magenta / magenta

Code Issues 267 Pull requests 34 Actions Security Insights

master

magenta / magenta / models / sketch_rnn / README.md

proppy README: fix typo in Jupyter Notebook link (#908) ... History

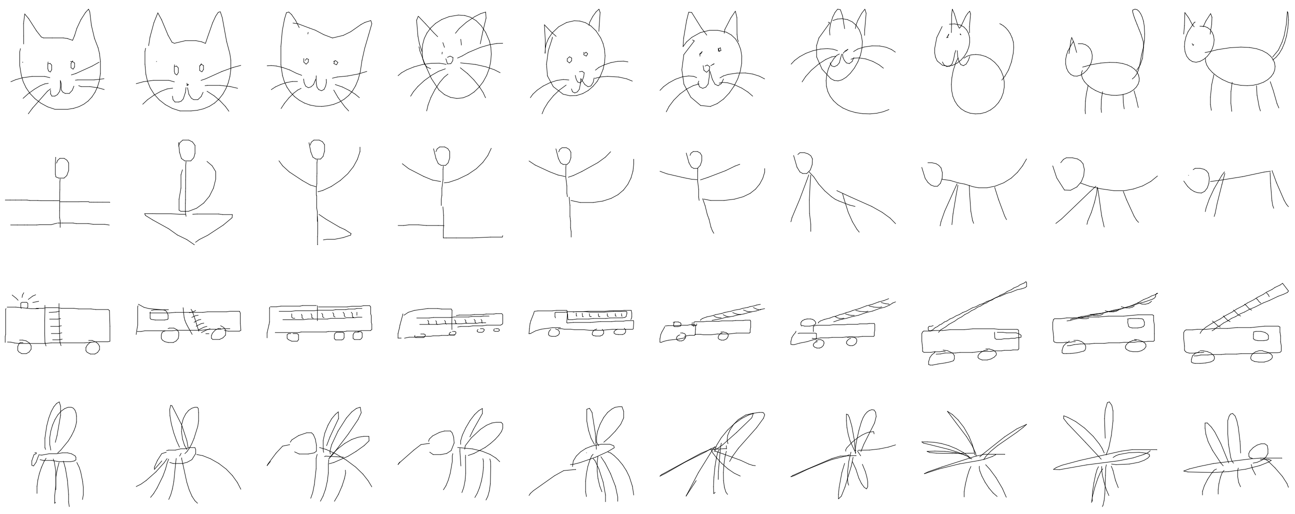
6 contributors

Raw Blame

160 lines (110 sloc) 14 KB

Sketch-RNN: A Generative Model for Vector Drawings

Before jumping in on any code examples, please first set up your Magenta environment.



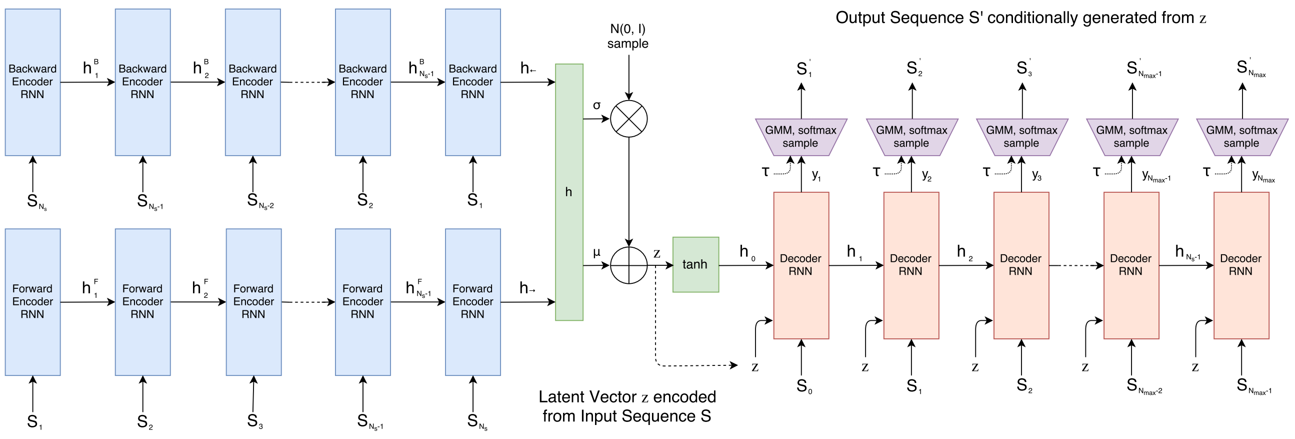
Examples of vector images produced by this generative model.

This repo contains the TensorFlow code for `sketch-rnn`, the recurrent neural network model described in Teaching Machines to Draw and A Neural Representation of Sketch Drawings. We've also provided a Jupyter notebook Sketch_RNN.ipynb in our Magenta Demos repository which demonstrates many of the examples discussed here.

Overview of Model

`sketch-rnn` is a Sequence-to-Sequence Variational Autoencoder. The encoder RNN is a bi-directional RNN, and the decoder is an autoregressive mixture-density RNN. You can specify the type of RNN cell to use, and the size of the RNN using the settings `enc_model`, `dec_model`, `enc_size`, `dec_size`.

The encoder will sample a latent code *z*, a vector of floats with a dimension of `z_size`. Like in the VAE, we can enforce a Gaussian IID distribution to *z*, and the strength of the KL Divergence loss term is controlled using `kl_weight`. There will be a tradeoff between KL Divergence Loss and the Reconstruction Loss. We also allow some room for the latent code to store information, and not be pure Gaussian IID. Once the KL Loss term gets below `kl_tolerance`, we will stop optimizing for this term.


Output Sequence S' conditionally generated from z
N(0, I) sample
Backward Encoder RNN
Forward Encoder RNN
h
σ
μ
z
tanh
GMM, softmax sample
Decoder RNN
Latent Vector z encoded from Input Sequence S

For small to medium sized datasets, dropout and data augmentation is very useful technique to avoid overfitting. We have provided options for input dropout, output dropout, and recurrent dropout without memory loss. In practice, we only use recurrent dropout, and usually set it to between 65% to 90% depending on the dataset. Layer Normalization and Recurrent Dropout can be used together, forming a powerful combination for training recurrent neural nets on a small dataset.

There are two data augmentation techniques provided. The first one is a `random_scale_factor` to randomly scale the size of training images. The second augmentation technique (not used in the `sketch-rnn` paper) is dropping out random points in a line stroke. Given a line segment with more than 2 points, we can randomly drop points inside the line segments with a small probability of `augment_stroke_prob`, and still maintain a similar-looking vector image. This type of data augmentation is very powerful when used on small datasets, and is unique to vector drawings, since it is difficult to dropout random characters or notes in text or midi data, and also not possible to dropout random pixels without causing large visual differences in pixel image data. We usually set both data augmentation parameters to 10% to 20%. If there is virtually no difference for a human audience when they compare an augmented example compared to a normal example, we apply both data augmentation techniques regardless of the size of the training dataset.

Using dropout and data augmentation effectively will avoid overfitting to a small training set.

Training a Model

To train the model you first need a dataset containing train/validation/test examples. We have provided links to the `aaron_sheep` dataset and the model will use this lightweight dataset by default.

Example Usage:

```
sketch_rnn_train --log_root=checkpoint_path --data_dir=dataset_path --hparams="data_
```

We recommend you create subdirectories inside `models` and `datasets` to hold your own data and checkpoints. The TensorBoard logs will be stored inside `checkpoint_path` for viewing training curves for the various losses on train/validation/test datasets.

Here is a list of full options for the model, along with the default settings:

```
data_set=['aaron_sheep.npz'],  # Our dataset. Can be list of multiple .npz sets.
num_steps=10000000,            # Total number of training set. Keep large.
save_every=500,                # Number of batches per checkpoint creation.
dec_rnn_size=512,              # Size of decoder.
dec_model='lstm',              # Decoder: lstm, layer_norm or hyper.
enc_rnn_size=256,              # Size of encoder.
```

```
  enc_model='lstm',             # Encoder: lstm, layer_norm or hyper.
  z_size=128,                   # Size of latent vector z. Recommend 32, 64 or 128.
  kl_weight=0.5,                # KL weight of loss equation. Recommend 0.5 or 1.0.
  kl_weight_start=0.01,         # KL start weight when annealing.
  kl_tolerance=0.2,             # Level of KL loss at which to stop optimizing for KL
  batch_size=100,               # Minibatch size. Recommend leaving at 100.
  grad_clip=1.0,                # Gradient clipping. Recommend leaving at 1.0.
  num_mixture=20,               # Number of mixtures in Gaussian mixture model.
  learning_rate=0.001,          # Learning rate.
  decay_rate=0.9999,            # Learning rate decay per minibatch.
  kl_decay_rate=0.99995,        # KL annealing decay rate per minibatch.
  min_learning_rate=0.00001,    # Minimum learning rate.
  use_recurrent_dropout=True,   # Recurrent Dropout without Memory Loss. Recomended.
  recurrent_dropout_prob=0.90,  # Probability of recurrent dropout keep.
  use_input_dropout=False,      # Input dropout. Recommend leaving False.
  input_dropout_prob=0.90,      # Probability of input dropout keep.
  use_output_dropout=False,     # Output droput. Recommend leaving False.
  output_dropout_prob=0.90,     # Probability of output dropout keep.
  random_scale_factor=0.15,     # Random scaling data augmention proportion.
  augment_stroke_prob=0.10,     # Point dropping augmentation proportion.
  conditional=True,             # If False, use decoder-only model.
```

Here are some options you may want to use to train the model on a very large dataset spanning three `.npz` files, and use HyperLSTM as the RNN cells. For small datasets of less than 10K training examples, LSTM with Layer Normalization (`layer_norm` for both `enc_model` and `dec_model`) works best.

```
sketch_rnn_train --log_root=models/big_model --data_dir=datasets/big_dataset --hpara
```

We have tested this model on TensorFlow 1.0 and 1.1 for Python 2.7.

Datasets

Due to size limitations, this repo does not contain any datasets.

We have prepared many datasets that work out of the box with Sketch-RNN. The Google QuickDraw Dataset is a collection of 50M vector sketches across 345 categories. In the repo of quickdraw-dataset, there is a section called *Sketch-RNN QuickDraw Dataset* that describes the pre-processed datafiles that can be used with this project. Each category class is stored in its own file, such as `cat.npz` , and contains training/validation/test set sizes of 70000/2500/2500 examples.

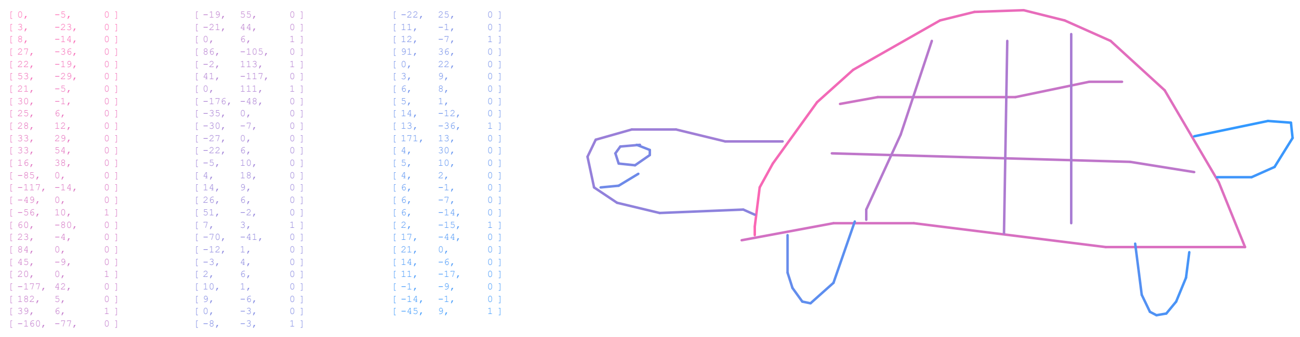
You download the `.npz` datasets from google cloud for local use. We recommend you create a sub directory called `datasets/quickdraw`, and save these `.npz` files in this sub directory.

In addition to the QuickDraw dataset, we have also tested this model on smaller datasets. In the sketch-rnn-datasets repo, there are 3 other datasets: Aaron Koblin Sheep Market, Kanji, and Omniglot. We recommend you create a sub directory for each of these dataset, such as `datasets/aaron_sheep`, if you wish to use them locally. As mentioned before, recurrent dropout and data augmentation should be used when training models on small datasets to avoid overfitting.

Creating Your Own Dataset

Please create your own interesting datasets and train this algorithm on them! Getting your hands dirty and creating new datasets is part of the fun. Why settle on existing pre-packaged datasets when you are potentially sitting on an interesting dataset of vector line drawings? In our experiments, a dataset size consisting of a few thousand examples was sufficient to produce some meaningful results. Here, we describe the format of the dataset files the model expects to see.

Each example in the dataset is stored as list of coordinate offsets: Δx, Δy, and a binary value representing whether the pen is lifted away from the paper. This format, we refer to as *stroke-3*, is described in this paper. Note that the data format described in the paper has 5 elements (*stroke-5* format), and this conversion is done automatically inside the `DataLoader`. Below is an example sketch of a turtle using this format:



```
[ 0,    -5,     0 ]      [ -19,  55,     0 ]      [ -22,  25,     0 ]
[ 3,    -23,    0 ]      [ -21,  44,     0 ]      [ 11,   -1,     0 ]
[ 8,    -14,    0 ]      [ 0,    6,      1 ]      [ 12,   -7,     1 ]
[ 27,   -36,    0 ]      [ 86,   -105,   0 ]      [ 91,   36,     0 ]
[ 22,   -19,    0 ]      [ -2,   113,    1 ]      [ 0,    22,     0 ]
[ 53,   -29,    0 ]      [ 41,   -117,   0 ]      [ 3,    9,      0 ]
[ 21,   -5,     0 ]      [ 0,    111,    1 ]      [ 6,    8,      0 ]
[ 30,   -1,     0 ]      [ -176, -48,    0 ]      [ 5,    1,      0 ]
[ 25,   6,      0 ]      [ -35,  0,      0 ]      [ 14,   -12,    0 ]
[ 28,   12,     0 ]      [ -30,  -7,     0 ]      [ 13,   -36,    1 ]
[ 33,   29,     0 ]      [ -27,  0,      0 ]      [ 171,  13,     0 ]
[ 33,   54,     0 ]      [ -22,  6,      0 ]      [ 4,    30,     0 ]
[ 16,   38,     0 ]      [ -5,   10,     0 ]      [ 5,    10,     0 ]
[ -85,  0,      0 ]      [ 4,    18,     0 ]      [ 4,    2,      0 ]
[ -117, -14,    0 ]      [ 14,   9,      0 ]      [ 6,    -1,     0 ]
[ -49,  0,      0 ]      [ 26,   6,      0 ]      [ 6,    -7,     0 ]
[ -56,  10,     1 ]      [ 51,   -2,     0 ]      [ 6,    -14,    0 ]
[ 60,   -80,    0 ]      [ 7,    3,      1 ]      [ 2,    -15,    1 ]
[ 23,   -4,     0 ]      [ -70,  -41,    0 ]      [ 17,   -44,    0 ]
[ 84,   0,      0 ]      [ -12,  1,      0 ]      [ 21,   0,      0 ]
[ 45,   -9,     0 ]      [ -3,   4,      0 ]      [ 14,   -6,     0 ]
[ 20,   0,      1 ]      [ 2,    6,      0 ]      [ 11,   -17,    0 ]
[ -177, 42,     0 ]      [ 10,   1,      0 ]      [ -1,   -9,     0 ]
[ 182,  5,      0 ]      [ 9,    -6,     0 ]      [ -14,  -1,     0 ]
[ 39,   6,      1 ]      [ 0,    -3,     0 ]      [ -45,  9,      1 ]
[ -160, -77,    0 ]      [ -8,   -3,     1 ]
```

Figure: A sample sketch, as a sequence of (Δx, Δy, binary pen state) points and in rendered form. In the rendered sketch, the line color corresponds to the sequential stroke ordering.

In our datasets, each example in the list of examples is represented as a `np.array` with `np.int16` datatypes. You can store them as `np.int8` if you can get away with it to save storage space. If your data must be in floating-point format, `np.float16` also works. `np.float32` can be a waste of storage space. In our data, the Δx and Δy offsets are represented in pixel locations, which are larger than the range of numbers a neural network model likes to see, so there is a normalization scaling process built into the model. When we load the training data, the model will automatically convert to `np.float` and normalize accordingly before training.

If you want to create your own dataset, you must create three lists of examples for training/validation/test sets, to avoid overfitting to the training set. The model will handle the early stopping using the validation set. For the `aaron_sheep` dataset, we used a split of 7400/300/300 examples, and put each inside python lists called `train_data`, `validation_data`, and `test_data`. Afterwards, we created a subdirectory called `datasets/aaron_sheep` and we use the built-in `savez_compressed` method to save a compressed version of the dataset in a `aaron_sheep.npz` file. In all of our experiments, the size of each dataset is an exact multiple of 100, and use a `batch_size` of 100. Deviate at your own peril.

```
filename = os.path.join('datasets/your_dataset_directory', 'your_dataset_name.npz')
np.savez_compressed(filename, train=train_data, valid=validation_data, test=test_dat
```

We also performed simple stroke simplification to preprocess the data, called Ramer-Douglas-Peucker. There is some easy-to-use open source code for applying this algorithm here. In practice, we can set the `epsilon` parameter to a value between 0.2 to 3.0, depending on how aggressively we want to simply the lines. In the paper we used an `epsilon` parameter of 2.0. We suggest you build a dataset where the maximum sequence length is less than 250.

If you have a large set of simple SVG images, there are some available libraries to convert subsets of SVGs into line segments, and you can then apply RDP on the line segments before converting the data to *stroke-3* format.

Pre-Trained Models

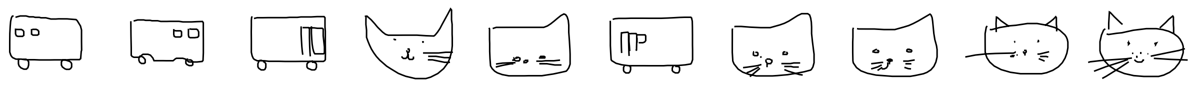
We have provided pre-trained models for the `aaron_sheep` dataset, for both conditional and unconditional training mode, using vanilla LSTM cells and LSTM cells with Layer Normalization. These models will be downloaded by running the Jupyter Notebook. They are stored in:

- `/tmp/sketch_rnn/models/aaron_sheep/lstm`
- `/tmp/sketch_rnn/models/aaron_sheep/lstm_uncond`
- `/tmp/sketch_rnn/models/aaron_sheep/layer_norm`
- `/tmp/sketch_rnn/models/aaron_sheep/layer_norm_uncond`

In addition, we have provided pre-trained models for selected QuickDraw datasets:

- `/tmp/sketch_rnn/models/owl/lstm`
- `/tmp/sketch_rnn/models/flamingo/lstm_uncond`
- `/tmp/sketch_rnn/models/catbus/lstm`
- `/tmp/sketch_rnn/models/elephantpig/lstm`

Using a Model with Jupyter Notebook



Let's get the model to interpolate between a cat and a bus!

We've included a simple Jupyter Notebook to show you how to load a pre-trained model and generate vector sketches. You will be able to encode, decode, and morph between two vector images, and also generate new random ones. When sampling images, you can tune the `temperature` parameter to control the level of uncertainty.

Citation

If you find this project useful for academic purposes, please cite it as:

```
@ARTICLE{sketchrnn,
  author          = {{Ha}, David and {Eck}, Douglas},
  title           = "{A Neural Representation of Sketch Drawings}",
  journal         = {ArXiv e-prints},
  archivePrefix   = "arXiv",
  eprinttype      = {arxiv},
  eprint          = {1704.03477},
  primaryClass    = "cs.NE",
  keywords        = {Computer Science - Neural and Evolutionary Computing,
Computer Science - Learning, Statistics - Machine Learning},
  year            = 2017,
  month          = apr,
}
```