta but downweight each synthetic sentence in loss calc?

As sentence length increases, performance degrades slightly, although not as much with full GAN – because gan always prefers to generate shorter sentences.

Suggestions: Alter the Beta, use the setup in VAE Controllable text generation enforced independence contstraints. Add global sentence-embedding context to increase length of generated text/enforce global consistency.

Papers to Read: SeqGAN, Dialogue-GAN, Controllable Text Generation

MaskGAN

A different problem set-up: Rather than generating sentences autoregressively from whole-cloth, the problem is actual sentences with stochastic masks applied to words.

Still the GAN setup so there is a cost function which directly ties sample quality.

Advantages is reduced mode dropping, and improved training stability because the probability of true/fake from the discriminator can be generated per token instead of at end.

Architecture set-up

Instead of using monte carlo rollouts to estimate the state-action value for policy gradient update, use an actor-critic model where the actor is again the generator as it infills a token, and the critic is a head off the discriminator trained on the actual observed discounted total reward from the discriminator.

They tried a number of different setups for the generator/discriminator, but ultimately went with a seq2seq model which takes in the masked sentence, generates a context, and then feeds that to the decoder model which autoregressively generates tokens.

The *unmasked* sentence is then fed into the encoder again, which generates the true context which is handed to the discriminator (also an RNN) along with the filled in token sequence, which outputs per masked timestep the probability that that token comes from the true sentence or the filled in one. It receives the true context to avoid the “director director” could be “assistant director” or “director expertly” failure mode.

Reward per timestep is log probability of being true. Critic estimates the discounted total return.

Training:

Policy gradient updates, maximizing expected total reward, use estimated critic value as baseline for total reward which reduces variance of gradients. That is, move parameters in direction of advantage, not just reward.

Task becomes harder with longer sentences and larger vocabularies – use dynamic training to help alleviate, by gradually increasing maximum sentence length upon convergence.

Also, to reduce variance of REINFORCE algorithm, at each time step use full info of distribution by computing reward (from discriminator, log prob of being true) for each possible token, which is then weighted by the probability of that token being generated when it is turned into a gradient. We then sample an actual next token to continue generation. This effectively increases the number of reward trajectories we are considering, and so reduces the variance of the gradient updates.

There are various pretraining, first as a language model via maximum likelihood, then pre-trained specifically on the in-filling task.

Evaluation, they found that perplexity of the generated sentences on the trained LM is not actually good, because the generator is exploring off-manifold. They also tried # of unique generated n-grams for small n, which captured mode dropping.

Sampling unconditionally is essentially equal to decoding a sentence which consisted entirely of blanks (does that require fixing a sequence length ahead of time?)

Trained on PTB and IMDB data sets at word-level.

Problem with mode-dropping at end of sentence with unconditional generation, due to incorrect generation of previous words.

They also test with human evaluation, which shows that MaskGAN outperforms MaskMLE, normal LM, Mask MLE, and SeqGAN and even performs only pretty badly against the real samples.

They found attention to be crucial for the in-filled words to be sufficiently conditioned on surrounding words, and they also preferred contiguous blanking.

Additions: Can easily make controllable by adding context info to post-encoder hidden state (or pre-encoder hidden state, or both!). Question of how to create classifier ala CS-GAN is a little harder, as it is unclear if the classifier reward can be given token-by-token effectively, or if it would require generating the entire sentence.

Potential Idea: Discriminator reward is possible because have two sentences, one real and one fake, which only differ on a handful of words – making it meaningful to ask on a per word basis whether or not the word came from the real sentence or fake. Classifier is more difficult, as one would assume that the differences in classes would emerge at the sentence level. One possibility is, if the class is controllable, using the same masked sentence and generating a sentence per class by changing the desired class input. The question of which word came from which class sentence is then more meaningful (potentially), and so could be used to create a per-word reward function from the classifier as well as the discriminator.

Further Reading: WGAN-GP which generates text in one-shot manner so no backpropagating through discrete nodes. Or, Zhange 2017 has RNN generator which matches high dimensional representations and so can backpropagate directly (no RL).

Realistic Evaluation of Deep Semi-Supervised Learning Algorithms

Basically goal of the paper is to examine flaws with ways that SSL algorithms are typically compared/evaluated and show more standardized ways of doing it to compare to “Real-World” usage applicability.

Findings: Power of SSL is often overstated – with equal hypertuning budget, using just labelled data often has a smaller gap from using both labelled and unlabeled. Large classifier with well-tuned regularization on just labelled data is often almost as good as SSL, so need to evaluate SSL algorithms on same underlying model.

Transfer learning from similar large labelled data set to desired small labelled dataset can beat out SSL in many cases, if similar enough.

If classes in unlabeled dataset are sufficiently different/underlying distribution is sufficiently different, can actually hurt performance on labelled dataset.

SSL algorithms have different levels of sensitivity to amounts of labelled/unlabeled datasets. Most interestingly, if validation datasets are realistically small, prevents reliable comparison of SSL models/methods/hyperparameters.

Recommended procedure for comparing SSL algorithms:

P1 Shared implementation of base model across all SSL algos.

P2 High-quality supervised baseline of base model on just labeled dataset, with well-tuned hyperparameters is crucial.

P3 Comparison to transfer learning where possible as it is powerful, widely-used, and rarely reported baseline to compare against.

P4 Consider class distribution match between labelled and unlabeled dataset, because can have major impact on performance.

P5 Vary amount of both labelled and unlabeled data, to compare 2 scenarios of cheap acquisition and expensive acquisition of unlabeled data.

P6 In reality, validation dataset isn’t going to be larger than training, but most SSL evaluation pretends it is because small labelled dataset is created by throwing away labels from big dataset. In reality, this is not how this method will be used.

The point of the unlabeled dataset is to provide hints as to the shape of the underlying manifold.

This paper only evaluates methods which add a term to the loss function, thought ther are other SSL algos these are simple and perform well. Two classes, consistency regulation which try to make the assigned class probabilities robust to small changes in data or parameters. And entropy minimization which tries to make predictions on unlabeled points more confident.

PI-Model: With stochastic classing function, pass the same point through multiple times and penalize deviations.

Mean teacher: Use exponential moving average of previous parameter settings to create target class prob for each point, then penalize deviations from that.

Virtual Adversarial training: Add small noise to point in direction most likely to affect output, penalize that change.

VAT + EntMIN, same thing, adds loss term to penalize low-confidence unlabeled examples.

Psuedo-labelling: Frequently used, for sufficiently confident predictions on unlabeled points, count those as basically being real labels for training purposes.