# Adversarial Learning for Neural Dialogue Generation

Jiwei Li[1], Will Monroe[1], Tianlan Shi[1],
Sébastian Jean[2], Alan Ritter[3], Dan Jurafsky[1]

[1]Stanford University, [2]New York University,
[3]Ohio State University

Some slides/images taken from Ian Goodfellow, Jeremy Kawahara, Andrej Karpathy

# Talk Outline

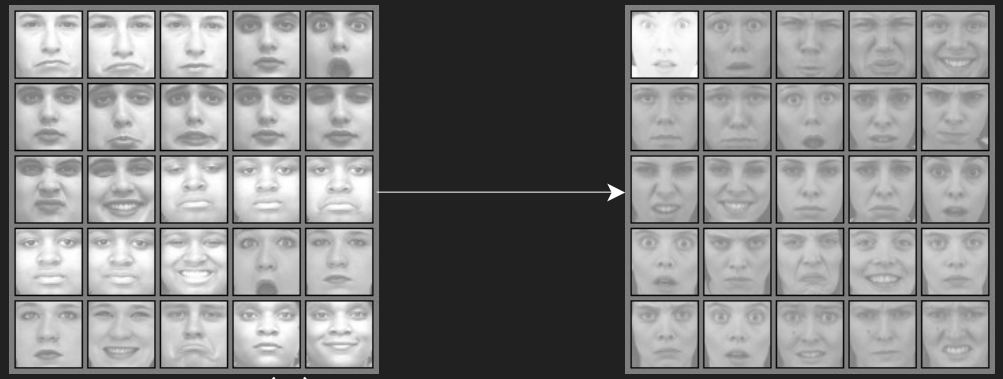
- Generative Adversarial Networks (Introduced by Goodfellow et. al, 2014)
- Policy gradients and REINFORCE
- GANs for Dialogue Generation (this paper)

# Talk Outline

- Generative Adversarial Networks (Introduced by Goodfellow et. al, 2014)
- Policy gradients and REINFORCE
- GANs for Dialogue Generation (this paper)

# Generative Modelling

- Have training examples $x \sim p_{data}(x)$
- Want a model that can draw samples: $x \sim p_{model}(x)$
- Where $p_{model} \approx p_{data}$



$x \sim p_{data}(x)$ $\qquad$ $x \sim p_{model}(x)$

# Why Generative Modelling?

- Conditional generative models
  - Speech synthesis: Text > Speech
  - Machine Translation: French > English
    - French: Si mon tonton tond ton tonton, ton tonton sera tondu.
    - English: If my uncle shaves your uncle, your uncle will be shaved
  - Image > Image segmentation
  - Dialogue Systems: Context > Response
- Environment simulator
  - Reinforcement learning
  - Planning
- Leverage unlabeled data

# Adversarial Nets Framework

- A game between two players:

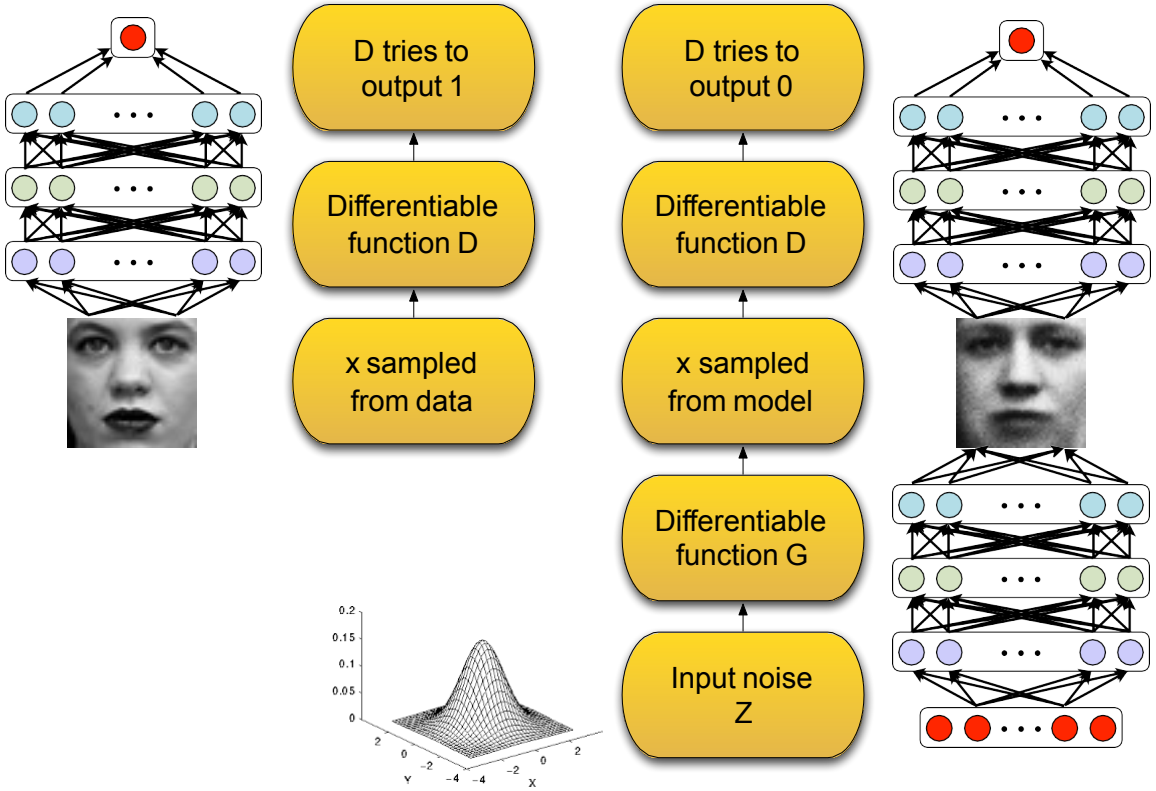
  1. Discriminator D
  2. Generator G

- D tries to discriminate between:

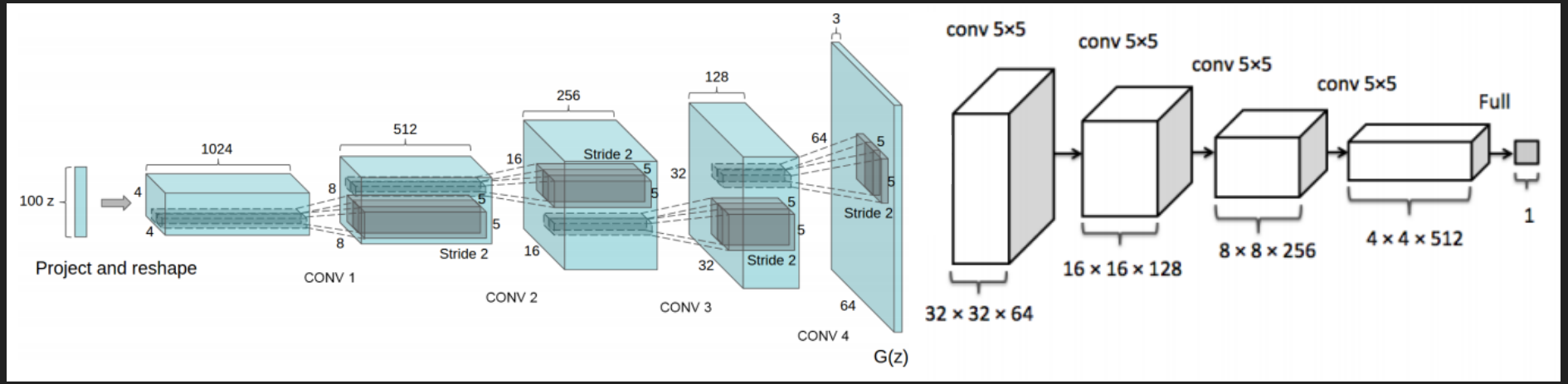
  - A sample from the data distribution and
  - A sample from the generator G

- G tries to “trick” D by generating samples that are hard for D to distinguish from true data.

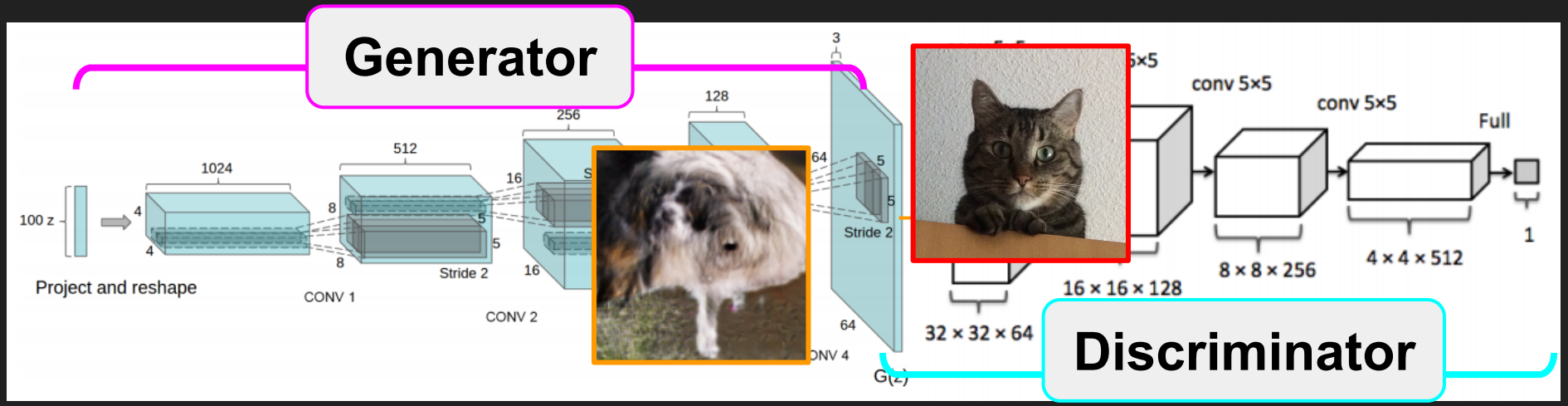
# Adversarial Nets Framework



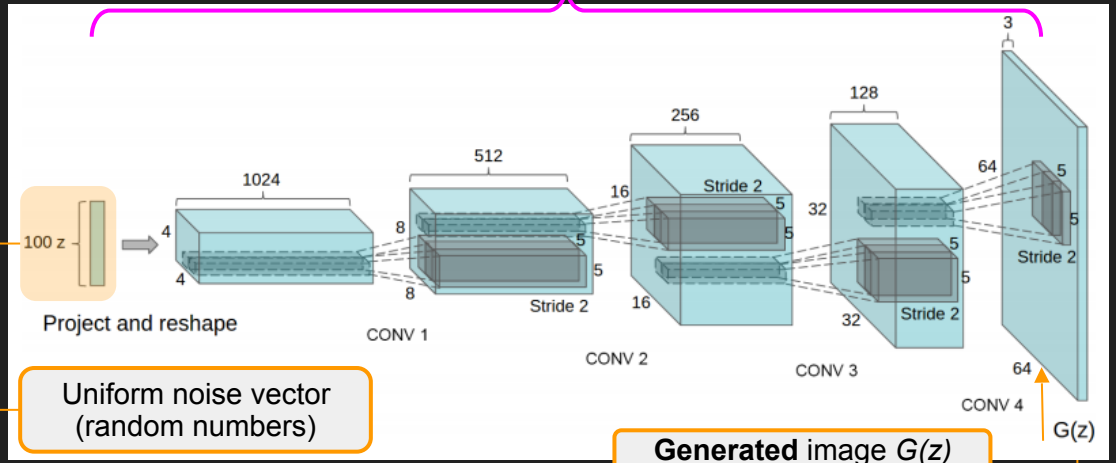
# Deep Convolutional Generative Adversarial Network




Can be thought of as two separate networks


Generator
Discriminator
100 z
Project and reshape
1024
512
256
128
3
CONV 1
CONV 2
Stride 2
Stride 2
conv 5×5
conv 5×5
Full
32 × 32 × 64
16 × 16 × 128
8 × 8 × 256
4 × 4 × 512
1

**Generator G(.)**
input=*random numbers*,
output=*generated image*

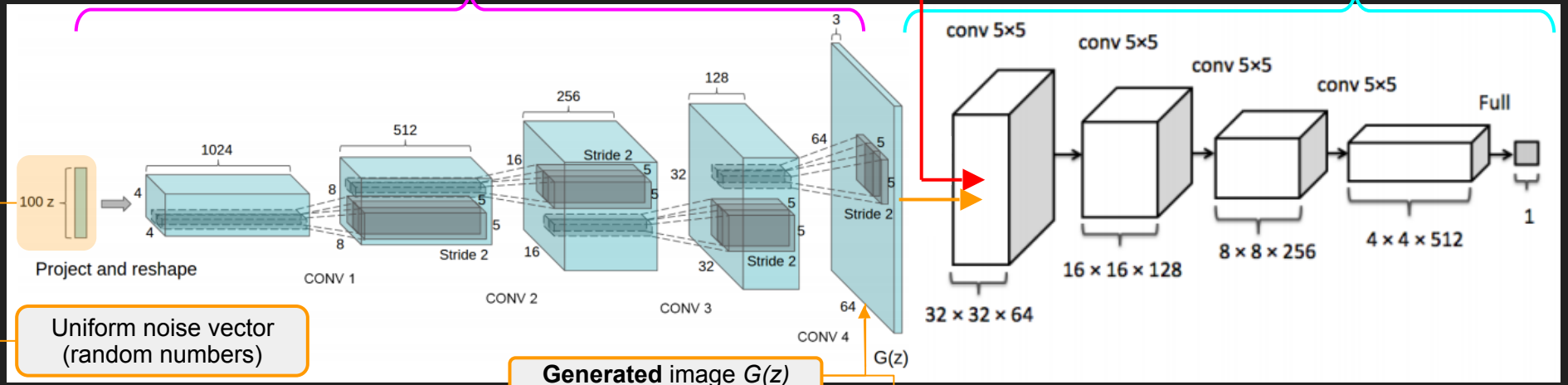



Uniform noise vector
(random numbers)

**Generated** image *G(z)*

**Generator G(.)**
input=*random numbers*,
output=*generated image*

**Discriminator D(.)**
input=*generated/real image*,
output=*prediction of real image*



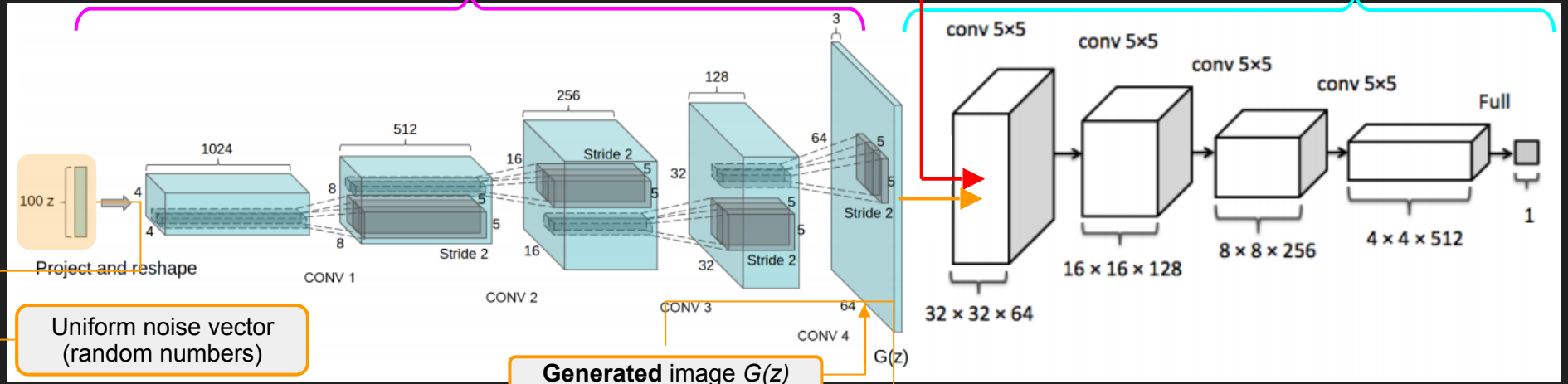

Uniform noise vector
(random numbers)

**Generated** image *G(z)*

**Real** image, so goal is *D(x)*=1

**Generator G(.)**
input=*random numbers*,
output=*generated image*

**Discriminator D(.)**
input=*generated/real image*,
output=*prediction of real image*




Uniform noise vector
(random numbers)

**Generated** image *G(z)*

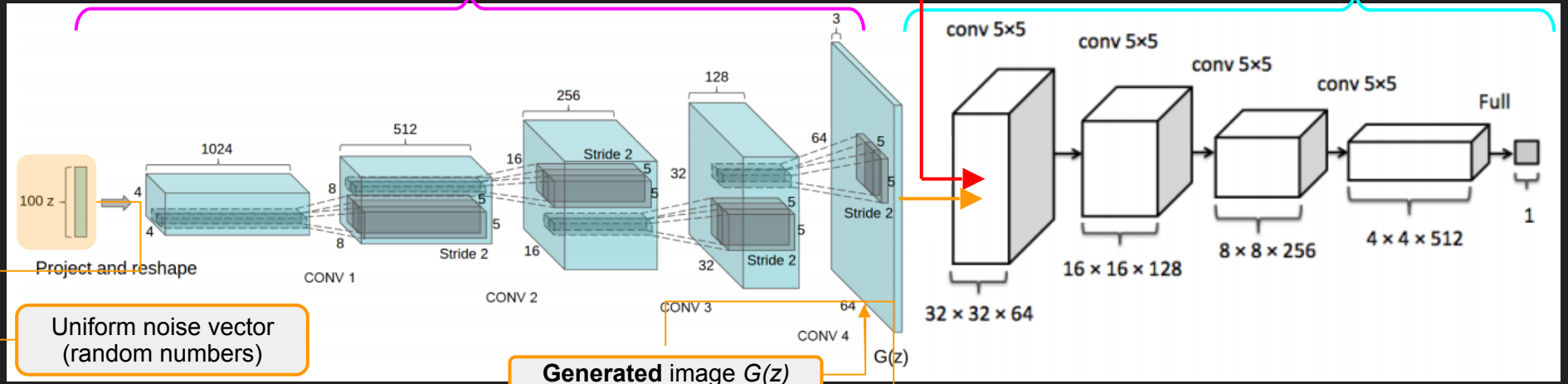
**Discriminator Goal**: discriminate between **real** and **generated** images
*i.e., D(x)=1,* where *x* is a real image
*D(G(z))=0,* where *G(z)* is a generated image

**Generated** image, so goal is *D(G(z))*=0

**Real** image, so goal is *D(x)*=1

**Generator G(.)**
input=*random numbers*,
output=*generated image*

**Discriminator D(.)**
input=*generated/real image*,
output=*prediction of real image*




Uniform noise vector
(random numbers)

**Generated** image *G(z)*

**Generator Goal:** Fool *D(G(z))*
i.e., generate an image *G(z)*
such that *D(G(z))* is wrong.
i.e., *D(G(z)) = 1*

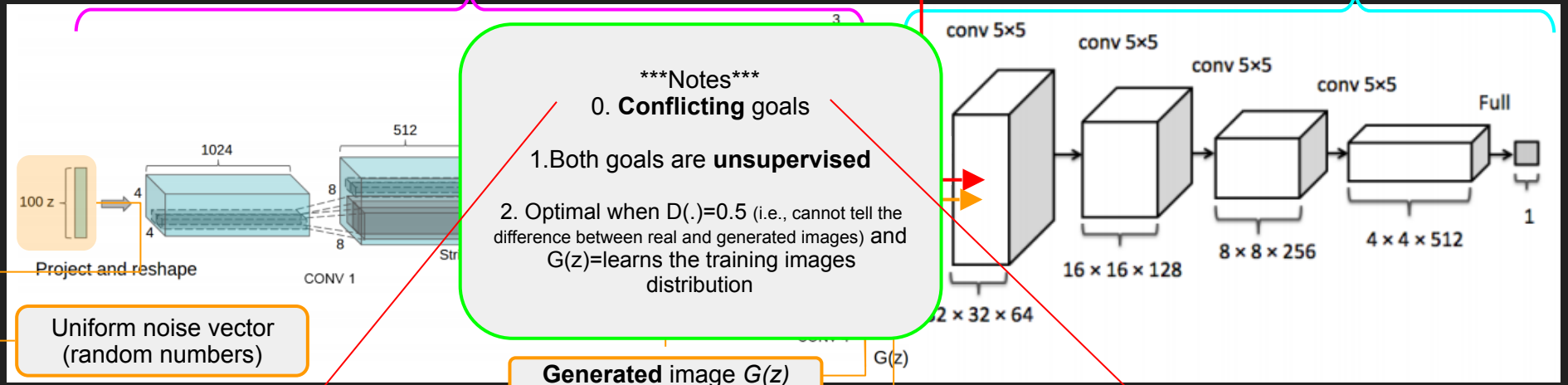
**Discriminator Goal**: discriminate between
**real** and **generated** images
*i.e., D(x)=1,* where *x* is a real image
*D(G(z))=0,* where *G(z)* is a generated image

**Generated** image, so goal is *D(G(z))*=0

**Real** image, so goal is *D(x)*=1

**Generator G(.)**
input=*random numbers*,
output=*generated image*

**Discriminator D(.)**
input=*generated/real image*,
output=*prediction of real image*




***Notes***
0. **Conflicting** goals

1.Both goals are **unsupervised**

2. Optimal when D(.)=0.5 (i.e., cannot tell the difference between real and generated images) and G(z)=learns the training images distribution

Uniform noise vector
(random numbers)

**Generated** image *G(z)*

**Generator Goal:** Fool *D(G(z))*
i.e., generate an image *G(z)*
such that *D(G(z))* is wrong.
i.e., *D(G(z)) = 1*

**Discriminator Goal**: discriminate between **real** and **generated** images
*i.e., D(x)=1,* where *x* is a real image
*D(G(z))=0,* where *G(z)* is a generated image

**Generated** image, so goal is *D(G(z))*=0

# Zero-Sum Game

- Minimax objective function:

$$\min_{G} \max_{D} V(D, G) = \mathrm{E}_{\boldsymbol{x} \sim p_{\mathrm{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathrm{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

**for** number of training iterations **do**

**for** $k$ steps **do**

- Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of $m$ examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

**end for**

- Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

**end for**

**for** number of training iterations **do**

**for** $k$ steps **do**

- Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of $m$ examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

maximize

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

Loss function to maximize for the **Discriminator**

**end for**

- Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

minimize

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

Loss function to minimize for the **Generator**

**end for**

**for** number of training iterations **do**

**for** $k$ steps **do**

- Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of $m$ examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

maximize

Gradient w.r.t the parameters of the **Discriminator**

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

Loss function to maximize for the **Discriminator**

**end for**

- Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

minimize

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

Loss function to minimize for the **Generator**

**end for**

Gradient w.r.t the parameters of the **Generator**

**for** number of training i

**for** $k$ steps **do**

- Sample minibatc … e prior $p_g(\boldsymbol{z})$.
- Sample minibatch of $m$ examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

**end for**

- Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

**end for**

[interpretation] compute the gradient of the loss function, and then update the parameters to min/max the loss function (gradient descent/ascent)

maximize

Gradient w.r.t the parameters of the **Discriminator**

Loss function to maximize for the **Discriminator**

minimize

Loss function to minimize for the **Generator**

Gradient w.r.t the parameters of the **Generator**

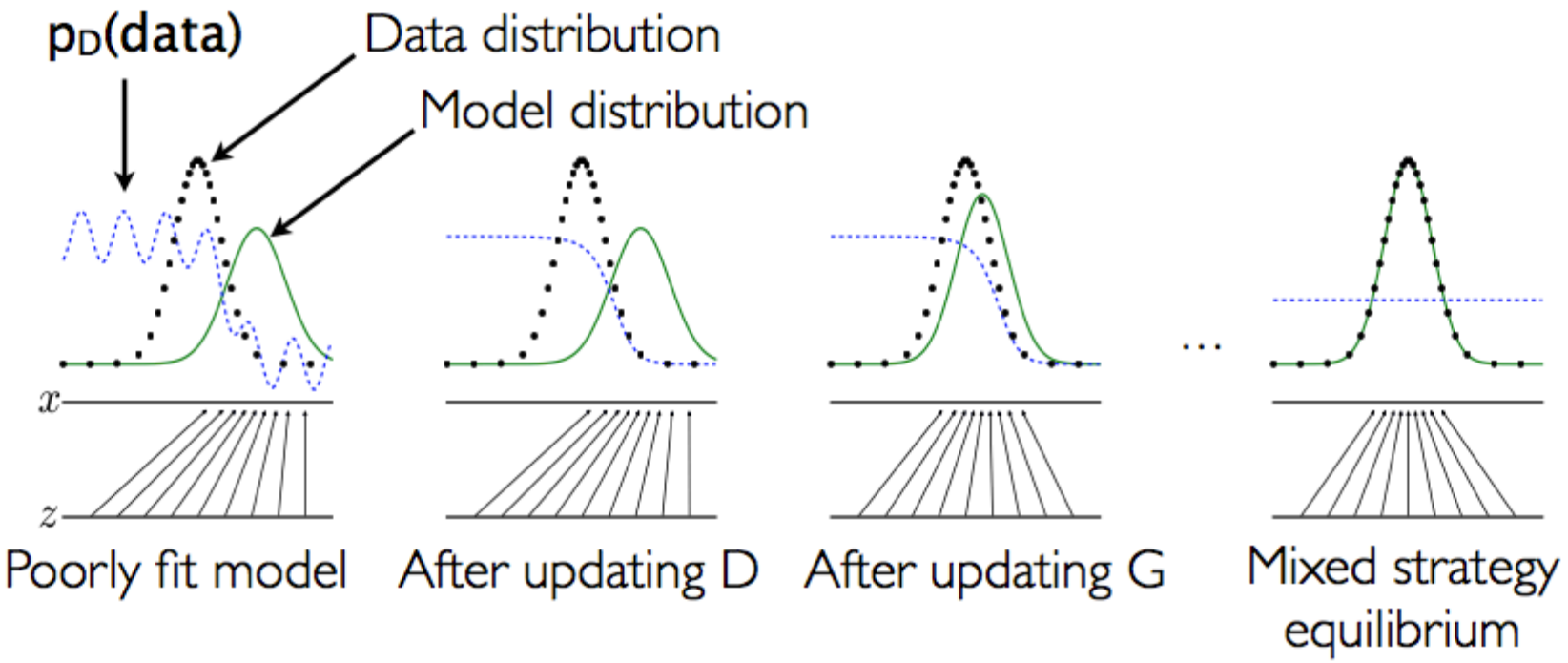
# Theoretical Results

- Assuming enough data and model capacity, we have a unique global optimum
  - Generator distribution corresponds to data distribution
  - For a fixed generator, the optimal discriminator is:

$$D_G^*(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$$

  - So at optimum, discriminator outputs 0.5 (can't tell if input is generated by G or from data)

# Learning Process



# GANs - The Good and the Bad

- Generator is forced to discover features that explain the underlying distribution
- Produce sharp images instead of blurry like MLE.
- However, generator can be quite difficult to train
- Can suffer from problem of ‘missing modes’

# Talk Outline

- Discussion of Generative Adversarial Networks (Introduced by Goodfellow et. al, 2014)
- Policy Gradients and REINFORCE
- Discussion of GANs for Dialogue Generation (this paper)

# Policy Gradient

- We have a differentiable stochastic policy $\pi(x;\theta)$
- We sample an action x from $\pi(x;\theta)$ — the future reward or ‘return’ for action x is r(x)
- We want to maximize the expected return $E_{x\sim\pi(x;\theta)}[r(x)]$

# Policy Gradient

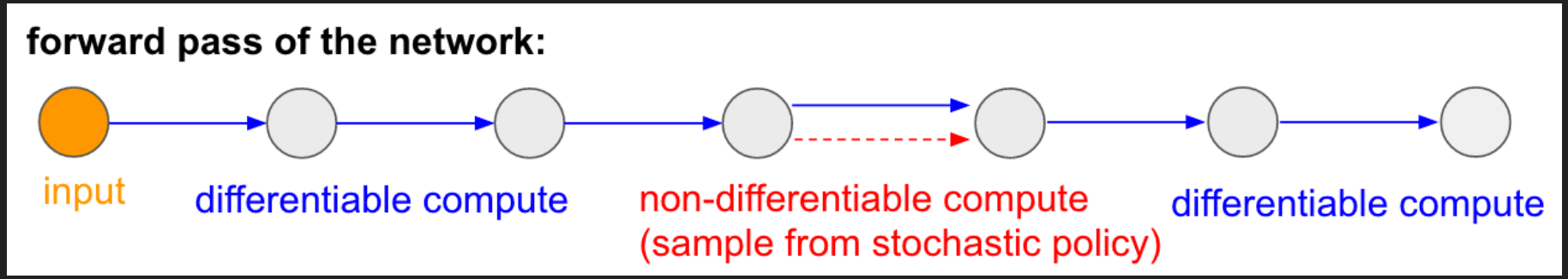
- We want to maximize the expected return $E_{x\sim\pi(x;\theta)}[r(x)]$
- So we'd like to compute the gradient $\nabla_\theta E_{x\sim\pi(x;\theta)}[r(x)]$

$$
\begin{aligned}
\nabla_\theta E_x[f(x)] &= \nabla_\theta \sum_x p(x) f(x) && \text{definition of expectation} \\
&= \sum_x \nabla_\theta p(x) f(x) && \text{swap sum and gradient} \\
&= \sum_x p(x) \frac{\nabla_\theta p(x)}{p(x)} f(x) && \text{both multiply and divide by } p(x) \\
&= \sum_x p(x) \nabla_\theta \log p(x) f(x) && \text{use the fact that } \nabla_\theta \log(z) = \frac{1}{z}\nabla_\theta z \\
&= E_x[f(x) \nabla_\theta \log p(x)] && \text{definition of expectation}
\end{aligned}
$$

# REINFORCE

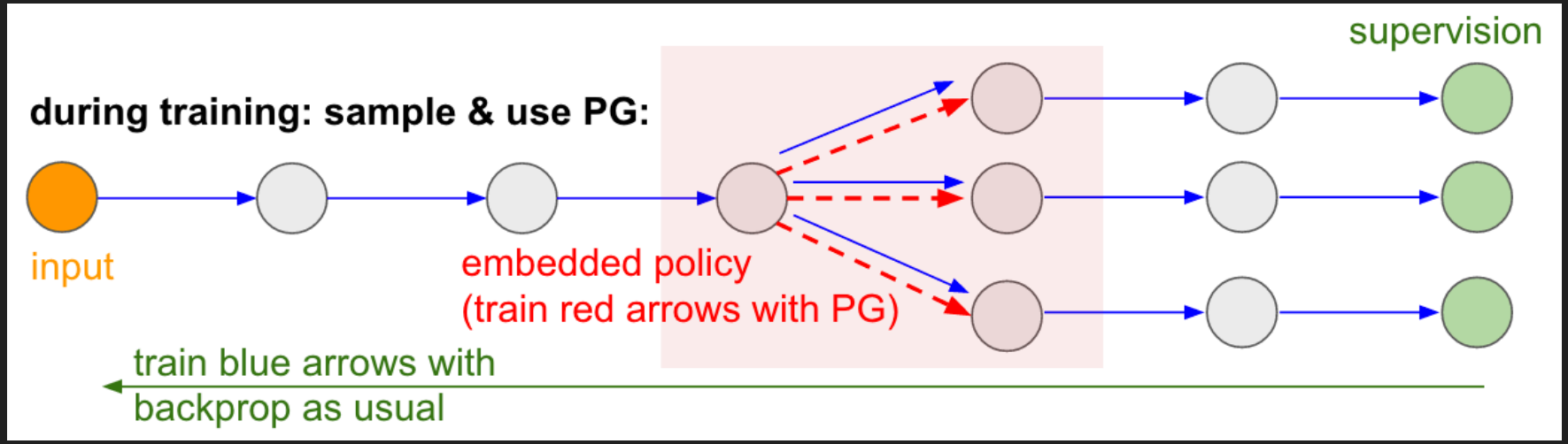
- We know that $\nabla_\theta E_{x\sim\pi(x|\theta)}[r(x)]$ is nothing but $E_{x\sim\pi(x;\theta)}[r(x)\nabla_\theta \log(\pi(x;\theta))]$
- We can estimate this gradient using samples from one or more episodes — we can do this because the policy itself is differentiable
- This can be seen as a Monte Carlo Policy Gradient, which is nothing but REINFORCE

# Estimate gradient of sampling operation




- Sampling operation inside a neural network — this is the policy

# Estimate gradient of sampling operation




- We sample an action x from $\pi(x;\theta)$, which gives us a reward r(x) — this could be a supervised loss
- We can now use REINFORCE to estimate gradient

# Talk Outline

- Discussion of Generative Adversarial Networks (Introduced by Goodfellow et. al, 2014)
- Policy Gradients and REINFORCE
- Discussion of GANs for Dialogue Generation (this paper)

# GANs for NLP: Dialogue systems

- Given dialogue history x, want to generate response y
- Generator G
  - Input to G: x
  - Output from G: y
- Discriminator D
  - Input to D: x, y
  - Output from D: Probability that (x, y) is from training data

# GANs for NLP: Dialogue systems

+ Gagan
+ Barun

- Given dialogue history x, want to generate response y
- Generator G
  - Input to G: x
  - Output from G: y
- Discriminator D
  - Input to D: x, y
  - Output from D: Probability that (x, y) is from training data

# GANs for NLP: Dialogue systems

Challenge:

- Typical seq2seq models for machine translation, dialogue generation etc. involve sampling from a distribution — can’t directly backpropagate from discriminator to generator

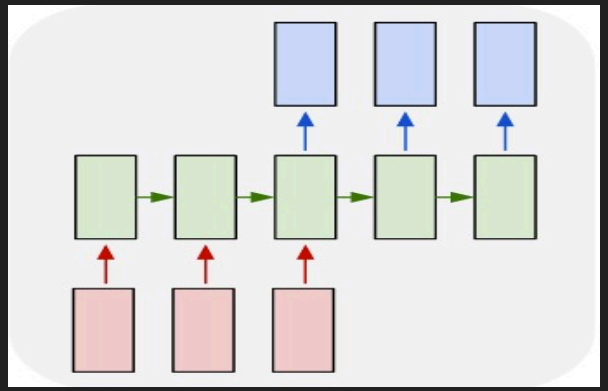
Workarounds:

- Use intermediate layer from generator as input to discriminator (not very appealing)
- Use reinforcement learning to train generator (this paper)

# Architecture

$y_t$ sampled from policy $\pi$

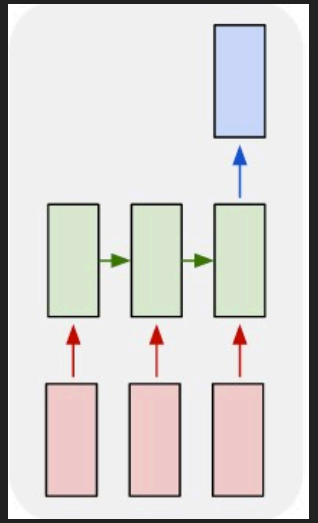
$y_1$ $y_2$ $y_T$ : Response y



Dialogue History x : $x_1$ $x_2$ $x_T$

Generator

$Q_+(\{x,y\})$



Full dialogue: (x, y)

Discriminator

# Architecture

Generator:

- Encoder-Decoder with attention (Think machine translation)
- Last two utterances in x are concatenated and fed as input

Discriminator:

- HRED model
- After feeding {x,y} as input, we get a hidden representation at the dialogue level
- This is transformed to a scalar between 0 and 1 through an MLP

# Training

Discriminator:

- Simple back propagation with SGD or any other optimizer

Generator:

- REINFORCE: $\pi$ is our policy, $Q_+(\{x, y\})$ is the return (same for each action)
- $J(\theta) = E_{y\sim\pi(y|x;\theta)}[Q_+(\{x,y\})]$ is our loss function
- As discussed before $\nabla J(\theta) \sim [Q_+(\{x, y\})] \nabla \Sigma_t \log \pi(y_t \mid x, y_{1:t-1})$
- A baseline $b(\{x,y\})$ is subtracted from Q to reduce variance

# Reward for Every Generation Step

- Till now, same reward is given to each action (that is, for each word token generated by G)

  Example:

  History: *What's your name?*
  Gold Response: *I am John*
  Machine Response: *I don't know*
  Discriminator Output for machine response: 0.1

  Same reward given for *I, don't* and *know*

# Reward for Every Generation Step

- Till now, same reward is given to each action (that is, for each word token generated by G)
- Assign rewards for partially generated sequences
- Two ways to do this:
  - Monte Carlo search
  - Train discriminator D on partial sequences

# Reward for Every Generation Step

Monte Carlo search

- For a partially decoded sequence $Y_t = y_{1:t}$, sample N responses with prefix $Y_t$.
- Discriminator judges each of these N responses.
- Average score is provided as reward for $y_t$.
- N is set to 5.

# Reward for Every Generation Step

Train D on partial sequences

- Discriminator is trained to give a score for both full and partial responses.
- Generated response/real response is broken into all partial sequences. One partial sequence is sampled and given to discriminator.
- Less time consuming than MC, but discriminator becomes weaker

# Reward for Every Generation Step

+ Dinesh
+ Barun
+ Arindam

Train D on partial sequences

- Discriminator is trained to give a score for both full and partial responses.
- Generated response/real response is broken into all partial sequences. One partial sequence is sampled and given to discriminator.
- Less time consuming than MC, but discriminator becomes weaker

# Teacher Forcing

- ‘Pretend’ that ground truth word was sampled, give this a reward of 1
- Equivalent to the standard method of training a seq2seq model, which uses maximum likelihood objective, called teacher forcing
- To make the life of the generator easier, it is periodically trained using teacher forcing
- Alternatively: Use discriminator to give score to human response, use this as reward for generator (instead of flat 1), but only if this reward is greater than baseline

# Teacher Forcing

+ Gagan
+ Arindam
+ Rishab

- ‘Pretend’ that ground truth word was sampled, give this a reward of 1
- Equivalent to the standard method of training a seq2seq model, which uses maximum likelihood objective, called teacher forcing
- To make the life of the generator easier, it is periodically trained using teacher forcing
- Alternatively: Use discriminator to give score to human response, use this as reward for generator (instead of flat 1), but only if this reward is greater than baseline

# Heuristics

- Pre-train the generator and the discriminator
- Remove responses shorter than 5 words
- Weighted learning rate that considers the average tf-idf score for tokens within the response.
- Promoting diversity in beam search by penalizing sentences with same prefix.
- Penalizing word types that have already been generated.

# Heuristics

+ Dinesh

- Pre-train the generator and the discriminator + Arindam
- Remove responses shorter than 5 words
- Weighted learning rate that considers the average tf-idf score for tokens within the response. + Rishab
- Promoting diversity in beam search by penalizing sentences with same prefix.
- Penalizing word types that have already been generated.

# Final algorithm

**For** number of training iterations **do**
. **For** i=1,D-steps **do**
. Sample (X,Y) from real data
. Sample $\hat{Y} \sim G(\cdot|X)$
. Update $D$ using $(X, Y)$ as positive examples and $(X, \hat{Y})$ as negative examples.
. **End**
.
. **For** i=1,G-steps **do**
. Sample (X,Y) from real data
. Sample $\hat{Y} \sim G(\cdot|X)$
. Compute Reward $r$ for $(X, \hat{Y})$ using $D$.
. Update $G$ on $(X, \hat{Y})$ using reward $r$
. Teacher-Forcing: Update $G$ on $(X, Y)$
. **End**
**End**

# Adversarial Evaluation

- Train a separate discriminator that can be used as an evaluator during testing
- On a test set, if discriminator gives an average score of 0.5, then machine response is indistinguishable from human response (assuming discriminator is good)
- *Adversarial Success* (or *AdverSuc*): fraction of instances in which a model is capable of fooling the evaluator.

# Is the Discriminator reliable?

- Sanity checks to test the reliability of the discriminator
  - Human-generated responses as both +ve and -ve examples: Ideal score - 0.5
  - Machine-generated responses as both +ve and -ve examples: Ideal score - 0.5
  - Human-generated responses as +ve, random responses as -ve examples: Ideal Score - 0
  - Human-generated responses as +ve, utterance following true response as -ve examples: Ideal Score - 0
- Evaluator Reliability Error: average deviation of an evaluator's adversarial error from the gold-standard error

# Is the Discriminator reliable?

+ Gagan
+ Barun

- Sanity checks to test the reliability of the discriminator
  - Human-generated responses as both +ve and -ve examples: Ideal score - 0.5
  - Machine-generated responses as both +ve and -ve examples: Ideal score - 0.5
  - Human-generated responses as +ve, random responses as -ve examples: Ideal Score - 0
  - Human-generated responses as +ve, utterance following true response as -ve examples: Ideal Score - 0
- Evaluator Reliability Error: average deviation of an evaluator's adversarial error from the gold-standard error

# Is the Discriminator reliable?

| Setting | ERE |
|---|---|
| SVM+Unigram | 0.232 |
| Concat Neural | 0.209 |
| Hierarchical Neural | 0.193 |
| SVM+Neural+multil-features | 0.152 |

Table 2: ERE scores obtained by different models.

# Machine-vs-Random Accuracy

- Adversarial Success metric not enough
- Additional check: Accuracy of distinguishing between machine-generated responses and randomly sampled human responses
- Ensures that generative model is not fooling the discriminator simply by introducing randomness

# Evaluation

- Automatic Evaluation
  - *AdverSuc*
  - *Machine vs Random*

- Human Evaluation
  - Single turn and multi-turn (3 messages)
  - Provide responses from 2 dialogue systems to 3 judges, judges choose better response (ties allowed)

# Evaluation

- Automatic Evaluation
  - *AdverSuc*
  - *Machine vs Random*

+ Dinesh
+ Arindam

- Human Evaluation
  - Single turn and multi-turn (3 messages)
  - Provide responses from 2 dialogue systems to 3 judges, judges choose better response (ties allowed)

# Automatic Evaluation Results

| Model | *AdverSuc* | *machine-vs-random* |
|---|---|---|
| MLE-BS | 0.037 | 0.942 |
| MLE-Greedy | 0.049 | 0.945 |
| MMI+$p(t\|s)$ | 0.073 | 0.953 |
| MMI-$p(t)$ | 0.090 | 0.880 |
| Sampling | 0.372 | 0.679 |
| Adver-Reinforce | 0.080 | 0.945 |
| Adver-REGS | 0.098 | 0.952 |

Table 3: *AdverSuc* and *machine-vs-random* scores achieved by different training/decoding strategies.

# Human Evaluation Results

| Setting | adver-win | adver-lose | tie |
|---|---|---|---|
| single-turn | 0.62 | 0.18 | 0.20 |
| multi-turn | 0.72 | 0.10 | 0.18 |

Table 4: The gain from the proposed adversarial model over the mutual information system based on pairwise human judgments.

# Human Evaluation Results

+ Barun

| Setting | adver-win | adver-lose | tie |
|---|---|---|---|
| single-turn | 0.62 | 0.18 | 0.20 |
| multi-turn | 0.72 | 0.10 | 0.18 |

Table 4: The gain from the proposed adversarial model over the mutual information system based on pairwise human judgments.

# Sample Responses

| | |
|---|---|
| **Input** | tell me ... how long have you had this falling sickness ? |
| **Vanilla-MLE** | i 'm not a doctor . |
| **Vanilla-Sample** | well everything you did was totally untrue . |
| **REINFORCE** | i don 't know how long it 's been . |
| **REGS Monte Carlo** | A few months, I guess . |
| **Input** | so i had the doctors test sammy 's response to conditioning . |
| **Vanilla-MLE** | sammy wrote the test sammy wrote the test . |
| **Vanilla-Sample** | objects pick objects up objects objects objects objects objects |
| **REINFORCE** | i 'm not sure that 's relevant . |
| **REGS Monte Carlo** | so he took the pills . |
| **Input** | what are you going to do ? |
| **Vanilla-MLE** | i 'm going to the movies . |
| **Vanilla-Sample** | get him outta here first ! |
| **REINFORCE** | i 'm going to get you . |
| **REGS Monte Carlo** | i 'm going to get you a new car . |
| **Input** | they fear your power your intellect . |
| **Vanilla-MLE** | you 're the only one who knows what 's going on . |
| **Vanilla-Sample** | when they are conquered and you surrender they will control all of us . |
| **REINFORCE** | i 'm afraid i 'm not ready yet . |
| **REGS Monte Carlo** | i 'm not afraid of your power . |

# Key Takeaways

- GANs can be trained for NLP tasks using policy gradient methods.
- GANs + teacher forcing significantly outperforms the best teacher forcing model for dialogue implying this is a viable and helpful model
- Rewards for partial sequences using MC search
- Four useful heuristics to make model responses more coherent and less generic
- Generator training is unstable — this is a hot topic of research in the vision space, ideas that emerge there could be used in NLP space as well
- Adversarial evaluation is an interesting automatic evaluation metric — but its effectiveness needs to be studied carefully

# Extensions

- Gagan: Active Learning
- Barun: Weighted score, using both the discriminator score and a language model score, may help recognizing grammatically incoherent sentences
- Arindam: Half and half approach of pre training could be converted into a better graduated method, where the negative examples get gradually more difficult
- Arindam: The heuristics used for the generator, use them for the discriminator in some way, like generating negative training examples that violate these rules
- Arindam: Bidirectional LSTM

# Extensions

- Rishab: Wasserstein GAN for more stable training
- Rishab: Use GAN discriminator as pretrained model for evaluator
- Rishab: Model could be tried out for QA
- Haroun: Adversarially train the discriminator?
- Haroun: Check what tells the evaluator learns, whether it is similar to a human evaluator
- Anshul: Can further formalize the 4 strategies/heuristics
- Anshul: Deeper discriminator
- Prachi: Trained discriminator can additionally be made to predict sentiment of an utterance. This will work in situations when we have limited labelled data.