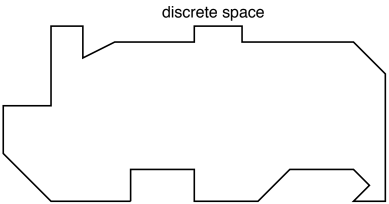
# Plug and Play Autoencoders for Conditional Text Generation

Florian Mai [†,♠] Nikolaos Pappas [♣]
Ivan Montero [♣] Noah A. Smith [♣,♢] James Henderson [†]

[†]Idiap Research Institute, [♠]EPFL, Switzerland
[♣] University of Washington, Seattle, USA
[♢] Allen Institute for Artificial Intelligence, Seattle, USA

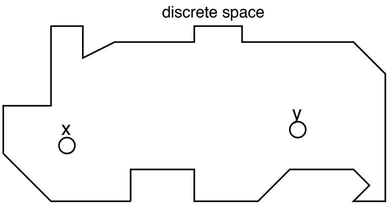
florian.mai@idiap.ch

# The Problem with Conditional Text Generation



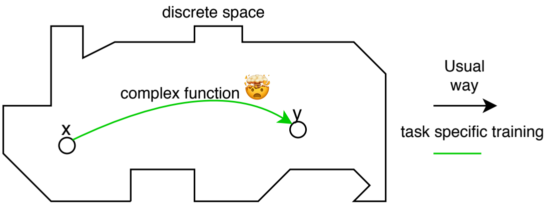

- text lives in a messy, discrete space

# The Problem with Conditional Text Generation




- text lives in a messy, discrete space
- conditional text generation requires mapping from discrete input to discrete output

# The Problem with Conditional Text Generation

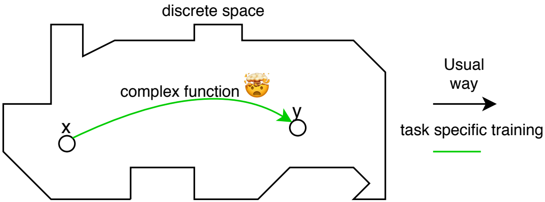



- text lives in a messy, discrete space
- conditional text generation requires mapping from discrete input to discrete output

**Usual way:**

- learning a complex, task-specific function, which is difficult to train in discrete space

# The Problem with Conditional Text Generation



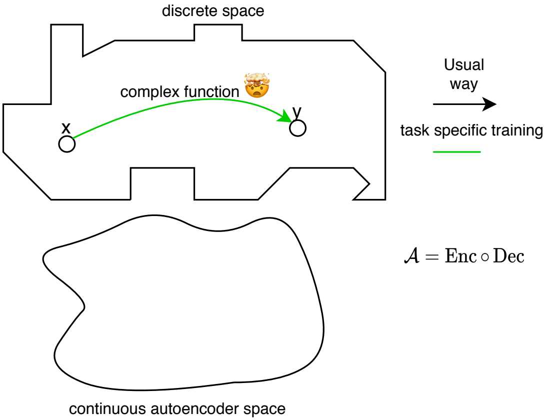

- text lives in a messy, discrete space
- conditional text generation requires mapping from discrete input to discrete output

**Usual way:**

- learning a complex, task-specific function, which is difficult to train in discrete space

**Our way:**

# The Problem with Conditional Text Generation




- text lives in a messy, discrete space
- conditional text generation requires mapping from discrete input to discrete output

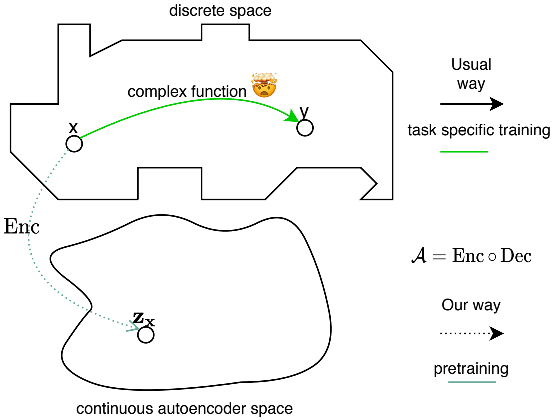
**Usual way:**

- learning a complex, task-specific function, which is difficult to train in discrete space

**Our way:**

- obtain a continuous space by training an autoencoder

# The Problem with Conditional Text Generation




- text lives in a messy, discrete space
- conditional text generation requires mapping from discrete input to discrete output

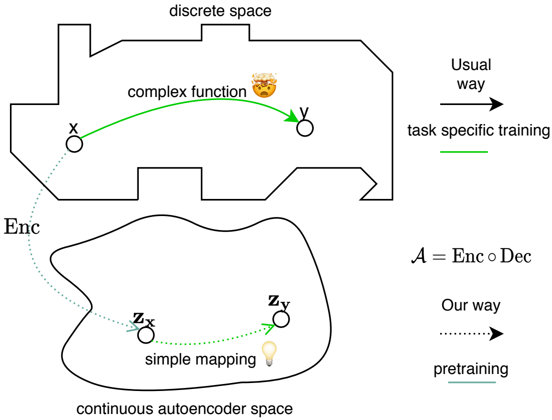
**Usual way:**

- learning a complex, task-specific function, which is difficult to train in discrete space

**Our way**:

- obtain a continuous space by training an autoencoder

# The Problem with Conditional Text Generation




- text lives in a messy, discrete space
- conditional text generation requires mapping from discrete input to discrete output

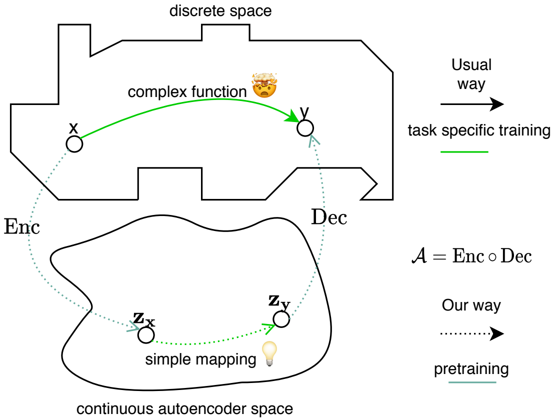
**Usual way:**

- learning a complex, task-specific function, which is difficult to train in discrete space

**Our way:**

- obtain a continuous space by training an autoencoder
- reduce task-specific learning to the continuous space

# The Problem with Conditional Text Generation




- text lives in a messy, discrete space
- conditional text generation requires mapping from discrete input to discrete output

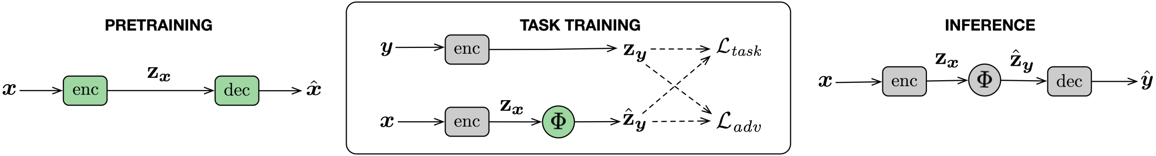
**Usual way:**

- learning a complex, task-specific function, which is difficult to train in discrete space

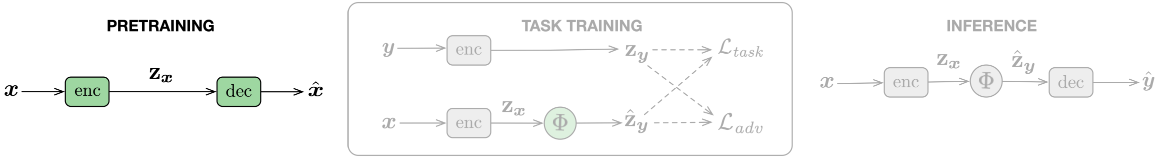
**Our way**:

- obtain a continuous space by training an autoencoder
- reduce task-specific learning to the continuous space

Our framework (*Emb2Emb*) consists of three stages:



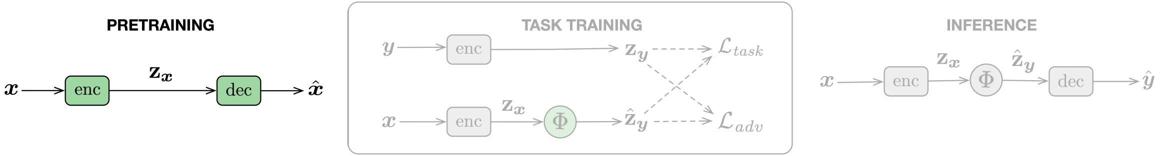
# Framework Overview




**Pretraining**:

- Train a model of the form $\mathcal{A}(x) = \text{Dec}(\text{Enc}(x))$ on corpus of sentences
- Assume a fixed-size continuous embedding $\mathbf{z_x} := \text{Enc}(x) \in \mathbb{R}^d$
- Enc and Dec can be any function trained with any objective so long as $\mathcal{A}(x) \approx x$
- training corpus can be any unlabeled corpus $\Rightarrow$ large-scale pretraining?

# Framework Overview

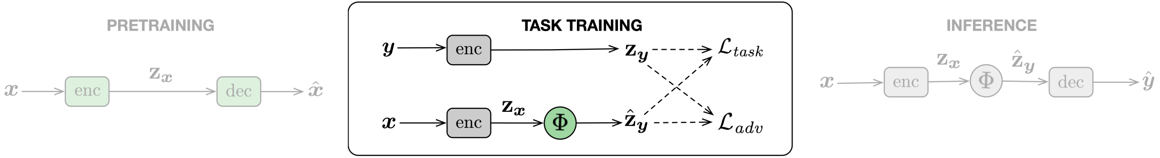



**Pretraining**:

- Train a model of the form $\mathcal{A}(x) = \mathrm{Dec}(\mathrm{Enc}(x))$ on corpus of sentences
- Assume a fixed-size continuous embedding $\mathbf{z_x} := \mathrm{Enc}(x) \in \mathbb{R}^d$
- Enc and Dec can be any function trained with any objective so long as $\mathcal{A}(x) \approx x$
- training corpus can be any unlabeled corpus $\Rightarrow$ large-scale pretraining?

**Plug and Play**

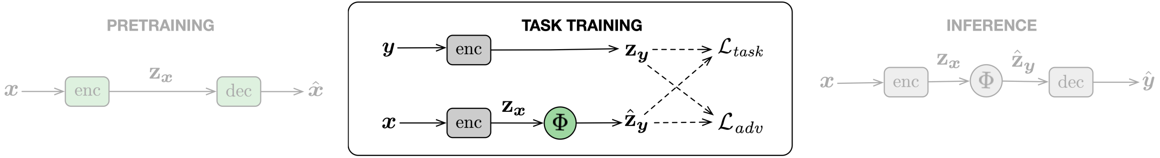
Our framework is *plug and play* because any autoencoder can be used with it.




**Task Training**:

- Supervised case: $\mathcal{L}_{task}(\hat{\mathbf{z}}_{\mathbf{y}}, \mathbf{z}_{\mathbf{y}}) = d(\hat{\mathbf{z}}_{\mathbf{y}}, \mathbf{z}_{\mathbf{y}})$ where $d$ is a distance function (cosine distance loss in our experiments).

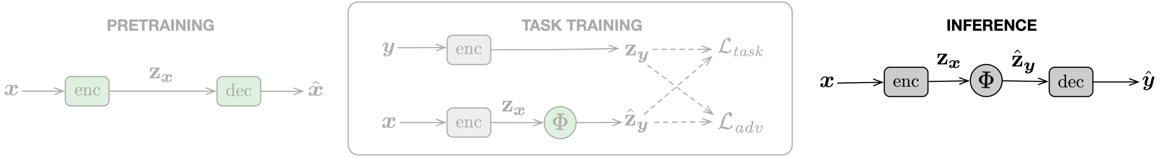
# Framework Overview




**Task Training**:

- Supervised case: $\mathcal{L}_{task}(\hat{\mathbf{z}}_\mathbf{y}, \mathbf{z}_\mathbf{y}) = d(\hat{\mathbf{z}}_\mathbf{y}, \mathbf{z}_\mathbf{y})$ where $d$ is a distance function (cosine distance loss in our experiments).
- Training objective: $\mathcal{L} = \mathcal{L}_{task} + \lambda_{adv} \cdot \overset{?}{\boxed{\mathcal{L}_{adv}}}$

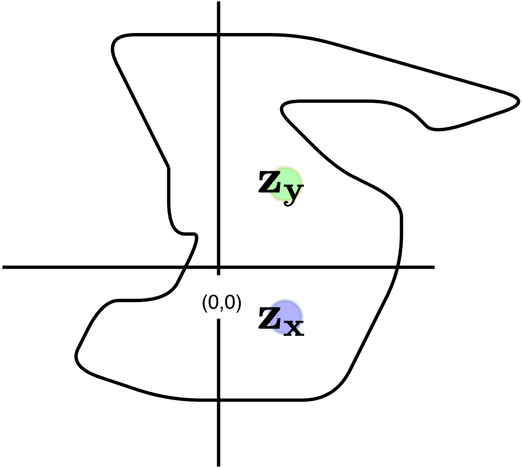
# Framework Overview

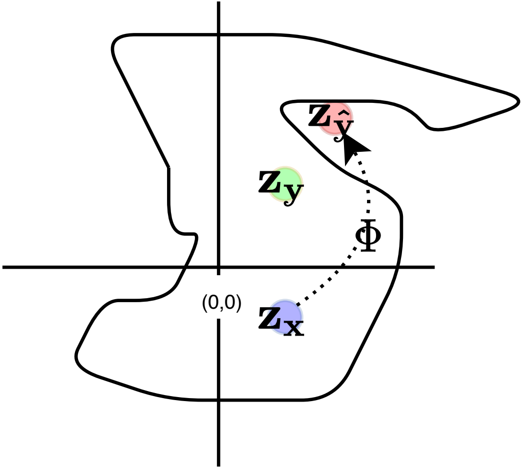



**Inference**:

- compose inference model as Enc $\circ \Phi \circ$ Dec
- but: Dec not involved in training. Can it handle outputs of $\Phi$?
- $\Rightarrow$ yes, if using $\mathcal{L}_{adv}$.

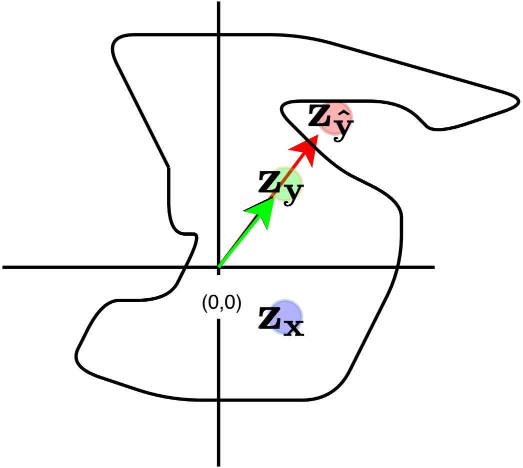
# What can happen when learning in the embedding space?





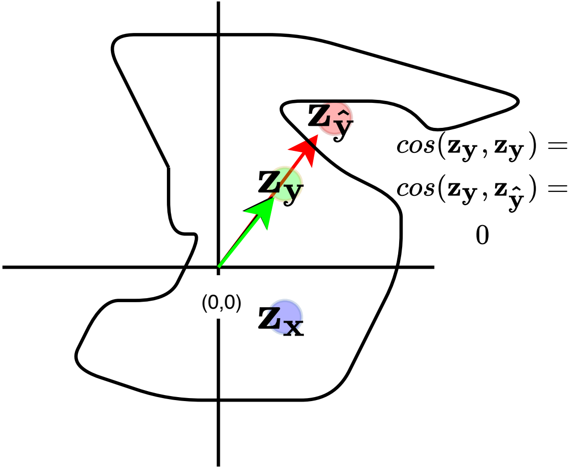

- A prediction may end up **off** the manifold, and by definition, the decoder cannot handle off-manifold data well, but ...

# WHAT CAN HAPPEN WHEN LEARNING IN THE EMBEDDING SPACE?



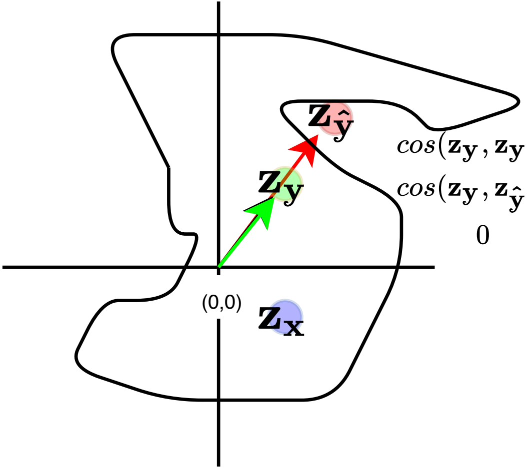

- A prediction may end up **off** the manifold, and by definition, the decoder cannot handle off-manifold data well, but ...
- ... but the predicted embedding may still have the same angle as the true output embedding...

# WHAT CAN HAPPEN WHEN LEARNING IN THE EMBEDDING SPACE?




- A prediction may end up **off** the manifold, and by definition, the decoder cannot handle off-manifold data well, but ...
- ... but the predicted embedding may still have the same angle as the true output embedding...
- resulting in zero cosine distance loss despite being off the manifold.

# What can happen when learning in the embedding space?




$$cos(\mathbf{z}_{\mathbf{y}}, \mathbf{z}_{\mathbf{y}}) = cos(\mathbf{z}_{\mathbf{y}}, \mathbf{z}_{\hat{\mathbf{y}}}) = 0$$

- A prediction may end up **off** the manifold, and by definition, the decoder cannot handle off-manifold data well, but ...
- ... but the predicted embedding may still have the same angle as the true output embedding...
- resulting in zero cosine distance loss despite being off the manifold.
- Similar problems arise for L2 distance - how do we keep the embeddings **on** the manifold?

# Adversarial Loss Term

- train a discriminator $\mathrm{disc}$ to distinguish between embeddings produced by the encoder and embeddings resulting from the mapping:

$$\max_{\mathrm{disc}} \sum_{i=1}^{N} \log(\mathrm{disc}(\mathbf{z}_{\tilde{\mathbf{y}}_i})) + \log(1 - \mathrm{disc}(\Phi(\mathbf{z}_{\mathbf{x}_i}))$$

- using the adversarial learning framework, mapping acts as the adversary and tries to fool the discriminator:

$$\mathcal{L}_{adv}(\Phi(\mathbf{z}_{\mathbf{x}_i}); \boldsymbol{\theta}) = -\log(\mathrm{disc}(\Phi(\mathbf{z}_{\mathbf{x}_i}); \theta))$$

- at convergence, the mapping should only produce embeddings that are on the manifold

## Supervised Style Transfer Experiments

- *WikiLarge* dataset: transform “normal“ English to “simple“ English
- parallel sentences (input and output) are available

| **Model** | **BLEU** (relative imp.) | **SARI** (relative imp.) |
|---|---|---|
| Emb2Emb (no $\mathcal{L}_{adv}$) | 15.7 (-) | 21.1 (-) |
| Emb2Emb | **34.7** (+121%) | **25.4** (+20.4%) |

The adversarial loss term $\mathcal{L}_{adv}$ is crucial for embedding-to-embedding training!

# Supervised Style Transfer Experiments

- we conducted controlled experiments of models **with a fixed-size bottleneck**
- best Seq2Seq model: best performing variant among fixed-size bottleneck models that are trained end-to-end via token-level cross-entropy loss (like Seq2Seq)

| **Model** | **BLEU** (relative imp.) | **SARI** (relative imp.) | **Speedup** |
|---|---|---|---|
| Best Seq2Seq model | 23.3 ($\pm$0%) | 22.4 ($\pm$0%) | - |
| Emb2Emb | **34.7** (+48.9%) | **25.4** (+13.4%) | 2.2$\times$ |

Training models with a fixed-size bottleneck may be *easier, faster*, and *more effective* when training embedding-to-embedding!

# Unsupervised Task Training

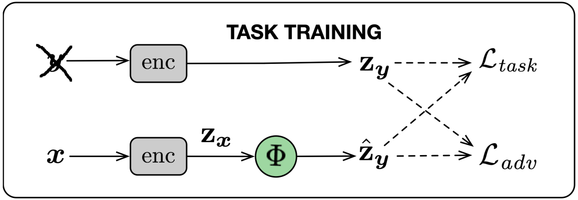
- Fixed-size bottleneck autoencoders are commonly used for unsupervised style transfer

# Unsupervised Task Training

- Fixed-size bottleneck autoencoders are commonly used for unsupervised style transfer
- The goal is to change the **sty**le of a text, but retain the **cont**ent: e.g., in machine translation, sentence simplification, sentiment transfer

# Unsupervised Task Training

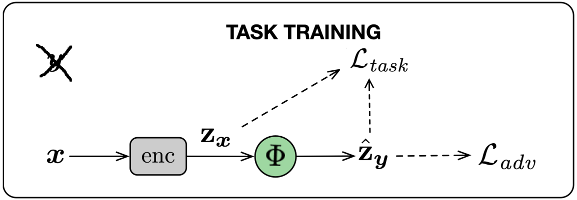
- Fixed-size bottleneck autoencoders are commonly used for unsupervised style transfer
- The goal is to change the **sty**le of a text, but retain the **cont**ent: e.g., in machine translation, sentence simplification, sentiment transfer




- training objective: $\mathcal{L} = \mathcal{L}_{task} + \lambda_{adv} \cdot \mathcal{L}_{adv}$

# Unsupervised Task Training

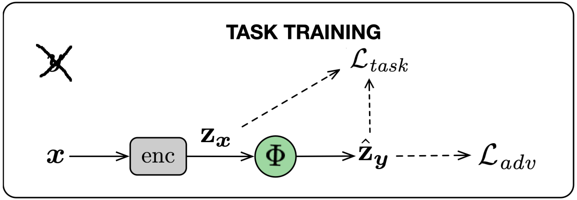
- Fixed-size bottleneck autoencoders are commonly used for unsupervised style transfer
- The goal is to change the **sty**le of a text, but retain the **cont**ent: e.g., in machine translation, sentence simplification, sentiment transfer




- training objective: $\mathcal{L} = \mathcal{L}_{task} + \lambda_{adv} \cdot \mathcal{L}_{adv}$
- $\mathcal{L}_{task}(\hat{\mathbf{z}}_{\mathbf{y}}, \mathbf{z}_{\mathbf{x}}) = \lambda_{sty}\mathcal{L}_{sty}(\hat{\mathbf{z}}_{\mathbf{y}}) + (1 - \lambda_{sty})\mathcal{L}_{cont}(\hat{\mathbf{z}}_{\mathbf{y}}, \mathbf{z}_{\mathbf{x}})$

# Unsupervised Task Training

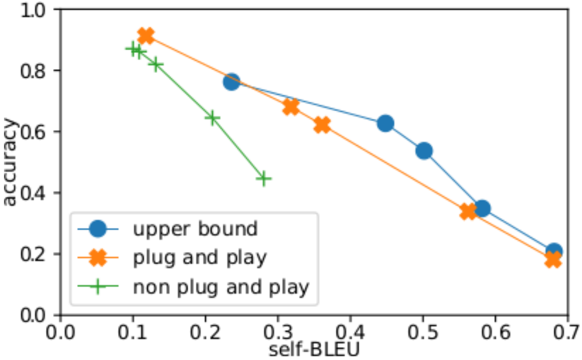
- Fixed-size bottleneck autoencoders are commonly used for unsupervised style transfer
- The goal is to change the **sty**le of a text, but retain the **cont**ent: e.g., in machine translation, sentence simplification, sentiment transfer




- training objective: $\mathcal{L} = \mathcal{L}_{task} + \lambda_{adv} \cdot \mathcal{L}_{adv}$
- $\mathcal{L}_{task}(\hat{\mathbf{z}}_{\mathbf{y}}, \mathbf{z}_{\mathbf{x}}) = \lambda_{sty}\mathcal{L}_{sty}(\hat{\mathbf{z}}_{\mathbf{y}}) + (1 - \lambda_{sty})\mathcal{L}_{cont}(\hat{\mathbf{z}}_{\mathbf{y}}, \mathbf{z}_{\mathbf{x}})$
- we set $\mathcal{L}_{cont}$ to cosine distance, and $\mathcal{L}_{sty}$ to a style classifier's negative log-likelihood of the target class

# Unsupervised Style Transfer Experiments

- *Yelp* sentiment transfer dataset: transform reviews with negative sentiment into reviews with positive sentiment (accuracy), but retain content (self-BLEU)
- if we have labels for only 10% of the data, how much better is a plug and play model?




**Effect of pretraining**

By leveraging autoencoder pretraining on unlabeled data, our plug and play method offers a distinct advantage on unsupervised style transfer!

# Conclusion

In this talk...

# CONCLUSION

In this talk...

- we propose to learn in the embedding space of a pretrained autoencoder, training embedding-to-embedding (Emb2Emb).

# Conclusion

In this talk...

- we propose to learn in the embedding space of a pretrained autoencoder, training embedding-to-embedding (Emb2Emb).
- we discuss why it's important to keep the predicted embedding on the manifold of the autoencoder, and how to achieve that.

# Conclusion

In this talk...

- we propose to learn in the embedding space of a pretrained autoencoder, training embedding-to-embedding (Emb2Emb).
- we discuss why it's important to keep the predicted embedding on the manifold of the autoencoder, and how to achieve that.
- we demonstrate that a plug and play method like ours has a distinct advantage on unsupervised style transfer.

# Conclusion

In this talk...

- we propose to learn in the embedding space of a pretrained autoencoder, training embedding-to-embedding (Emb2Emb).
- we discuss why it's important to keep the predicted embedding on the manifold of the autoencoder, and how to achieve that.
- we demonstrate that a plug and play method like ours has a distinct advantage on unsupervised style transfer.

**Additionally**, our paper...

- presents an architecture for the mapping $\Phi$ that is better than just MLPs.

# Conclusion

In this talk...

- we propose to learn in the embedding space of a pretrained autoencoder, training embedding-to-embedding (Emb2Emb).
- we discuss why it's important to keep the predicted embedding on the manifold of the autoencoder, and how to achieve that.
- we demonstrate that a plug and play method like ours has a distinct advantage on unsupervised style transfer.

**Additionally**, our paper...

- presents an architecture for the mapping $\Phi$ that is better than just MLPs.
- demonstrates how to further improve the performance on unsupervised style transfer at inference time.

*THANK YOU*