Autoencoders and Representation Learning

Deep Learning Decal

Hosted by Machine Learning at Berkeley

Agenda

Background

Review: Typical Neural Net Characteristics

So far, Deep Learning Models have things in common:

Review: Typical Neural Net Characteristics

So far, Deep Learning Models have things in common:

- Input Layer: (maybe vectorized), quantitative representation

Review: Typical Neural Net Characteristics

So far, Deep Learning Models have things in common:

- Input Layer: (maybe vectorized), quantitative representation
- Hidden Layer(s): Apply transformations with nonlinearity

Review: Typical Neural Net Characteristics

So far, Deep Learning Models have things in common:

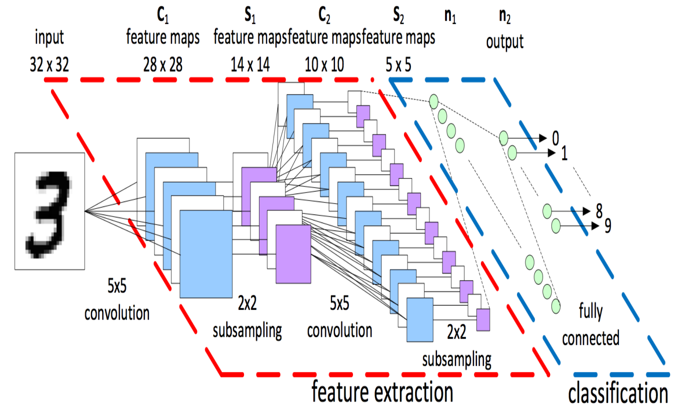
- Input Layer: (maybe vectorized), quantitative representation
- Hidden Layer(s): Apply transformations with nonlinearity
- Output Layer: Result for *classification*, *regression*, *translation*, *segmentation*, *etc.*

Review: Typical Neural Net Characteristics

So far, Deep Learning Models have things in common:

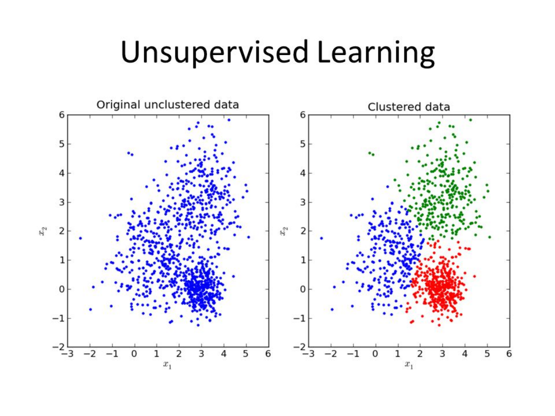
- Input Layer: (maybe vectorized), quantitative representation
- Hidden Layer(s): Apply transformations with nonlinearity
- Output Layer: Result for *classification*, *regression*, *translation*, *segmentation*, *etc.*
- Models used for **supervised learning**

Example Through Diagram



Changing the Objective

Today's lecture: **unsupervised learning** with neural networks.



Autoencoders

Autoencoders: Definition

Autoencoders are neural networks that are trained to *copy their inputs to their outputs.*

Autoencoders: Definition

Autoencoders are neural networks that are trained to *copy their inputs to their outputs.*

- Usually constrained in particular ways to make this task more difficult.

Autoencoders: Definition

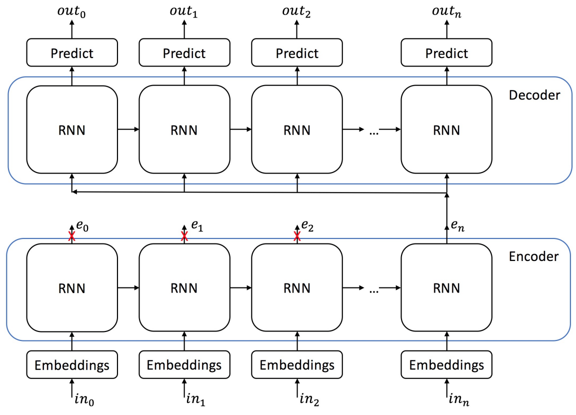
Autoencoders are neural networks that are trained to *copy their inputs to their outputs.*

- Usually constrained in particular ways to make this task more difficult.
- Structure is almost always organized into **encoder** network, $\mathbf{f}$, and **decoder** network, $\mathbf{g}$: *model* $= \mathbf{g}(\mathbf{f}(\mathbf{x}))$

Autoencoders are neural networks that are trained to *copy their inputs to their outputs.*

- Usually constrained in particular ways to make this task more difficult.
- Structure is almost always organized into **encoder** network, $\mathbf{f}$, and **decoder** network, $\mathbf{g}$: $\textit{model} = \mathbf{g}(\mathbf{f}(\mathbf{x}))$
- Trained by gradient descent with **reconstruction loss:** measures differences between input and output e.g. MSE : $J(\theta) = |\mathbf{g}(\mathbf{f}(\mathbf{x})) - \mathbf{x}|^2$

Not an Entirely New Idea



Undercomplete Autoeconders are defined to have a hidden layer **h**, with smaller dimension than input layer.

Undercomplete Autoencoders

Undercomplete Autoeconders are defined to have a hidden layer $\mathbf{h}$, with smaller dimension than input layer.

- Network must model $\mathbf{x}$ in lower dim. space + map latent space accurately back to input space.

Undercomplete Autoencoders

Undercomplete Autoeconders are defined to have a hidden layer **h**, with smaller dimension than input layer.

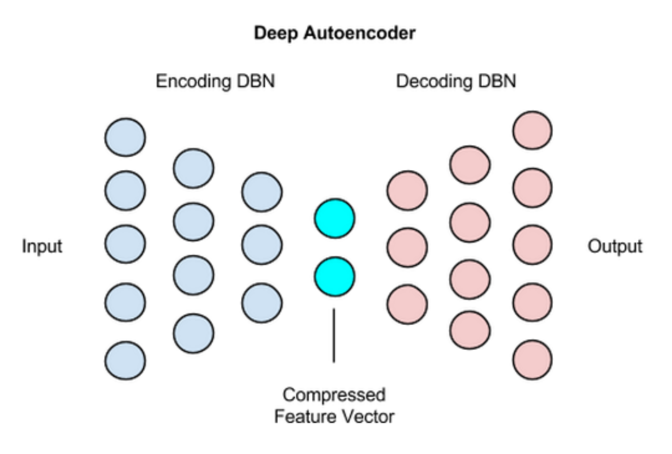
- Network must model **x** in lower dim. space + map latent space accurately back to input space.
- Encoder network: function that returns a useful, compressed representation of input.

Undercomplete Autoencoders

Undercomplete Autoeconders are defined to have a hidden layer **h**, with smaller dimension than input layer.

- Network must model **x** in lower dim. space + map latent space accurately back to input space.
- Encoder network: function that returns a useful, compressed representation of input.
- If network has only linear transformations, encoder learns PCA. With typical nonlinearities, network learns generalized, more powerful version of PCA.

Visualizing Undercomplete Autoencoders



Caveats and Dangers

Unless careful, autoencoders *will not* learn meaningful representations.

Caveats and Dangers

Unless careful, autoencoders *will not* learn meaningful representations.

- Reconstruction loss: **indifferent to latent space** characteristics. (not true for PCA).

Caveats and Dangers

Unless careful, autoencoders *will not* learn meaningful representations.

- Reconstruction loss: **indifferent to latent space** characteristics. (not true for PCA).
- Higher representational power gives flexibility for suboptimal encodings.

Caveats and Dangers

Unless careful, autoencoders *will not* learn meaningful representations.

- Reconstruction loss: **indifferent to latent space** characteristics. (not true for PCA).
- Higher representational power gives flexibility for suboptimal encodings.
- Pathological case: hidden layer is only one dimension, learns index mappings: $x^{(i)} \to i \to x^{(i)}$

Caveats and Dangers

Unless careful, autoencoders *will not* learn meaningful representations.

- Reconstruction loss: **indifferent to latent space** characteristics. (not true for PCA).
- Higher representational power gives flexibility for suboptimal encodings.
- Pathological case: hidden layer is only one dimension, learns index mappings: $x^{(i)} \to i \to x^{(i)}$
    - Not very realistic, but completely plausible.

How Constraints Correspond to Effective Manifold Learning

ML@B

We need to impose additional constraints besides reconstruction loss to learn **manifolds**.

How Constraints Correspond to Effective Manifold Learning

We need to impose additional constraints besides reconstruction loss to learn **manifolds**.

- Data manifold → *concentrated high probability* of being in training set.

How Constraints Correspond to Effective Manifold Learning

We need to impose additional constraints besides reconstruction loss to learn **manifolds**.

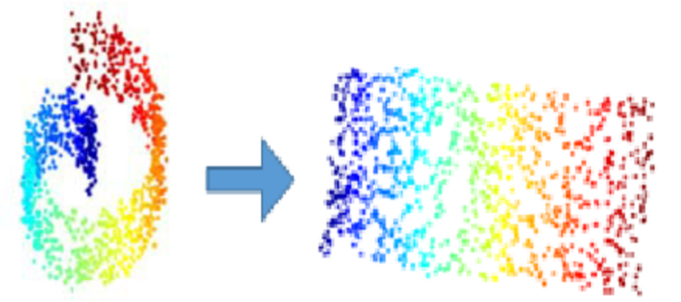
- Data manifold → *concentrated high probability* of being in training set.
- Constraining complexity or imposing regularization promotes learning a more defined "surface" and the variations that shape manifold.

How Constraints Correspond to Effective Manifold Learning

We need to impose additional constraints besides reconstruction loss to learn **manifolds**.

- Data manifold → *concentrated high probability* of being in training set.
- Constraining complexity or imposing regularization promotes learning a more defined "surface" and the variations that shape manifold.
- → Autoencoders should only learn necessary variations to reconstruct training examples.

Extract 2D manifold of data which exists in 3D:



Regularized Autoencoders

Stochastic Autoencoders

Rethink the underlying idea of autoencoders. Instead of encoding/decoding **functions**, we can see them as describing encoding/decoding **probability distributions** like so:

$$p_{encoder}(\mathbf{h}|\mathbf{x}) = p_{model}(\mathbf{h}|\mathbf{x})$$

$$p_{decoder}(\mathbf{x}|\mathbf{h}) = p_{model}(\mathbf{x}|\mathbf{h})$$

These distributions are called **stochastic** encoders and decoders respectively.

Distribution View of Autoencoders

Consider stochastic decoder $\mathbf{g}(\mathbf{h})$ as a **generative model** and its relationship to the joint distribution

Distribution View of Autoencoders

Consider stochastic decoder $\mathbf{g}(\mathbf{h})$ as a **generative model** and its relationship to the joint distribution

$$p_{model}(\mathbf{x}, \mathbf{h}) = p_{model}(\mathbf{h}) \cdot p_{model}(\mathbf{x}|\mathbf{h})$$

$$\ln p_{model}(\mathbf{x}, \mathbf{h}) = \ln p_{model}(\mathbf{h}) + \ln p_{model}(\mathbf{x}|\mathbf{h})$$

Distribution View of Autoencoders

Consider stochastic decoder $\mathbf{g}(\mathbf{h})$ as a **generative model** and its relationship to the joint distribution

$$p_{model}(\mathbf{x}, \mathbf{h}) = p_{model}(\mathbf{h}) \cdot p_{model}(\mathbf{x}|\mathbf{h})$$

$$\ln p_{model}(\mathbf{x}, \mathbf{h}) = \ln p_{model}(\mathbf{h}) + \ln p_{model}(\mathbf{x}|\mathbf{h})$$

- If $\mathbf{h}$ is given from encoding network, then we want most likely $\mathbf{x}$ to output.

Distribution View of Autoencoders

Consider stochastic decoder $\mathbf{g}(\mathbf{h})$ as a **generative model** and its relationship to the joint distribution

$$p_{model}(\mathbf{x}, \mathbf{h}) = p_{model}(\mathbf{h}) \cdot p_{model}(\mathbf{x}|\mathbf{h})$$

$$\ln p_{model}(\mathbf{x}, \mathbf{h}) = \ln p_{model}(\mathbf{h}) + \ln p_{model}(\mathbf{x}|\mathbf{h})$$

- If $\mathbf{h}$ is given from encoding network, then we want most likely $\mathbf{x}$ to output.
- Finding MLE of $\mathbf{x}, \mathbf{h} \approx$ maximizing $p_{model}(\mathbf{x}, \mathbf{h})$

Distribution View of Autoencoders

Consider stochastic decoder $\mathbf{g}(\mathbf{h})$ as a **generative model** and its relationship to the joint distribution

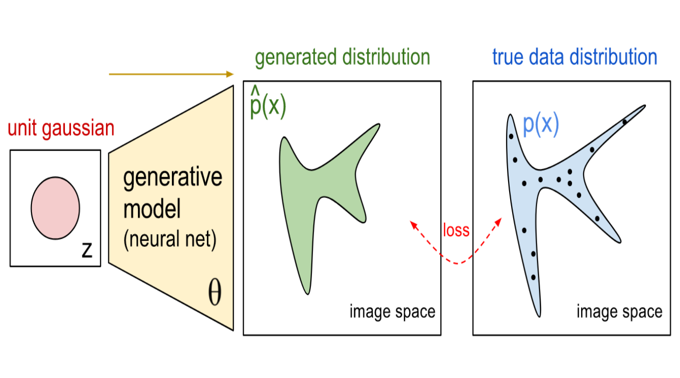
$$p_{model}(\mathbf{x}, \mathbf{h}) = p_{model}(\mathbf{h}) \cdot p_{model}(\mathbf{x}|\mathbf{h})$$

$$\ln p_{model}(\mathbf{x}, \mathbf{h}) = \ln p_{model}(\mathbf{h}) + \ln p_{model}(\mathbf{x}|\mathbf{h})$$

- If $\mathbf{h}$ is given from encoding network, then we want most likely $\mathbf{x}$ to output.
- Finding MLE of $\mathbf{x}, \mathbf{h} \approx$ maximizing $p_{model}(\mathbf{x}, \mathbf{h})$
- $p_{model}(\mathbf{h})$ is prior across latent space values. **This term can be regularizing.**

Meaning of Generative

By assuming a prior over latent space, can pick values from underlying probability distribution!



Sparse Autoencoders

Sparse Autoencoders have modified loss function with sparsity penalty on latent variables: $J(\theta) = L(x, g(f(x)) + \Omega(\mathbf{h})$

Sparse Autoencoders have modified loss function with sparsity penalty on latent variables: $J(\theta) = L(x, g(f(x)) + \Omega(\mathbf{h})$

- L1 reg as example: Assume Laplacian prior on latent space vars:

$$p_{model}(h_i) = \frac{\lambda}{2} e^{-\lambda|h_i|}$$

Sparse Autoencoders have modified loss function with sparsity penalty on latent variables: $J(\theta) = L(x, g(f(x)) + \Omega(\mathbf{h})$

- L1 reg as example: Assume Laplacian prior on latent space vars:

$$p_{model}(h_i) = \frac{\lambda}{2} e^{-\lambda |h_i|}$$

The log likelihood becomes:

$$-\ln p_{model}(\mathbf{h}) = \lambda \sum_i |h_i| + const. = \Omega(\mathbf{h})$$

Idea: Allocate space for storing parameters of probability distribution.

Idea: Allocate space for storing parameters of probability distribution.

- Latent space variables for **mean, std dev** of distribution

ML@B

Idea: Allocate space for storing parameters of probability distribution.

- Latent space variables for **mean, std dev** of distribution
- **Flow:** Input → encode to statistics vectors → *sample* a latent vector → decode for reconstruction

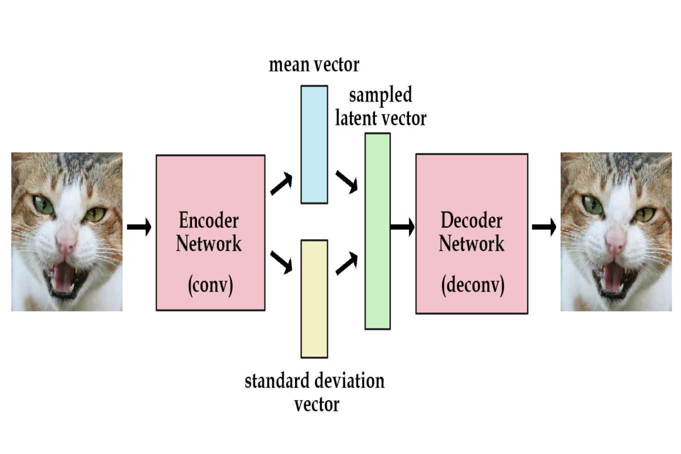
Variational Autoencoders

Idea: Allocate space for storing parameters of probability distribution.

- Latent space variables for **mean, std dev** of distribution
- **Flow:** Input → encode to statistics vectors → *sample* a latent vector → decode for reconstruction
- Loss: Reconstruction + K-L Divergence

Visualizing Variational Autoencoders

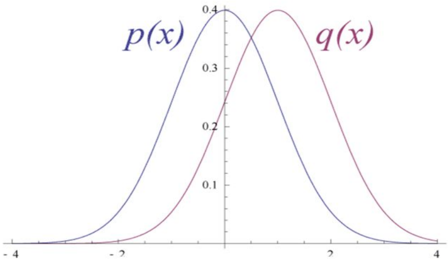
Latent space explicitly encodes distribution statistics! Typically made to encode unit gaussian.



K-L Divergence

Variational Autoencoder Loss also needs K-L divergence. Measures difference between distributions

K-L Divergence



$$D_{\mathrm{KL}}(P\|Q) = \int_{-\infty}^{\infty} p(x)\,\log\frac{p(x)}{q(x)}\,\mathrm{d}x$$

Taken from Wikipedia page 'Kullback-Liebler Divergence

Denoising Autoencoders

Sparse autoencoders motivated by particular purpose (generative modeling). Denoising autoencoders are useful for... denoising.

Denoising Autoencoders

Sparse autoencoders motivated by particular purpose (generative modeling). Denoising autoencoders are useful for... denoising.

- For every input $\mathbf{x}$, we apply corrupting function $C(\cdot)$ to create noisy version: $\tilde{\mathbf{x}} = C(\mathbf{x})$.

Denoising Autoencoders

Sparse autoencoders motivated by particular purpose (generative modeling). Denoising autoencoders are useful for... denoising.

- For every input $\mathbf{x}$, we apply corrupting function $C(\cdot)$ to create noisy version: $\tilde{\mathbf{x}} = C(\mathbf{x})$.
- Loss function changes: $J(\mathbf{x}, \mathbf{g}(\mathbf{f}(\mathbf{x}))) \rightarrow J(\mathbf{x}, \mathbf{g}(\mathbf{f}(\tilde{\mathbf{x}})))$.

Denoising Autoencoders

Sparse autoencoders motivated by particular purpose (generative modeling). Denoising autoencoders are useful for... denoising.

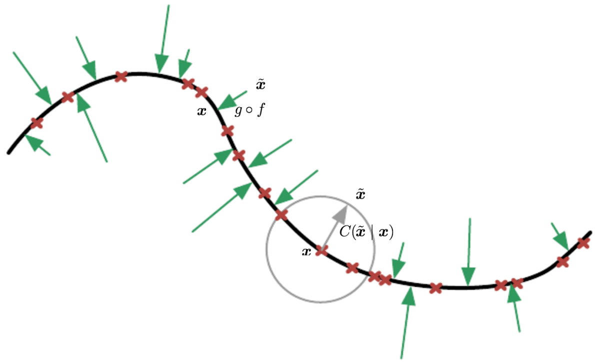
- For every input $\mathbf{x}$, we apply corrupting function $C(\cdot)$ to create noisy version: $\tilde{\mathbf{x}} = C(\mathbf{x})$.
- Loss function changes: $J(\mathbf{x}, \mathbf{g}(\mathbf{f}(\mathbf{x}))) \rightarrow J(\mathbf{x}, \mathbf{g}(\mathbf{f}(\tilde{\mathbf{x}})))$.
- $\mathbf{f}, \mathbf{g}$ will necessarily learn $p_{data}(\mathbf{x})$ because learning identity function will not give good loss.

Visualizing Denoising Autoencoders

By having to remove noise, model must know difference between noise and actual image.

Visualizing Denoising Autoencoders

The corrupting function $C(\cdot)$ can corrupt in any direction $\rightarrow$ autoencoder must learn "location" of data manifold and its distribution $p_{data}(\mathbf{x})$.



Contractive Autoencoders

Contractive Autoencoders are explicitly encouraged to learn a manifold through their loss function.

Desirable property: Points close to each other in input space maintain that property in the latent space.

Contractive Autoencoders are explicitly encouraged to learn a manifold through their loss function.

Desirable property: Points close to each other in input space maintain that property in the latent space.

- This will be true if $\mathbf{f}(\mathbf{x}) = \mathbf{h}$ is continuous, has small derivatives.

Contractive Autoencoders

Contractive Autoencoders are explicitly encouraged to learn a manifold through their loss function.

Desirable property: Points close to each other in input space maintain that property in the latent space.

- This will be true if $\mathbf{f}(\mathbf{x}) = \mathbf{h}$ is continuous, has small derivatives.
- We can use the **Frobenius Norm** of the **Jacobian Matrix** as a regularization term:

Contractive Autoencoders

Contractive Autoencoders are explicitly encouraged to learn a manifold through their loss function.

Desirable property: Points close to each other in input space maintain that property in the latent space.

- This will be true if $\mathbf{f}(\mathbf{x}) = \mathbf{h}$ is continuous, has small derivatives.
- We can use the **Frobenius Norm** of the **Jacobian Matrix** as a regularization term:

Contractive Autoencoders are explicitly encouraged to learn a manifold through their loss function.
Desirable property: Points close to each other in input space maintain that property in the latent space.

- This will be true if $\mathbf{f}(\mathbf{x}) = \mathbf{h}$ is continuous, has small derivatives.
- We can use the **Frobenius Norm** of the **Jacobian Matrix** as a regularization term:

$$\Omega(\mathbf{f}, \mathbf{x}) = \lambda \left| \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} \right|_F^2$$

Jacobian and Frobenius Norm

The Jacobian Matrix for vector-valued function $f(x)$:

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \dots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \dots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

Jacobian and Frobenius Norm

The Jacobian Matrix for vector-valued function $f(x)$:

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \dots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \dots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

The Frobenius Norm for a matrix M:

$$||M||_F = \sqrt{\sum_{i,j} M_{ij}^2}$$

Called contractive because they *contract* neighborhood of input space into *smaller*, *localized* group in latent space.

Called contractive because they *contract* neighborhood of input space into *smaller*, *localized* group in latent space.

- This contractive effect is designed to only occur locally.

Called contractive because they *contract* neighborhood of input space into *smaller*, *localized* group in latent space.

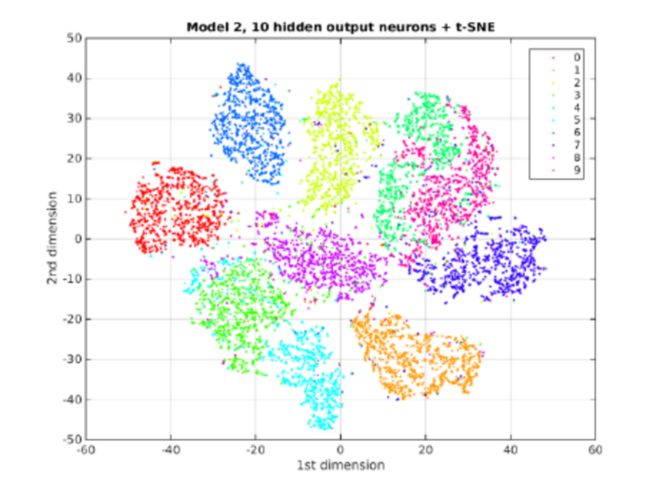
- This contractive effect is designed to only occur locally.
- The Jacobian Matrix will see most of its eigenvalues drop below 1 $\rightarrow$ contracted directions

In-Depth Look at Contractive Autoencoders

Called contractive because they *contract* neighborhood of input space into *smaller*, *localized* group in latent space.

- This contractive effect is designed to only occur locally.
- The Jacobian Matrix will see most of its eigenvalues drop below 1 $\rightarrow$ contracted directions
- But some directions will have eigenvalues (significantly) above 1 $\rightarrow$ directions that explain most of the variance in data

Example: MNIST in 2D manifold



The Big Idea of Regularized Autoencoders

Previous slides underscore the central balance of regularized autoencoders:

Previous slides underscore the central balance of regularized autoencoders:

- Be **sensitive** to inputs (reconstruction loss) → generate good reconstructions of data drawn from data distribution

The Big Idea of Regularized Autoencoders

Previous slides underscore the central balance of regularized autoencoders:

- Be **sensitive** to inputs (reconstruction loss) → generate good reconstructions of data drawn from data distribution
- Be **insensitive** to inputs (regularization penalty) → learn actual data distribution

Connecting Denoising and Contractive Autoencoders

Alain and Bengio (2013) showed that denoising penalty on tiny Gaussian noise is, in the limit, $\approx$ contractive penalty on $\mathbf{x}, \mathbf{g}(\mathbf{f}(\mathbf{x}))$.

Connecting Denoising and Contractive Autoencoders

Alain and Bengio (2013) showed that denoising penalty on tiny Gaussian noise is, in the limit, $\approx$ contractive penalty on $\mathbf{x}, \mathbf{g}(\mathbf{f}(\mathbf{x}))$.

- Denoising Autoencoders make **reconstruction function** resist small, finite-sized perturbations in input.

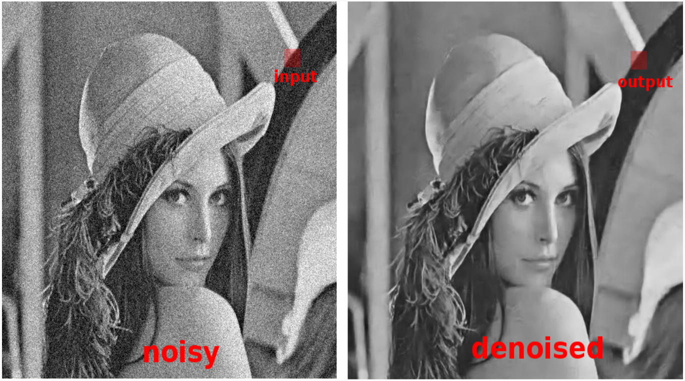
Connecting Denoising and Contractive Autoencoders

Alain and Bengio (2013) showed that denoising penalty on tiny Gaussian noise is, in the limit, $\approx$ contractive penalty on $\mathbf{x}, \mathbf{g}(\mathbf{f}(\mathbf{x}))$.

- Denoising Autoencoders make **reconstruction function** resist small, finite-sized perturbations in input.
- Contractive Autoencoders make **feature encoding function** resist infinitesimal perturbations in input.

Connecting Denoising and Contractive Autoencoders

Handling noise $\sim$ Contractive property



Representational Power, Layer Size and Depth

Deeper autoencoders tend to generalize better and train more efficiently than shallow ones.

Deeper autoencoders tend to generalize better and train more efficiently than shallow ones.

- Common strategy: greedily pre-train layers and stack them

Deeper autoencoders tend to generalize better and train more efficiently than shallow ones.

- Common strategy: greedily pre-train layers and stack them
- For contractive autoencoders, calculating Jacobian for deep networks is expensive. Good idea to do layer-by-layer.

Applications of Autoencoders

- **Dimensionality Reduction:** Make high-quality, low-dimension representation of data

Applications of Autoencoders

- **Dimensionality Reduction:** Make high-quality, low-dimension representation of data
- **Information Retrieval:** Locate value in database which is just autoencoded key.

Applications of Autoencoders

- **Dimensionality Reduction:** Make high-quality, low-dimension representation of data
- **Information Retrieval:** Locate value in database which is just autoencoded key.
    - If you need binary for hash table, use sigmoid in final layer.

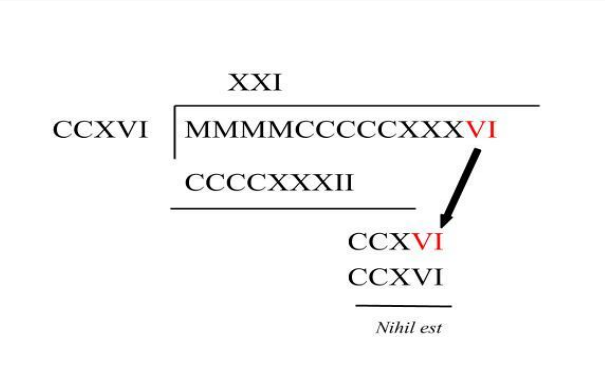
Representation Learning

The Power of Representations

Representations are important: try long division with Roman numerals

The Power of Representations

Representations are important: try long division with Roman numerals



Other examples: **Variables** in algebra, **cartesian grid** for analytic geometry, **binary encodings** for information theory, electronics

Representations in Deep Learning

A good representation of data makes subsequent tasks easier - **more tractable, less expensive.**

- Feedforward nets: Hidden layers make representation for output layer (linear classifier)

A good representation of data makes subsequent tasks easier - **more tractable, less expensive.**

- Feedforward nets: Hidden layers make representation for output layer (linear classifier)
- Conv nets: Maintain topology of input, convert into 3-D convolutions, pooling, etc.

A good representation of data makes subsequent tasks easier - **more tractable, less expensive.**

- Feedforward nets: Hidden layers make representation for output layer (linear classifier)
- Conv nets: Maintain topology of input, convert into 3-D convolutions, pooling, etc.
- Autoencoders: The entire mission of the architecture

Vector $\in \mathcal{R}^n$, *each spot* symbolizing exactly one category.

Vector $\in \mathcal{R}^n$, *each spot* symbolizing exactly one category.

- Example: Bag-of-words (one-hot or *n*-grams) in NLP.

Symbolic Representations

Vector $\in \mathcal{R}^n$, *each spot* symbolizing exactly one category.

- Example: Bag-of-words (one-hot or *n*-grams) in NLP.
- All words / n-grams equally distant from one another.

Symbolic Representations

Vector $\in \mathcal{R}^n$, *each spot* symbolizing exactly one category.

- Example: Bag-of-words (one-hot or n-grams) in NLP.
- All words / n-grams equally distant from one another.
- Representation does not capture features!

Symbolic Representations

Vector $\in \mathcal{R}^n$, *each spot* symbolizing exactly one category.

- Example: Bag-of-words (one-hot or n-grams) in NLP.
- All words / n-grams equally distant from one another.
- Representation does not capture features!

Symbolic Representations

Vector $\in \mathcal{R}^n$, *each spot* symbolizing exactly one category.

- Example: Bag-of-words (one-hot or n-grams) in NLP.
- All words / n-grams equally distant from one another.
- Representation does not capture features!

Fundamentally limited: $\sim \mathcal{O}(n)$ possible representations.

Distributed Representations

Have vector $\in \mathcal{R}^n$, *each possible vector* symbolizing one category.

Distributed Representations

Have vector $\in \mathcal{R}^n$, *each possible vector* symbolizing one category.

- Example: Word embeddings in NLP.

Distributed Representations

Have vector $\in \mathcal{R}^n$, *each possible vector* symbolizing one category.

- Example: Word embeddings in NLP.
- Can encode similarity and meaningful distance in embedding space.

Distributed Representations

Have vector $\in \mathcal{R}^n$, *each possible vector* symbolizing one category.

- Example: Word embeddings in NLP.
- Can encode similarity and meaningful distance in embedding space.
- Spots can encode features: *number of legs* vs. *is a dog*

Distributed Representations

Have vector $\in \mathcal{R}^n$, *each possible vector* symbolizing one category.

- Example: Word embeddings in NLP.
- Can encode similarity and meaningful distance in embedding space.
- Spots can encode features: *number of legs* vs. *is a dog*

Distributed Representations

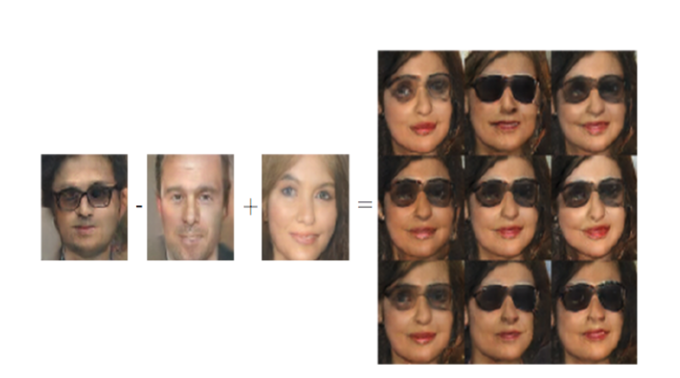
Have vector $\in \mathcal{R}^n$, *each possible vector* symbolizing one category.

- Example: Word embeddings in NLP.
- Can encode similarity and meaningful distance in embedding space.
- Spots can encode features: *number of legs* vs. *is a dog*

Pretty much always preferred: $\sim \mathcal{O}(k^n)$ possible representations, where k is number of values a feature can take on.

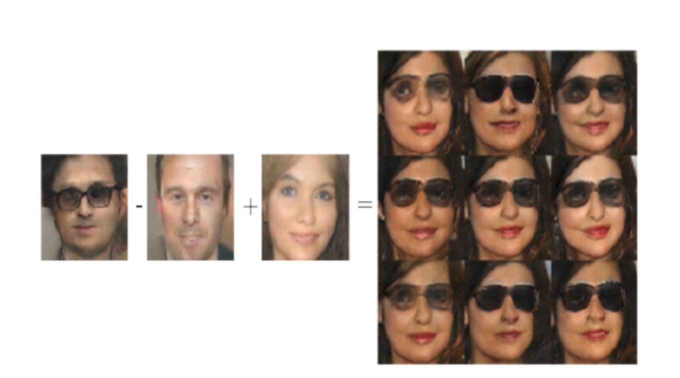
Benefits of Distributed Representations

Distributed representation → Data Manifold



Benefits of Distributed Representations

Distributed representation → Data Manifold



Also: less dimensionality, faster training

Representation Learning Techniques

Greedy Layer-Wise Unsupervised Pretraining

Pivotal technique that allowed training of deep nets without specialized properties (convolution, recurrence, etc.)

Greedy Layer-Wise Unsupervised Pretraining

Pivotal technique that allowed training of deep nets without specialized properties (convolution, recurrence, etc.)

- Key Idea: **Leverage representations** learned for one task to solve another.

Greedy Layer-Wise Unsupervised Pretraining

Pivotal technique that allowed training of deep nets without specialized properties (convolution, recurrence, etc.)

- Key Idea: **Leverage representations** learned for one task to solve another.
- Train **each layer** of feedforward net **greedily** as a representation learning alg. e.g. autoencoder.

Greedy Layer-Wise Unsupervised Pretraining

Pivotal technique that allowed training of deep nets without specialized properties (convolution, recurrence, etc.)

- Key Idea: **Leverage representations** learned for one task to solve another.
- Train **each layer** of feedforward net **greedily** as a representation learning alg. e.g. autoencoder.
- Continue stacking layers. Output of all prior layers is input for next one.

Greedy Layer-Wise Unsupervised Pretraining

Pivotal technique that allowed training of deep nets without specialized properties (convolution, recurrence, etc.)

- Key Idea: **Leverage representations** learned for one task to solve another.
- Train **each layer** of feedforward net **greedily** as a representation learning alg. e.g. autoencoder.
- Continue stacking layers. Output of all prior layers is input for next one.
- **Fine tune**, i.e. jointly train, all layers once each has learned representations.

Greedy Layer-Wise Unsupervised Pretraining (Contd.)

Works on two assumptions:

Greedy Layer-Wise Unsupervised Pretraining (Contd.)

Works on two assumptions:

- Picking initial parameters has regularizing effect and improves generalization, optimization. (Not well understood)

Greedy Layer-Wise Unsupervised Pretraining (Contd.)

Works on two assumptions:

- Picking initial parameters has regularizing effect and improves generalization, optimization. (Not well understood)
- Learning properties of input distribution can help in mapping inputs to outputs. (Better understood)

Greedy Layer-Wise Unsupervised Pretraining (Contd.)

Works on two assumptions:

- Picking initial parameters has regularizing effect and improves generalization, optimization. (Not well understood)
- Learning properties of input distribution can help in mapping inputs to outputs. (Better understood)

Greedy Layer-Wise Unsupervised Pretraining (Contd.)

Works on two assumptions:

- Picking initial parameters has regularizing effect and improves generalization, optimization. (Not well understood)
- Learning properties of input distribution can help in mapping inputs to outputs. (Better understood)

Sometimes helpful, sometimes not:

- Effective for word embeddings - replaces one-hot. Also for very complex functions shaped by input data distribution

Greedy Layer-Wise Unsupervised Pretraining (Contd.)

Works on two assumptions:

- Picking initial parameters has regularizing effect and improves generalization, optimization. (Not well understood)
- Learning properties of input distribution can help in mapping inputs to outputs. (Better understood)

Sometimes helpful, sometimes not:

- Effective for word embeddings - replaces one-hot. Also for very complex functions shaped by input data distribution
- Useful when few labeled, many unlabeled examples - **semi-supervised learning**

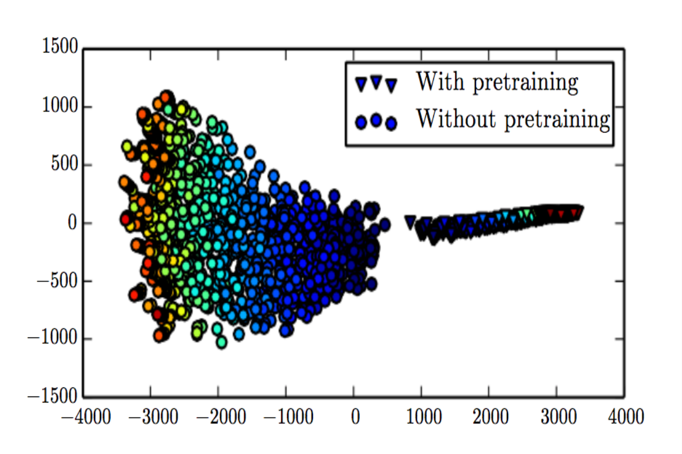
Greedy Layer-Wise Unsupervised Pretraining (Contd.)

Works on two assumptions:

- Picking initial parameters has regularizing effect and improves generalization, optimization. (Not well understood)
- Learning properties of input distribution can help in mapping inputs to outputs. (Better understood)

Sometimes helpful, sometimes not:

- Effective for word embeddings - replaces one-hot. Also for very complex functions shaped by input data distribution
- Useful when few labeled, many unlabeled examples - **semi-supervised learning**
- Less effective for images - topology is already present.


1500
1000
500
0
−500
−1000
−1500
−4000
−3000
−2000
−1000
0
1000
2000
3000
4000
With pretraining
Without pretraining

Multi-Task Learning

Given two similar learning tasks and their labeled data: D_1, D_2.

D_2 has few examples compared to D_1.

Multi-Task Learning

Given two similar learning tasks and their labeled data: D_1, D_2.
D_2 has few examples compared to D_1.

- Idea: (Pre-)Train network on D_1, then work D_2.

Multi-Task Learning

Given two similar learning tasks and their labeled data: D_1, D_2.
D_2 has few examples compared to D_1.

- Idea: (Pre-)Train network on D_1, then work D_2.
- Hopefully, **low-level** features from D_1 are useful for D_2, and fine-tuning is enough for D_2.

Inputs are similar, while **labels are different** between D_1, D_2.
Ex: Images of dogs or cats (D_1). Then classify images as horse or cow (D_2).

Transfer Learning

Inputs are similar, while **labels are different** between D_1, D_2.
Ex: Images of dogs or cats (D_1). Then classify images as horse or cow (D_2).

- Low-level features of **inputs**, are same: lighting, animal orientations, edges, faces.

Inputs are similar, while **labels are different** between D_1, D_2.
Ex: Images of dogs or cats (D_1). Then classify images as horse or cow (D_2).

- Low-level features of **inputs**, are same: lighting, animal orientations, edges, faces.
- **Labels** are fundamentally different.

Transfer Learning

Inputs are similar, while **labels are different** between D_1, D_2.
Ex: Images of dogs or cats (D_1). Then classify images as horse or cow (D_2).

- Low-level features of **inputs**, are same: lighting, animal orientations, edges, faces.
- **Labels** are fundamentally different.
- Learning of D_1 will establish latent space where dists. are separated. Then adjust to assign D_2 labels to transformed D_2 by pre-trained network.

Domain Adaptation

Labels are similar, while the **inputs are different**.

Ex: Speech-to-text system for person 1 (D_1). Then train to also work for person 2 (D_2).

Domain Adaptation

Labels are similar, while the **inputs are different**.
Ex: Speech-to-text system for person 1 (D_1). Then train to also work for person 2 (D_2).

- For both, text must be valid English sentences, so labels are similar.

Labels are similar, while the **inputs are different**.
Ex: Speech-to-text system for person 1 (D_1). Then train to also work for person 2 (D_2).

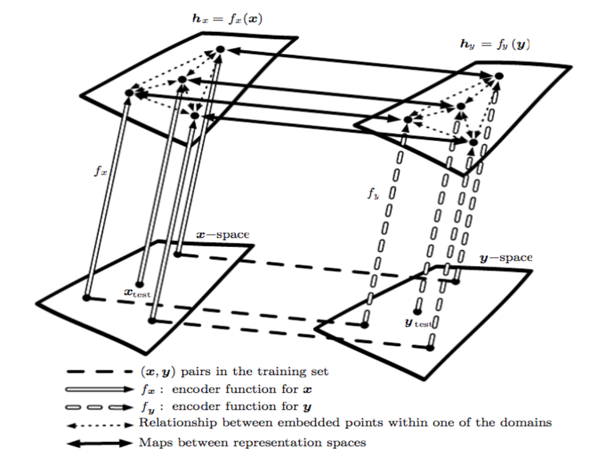
- For both, text must be valid English sentences, so labels are similar.
- Speakers may have diff. pitch depth, accents, etc $\rightarrow$different inputs.

Domain Adaptation

Labels are similar, while the **inputs are different**.

Ex: Speech-to-text system for person 1 (D_1). Then train to also work for person 2 (D_2).

- For both, text must be valid English sentences, so labels are similar.
- Speakers may have diff. pitch depth, accents, etc $\rightarrow$different inputs.
- Training on D_1 gives model power to map noise to English in general. Just adjust to assign D_2 input to D_2 labels.



$h_x = f_x(x)$
$h_y = f_y(y)$
f_x
f_y
x−space
y−space
x_{test}
y_{test}
(x, y) pairs in the training set
f_x : encoder function for x
f_y : encoder function for y
Relationship between embedded points within one of the domains
Maps between representation spaces

More About Multi-Task Learning

So far: supervised, but also works with unsupervised and RL.

Deeper networks make significant impact in Multi-Task Learning.

More About Multi-Task Learning

So far: supervised, but also works with unsupervised and RL.
Deeper networks make significant impact in Multi-Task Learning.

- **One-Shot** Learning only uses one labeled example from D_2. Training from D_1 gives clean separations in space and then can infer whole cluster labels.

More About Multi-Task Learning

So far: supervised, but also works with unsupervised and RL.

Deeper networks make significant impact in Multi-Task Learning.

- **One-Shot** Learning only uses one labeled example from D_2. Training from D_1 gives clean separations in space and then can infer whole cluster labels.
- **Zero-Shot** Learning is able to work with zero labeled training examples. Learn $p(y|x, T)$, T being a context variable

More About Multi-Task Learning

So far: supervised, but also works with unsupervised and RL.

Deeper networks make significant impact in Multi-Task Learning.

- **One-Shot** Learning only uses one labeled example from D_2. Training from D_1 gives clean separations in space and then can infer whole cluster labels.
- **Zero-Shot** Learning is able to work with zero labeled training examples. Learn $p(y|x, T)$, T being a context variable

More About Multi-Task Learning

So far: supervised, but also works with unsupervised and RL.

Deeper networks make significant impact in Multi-Task Learning.

- **One-Shot** Learning only uses one labeled example from D_2. Training from D_1 gives clean separations in space and then can infer whole cluster labels.
- **Zero-Shot** Learning is able to work with zero labeled training examples. Learn $p(y|x, T)$, T being a context variable
  - For example: T is sentences: cats have four legs, pointy ears, fur, etc. x is images, with y being label of cat or not.

Isolating Causal Factors

Two desirable properties of representations. They often coincide:

Isolating Causal Factors

Two desirable properties of representations. They often coincide:

- **Disentangled Causes:** for rep. $p(\mathbf{x})$, we want to know $p(\mathbf{y}|\mathbf{x})$ i.e., does $\mathbf{y}$ cause $\mathbf{x}$.

Isolating Causal Factors

Two desirable properties of representations. They often coincide:

- **Disentangled Causes:** for rep. $p(\mathbf{x})$, we want to know $p(\mathbf{y}|\mathbf{x})$ i.e., does $\mathbf{y}$ cause $\mathbf{x}$.
  - If $\mathbf{x}, \mathbf{y}$ correlated, then $p(\mathbf{x})$ and $p(\mathbf{y}|\mathbf{x})$ will be strongly tied. We want this relation to be clear, hence disentangled.

Isolating Causal Factors

Two desirable properties of representations. They often coincide:

- **Disentangled Causes:** for rep. $p(\mathbf{x})$, we want to know $p(\mathbf{y}|\mathbf{x})$ i.e., does $\mathbf{y}$ cause $\mathbf{x}$.
  - If $\mathbf{x}, \mathbf{y}$ correlated, then $p(\mathbf{x})$ and $p(\mathbf{y}|\mathbf{x})$ will be strongly tied. We want this relation to be clear, hence disentangled.
- **Easy Modeling:** representations that have sparse feature vectors which imply independent features

Ideal Latent Variables

Assume $\mathbf{y}$ is a causal factor of $\mathbf{x}$ and $\mathbf{h}$ represents all of those factors.

Ideal Latent Variables

Assume $\mathbf{y}$ is a causal factor of $\mathbf{x}$ and $\mathbf{h}$ represents all of those factors.

- Joint distribution of model is: $p(\mathbf{x}, \mathbf{h}) = p(\mathbf{x}|\mathbf{h})p(\mathbf{h})$

Ideal Latent Variables

Assume **y** is a causal factor of **x** and **h** represents all of those factors.

- Joint distribution of model is: $p(\mathbf{x}, \mathbf{h}) = p(\mathbf{x}|\mathbf{h})p(\mathbf{h})$
- Marginal probability of **x** is

$$p(\mathbf{x}) = \sum_h p(h)p(\mathbf{x}|h) = \mathbb{E}_{\mathbf{h}} p(\mathbf{x}|\mathbf{h})$$

Thus, best latent var **h** (w.r.t. p(**x**)) explains **x** from a **causal point of view.**

Ideal Latent Variables

Assume $\mathbf{y}$ is a causal factor of $\mathbf{x}$ and $\mathbf{h}$ represents all of those factors.

- Joint distribution of model is: $p(\mathbf{x}, \mathbf{h}) = p(\mathbf{x}|\mathbf{h})p(\mathbf{h})$
- Marginal probability of $\mathbf{x}$ is

$$p(\mathbf{x}) = \sum_h p(h)p(\mathbf{x}|h) = \mathbb{E}_{\mathbf{h}} p(\mathbf{x}|\mathbf{h})$$

  Thus, best latent var $\mathbf{h}$ (w.r.t. p($\mathbf{x}$)) explains $\mathbf{x}$ from a **causal point of view.**

- $p(y|x)$ depends on $p(x)$, hence h being causal is valuable.

The Real World

Real world data often has more causes than can/should be encoded.

Real world data often has more causes than can/should be encoded.

- Humans fail to detect changes in environment unimportant to current task.

Real world data often has more causes than can/should be encoded.

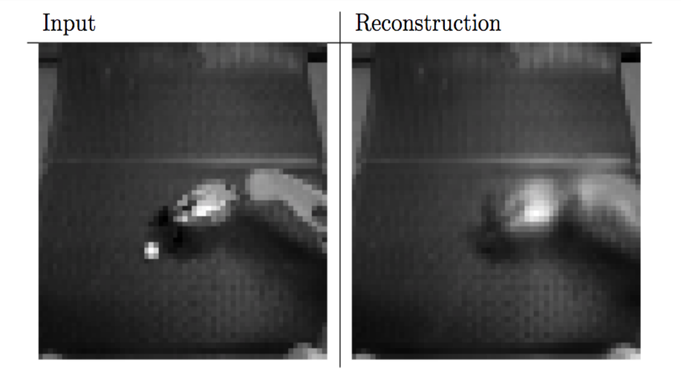
- Humans fail to detect changes in environment unimportant to current task.
- Must establish learnable measures of **saliency** to attach to features.

The Real World

Real world data often has more causes than can/should be encoded.

- Humans fail to detect changes in environment unimportant to current task.
- Must establish learnable measures of **saliency** to attach to features.
- Example: Autoencoders trained on images often fail to register important small objects like ping pong balls.

Failure of Traditional Loss Functions



The Adversarial Approach to Saliency

MSE: salience presumably affects pixel intensity for **large number** of pixels.

The Adversarial Approach to Saliency

MSE: salience presumably affects pixel intensity for **large number** of pixels.

Adversarial: *learn* saliency by tricking a discriminator network

The Adversarial Approach to Saliency

MSE: salience presumably affects pixel intensity for **large number** of pixels.

Adversarial: *learn* saliency by tricking a discriminator network

- Discriminator is trained to tell between ground truth and generated data

The Adversarial Approach to Saliency

MSE: salience presumably affects pixel intensity for **large number** of pixels.

Adversarial: *learn* saliency by tricking a discriminator network

- Discriminator is trained to tell between ground truth and generated data
- Discriminator can attach high saliency to **small number** of pixels

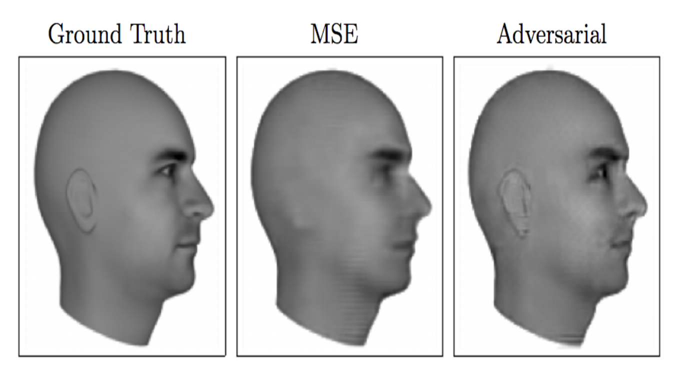
The Adversarial Approach to Saliency

MSE: salience presumably affects pixel intensity for **large number** of pixels.

Adversarial: *learn* saliency by tricking a discriminator network

- Discriminator is trained to tell between ground truth and generated data
- Discriminator can attach high saliency to **small number** of pixels
- Framework of **Generative Adversarial Networks** (more later in course).

Comparing Traditional to Adversarial



Conclusion

Conclusion

- The crux of autoencoders is representation learning.

Conclusion

- The crux of autoencoders is representation learning.
- The crux of deep learning is representation learning.

Conclusion

- The crux of autoencoders is representation learning.
- The crux of deep learning is representation learning.
- The crux of intelligence is probably representation learning.

Questions

Questions?