Learn Latent Representations

VLC = ∭ Vision Learning Control dVdLdC

Autoencoders and Self-supervised Learning

Jonathon Hare

Vision, Learning and Control
University of Southampton

Low Dimensional Representations

- One of the common features of many of the deep learning models we have looked at to this point is that they often try to reduce the dimensionality of the input data in order to capture some kind of underlying information.

Low Dimensional Representations

- One of the common features of many of the deep learning models we have looked at to this point is that they often try to reduce the dimensionality of the input data in order to capture some kind of underlying information.
- A few lectures ago this was particularly evident when when we looked at embedding models like word2vec which explictly try to capture relationships in the data in a low dimensional 'latent' space.

Self-supervised Learning

Yann LeCun

30 April 2019 ·

I now call it "self-supervised learning", because "unsupervised" is both a loaded and confusing term.

In self-supervised learning, the system learns to predict part of its input from other parts of it input. In other words a portion of the input is used as a supervisory signal to a predictor fed with the remaining portion of the input.

Self-supervised Learning

- The word2vec models are examples of *self-supervised learning*

Self-supervised Learning

- The word2vec models are examples of *self-supervised learning*
  - CBOW learns to predict the focus word from the context words

Self-supervised Learning

- The word2vec models are examples of *self-supervised learning*
  - CBOW learns to predict the focus word from the context words
  - Skip-gram learns to predict the context words from the focus word

Self-supervised Learning

- The word2vec models are examples of *self-supervised learning*
  - CBOW learns to predict the focus word from the context words
  - Skip-gram learns to predict the context words from the focus word
- Let's now consider a different type self-supervised of task where we want to learn a model that learns to **copy** its input to its output.

Autoencoders

- An **autoencoder** is a network that is trained to copy its input to its output

Autoencoders

- An **autoencoder** is a network that is trained to copy its input to its output
  - Internally there is some hidden vector $\boldsymbol{z}$ that describes a **code** that represents the input.

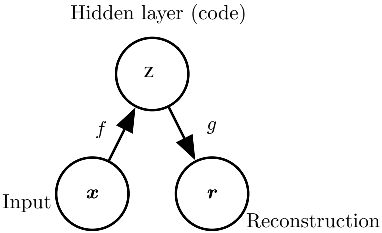
Autoencoders

- An **autoencoder** is a network that is trained to copy its input to its output
  - Internally there is some hidden vector $\boldsymbol{z}$ that describes a **code** that represents the input.
  - Conceptually the autoencoder consists of two parts:
    - The encoder $\boldsymbol{z} = f(\boldsymbol{x})$
    - The decoder $\boldsymbol{r} = g(\boldsymbol{z})$

Autoencoders

- An **autoencoder** is a network that is trained to copy its input to its output
    - Internally there is some hidden vector $\boldsymbol{z}$ that describes a **code** that represents the input.
    - Conceptually the autoencoder consists of two parts:
        - The encoder $\boldsymbol{z} = f(\boldsymbol{x})$
        - The decoder $\boldsymbol{r} = g(\boldsymbol{z})$
    - and has loss that tries to minimise the reconstruction error (typically SSE/MSE: $\|\boldsymbol{x} - \boldsymbol{r}\|_2^2$)

- An **autoencoder** is a network that is trained to copy its input to its output
  - Internally there is some hidden vector $\boldsymbol{z}$ that describes a **code** that represents the input.
  - Conceptually the autoencoder consists of two parts:
    - The encoder $\boldsymbol{z} = f(\boldsymbol{x})$
    - The decoder $\boldsymbol{r} = g(\boldsymbol{z})$
  - and has loss that tries to minimise the reconstruction error (typically SSE/MSE: $\|\boldsymbol{x} - \boldsymbol{r}\|_2^2$)



Autoencoder constraints

- Clearly a linear autoencoder with a sufficient number of weights (e.g. if the dimension of $\boldsymbol{z}$ was greater than or equal to that of x) could learn set $g(f(\boldsymbol{x})) = \boldsymbol{x}$ everywhere, but this obviously wouldn't be useful!

[1] these are 'inductive biases' and the 'innate priors' of the model and learning algorithm

Autoencoder constraints

- Clearly a linear autoencoder with a sufficient number of weights (e.g. if the dimension of $\boldsymbol{z}$ was greater than or equal to that of x) could learn set $g(f(\boldsymbol{x})) = \boldsymbol{x}$ everywhere, but this obviously wouldn't be useful!
- In practice we apply *restrictions*[1] to stop this happening.
- The objective is to use these restrictions to force the autoencoder to learn useful properties of the data.

[1] these are 'inductive biases' and the 'innate priors' of the model and learning algorithm

Undercomplete Autoencoders

- Undercomplete autoencoders have $\dim(\boldsymbol{z}) << \dim(\boldsymbol{x})$.
- This forces the encoder to learn a *compressed representation* of the input.
- The representation will capture the most *salient* features of the input data.

Undercomplete Autoencoders — Linear

Consider the single-hidden layer linear autoencoder network given by:

$$
\begin{aligned}
z &= \mathbf{W}_e \boldsymbol{x} + \boldsymbol{b}_e \\
r &= \mathbf{W}_d \boldsymbol{z} + \boldsymbol{b}_d
\end{aligned}
$$

where $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{z} \in \mathbb{R}^m$ and $m < n$.

Undercomplete Autoencoders — Linear

Consider the single-hidden layer linear autoencoder network given by:

$$z = \mathbf{W}_e \boldsymbol{x} + \boldsymbol{b}_e$$
$$r = \mathbf{W}_d \boldsymbol{z} + \boldsymbol{b}_d$$

where $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{z} \in \mathbb{R}^m$ and $m < n$.

With the MSE loss, this autoencoder will learn to span the same subspace as PCA for a given set of training data.

Note that the autoencoder weights are not however constrained to be orthogonal (like they would be in PCA)

Undercomplete Autoencoders — deeper and nonlinear

- A linear autoencoder with a single hidden layer learns to map into the same subspace as PCA.

[2] Bourlard, H., Kamp, Y. Auto-association by multilayer perceptrons and singular value decomposition. Biol. Cybern. 59, 291–294 (1988). https://doi.org/10.1007/BF00332918

Undercomplete Autoencoders — deeper and nonlinear

- A linear autoencoder with a single hidden layer learns to map into the same subspace as PCA.
- Clearly, a deeper, linear autoencoder would also do the same thing.

[2] Bourlard, H., Kamp, Y. Auto-association by multilayer perceptrons and singular value decomposition. Biol. Cybern. 59, 291–294 (1988). https://doi.org/10.1007/BF00332918

Undercomplete Autoencoders — deeper and nonlinear

- A linear autoencoder with a single hidden layer learns to map into the same subspace as PCA.
- Clearly, a deeper, linear autoencoder would also do the same thing.
- What happens if you introduce non-linearity?

[2] Bourlard, H., Kamp, Y. Auto-association by multilayer perceptrons and singular value decomposition. Biol. Cybern. 59, 291–294 (1988). https://doi.org/10.1007/BF00332918

Undercomplete Autoencoders — deeper and nonlinear

- A linear autoencoder with a single hidden layer learns to map into the same subspace as PCA.
- Clearly, a deeper, linear autoencoder would also do the same thing.
- What happens if you introduce non-linearity?
    - Interestingly, a single hidden layer network with non-linear activations on the encoder (keeping the decoder linear) and MSE loss also just learns to span the PCA subspace[2]!

[2] Bourlard, H., Kamp, Y. Auto-association by multilayer perceptrons and singular value decomposition. Biol. Cybern. 59, 291–294 (1988). https://doi.org/10.1007/BF00332918

Undercomplete Autoencoders — deeper and nonlinear

- A linear autoencoder with a single hidden layer learns to map into the same subspace as PCA.
- Clearly, a deeper, linear autoencoder would also do the same thing.
- What happens if you introduce non-linearity?
  - Interestingly, a single hidden layer network with non-linear activations on the encoder (keeping the decoder linear) and MSE loss also just learns to span the PCA subspace[2]!
  - But, if you add more hidden layers with non-linear activations (to either the encoder, decoder or both) you can effectively perform a powerful non-linear generalisation of PCA

[2] Bourlard, H., Kamp, Y. Auto-association by multilayer perceptrons and singular value decomposition. Biol. Cybern. 59, 291–294 (1988). https://doi.org/10.1007/BF00332918

Deep Autoencoders

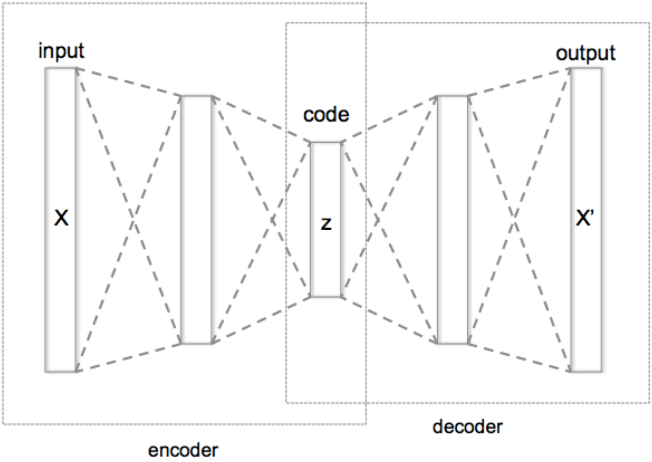


Image taken from wikipedia

Deep Autoencoders - caveat

- There is a slight catch: if you give the deep autoencoder network too much capacity (too many weights) it will learn to perform the copying task without extracting anything useful about the data.

Deep Autoencoders - caveat

- There is a slight catch: if you give the deep autoencoder network too much capacity (too many weights) it will learn to perform the copying task without extracting anything useful about the data.
- Of course this means that will likely not generalise to unseen data.

Deep Autoencoders - caveat

- There is a slight catch: if you give the deep autoencoder network too much capacity (too many weights) it will learn to perform the copying task without extracting anything useful about the data.
- Of course this means that will likely not generalise to unseen data.
- Extreme example:

Deep Autoencoders - caveat

- There is a slight catch: if you give the deep autoencoder network too much capacity (too many weights) it will learn to perform the copying task without extracting anything useful about the data.
- Of course this means that will likely not generalise to unseen data.
- Extreme example:
  - Consider a powerful encoder that maps $\boldsymbol{x}$ to $\boldsymbol{z} \in \mathbb{R}^1$

Deep Autoencoders - caveat

- There is a slight catch: if you give the deep autoencoder network too much capacity (too many weights) it will learn to perform the copying task without extracting anything useful about the data.
- Of course this means that will likely not generalise to unseen data.
- Extreme example:
  - Consider a powerful encoder that maps $\boldsymbol{x}$ to $\boldsymbol{z} \in \mathbb{R}^1$
  - Each training example $\boldsymbol{x}^{(i)}$ could e.g. be mapped to i.

Deep Autoencoders - caveat

- There is a slight catch: if you give the deep autoencoder network too much capacity (too many weights) it will learn to perform the copying task without extracting anything useful about the data.
- Of course this means that will likely not generalise to unseen data.
- Extreme example:
  - Consider a powerful encoder that maps $\boldsymbol{x}$ to $\boldsymbol{z} \in \mathbb{R}^1$
  - Each training example $\boldsymbol{x}^{(i)}$ could e.g. be mapped to i.
  - The decoder just needs to memorise the training examples so that it can map back from i.

Undercomplete Autoencoders — Convolutional

- Thus far, we only considered autoencoders with vector inputs/outputs and fully-connected layers.
- There is nothing stopping us using any other kinds of layers though...

Undercomplete Autoencoders — Convolutional

- Thus far, we only considered autoencoders with vector inputs/outputs and fully-connected layers.
- There is nothing stopping us using any other kinds of layers though...
- If we're working with image data, where we know that much of the structure is 'local', then using convolutions in both the decoder makes sense

Convolutional Autoencoder

Regularised Autoencoders

- Rather than (necessarily) forcing the hidden vector to have a lower dimensionality than the input, we could instead utilise some form of regularisation to force the network to learn interesting representations...

Regularised Autoencoders

- Rather than (necessarily) forcing the hidden vector to have a lower dimensionality than the input, we could instead utilise some form of regularisation to force the network to learn interesting representations...
- Many ways to do this; let's look at two of them:
  - Denoising Autoencoders
  - Sparse Autoencoders

Denoising Autoencoders

- Denoising autoencoders take a partially corrupted input and train to recover the original undistorted input.

Denoising Autoencoders

- Denoising autoencoders take a partially corrupted input and train to recover the original undistorted input.
- To train an autoencoder to denoise data, it is necessary to perform a preliminary stochastic mapping to corrupt the data ($x \rightarrow \tilde{x}$).
    - E.g. by adding Gaussian noise.

Denoising Autoencoders

- Denoising autoencoders take a partially corrupted input and train to recover the original undistorted input.
- To train an autoencoder to denoise data, it is necessary to perform a preliminary stochastic mapping to corrupt the data ($x \to \tilde{x}$).
  - E.g. by adding Gaussian noise.
- The loss is computed between the reconstruction (computed from the noisy input) against the original noise-free data.

Sparse Autoencoders

- In a sparse autoencoder, there can be more hidden units than inputs, but only a small number of the hidden units are allowed to be active at the same time.

Sparse Autoencoders

- In a sparse autoencoder, there can be more hidden units than inputs, but only a small number of the hidden units are allowed to be active at the same time.
- This is simply achieved with a regularised loss function:
  $\ell = \ell_{mse} + \Omega(\boldsymbol{z})$

Sparse Autoencoders

- In a sparse autoencoder, there can be more hidden units than inputs, but only a small number of the hidden units are allowed to be active at the same time.
- This is simply achieved with a regularised loss function: $\ell = \ell_{mse} + \Omega(\boldsymbol{z})$
- A popular choice that you've seen before would be to use an l1 penalty $\Omega(\boldsymbol{z}) = \lambda \sum_i |z_i|$

Sparse Autoencoders

- In a sparse autoencoder, there can be more hidden units than inputs, but only a small number of the hidden units are allowed to be active at the same time.
- This is simply achieved with a regularised loss function: $\ell = \ell_{mse} + \Omega(\boldsymbol{z})$
- A popular choice that you've seen before would be to use an l1 penalty $\Omega(\boldsymbol{z}) = \lambda \sum_i |z_i|$
  - this of course does have a slight problem... what is the derivative of $y = |x|$ with respect to x at $x = 0$?

Autoencoder Applications

- Any basic AE (or its variant) can be used to learn a compact representation of data.

Autoencoder Applications

- Any basic AE (or its variant) can be used to learn a compact representation of data.
  - You can learn useful features from data without the need for labelled data.

Autoencoder Applications

- Any basic AE (or its variant) can be used to learn a compact representation of data.
    - You can learn useful features from data without the need for labelled data.
    - Denoising can help generalise over the test set since the data is distorted by adding noise.

Autoencoder Applications

- Any basic AE (or its variant) can be used to learn a compact representation of data.
    - You can learn useful features from data without the need for labelled data.
    - Denoising can help generalise over the test set since the data is distorted by adding noise.
- Pretraining networks

Autoencoder Applications

- Any basic AE (or its variant) can be used to learn a compact representation of data.
  - You can learn useful features from data without the need for labelled data.
  - Denoising can help generalise over the test set since the data is distorted by adding noise.
- Pretraining networks
- Anomaly Detection

Autoencoder Applications

- Any basic AE (or its variant) can be used to learn a compact representation of data.
    - You can learn useful features from data without the need for labelled data.
    - Denoising can help generalise over the test set since the data is distorted by adding noise.
- Pretraining networks
- Anomaly Detection
- Machine translation

Autoencoder Applications

- Any basic AE (or its variant) can be used to learn a compact representation of data.
  - You can learn useful features from data without the need for labelled data.
  - Denoising can help generalise over the test set since the data is distorted by adding noise.
- Pretraining networks
- Anomaly Detection
- Machine translation
- Semantic segmentation

Beyond Deterministic Autoencoders: Stochastic Encoders and Decoders

- When we trained supervised classification networks we usually assume that the network produces an output distribution $p(\boldsymbol{y}|\boldsymbol{x})$ and try to minimise the negative log-likelihood $-\log(p(\boldsymbol{y}|\boldsymbol{x}))$.

Beyond Deterministic Autoencoders: Stochastic Encoders and Decoders

- When we trained supervised classification networks we usually assume that the network produces an output distribution $p(\boldsymbol{y}|\boldsymbol{x})$ and try to minimise the negative log-likelihood $-\log(p(\boldsymbol{y}|\boldsymbol{x}))$.
- In a decoder of an autoencoder we could do the same thing and have the decoder learn $p_{decoder}(\boldsymbol{x}|\boldsymbol{z})$ by minimising $-\log(p(\boldsymbol{x}|\boldsymbol{z}))$.

Beyond Deterministic Autoencoders: Stochastic Encoders and Decoders

- When we trained supervised classification networks we usually assume that the network produces an output distribution $p(\boldsymbol{y}|\boldsymbol{x})$ and try to minimise the negative log-likelihood $-\log(p(\boldsymbol{y}|\boldsymbol{x}))$.
- In a decoder of an autoencoder we could do the same thing and have the decoder learn $p_{decoder}(\boldsymbol{x}|\boldsymbol{z})$ by minimising $-\log(p(\boldsymbol{x}|\boldsymbol{z}))$.
  - A linear output layer could parameterise the mean of a Gaussian distribution for real-valued $\boldsymbol{x}$; in this case the negative log likelihood yields the MSE criterion.

Beyond Deterministic Autoencoders: Stochastic Encoders and Decoders

- When we trained supervised classification networks we usually assume that the network produces an output distribution $p(\boldsymbol{y}|\boldsymbol{x})$ and try to minimise the negative log-likelihood $-\log(p(\boldsymbol{y}|\boldsymbol{x}))$.
- In a decoder of an autoencoder we could do the same thing and have the decoder learn $p_{decoder}(\boldsymbol{x}|\boldsymbol{z})$ by minimising $-\log(p(\boldsymbol{x}|\boldsymbol{z}))$.
  - A linear output layer could parameterise the mean of a Gaussian distribution for real-valued $\boldsymbol{x}$; in this case the negative log likelihood yields the MSE criterion.
  - Binary $\boldsymbol{x}$ would correspond to a Bernoulli distribution parameterised by sigmoid outputs

Beyond Deterministic Autoencoders: Stochastic Encoders and Decoders

- When we trained supervised classification networks we usually assume that the network produces an output distribution $p(\boldsymbol{y}|\boldsymbol{x})$ and try to minimise the negative log-likelihood $-\log(p(\boldsymbol{y}|\boldsymbol{x}))$.
- In a decoder of an autoencoder we could do the same thing and have the decoder learn $p_{decoder}(\boldsymbol{x}|\boldsymbol{z})$ by minimising $-\log(p(\boldsymbol{x}|\boldsymbol{z}))$.
  - A linear output layer could parameterise the mean of a Gaussian distribution for real-valued $\boldsymbol{x}$; in this case the negative log likelihood yields the MSE criterion.
  - Binary $\boldsymbol{x}$ would correspond to a Bernoulli distribution parameterised by sigmoid outputs
  - Discrete (or categorical) $\boldsymbol{x}$ would correspond to a softmax distribution.

Beyond Deterministic Autoencoders: Stochastic Encoders and Decoders

- When we trained supervised classification networks we usually assume that the network produces an output distribution $p(\boldsymbol{y}|\boldsymbol{x})$ and try to minimise the negative log-likelihood $-\log(p(\boldsymbol{y}|\boldsymbol{x}))$.
- In a decoder of an autoencoder we could do the same thing and have the decoder learn $p_{decoder}(\boldsymbol{x}|\boldsymbol{z})$ by minimising $-\log(p(\boldsymbol{x}|\boldsymbol{z}))$.
    - A linear output layer could parameterise the mean of a Gaussian distribution for real-valued $\boldsymbol{x}$; in this case the negative log likelihood yields the MSE criterion.
    - Binary $\boldsymbol{x}$ would correspond to a Bernoulli distribution parameterised by sigmoid outputs
    - Discrete (or categorical) $\boldsymbol{x}$ would correspond to a softmax distribution.
- What about the encoder - could we make that output $p(\boldsymbol{z}|\boldsymbol{x})$?