Mixtures

- *Mixture of f and g*:

$$p(x) = \eta f(x) + (1-\eta) g(x)$$

  Simplest, most common kind of latent variable model

- *Hidden variable represention*: Define $Z \sim \text{Bernoulli}(\eta)$ and

$$p(x) = \sum_{z=0,1} p(x \mid z)\, p(z)$$

  with $p(x \mid 0) = f(x)$, $p(x \mid 1) = g(x)$, $p(z) = \eta^{z}(1-\eta)^{(1-z)}$.

Bayesian Inference

The parameter θ of a model is viewed as a random variable. Inference usually carried out as follows:

- Choose a *generative model* $p(x \mid \theta)$ for the data.
- Choose a *prior distribution* $\pi(\theta)$ that expresses beliefs about the parameter before seeing any data.
- After observing data $\mathcal{D}_n = \{x_1, \ldots, x_n\}$, update beliefs and calculate the *posterior distribution* $p(\theta \mid \mathcal{D}_n)$.

Bayes' Theorem

The posterior distribution can be written as

$$p(\theta \mid x_1, \ldots, x_n) = \frac{p(x_1, \ldots, x_n \mid \theta)\pi(\theta)}{p(x_1, \ldots, x_n)} = \frac{\mathcal{L}_n(\theta)\pi(\theta)}{c_n} \propto \mathcal{L}_n(\theta)\pi(\theta)$$

where $\mathcal{L}_n(\theta) = \prod_{i=1}^n p(x_i \mid \theta)$ is the *likelihood function* and

$$c_n = p(x_1, \ldots, x_n) = \int p(x_1, \ldots, x_n \mid \theta)\pi(\theta)d\theta = \int \mathcal{L}_n(\theta)\pi(\theta)d\theta$$

is the normalizing constant, which is also called *evidence*.

Example

$X \sim \text{Bernoulli}(\theta)$ with data $\mathcal{D}_n = \{x_1, \dots, x_n\}$. Prior Beta$(\alpha, \beta)$ distribution

$$\pi_{\alpha,\beta}(\theta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\theta^{\alpha-1}(1-\theta)^{\beta-1}$$

Let $s = \sum_{i=1}^n x_i$ be the number of “successes.”

Posterior distribution $\theta \,|\, \mathcal{D}_n$ is Beta$(\alpha + s, \beta + n - s)$. Posterior mean is a mixture:

$$\bar{\theta} = \frac{\alpha + s}{\alpha+\beta+n} = \left(\frac{n}{\alpha+\beta+n}\right)\widehat{\theta} + \left(\frac{\alpha+\beta}{\alpha+\beta+n}\right)\theta_0$$

where $\widehat{\theta} = s/n$ is the MLE and $\theta_0 = \alpha/(\alpha+\beta)$ is the prior mean.

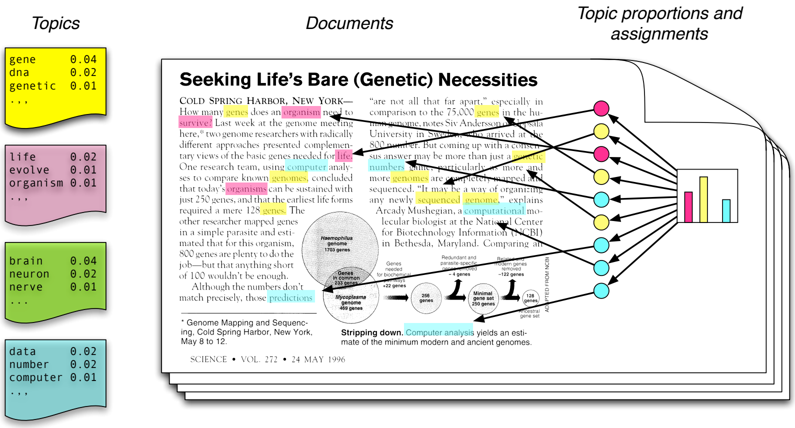
Dirichlet

Multinomial model with Dirichlet prior is generalization of the Bernoulli/Beta model.

$$\text{Dirichlet}_{\alpha}(\theta) = \frac{\Gamma(\sum_{j=1}^{K} \alpha_j)}{\prod_{j=1}^{K} \Gamma(\alpha_j)} \prod_{j=1}^{K} \theta_j^{\alpha_j - 1}$$

where $\alpha = (\alpha_1, \ldots, \alpha_K) \in \mathbb{R}_{+}^{K}$ is a non-negative vector.

Generative model for LDA



Topics
Documents
Topic proportions and assignments
gene 0.04
dna 0.02
genetic 0.01
...
life 0.02
evolve 0.01
organism 0.01
...
brain 0.04
neuron 0.02
nerve 0.01
...
data 0.02
number 0.02
computer 0.01
...
Haemophilus genome 1703 genes
Genes in common 233 genes
Mycoplasma genome 469 genes
Genes needed for biochemical pathways +22 genes
256 genes
Redundant and parasite-specific genes removed – 4 genes
Minimal gene set 250 genes
Related and modern genes removed –122 genes
128 genes
Ancestral gene set
ADAPTED FROM NCBI

Seeking Life's Bare (Genetic) Necessities

COLD SPRING HARBOR, NEW YORK— How many genes does an organism need to survive? Last week at the genome meeting here,* two genome researchers with radically different approaches presented complementary views of the basic genes needed for life. One research team, using computer analyses to compare known genomes, concluded that today's organisms can be sustained with just 250 genes, and that the earliest life forms required a mere 128 genes. The other researcher mapped genes in a simple parasite and estimated that for this organism, 800 genes are plenty to do the job—but that anything short of 100 wouldn't be enough.

Although the numbers don't match precisely, those predictions "are not all that far apart," especially in comparison to the 75,000 genes in the human genome, notes Siv Andersson of Uppsala University in Sweden, who arrived at the 800 number. But coming up with a consensus answer may be more than just a genetic numbers game, particularly as more and more genomes are completely mapped and sequenced. "It may be a way of organizing any newly sequenced genome," explains Arcady Mushegian, a computational molecular biologist at the National Center for Biotechnology Information (NCBI) in Bethesda, Maryland. Comparing an

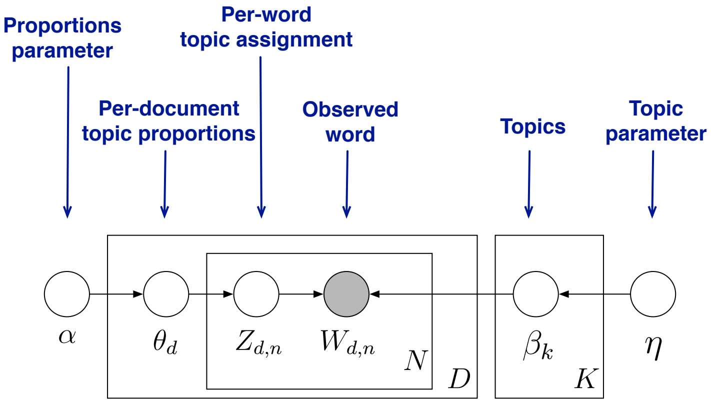
Stripping down. Computer analysis yields an estimate of the minimum modern and ancient genomes.

* Genome Mapping and Sequencing, Cold Spring Harbor, New York, May 8 to 12.

SCIENCE • VOL. 272 • 24 MAY 1996

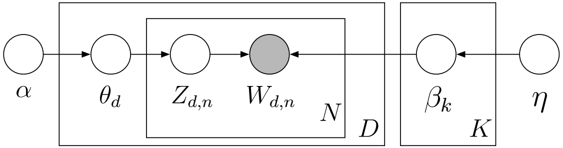
- Each **topic** is a distribution over words
- Each **document** is a mixture of corpus-wide topics
- Each **word** is drawn from one of those topics

LDA as a graphical model



- Nodes are random variables; edges indicate dependence.
- Shaded nodes are observed.
- Plates indicate replicated variables.

Posterior inference for LDA



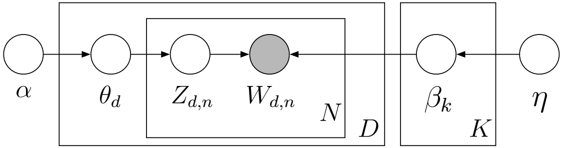
- The joint distribution of the latent variables and documents is

$$\prod_{i=1}^{K} p(\beta_i \mid \eta) \prod_{d=1}^{D} p(\theta_d \mid \alpha) \left(\prod_{n=1}^{N} p(z_{d,n} \mid \theta_d) p(w_{d,n} \mid \beta_{1:K}, z_{d,n}) \right).$$

- The posterior of the latent variables given the documents is

$$p(\beta_{1:K}, \theta_{1:D}, z_{1:D,1:N} \mid w_{1:D,1:N}).$$

Posterior inference for LDA

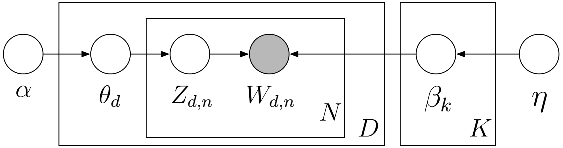


- This is equal to

$$\frac{p(\beta_{1:K}, \theta_{1:D}, \mathbf{z}_{1:D}, \mathbf{w}_{1:D})}{\int_{\beta_{1:K}} \int_{\theta_{1:D}} \sum_{\mathbf{z}_{1:D}} p(\beta_{1:K}, \theta_{1:D}, \mathbf{z}_{1:D}, \mathbf{w}_{1:D})}.$$

- We can't compute the denominator, the marginal $p(\mathbf{w}_{1:D})$.
- This is the crux of the inference problem.

Posterior inference for LDA

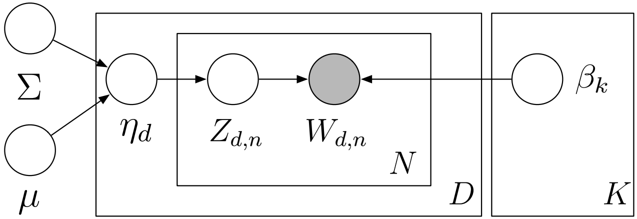


- There is a large literature on approximating the posterior.
- We will focus on
  - Gibbs sampling
  - Mean-field variational methods (batch and online)

Modeling richer assumptions

- Correlated topic model
- Dynamic topic model

The correlated topic model (CTM) (Blei and Lafferty, 2007)



Noconjugate prior
on topic proportions

- Draw topic proportions from a logistic normal, where topic occurrences can exhibit correlation.
- Use for:
  - Providing a “map” of topics and how they are related
  - Better prediction via correlated topics

Language models

- A language model is a way of assigning a probability to any sequence of words (or string of text)

$$p(w_1, \ldots, w_n)$$

- By the basic rules of conditional probability we can factor this as

$$p(w_1, \ldots, w_n) = p(w_1)p(w_2 \mid w_1) \ldots p(w_n \mid w_1, \ldots, w_{n-1})$$

- The number of *histories* grows as V^{n-1}. Number of parameters in model grows as V^n, where V is number of words in vocabulary.
- What are some ways of reducing the number of parameters?

Grouping histories

- We need to group histories.
- Let $\pi_n : V^n \to \mathcal{C}$ be a mapping from word sequences of length n to some finite set.
- Our model becomes

$$p(w_{n+1} \mid w_1, \ldots, w_n) = p(w_{n+1} \mid \pi_n(w_1, \ldots, w_n))$$

- Number of parameters: $O(V \cdot |\mathcal{C}|)$
- What are some example groupings?

Grouping histories

- Unigrams: $\pi(w_1, \ldots, w_n) = \emptyset$.
- Bigrams: $\pi(w_1, \ldots, w_n) = w_n$.
- Trigrams: $\pi(w_1, \ldots, w_n) = (w_{n-1}, w_n)$.
- Number of parameters grows as $O(V)$, $O(V^2)$, and $O(V^3)$, respectively.
- Number of parameters in topics for latent Dirichlet allocation:

$$O(K \cdot V)$$

Estimating parameters

- The maximum likelihood estimate of a trigram model:

$$\widehat{p}(w_3 \mid w_1, w_2) = \frac{\text{count}(w_1, w_2, w_3)}{\text{count}(w_1, w_2)}$$

- What are some problems with this model?

Bayesian inference

- The posterior mean under a Dirichlet prior would be

$$\widehat{p}(w_3 \mid w_1, w_2, \text{corpus}) \propto \text{count}(w_1, w_2, w_3) + \eta$$

Good-Turing

- Suppose we have a corpus of 500 million words. I count trigrams and find that 50 million of them are unique.
- When I see a new trigram, 10% of the time it won't have been seen before. (This is a typical number.)
- This means that the MLE is zero, and the probability that my model predicts the next word will be zero.
- The MLE is supported on the observed data. We need to spread out the probability over unseen events.

Class-based bigram model

- Model takes form

$$\begin{aligned} p(w_2 \mid w_1) &= p(\text{class}(w_2) \mid \text{class}(w_1))\, p(w_2 \mid \text{class}(w_2)) \\ &= p(c_2 \mid c_1)\, p(w_2 \mid c_2) \end{aligned}$$

- Use bottom-up agglomerative clustering to group the words.
- In each step, merge the pair of classes that gives the smallest reduction in likelihood of the data. (The MLE bigram model has the greatest likelihood.)
- $O(V^5)$ complexity to go down to $O(1)$ classes.
- V is number of words in vocabulary

Brown et al., "Class-based *n*-gram models of natural language"

Group globally, compute locally

- Clusters contain syntactic and semantic elements
- Surprising, since use local statistics only
- “A word is known by the company it keeps”

Perplexity

“perplexity” is evaluated as

$$\text{Perplexity}(\theta) = \left(\prod_{i=1}^{N} p_\theta(w_i \mid w_{1:i-1}) \right)^{-1/N}$$

If perplexity is 100, then the model predicts, on average, as if there were 100 equally likely words to follow.

Pointwise mutual information (PMI)

Related statistic is “pointwise mutual information” (PMI)

$$\log \left(\frac{p_{\text{near}}(w_1, w_2)}{p(w_1)p(w_2)} \right)$$

- How likely are specific words/clusters to co-occur together within some window, compared to if they were independent?

Shortcomings of word clusters

- Clusters are still “categorical”
- Can’t use vector space operations
- “One hot” representation wasteful
- These are addressed with *distributed representations* (next)

Insight of embeddings

- Use PMI-like scores to get embedding vectors.
- Can be applied whenever have large amounts of cooccurrence data.
- We'll go further into this next time

Pointwise mutual information (PMI)

Average mutual information

$$I(W_1, W_2) = \sum_{w_1, w_2} p(w_1, w_2) \log \frac{p(w_1, w_2)}{p(w_1)p(w_2)}$$

Pointwise mutual information (PMI)

$$\log \left(\frac{p_{\text{near}}(w_1, w_2)}{p(w_1)p(w_2)} \right)$$

- How likely are specific words/clusters to co-occur together within some window, compared to if they were independent?

Core idea of embeddings

- Form a language model but replace classes by vectors, one for each word
- Use PMI-like scores to fit the vectors
- Can be applied whenever have cooccurrence data.

Embedding LM

Model is

$$p(w_2 \mid w_1) = \frac{\exp(\phi(w_2)^T \phi(w_1))}{\sum_w \exp(\phi(w)^T \phi(w_1))}.$$

As before,

$$\begin{aligned} \ell(\phi) &= \sum_{w_1, w_2} p(w_1, w_2) \log p(\phi_2 \mid \phi_1) p(w_2 \mid \phi_2) \\ &= I(\Phi_1, \Phi_2) - H(W) \end{aligned}$$

Thus, we want embedding vectors with high mutual information.

Constructing embeddings

Carry out stochastic gradient descent over the embedding vectors $\phi \in \mathbb{R}^d$ (where $d \approx 50$–100 is chosen by trial and error)

This is what Mikolov et al. (2014, 2015) did at Google. With a couple of heuristics:

"Distributed representations of words," (2014) "Efficient representations of words in vector space" (2015)

Constructing embeddings

Heuristics used:

- Skip-gram: predict surrounding words from current word
- An issue with this is that it "over generates" the data. With text `the lazy brown fox jumped` we will have $p(\texttt{brown}|\texttt{lazy})$ and $p(\texttt{brown}|\texttt{fox})$
- Second is computational. The bottleneck is computing the denominator in the logistic (softmax) probability.
- Use "negative sampling": Approximation

$$\sum_{w} \exp(\phi(w)^T \phi(w_1))$$

$$\approx \exp(\phi(w_2)^T \phi(w_1)) + \sum_{\text{random } w} \exp(\phi(w)^T \phi(w_1))$$

Analogies

These heuristics enable training on very large text collections. Leads to vector representations of words with interesting properties.

For example, analogies:

`king` is to `man` as ? is to `woman`

`Paris` is to `France` as ? is to `Germany`

$$\phi(\texttt{king}) - \phi(\texttt{man}) \overset{?}{\approx} \phi(\texttt{queen}) - \phi(\texttt{woman})$$

$$\widehat{w} = \arg\min_{w} \|\phi(\texttt{king}) - \phi(\texttt{man}) + \phi(\texttt{woman}) - \phi(w)\|^2$$

Does $\widehat{w} = \texttt{queen}$?

GloVe

Shortly after: Stanford group introduced a computational expedient (with attempt to give a "principled" motivation)

$$\mathcal{O}(\phi) = \sum_{w_1, w_2} f(c_{w_1,w_2}) \left(\phi(w_1)^T \phi(w_2) - \log c_{w_1,w_2}\right)^2$$

where $c_{w,w'}$ are cooccurrence counts.

- A type of regression estimator. Can interpret/relate this to other objectives.
- Main advantage is that SGD can be carried out much more efficiently

Pennington et al., "GloVe: Global vectors for word representation," (2015)

GloVe

$$\mathcal{O}(\phi) = \sum_{w_1, w_2} f(c_{w_1,w_2}) \left(\phi(w_1)^T \phi(w_2) - \log c_{w_1,w_2} \right)^2$$

where $c_{w,w'}$ are cooccurrence counts.

- Heuristic weighting function

  $$f(x) = \left(\frac{x}{x_{\max}} \right)^{\alpha}$$

  where $\alpha = 3/4$ set empirically.
- So $10^{-4} \mapsto 10^{-3}$. Each order of magnitude down gets “boosted” by 1/4-magnitude.

Pennington et al., “GloVe: Global vectors for word representation,” (2015)

Embedding embeddings: t-SNE

- How can we visualize the embeddings?
- We’re in a very high dimensional space
- Can do PCA, but this will introduce an additional projection/approximation step
- Many visualization techniques exist. A currently popular one is t-SNE: “Student-t Stochastic Neighborhood Embedding”

t-SNE

Here's the idea behind t-SNE:

- Form a language model using the embeddings
- Scale and symmetrize, giving a matrix $P = [P_{ij}]$
- Represent word *i* by $y_i \in \mathbb{R}^2$. Use a heavy-tailed distribution (Student-t with one degree of freedom)
- Select y_i using stochastic gradient descent

Pronounced: **tee**-snee

Summary: Word embeddings

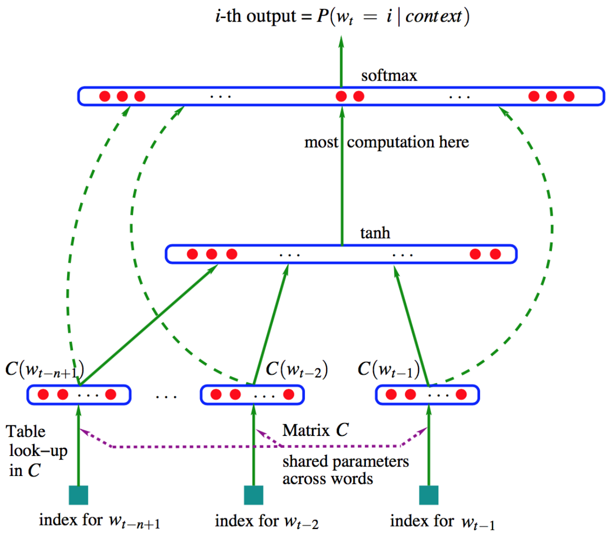
- Word embeddings are vector representations of words, learned from cooccurrence statistics
- The models can be viewed in terms of logistic regression and class-based bigram models
- Surprising semantic relations are encoded in linear relations
- Various heuristics have been introduced to get scalability
- Embeddings improve with more data
- t-SNE is an algorithm for visualizing embeddings

Neural LM: Idea

- Associate each word in vocabulary with a feature vector $C(w) \in \mathbb{R}^d$
- Express probabilities in terms of those vectors
- Form a big logistic regression to predict the next word. Can introduce some nonlinearites.
- Simultaneously learn vectors (word representations) and weightings (model parameters), using SGD

Bengio et al. “A neural probabilistic language model,” Journal of Machine Learning Research (2003).

Neural LM architecture



Neural LM: Simplified further!

Suppose

$$p(w_t \mid w_{t-1}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}}$$

with

$$\begin{aligned} y &= b + Ax \\ x &= C(w_{t-1}) \\ \text{so:}\quad y_w &= b_w + A_w^T C(w_{t-1}) \end{aligned}$$

Key property is that the word representations $C(w) \in \mathbb{R}^d$ are *learned* as part of the model.

This is the essence of the model. Note that we get two “embedding” vectors: A_w and $C(w)$.

Important aspects

- Unsupervised: No labels used, discovers useful features of input
- Compression: Code reduces dimension of data
- Lossy: Input won’t be reconstructed exactly
- Trained: The compression algorithm is learned for specific data

Simple autoencoder example

Encoder network of the form

$$h = \mathrm{ReLU}(Wx + b)$$

where $W \in \mathbb{R}^{H \times D}$ and $b \in \mathbb{R}^{H}$, decoder network is

$$\widehat{x} = \mathrm{ReLU}(\widetilde{W}h + \widetilde{b})$$

where $\widetilde{W} \in \mathbb{R}^{D \times H}$ and $\widetilde{b} \in \mathbb{R}^{D}$.

Objective function:

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} \|x_i - \mathrm{ReLU}(\widetilde{W}\,\mathrm{ReLU}(Wx_i + b) + \widetilde{b})\|^2.$$

$\mathrm{ReLU}(x) = \max(0, x)$, applied component-wise.

Simple example – code of dimension K

Encoder network of the form

$$\begin{aligned} h &= \mathrm{ReLU}(W_1 x + b_1) \\ c &= \mathrm{ReLU}(W_2 h + b_2) \end{aligned}$$

where $W_1 \in \mathbb{R}^{H \times D}$ and $b_1 \in \mathbb{R}^{H}$, and $W_2 \in \mathbb{R}^{K \times H}$ and $b_2 \in \mathbb{R}^{K}$

Decoder network of the form

$$\begin{aligned} \widehat{h} &= \mathrm{ReLU}(\widetilde{W}_1 c + \widetilde{b}_1) \\ \widehat{x} &= \mathrm{Sigmoid}(\widetilde{W}_2 \widehat{h} + \widetilde{b}_2) \end{aligned}$$

where $\widetilde{W}_1 \in \mathbb{R}^{H \times K}$ and $\widetilde{b}_1 \in \mathbb{R}^{H}$, and $\widetilde{W}_2 \in \mathbb{R}^{D \times H}$ and $\widetilde{b}_2 \in \mathbb{R}^{D}$

Objective function: binary cross-entropy

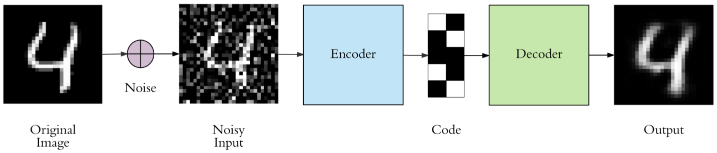
$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} \left(x_i \log \widehat{x}_i + (1 - x_i) \log(1 - \widehat{x}_i) \right)$$

Adam optimizer

- Variant of stochastic gradient descent where separate learning rate (step size) is maintained for each network weight (parameter)
- Each step size adapted as learning progresses based on moments of the derivatives

"Adam: A method for stochastic optimization," D. Kingma and J. Ba, `https://arxiv.org/abs/1412.6980`

Variant: Denoising autoencoder



- Feed in noisy data
- Train to match to denoised data

Variant: Sparse autoencoder

- Add penalty to encourage sparsity
- Forces autoencoder to discover interesting structure in data
- Same principle as sparse coding

In Keras this is easy to do:

```
code = Dense(code_size, activation='relu',
activity_regularizer=l1(10e-6))(input_img)
```

Variational autoencoders

Start with a generative model

$$z \sim N(0, I_K)$$
$$x \mid z \sim \mathrm{Sigmoid}(G(z))$$

$G(z)$ is the *generator network* or *decoder*

As a simple example:

$$G(z) = A_2 \mathrm{ReLU}(A_1 z + b_1) + b_2$$

Déjà vu all over again

We saw this before—almost.

Simple autoencoder

$$\|x - \mathrm{ReLU}(A\,\mathrm{ReLU}(Bx + d) + b)\|^2$$

Variational autoencoder

$$\frac{1}{N}\sum_{s=1}^{N}\frac{1}{2}\|x - \mathrm{ReLU}(A\,(\mathrm{ReLU}(Bx + d) + \epsilon_s) + b)\|^2$$

But the perspective and thinking have changed significantly. We're back to statistics!

Summary: What did we learn today?

- Autoencoders compress the input and then reconstruct it
- Bottleneck forces extraction of useful features
- Will overfit and “memorize” the data
- Overfitting mitigated by denoising autoencoders
- More fundamental view: Latent variable generative models and posterior inference
- Parameterize variational parameters in terms of neural networks
- Reparameterization trick allows simultaneous training of both networks

S&DS 355 / 555

Introductory Machine Learning

Some Notes on Backpropagation

Backpropagation is—at the heart of it—nothing more than the chain rule from high school calculus. It boils down to the following fact:

If $\mathcal{L}$ is a scalar function and $A = BC$, then

$$\frac{\partial \mathcal{L}}{\partial B} = \frac{\partial \mathcal{L}}{\partial A} C^T \tag{1}$$

$$\frac{\partial \mathcal{L}}{\partial C} = B^T \frac{\partial \mathcal{L}}{\partial A}. \tag{2}$$

This can be shown directly using the "usual" chain rule. You should check that the dimensions match up!

The function $\mathcal{L}$ is our loss function. The use of this is called "backpropagation" because we start with the derivatives in the last layer of the network, and recursively send these "back" to compute the derivatives in the earlier part of the network.

So, if $f = Wx + b$ then

$$\frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}}{\partial f} x^T. \tag{3}$$

If the loss $\mathcal{L}$ is squared error $\mathcal{L} = \frac{1}{2}(y - f)^2$ then

$$\frac{\partial \mathcal{L}}{\partial f} = (f - y). \tag{4}$$

Now, if

$$f = W_2 h + b_2 \tag{5}$$

$$h = W_1 x + b_1 \tag{6}$$

then

$$\frac{\partial \mathcal{L}}{\partial W_2} = \frac{\partial \mathcal{L}}{\partial f} h^T = (f - y) h^T \tag{7}$$

$$\frac{\partial \mathcal{L}}{\partial h} = W_2^T \frac{\partial \mathcal{L}}{\partial f} = W_2^T (f - y) \tag{8}$$

$$\frac{\partial \mathcal{L}}{\partial b_2} = \frac{\partial \mathcal{L}}{\partial f} = (f - y). \tag{9}$$

Then, we have that

$$\frac{\partial \mathcal{L}}{\partial W_1} = \frac{\partial \mathcal{L}}{\partial h} x^T = W_2^T \frac{\partial \mathcal{L}}{\partial f} x^T = W_2^T (f - y) x^T \tag{10}$$

$$\frac{\partial \mathcal{L}}{\partial b_1} = \frac{\partial \mathcal{L}}{\partial h} = W_2^T (f - y). \tag{11}$$

This is just another linear model, so it will train to be equivalent to least squares regression. But if we add a ReLU nonlinearity, we get the two-layer neural network

$$h = \text{ReLU}(W_1 x + b_1) \tag{12}$$

$$f = W_2 h + b_2. \tag{13}$$

Let the loss for an example (x, y) be $\mathcal{L} = \frac{1}{2}(y - f(x))^2$. From the above calculations we then have for the second layer

$$\frac{\partial \mathcal{L}}{\partial W_2} = \frac{\partial \mathcal{L}}{\partial f} h^T \tag{14}$$

$$\frac{\partial \mathcal{L}}{\partial h} = W_2^T \frac{\partial \mathcal{L}}{\partial f} \tag{15}$$

$$\frac{\partial \mathcal{L}}{\partial b_2} = \frac{\partial \mathcal{L}}{\partial f}. \tag{16}$$

For the hidden layer we have

$$\frac{\partial \mathcal{L}}{\partial W_1} = \mathbb{1}(h > 0) \frac{\partial \mathcal{L}}{\partial h} x^T \tag{17}$$

$$= \mathbb{1}(h > 0) W_2^T \frac{\partial \mathcal{L}}{\partial f} x^T \tag{18}$$

$$\frac{\partial \mathcal{L}}{\partial b_2} = \mathbb{1}(h > 0) \frac{\partial \mathcal{L}}{\partial h} \tag{19}$$

$$= \mathbb{1}(h > 0) W_2^T \frac{\partial \mathcal{L}}{\partial f}. \tag{20}$$

For a two-layer network *for classification* we have

$$h = \mathrm{ReLU}(W_1 x + b_1) \tag{21}$$
$$f = W_2 h_1 + b_2 \tag{22}$$
$$p = \mathrm{Softmax}(f) \tag{23}$$

where again W_j and b_j are matrices or vectors of the appropriate dimensions. Let the loss for an example (x, y) be the log-loss $\mathcal{L} = -\log p(y \mid x)$. Then

$$\frac{\partial \mathcal{L}}{\partial f} = \begin{pmatrix} p_1 - \mathbb{1}(y = 1) \\ p_2 - \mathbb{1}(y = 2) \\ p_3 - \mathbb{1}(y = 3) \end{pmatrix} \in \mathbb{R}^3 \tag{24}$$

Backpropagating to the second layer we have

$$\frac{\partial \mathcal{L}}{\partial W_2} = \frac{\partial \mathcal{L}}{\partial f} h_1^T \tag{25}$$
$$\frac{\partial \mathcal{L}}{\partial h} = W_2^T \frac{\partial \mathcal{L}}{\partial f} \tag{26}$$
$$\frac{\partial \mathcal{L}}{\partial b_2} = \frac{\partial \mathcal{L}}{\partial f} \tag{27}$$

Then backpropagating to the first layer

$$\frac{\partial \mathcal{L}}{\partial W_1} = \mathbb{1}(h > 0) \frac{\partial \mathcal{L}}{\partial h} x^T \tag{28}$$
$$= \mathbb{1}(h > 0) W_2^T \frac{\partial \mathcal{L}}{\partial f} x^T \tag{29}$$
$$\frac{\partial \mathcal{L}}{\partial b_1} = \mathbb{1}(h > 0) \frac{\partial \mathcal{L}}{\partial h} \tag{30}$$
$$= \mathbb{1}(h > 0) W_2^T \frac{\partial \mathcal{L}}{\partial f}. \tag{31}$$

We'll leave it to you to extend the calculations (and implementation!) to a three-layer network, if you are so-inclined.