# Reinforcement Learning, Autoencoders, and Transformers, Oh My!

Steve Avsec

Illinois Institute of Technology

April 1, 2024

# Overview

# Autoencoders

These are ”simple” two-layer neural networks meant to approximate the identity function.

# Autoencoders

These are ”simple” two-layer neural networks meant to approximate the identity function.

$$e : \mathbb{R}^N \to \mathbb{R}^n$$

and

$$d : \mathbb{R}^n \to \mathbb{R}^N$$

such that $id = d \circ e$

# Regularization

If we just use

$$L(\mathbf{x}, d, e) = \|\mathbf{x} - d \circ e(\mathbf{x})\|_2,$$

we are quite constrained what we can ”learn”.

# Regularization

If we just use

$$L(\mathbf{x}, d, e) = \|\mathbf{x} - d \circ e(\mathbf{x})\|_2,$$

we are quite constrained what we can "learn".

We can add a regularization constraint

$$L(\mathbf{x}) = \|\mathbf{x} - d \circ e(\mathbf{x})\|_p + \Omega(h)$$

# Regularization

If we just use

$$L(\mathbf{x}, d, e) = \|\mathbf{x} - d \circ e(\mathbf{x})\|_2,$$

we are quite constrained what we can "learn".

We can add a regularization constraint

$$L(\mathbf{x}) = \|\mathbf{x} - d \circ e(\mathbf{x})\|_p + \Omega(h)$$

This can allow us to capture more nuanced information even allowing $n > N$.

# Back to RNNs

Last week, we saw that an RNN can be used to map a sequence to a single output (or with a small generalization to a vector of fixed length.)

# Back to RNNs

Last week, we saw that an RNN can be used to map a sequence to a single output (or with a small generalization to a vector of fixed length.)

We can construct an RNN that takes a fixed vector and gives a sequence of arbitrary length.

- $\mathbf{a}_t = \mathbf{b} + W\mathbf{h}_{t-1} + U\mathbf{x}$
- $\mathbf{h}_t = g(\mathbf{a}_t)$
- $\mathbf{o}_t = \mathbf{c} + V\mathbf{h}_t$
- $\mathbf{y}_t = d(\mathbf{o}_t)$

# Back to RNNs

Last week, we saw that an RNN can be used to map a sequence to a single output (or with a small generalization to a vector of fixed length.)

We can construct an RNN that takes a fixed vector and gives a sequence of arbitrary length.

- $\mathbf{a}_t = \mathbf{b} + W\mathbf{h}_{t-1} + U\mathbf{x}$
- $\mathbf{h}_t = g(\mathbf{a}_t)$
- $\mathbf{o}_t = \mathbf{c} + V\mathbf{h}_t$
- $\mathbf{y}_t = d(\mathbf{o}_t)$

We can combine these to produce *encoder-decoder* sequence-to-sequence architecures.

# Context

The loss function in this case usually takes the form

$$L((y_j),(x_k)) = \log(P((y_j),(x_k)))$$

# Context

The loss function in this case usually takes the form

$$L((y_j),(x_k)) = \log(P((y_j),(x_k)))$$

The hidden vector **C** which is the output of the encoder/input to the decoder is commonly called the *context*.

# Context

The loss function in this case usually takes the form

$$L((y_j), (x_k)) = \log(P((y_j), (x_k)))$$

The hidden vector $\mathbf{C}$ which is the output of the encoder/input to the decoder is commonly called the *context*.

Having a fixed length vector as context has the drawback that arbitrary length sequences cannot be correctly encoded by fixed length

# Attention

To work around this limitation of a fixed context lenth, Badanau et al. (2015) introduced an attention mechanism.

# Attention

To work around this limitation of a fixed context lenth, Badanau et al. (2015) introduced an attention mechanism. The input sequence $(x_k)$ is mapped to a hidden sequence $(h_l)$ of ”annotations”.

# Attention

To work around this limitation of a fixed context lenth, Badanau et al. (2015) introduced an attention mechanism. The input sequence $(x_k)$ is mapped to a hidden sequence $(h_l)$ of "annotations".
The context vector $\mathbf{c}_j$ is "learned" as a weighted average

$$\mathbf{c}_j = \sum_l \alpha_{j,l} h_l$$

where the weights $\alpha_{j,l}$ are learned from a hidden state $s_j$ where

$$s_j = f(y_{j-1}, s_{j-1}, \mathbf{c}_j)$$

and

$$\alpha_{j,l} = \text{softmax}(a(s_{j-1}, h_l))$$

# Attention Continued

The model $a$ is a feedforward model called ”alignment”.

# Attention Continued

The model $a$ is a feedforward model called ”alignment”.

The hidden units $h_l$ are produced from a bidirectionl RNN to get context from both the forward sequence and backward.

# Attention Continued

The model $a$ is a feedforward model called ”alignment”.

The hidden units $h_l$ are produced from a bidirectionl RNN to get context from both the forward sequence and backward.

If this sounds complicated and hard to train, Google’s AI research team thought so too.

# High-Level

Transformers still work on an encoder-decoder type of premise, but use some clever architecture to avoid recurrent units.

# High-Level

Transformers still work on an encoder-decoder type of premise, but use some clever architecture to avoid recurrent units.

Encoder is 6 layers with two sub-layers each.

# High-Level

Transformers still work on an encoder-decoder type of premise, but use some clever architecture to avoid recurrent units.

Encoder is 6 layers with two sub-layers each.

Decoder is 6 layers of three sub-layers each.

# Multi-Head Attention

Define an *attention* function by

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^t}{\sqrt{d_k}} V)$$

# Multi-Head Attention

Define an *attention* function by

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^t}{\sqrt{d_k}}V)$$

And then a multi-head attention layer by

$$\text{MultiHead}(Q, K, V) = \text{Concat}(h_1, h_2, \ldots, h_l)W^o$$

where $h_j = \text{Attention}(QW_j^Q, KW_j^K, VW_j^V)$ and $W_j^Q$, $W_j^K$, $W_j^V$ are projection matrices.

# The Wedding Problem

Suppose there are $N$ people attending a wedding.

# The Wedding Problem

Suppose there are $N$ people attending a wedding.

Let $g_{i,j}$ be a random variable denoting that Person i and Person j get along (or not).

# The Wedding Problem

Suppose there are $N$ people attending a wedding.

Let $g_{i,j}$ be a random variable denoting that Person i and Person j get along (or not).

Suppose

$$P(g_{i,j} = 1) = P(g_{i,j} = -1) = \frac{1}{2}$$

and that

$$g_{i,j} = g_{j,i}$$

# The Wedding Problem

Suppose there are $N$ people attending a wedding.

Let $g_{i,j}$ be a random variable denoting that Person i and Person j get along (or not).

Suppose

$$P(g_{i,j} = 1) = P(g_{i,j} = -1) = \frac{1}{2}$$

and that

$$g_{i,j} = g_{j,i}$$

Suppose everyone has to be seated at one of two tables for dinner.

# A Hamiltonian

Let $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_N)$ denote a seating arrangement each $\sigma_j \in \{-1, 1\}$.

# A Hamiltonian

Let $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_N)$ denote a seating arrangement each $\sigma_j \in \{-1, 1\}$.
We can look to minimize

$$H(\sigma) = -\frac{1}{\sqrt{N}} \sum_{i<j} \sigma_j \sigma_i g_{i,j}$$

# A Hamiltonian

Let $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_N)$ denote a seating arrangement each $\sigma_j \in \{-1, 1\}$.
We can look to minimize

$$H(\sigma) = -\frac{1}{\sqrt{N}} \sum_{i<j} \sigma_j \sigma_i g_{i,j}$$

Gibbs measure: A probability measure on all configurations given by

$$G(\sigma) = \frac{e^{-\beta H(\sigma)}}{Z}$$

where

$$Z = \sum e^{-\beta H(\sigma)}$$

is taken over all configurations.

# Basic Structure

Given a graph (directed or undirected)$\mathcal{G}$, a *clique factor* $\varphi(\mathcal{C})$ is learned and unnormalized probabilities

$$\tilde{p}(\mathbf{x}) = \prod_{\mathcal{C}\in\mathcal{G}} \varphi(\mathcal{C})$$

# Basic Structure

Given a graph (directed or undirected)$\mathcal{G}$, a *clique factor* $\varphi(\mathcal{C})$ is learned and unnormalized probabilities

$$\tilde{p}(\mathbf{x}) = \prod_{\mathcal{C}\in\mathcal{G}} \varphi(\mathcal{C})$$

This is a very broad class of models encompassing the previous example.

# Basic Structure

Given a graph (directed or undirected)$\mathcal{G}$, a *clique factor* $\varphi(\mathcal{C})$ is learned and unnormalized probabilities

$$\tilde{p}(\mathbf{x}) = \prod_{\mathcal{C}\in\mathcal{G}} \varphi(\mathcal{C})$$

This is a very broad class of models encompassing the previous example.

There are few guarantees that anything converges or fits correctly.

# Variational Autoencoders

Idea: Learn a non-linear function $g$ such that if $\mathbf{z}$ is pulled from some latent known probability distribution, $p(\mathbf{x}; g(\mathbf{z}))$ accurately describes the distribution of $\mathbf{x}$.

# Variational Autoencoders

Idea: Learn a non-linear function $g$ such that if $\mathbf{z}$ is pulled from some latent known probability distribution, $p(\mathbf{x}; g(\mathbf{z}))$ accurately describes the distribution of $\mathbf{x}$.

Encoder is given by learning a distribution

$$q(\mathbf{z}|\mathbf{x})$$

and the decoder is just the distribution described above:

$$p(\mathbf{x}|\mathbf{z}) = p(\mathbf{x}; g(\mathbf{z}))$$

# Variational Autoencoders

Idea: Learn a non-linear function $g$ such that if $\mathbf{z}$ is pulled from some latent known probability distribution, $p(\mathbf{x}; g(\mathbf{z}))$ accurately describes the distribution of $\mathbf{x}$.

Encoder is given by learning a distribution

$$q(\mathbf{z}|\mathbf{x})$$

and the decoder is just the distribution described above:

$$p(\mathbf{x}|\mathbf{z}) = p(\mathbf{x}; g(\mathbf{z}))$$

Loss function:

$$\mathcal{L}(q) = E[\log(p(x|z))] - D_{KL}(q)||p(z)$$