Today

Autoencoders

Variational Autoencoders

Generative Adversarial Networks
Autoencoders

Using a CNN, try to “reconstruct” the input in the output.

Put a strong bottleneck on the network: This forces the network to learn a **compact representation** of the data.

The best thing about it - unsupervised learning. No need for annotations. Excellent for bootstrapping transfer learning.
Autoencoders

Input → Encoder
- Conv-Pool
- Strided-Conv

Features

Decoder
- De-conv (fractional)
- Un-pool

Reconstruction
Unpooling, Deconvolution

Figure 3. Illustration of deconvolution and unpooling operations.

“Sparse”...

“Densify”...

[Noh]
Unpooling, Deconvolution

Figure 4. Visualization of activations in our deconvolution network. The activation maps from (b) to (j) correspond to the output maps from lower to higher layers in the deconvolution network. We select the most representative activation in each layer for effective visualization. The image in (a) is an input, and the rest are the outputs from (b) the last $14 \times 14$ deconvolutional layer, (c) the $28 \times 28$ unpooling layer, (d) the last $28 \times 28$ deconvolutional layer, (e) the $56 \times 56$ unpooling layer, (f) the last $56 \times 56$ deconvolutional layer, (g) the $112 \times 112$ unpooling layer, (h) the last $112 \times 112$ deconvolutional layer, (i) the $224 \times 224$ unpooling layer and (j) the last $224 \times 224$ deconvolutional layer. The finer details of the object are revealed, as the features are forward-propagated through the layers in the deconvolution network. Note that noisy activations from background are suppressed through propagation while the activations closely related to the target classes are amplified. It shows that the learned filters in higher deconvolutional layers tend to capture class-specific shape information.
Autoencoders

No need for labeled data
Unsupervised

Input → Encoder - Conv-Pool - Strided-Conv → Features → Decoder - De-conv (fractional) - Un-pool → Reconstruction

Loss: $\|x - \hat{x}\|^2$
Autoencoders

After training AE: **Transfer Learning**

“Fine tune” backprop

Input

Label: “Cat”

- Conv-Pool
- Strided-Conv

Encoder

Features

Soft Max

“Cat”, p = 0.75
Autoencoders

Low-dim representations: “fur”, “eye”, “ear”, … & also reconstruction!

Can we use this to generate NEW data? Cats the world hasn't seen before?
Variational Autoencoders

Probabilistic approach to AE.

Latent variable $z$ (e.g.: “pose”, “eye?”, “fur?”) has PDF (based on some parametric model $\theta$).

Image $x$ is conditional on $z$.

We want to find the true $\theta$, given samples from the PDFs $p(z)$ and $p(x|z)$.

For $p(z)$ we can use simple Gaussian.

**But how to model the very complex $p(x|z)$?**

Hint: what do we have that goes from $z$ to $x$?
Variational Autoencoders

For $p(x|z)$ - We use a Decoder NN!

Now, we want to find the best $\theta$ s.t. the data $(x)$ likelihood is maximal ("cats that make most sense"). Essentially: *Train the model.*

Likelihood term for data:

$$p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz$$

**What’s the difficulty with this?**

Hint: think of the magnitude of $z$ and $x$
Variational Autoencoders

For $p(x|z)$ - We use a Decoder NN!

Now, we want to find the best $\theta$ s.t. the data $(x)$ likelihood is maximal ("cats that make sense"). Essentially: Train the model.

Likelihood:

$$p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz$$

**Intractable:** Can't compute all $p(x|z)$ for any $z$ (super highly dimensional, and expensive)
Variational Autoencoders

How to bypass intractability of data likelihood calculation? $p(z)$ is enormous...

Let’s try the posterior for $z$:

$$p_{\theta}(z|x) = p_{\theta}(x|z)p_{\theta}(z)/p_{\theta}(x)$$

Think - what’s the pixel-level distribution $p(x)$ of all images of cats?

Intractable! again...

Sample from true prior $p_{\theta^*}(z)

Sample from true conditional $p_{\theta^*}(x | z^{(i)})$
Variational Autoencoders

\[ p_{\theta}(z|x) = \frac{p_{\theta}(x|z)p_{\theta}(z)}{p_{\theta}(x)} \]

How did we deal with such complexity before?

With the power of...
Variational Autoencoders

Yes - Another DNN for the encoder.

The encoder will approximate \( p(z|x) \).

We call the density:

\[
q_\phi(z|x) \sim p_\theta(z|x)
\]

True \( p(z|x) \) remains intractable...

Decoder

Features

Sample from true prior
\( p_{\theta^*}(z) \)

Sample from true conditional
\( p_{\theta^*}(x \mid z^{(i)}) \)
Variational Autoencoders

Now we have a probabilistic framework for AEs.

Encoder and decoder model distributions explicitly (they give you mean and variance).

What do we know? What do we have on hand?
Hint: We’ve samples - which ones? We’ve made assumptions on some of the PDFs - which?

[Fei Fei Li]
Questions?
Variational Autoencoders

Solve for the (log) likelihood of the data $p(x)$:

$$\log p_{\theta}(x^{(i)}) = \mathbb{E}_{z \sim q_{\phi}(z|x^{(i)})} \left[ \log p_{\theta}(x^{(i)}) \right] \quad (p_{\theta}(x^{(i)}) \text{ Does not depend on } z)$$

Marginalize over all $z$'s.
Wait and see what happens...
Variational Autoencoders

Solve for the (log) likelihood of the data $p(x)$:

$$
\begin{align*}
\log p_\theta(x^{(i)}) &= E_{z \sim q_\phi(z|x^{(i)})} \left[ \log p_\theta(x^{(i)}) \right] \\
&= E_z \left[ \log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \right] \\
&\quad (p_\theta(x^{(i)}) \text{ Does not depend on } z) \\
&\quad (\text{Bayes' Rule})
\end{align*}
$$
Variational Autoencoders

Solve for the (log) likelihood of the data \( p(x) \):

\[
\log p_\theta(x^{(i)}) = \mathbb{E}_{z \sim q_\phi(z|x^{(i)})} \left[ \log p_\theta(x^{(i)}) \right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)
\]

\[
= \mathbb{E}_z \left[ \log \frac{p_\theta(x^{(i)} | z) p_\theta(z)}{p_\theta(z | x^{(i)})} \right] \quad (\text{Bayes’ Rule})
\]

\[
= \mathbb{E}_z \left[ \log \frac{p_\theta(x^{(i)} | z) p_\theta(z)}{p_\theta(z | x^{(i)})} \frac{q_\phi(z | x^{(i)})}{q_\phi(z | x^{(i)})} \right] \quad (\text{Multiply by constant})
\]
Variational Autoencoders

Solve for the (log) likelihood of the data $p(x)$:

$$
\log p_\theta(x^{(i)}) = \mathbb{E}_{z \sim q_\phi(z|x^{(i)})} \left[ \log p_\theta(x^{(i)}) \right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)
$$

$$
= \mathbb{E}_z \left[ \log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \right] \quad \text{(Bayes’ Rule)}
$$

$$
= \mathbb{E}_z \left[ \log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \right] \frac{q_\phi(z | x^{(i)})}{q_\phi(z | x^{(i)})} \quad \text{(Multiply by constant)}
$$

$$
= \mathbb{E}_z \left[ \log p_\theta(x^{(i)} | z) \right] - \mathbb{E}_z \left[ \log \frac{q_\phi(z | x^{(i)})}{p_\theta(z)} \right] + \mathbb{E}_z \left[ \log \frac{q_\phi(z | x^{(i)})}{p_\theta(z | x^{(i)})} \right] \quad \text{(Logarithms)}
$$

Analysis:

- **Decoder approx.**: Gives samples from this PDF
- $p(z)$ - Gaussian.
- $q(z|x)$ - Encoder approximates.
- $p(z|x)$ is intractable...
Variational Autoencoders

Solve for the (log) likelihood of the data $p(x)$:

$$\log p_\theta(x^{(i)}) = E_{z \sim q_\phi(z|x^{(i)})} \left[ \log p_\theta(x^{(i)}) \right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= E_z \left[ \log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})} \right] \quad \text{(Bayes’ Rule)}$$

$$= E_z \left[ \log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{q_\phi(z \mid x^{(i)})} \right] \quad \text{(Multiply by constant)}$$

$$= E_z \left[ \log p_\theta(x^{(i)} \mid z) \right] - E_z \left[ \log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)} \right] + E_z \left[ \log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})} \right] \quad \text{(Logarithms)}$$

Write KLDs:

$$= E_z \left[ \log p_\theta(x^{(i)} \mid z) \right] - D_{KL}(q_\phi(z \mid x^{(i)}) \parallel p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \parallel p_\theta(z \mid x^{(i)}))$$

We got this covered.

Intractable, but always $\geq 0$
Variational Autoencoders

How to train: \( \theta^*, \phi^* = \arg \max_{\theta, \phi} \sum_{i=1}^{N} \mathcal{L}(x^{(i)}, \theta, \phi) \)

Find the best parameters for encoder and decoder
s.t. the data likelihood is largest.

All we have is a lower bound... (the Variational Lower Bound)

Differentiable, we can use backprop.

Reconstruct using decoder
Bring approx posterior as close as possible to prior.

\[
\mathcal{L}(x^{(i)}, \theta, \phi) \rightarrow E_z \left[ \log p_{\theta}(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) \| p_{\theta}(z)) + D_{KL}(q_\phi(z | x^{(i)}) \| p_{\theta}(z | x^{(i)}))
\]
Variational Autoencoders

- **Input**
- **Encoder**
  - \( q_\phi(z|x) \rightarrow \mu_{z|x}, \Sigma_{z|x} \)
  - Run encoder. Get approx for \( z|x \) distribution.
- **Decoder**
  - \( p_\theta(z) \sim N(\mu, \Sigma) \)
- **Reconstruction**
Variational Autoencoders

Input → Encoder → \( z \sim N(\mu_{z|x}, \Sigma_{z|x}) \) → Decoder → Reconstruction

Sample \( z \) from approx \( z|x \)

\[
\mathbb{E}_z \left[ \log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))
\]
Variational Autoencoders

Input → Encoder → z → Decoder → Reconstruction

$p_\theta(x|z)$

Run decoder. Get approx $x|z$. 

$L(x^{(i)}, \theta, \phi) = \mathbb{E}_z \left[ \log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))$
Variational Autoencoders

Reconstruct from $x|z$ - Calculate reconstruction loss.

$$
\mathbb{E}_{z} \left[ \log p_{\theta}(x^{(i)} | z) \right] - D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\phi}(z))
$$

$\mathcal{L}(x^{(i)}, \theta, \phi)$
Variational Autoencoders

In TensorFlow...

```python
my_dense_layer = partial(
    tf.layers.dense,
    activation=tf.nn.elu,
    kernel_initializer=initializer)

X = tf.placeholder(tf.float32, [None, n_inputs])
hidden1 = my_dense_layer(X, n_hidden1)
hidden2 = my_dense_layer(hidden1, n_hidden2)
hidden3_mean = my_dense_layer(hidden2, n_hidden3, activation=None)
hidden3_sigma = my_dense_layer(hidden2, n_hidden3, activation=None)
noise = tf.random_normal(tf.shape(hidden3_sigma), dtype=tf.float32)
hidden3 = hidden3_mean + hidden3_sigma * noise
hidden4 = my_dense_layer(hidden3, n_hidden4)
hidden5 = my_dense_layer(hidden4, n_hidden5)
logits = my_dense_layer(hidden5, n_outputs, activation=None)
outputs = tf.sigmoid(logits)

xentropy = tf.nn.sigmoid_cross_entropy_with_logits(labels=X, logits=logits)
reconstruction_loss = tf.reduce_sum(xentropy)
latent_loss = 0.5 * tf.reduce_sum(
    tf.square(hidden3_sigma) + tf.square(hidden3_mean) - 1.0 -
    tf.log(eps + tf.square(hidden3_sigma)))
loss = reconstruction_loss + latent_loss
```
VAEs | Generate Data

Simply sample $z$ from $\mathcal{N}(0, I)$ and feedforward:

```python
codings_rnd = np.random.normal(size=[n_digits, n_hidden3])
outputs_val = outputs.eval(feed_dict={hidden3: codings_rnd})
```
VAEs | Generate Data

Data manifold for 2-d \( z \)

Vary \( z_1 \)

Vary \( z_2 \)

Degree of smile

Vary \( z_1 \)

Vary \( z_2 \)

Head pose
VAEs | Generate Data

32x32 CIFAR-10

Labeled Faces in the Wild

[Fei Fei Li]
Questions?
Variational Autoencoders

Explicit term for data likelihood:

$$p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz$$

Albeit, intractable, but we made some assumptions and approximations and VAEs are useful.

But we have to explicitly model the PDFs with a DNN.

This involves heavy approximation and output is blurry...

What to do???

Hint: can we relax some assumptions?
Generative Adversarial Networks

No explicit model of the PDF parameters (mean, variance).

We just want to take a random set of numbers and transform them into an image! (because we’re lazy like that)

Problem: the (output) image space is incredibly complex and high-dimensional...

What do we do with high dimensional complex PDFs?
Generative Adversarial Networks

What do we do with high dimensional complex PDFs?

**We use a DNN.**

But this is still very very hard to train.

We need a “trick”...
Generative Adversarial Networks

Enter: Game Theory.

Let’s play a game...

Random noise $\rightarrow$ Generator $\rightarrow$ Generated output $\rightarrow$ Discriminator $\rightarrow$ “Yes this is a cat.”

Training data
Generative Adversarial Networks

Minimax objective:

$$\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log (1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Discriminator output for real data $x$

Discriminator output for generated fake data $G(z)$

Random noise $Z$  \rightarrow Generator \rightarrow Generated output \rightarrow Discriminator \rightarrow “Yes this is a cat.”

Fei Fei Li

Training data
Generative Adversarial Networks

Minimax objective:

$$\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. **Gradient ascent** on discriminator

   $$\max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Gradient descent** on generator

   $$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$
Generative Adversarial Networks

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, $k$, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations do
  for $k$ steps do
    • Sample minibatch of $m$ noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
    • Sample minibatch of $m$ examples $\{x^{(1)}, \ldots, x^{(m)}\}$ from data generating distribution $p_{data}(x)$.
    • Update the discriminator by ascending its stochastic gradient:
      $$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D \left( x^{(i)} \right) + \log \left( 1 - D \left( G \left( z^{(i)} \right) \right) \right) \right].$$
  end for
  • Sample minibatch of $m$ noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
  • Update the generator by descending its stochastic gradient:
    $$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log \left( 1 - D \left( G \left( z^{(i)} \right) \right) \right).$$
end for

“Dual SGD” on the two objectives within a minibatch.
Generative Adversarial Networks

Once done training, just use Generator to make new data from random noise...

Random noise \rightarrow \text{Generator} \rightarrow \text{Generated output} \rightarrow \text{Discriminator} \rightarrow \text{“Yes this is a cat.”}

Training data
Deep Convolutional GANs (DCGANs)

[Radford]

Figure 3: Generated bedrooms after five epochs of training. There appears to be evidence of visual under-fitting via repeated noise textures across multiple samples such as the base boards of some of the beds.
GANs | Results

Deep Convolutional GANs (DCGANs)

[Radford]

Can do arithmetics!
GANs | Results

Deep Convolutional GANs (DCGANs)

[Radford]

Can do arithmetics!
GANs | Results

Interactive GANs

[Zhu]
GANs | Results

Image-to-Image

[Isola]
Wrap-Up

What we’ve seen today:

- Autoencoders
- Variational Autoencoders
- Generative Adversarial Networks