Regressive and Autoregressive Generative Methods

Denis Derkach, Maksim Artemev, Artem Ryzhikov

CS HSE faculty, Generative Models, spring 2020

Contents

Neural Networks for Density Estimation

Motivation

- In previous lecture we have seen: KDE, kNN methods.
- A common problem for non-parametric case is the bandwidth or number of components estimation.
- Also for high dimensions non-parametric methods do not converge as quickly as one would desire ($\mathcal{O}(n^{\frac{-4}{4+d}}$.

Reminder from Lec 1: Sampling from CDF

If we have a parametric model, the life simplifies dramatically:

- Specify a latent $p(z)$ followed by a procedure $f_\theta : Z \to X$.
- Key point: in this setting, sampling data is almost always easy.
- Inversion sampling?

  $$z \sim Unif(0;1); x = F_\phi^{-1}(z); x \sim Exp(\phi).$$

  Here F_ϕ is the CDF of the exponential distribution, $F_\phi(x) = 1 - \exp(x\phi)$, with $F_\phi^{-1}(z) = -\phi \log(1 - z)$.
- Let’s try to see what we can do about it, when we do not have the functional form.

Stochastic Learning of the Cumulative

1. Let $x_1 \leq x_2 \leq \dots x_N \in \mathbb{R}$ be the data points with PDF $g(x)$ and a corresponding CDF $G(x) = \int\limits_{-\infty}^{x} g(x')dx'$. $G(x) \sim U[0;1]$.
2. We want to train a network that outputs $H(x,w) = G(x)$.
3. We can fit regression to a random ranked variable $u_1 \leq \dots \leq u_N \sim U[0;1]$:

$$L = \sum_{n=1}^{N} (H(x_n) - u_n)^2 +$$

$$+\lambda \sum_{h=1}^{N_h} \Theta(H(y_k) - H(y_k + \Delta))(H(y_k) - H(y_k + \Delta))^2,$$

with $\Theta(x)$ being step function and λ and Δ tunable constants.
4. repeat from (2) until convergence.

M. Magdon-Ismail, A. Grover, Neural Networks for Density Estimation, NIPS'99

SLC: Discussion

- Allows to fairly simple create a 1D sampling procedure;
- randomly generated targets are different for every cycle, which has a smoothing effect;
- only applicable to estimating univariate densities, because, for the multivariate case, the nonlinear mapping $y = G(x)$ will not necessarily result in a uniformly distributed output y.
- Can be extended by smooth interpolation of the cumulative (SIC) with neural network.

SLC, SIC: Convergence

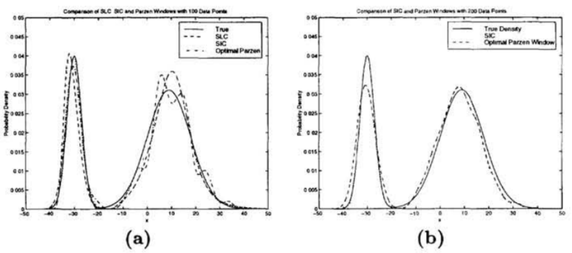


Figure 1: Comparison of optimal Parzen windows, with neural network estimators. Plotted are the true density and the estimates (SLC, SIC, Parzen window with optimal kernel width [6, pg 40]). Notice that even the optimal Parzen window is bumpy as compared to the neural network.

The convergence is better than for Parzen window, for 1D $\mathcal{O}(\log\log(n)/n)$, which is faster convergence than the kernel density estimator.

Autoregressive Generative Models

Motivation: Back Propagation Properties

- requires labeled training data, however, almost all data is unlabeled;
- learning time does not scale well;
- can get stuck in poor local optima;
- MLP is not a generative model, it only focuses on P(Y|X). We would like a generative approach that could learn P(X) as well.
- Solution: Deep Belief Networks, a generative graphical model.

Reminder: Chain Rule

- Probability Chain Rule:

$$p(x_1, x_2, \dots, x_n) = p(x_1)p(x_2|x_1)\dots p(x_n|x_1, \dots, x_{n-1}).$$

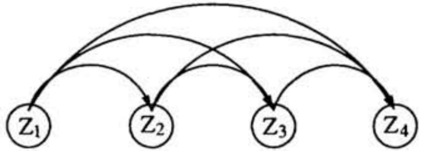
- What if we can approximate factor by a neural net?

$$p(x_1, x_2, \dots, x_n) = p(x_1)p_{NN}(x_2|x_1)\dots p_{NN}(x_n|x_1, \dots, x_{n-1}).$$

- Should we be able to estimate it by using ever deeper network? What can be done to estimate this?

Autoregressive modeling

The term autoregressive originates from the literature on time-series models where observations from the previous time-steps are used to predict the value at the current time step. Example of Bayes net (no NN assumed):



Fully Visible Sigmoid Belief Network: Motivation

- Imagine that we need to generate 2D distribution with only 2 outcomes possible per observable.
- The chain rule will become:

$$p(x_1, x_2) = p(x_1)p(x_2|x_1).$$

- What is the distribution for $p(x_2|x_1)$? Seems like a Bernoulli distribution.
- How can we parameterize it? Using sigmoid function:

$$p(x_2|x_1) = \sigma(a_0 + a_1 p(x_1)).$$

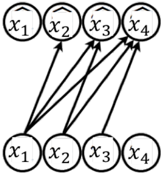
FVSBN: MNIST

We can try to do the same with MNIST (we just need to make binarization to follow previous logic).

In case of MNIST we have $n = 28 \times 28 = 784$ pixels, each pixel is black or white.
The conditional variables $X_i|X_1,,X_{i1}$ are Bernoulli with parameters.

$$\hat{x}_i = p(X_i = 1|x_1,,x_{-1};\alpha^i) = \sigma(\alpha_0^i + \sum_{j=1}^{i-1} \alpha_j^i x_j)$$

FVSBN: evaluation



FVSBN

We can evaluate by computing corresponding conditional probabilities:

$$p(X_1 = 0, X_2 = 1, X_3 = 1, X_4 = 0) =$$
$$= (1 - \hat{x}_1) \times \hat{x}_2 \times \hat{x}_3 \times (1 - \hat{x}_4) =$$
$$= (1 - \hat{x}_1) \times \hat{x}_2(X_1 = 0) \times$$
$$\times x^3(X_1 = 0, X_2 = 1) \times (1 - \hat{x}_4(X_1 = 0, X_2 = 1, X_3 = 1)).$$

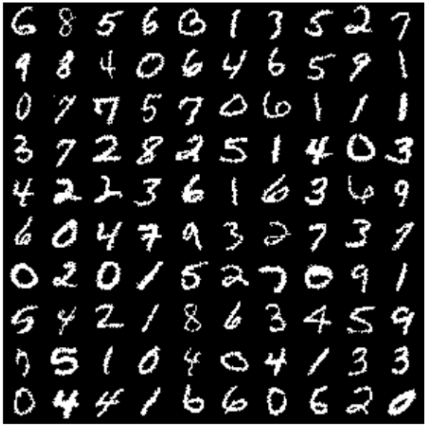
FVSBN: sampling

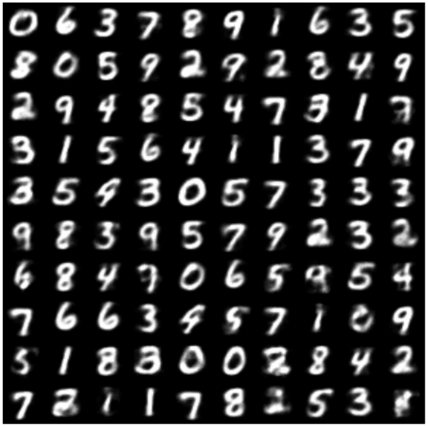
Consecutive sampling $p(x_1, \dots, x_{784})$:

1. Sample $\bar{x}_1 \sim p(x_1)$.
2. Sample $\bar{x}_2 \sim p(x_2|x_1 = \bar{x}_1)$
3. Sample $\bar{x}_3 \sim p(x_3|x_1 = \bar{x}_1, x_2 = \bar{x}_2)$

.

MNIST test





Training data on the left (MNIST binarized). Samples from the model on the right.

Z. Gan et al., Learning Deep Sigmoid Belief Networks with Data Augmentation, AISTAT-2015

FVSBN: discussion

- easy to sample;
- easy to evaluate;
- cannot obtain additional capacity;
- hard to scale the capacity, still can suffer from curse of dimensionality;
- number of parameters $n^2/2$.

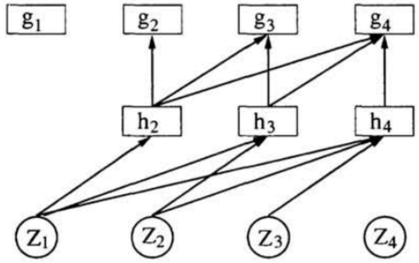
I. Goodfellow et al., Deep Learning Book, Ch. 20 (Russian edition is better here)

Neural Autoregressive Network

We can increase the capacity of model by inserting a one layer neural network instead of sigmoid.

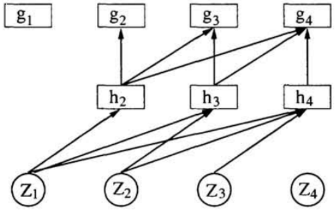
This brings two advantages:

- NN is more capable of catching better parameterisation.
- NN(i-1) can be reused to predict i-th iteration.



Y. Bengio et al. Modeling High-Dimensional Discrete Data with Multi-Layer Neural Networks, NIPS-00)

NAN: math



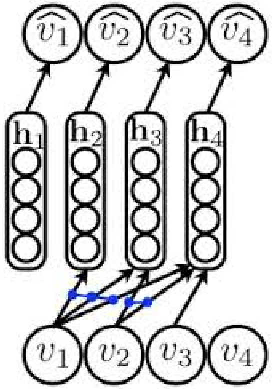
$$\mathbf{h}_i = \sigma(A_i \mathbf{x}_{<i} + \mathbf{c}_i)$$

$$\hat{x}_i = p(x_i | x_1, \cdots, x_{i-1}; \underbrace{A_i, \mathbf{c}_i, \boldsymbol{\alpha}_i, b_i}_{\text{parameters}}) = \sigma(\boldsymbol{\alpha}_i \mathbf{h}_i + b_i)$$

For example $\mathbf{h}_2 = \sigma\left(\underbrace{\begin{pmatrix} \vdots \end{pmatrix}}_{A_2} x_1 + \underbrace{\begin{pmatrix} \vdots \end{pmatrix}}_{c_2}\right)$ $\mathbf{h}_3 = \sigma\left(\underbrace{\begin{pmatrix} \vdots & \vdots \end{pmatrix}}_{A_3} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \underbrace{\begin{pmatrix} \vdots \end{pmatrix}}_{c_3}\right)$

NADE: Neural Autoregressive Density Estimation

Once can go even further: tie weights to reduce the number of parameters and speed up computation (see blue dots in the figure).



H. Larochelle The Neural Autoregressive Distribution Estimator, AISTATS-11)

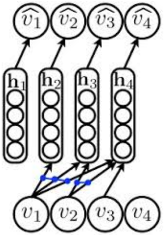
NADE: math

The amount of parameters will be reduced due to the shared weights:

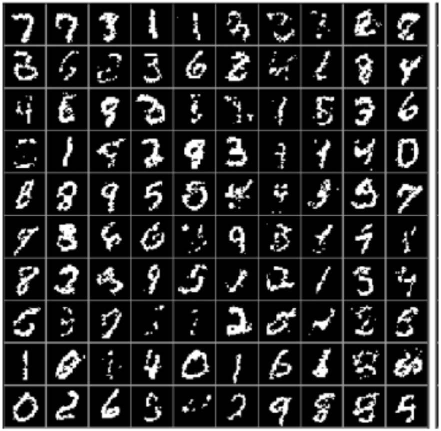
$$\mathbf{h}_i = \sigma(W_{\cdot,<i}\mathbf{x}_{<i} + \mathbf{c})$$

$$\hat{x}_i = p(x_i|x_1, \cdots, x_{i-1}) = \sigma(\boldsymbol{\alpha}_i \mathbf{h}_i + b_i)$$

For example $\mathbf{h}_2 = \sigma\left(\underbrace{\begin{pmatrix} \vdots \\ \mathbf{w}_1 \\ \vdots \end{pmatrix}}_{A_2} x_1\right)$ $\mathbf{h}_3 = \sigma\left(\underbrace{\begin{pmatrix} \vdots & \vdots \\ \mathbf{w}_1 & \mathbf{w}_2 \\ \vdots & \vdots \end{pmatrix}}_{A_3} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right)$ $\mathbf{h}_4 = \sigma\left(\underbrace{\begin{pmatrix} \vdots & \vdots & \vdots \\ \mathbf{w}_1 & \mathbf{w}_2 & \mathbf{w}_3 \\ \vdots & \vdots & \vdots \end{pmatrix}}_{A_3} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}\right)$



NADE: results and discussion



- NADE allows for better image generation.
- The idea of calculations is closely related to Restricted Boltzman Machines.
- Can be improved to make k steps (T. Raiko et al., NADE-k).
- The amount of parameters is $\mathcal{O}(nd)$.

General discrete distributions

So far, we used only two-state pixels (black and white). How to extend the models to any amount of states (for example, pixels can have $0, \ldots, 255$)?

Let $\hat{x}_i$ parameterize a categorical distribution (and use softmax):

$$\mathbf{h}_i = \sigma(W_{\cdot,<i}\mathbf{x}_{<i} + \mathbf{c})$$
$$p(x_i|x_1, \cdots, x_{i-1}) = Cat(p_i^1, \cdots, p_i^K)$$
$$\hat{\mathbf{x}}_i = (p_i^1, \cdots, p_i^K) = softmax(X_i\mathbf{h}_i + \mathbf{b}_i)$$

Softmax generalizes the sigmoid/logistic function $\sigma(\cdot)$ and transforms a vector of K numbers into a vector of K *probabilities* (non-negative, sum to 1).

$$softmax(\mathbf{a}) = softmax(a^1, \cdots, a^K) = \left(\frac{\exp(a^1)}{\sum_i \exp(a^i)}, \cdots, \frac{\exp(a^K)}{\sum_i \exp(a^i)}\right)$$

Real valued distributions

We also can deal with the real-valued distributions. For this we can let $\hat{x}_i$ parametrize a continuos distribution. For example, a mixture of K Gaussians. The mixture of Gaussians parameters for the d-th conditional, $\theta_d = \alpha_d, \mu_d, \sigma_d$, are set by

$$\begin{aligned}
K \text{ mixing fractions,} \qquad & \boldsymbol{\alpha}_d = \operatorname{softmax}\left(\boldsymbol{V}_d^{\alpha\top}\boldsymbol{h}_d + \boldsymbol{b}_d^{\alpha}\right) \\
K \text{ component means,} \qquad & \boldsymbol{\mu}_d = \boldsymbol{V}_d^{\mu\top}\boldsymbol{h}_d + \boldsymbol{b}_d^{\mu} \\
K \text{ component standard deviations,} \qquad & \boldsymbol{\sigma}_d = \exp\left(\boldsymbol{V}_d^{\sigma\top}\boldsymbol{h}_d + \boldsymbol{b}_d^{\sigma}\right),
\end{aligned}$$

Where V's are $H \times K$ matrices and b's are K-dimensional vectors.

B. Uria et al. RNADE: The real-valued neural autoregressive density-estimator, NIPS-12)

RNADE: results and discussion

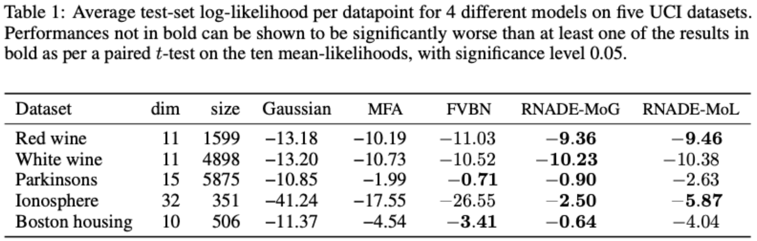


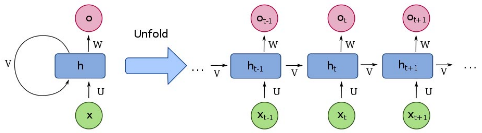
Table 1: Average test-set log-likelihood per datapoint for 4 different models on five UCI datasets. Performances not in bold can be shown to be significantly worse than at least one of the results in bold as per a paired t-test on the ten mean-likelihoods, with significance level 0.05.

Dataset	dim	size	Gaussian	MFA	FVBN	RNADE-MoG	RNADE-MoL
Red wine	11	1599	−13.18	−10.19	−11.03	**−9.36**	**−9.46**
White wine	11	4898	−13.20	−10.73	−10.52	**−10.23**	−10.38
Parkinsons	15	5875	−10.85	−1.99	**−0.71**	**−0.90**	−2.63
Ionosphere	32	351	−41.24	−17.55	−26.55	**−2.50**	**−5.87**
Boston housing	10	506	−11.37	−4.54	**−3.41**	**−0.64**	−4.04

- RNADE deals with real numbered observables;
- it outperforms mixture models on many datasets;
- it is unclear how to select the factorization order in many applications (see EoNADE that takes into account this problem);
- tend to get complicated with many dimensions.

Recurrent Neural Nets

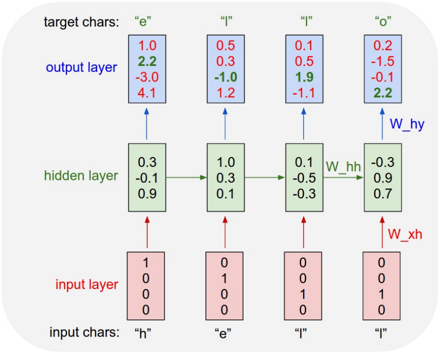
One way to approach the problem of having long tails of previous dimensions is to use a recurrent neural nets:



The update rule will thus be: $h_{t+1} = \tanh(W_{hh}h_t + W_{xh}x_{t+1})$.
The prediction: $o_{t+1} = W_{hy}h_{t+1}$.

- hidden layer h_t is a summary of the inputs seen till time t;
- output layer o_{t1} specifies parameters for conditional $p(x_t|x_{1:t1})$;
- Parameterized by b_0 (initialization), and matrices W_{hh}, W_{xh}, W_{hy}. Constant number of parameters w.r.t n.

RNN: example



- $xi \in \{h, e, l, o\}$. One can use one-hot encoding. We want to build "Hello".
- Autoregressive rule:

  $p(x = hello) = p(x_1 = h)p(x_2 = e|x_1 = h)p(x_3 = l|x_1 = h, x_2 = e) \dots p(x_5 = o|x_1 = h, x_2 = e, x_3 = l, x_4 = l)$.
- $p(x_2 = e|x_1 = h) = softmax(o_1)$;

$$o_1 = W_{hy}h_1;$$

$$h_1 = tanh(W_{hh}h_0 + W_{xh}x_1).$$

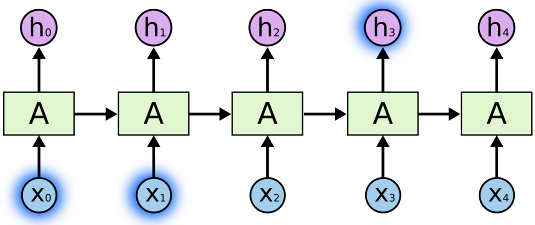
From A. Karpathy blog)

Problem of Long-Term Dependencies

When dealing with longer sequences there might be some problems. Imagine we try to predice the last word of the sentence (and we trained on words:

> “the clouds are in the sky”

The RNN works perfectly:



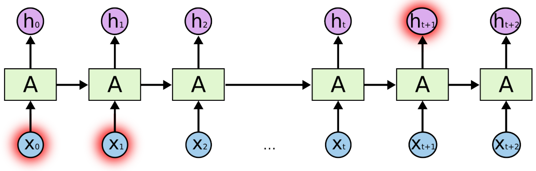
From C. Olah blog)

Problem of Long-Term Dependencies

The longer sequence, however, might suffer from the problems with long-term memory:

> “I grew up in France… I speak fluent French.”

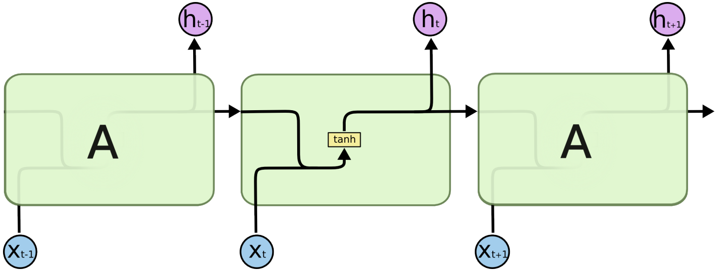
The RNN works perfectly:



Y. Bengio Learning long-term dependencies with gradient descent is difficult, IEEE TNN
From C. Olah blog)

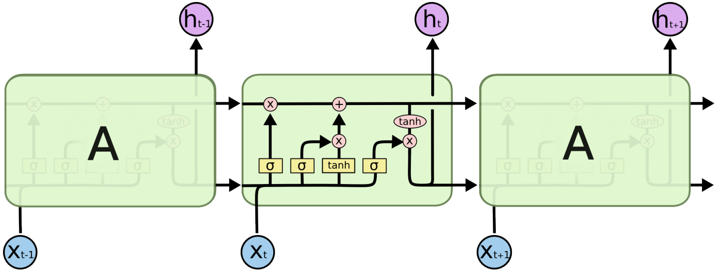
Long Short Term Memory networks

This can be taken into account by using LSTM cells that gives hints to RNN what to store for longer term.
A usual one-layer RNN looks like:



Long Short Term Memory networks

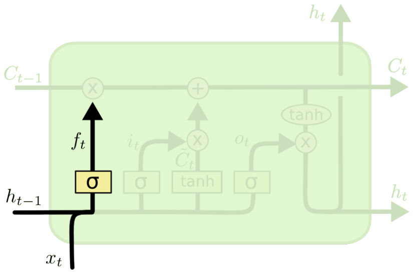
This can be taken into account by using LSTM cells that gives hints to RNN what to store for longer term.
We can update it to:



Where additional layers store the information on what to store only.

LSTM: explained

Language model trying to predict the next word based on all the previous ones.

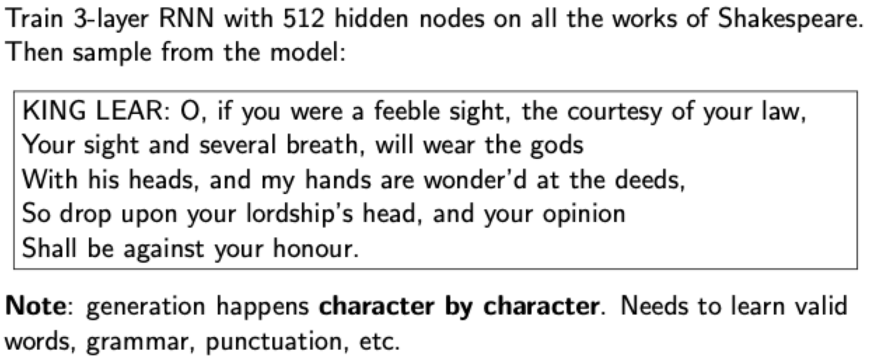


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

We first want to forget the gender.

Generation of Characters

Addition of LSTM layers creates a possibility to generate long texts.



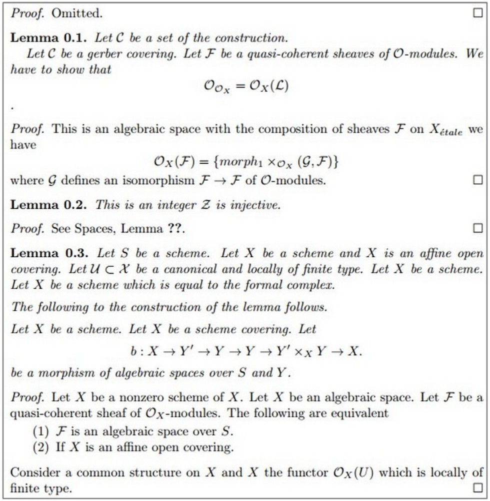
Train 3-layer RNN with 512 hidden nodes on all the works of Shakespeare. Then sample from the model:

KING LEAR: O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.

Note: generation happens **character by character**. Needs to learn valid words, grammar, punctuation, etc.

From A. Karpathy blog)

Generation of LaTeX Markup



Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*

Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{étale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

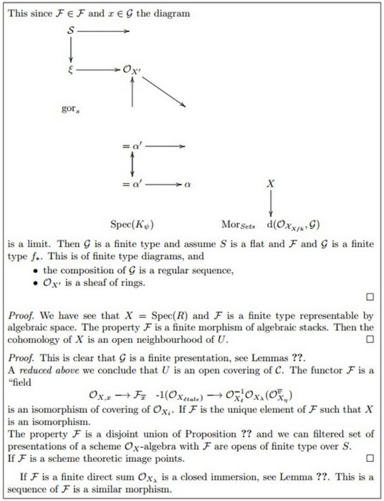
$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □



This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram

$$\begin{tikzcd}
\mathcal{S} \arrow[r] \arrow[d] & {} & & \\
\xi \arrow[r] & \mathcal{O}_{X'} \arrow[dr] & & \\
\mathrm{gor}_s & {} \arrow[u] & {} & \\
& =\alpha' \arrow[r] \arrow[d, leftrightarrow] & {} & \\
& =\alpha' \arrow[r] & \alpha & X \arrow[d] \\
& \mathrm{Spec}(K_\psi) & \mathrm{Mor}_{Sets} & \mathrm{d}(\mathcal{O}_{\mathcal{X}_{X/k}}, \mathcal{G})
\end{tikzcd}$$

is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type f_*. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

□

Proof. We have see that $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. □

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \ \text{-}1(\mathcal{O}_{X_{étale}}) \longrightarrow \mathcal{O}^{-1}_{X_\ell}\mathcal{O}_{X_\lambda}(\mathcal{O}^{\overline{v}}_{X_\eta})$$

is an isomorphism of covering of $\mathcal{O}_{X_i}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over S.
If $\mathcal{F}$ is a scheme theoretic image points. □

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma ??. This is a sequence of $\mathcal{F}$ is a similar morphism.

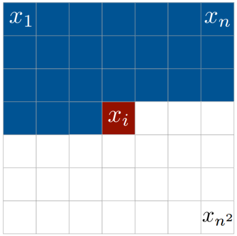
The Generator even able to omit the proof!

RNN: summary

- Pros:
  - Can be applied to sequences of arbitrary length.
  - Very general: For every computable function, there exists a finite RNN that can compute it.
  - Generalised for long sequences with LSTM.
- Cons:
  - Still requires an ordering.
  - Sequential likelihood evaluation (very slow for training).
  - Sequential generation (unavoidable in an autoregressive model).
  - Can be difficult to train (vanishing/exploding gradients).

PixelRNN

The idea can also be applied to generate images. We can model images pixel by pixel using raster scan order:

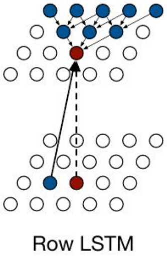


We need to make a conditional probability based on all colours of previous pixels and also RGB direction of current pixel.

LSTM for PixelRNN

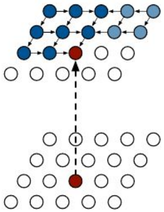
We can use to types of LSTMs that concentrate on different structure

Row LSTM



Hidden state(i,j) = Hidden state(i-1,j-1)+ Hidden state(i-1,j+1)+ Hidden state(i-1,j)+ p(i,j)

Diagonal BiLSTM



pixel(i, j) = pixel(i, j-1) + pixel(i-1, j)

.

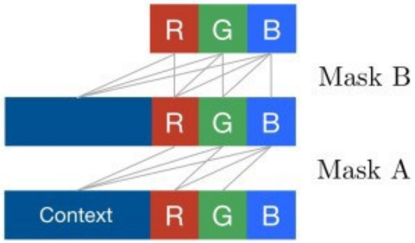
.

.

Masked Convolutions

We use two types of masks :

- Type A : this mask is only applied to the first convolutional layer and restricts connections to those colors in current pixels that have already been predicted.
- Type B : this mask is applied to other layers and allows connections to predicted colors in the current pixels



PixelRNN: results

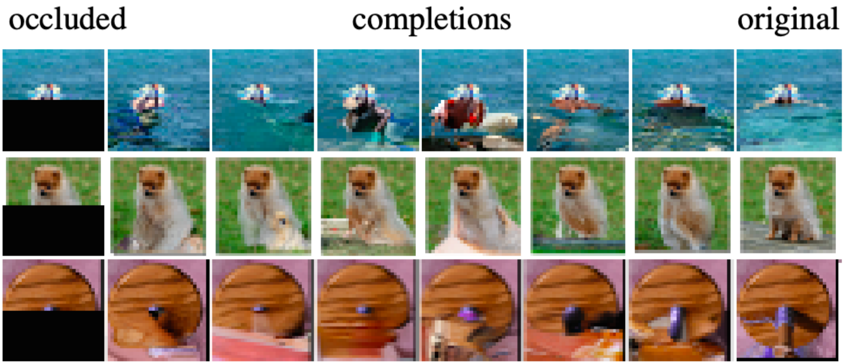
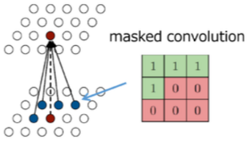


Figure 1. Image completions sampled from a PixelRNN.

- The evaluation is very slow.
- Basic features of images are caught.

PixelCNN

- Convolutions are natural for image data and easy to parallelize on modern hardware.
- We can use convolutional architecture to predict next pixel given context (a neighborhood of pixels).
- Since masked convolutions preserve raster scan order, need additional masking for colors order.



PixelCNN: results

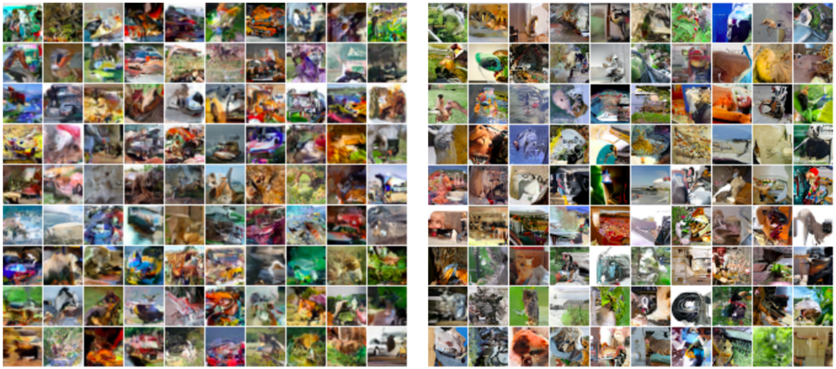


Figure 7. Samples from models trained on CIFAR-10 (left) and ImageNet 32x32 (right) images. In general we can see that the models capture local spatial dependencies relatively well. The ImageNet model seems to be better at capturing more global structures than the CIFAR-10 model. The ImageNet model was larger and trained on much more data, which explains the qualitative difference in samples.

Similar performance to PixelRNN, but much faster.

Adversarial Attacks Defence with PixelRNN

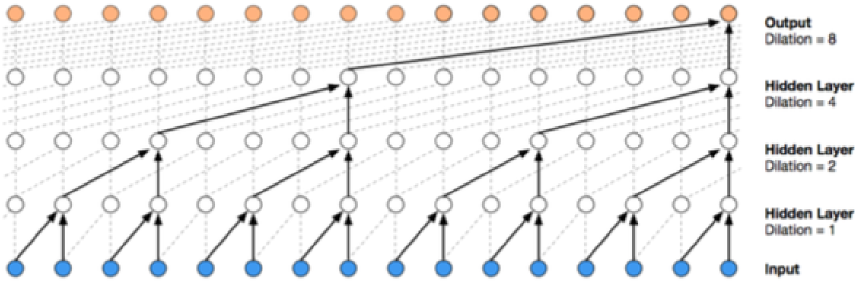
PixelDefend:

- Train a generative model $p(x)$ on clean inputs (PixelCNN)
- Given a new input x, evaluate $p(x)$.
- Adversarial examples are significantly less likely under $p(x)$.

Can this be used to determine the attack?

Dilated Convolution

- The advantage of RNNs over CNNs is that their memory lets them learn arbitrary long-distance dependencies.
- But we can dramatically increase a CNN's receptive field using dilated convolution.



WaveNET

- WaveNet is an autoregressive model for raw audio based on causal dilated convolutions.
- Audio needs to be sampled at at least 16k frames per second for good quality. So the sequences are very long.
- WaveNet uses dilations of 1, 2, . . . , 512, so each unit at the end of this block as a receptive field of length 1024, or 64 milliseconds.
- It stacks several of these blocks, so the total context length is about 300 milliseconds.
- https://deepmind.com/blog/wavenet-generative-model-raw-audic

Summary for Autoregressive models

- Easy to sample from
  - Sample $\bar{x}_0 \sim p(x_0)$;
  - Sample $\bar{x}_1 \sim p(x_1|x_0 = \bar{x}_0)$;
  - ...
- Easy to compute probability $p(x = \bar{x})$
  - compute $p(x_0 = \bar{x}_0)$;
  - compute $p(x_1 = \bar{x}_1|x_0 = \bar{x}_0)$;
  - multiply together (sum their logarithms);
  - ...
  - Ideally, can compute all these terms in parallel for fast training
- Easy to extend to continuous variables.
- No natural way to get features, cluster points, do unsupervised learning.