Chapter One

1. AI.

① want to create machines that think

② early days of AI tackled problems that can be described by mathematical rules.

③ true challenge to AI is solving tasks that we solve intuitively.

④ solution is allowing system to learn from experience and understand world in terms of a hierarchy of concepts.

2 Evolve.

① knowledge base: hard-coded knowledge

② machine learning:
system has ability to aquire their own knowledge.

③ representation learning: learn representation

④ deep learning:
learn a hierarchy of representation.
i.e. learn right representation, and learn composition of representation.

3. history.

3.1

① cybernetics: 1940s-1960s

① neural perspective of ann:
i) duplicate brain functionality.
ii) interesting to understand brain.

② dl: not simulate brain, but build system able to solve AI tasks.

③ computational neuroscience:
build brain model.

3.2 connectionism: 1980s-1990s.

① idea: a large number of simple computational units can achieve intelligent behavior.

② distributed representation:
each input should be represented by many features, and each feature should be involved in the representation of many possible inputs.

③ Since 1995, kernel machines, graphical models led to decline in the popularity of deep learning.

3.3. deep learning: 2006 - today

success reason: have resources deep learning can succeed.

① "Big Data" gives more data.

② hardware, software allow larger model size.

5.4. A is real symmetric
$\Rightarrow A$ has eigendecomposition
$A = Q \Lambda Q^T$
Q is orthogonal matrix composed of eigenvectors of A.
Λ is diagonal, Λ_{ii} is eigenvalue.

5.5. A, $\{\lambda_i\}$ are eigenvalue of A.
if $\lambda_i > 0, \forall i$, then A is called positive definite.

6. Singular Value Decomposition
$\forall A$ is matrix $\Rightarrow A$ has SVD
$\not\Rightarrow A$ has eigendecomposition.

6.2 $A_{m \times n} = U D V^T$ is called SVD of A.
U, V are orthogonal,
$D \in R^{m \times n}$ is diagonal.

7. trace: $Tr(A) := \sum_i A_{ii}$
determinant of square A:
$det(A) := \lambda_1 \cdots \lambda_n$.

8. We use decomposition to represent A in other form, so we can see some different things.

Frobenius norm:

Chapter 2 linear algebra

$\|A\|_F = \sqrt{\sum_{i,j} A_{ij}^2}$

4.1 unit vector x
$\Leftrightarrow \|x\|_2 = 1$

4.2 vector x, y are orthogonal to each other
$\Leftrightarrow x^T y = 0$

4.3 orthonormal of vectors
$\Leftrightarrow$ they are unit vector, and orthogonal to each other?

4.4 A is orthogonal matrix
$\Leftrightarrow A^T A = A A^T = I$

II. eigendecomposition

5.1. $12 = 2 \times 2 \times 3$.
this expression tells us some property about 12.

5.2. v is eigenvector of A
$\Leftrightarrow v \neq 0, Av = \lambda v$.

5.3. eigendecomposition of A
$\Leftrightarrow A = V diag(\lambda) V^{-1}$

2.3 $\sum_i c_i v^{(i)}$ is called linear combination of set $\{v^{(i)}\}$

2.4 set{linear combination of set V}
is called span of V

2.5 a set $V = \{v^{(i)}, i = 1, \cdots, n\}$
is linear independent
$\Leftrightarrow$ no vector in V is a linear combination of the other vectors.

2.6 A is singular matrix
$\Leftrightarrow A$ is squar matrix with linear dependent columns.

2.7. left inverse: $A^{-1} A = I$

3. f is called a norm, if
$f(x) \geq 0, \forall x$
$f(x) = 0 \Rightarrow x = 0$
$f(x+y) \leq f(x) + f(y)$
$\forall \alpha \in R, f(\alpha x) = |\alpha| f(x)$

3.1 L^p norm: $\|x\|_p = (\sum_i |x_i|^p)^{\frac{1}{p}}, p \geq 1$
L^∞ norm: $\|x\|_\infty := \max_i |x_i|$

1. definition

1.1 $f(x_1, \cdots, x_n)$, where $x_i \in Z^+$.
we write f in a table-like form
scalar: $n=1, x_1 = 1$.
vector: $n=1, x_1 \in \{1, 2, \cdots\}$
matrix: $n=2$,
tensor: $n \geq 3$.

1.2 $g(x, y) := f(y, x)$
g is called transpose of matrix f.

1.3 matrix product
$C = AB$, where $C_{ij} = \sum_k A_{ik} B_{kj}$

1.4 Hadmard product
$C = A \odot B$, where $C_{ij} = A_{ij} \cdot B_{ij}$

2. $A \in R^{m \times n}$, $b \in R^m$

2.1 $Ax = b$ has solution
$\Leftrightarrow b$ in collumn space of A.
or say range of A.

2.2 $\forall b \in R^m$, $Ax = b$ has solution
$\Leftrightarrow A$ has m linear independent collumns

Chapter 3. Probability and Info theory

1. random variable X is a variable that can take on different values randomly.
2. for discrete variables, we describe probability distribution P using probability mass function (PMF), for continuous variables, we use probability density function. (PDF).
3. marginal distribution is the probability distribution over a subset when we know the probability distr. over a set of variables
4. conditional probability is the pr. of some event given that some other event has happened.

$$P(Y=y|X=x) = \frac{P(Y=y, X=x)}{P(X=x)}$$

5. chain rule or product rule of pr., which can get from definition of conditional pr.: is

$$P(X^{(1)}, \cdots, X^{(n)}) = P(X^{(1)}) \prod_{i=2}^{n} P(X^{(i)} | X^{(1)}, \cdots, X^{(i-1)})$$

6. X and Y are independent, $X \perp Y$

$$\Leftrightarrow p(X=x, Y=y) = p(X=x)p(Y=y) \quad \forall x \in X, y \in Y$$

X and Y conditional independent given Z, $X \perp Y | Z$

$$\Leftrightarrow p(X=x, Y=y|Z=z) = p(X=x|Z=z)p(Y=y|Z=z) \quad \forall x \in X, y \in Y, z \in Z.$$

7. expectation of function $f(x)$ with respect to pr. distr. $P(x)$:

$$E_{x \sim P}[f(x)] = \sum_x P(x)f(x)$$

7.1 variance:

$$Var(f(x)) = E[(f(x) - E[f(x)])^2]$$

7.2 standard deviation: $\sqrt{Var(f(x))}$

7.3 covariance

$$Cov(f(x), g(y)) = E[(f - Ef)(g - Eg)]$$

7.4 covariance matrix of a random vector $x \in \mathbb{R}^n$ is an $n \times n$ matrix.

$$Cov(X)_{ij} = Cov(X_i, X_j)$$

8. common pr. di.

8.1 Bernoulli distribution is a distribution over a single binary random variable, It's controlled by $\phi \in [0,1]$.

$$P(X=1) = \phi, \quad P(X=0) = 1 - \phi.$$

8.2 multinoulli or categorical dis. is a dis. over a single discrete variable with k different state, where k is finite. It's parametrized by $p \in [0,1]^{k-1}$

8.3 normal dis. or Gaussian dis.

$$\mathcal{N}(x; \mu, \sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad \forall x \in \mathbb{R}$$

where $\mu \in \mathbb{R}$, $\sigma \in (0, \infty)$

8.4 exponential distribution

$$p(x; \lambda) = \lambda \mathbb{1}_{x \geq 0} e^{-\lambda x}, \quad \forall x \in \mathbb{R}$$

8.5 Laplace dis.

$$Laplace(x; \mu, \gamma) = \frac{1}{2\gamma} e^{-\frac{|x-\mu|}{\gamma}}$$

8.6 Dirac delta distribution

$p(x) = \delta(x - \mu)$, where $\delta(x)$ is Dirac f. $X = (X_1, \cdots, X_n)$

9. Common functions.

9.1 logistic ~~function~~ sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}, \quad \forall x \in \mathbb{R}$$

9.2 softplus function

$$\zeta(x) = \log(1 + e^x), \quad \forall x \in \mathbb{R}$$

10. Bayes' rule

$$P(X|Y) = \frac{P(X)P(Y|X)}{P(Y)}$$

11. Information theory

11.1 self-information of an event $X=x$ is $I(x) = -\log P(x)$

11.2 Shannon entropy

$$H(P) = H(X) = E_{X \sim P}[I(x)]$$

11.3 KL divergence

$$D_{KL}(P||Q) = E_{X \sim P}\left[\log \frac{P(x)}{Q(x)}\right]$$

11.4 cross-entropy

$$H(P, Q) = H(P) + D_{KL}(P||Q)$$

12. when the factorization of a probability dis. is represented with a graph, we call it a graphical model.

8.2. $f''(x_0)<0$

$\Rightarrow f'(x)\downarrow$ at x_0

$\therefore f(x)$ concave at x_0

likewise, $f''(x_0)>0 \Rightarrow f(x)$ convex at x_0.

9. $f:\mathbb{R}^n\to\mathbb{R}$, then Hessian matrix is H, $H_{ij}=\frac{\partial^2 f(x)}{\partial x_i \partial x_j}$

9.2 directional second derivative

d^THd, $\|d\|=1$.

9.3 directional second derivative tells us how well we can expect a gradient descent step to perform.

9.4 $f(x)\approx f(x^{(0)})+(x-x^{(0)})^Tg+\frac{1}{2}(x-x^{(0)})^TH(x-x^{(0)})$

where g is gradient, H is Hessian at $x^{(0)}$.

10. constrained optimization.

if at least one feasible point exists,

f is not permitted to have value ∞,

then $\min_{x\in S} f(x) \Leftrightarrow \min_x \max_\lambda \max_{\alpha\geq 0} L(x,\lambda,\alpha)$

where $S=\{x \mid g^{(i)}(x)=0,\ h^{(j)}(x)\leq 0,\ \forall i,j\}$

$L=f(x)+\sum_i \lambda_i g^{(i)}(x)+\sum_j \alpha_j h^{(j)}(x)$

5. $\because$ we get minimum for this u, $\therefore$ we downhill most (using 4.4).

6.3. gradient descent:

decrease f by moving in the direction of the negative gradient.

4.4. $f(x_2)=f(x_1)$, $f'(x_2)<f'(x_1)$

$\Rightarrow \exists \varepsilon>0$, such that

$f'(x_2)+\varepsilon<f'(x_1)-\varepsilon$,

for this ε, $\exists \delta>0$, $\forall 0<\alpha<\delta$,

we have

$$\begin{cases} \frac{f(x_2+\alpha)-f(x_2)}{\alpha} < f'(x_2)+\varepsilon \\ \frac{f(x_1+\alpha)-f(x_1)}{\alpha} > f'(x_1)-\varepsilon \end{cases}$$

$\therefore f(x_2+\alpha)<f(x_1+\alpha)$

7. Jacobian matrix.

$f:\mathbb{R}^m\to\mathbb{R}^n$

then $J\in\mathbb{R}^{n\times m}$, $J_{ij}=\frac{\partial f(x)_i}{\partial x_j}$

8. second derivative tells us how first derivative will change as we vary the input.

4.3. gradient descent: Chapter 4

reduce f by moving x in small steps with opposite sign of the derivative.

5. directional derivative

$f:\mathbb{R}^n\to\mathbb{R}$, $\frac{\partial f(x+\alpha u)}{\partial \alpha}\Big|_{\alpha=0}$

$=u^T\nabla_y f(y)\big|_{\alpha=0}$, $y=x+\alpha u$

$=u^T\nabla_x f(x)$.

where $\|u\|_2=1$.

6. $\min_{\|u\|=1} u^T\nabla_x f(x)$

$\Leftrightarrow \min_{\|u\|=1} \|\nabla_x f(x)\|\cos\theta$.

$\therefore$ if $u=-\frac{\nabla_x f(x)}{\|\nabla_x f(x)\|}$, we get minimum.

6.2 $\because \frac{\partial f(x+\alpha u)}{\partial \alpha}<0$ if we move in direction $u=-\frac{\nabla_x f}{\|\nabla_x f\|}$

$\therefore$ from 4.2, we downhill for small enough α.

1. on digital computer, we represent numerical computation infinitely many real numbers with a finite number of bit patterns, incurring approximation error.

2. rounding error: underflow, overflow.

3. condition number of matrix A

$\max_{i,j}\frac{|\lambda_i|}{|\lambda_j|}$

when it is large, $f(x)=A^{-1}x$ is sensitive to error in the input.

4. $f'(x_0)$ exists

$\Leftrightarrow \forall \varepsilon>0, \exists \delta>0, \forall x\in O(x_0,\delta)$,

we have $\frac{f(x)-f(x_0)}{x-x_0}\in O(f(x_0),\varepsilon)$

4.1. $f'(x_0)>0 \Rightarrow \exists \delta>0, \forall x\in O(x_0,\delta)$,

$f(x)\uparrow$ only at x_0.

$f'(x_0)<0 \Rightarrow \exists \delta>0, \forall x\in O(x_0,\delta)$,

$f(x)\downarrow$ at x_0.

4.2 $f(x_0-\alpha\,\mathrm{sign}(f'(x_0)))<f(x_0)$ for small enough α; $\alpha>0$.

Chapter 5 Machine learning basics.

1. learning algorithms

algorithm is said to learn from experience E with respect to task T and performance measure P, if P at T improves with E.

1.2 a maching learning algorithm is an algorithm that is able to learn from data.

2. the central challenge in machine learning is that we must generalize well.

3. underfitting: training error is not small.

overfitting: gap between training and test error is not small.

3.2 The factors determining how well a machine learning algorithm will perform are its ability not to underfit and overfit.

4. we can control whether a model is more likely to overfit or underfit by altering its capacity.

4.2 model specifies which family of functions the learning algorithm can choose from, this is called the representational capacity of the model.

4.3 learning algorithm's effective capacity may be less than representational capacity of model, due to limitations of the capabilities of the optimization algorithm.

4.4 Occam's razor

among competing hypotheses that explain known observations equally well, one should choose the simplest one.

4.5 statistical learning theory provides various means of quantifying model capacity, such as VC dimension.

4.6. VC dimension of a binary classifier is defined as being the largest possible value of m for which there exists a training set of m different points that the classifier can label arbitrarily.

4.7. the discrepancy between training error and generalization error is bounded from above by a quantity that grows as the model capacity grows but shrinks as the number of training examples increases.

5. parametric models learn a function described by a parameter vector whose size is finite and fixed before any data is observed. Non-parametric models have no such limitation.

6. The error incurred by an oracle making predictions from the true distribution $p(x, y)$ is called the Bayes error.

7. no free lunch theorem.

Averaged over all possible data generating distributions, every classification algorithm has the same error rate when classifying previously unobserved points.

8. We must design our machine algorithms to perform well on a specific task. (from 7).

generating distribution, $p_{model}(x;\theta)$ model distribution.

12.1 maximizing the likelihood is equivalent to minimization of cross entropy.

12.2. from maximum likelihood principle we can derive specific functions that are good estimators for different models.

12.3 under appropriate conditions, maximum likelihood estimator is consistent.

13.1 frequentist perspective:

true parameter θ is fixed but unknown.

point estimate $\hat{\theta}$ is a function of dataset.

Here dataset is random, so $\hat{\theta}$ is r.v..

13.2 Bayesian perspective:

dataset is observed and is not random.

true parameter θ is unknown or uncertain and thus is a r.v.

14. The Bayesian uses probability to reflect degrees of certainty of state of knowledge.

11.1. bias of an estimator

$$bias(\hat{\theta}_m) = E(\hat{\theta}_m) - \theta$$

where expectation is over the data.

θ is true underlying value of θ used to define the data generating distr..

11.2 variance of an estimator is simply the variance $Var(\hat{\theta})$.

standard error: $SE(\hat{\theta}) = \sqrt{Var(\hat{\theta})}$

11.3 MSE of an estimator

$$\begin{aligned} MSE &= E[(\hat{\theta}_m - \theta)^2] \\ &= E\hat{\theta}_m^2 - (E\hat{\theta}_m)^2 + (E\hat{\theta}_m)^2 - 2E\hat{\theta}_m\theta + \theta^2 \\ &= Var(\hat{\theta}_m) + (Bias\,\hat{\theta})^2 \end{aligned}$$

12. maximum likelihood estimator

$$\begin{aligned} \theta_{ML} &= \arg\max_\theta p_{model}(X;\theta) \\ &= \arg\max_\theta E_{x\sim\hat{p}_{data}} \log p_{model}(x;\theta) \\ &= \arg\min_\theta D_{KL}(\hat{p}_{data} \| p_{model}) \\ &= \arg\min_\theta H(\hat{p}_{data} \| p_{model}) \end{aligned}$$

where X is training data, $\hat{p}_{data}$ empirical distribution defined by X, p_{data} true data

9. Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.

10. hyperparameters are settings that we can use to control the behavior of a learning algorithm. The values of hyperparameters are not adapted by the learning algorithm itself.

10.2 validation set is used to "train" the hyperparameters.

training error < validation error < generalization error.

10.3 cross-validation.

repeating training and testing computation on different randomly chosen subsets or splits of the original dataset.

11. statistic $\hat{\theta}_m$ is a function of data.

'cause data is random, it is a r.v.

$\hat{\theta}_m = g(x^{(1)}, \cdots, x^{(m)})$

18. machine learning algorithm recipe:

dataset, cost function, optimization procedure, a model.

18.1 linear regression algorithm.

dataset $\{X, y\}$

cost function: $J(w,b) = -E_{x,y\sim\hat{p}_{data}} \log p_{model}(y|x)$

model: $p_{model}(y|x) = \mathcal{N}(y; x^T w + b, 1)$

optimization algorithm: solve $\min J(w,b)$ using normal equations.

19.1 curse of dimensionality

machine learning problems become exceedingly difficult when the number of dimensions in the data is high.

19.2 The exponential advantages conferred by the use of deep, distributed representations counter the exponential challenges proposed by the curse of dimensionality.

20. manifold learning algorithms assum that the data lies along a low-dimensional manifold.

15.4 SVM applies $\phi(x)$ to all inputs, then learns a linear model in the new transformed space. It does not provide probability, but only outputs a class identity.

$f(x) = b + \sum_i \alpha_i k(x, x^{(i)})$

15.5 kernel machines.

The category of algorithms that employ the kernel trick is known as kernel machines or kernel methods.

15.6 other supervised algorithms.

k-nearest neighbors, decision tree.

16. unsupervised learning algorithms.

PCA, k-means clustering.

17. insight of stochastic gradient descent is that gradient is an expectation which may be approximately estimated using a small set of samples.

14.2 full Bayesian inference

unlike maximum likelihood approach that make predictions using a point estimate of θ, the Bayesian approach is to make predictions using a full Bayesian posterior distribution over θ.

14.3 MAP estimate.

$\theta_{MAP} = \arg\max_\theta p(\theta|X)$

$= \arg\max_\theta \log p(X|\theta) + \log p(\theta)$

where X is dataset, $p(\theta)$ is prior dist.

15. supervised learning algorithms.

15.1 most supervised learning algorithms are based on estimating $p(y|x)$.

15.2 linear regression corresponds to the family $p(y|x;\theta) = \mathcal{N}(y; \theta^T x, I)$

15.3 logistic regression (used for classification)

$p(y=1|x;\theta) = \sigma(\theta^T x) \in (0,1)$

y is binary variable.

Chapter 6 MLP

1. deep feedforward networks, also called feedforward neural networks, or multilayer perceptrons (MLP), are deep learning models.

1.2 the goal of MLP is to approximate some function f^*.

1.3 when MLP is extended to include feedback connections, it is called rnn.

1.4 cnn is a specialized kind of MLP.

1.5 Names.

MLP model has a form like $f(x)=f^{(3)}(f^{(2)}(f^{(1)}(x)))$,

the depth of model is 3.

$f^{(i)}$ is typically vector-valued function.

width of the model is

$$\max\{\text{dimension of } f^{(i)} \mid i=1,2,3\}$$

$f^{(1)}, f^{(2)}$ are hidden layers,

$f^{(3)}$ is output layer.

x is called input layer.

if $f^{(i)}=(f_1^{(i)}, f_2^{(i)})$, $f_1^{(i)}, f_2^{(i)}$ is vector-to-scalar function, called a neuron.

2. function $f=f_2(f_1(x))$,

$f_2(x)=w^Tx$, $f_1(x)=\phi(x;\theta)$.

f_1 is not linear.

2.1 in machine learning, like kernel machines, we use a generic ϕ.

'cause ϕ is usually based only on the principle of local smoothness, f often generalizes poor.

2.2 manually engineer ϕ.

2.3 learn ϕ using deep learning

2.4 dnn can be thought of having this form $f=f_2(f_1(x))$

f_1 is all hidden layers, and is learned,

f_2 is output layer.

3. design decisions of deep learning algorithm:

choosing optimizer,

cost function,

and model family.

3.1. model:

the form of output layer,

choose activation functions,

design architecture of network, including depth of network, width of network, how each layer should connect to each other.

4. cost function

if our parametric model defines a distribution $p(y|x;\theta)$, then we can use cross-entropy between training data and model distribution as the cost function.

That is to say we train dnn using maximum likelihood.

$$J(\theta)=-\mathbb{E}_{x,y\sim\hat{p}_{data}}\log p_{model}(y|x)$$

4.1 specifying a model $p(y|x;\theta)$ automatically determines a cost function $J(\theta)$.

4.2 if $f(x;\theta)$ is used to predict mean of Gaussian, then maximum likelihood estimation is equivalent to mean squared error.

4.3 one recurring theme throughout dnn design is that the gradient of the cost function must be large and predictable enough to serve as a good guide for the learning algorithm.

4.4 If we use a sufficiently powerful dnn, i.e., our model is able to represent any function from a wide class of functions, with this class being limited only by features such as continuity and boundedness rather than by having a specific parametric form, we can view cost function a being a functional rather than just a function.

A maxout unit can learn a piecewise linear, convex function with up to k pieces.

It can be seen as learning the activation function itself rather than just the relationship between units.

6.5 ReLU is based on the principle that models are easier to optimize if their behavior is close to linear.

6.6 prior to introduction of ReLU, most neural networks used logistic sigmoid activation function

$$g(z)=\sigma(z),$$

or hyperbolic tangent activation function $g(z)=\tanh(z)$

6.7 other hidden units.

RBF unit, Softplus, Hard tanh, softmax, or cos.

5.6. we can represent dnn as $f(x;\theta)$ which provides the parameter $w=f(x;\theta)$ for our model $p(y|x;\theta)=p(y;w)$.

We predict y using our model $p(y|x;\theta)$, not dnn $f(x;\theta)$.

6. Hidden units

6.2 rectified linear units use the activation function

$$h=g(z)=\max\{0,z\},$$
$$z=W^Tx+b$$

6.3 variants of ReLU

absolute value rectification,

leaky ReLU,

parametric ReLU.

6.4 maxout units

$$g(z)_i=\max_{j\in G^{(i)}} z_j$$

$G^{(i)}$ is the indices of the inputs for group i, $\{(i-1)k+1,\dots,ik\}$.

is then to provide some additional transformations from the features h to complete the task that dnn must perform.

5.2 linear units for Gaussian output distribution.

$$p(y|x)=\mathcal{N}(y;\hat{y},I)$$
$$\hat{y}=W^Th+b$$

5.3 sigmoid units for Bernoulli output distribution.

$$P(y=1|x)=\sigma(w^Th+b)$$

y is binary ~~dis~~ variable.

5.4 softmax units for multinoulli output distributions.

$$\hat{y}_i=P(y=i|x)$$
$$\hat{y}_i=\text{softmax}(z)_i=\frac{e^{z_i}}{\sum_{j=1}^{n}e^{z_j}}$$
$$z=W^Th+b$$

where y is discrete variable, with n possible values

5.5 we can model arbitary probability distribution.

4.5 We can learn one conditional statistic of y given x, instead of learning a full probability distribution $p(y|x;\theta)$.

4.6 different cost functions give different statistics.

suppose f^* lies within the class we optimize, $f^*=\arg\min_f E_{x,y\sim p_{data}} L(y,f(x))$

if we use mean squared error,

$$L(y,f(x))=\|y-f(x)\|^2,$$

then $f^*(x)=E_{y\sim p_{data}(y|x)}[y]$

if we use mean absolute error,

$$L(y,f(x))=\|y-f(x)\|_1,$$

then $f^*(x)=$ median of $p_{data}(y|x)$

4.7 cross-entropy cost function is more popular than mse or mae.

5. given dnn model $f(x)=f_2(f_1(x))$, here f_2 is output layer, $h=f_1(x)$ is all hidden layers, the role of f_2

7. most neural networks have their layers connected in a chain structure, like $f^{(3)}(f^{(2)}(f^{(1)}(x)))$, but they need not to.

7.2 universal approximation theorem.

A MLP with a linear output layer and at least one hidden layer with any "squashing" activation function can approximate any Borel measurable function from one finite-dimensional space to another with any desired non-zero amount of error, provided that the network is given enough hidden units.

The derivative of the MLP can also approximate the derivatives of the function arbitrarily well.

7.3 The number of linear regions carved out by a deep rectifier network with d input, depth L, and n units per hidden layer, is $O\left(\binom{n}{d}^{d(L-1)} n^d\right)$, i.e., exponential in the depth L.

7.4 Why we choose model that is deep?

It's because, first, we can get exponential advantage (from 7.3), and second, we have a statistical reason that choosing a deep model encodes a belief that the function we want to learn should involve composition of several simpler functions. (distributed representation of data).

8. back-propagation.

8.1 forward propagation.

The input x provides the initial info that then propagates up to hidden units at each layer and finally produces $\hat{y}$.

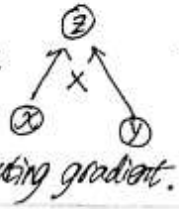
8.2 bp algorithm allows the info from the cost to then flow backwards through the network, in order to compute the gradient.

It refers only to the method of computing the gradient.

It is an algorithm that computes the chain rule, with a specific order of operations that is highly efficient.

8.3 computational graph language.

Each node in the graph indicates a variable, graph language is accompanied by a set of allowable operations. An operation is a function.



$z = xy$.

8.4 chain rule of calculus.

$x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$, $z \in \mathbb{R}$

$y = g(x)$, $z = f(y)$.

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

$$\nabla_x z = \left(\frac{\partial y}{\partial x}\right)^T \nabla_y z$$

$\frac{\partial y}{\partial x}$ is $n \times m$ Jacobian matrix.

8.5 back-propagation algorithm is designed to reduce the number of common subexpressions without regard to memory.

It performs on the order of one Jacobian product per node in the graph. Thus it avoids the exponential explosion in repeated subexpressions.

8.6 bp algorithm is only one approach to automatic differentiation that is concerned with how to compute derivatives algorithmically. It is not the only way or the optimal way of computing gradient.

much than along v_j.

∴ we shrink those small eigenvalues of H, to make $\tilde{w}$ small, at the same time, ensuring $\hat{J}(\tilde{w})$ won't increase much from $\hat{J}(w^*)$.

④ For linear regression,

$J(w)=(Xw-y)^T(Xw-y)$

$\tilde{J}(w)=J(w)+\frac{\alpha}{2}\|w\|^2$

let $\frac{\partial J}{\partial w}=0$, $\Rightarrow w=(X^TX+\alpha I)^{-1}X^Ty$

∵ X^TX is proportional to the covariance matrix $\frac{1}{m}X^TX$, and diagonal entries of $\frac{1}{m}X^TX$ correspond to variance of each input feature,

∴ L^2 regularization causes learning algorithm to "perceive" input X as having higher variance.

∵ $\hat{J}(w)=J(w^*)+\frac{1}{2}(w-w^*)^TH(w-w^*)$

∴ $\nabla_w\hat{J}(w)=H(w-w^*)$

add weight decay to $\hat{J}$, we get

$\alpha\tilde{w}+H(\tilde{w}-w^*)=0$

where $\tilde{w}$ is $\arg\min_w \hat{J}(w)+\frac{\alpha}{2}\|w\|^2$,

let $H=Q\Lambda Q^T$ be eigendecomposition,

∴ $Q^T\tilde{w}=\mathrm{diag}(\frac{\lambda_i}{\lambda_i+\alpha})Q^Tw^*$

∴ $\lambda_i \gg \alpha$, effect of regularization is small;

$\lambda_i \ll \alpha$, components of w^* along v_i will be shrunk to zero.

where v_i is eigenvector, λ_i eigenvalue of H.

③ from ②, we know H is pd,

∴ if $\lambda_i>\lambda_j$, we have

$\exists\,\delta>0,\ \forall\,\alpha\in(0,\delta)$,

$\hat{J}(w^*\pm\alpha v_i)>\hat{J}(w^*\pm\alpha v_j)$.

∴ If we move along v_i from w^*, $\hat{J}$ will increase

2.3 different choices for Ω can result in different solutions being preferred.

3. L^2 regularization

L^2 regularization, also known as weight decay, ridge regression, Tikhonov regularization.

$\tilde{J}(w)=J(w)+\frac{\alpha}{2}\|w\|_2^2$

we use $\|w\|$ to stand for $\|w\|_2$.

3.2 Four explanations.

① $\min\tilde{J}(w)=J(w)+\frac{\alpha}{2}\|w\|^2$

'cause we must balance the two terms, it keeps $J(w)$ low, at the same time, drives w closer to origin.

② let $\hat{J}$ approximate to J.

$\hat{J}(w)=J(w^*)+(w-w^*)g(w^*)$

$+\frac{1}{2}(w-w^*)^TH(w-w^*)$

where $w^*=\arg\min_w J(w)$

∵ w^* is local minimum,

∴ H is pd (positive definite), $g=0$.

Chapter 7. regularization for DL.

1. The goal of regularization is to take a model from overfitting regime to regime where the model family being trained matched the true data generating process.

1.2 In practical DL scenarios, we almost always do find that the best fitting model is a large model that has been regularized appropriately.

2. parameter norm penalty is a kind of regularization. We limit the capacity of models by adding a parameter norm penalty $\Omega(\theta)$ to objective function J.

$\tilde{J}(\theta;X,y)=J(\theta;X,y)+\alpha\Omega(\theta)$.

where $\tilde{J}$ is regularized objective function, $\alpha\in[0,\infty)$ is a hyperparameter, X is training data, y is labels.

2.2 For neural networks, we penalize only weights w, and leave biases b unregularized.

6.2 If Ω is L^2 norm, then weights are constrained to lie in an L^2 ball, same for L^1 norm.

7. We can use regularization to solve underdetermined problems. For example, pseudoinverse can be interpreted as stabilizing underdetermined problems using regularization.

8. Dataset augmentation schemes can reduce generalization, thus is a kind of regularization.
Example: apply transformation, inject noise.

Here sparse refers to the fact that some parameters have an optimal value of zero.

$\therefore$ L^1 regularization has sparsity property, L^2 does not have.

6. Norm penalty as constrained optimization.

$\therefore$ $\min_\theta J(\theta)$, subject to $\Omega(\theta) < k$,

$\Leftrightarrow \theta^* = \arg\min_\theta \max_{\alpha,\alpha\geq 0} L(\theta,\alpha)$

where $L(\theta,\alpha) = J(\theta) + \alpha(\Omega(\theta)-k)$ is generalized Lagrange function.

$\therefore$ if we fix $\alpha = \alpha^*$

let $\theta^* = \arg\min_\theta L(\theta,\alpha^*)$

$= \arg\min J(\theta) + \alpha^*\Omega(\theta)$,

we can see that this is exactly regularized training of minimizing J.

$\therefore$ norm penalty $\overset{\text{not strict}}{\Leftrightarrow}$ constrained optimization.

we suppose H is diagonal,

$\therefore \alpha\|w\|_1 + \hat{J}(w) = J(w^*) + \sum_i (\frac{1}{2}H_{ii}(w_i - w_i^*)^2 + \alpha|w_i|)$

let $\tilde{w} = \arg\min_w \hat{J}(w) + \alpha\|w\|_1$

$\therefore \tilde{w}_i = \begin{cases} w_i^* - \frac{\alpha}{H_{ii}}, & \text{if } w_i^* \geq \frac{\alpha}{H_{ii}} \\ w_i^* + \frac{\alpha}{H_{ii}}, & \text{if } w_i^* \leq -\frac{\alpha}{H_{ii}} \\ 0, & \text{otherwise.} \end{cases}$

4.3. L^1 regularization
$\Leftrightarrow$ MAP Bayesian inference with isotropic Laplace prior.

proof: $\log p(w)$

$= \sum_i \log \text{Laplace}(w_i; 0, \frac{1}{\alpha})$

$= K - \alpha\|w\|_1$

$\Leftrightarrow L^1$ regularization.

where K doesn't depend on w.

5. difference between L^1 and L^2.

from 3.2 and 4.2, we know L^1 regularization results in a solution that is more sparse.

3.3 L^2 regularization
$\Leftrightarrow$ MAP Bayesian inference with Gaussian prior.

proof:

$\theta_{MAP} = \arg\max_\theta p(\theta|x)$

$= \arg\max_\theta \log p(x|\theta) + \log p(\theta)$

if $p(w) \sim N(w; 0, \frac{1}{\lambda}I^2)$

then $\log p(w) = K + (-\frac{\lambda}{2}\|w\|^2)$ proportional to $\lambda w^T w$ in L^2.

where K doesn't depend on w, and does not affect learning process.

4. L^1 regularization.

$\tilde{J}(w) = J(w) + \alpha\|w\|_1$.

4.2. same as L^2,

we use quadratic Taylor polynomial to approximate $J(w)$ at $w^* = \arg\min J(w)$.

$\hat{J}(w) = J(w^*) + \frac{1}{2}(w-w^*)^T H(w-w^*)$

15. Bagging, short for bootstrap aggregating, is a technique for reducing generalization error by combining several models. It's an example of a general strategy in ml, called model averaging. Techniques applying this strategy are known as ensemble methods.

15.1 The reason that model averaging works is that different models will usually not make all the same errors on the test set.

15.2. consider a set of k regression models. Suppose each model makes an error ϵ_i on every example, with errors drawn from a zero-mean multivariate normal distribution with variances $E[\epsilon_i^2] = v$, and covariances $E[\epsilon_i \epsilon_j] = c$,

13. parameter sharing is a way to express our prior knowledge about suitable values of model parameters.

13.2 Parameter sharing used by CNN is an example of how to effectively incorporate domain knowledge into network architecture.

14. Sparse Representation

We can place a penalty on the activations of units, encouraging their activations to be sparse. This indirectly impose a complicated penalty on parameters.

14.2 norm penalty regularization of representation:

$\tilde{J}(\theta) = J(\theta) + \alpha \Omega(h)$

where $\alpha \in [0, \infty)$, h is representation.

shares parameters with a discriminative model of $P(y|x)$, one can then trade-off the supervised criterion $-\log P(y|x)$ with the unsupervised $-\log P(x)$ or generative one $-\log P(x,y)$. The generative criterion then expresses a particular form of prior belief about solution to supervised learning problem.

11. multi-task learning is a way to improve generalization by pooling the examples arising out of several tasks.

12. Early stopping is probably the most commonly used form of regularization in deep learning. It can be viewed as a hyperparameter selection algorithm

9. Noise Robustness

9.2 Noise injection to input can be much more powerful than parameter norm penalty.

9.3 Noise applied to weights can be interpreted as a stochastic implementation of Bayesian inference over the weights, it can also be interpreted as equivalent to a form of regularization, encouraging stability of the function to be learned.

9.4 We can also inject noise at output target.

10. we can use semi-supervised learning to provide regularization.

10.2 If we construct models in which a generative model of either $P(x)$ or $P(x,y)$

16.6 dropout is more effective than other standard computationally inexpensive regularizers, such as weight decay, sparse activity regularization

It's very computationally cheap and does not significantly limit type of model or training procedure.

17. we can reduce error rate on original i.i.d. test set via adversarial training.

18. Many ml algorithms aim to overcome the curse of dimensionality using manifold hypothesis.

Manifold hypothesis can be used as a kind of regularization.

19. central themes of machine learning

① regularization.

② optimization.

16.3 let a mask random vector μ specifies which units to include, $J(\theta, \mu)$ defines the cost of the model.

Then dropout training is:

$\min_{\theta} \min_{\mu} E J(\theta, \mu)$

16.4. parameter sharing makes it possible to represent an exponential number of models with a tractable amount of memory, and make most models do not have to be explicitly trained.

16.5. we can use weight scaling inference rule to approximate $p_{ensemble}$ by evaluating $p(y|x)$ in one model.

15. boosting constructs an ensemble with higher capacity than individual models. In this technique, ensemble is not a regularization.

So,

16. Dropout

Dropout provides a computationally inexpensive but powerful method of regularizing a broad family of models. It provides an inexpensive approximation to training and evaluating a bagged ensemble of exponentially many neural networks.

16.2 Specifically, dropout trains the ensemble consisting of all sub-networks that can be formed by removing non-output units from an underlying base network.

then error ε made by average prediction of all ensemble models is $\varepsilon = \frac{1}{k}\sum_{i=1}^{k} \varepsilon_i$, with

$E[\varepsilon] = 0$,

$Var[\varepsilon] = E[\varepsilon^2] = \frac{1}{k}v + \frac{k-1}{k}c$

∴ If errors are perfectly correlated and $c = v$, $Var[\varepsilon] = v$.

If errors are perfectly uncorrelated, and $c = 0$, $Var[\varepsilon] = \frac{v}{k}$.

∴ On average, ensemble will perform at least as well as any of its members, and if members make independent errors, it will perform significantly better than its members.

15.3 bagging uses k different datasets to get k different members.

15.4 model averaging is an extremely powerful and reliable method for reducing generalization error.

Chapter 8 Optimization for training deep models.

1. learning differs from pure optimization
   ① ml acts indirectly.
   ② cost function can be written as average.
   ③ training algorithms do not halt at a local minimum.
2. erm : empirical risk minimization.
   $\min_\theta E_{x,y\sim\hat{p}_{data}} L(f(x;\theta),y) = \frac{1}{m}\sum L(f(x^{(i)};\theta),y^{(i)})$
3. surrogate loss function
   3.1 learning algorithm minimizes a surrogate loss function. It rarely uses erm.
   3.2 early stopping criterion is based on true underlying loss function.
4. mle : maximum likelihood estimation.
   $\theta_{ML} = \arg\max_\theta \sum^m \log p_{model}(x^{(i)},y^{(i)};\theta)$
   $\Leftrightarrow \min J(\theta) = -E_{x,y\sim\hat{p}_{data}} \log p_{model}(x,y;\theta)$
5. minibatch stochastic gradient descent
   5.1 it follows the gradient of true generalization error so long as no examples are repeated.
6. challenges in optimization
   6.1 ill-conditioning : i.e., ill-conditioning of Hessian H.
   6.2 local minima
   some arising from model identifiability problem, and some being with high cost.
   6.3. plateaus, saddle points and other flat regions:
   Newton's method is designed to seek a critical point, and because saddle points are more common than local minima, so it has not succeeded in replacing gradient descent for neural network training.
   6.4. cliff structure
   6.5 vanishing and exploding gradient problem.
   6.6 inexact gradient : approximate an intractable gradient.
   6.7 difficult global structure.
   6.8 theoretical analysis of the performance of an optimization algorithm is extremely difficult.
7. random functions behavior:
   7.1 in low-dimensional space, local minima are common.
   In higher dim, local minima are rare and saddle points are more common.
   7.2 local minima are much likely to have low cost than high cost.
   Critical points with high cost are far more likely to be saddle points.
8. SGD : stochastic gradient descent.
   8.2 SGD is the most used algorithm for deep learning.
   8.3 while stopping criterion not met do:
   sample minibatch $\{x^{(1)},\cdots,x^{(m)}\}$,
   $\hat{g} \leftarrow \frac{1}{m}\nabla_\theta \sum_i L(f(x^{(i)};\theta),y^{(i)})$
   $\theta \leftarrow \theta - \varepsilon\hat{g}$
   end while.
   8.4 learning with SGD can sometimes be slow.
9. excess error
   $J(\theta) - \min_\theta J(\theta)$
   9.2 for SGD, it is $O(\frac{1}{\sqrt{k}})$ in convex problem.
10. Cramer-Rao bound states that generalization cannot decrease faster than $O(\frac{1}{k})$, k is iteration time.

13.4 RMSProp

① Root Mean Square Propagation

② update rule:

$g \leftarrow \frac{1}{m}\nabla_\theta \sum_i^m L(f(x^{(i)};\theta), y^{(i)})$

$r = \rho r + (1-\rho) g \odot g$

$\Delta\theta = -\frac{\varepsilon}{\sqrt{r}} \odot g$

$\theta \leftarrow \theta + \Delta\theta$

③ for i-th parameter, time t,

$r_i^{(t)} = \rho^{t-1}(1-\rho)[g_i^{(1)}]^2 + \rho^{t-2}(1-\rho)[g_i^{(2)}]^2 + \cdots + \rho^0 (1-\rho)[g_i^{(t)}]^2$

$\because 0 \le \rho < 1$,

$\therefore \rho^{t-k}(1-\rho)[g_i^{(k)}]^2 \to 0$

when $t \to \infty$.

here $k < t$, and is fixed.

$\therefore$ RMSProp can discard history from the extreme past.

④ if we observe g t times,

then $r = (\rho^{t-1} + \cdots + \rho^0)(1-\rho) g \odot g$

$= (1-\rho^t) g \odot g$

learning rate of i-th para is: $\frac{\varepsilon}{\sqrt{r_i}} = \frac{\varepsilon}{\sqrt{1-\rho^t}} \cdot \frac{1}{g_i}$

13. algorithms with adaptive learning rates.

13.1 learning rate is a hyper-parameter that has a significant impact on model performance.

13.2 ~~AdaGrad algorithm~~ axis-aligned

If we believe that the directions of sensitivity of the cost are somewhat axis-aligned, it can make sense to use a separate learning rate for each parameter.

13.3 AdaGrad algorithm

① adaptive gradient

② update rule:

$g \leftarrow \frac{1}{m}\nabla_\theta \sum_i^m L(f(x^{(i)};\theta), y^{(i)})$

$r \leftarrow r + g \odot g$

$\Delta\theta \leftarrow -\frac{\varepsilon}{\sqrt{r}} \odot g$

$\theta \leftarrow \theta + \Delta\theta$

③ if $r_i = \sum g_i^2$, $r_j = \sum g_j^2$,

where i is one parameter, t is iteration time,

if $g_i > g_j \Rightarrow$ learning rate $\frac{\varepsilon}{\sqrt{r_i}} < \frac{\varepsilon}{\sqrt{r_j}}$,

and $\theta_i = \theta_j = -\frac{\varepsilon}{\sqrt{t}}$

12.2. we almost always initialize all weights in the model to values drawn randomly from a distribution. The scale of the initial distribution has a large effect on both optimization and generalization.

12.3 Large initial weights yield a stronger symmetry breaking effect, but may result in exploding values, result in chaos, or cause activation function to saturate.

12.4 optimization perspective suggests that weights should be large enough to propagate info successfully, but some regularization concerns encourage making them smaller.

12.5 Besides random initialization, we can also initialize model parameters using ml.

12. initial parameters need to break symmetry ~~on~~ between different units.

11. momentum.

11.1 It derives from a physical analogy. momentum in physics is mass times velocity.

11.2 It is designed to accelerate learning

11.3 update rule:

$v \leftarrow \alpha v - \varepsilon \nabla_\theta \left(\frac{1}{m}\sum_i^m L(f(x^{(i)};\theta), y^{(i)})\right)$

$\theta \leftarrow \theta + v$

where v plays the role of velocity.

11.4. why it can accelerate?

If momentum algorithm always observes gradient g, say t times, then stepsize is $\frac{1-\alpha^t}{1-\alpha} \cdot \varepsilon \|g\|$

$\therefore$ terminal velocity is $\frac{\varepsilon g}{1-\alpha}$, $0 < \alpha < 1$

11.5 Nesterov momentum.

$v \leftarrow \alpha v - \varepsilon \nabla_\theta \left(\frac{1}{m}\sum_i^m L(f(x^{(i)};\theta+\alpha v), y^{(i)})\right)$

$\theta \leftarrow \theta + v$

15.4. Polyak averaging

It consists of averaging together several points in the trajectory through parameter space visited by an optimization algorithm.

15.5 pretraining is strategies that involve training simple models on simple tasks before confronting the challenge of training the desired model to perform the desired task.

15.6 It is more important to choose a model family that is easy to optimize than to use a powerful optimization algorithm.

15.7 continuation methods.

15.8 curriculum learning is a continuation method. It's based on the idea of planning a learning process to begin by learning simple concepts and progress to learning more complex concepts that depend on these simpler concepts.

14.4 BFGS

14.4.1 BFGS is to approximate H^{-1} with M_t that is iteratively refined by low rank updates.

14.4.2 BFGS spends less time refining each line search, but require store inverse Hessian approximation M from one step to the next than & ca.

15. optimization strategies and meta-algorithms.

15.2 batch normalization is a method of adaptive reparametrization. It acts to standardize only the mean and variance of each unit in order to stabilize learning, but allows the relationships between units and the nonlinear statistics of a single unit to change.

15.3 block coordinate descent refers to minimize with respect to a subset of variables simultaneously.

② $J_2(\theta) := J(\theta) + (\theta-\theta_0)^T \nabla_\theta J(\theta_0) + \frac{1}{2}(\theta-\theta_0)^T H (\theta-\theta_0)$

$\therefore J(\theta) \approx J_2(\theta)$

set $\frac{\partial J_2(\theta)}{\partial \theta} = 0$

$\Rightarrow H(\theta-\theta_0) + \nabla_\theta J(\theta_0) = 0$

$\therefore$ update rule is:

$\theta = \theta_0 - H^{-1} \nabla_\theta J(\theta_0)$

③ regularized update rule:

$\theta = \theta_0 - [H+\alpha I]^{-1} \nabla_\theta J(\theta_0)$

④ $\because \Delta\theta = -H^{-1} g$, $H \in R^{k\times k}$.

$\therefore$ computation complexity is $O(k^3)$, k is the number of parameters.

14.3 conjugate gradients.

update rule:

$\beta_t = \frac{(g_t - g_{t-1})^T g_t}{g_{t-1}^T g_{t-1}}$ (Polak-Ribière)

$\rho_t = -g_t + \beta_t \rho_{t-1}$

$\varepsilon^* = \arg\min_\varepsilon \frac{1}{m}\sum_{i=1}^m L(f(x^{(i)}; \theta_t + \varepsilon\rho_t), y^{(i)})$

$\theta_{t+1} = \theta_t + \varepsilon^* \rho_t$

3.5 Adam

① adaptive moments.

② It can be seen as a variant on the combination of RMSProp and momentum.

③ update rule:

$g \leftarrow \frac{1}{m}\nabla_\theta \sum_i^m L(f(x^{(i)};\theta), y^{(i)})$

$s \leftarrow \rho_1 s + (1-\rho_1) g$ // first moment

$r \leftarrow \rho_2 r + (1-\rho_2) g \odot g$ // second moment

$\hat{s} \leftarrow \frac{s}{1-\rho_1^t}$, $\hat{r} \leftarrow \frac{r}{1-\rho_2^t}$

$\Delta\theta = -\varepsilon \frac{\hat{s}}{\sqrt{\hat{r}}}$

$\theta \leftarrow \theta + \Delta\theta$

14. approximate second-order models

14.2 Newton's method

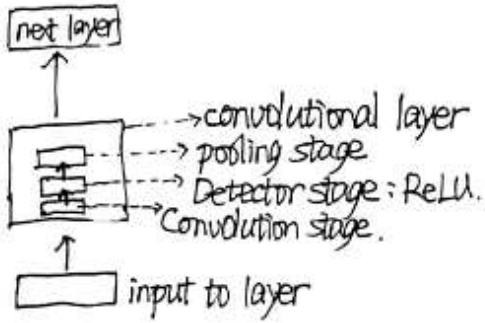
① It is an optimization scheme based on using a second-order Taylor series expasion to approximate $J(\theta)$ near some point θ_0, ignoring derivatives of higher order.

convolution ⇒ sharing parameter

5.3 convolution is equivariant to translation.

5.4. $f(x)$ is equivariant to $g(x)$ if $f(g(x)) = g(f(x))$.

6. a typical layer of CNN consists of three stages. See the figure.



7. pooling

A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs.

7.2 pooling can be viewed as adding an infinitely strong prior.

3.7. any neural network that works with matrix multiplication and does not depend on specific properties of the matrix structure should work with convolution, without any further changes to neural network.

4. convolution leverages three ideas that can help improve a machine learning system: sparse interactions, parameter sharing and equivariant representations.

5.

5.1 convolution ⇒ sparse interaction

5.2. parameter sharing refers to using the same parameter for more than one function in a model. That a network has tied weights is a synonym for parameter sharing.

3.3 the only reason to flip the kernel is to obtain the commutative property.

3.4 cross-correlation

$$S(i,j) = (I * K)(i,j)$$

$$= \sum_m \sum_n K(m,n) I(i+m, j+n)$$

we also call this operation convolution.

3.5 univariate discrete convolution corresponds to Toeplitz matrix.

In two dimensions, doubly block circulant matrix corresponds to convolution.

3.6 convolution corresponds to a sparse matrix.

prove: 1) convolution is a linear operation.

1) it can be described as a matrix multiplication.

2) the matrix is a function of kernel, and the kernel is much smaller,

3) the matrix is sparse.

Chapter 9: Convolutional Networks.

1. convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

2. a new best CNN architecture for a given benchmark is announced every few weeks to months.

3. convolution operation.

$$s(t) = \int x(a)\, w(t-a)\, da$$

this operation is called convolution, denoted as: $s(t) = (x * w)(t)$.

where x is input, w is called kernel, s is called feature map.

if w is a pdf, then s is a smoothed estimate of function x.

3.2 two-dimensional convolution

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n) K(i+m, j+n)$$

$$\because S(i,j) = \sum_m \sum_n K(m,n) I(i-m, j-n)$$

$$= (K * I)(i,j)$$

∴ convolution is commutative.

9.5 let K is a 4-D kernel tensor with $K_{i,l,m,n}$, where i indexing output channel, l input channel, m raw offset within the input, n column offset within the input.

Z is produced by convolving K across V without flipping K.

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m,k+n} K_{i,l,m,n}.$$

9.6. downsampled convolution function.

$$Z_{i,j,k} = c(K, V, s)_{i,j,k}$$

$$= \sum_{l,m,n} V_{l,j\cdot s+m,k\cdot s+n} K_{i,l,m,n}.$$

downsampled convolution is equivalent to convolution followed by downsampling, but has less computational cost.

9.7. zero padding the input allows us to control the kernel width and the size of output independently.

definition of convolution as used in mathematics.

9.3 convolution with a single kernel can only extract one kind of feature.
We want to each layer to extract many kinds of features.

9.4 we usually think of the input and output of the convolution as being 3-D tensors, with one index into different channels and two indices into spatial coordinates of each channel. Software implementations use 4-D tensors with the fourth axis indexing different examples in the batch.

8. The use of convolution is an infinitely strong prior that the function the layer should learn contains only local interactions and is equivariant to translation.
Likewise, the use of pooling is an infinitely strong prior that each unit should be invariant to small translations.

8.2 convolution and pooling can cause underfitting. A prior is only useful when the assumptions made by the prior are reasonably accurate.

8.3. some CNN architectures are designed to use pooling on some channels but not on other channels.

9. variants of basic convolution function.

9.2 convolution in CNN does not correspond precisely to the

7.3. Pooling over spatial regions produces invariance to small translation.
If pooling over the outputs of separately parametrized convolutions, the features can learn which transformations to become invariant to.

7.4 pooling with downsampling can improves computational efficiency.

7.5 pooling is essential for handling inputs of varying size.

7.6 It is also possible to dynamically pool features together rather than using pooling function, for example, by running a clustering algorithm on the locations of interesting features.

13.

13.1 one way to reduce the cost of convolutional network training is to use features that are not trained in a supervised fashion.

13.2 There are three ways for obtaining convolution kernels without supervised training. One is simply initialize them randomly. Another is to design them by hand. Finally, one can learn kernels with an unsupervised criterion.

13.3. Random filters often work surprisingly well in CNN.

', layers consisting of convolution following by pooling naturally become frequency selective and translation invariant when assigned random weights.

', we can choose architecture in an inexpensive way.

First evaluate the performance of several convolutional network architectures by training only the last layer. Then take the best of these architectures. And train the entire architecture using a more expensive approach.

of the same kind of thing, for example, recordings over time, different widths of observations over space.

12. effecient convolution algorithms.

It is possible to speed up convolution by selecting an appropriate convolution algorithm.

12.2 convolution is equivalent to converting both the input and the kernel to the frequency domain using a Fourier transform, performing point-wise multiplication of the two signals, and converting back to the time domain using an inverse Fourier transform.

For some problem size, this can be faster than the naive implementation of discrete convolution.

12.3 If kernel is separable, naive convolution is inefficient.

task or a real value for a regression task.

10.2 The model might emit a tensor S, where $S_{i,j,k}$ is probability that pixel (j,k) of the input to the network belongs to class i.

11. If computational cost and overfitting are significant issues, we can choose to use CNNs instead of MLP. If not, we can still choose CNNs because they can process inputs with varying spatial extents.

11.2 The use of convolution for processing variable sized inputs only makes sense for inputs that have variable size because they contain varying amounts of observation

9.8. locally connected layer (or unshared convolution)

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m,k+n} w_{i,j,k,l,m,n}.$$

9.9 tiled convolution offers a compromise between a convolutional layer and a locally connected layer.

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m,k+n} K_{i,l,m,n,j\%t,k\%t}$$

where t indexes different choices of kernel.

9.10 convolutional layers are hard-coded to be invariant to translation, while locally connected layers and tiled convolutional layers with max-pooling can make max-pooled units become invariant to some transformation.

10 CNNs can be used to output a tensor rather than just predicting a class label for a classification

4.3 the network with recurrent connections only from the output at one time step to hidden units at the next time step is strictly less powerful because it lacks hidden-to-hidden recurrent connections.
for example, it cannot simulate a universal Turing machine.

5. a computational graph is a way to formalize the structure of a set of computations.

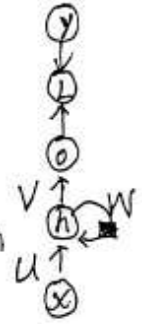
6. The advantage of eliminating hidden-to-hidden recurrence is that, for any loss function based on comparing the prediction at time t to the training target at time t, all the time steps are decoupled. Training can thus be parallelized, with the gradient for each step t computed in isolation.

3.4 RNNs with recurrent connections between hidden units, that read an entire sequence and then produce a single output.

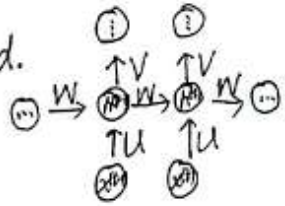
4. models.

4.2 $a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$

① recurrent graph:



② unfolded computational graph.



this RNN is universal in the sense that any function computable by a Turing machine can be computed by such a RNN of a finite size.

first, regardless of the sequence length, the learned model always has the same input size.

second, it is possible to use the same transition function of f with the same parameters at every time step.

2. parameter sharing of CNNs is shallow, while parameter sharing of RNNs is deep.

3. some design patterns for RNNs.

3.2 RNNs that produce an output at each time step and have recurrent connections between hidden units.

3.3 RNNs that produce an output at each time step and have recurrent connections only from the output at one time step to the hidden units at the next time step.

Chapter 10: Sequence Modeling

1. much as almost any function can be considered a MLP, essentially any function involving recurrence can be considered a RNN.

1.2 RNNs provide a way to extend deep learning to sequencial data.

1.3 one RNN that defines the values of their hidden units.

$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta)$.

When the RNN is trained to perform a task that requires predicting the future from the past, the network typically learns to use $h^{(t)}$ as a kind of lossy summary of sequence $(x^{(t)}, \cdots, x^{(1)})$.

1.4 ~~Re~~ $h^{(t)} = g^{(t)}(x^{(t)}, \cdots, x^{(0)})$

unfolded recurrence: $h^{(t)} = f(h^{(t-1)}, x^{(t-1)}; \theta)$

factorizing $g^{(t)}$ into repeated application of f introduces two major advantages:

$h^{(t)}$ nodes in the graphical model decouples the past and the future, and the model can be efficiently parametrized by using the same conditional probabilities distribution at each time step, and that, when the variables are all observed, the probability of the joint assignment of all variables can be evaluated efficiently.

10.4 the price RNNs pay for their reduced number of parameters is that optimizing the parameters may be difficult.

10.5. parameter sharing in RNNs
⇔ the conditional probability distribution over the variables at time $t+1$ given the variables at time t is stationary.

We parametrize the joint distribution of these observations using the chain rule for conditional probabilities:

$$P(Y)=P(y^{(1)},\cdots,y^{(\tau)})=\prod_{t=1}^{\tau} P(y^{(t)}\mid y^{(t-1)},\cdots,y^{(1)})$$

RNNs are useful when we believe that the distribution over $y^{(t)}$ may depend on a value of $y^{(i)}$ from the distant past in a way that is not captured by the effect of $y^{(i)}$ on $y^{(t-1)}$.

One way to interpret RNN as graphical model is to view the RNN as defining a graphical model whose structure is the complete graph, able to represent direct dependencies between any pair of y values.

10.3 including the hidden units in the graphical model reveals that the RNN provides a very efficient parametrization of the joint distribution over the observations.

9. As with a MLP, we usually wish to interpret the output of the RNN as a probability distribution, and we usually use the cross-entropy associated with that distribution to define the loss. Mean squared error is the cross-entropy loss associated with an output distribution that is unit Gaussian, for example, just as with a MLP.

10. RNNs as directed graphical models.

10.2 consider the case where RNN models only a sequence of scalar random variables $Y=\{y^{(1)},\cdots,y^{(\tau)}\}$, with no additional inputs x.
The input at time step t is simply the output at time step $t-1$.
The RNN then defines a directed graphical model over the y variables.

7. models that have recurrent connections from their outputs leading back into the model may be trained with teacher forcing.

7.2 teacher forcing is a procedure that emerges from maximum likelihood criterion, in which during training the model receives the ground truth output $y^{(t)}$ as input at time $t+1$.

8. computing the gradient through a RNN simply applies the generalized back-propagation algorithm to the unrolled computational graph of RNN.

8.2 BPTT.
back-propagation algorithm applied to the unrolled graph of RNN with $O(\tau)$ cost is called back-propagation through time, where τ is time length.

many stages tend to either vanish (most of the time) or explode (rarely). This problem is particular to RNNs. Various approaches have been proposed to reduce the difficulty of learning long-term dependencies.

18. Both ESNs and liquid state machines are termed reservior computing to denote the fact that the hidden units form of reservior of temporal features which may capture different aspects of the history of inputs.

18.2 ESNs: Echo State Networks.

18.3 The recurrent weights and input weights are some of the most difficult parameters to learn in a RNN.

Reservior computing avoids this difficulty by setting the recurrent and associated transformations

① from input to hidden state.

② from previous hidden state to the next hidden state.

③ from hidden state to output.

When RNN is unfolded, each of these three blocks corresponds to a shallow transformation. It is advantageous to introduce depth in each of these operations.

16. recursive neural networks represent yet another generalization of RNNs, with a different kind of computational graph, which is structured as a deep tree, rather than the chain-like structure of RNNs.

17. the basic problem for RNNs is that gradients propagated over represent a distribution $P(y|x)$ by using the same $P(y|w)$ as before, but making w a function of x.

13. bidirectional RNNs combine an RNN that moves forward through time beginning from the start of the sequence with another RNN that moves backward through time beginning from the end of the sequence.

14. the encoder-decoder or sequence-to-sequence architecture is a RNN architecture for mapping a variable-length sequence to another variable-length sequence.

15. the computation in most RNNs can be decomposed into three blocks of parameters

11. various ways to determine the length of sequence

One option is adding an extra symbol to the vocabulary.

Another approach is to introduce an extra Bernoulli output to the model that represents the decision to either continue generation or halt generation at each time step.

Another way to determine sequence length τ is to add extra output to the model that predicts the integer τ itself.

12. modeling sequences conditioned on context with RNNs.

12.2 Any model representing a variable $P(y;\theta)$ can be reinterpreted as a model representing a conditional distribution $P(y|w)$ with $w=\theta$. We can extend such a model to

controls the forgetting factor and the decision to update the state unit.

21. optimization for long-term dependencies

21.2 gradient clipping helps to deal with exploding gradients, but it does not help with vanishing gradients. To address vanishing gradients and better capture long-term dependencies, the idea is to create paths in the computational graph of an unfolded recurrent architecture along with the product of gradients associated with arcs is near 1. One approach to achieve this is with LSTMs and other self-loops and gating mechanisms. Another idea is to regularize the parameters so as to encourage "info flow".

22. neural networks excels at storing implicit knowledge. They struggle to memorize facts. Memory networks are introduced to resolve this difficulty. They include a set of memory cells that can be thought of as an extension of the memory cells in LSTMs and GRUs.

idea of creating paths through time that have derivatives that neither vanish nor explode. Leaky units did this with connection weights that were either manually chosen constants or were parameters. Gated RNNs generalize this to connection weights that may change at each time step.

20.3 LSTM: long short-term memory.

LSTM recurrent networks have "LSTM cells" that have an internal recurrence (a self-loop), in addition to the outer recurrence of the RNN.

It has three kinds of gating units, forget gate unit f, external input gate units g, and output gate q.

20.4 GRU: gated recurrent unit.

The main difference with LSTM is that a single gating unit simultaneously

way of ensuring that the unit can access values from the past.

19.3 hidden units with linear self-connections are called leaky ~~self~~ units.

19.4 another approach to handle long-term dependencies is the idea of organizing the state of the RNN at multiple timescale, with info flowing more easily through long distances at the slower time scales. It involves actively removing length-one connections and replacing them with longer connections.

20. The most effective sequence models used in practical applications are called gated RNNs, which include the LSTM and GRU.

20.2. like leaky units, gated RNNs are based on the

weights such that the recurrent hidden units do a good job of computing the history of past inputs and learning only the output weights.

19. one way to deal with long-term dependencies is to design a model that operates at multiple time scales, so that some parts of the model operate at fine-grained time scales and can handle small details, while other parts operate at coarse time scales and transfer information from the distant past to the present more efficiently.

19.2 adding skip connections through time

Skip connections through d time steps are a way of ensuring that a unit can always learn to be influenced by a value from d time steps earlier.

The use of linear self-connection with a weight near one is a different

that is already known to perform best on the previously studied task, even copy a trained model from that task.

(vi) add unsupervised learning later.

4. determine whether to gather more data.

(i) determine whether performance on the training set is acceptable.

If performance on the training set is poor, then try increasing the size of the model and try improving the learning algorithm, for example, by tuning the learning rate hyperparameter.

If large models and tuned optimization algorithm do not work well, then the problem might be the quality of the training data. This suggests starting over, collecting cleaner data or collecting a richer set of features.

(ii) if the performance on the training set is acceptable, then measure the performance on a test set.

examples for which the machine learning system is able to produce a response.

It's possible to trade coverage for accuracy.

(iv) click-through rates, user satisfaction.

3. default baseline models.

3.1 after choosing performance metrics and goals, the next step is to establish a reasonable end-to-end system asap.

3.2 (i) choose the general category of model based on the structure of your data.

(ii) choose optimization algorithm such as SGD with momentum with a decaying learning rate, or Adam.

(iii) batch normalization

(iv) early stopping, dropout

(v) copy model and algorithm

2.3 determine the target value of the performance metric and choose which metric to use.

2.4 metrics.

(i) accuracy ⇔ error rate

(ii) precision p.:

It is the fraction of detections reported by the model that were correct.

recall r.:

It is the fraction of true events that were detected.

PR curve: a curve with precision on y-axis and recall on x-axis.

we can trade precision for recall.

F-score: $F = \frac{2pr}{p+r}$

(iii) coverage:

It is the fraction of

Chapter 11 Practical Methodology

1. pratical design process

(i) determine your goals: what error metric to use, and your target value for this error metric.

(ii) establish a working end-to-end pipeline as soon as possible.

(iii) instrument the system well to determine bottlenecks in performance.

(iv) repeatedly make incremental changes such as gathering new data, adjusting hyperparameters, or changing algorithms, based on specific findings from your instrumentation.

2. performance metric

2.2 Bayes error defines the minimum error rate that you can hope to achieve, even if you can have infinite training data and can recover the true probability distribution.

1.2 Usually the best performance on the test set comes from a large model that is regularized well.

7.3 Your end goal is good performance on the test set. The brute force way to practically guarantee success is to continually increase model capacity and training set size until the task is solved. In principle, this approach could fail due to optimization difficulties.

8 debugging strategies.

8.2 Some debugging tests:

(i) visualize the model in action.

(ii) visualize the worse mistakes.

(iii) reasoning about software using training and test error.

low training error and high test error
⇒ model is overfitting or test error is measured incorrectly.

high training error and high test error
⇒ underfitting or software defect.

(iv) fit a tiny dataset.

high error on training set with few examples
⇒ software defect.

(v) compare backpropagated derivatives to numerical derivatives.

(vi) monitor histograms of activations and gradients.

(vii) test guarantees algorithms provide.

6.2 hyperparameter opti algs.

wrap a learning algorithm and choose its hyperparameters, thus ~~hidd~~ hiding the hyperparameters of the learning algorithm from the user.

6.3 grid search

When there are three or fewer hyperparameters, the common practice is to perform grid search.

6.4 random search

random search is exponentially more efficient than a grid search.

6.5 model-based hyperparameter opti

Most model-based algorithms for hyperparameter search use a Bayesian regression model to estimate both the expected value of validation set error for each hyperparameter and the uncertainty around this expectation.

7.

7.1 test error is the sum of the training error and the gap between training and test error.

6. selecting hyperparameters.

6.1 manual hyperparameter tuning

the goal of (manual) hyperparameter search is usually to find the lowest generalization error subject to some runtime and memory budget.

The generalization error typically follows a U-shaped curve when plotted as a function of one of the hyperparameters. If you have time to tune only one hyperpara, tune the learning rate. The effective capacity of the model is highest when the learning rate is correct for the optimization problem.

The learning rate has a U-shaped curve for training error.

Tuning the parameters other than the learning rate requires monitering both training and test error to diagnose whether your model is overfitting or underfitting, then adjusting its capacity.

If the performance on the test set is also acceptable, then there is nothing left to be done.

If the test set performance is much worse than the training set performance, then gathering more data is one of the most effective solutions.

If it is costly to gather more data, then reduce the size of the model or improve regularization.

If the gap between train and test performance is still unacceptable, then gathering more data is advisable.

(iii) If gathering much data is not feasible, the only other way to improve generalization error is to improve the learning algorithm itself.

5. It is recommended to experiment with training set sizes on a logarithmic scale.

a smaller model that requires less memory and runtime to store and evaluate.

2.6 one strategy for accelerating data processing systems in general is to build systems that have dynamic structure in the graph describing the computation needed to process an input.

One major obstacle to using dynamically structured systems is the decreased degree of parallelism that results from the system following different code branches for different inputs.

2.7 specialized hardware implementations.

can be run by a separate machine.

(ii) model parallelism: multiple machines work together on a single datapoint, with each machine running a different part of the model.

(iii) data parallelism is feasible for inference. Model parallelism is feasible for both inference and training.

(iv) data parallelism for training is somewhat harder, and can be solved using asynchronous stochastic gradient descent.

2.5 a key strategy for reducing the cost of inference is model compression. The basic idea of model compression is to replace the original expensive model with

2. implementations.

2.2 Fast CPU implementations.

Careful implementation for specific CPU families can yield large improvements in runtime.

2.3 GPU implementations

GPUs are specialized hardware components that were originally developed for graphics applications. General purpose GPUs could execute arbitrary code, not just rendering subroutines. Due to the difficulty of writing high performance GPU code, researchers should structure their workflow to avoid needing to write new GPU code in order to test new models or algorithms.

2.4 large-scale distributed implementations.

(i) data parallelism: each input example we want to process

Chapter 12 Applications

1. large-scale deep learning

① deep learning is based on the philosophy of connectionism: while an individual biological neuron or an individual feature in a machine learning model is not intelligent, a large population of these neurons or features acting together can exhibit intelligent behavior.

② One of the key factors responsible for the improvement in neural network's accuracy and the improvement of the complexity of tasks they can solve between the 1980s and today is the dramatic increase in the size of the networks we use.

③ Because the size of neural networks is of paramount importance, deep learning requires high performance hardware and software infrastructure.

(v). Global contrast normalization (GCN) can be understood as mapping examples to a ~~sphere~~ spherical shell, reducing each example to a direction rather than a direction and a distance. This is a useful property because neural networks are often better at responding to directions in space rather than exact locations.

(vi) local contrast normalization ensures that the contrast is normalized across each small window, rather than over the image as a whole. It is a differentiable operation and can also be used as a nonlinearity applied to the hidden layers, as well as a preprocessing operation applied to the input.

A related idea applicable at test time is to show the model many different versions of the same input and have the different instantiations of the model vote to determine the output. This idea can be interpreted as an ensemble approach, and helps to reduce generalization error.

(iv) contrast simply refers to the magnitude of the difference between the bright and the dark pixels in an image. The mount of contrast can be safely removed for many tasks Contrast usually refers to the standard deviation of the pixels in an image.

is the only kind of preprocessing that is strictly necessary.

(ii) Rescaling (resizing) images is not always strictly necessary. Some convolutional models accept variably-sized inputs and dynamically adjust the size of their pooling regions to keep the output size constant. Other convolutional models have variable-sized output that automatically scales in size with the input.

(iii) dataset augmentation may be seen as a way of preprocessing the training set only. It is an excellent way to reduce the generalization error of most computer vision models.

different forms of specialized hardware have been developed over the last decades.

Using fixed point rather than floating point representations and using less bits per number reduces the hardware surface area, power requirements and computing time needed for performing multiplications.

3. computer vision.

3.2 computer vision is a very broad field encompassing a wide variety of ways of processing images, and an amazing diversity of applications

3.3 preprocessing

(i) formatting images to have the same scale, i.e. standardizing images so that their ~~images~~ pixels all lie in the same range,

between words that are in the same category.

5.4 Neural language models (NLMs) are a class of language models designed to overcome the curse of dimensionality problem for modeling natural languge sequences by using a distributed representation of words. Distributed representation can counter the curse of dimensionality.

5.5 high-dimensional outputs

(i) using a softmax over a large number of output words has high cost. The first neural language models dealt with the high cost by using a short list. The obvious disadvantage of the short list approach is that the potential generalization advantage of the neural language models is limited to the most frequent words, where, arguably, it is the least useful.

(iii) Models based on n-grams define the conditional probability of the nth token given the preceding n-1 tokens. The model uses products of these conditional distributions to define the probability distribution over longer sequences:

$$P(x_1, \cdots, x_T) = P(x_1, \cdots, x_{n-1}) \prod_{t=n}^{T} P(x_t | x_{t-n+1}, \cdots, x_{t-1})$$

(iv) classical n-gram models are particularly vulnerable to the curse of dimensionality.

(v) To improve the statistical efficiency of n-gram models, class-based language models introduce the notion of word categories and then share statistical strength between words that are in the same category.

the corresponding sequence of words intended by the speaker.

5. NLP: natural language processing

5.2 NLP is the use of human languages by a computer. Many NLP applications are based on language models.

5.3. n-grams

(i) A language model defines a probability distribution over sequences of tokens in a natural language. Depending on how the model is designed, a token may be a word, a character, or even a byte.

(ii) An n-gram is a sequence of n tokens.

4. speech recognition

4.2 ASR: automatic speech recognition.

$$f^*_{ASR}(X) = \arg\max_y P^*(y | X = X)$$

4.3 For GMM-HMM systems, GMMs modeled the association between acoustic features and phonemes, while HMMs modeled the sequence of phonemes.

4.4. GMM-HMM (1980s – 2012)

↓

deep LSTM RNN (2013 – now)

4.5 some deep learning systems learn feature from raw acoustic input.

4.6 The task of speech recognition is to map an acoustic signal containing a spoken natural language utterance into

6. other applications

6.2 recommender systems.

Both online advertising and item recommendations rely on predicting the association between a user and an item.

6.3 (i) Many recommendation problems are most accurately described theoretically as contextual bandits.

(ii) reinforcement learning can involve a sequence of many actions and many rewards. The bandits scenario is a special case of reinforcement learning, in which the learner takes only a single action and receives a single reward. The contextual bandits refer to the case where an action is taken in the context of some input variable that can inform the decision. The mapping from context to action is called a policy.

6.4. Knowledge of relations combined with a reasoning process and understanding of natural language could allow us to build a general question answering system.

meaning in another language.

(ii) some components of machine translation systems:

(a) component that proposes many candidate translations.

(b). a language model which evaluates the proposed translations.

(iii) The encoder-decoder architecture maps back and forth between a surface representation and a semantic representation, where a surface representation is a sentence or an image, the semantic representation of the input is a summary, called context.

(iv) Attention mechanism focuses on specific parts of the input sequence at each time step. It is essentially a weighted average.

to the number of negative samples rather than proportional to the size of the output vector.

5.6 combining neural language models with n-grams.

A major advantage of n-gram models over neural networks is that n-gram models achieve high model capacity while requiring very little computation to process an example. One easy way to add capacity is thus to combine both approaches in an ensemble consisting of a neural language model and an n-gram language model.

5.7. Neural machine translation

(i) machine translation is the task of reading a sentence in one natural language and emitting a sentence with the equivalent

(ii) The hierarchical softmax is introduced to the context of neural language model. It brings computational benifits both at training time and at test time, if at test time we want to compute the probability of specific words.

A disadvantage is that in practice the hierarchical softmax tends to give worse test results than sampling-based methods.

(iii) importance sampling (Chapter 17) is useful for speeding up models with large softmax outputs. More generally, it is useful for accelerating training with large sparse output layers.

The computational complexity of gradient estimation for the output layer is reduced to be proportional

3.4 linear factor models including PCA and factor analysis can be interpreted as learning a manifold.

4. ICA

4.2 Independent component analysis (ICA) is an approach to modeling linear factors that seeks to separate an observed signal into many underlying signals that are scaled and added together to form the observed data. These signals are intended to be fully independent.

4.3 All variants of ICA require that $p(h)$ be non-Gaussian.

4.4 A generative model either represents $p(x)$ or can draw samples from it.

2.2 $\therefore x \sim N(x; b, WW^T + \psi)$

$\therefore$ the role of the latent variables is to capture the dependencies between the different observed variables x_i.

3. Probilistic PCA

3.1 $h \sim N(h; 0, I)$.

$x = Wh + b + \sigma z$

where $z \sim N(z; 0, I)$

σ is a scalar.

3.2 $\therefore x \sim N(x; b, WW^T + \sigma^2 I)$

3.3 Probabilistic PCA model takes advantage of the observation that most variations in the data can be captured by the latent variables h, up to some small residual reconstruction error σ^2.

It becomes PCA as $\sigma \to 0$.

④ linear factor models are some of the simplest generative models and some of the simplest models that learn a representation of data. Much as linear classifiers and linear regression models may be extended to MLPs, these linear factor models may be extended to autoencoder networks and deep probabilistic models that perform the same tasks but with a much more powerful and flexible model family.

2. Factor analysis.

2.1 $h \sim N(h; 0, I)$

$x = Wh + b + \text{noise}$,

$\text{noise} \sim N(\text{noise}; 0, \psi)$

where $\psi = \text{diag}(\sigma^2)$, with $\sigma^2 = [\sigma_1^2, \cdots, \sigma_n^2]^T$

Chapter 13, Linear Factor Models

1. ① linear factor models are the simplest probability models with latent variables.

② A linear factor model is defined by the use of a stochastic, linear decoder function that generates x by adding noise to a linear transformation of h. Probabilistic PCA, factor analysis, and many other linear factor models make different choises about the form of the noise and of the prior $p(h)$.

③ How does a linear factor model describe the data generating process?

First, we sample the factors h from a distribution $h \sim p(h)$, where $p(h)$ is a factorial distribution, with $p(h) = \prod_i p(h_i)$.

Next we sample the real-valued observable variables given the factors:

$x = Wh + b + \text{noise}$, where the noise is typically Gaussian and diagonal.

6.3 The encoder that we use with sparse coding is not a parametric encoder. Instead, the encoder is an optimization algorithm, that solves an optimization problem in which we seek the single most likely code value: $h^* = f(x) = \arg\max_h p(h|x)$.

6.4 advantages

(i) It can in principle minimize the combination of reconstruction error and log-prior better than any specific parametric encoder.

(ii) there is no generalization error to the encoder.

6.5 disadvantages.

(i) the non-parametric encoder requires greater time to compute h given x.

(ii) it is not straight-forward to back-propagate through to non-parametric encoder.

6.6 sparse coding, like other linear factor models, often produces poor samples. This happens even when the model is able to reconstruct the data well and provide useful features for a classifier.

and solving the ~~optimation~~ optimization problem

$$\min_\theta E_t (f(x^{(t+1)})_i - f(x^{(t)})_i)^2$$

s.t. $E_t f(x^{(t)})_i = 0$, $E_t f(x^{(t)})_i^2 = 1$

$E_t[f(x^{(t)})_i f(x^{(t)})_j] = 0, \forall i<j$

5.7 SFA seems to be a reasonably biologically plausible model.

5.8 A major advantage of SFA is that it is possibly to theoretically predict which feature SFA will learn, even in the deep, nonlinear setting.

6. Sparse coding

6.1 $x = Wh + b + \text{noise}$.

$\text{noise} \sim N(\text{noise}; 0, \frac{1}{\beta} I)$

$p(h)$ is chosen to be one with sharp peaks around near 0.

6.2 $p(x|h) = N(x; Wh+b, \frac{1}{\beta} I)$ where β is precision.

to learn invariant features.

5.3 The idea of slowness principle is that the important characteristics of scenes change very ~~ie~~ slowly compared to the individual measurements that make up a description of a scene. We can apply this principle to any ~~differentiable~~ model trained with gradient descent.

5.4 SFA is motivated by the slowness principle.

5.5 SFA is not quite a generative model perse, in the sense that it defines a linear map between input space and feature space but does not define a prior over feature space and thus does not impose a $p(x)$ on input space.

5.6 The SFA algorithm consists of defining $f(x;\theta)$ to be a linear transformation,

4.5 Many variants of ICA are not generative models. Because ICA is more often used as an analysis tool for separating signals, rather than for generating data or estimating its density.

4.6 Just as PCA can be generalized to the nonlinear autoencoders, ICA can be generalized to a nonlinear generative model, in which we use a nonlinear function f to generate the observed data.

4.7 other generalizations of ICA.

(i) NICE: nonlinear independent components estimation.

(ii) independent subspace analysis

(iii) topographic ICA.

5. Slow feature analysis (SFA)

5.2 SFA is a linear factor model that uses info from time signals

Chapter 14 Autoencoders

1. An autoencoder is a neural network that is trained to attempt to copy its input to its output. Internally, it has a hidden layer h that describes a code used to represent the input. The autoencoder has two components: an encoder function $h=f(x)$ and a decoder function $r=g(h)$.

1.2 Modern autoencoders have generalized the idea of an encoder and a decoder beyond deterministic functions to stochastic mappings $p_{encoder}(h|x)$ and $p_{decoder}(x|h)$

1.3 Autoencoders may be thought of as being a special case of feedforward networks, and may be trained with all of the same techniques.

1.4 recirculation is a learning algorithm based on comparing the activations of the network on the original input to the activations on the reconstructed input. It is regarded as more biologically plausible than back-propagation, but is rarely used for machine learning applications.

2. Undercomplete autoencoders

2.2 An autoencoder whose code dimension is less than the input dimension is called undercomplete. Learning an undercomplete representation forces the autoencoder to capture the most salient features of the training data.

Unlike general feedforward networks, autoencoders may also be trained using recirculation.

2.3 The learning process is described simply as minimizing a loss function $L(x, g(f(x)))$

2.4 When the decoder is linear and L is mse, an undercomplete autoencoder learns to span the same subspace as PCA. Autoencoders with nonlinear encoder functions f and nonlinear decoder g can thus learn a more powerful nonlinear generalization of PCA.

2.5 An autoencoder can fail to learn anything useful about the dataset if the capacity of the autoencoder is allowed to become too great.

3. Regularized Autoencoders.

3.2 A regularized autoencoder can be nonlinear and overcomplete but still learn something useful about the data distribution even if the model capacity is great enough to learn a trivial identity function.

3.3 nearly any generative model with latent variables and equipped with an inference procedure (for computing latent representations given input) may be viewed as a particular form of autoencoder.

3.4 examples:

- sparse autoencoder,
- DAE,
- CAE.

4. Sparse Autoencoders

A sparse autoencoder is simply an autoencoder whose training criterion involves a sparsity penalty $\Omega(h)$ on the code layer h, in addition to the reconstruction error:

$$L(x, g(f(x))) + \Omega(h)$$

4.3 We can think of the penalty $\Omega(h)$ as a regularizer term added to a feedforward network whose primary task

(ii) sample $\tilde{x}$ from corruption $C(\tilde{X}|X=x)$

(iii) use $(\tilde{x}, x)$ as a training example for estimating the reconstruction distribution

$P_{reconstruct}(x|\tilde{x}) = P_{decoder}(x|h)$, with $h=f(\tilde{x})$

5.4 If the encoder is deterministic, then DAE is a feedforward network. We can therefore view the DAE as performing stochastic gradient descent on the expectation:

$-E_{x\sim\hat{P}_{data}(x)} E_{\tilde{x}\sim C(\tilde{x}|x)} \log P_{decoder}(x|h=f(\tilde{x}))$

where $\hat{P}_{data}(x)$ is the training distribution.

5.5 (i) score matching is alternative to maximum likelihood. It provides a consistent estimator of probability distributions based on encouraging

(iii) This view provides a different motivation for training an autoencoder: it is just a way of approximately training a generative model. It also provides a different reason for why the features learned by autoencoder are useful: they describe the latent variables that explain the input.

5. denoising autoencoders

5.2 A denoising autoencoder (DAE) minimizes $L(x, g(f(\tilde{x})))$, where $\tilde{x}$ is a copy of x that has been corrupted by some form of noise.

5.3 learn a reconstruction distribution.

(i) sample x from training data.

We can think of the autoencoder as approximating this sum with a point estimate for just one highly likely value for h. With this chosen h, we are maximizing

$\log P_{model}(h,x) = \log P_{model}(h) + \log P_{model}(x|h)$

The $\log P_{model}(h)$ term can be sparsity-inducing.

From this point of view of sparsity as resulting from the effect of $P_{model}(h)$ on approximate maximum likelihood learning, the sparsity penalty is not a regularization term at all. It is just a consequence of the model's distribution over its latent variables.

is to copy the input to the output.

∴ the regularizer depends on the data,

∴ there is not a straightforward Bayesian interpretation to this regularizer. But we can still think of these regularization terms as implicitly expressing a preference over functions.

4.4 (i) Rather than thinking of the sparsity penalty as a regularizer for the copying task, we can think of the entire sparse autoencoder framework as approximating maximum likelihood training of a generative model that has latent variables.

(ii) For a model with visible variables x and latent variables h, with an explicit $P_{model}(x,h) = P_{model}(h)P_{model}(x|h)$, the log-likelihood can be decomposed as $\log P_{model}(x) = \log \sum_h P_{model}(h,x)$.

9. Stochastic encoders and decoders.

9.2 To make a more radical departure from feedforward networks, we can generalize the notion of an encoding function $f(x)$ to an encoding distribution $p_{encoder}(h|x)$.

9.3 Any latent variable model $p_{model}(h,x)$ defines a stochastic encoder $p_{encoder}(h|x) = p_{model}(h|x)$ and a stochastic decoder $p_{decoder}(x|h) = p_{model}(x|h)$.

9.4 So long as the encoder is deterministic, the autoencoder is a feedforward network.

10. Learning manifolds with autoencoders.

$\min L(x, g(f(x))) + \Omega(h,x)$
with $\Omega(h,x) = \lambda \sum_i \|\nabla_x h_i\|^2$

An autoencoder regularized in this way is called contractive autoencoder (CAE).

7. Autoencoders are feedforward networks. Moreover, the encoder is itself a feedforward network as is the decoder.

8. Advantages of depth in feedforward networks

8.2 Depth can exponentially reduce the computational cost of representing some functions. Depth can also exponentially decrease the amount of training data needed to learn some functions.

RBM with Gaussian visible units. When the RBM is trained using denoising score matching, its learning algorithm is equivalent to denoising training in the corresponding autoencoder.

5.6 Like sparse autoencoders, sparse coding, contractive autoencoders and other regularized autoencoders, the motivation for DAEs was to allow the learning of a very high-capacity encoder while preventing the encoder and decoder from learning a useless identity function.

6. Regularizing by penalizing derivatives.

the model to have the same score as the data distribution at every training point x. In this context, the ~~score~~ score is a particular gradient field: $\nabla_x \log p(x)$

(ii) Learning the gradient field of $\log p_{data}$ is one way to learn the structure of p_{data} itself.

(iii) A very important property of DAEs is that their training criterion (with conditionally Gaussian $p(x|h)$) makes the autoencoder learn a vector field $(g(f(x)) - x)$ that estimates the score of the data distribution.

(iv) Denoising training of a specific kind of autoencoder (sigmoidal hidden units, linear reconstruction units) using Gaussian noise and mse as the reconstruction cost is equivalent to training

13.

13.2 lower-dimensional representations can improve performance on many tasks. Models of smaller spaces consume less memory and runtime.

13.3 The task of information retrieval is to find entries in a database that resemble a query entry. This task derives the benefit that search can become extremely efficient in certain kinds of low dimensional spaces.

13.4 semantic hashing is an approach to information retrieval via dimensionality reduction and binarization.

11.7. The construction penalty can obtain useless results if we do not pose some sort of scale on the decoder.

12. PSD.

12.2 Predictive sparse decomposition (PSD) is a model that is a hybrid of sparse coding and parametric autoencoders. Here $f(x)$ and $g(h)$ are both parametric. A parametric encoder is trained to predict the output of iterative inference.

12.3 training proceeds by minimizing
$\|x-g(h)\|^2+\lambda\|h\|_1+\gamma\|h-f(x)\|^2$

12.4 PSD is an example of learned approximate inference. (Chapter 19).

11.3. In the limit of small Gaussian input noise, the denoising reconstruction error is equivalent to a contractive penalty on the reconstruction function that maps x to $r=g(f(x))$

11.4 The contractive autoencoder introduces an explicit regularizer on the code $h=f(x)$:
$\Omega(h)=\lambda\left\|\frac{\partial f(x)}{\partial x}\right\|_F^2$
$\|\cdot\|_F$ is Frobenius norm.

11.5 CAE is contractive only locally: all perturbations of a training point x are mapped near to $f(x)$.

11.6. The goal of CAE is to learn the manifold structure of the data.

10.2 Like many other machine learning algorithms, autoencoders exploit the idea that data concentrates around a low-dimensional manifold or a small set of such manifolds. Autoencoders aim to learn the structure of the manifold.

10.3. The autoencoder can afford to represent only the variations that are needed to reconstruct training examples. The encoder learns a mapping that is only sensitive to changes along the manifold directions, but is insensitive to changes orthogonal to the manifold.

11. Contractive Autoencoders.

2.4 unsupervised pretraining combines two different ideas. First, it makes use of the idea that the choice of initial parameters for a dnn can have a significant regularizing effect on the model. Second, it makes use of the more general idea that learning about the input distribution can help to learn about the mapping from inputs to outputs.

2.5 The first idea is the least well understood. Our ability to characterize exactly what aspects of the pretrained parameters are retained during the supervised training stage is limted. This is one reason that modern approaches typically use simultaneous unsupervised learning and supervised learning rather than two sequential stages.

2.6. The other idea is better understood. The basic idea is that some features that are

2.1 unsupervised pretraining, or more precisely, greedy unsupervised pretraining, played a key historical role in the revival of dnns, enabling researchers for the first time to train a deep supervised network without requiring architectural specializations like convolution or recurrence.

2.2 The deep learning renaissance of 2006 began with the discovery that this greedy learning procedure could be used to find a good initialization for a joint learning procedure over all the layers, and that this approach could be used to successfully train even fully connected architectures.

2.3 unsupervised pretraining is sometimes helpful but often harmful.

1.4 Most representation learning problems face a tradeoff between persevring as much information about the input as possible and attaining nice properties.

1.5 We can learn good representations for the unlabeled data, and then use these representations to solve the supervised learning task.

1.6. Humans and animals are able to learn from very few labeled examples. We do not yet know how this is possible. Many factors could explain improved human performance — for example, the brain may use very large ensembles of classifies or Bayesian inference techniques. One popular hypothesis is that the brain is able to leverage unsupervised or semi-supervised learning.

2. greedy layer-wise unsupervised pretraining

Chapter 15 Representation learning

1. Many information processing tasks can be very easy or very difficult depending on how the info is represented. A good representation is one that makes a subsequent learning task easier.

1.2 We can think of feedforward networks trained by supervised learning as performing a kind of representation learning. Training with a supervised criterion naturally leads to the representation at every hidden layer taking on properties that make the classification task easier.

1.3 supervised training of feedforward networks does not involve explicitly imposing any condition on the learned intermediate features. Other kinds of representation learning algorithms are often explicitly designed to shape the representation in some particular way.

3.5 concept drift can be viewed as a form of transfer learning, due to gradual changes in the data distribution over time.

3.6 Both concept drift and transfer learning can be viewed as particular forms of multi-task learning.

3.7 The core idea of representation learning is that the same representation may be useful in both settings.

3.8 two extreme forms of transfer learning are one-shot learning and zero-shot learning, sometimes also called zero-data learning.

4. Semi-supervised disentangling of causal factors.

4.2. an ideal representation is one that disentangles the causes from one another.

3. Transfer learning and domain adaptation.

3.2 transfer learning and domain adaptation refer to the situation where what has been learned in one setting (i.e., distribution P_1) is exploited to improve generalization in another setting (say distribution P_2).

3.3 In transfer learning, the learner must perform two or more different tasks, but we assume that many of the factors that explain the variations in P_1 are relevant to the variations that need to be captured for learning P_2.

3.4 In the related case of domain adaptation, the task remains the same between each setting, but the input distribution is slightly different.

2.9. disadvantages of unsupervised pretraining.

(i) there is not a way of flexibly adapting the strength of the regularization arising from the unsupervised stage.

(ii) Another disadvantage of having two separate training phases is that each phase has its own hyperparameters.

2.10. deep learning techniques based on supervised learning, regularized with dropout or batch normalization, outperform unsupervised pretraining on medium-sized datasets. On extremely small datasets, Bayesian methods outperform methods based on unsupervised pretraining.

2.11 The idea of pretraining has been generalized to supervised pretraining, as a very common approach for transfer learning.

useful for the unsupervised task may also be useful for the supervised task. This is not yet understood at a mathematical, theoretical level, so it is not always possible to predict which task will benefit from unsupervised learning in this way. Many aspects of this approach are highly dependent on the specific models used. This is another reason that simultaneous supervised and unsupervised learning can be more preferable.

2.7 unsupervised pretraining can act to improve optimization rather than being merely a regularizer.

2.8 how can unsupervised pretraining act as a regularizer? One hypothesis is that pretraining encourages the learning algorithm to discover features that relate to the underlying causes that generate the observed data.

one is that generalization arises due to shared attributes between different concepts.

5.6 Distributed representations introduce a rich similarity space

5.7. When and why can there be a statistical advantage from using a distributed representation as part of a learning algorithm?

(i) Distributed representations can have a statistical advantage when an apparently complicated structure can be compactly represented using a small number of parameters.

(ii) with $O(nd)$ parameters (for n linear-threshold features in R^d) we can distinctly represent $O(n^d)$ regions in input space. If instead we made no assumption at all about the data, and used a representation with one unique symbol for each region, and separate parameters for each symbol to recognize its corresponding portion of R^d, to describe k^n different concepts

5.3 A symbolic representation is a specific example of the broader class of non-distributed representations, which are representations that may contain many entries but without significant meaningful separate control over each entry.

5.4 examples of learning algorithms based on non-distributed representations:

clustering methods,

k-nearest neighbors algorithms,

decision trees,

Gaussian mixtures and mixtures of experts,

kernel machines with a Gaussian kernel,

language or translation models based on n-grams.

5.5 An important related concept that distinguishes a distributed representation from a symbolic

We expect that future research will develop mechanisms for representing different factors depending on the task.

4.6 A benefit of learning the underlying causal factors is that if the true generative process has x as an effect and y as a cause, then modeling $p(x|y)$ is robust to changes in $p(y)$. If the cause-effect relationship was reversed, this would not be true, since the by Bayes' rule, $p(x|y)$ would be sensitive to changes in $p(y)$.

5. Distributed representation.

5.2 distributed representations compose of many elements that can be set separately from each other. They are powerful because they can use n features with k values

4.3 A representation that cleanly separates the underlying causal factors may not necessarily be one that is easy to model. If a representation h represents many of the underlying causes of the observed x, and the outputs y are among the most salient causes, then it is easy to predict y from h.

4.4. human beings fail to perceive changes in their environment that are not immediately relevant to the task they are performing.

4.5 An important research frontier in semi-supervised learning is determining what to encode in each situation. GANs are only one step toward determining which factors should be represented.

learning provides a very strong clue: a label that specifies the value of at least one of the factors of variation directly. Representation learning makes use of other, less direct hints about the underlying factors.

7.3 generic regularization strategies:
smoothness,
linearity,
multiple explanatory factors,
causal factors,
depth (or a hierarchical organization of explanatory factors),
shared factors across tasks,
manifolds,
natural clustering,
temporal and spatial coherence,
sparsity,
simplicity of factor dependencies.

models with a single hidden layer of latent variables, including RBMs and deep belief networks, are universal approximators of probability distributions.

7.

7.2 An ideal representation is one that disentangles the underlying causal factors of variation that generated the data, especially those factors that are relevant to our applications. Most strategies for representation learning are based on introducing clues that help the learning to find these underlying factors of variations. Supervised

such as non-distributed models generalize only locally via the smoothness prior.

6. Exponential gains from depth

6.2 It has been proven in many different setting that organizing computation through the composition of many nonlinearities and a hierarchy of reused features can give an exponential boost to statistical efficiency, on top of the exponential boost given by using a distributed representations.

6.3 deterministic feedforward networks are universal approximators of functions. Many structured probabilistic

then specifying $O(nd)$ regions would require $O(nd)$ examples.

(iii) A further part of the argument for why models based on distributed representations generalize well is that their capacity remains limited despite being able to distinctly encode so many different regions.

5.8. Non-distributed algorithms break the input space into regions, with a separate set of parameters for each region. The advantage of a non-distributed approach is that, given enough parameters, it can fit the training set without solving a difficult optimization algorithm, because it is straightforward to choose a different output independently for each region. The disadvantage is that

Chapter 16 Graphical models for dl

1. A structured probabilistic model (graphical model) is a way of describing a probability distribution, using a graph to describe which random variables in the probability distribution interact with each other directly.

2. The challenge of unstructured modeling

2.1 The goal of deep learning is to scale machine learning to the kinds of challenges needed to solve AI. This means being able to understand high-dimensional data with rich structure.

2.2 classification algorithms are often able to ignore many parts of the input.

2.3 Many other tasks are often more expensive than classification. Most require a complete understanding of the entire structure of the input, with no option to ignore sections of it. These tasks include the following: density estimation, denoising, missing value imputation, sampling.

2.5 ~~modeling~~ model a distribution over a random vector x containing n discrete variables capable of taking on k values each, then the naive approach of representing $P(x)$ by storing a lookup table with one probability value per possible outcome requires k^n parameters. This is not feasible for several reasons:

(i) memory: the cost of storing the representation.

(ii) statistical efficiency.

(iii) runtime: the cost of inference.

(iv) runtime: the cost of sampling.

2.6 The table-based approach explicitly models every possible kind of interaction between every possible subset of variables. Structured probabilistic models provide a formal framework for modeling only direct interactions between random variables.

3. Using graphs to describe model structure

3.1 structured probabilistic models use graphs to represent interactions between random variables. Each node represents a random variable, Each edge represents a direct interaction.

3.2 graphical models can be largely divided into two categories: models based on directed acyclic graphs, and models based on undirected graphs.

4. directed models

4.2 directed graphical models, also known as belief network or Bayesian network.

4.3 $p(x) = \prod_i p(x_i \mid Pa_G(x_i))$ where $Pa_G(x_i)$ gives the parents of x_i in G.

4.4 The graph encodes only simplifying assumptions about which variables are conditionally independent from each other. It is also possible to make other kinds of simplifying assumptions.

4.5 The directed graphical model syntax does not place any constraint on how we define our conditional distributions. It only defines which variables they are allowed to take in as arguments.

5. undirected models

5.2 undirected models also known as Markov random fields (MRFs) or Markov networks.

5.3 unnormalized probability distribution.

$$\tilde{p}(x) = \prod_{C \in G} \phi(C)$$

where C is clique in the graph, factor $\phi(C)$ is clique potential.

8.3 two variables are dependent if there is an active path between them, and independent if no such path exists.

8.4. In the case of undirected models, we refer to paths involving only unobserved variables as "active" and paths including an observed variable as "inactive".

8.5 In directed models, all ~~the~~ of the kinds of active paths of length two that can exist between a and b.

(i) ⓐ→Ⓢ→ⓑ
ⓐ←Ⓢ←ⓑ

(ii) ⓐ←Ⓢ→ⓑ

(iii) ⓐ→◉←ⓑ

(iv) ⓐ→Ⓢ←ⓑ
↓
◉

here the shaded variables are indicating that ~~they~~ are observed.

7.5 An energy-based model is just a special kind of Markov network: the exponentiation makes each term in the energy function correspond to a factor for a different clique.

7.6 free energy:

$$F(x) = -\log \sum_h \exp(-E(x,h))$$

8. separation and d-separation

8.2 we say that a set of variables A is separated (d-separated) from another set of variables B given a third set of variables S if the graph structure implies that A is independent from B given S, where separation is for undirected models, d-separation for directed models, and "d" stands for "dependence".

7. Energy-based models (EBMs)

7.2 Many interesting theoretical results about undirected models depend on the assumption that $\tilde{p}(x) > 0, \forall x$. A convenient way to enforce this condition is to use an EBM where

$$\tilde{p}(x) = \exp(-E(x)), \quad (1)$$

and $E(x)$ is known as the energy function.

7.3 Any distribution of the form given by equation (1) is an example of a Boltzmann distribution. For this reason, many energy-based models are called Boltzmann machines.

7.4. The term Boltzmann machine is today most often used to designate models with latent variables, while Boltzmann machines without latent variables are more often called Markov random fields or log-linear models.

6. The partition function

6.2 $p(x) = \frac{1}{Z}\tilde{p}(x)$

where $Z = \int \tilde{p}(x)\,dx$ is known as partition function.

6.3 A distribution defined by normalizing ~~by~~ a product of clique potentials is also called a Gibbs distribution.

6.4 one key difference between directed modeling and undirected modeling is that directed models are defined directly in terms of probability distributions from the start, while undirected models are defined more loosely by ϕ functions that are then converted into probability distributions.

6.5 It is possible to leverage the effect of a carefully chosen domain of a variable in order to obtain complicated ~~both~~ behavior from a relatively simple set of ϕ functions.

generally very fast (assuming sampling from each conditional is easy) and convenient. The drawbacks are that it only applies to directed models and does not support every conditional or sampling operation.

11.4 sampling techniques for undirected models are an advanced topic, covered in more detail in Chapter 17.

12. Advantages of structured modeling

12.2 (i) dramatically reduce the cost of representing probability distributions as well as learning and inference.
(ii) sampling is accelerated in the case of directed models.
(iii) allow us to explicitly separate representation of knowledge from learning of knowledge or inference given existing knowledge. This makes our models easier to develop and debug.

10. Factor graphs

10.2 factor graphs are another way of drawing undirected models that resolve an ambiguity in the graphical representation of standard undirected model syntax.

10.3 factor graphs explicitly represent the scope of each ϕ function.

11. sampling from graphical models

11.2 one advantage of directed graphical models is that a simple and efficient procedure called ancestral sampling can produce a sample from the joint distribution represented by the model.

11.3 ancestral sampling is

we can always represent any distribution by using a complete graph.

9.4. When we represent a probability distribution with a graph, we want to choose a graph that implies as many independencies as possible without implying any independencies that do not actually exist.

9.5 Directed models are able to use one specific kind of substructure that undirected models cannot represent perfectly. This substructure is called an immorality.

9.6 Likewise, undirected models can include substructures that no directed model can represent perfectly.

8.6 context-specific independencies are independencies that are present dependent on the value of some variables in the network.

8.7 In general, a graph will never imply that an independence exists when it does not. However, a graph may fail to encode an independence.

9. converting between undirected and directed graphs.

9.2 no probabilistic model is inherently directed or undirected. Instead, some models are most easily described using a directed graph, or most easily described using an undirected graph.

9.3 Every probability distribution can be represented by either a directed model or by an undirected model. In the worst case,

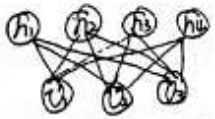
15.6 Most deep models are designed to make Gibbs sampling or variational inference algorithms efficient.

15.7 the deep learning approach to graphical modeling is characterized by a marked tolerance of the unknown.

16. The ~~restrictive~~ restricted boltzmann machine (RBM)

16.2 The canonical RBM is an energy-based model with binary visible and hidden units. Its energy function is

$$E(v,h) = -b^T v - c^T h - v^T W h$$



16.3 RBM demonstrates the typical deep learning approach to graphical models: representation learning accomplished via layers of latent variables, combined with efficient interactions between layers parametrized by matrices.

context of graphical models, we can define the depth of a model in terms of the graphical model rather than the computational graph.

15.3 deep learning essentially always makes use of the idea of distributed representations. Deep learning models typically have more latent variables than observed variables.

15.4 In deep learning, the latent variables do not take on specific semantics ahead of time.

15.5 deep graphical models typically have large groups of units that are connected to other groups of units, so that the interactions between two groups may be described by a single matrix.

in order to implement a learning rule. It is an example of inference problems in which we must predict the value of some variables given other variables, or predict the probability distribution over some variables given the value of other variables.

14.3 The graphs used for deep learning are usually not restrictive enough to allow efficient inference. This motivates the use of approximate inference. (See Chapter 19).

15. The deep learning approach to graphical models

15.2 deep learning does not always involve especially deep graphical models. In the

13. learning about dependencies

13.2 often the different elements of v are highly dependent on each other. In the context of deep learning, hidden variables h are introduced to model these dependencies.

13.3 latent variables have advantages beyond their role in efficiently capturing $p(v)$. The new variables h also provide an alternative representation for v.

13.4 often, given some model of v and h, experimental observations show that $E[h|v]$ or $\operatorname{argmax}_h p(h,v)$ is a good feature mapping for v.

14. Inference and approximate inference

14.2 Because $\log p(v) = E_{h\sim p(h|v)}(\log p(h,v) - \log p(h|v))$, we often want to compute $p(h|v)$

3.5 biased importance sampling

biased importance sampling estimator:

$$\hat{s}_{BIS} = \frac{\sum_{i=1}^{n} \frac{p(x^{(i)})}{q(x^{(i)})} f(x^{(i)})}{\sum_{i=1}^{n} \frac{p(x^{(i)})}{q(x^{(i)})}} = \frac{\sum_{i=1}^{n} \frac{\tilde{p}(x^{(i)})}{\tilde{q}(x^{(i)})} f(x^{(i)})}{\sum_{i=1}^{n} \frac{\tilde{p}(x^{(i)})}{\tilde{q}(x^{(i)})}}$$

where $\tilde{p}$ and $\tilde{q}$ are the unnormalized forms of p and q and the $x^{(i)}$ are the samples from q. This estimator is biased because $E[\hat{s}_{BIS}] \neq s$, but is asymptotically unbiased.

4. MCMC methods

4.2 we have use Markov chain to approximately sample from $p_{model}(x)$. The family of algs that use Markov chains to perform Monte Carlo estimates is called Markov chain Monte Carlo methods (MCMC).

4.3 The most standard, generic guarantees for MCMC techniques are only applicable when the model does not assign zero probability to any state.

to confront the intractable partition function of undirected models. (See Chapter 18).

3. importance sampling

3.2 importance sampling and its variants have been found very useful in many machine learning algs, including dl algs.

3.3 $\because s = \int p(x) f(x) dx = E_p[f(x)]$

$$p(x) f(x) = q(x) \frac{p(x) f(x)}{q(x)}$$

$\therefore$ any Monte Carlo estimator

$\hat{s}_p = \frac{1}{n} \sum_{x^{(i)} \sim p} f(x^{(i)})$

can be transformed into an importance sampling estimator

$\hat{s}_q = \frac{1}{n} \sum_{x^{(i)} \sim q} \frac{p(x^{(i)}) f(x^{(i)})}{q(x^{(i)})}$

3.4 ① $E_q[\hat{s}_q] = E_p[\hat{s}_p] = s$

② $Var[\hat{s}_q] = Var[\frac{p(x) f(x)}{q(x)}]/n$

$\therefore$ The minimum variance occurs where q is $q^*(x) = \frac{p(x)|f(x)|}{Z}$, where Z is the normalization constant.

Better importance sampling distributions put more weight where the integrand is larger.

view a sum or integral as if it was an expectation under some distribution and to approximate the expectation by a corresponding average.

2.5 $s = \sum_x p(x) f(x) = E_p[f(x)]$

or $s = \int p(x) f(x) dx = E_p[f(x)]$,

we approximate s by $\hat{s}_n$.

$\hat{s}_n = \frac{1}{n} \sum_{i=1}^{n} f(x^{(i)})$

where $x^{(i)}$ are samples from p.

2.6 properties

① $E[\hat{s}_n] = s$

② if $x^{(i)}$ are i.i.d., $Var[f(x^{(i)})]$ is bounded, then

(i) $Var[\hat{s}_n] = \frac{1}{n^2} \sum^{n} Var[f(x^{(i)})] = \frac{Var[f(x)]}{n}$

(ii) $\lim_{n \to \infty} \hat{s}_n = s$, almost surely.

(iii) the distribution of $\hat{s}_n$ converges to a normal distribution with mean s and variance $\frac{Var[f(x)]}{n}$.

2.7 Monte Carlo techniques are the primary tool used

Chapter 17 Monte Carlo methods

1. randomized algorithms fall into two rough categories: Las Vegas algorithms and Monte Carlo algorithms.

1.3 Las Vegas algorithms always return precisely the correct answer (or report that they failed).

1.4. Monte Carlo algorithms return answers with a random amount of error. The amount of error can typically be reduced by expending more resources.

1.5 Many problems in machine learning are so difficult that we can never expect to obtain precise answers to them. This excludes precise deterministic algs and Las Vegas algs. Instead, we must use deterministic approximate algs or Monte Carlo approximations. Both approaches are ubiquitous in machine learning.

2. sampling and Monte Carlo methods

2.3 why sampling?

It provides a flexible way to approximate many sums and integrals at reduced cost. In many cases, it is actually our goal, in the sense that we want to train a model that can sample from the training distribution.

2.4. The idea of Monte Carlo sampling is to

5. Gibbs sampling

5.2 Now we have described how to draw samples from a distribution $q(x)$ by repeatedly updating

$x \leftarrow x' \sim T(x'|x)$.

In order to ensure $q(x)$ is a useful distribution, there are two basic approaches. The first one is to derive T from a given learned p_{model} (example 5.3). The second one is to directly parametrize T and learn it, so that its stationary distribution implicitly defines the p_{model} of interest.

5.3 A conceptually simple and effective approach to building a Markov chain that samples from an energy-based model defining a distribution $p_{model}(x)$ is to use Gibbs sampling, in which sampling from $T(x'|x)$ is accomplished by selecting one

4.9 running the Markov chain until it reaches its equilibrium distribution is called "burning in" the Markov chain.

The length of time a Markov chain must run before reaching its equilibrium distribution is called mixing time.

4.10 Markov chains are expensive to use because of the time required to burn in the equilibrium distribution and the time required to transition from one sample to another reasonably decorrelated sample after reaching equilibrium.

Another difficulty is that we do not know in advance the mixing time. It is also very difficult to test whether a Markov chain has reached equilibrium.

We describe q using a vector v, with $q(x=i) = v_i$, we define A so that $A_{i,j} = T(x'=i|x=j)$.

$\therefore q^{(t+1)}(x') = \sum_x q^{(t)}(x) T(x'|x)$

$\therefore v^{(t)} = A^t v^{(0)}$

Under some mild conditions, the process converges to a stationary distribution, also called the equilibrium distribution.

At convergence, $v' = Av = v$.

4.7. If we have chosen T correctly, then the stationary distribution q will be equal to the distribution p we wish to sample from.

4.8 Most properties of Markov chains with countable states can be generalized to continuous variables. In general, a Markov chain with transition operator T will converge, under mild conditions, to a fixed point described by the equation

$q'(x') = E_{x \sim q} T(x'|x)$

4.4 The core idea of a Markov chain is to have a state x that begins as an arbitrary value. Over time, we randomly update x repeatedly. Eventually, x becomes (very nearly) a fair sample from $p(x)$.

4.5 Formally, a Markov chain is defined by a random state x and a transition distribution $T(x'|x)$ specifying the probability that a random state will go to state x' if it starts in state x. Running the Markov chains means repeatedly updating the state x to a value x' sampled from $T(x'|x)$.

4.6. Suppose x has countably many states, then we can represent the states as just a positive integer x. We run infinitely many Markov chains in parallel. All of the states of the different Markov chains are drawn from some distribution $q^{(t)}(x)$, where t indicates the number of time steps that have elapsed. Our goal is for $q^{(t)}(x)$ to converge to $p(x)$.

(i) $P_\beta(x) \propto \exp(-\beta E(x))$ where β is described as being the reciprocal of the temperature, reflecting the origin of energy-based models in statistical physics. Tempering is a general strategy of mixing between modes of P_1 rapidly by drawing samples with $\beta < 1$.

(ii) Markov chains based on tempered transitions temporarily sample from high-temperature distributions in order to mix to different modes, then resume sampling from the unit temperature distribution.

Another approach is to use parallel tempering.

6.7 depth may help mixing

One way to resolve the problem of 6.4 is to make h be a deep representation that encodes x into h in such a way that a Markov chain in the space of h can mix more easily.

it well, which implies that h and x should have very high mutual info. We often learn generative models that very precisely encode x into h but are not able to mix very well. This situation arises frequently with Boltzmann machines.

6.5 In many classification problems, we expect a distribution that has a manifold structure with a separate manifold for each class: the distribution is concentrated around many modes and these modes are separated by vast regions of high energy. This type of distribution would make MCMC methods converge very slowly because of poor mixing between modes.

6.6. Tempering to mix between modes.

they have a tendency to mix poorly.

6.3 In high dimensional cases, MCMC samples become very correlated. We refer to such behavior as slow mixing or even failure to mix.

6.4. In the context of models with latent variables, which define a joint distribution $P_{model}(x, h)$, we often draw samples of x by alternating between sampling from $P_{model}(x|h)$ and sampling from $P_{model}(h|x)$. From the point of view of mixing rapidly, we would like $P_{model}(h|x)$ to have very high entropy. However, from the point of views of learning a useful representation of h, we would like h to have encode enough info about x to reconstruct

variable x_i and sampling it from P_{model} conditioned on its neighbors in the undirected graph G defining the structure of the energy-based model. It is also possible to sample several variables at the same time so long as they are conditionally independent given all of their neighbors. Gibbs sampling approaches that update many variables simultaneously in this way are called block Gibbs sampling.

5.4 In the context of the deep learning approach to undirected modeling, it is rare to use any approach other than Gibbs sampling.

6. The challenge of mixing between separated modes.

6.2 The primary difficulty involved with MCMC methods is that

⑤ $g \leftarrow g - \frac{1}{m}\sum^{m} \nabla_\theta \log \tilde{p}(\tilde{x}^{(i)};\theta)$

⑥ $\theta \leftarrow \theta + \varepsilon g$

end while.

3.4. The contrastive divergence (CD) algorithm:

(i) Set ε.

(ii) Set k. Perhaps 1-20 to train an RBM on a small image patch.

(iii) while not converged do

①, ②, ③, ④, ⑤, ⑥.

// ①, ②, ④, ⑤, ⑥ are same as 3.3

//③: for i=1 to m do

$\tilde{x}^{(i)} \leftarrow x^{(i)}$

end for

end while

3.5 The CD algorithm initializes the Markov chain at each step with samples from the data distribution.

partition function using gradient ascent:

(i) Set ε, the step size, to a small positive number.

(ii) Set k, the number of Gibbs ~~steps~~ steps, high enough to allow burn in. Perhaps 100 to train an RBM on a small image patch.

(iii) while not converged do

① sample a minibatch of m examples $\{x^{(1)}, \cdots, x^{(m)}\}$ from the training set.

② $g \leftarrow \frac{1}{m}\sum^{m} \nabla_\theta \log \tilde{p}(x^{(i)};\theta)$

③ Initialize a set of m samples $\{\tilde{x}^{(1)}, \cdots, \tilde{x}^{(m)}\}$ to random values.

④ for i=1 to k do

for j=1 to m do

$\tilde{x}^{(j)} \leftarrow$ gibbs_update($\tilde{x}^{(j)}$)

end for

end for

regularity conditions on $\tilde{p}$ and $\nabla_\theta \tilde{p}(x)$,

$$\nabla_\theta \log Z = \nabla_\theta E_{x \sim p(x)} \log \tilde{p}(x)$$

This identity is the basis for a variety of Monte Carlo methods for approximately maximizing the likelihood of models with intractable partition functions.

2.4 The Monte Carlo approach to learning undirected models provides an intuitive framework in which we can interpret the positive phase as pushing down on the energy of training examples and the negative phase as pushing up the energy of samples drawn from the model.

3. CD and SML

3.3 a naive MCMC algorithm for maximizing the log-likelihood with an intractable

Chapter 18 confronting the partition function

1. We must normalize $\tilde{p}$ by dividing by a partition function $Z(\theta)$ in order to obtain a valid probability distribution:

$$p(x;\theta) = \frac{1}{Z(\theta)} \tilde{p}(x;\theta)$$

The partition function is an integral or sum over the unnormalized probability of all states:

$$\int \tilde{p}(x)\,dx \text{ or } \sum_x \tilde{p}(x)$$

2. The log-likelihood gradient

2.2 The gradient of the log-likelihood with respect to the parameters has a term corresponding to the gradient of the partition function:

$$\nabla_\theta \log p(x;\theta) = \nabla_\theta \log \tilde{p}(x;\theta) - \nabla_\theta \log Z(\theta)$$

This is a well-known decomposition into the positive phase and negative phase of learning.

2.3 For models that guarantee $p(x) > 0$ for all x, and under

4. pseudolikelihood

4.2 $\frac{p(x)}{p(y)} = \frac{\frac{1}{Z}\tilde{p}(x)}{\frac{1}{Z}\tilde{p}(y)} = \frac{\tilde{p}(x)}{\tilde{p}(y)}$

The pseudolikelihood is based on the observation that conditional probabilities take this ratio-based form, and thus can be computed without knowledge of the partition function.

4.3 $p(a|b) = \frac{p(a,b)}{p(b)} = \frac{p(a,b)}{\sum_{a,c} p(a,b,c)}$

$= \frac{\tilde{p}(a,b)}{\sum_{a,c} \tilde{p}(a,b,c)}$

$\therefore p(x_i|x_{-i}) = \frac{\tilde{p}(x_i, x_{-i})}{\sum_{x_i} \tilde{p}(x_i, x_{-i})}$

4.4. log

① likelihood objective function:

$\log p(x) = \log p(x_1) + \log p(x_2|x_1) + \cdots + \log p(x_n|x_{1:n-1})$

② pseudolikelihood objective function:

$\sum_{i}^{n} \log p(x_i|x_{-i})$

hidden and visible units. CD is only able to provide an initialization for the visible units. SML is able to train deep models efficiently.

3.13 SML is vulnerable to becoming inaccurate if the stochastic gradient algorithm can move the model faster than the Markov chain can mix between steps.

3.14 It is necessary to draw the samples starting from a fresh Markov chain initialized from a random starting point after the model is done training.

3.15 CD proves to have lower variance than the estimator based on exact sampling. SML has higher variance.

3.11. The SML/PCD algorithm using gradient ascent as the optimization procedure:

(i) Set ϵ

(ii) Set k, high enough to allow a Markov chain sampling from $p(x; \theta+\epsilon g)$ to burn in, starting from samples from $p(x;\theta)$. Perhaps 1 for RBM on a small image patch. Initialize a set of m samples $\{\tilde{x}^{(1)}, \cdots, \tilde{x}^{(m)}\}$ to random values.

(iii) while not converged do

①②④⑤⑥

// same as 3.3

end while

3.12 SML is considerably more resistant to forming models with spurious modes than CD is. SML provides an initialization point for both the

3.6 The regions that have high probability under the model but low probability under the data generating distribution are called spurious modes.

3.7 It was shown experimentally that the CD estimator is biased for RBMs and fully visible Boltzmann machines in that it converges to different points than the maximum likelihood estimator.

3.8 CD is useful for training shallow models like RBMs. It does not provide much help for training deeper models directly.

3.9 the CD update direction is not the gradient of any function.

3.10 stochastic maximum likelihood (SML) also called persistent contrastive divergence (PCD).

where c is explicitly introduced as an approximation of $-\log Z(\theta)$.

7.3 NCE works by reducing the unsupervised learning problem of estimating $p(x)$ to that of learning a probabilistic binary classifier in which one of the categories corresponds to the data generated by the model.

7.4 We use standard maximum likelihood learning on the supervised learning problem of fitting P_{joint} to P_{train}.

$$\theta, c = \arg\max_{\theta, c} E_{x,y \sim P_{train}} \log P_{joint}(y|x)$$

where $P_{joint}(y=1) = \frac{1}{2}$

$$P_{joint}(x|y=1) = P_{model}(x)$$

$$P_{joint}(x|y=0) = P_{noise}(x)$$

pushing down on all fantasy states that have only one variable different from a training example.

6. denoising score matching

6.2 we fit a distribution $P_{smoothed}(x) = \int P_{data}(y) q(x|y) dy$ rather than the true P_{data}, where $q(x|y)$ is a corruption process.

6.3 several autoencoder training algorithms are equivalent to score matching or denoising score matching.

7. Noise-contrastive estimation (NCE)

7.2 In NCE, the probability distribution estimated by the model is represented explicitly as ~~$\log p$~~ $\log p$

$$\log P_{model}(x) = \log \tilde{P}_{model}(x;\theta) + c,$$

5.4 score matching is not applicable to models of discrete data.

5.5. score matching can be viewed as a version of contrastive divergence using a specific kind of Markov chain. The Markov chain in this case is not Gibbs sampling, but rather a different approach that makes local moves guided by the gradient.

5.6 ratio matching applies specifically to binary data.

$$L^{(RM)}(x, \theta) = \sum_{j=1}^{n} \left(\frac{1}{1 + \frac{P_{model}(x;\theta)}{P_{model}(f(x,j);\theta)}} \right)^2$$

where $f(x,j)$ returns x with the bit at position j flipped.

5.7 As with the pseudolikelihood estimator, ratio matching can be thought of as

4.5 estimation by maximizing the pseudolikelihood is asymptotically consistent.

4.6 pseudolikelihood estimator can be thought of as having something resembling a negative phase.

5. score matching and ratio matching

5.2 score matching

$$L(x, \theta) = \frac{1}{2} \| \nabla_x \log P_{model}(x;\theta) - \nabla_x \log P_{data}(x) \|_2^2$$

$$J(\theta) = \frac{1}{2} E_{P_{data}(x)} L(x, \theta)$$

$$\vec{\theta} = \arg\min_{\theta} J(\theta)$$

5.3 minimizing the expected value of $L(x, \theta)$ is equivalent to minimizing the expected value of

$$\tilde{L}(x, \theta) = \sum_{j=1}^{n} \left(\frac{\partial^2}{\partial x_j^2} \log P_{model}(x;\theta) + \frac{1}{2} \left(\frac{\partial}{\partial x_j} \log P_{model}(x;\theta) \right)^2 \right)$$

where n is the dimensionality of x.

being too far from p_1 by introducing intermediate distributions that attempt to bridge the gap between p_0 and p_1.

8.6 AIS:

$$\frac{Z_1}{Z_0} = \frac{Z_{\eta_1}}{Z_0} \cdot \frac{Z_{\eta_2}}{Z_{\eta_1}} \cdots \frac{Z_{\eta_{n-1}}}{Z_{\eta_{n-2}}} \cdot \frac{Z_1}{Z_{\eta_{n-1}}}$$

$$= \prod_{j=0}^{n-1} \frac{Z_{\eta_{j+1}}}{Z_{\eta_j}}$$

where a sequence of distributions $p_{\eta_0}, \cdots, p_{\eta_n}$, with $0 = \eta_0 < \eta_1 < \cdots < \eta_{n-1} < \eta_n = 1$.

Provided the distributions p_{η_j} and $p_{\eta_{j+1}}$, for $0 \leq j \leq n-1$, are sufficiently close, we can reliably estimate each of the factors $\frac{Z_{\eta_{j+1}}}{Z_{\eta_j}}$ using simple importance sampling and then use these to obtain an estimate of $\frac{Z_1}{Z_0}$.

$$Z_1 = \int \tilde{p}_1(x)\,dx$$

$$= Z_0 \int p_0(x) \frac{\tilde{p}_1(x)}{\tilde{p}_0(x)}\,dx$$

$$\hat{Z}_1 = \frac{Z_0}{K} \sum_{k=1}^{K} \frac{\tilde{p}_1(x^{(k)})}{\tilde{p}_0(x^{(k)})},$$

s.t.: $x^{(k)} \sim p_0$

8.3 an estimate of the variance of $\hat{Z}_1$:

$$\widehat{Var}(\hat{Z}_1) = \frac{Z_0}{K^2} \sum_{k=1}^{K} \left(\frac{\tilde{p}_1(x^{(k)})}{\tilde{p}_0(x^{(k)})} - \hat{Z}_1 \right)^2$$

8.4 if p_0 is close to p_1, 8.2 can be an effective way of estimating the partition function. If not, $\hat{Z}_1$ will be of poor quality due to high variance.

8.5 annealed importance sampling (AIS) and bridge sampling attempt to overcome the problem of the proposal p_0 should be able to distinguish data from noise. GANs are based on the idea that a good generative model should be able to generate samples that no classifier can distinguish from data.

7.8 SML and CD estimate only the gradient of the log partition function. Score matching and pseudolikelihood avoid computing quantities related to the partition function altogether.

8. estimating the partition function

8.2 a simple way to estimate the partition function is to use a Monte Carlo method such as simple importance sampling.

$$p_{train}(y=1) = \frac{1}{2}$$

$$p_{train}(x \mid y=1) = p_{data}(x)$$

$$p_{train}(x \mid y=0) = p_{noise}(x)$$

7.5 p_{joint} is essentially a logistic regression model.

$$p_{joint}(y=1 \mid x) = \frac{p_{model}(x)}{p_{model}(x) + p_{noise}(x)}$$

$$= \sigma(\log p_{model}(x) - \log p_{noise}(x))$$

7.6. NCE is most successful when applied to problems with few random variables, but can work well even if those random variables can take on a high number of values

7.7 NCE is based on the idea that a good generative model

3.6. Even though the E-step involves exact inference, we can think of the EM algorithm as using approximate inference in some sense.

3.7 insights.

First, there is the basic structure of the learning process, in which we update the model parameters to improve the likelihood of a completed dataset, where all missing values provided by an estimate of the posterior distribution. Second, we can continue to use one value of q even after we have moved to a different value of θ.

4. MAP inference and sparse coding

4.3 inference refers to computing the probability distribution over one set of variables given another.

approach to learning with an approximate posterior.

3.3. E-step:

Set $q(h^{(i)}|v) = p(h^{(i)}|v^{(i)};\theta^{(0)})$ for all indices i of the training examples $v^{(i)}$ we want to train on (both batch and minibatch variants are valid), where $\theta^{(0)}$ is current parameter value.

M-step: Completely or partially maximize $\sum_i L(v^{(i)},\theta,q)$ with respect to θ.

3.4. EM can be viewed as a coordinate ascent algorithm to maximize L.

3.5 stochastic gradient ascent on latent variable models can be seen as a special case of the EM algorithm where the M step consists of taking a single gradient step.

2.4 we can rearrange L into a much more convenient form:

$$L(v,\theta,q) = E_{h\sim q}[\log p(h,v)] + H(q)$$

2.5 we can think of inference as the procedure for finding the q that maximizes L. Exact inference maximizes L perfectly by searching over a family of functions q that includes $p(h|v)$. We can think of learning as maximizing L with respect to θ.

3. expectation maximization (EM).

3.2 EM is not an approach to approximate inference, but rather an

Chapter 19 Approximate Inference

1. The challenge of inference usually refers to the difficult problem of computing $p(h|v)$ or taking expectations with respect to it. Intractable inference problems in deep learning usually arise from interactions between latent variables in a structured graphical ~~models~~ model.

2. Inference as optimization

2.2 Exact inference can be described as an optimization problem. Approximate inference algorithms may be derived by approximating the underlying optimization problem.

2.3 evidence lower bound (ELBO), also called variational free energy.

$$L(v,\theta,q) = \log p(v;\theta) - D_{KL}(q(h|v)\|p(h|v;\theta))$$

where q is an arbitrary probability distribution.

is the origin of the names "variational learning" and "variational inference", though these names apply even when the latent variables are discrete and calculus of variations is not needed.

5.5 mean field approach imposes the restriction that q is a factorial distribution: $q(h|v) = \prod_i q(h_i|v)$

5.6. structured variational inference imposes a graphical model structure we choose on q.

5.2 The core idea behind variational learning is that we can maximize L over a restricted family of distributions q. This family should be chosen so that it is easy to compute $E_q \log p(h,v)$.

A typical way to do this is to introduce assumptions about how q factorizes.

5.3 The beauty of the variational approach is that we do not need to specify a specific parametric form for q. We specify how it should factorize, but then the optimization problem determines the optimal probability distribution within those factorization constraints.

5.4 calculus of variations

MAP inference allow us to learn using a point estimate of $p(h|v)$ rather than inferring the entire distribution.

4.8 MAP inference is commonly used in deep learning as both a feature extractor and a learning mechanism. It is primarily used for sparse coding models.

4.9. Training sparse coding with maximum likelihood is intractable. Instead, we use MAP inference and learn parameters by maximizing the ELBO defined by the Dirac distribution around the MAP estimate of h.

5. variational inference and learning

4.4. maximum a posteriori inference (MAP inference).

$h^* = \arg\max_h p(h|v)$

4.5 MAP inference is usually not thought of as approximate inference — it does compute the exact most likely value of h^*. In some sense, we can think of MAP inference as approximate inference, because it does not provide the optimal q.

4.6 If $q(h|v) = \delta(h-\mu)$, // Dirac distribution

then $\arg\max_q L(v,\theta,q)$

$\Leftrightarrow \mu^* = \arg\max_\mu \log p(h=\mu, v)$

$\Leftrightarrow h^* = \arg\max_h p(h|v)$

∴ MAP inference can be thought of as approximate inference.

4.7 learning algorithms based on

$J(y)=\int_{x_1}^{x_2} L(x, y(x), y'(x))dx$

where x_1, x_2 are constants, $y(x)$ is twice continuously differentiable.

$y'(x)=dy/dx$

$L(x, y(x), y'(x))$ is twice continuously differentiable with respect to x, y, y'.

If the functional $J(y)$ attains a local minimum at f, then

$$\frac{\delta}{\delta f(x)}J=\frac{\partial L}{\partial f}-\frac{d}{dx}\frac{\partial L}{\partial f'}=0$$

(v) normal distribution has the maximum entropy for fixed variance σ^2.

proof:

The Lagrangian functional:

$L(p)=\lambda_1(\int p(x)dx-1)+\lambda_2(E[x]-\mu)$
$+\lambda_3(E[(x-\mu)^2]-\sigma^2)+H(p)$

5.8 calculus of variations

(i) functional derivative, also known as variational derivative.

(ii) the functional derivative of the functional $J(f)$ with respect to the value of the function f at point x is denoted $\frac{\delta}{\delta f(x)}J$.

(iii) for differentiable functions $f(x)$ and differentiable functions $g(y, x)$ with continuous derivatives, that

$$\frac{\delta}{\delta f(x)}\int g(f(x), x)\,dx$$
$$=\frac{\partial}{\partial y}g(f(x), x)$$

(iv) Euler-Lagrange equation. equations defined on RNN.

5.7. discrete latent variables

① we show how to apply variational inference to the binary sparse coding model.

② binary sparse coding:

$v\in R^n$, $h\in\{0,1\}^m$,

$p(h_i=1)=\sigma(b_i)$

$p(v|h)=\mathcal{N}(v; Wh, \beta^{-1})$

③ train this model with maximum likelihood,

$\because \frac{\partial}{\partial b_i}\log p(v)$

$=E_{h\sim p(h|v)}\frac{\partial}{\partial b_i}\log p(h)$

but $p(h|v)$ is a complicated distribution.

$\therefore$ log-likelihood is intractable.

We can resolve this difficulty by using variational inference and variational learning instead.

④ we make a mean field approximation:

$q(h|v)=\prod_i q(h_i|v)$

$q(h_i=1|v)=\hat{h}_i$,

where $\hat{h}_i\neq 0, 1$.

⑤ The use of this mean field approximation makes learning tractable.

The ELBO L is tractable, and we use L as a replacement for the intractable log-likelihood.

⑥ we use fixed point equations to estimate $\hat{h}$.

$$\frac{\partial}{\partial \hat{h}_i}L(v, \theta, \hat{h})=0$$

$\Rightarrow \hat{h}_i$

The mean field fixed point

implements an approximation $\hat{f}(v;\theta)$

6.3 one possible explanation for the role of dream sleep is that dream could provide the negative phase samples that Monte Carlo training algorithms use to approximate the negative gradient of the log partition function of undirected models. ~~Aother~~ Another possible explanation is that it is providing samples from $p(h,v)$ which can be used to train an inference network to predict h given v. Dreaming may also serve reinforcement learning rather than probabilistic modeling, by sampling synthetic experiences from the animal's transition model, on which to train the animal's policy.

affects the learning process, and this in turn affects the accuracy of the inference algorithm.

(ii) The training algorithm tends to adapt the model in a way that makes the approximating assumptions underlying the approximate inference algorithm become more true.

6. learned approximate inference

6.2 ~~Jf~~ Inference can be thought of as an optimization procedure that increases the value of $\mathcal{L}$. Once we think of the multi-step iterative optimization process as just being a function f that maps an input v to an approximate distribution

$$q^* = \arg\max_q \mathcal{L}(v, q),$$

we can approximate it with a neural network that

5.9. continuous latent variables

If we make the mean field approximation $q(h|v) = \prod_i q(h_i|v)$ and fix $q(h_j|v)$ for all $j \neq i$, then the optimal $q(h_i|v)$ may be obtained by ~~max~~ normalizing the unnormalized distribution

$$\tilde{q}(h_i|v) = \exp\left(E_{h_{-i}\sim q(h_{-i}|v)} \log \tilde{p}(v,h)\right)$$

so long as p does not assign 0 probability to any joint configuration of variables.

5.10 interactions between learning and inference

(i) using approximate inference as part of a learning algorithm

To minimize the Lagrangian with respect to p, we set the functional derivatives equal to 0:

$$\forall x, \frac{\delta}{\delta p}\mathcal{L} = \lambda_1 + \lambda_2 x + \lambda_3 (x-\mu)^2 - 1 - \log p(x) = 0$$

$$\Rightarrow p(x) = \exp(\lambda_1 + \lambda_2 x + \lambda_3 (x-\mu)^2)$$

We are free to choose any λ values, because the gradient of the Lagrangian with respect to the λ variables is zero so long as the constraints are satisfied. We may set $\lambda_1 = 1 - \log\sqrt{2\pi\sigma^2}$, $\lambda_2 = 0$, and $\lambda_3 = -\frac{1}{2\sigma^2}$ to obtain $p(x) = \mathcal{N}(x;\mu,\sigma^2)$

training of deep architectures. The introduction of DBNs in 2006 began the current deep learning renaissance.

3.2 DBNs are generative models with several layers of latent variables. The latent variables are typically binary, while the visible units may be binary or real. There are no intra-layer connections. The connections between the top two layers are undirected. The connections between all other layers are directed, with the arrows pointed toward the layer that is closest to the data.

3.3 A DBN with l hidden layers:

$$p(h^{(l)}, h^{(l-1)}) \propto \exp(b^{(l)\top} h^{(l)} + b^{(l-1)\top} h^{(l-1)} + h^{(l-1)\top} W^{(l)} h^{(l)})$$

2.5 $\because p(h|v) = \frac{p(h,v)}{p(v)}$

$$= \frac{1}{Z} \exp(c^\top h + v^\top W h)$$

$$= \frac{1}{Z} \prod_{j=1}^{n_h} \exp(c_j h_j + v^\top W_{:,j} h_j)$$

$\therefore p(h|v)$ is factorial.

2.6 $P(h_j = 1 | v) = \sigma(c_j + v^\top W_{:,j})$

$$\therefore p(h|v) = \prod_{j=1}^{n_h} \sigma((2h-1) \odot (c + W^\top v))_j$$

Similarly, $p(v|h) = \prod_{i=1}^{n_v} \sigma((2v-1) \odot (b + W h))_i$

2.7 compared to other undirected models used in deep learning, the RBM is relatively straightforward to train because we can compute $p(h|v)$ exactly in closed form.

2.8 DBMs are less closely tied to RBMs than DBNs are.

3. deep belief networks (DBNs)

3.1 DBNs were one of the first non-convolutional models to successfully admit training of deep generative models. This ability helped make Boltzmann machine learning somewhat biologically plausible.

2. RBMs

2.2 RBMs are undirected probabilistic graphical models containing a layer of observable variables and a single layer of latent variables. (See RBM in Chapter 16).

$$p(v,h) = \frac{1}{Z} \exp(-E(v,h))$$

$$E(v,h) = -b^\top v - c^\top h - v^\top W h.$$

Now we begin with the binary version of RBM.

2.3 It was proved that Z is intractable. So $p(v)$ is also intractable to evaluate.

2.4. $p(h|v)$ and $p(v|h)$ are factorial and relatively simple to compute and to sample from.

Chapter 20 deep generative models

1. Boltzmann machines

1.2 Boltzmann machines were originally introduced as a general "connectionist" approach to learning arbitrary probability distribution over binary vectors.

1.3 $$p(x) = \frac{\exp(-E(x))}{Z},$$

$$E(x) = -x^\top U x - b^\top x,$$

when all the variables are observed

$$E(v,h) = -v^\top R v - v^\top W h - h^\top S h - b^\top v - c^\top h,$$

where we decompose the units x into two subsets: the visible units v and the latent units h.

1.4 a Boltzmann machine with hidden units becomes a universal approximator of probability mass functions over discrete variable

1.5 The learning rule of Boltzmann machine is "local", which

4.7 The use of proper mean field allows the approximate inference procedure for DBM to capture the influence of top-down feedback interactions. This makes DBMs interesting from the point of view of neuroscience.

4.8 sampling from DBMs is relatively difficult.

4.9 learning in the DBM must confront both the challenge of an intractable partition function and the challenge of an intractable posterior distribution.

4.10 DBMs are typically trained using SML (variational SML). Many of the other techniques described in chapter 18 are not applicable.

on one side and even layers on the other.

4.5 The bipartite structure makes Gibbs sampling in a DBM efficient. The naive approach to Gibbs sampling is to update only one variable at a time. Gibbs sampling in a RBM can be divided into two blocks of updates, one including all even layers (including the visible layer) and the other including all odd layers.

4.6 DBM were developed after DBN. Compared to DBN, the posterior distribution $p(h|v)$ is simpler for DBM. The simplicity of this posterior distribution allows richer approximations of the posterior.

3.6 We use a greedy layer-wise procedure to train a DBN.

4. deep Boltzmann machines (DBMs)

4.2. DBM is another kind of deep, generative model. Unlike DBN, it is an entirely undirected model. Unlike RBM, DBM has several layers of latent variables. But like RBM, within each layer, each of the variables are mutually independent, conditioned on the variables in the neighboring layers.

4.3 In the case of a DBM with three hidden layers, to omit the bias parameters,

$$E(v, h^{(1)}, h^{(2)}, h^{(3)}; \theta) = -v^T W^{(1)} h^{(1)} - h^{(1)T} W^{(2)} h^{(2)} - h^{(2)T} W^{(3)} h^{(3)}$$

4.4 the DBM layer can be organized into a bipartite graph, with odd layers

$$p(h_i^{(k)} = 1 \mid h^{(k+1)}) = \sigma(b_i^{(k)} + W_{:,i}^{(k+1)T} h^{(k+1)}), \quad \forall i, \forall k \in 1, m, l-2.$$

$$p(v_i = 1 \mid h^{(1)}) = \sigma(b_i^{(1)} + W_{:,i}^{(1)T} h^{(1)}), \quad \forall i.$$

In the case of real-valued visible units, substitute

$$v \sim N(v; b^{(1)} + W^{(1)T} h^{(1)}, \beta^{-1})$$

with β diagonal for tractability.

3.4 A DBN with only one hidden layer is just an RBM.

3.5 To generate a sample from a DBN, we first run several steps of Gibbs sampling on the top two hidden layers, then use a single pass of ancestral sampling through the rest of the model to draw a sample from the visible units.

the canonical energy model for real-valued data.

5.4 Since the Gaussian RBM only models the conditional mean of the input given the hidden units, it cannot capture conditional covariance information. Alternative models have been proposed that attempt to better account for the covariance of real-valued data. These models include the mean and covariance RBM (mcRBM), the mean-product of t-distribution (mPoT) model and the spike and slab RBM (ssRBM).

5. convolutional Boltzmann machines.

back-propagation computes the exact gradient of the loss. The disadvantage is that it does not provide a way to optimize the log-likelihood, but rather a heuristic approximation of the generalized pseudolikelihood.

4.17 The MP-DBM has some connections to dropout.

5. Boltzmann machines for real-valued data

5.2 Gaussian-Bernoulli RBM:

$$E(v,h) = \frac{1}{2} v^T(\beta \odot v) - (v \odot \beta)^T W h - b^T h$$

$$\Rightarrow p(v|h) = N(v; Wh, \beta^{-1}),$$

where h is binary, v is real-valued.

5.3 Gaussian RBM has been

with MCMC estimates of the gradient.

4.15 MP-DBM works by viewing the mean field equations as defining a family of recurrent networks for approximately solving every possible inference problem. Rather than training the model to maximize the likelihood, the model is trained to make each recurrent network obtain an accurate answer to the corresponding inference problem.

4.16 back-propagation through the approximate inference graph has two main advantages. First, it trains the model as it is really used. The other is that

4.11 Greedy layer-wise pretraining of a DBM differs from greedy layer-wise pretraining of a DBN.

4.12 There are two main ways to resolve the joint training problem of DBM. The first is centered DBM. The second way to jointly train a DBM is to use a multi-prediction DBM (MP-DBM).

4.13 centered DBM reparametrizes the model in order to make the Hessian of the cost function better-conditioned at the beginning of the learning process. This yields a model that can be trained without a greedy layer-wise pretraining stage.

4.14 MP-DBM uses an alternative training criterion that allows the use of the back-propagation algorithm in order to avoid the problems

derivatives of y with respect to the parameters of its distribution, μ and σ.

9.3 We can express any probability distribution of the form $p(y;\theta)$ or $p(y|x;\theta)$ as $p(y|w)$, where w is a variable containing both parameters θ, and if applicable, the inputs x.

9.4 Given a value y sampled from distribution $p(y|w)$, where w may in turn be a function of other variables, we can rewrite $y \sim p(y|w)$ as $y = f(z;w)$, where z is a source of randomness. We may then compute the derivatives of y with respect to w applied to f, so long as f is continuous and differentiable almost everywhere. Crucially, w must not be a function of z, and z must not be a function of w. This technique, using back-propagation algorithm we can take the

as powerful supervised learners as MLPs, at least using existing methodology.

8.3 Most Boltzmann machines used in practice have only second-order interactions in their energy functions. It is possible to train higher-order Boltzmann machines.

8.4 The Boltzmann machine framework is a rich space of models permitting many more model structures than have been explored so far.

9. back-propagation through random operations

9.2 sampling operation

$$y \sim \mathcal{N}(\mu, \sigma^2)$$

$$\Leftrightarrow y = \mu + \sigma z$$

$$z \sim \mathcal{N}(z; 0, 1)$$

RBM modeling $p(x^{(t)}|x^{(t-1)}, \ldots x^{(t-m)})$ for small m can be applied to this task.

7.4 Another sequence modeling task is to model the distribution over sequences of musical notes used to compose songs. The RNN-RBM sequence model can be applied to this task.

8. other Boltzmann machines

8.2 Boltzmann machines may be extended with different training criteria. We have focused on Boltzmann machines trained to approximately maximize the generative criterion $\log p(v)$. It is also possible to train discriminative RBMs that aim to maximize $\log p(y|v)$ instead. Unfortunately, RBMs do not seem to be

8.3 Boltzmann machines that use probabilistic max pooling may accept variable-sized input images so long as the output of the model is a feature map that can scale in size proportional to the input image.

7. Boltzmann machines for structured or sequential outputs

7.2 conditional modeling with a Boltzmann machine can be used not just for structured output tasks, but also for sequence modeling.

7.3 An important sequence modeling task for the video game and film industry is modeling sequences of joint angles of skeletons used to render 3-D characters. A conditional

samples x or to distributions over samples x using a differentiable function $g(z;\theta^{(g)})$ which is typically represented by a neural network. This model class includes VAEs, GANs, and techniques that train generator networks in isolation.

10.7 variational autoencoder (VAE)

(i) VAE is a directed model that uses learned approximate inference and can be trained purely with gradient-based methods.

(ii) to generate a sample from the model, the VAE first draws a sample z from the code distribution $p_{model}(z)$. The sample is then run through a differentiable generator network $g(z)$. Finally, x is sampled from a distribution

10.3 DBNs are partially directed model.

10.4 sparse coding models can be thought of as shallow directed generative models.

10.5 sigmoid belief networks are a simple form of directed graphical model with a specific kind of conditional probability distribution. We can think of a sigmoid belief network as having a vector of binary states s, with each element of the state influenced by its ancestors:

$$p(s_i) = \sigma\left(\sum_{j<i} W_{j,i} s_j + b_i\right)$$

10.6 a differentiable generator network transforms samples of latent variables z to

9.7 $E_z[J(y)] = \sum_y J(y)p(y)$

$$\frac{\partial E[J(y)]}{\partial w} = \sum_y J(y)\frac{\partial p(y)}{\partial w}$$

// asume J does not reference w directly, but we can relax the assumption.

$$= \sum_y J(y)p(y)\frac{\partial \log p(y)}{\partial w}$$

$$\approx \frac{1}{m}\sum_{y^{(i)}\sim p(y),\, i=1}^{m} J(y^{(i)})\frac{\partial \log p(y^{(i)})}{\partial w}$$

// an unbiased Monte Carlo estimator

// simple REINFORCE estimator.

10. directed generative nets

10.2 While directed graphical models have been very popular within the greater machine learning community, within the smaller deep learning community, they have until roughly 2013 been overshadowed by undirected models such as the RBM.

is often called the reparametrization trick, stochastic back-propagation or perturbation analysis.

9.5 If we wish to back-propagate through a sampling process that produces discrete-valued samples, it may still be possible to estimate a gradient on w, using reinforcement learning algorithms such as variants of the REINFORCE algorithm.

9.6 The core idea of the REINFORCE algorithm is that even though $J(f(z;w))$ is a step-function with useless derivatives, the expected cost $E_{z\sim p(z)} J(f(z;w))$ is often a smooth function amenable to gradient descent.

(ii) The two networks are trained with a technique called moment matching.

(iii) generative moment matching networks can be trained by minimizing a cost function called maximum mean discrepancy or MMD. The cost function measures the error in the first moments in an infinite-dimensional space, using an implicit mapping to feature space defined by a kernel function in order to make computations on infinite-dimensional vectors tractable. The MMD cost is zero if and only if the two distributions being compared are equal.

the training data and samples drawn from the generator. The discriminator emits a probability value given by $d(x; \theta^{(d)})$, indicating the probability that x is a real training example rather than a fake sample drawn from the model.

10.9 generative moment matching networks

(i) generative moment matching networks are another form of generative model based on differentiable generator networks. Unlike VAEs and GANs, they do not need to pair the generator network with any other network — neither an inference network as used with VAEs nor a discriminator network as used with GANs.

encoder and decoder.

(iv) The VAE approach is among the state of the art approaches to generative modeling.

(v) VAE is an excellent manifold learning algorithm.

10.8 generative adversarial networks (GANs)

(i) GANs are another generative modeling approach based on differentiable generator networks.

(ii) GANs are based on a game theoretic scenario in which the generator network must compete against an adversary. $q(z|x) = q(z; f(x; \theta))$ The generator network directly produces samples $x = g(z; \theta^{(g)})$. Its adversary, the discriminator network, attempts to distinguish between samples drawn from

$p_{model}(x; g(z)) = p_{model}(x|z)$ However, during training, the approximate inference network (encoder) $q(z|x)$ is used to obtain z and $p_{model}(x|z)$ is then viewed as a decoder network.

(iii) The main idea behind VAE is to train a parametric encode (also called an inference network that produces the parameters of q. So long as z is a continuous variable, we can then back-propagate through

then consists solely of maximizing L with respect to the parameters of the

denoising autoencoders:

(i) starting from the previous state x, inject corruption noise, sampling $\tilde{x}$ from $C(\tilde{x}|x)$.

(ii) encode $\tilde{x}$ into $h=f(\tilde{x})$.

(iii) decode h to obtain the parameters $\omega=g(h)$ of $p(x|\omega=g(h))=p(x|\tilde{x})$

(iv) sample the next state x from $p(x|\omega=g(h))=p(x|\tilde{x})$

12. generative stochastic networks (GSNs)

12.2 GSNs are generalizations of denoising autoencoders that include latent variables h in the generative Markov chain, in addition to the visible variables x.

13. generative models hold the promise to provide AI systems with a framework for all of the many different intuitive concepts they need to understand, and the ability to reason about these concepts in the face of uncertainty.

increased as much as needed. The new parametrization can also improve generalization by introducing a parameter sharing and feature sharing principle.

10.13 neural autoregressive density estimator (NADE)

(i) NADE is a very successful recent form of neural auto-regressive network. The parameters of the hidden units of different group are shared.

(ii) forward propagation in a NADE model loosely resembles the computations performed in mean field inference to fill in missing inputs in an RBM.

11. drawing samples from autoencoders

11.2 one step of Markov chain for generalized

Each $p(x_i|x_{i-1},\cdots,x_1)$ is parametrized as a linear model.

(ii) linear auto-regressive nets are essentially the generalization of linear classification methods to generative modeling.

(iii) like linear classifiers, the model itself does not offer a way of increasing its capacity.

10.12 neural auto-regressive nets have the same left-to-right graphical model as logistic auto-regressive networks but employ a different parametrization of the conditional distributions within that graphical model structure. The new parametrization is more powerful in the sense that its capacity can be

10.10 auto-regressive networks

auto-regressive networks are directed probabilistic models with no latent random variables. The conditional probability distributions in these models are represented by neural networks. They decompose a joint probability over the observed variables using the chain rule of probability to obtain a product of conditionals of the form $p(x_d|x_{d-1},\cdots,x_1)$. Such models have been called fully-visible Bayes networks (FVBNs).

10.11 linear auto-regressive nets

(i) linear auto-regressive network is the simplest form of auto-regressive network which has no hidden units and no sharing of parameters or features.

2.2 algebraic structures

set S

1. magma $(S,\cdot)$ $\Leftrightarrow$ set with a binary operation $\cdot: S\times S \to S$.

↓

semigroup $\Leftrightarrow$ associative magma

↓

monoid $\Leftrightarrow$ semigroup with identity

↓

group $\Leftrightarrow$ monoid with existence of inverses

↓

abelian group (or commutative group) $\Leftrightarrow$ group obeying axiom of commutativity.

2. ring $(R,+,\cdot)$

$\Leftrightarrow$ ① $(R,+)$ is abelian group

② $(R,\cdot)$ is monoid.

③ multiplication is distributive with respect to addition

$a\cdot(b+c)=(a\cdot b)+(a\cdot c)$

$(b+c)\cdot a=(b\cdot a)+(c\cdot a)$

$\forall a,b,c\in R$

rng (or pseudo-ring or non-unital ring)

$\Leftrightarrow$ ring without assuming the existence of multiplicative identity.

2.3. field $(F,+,\cdot)$

$\Leftrightarrow$ ① $(F,+)$ is abelian group.

② $(F,\cdot)$ is abelian group except 0 doesn't have multiplicative inverse.

$1\neq 0$.

③ distributivity of multiplication over addition.

$a\cdot(b+c)=(a\cdot b)+(a\cdot c)$

$\forall a,b,c\in F$.

2.4

field $\subset$ ring $\subset$ rng

3. semilattice $(S,\wedge)$

$\Leftrightarrow$ $(S,\wedge)$ is a commutative, idempotent semigroup.

where idempotency means: $x\wedge x=x$, $\forall x\in S$.

3.2 lattice $(L,\vee,\wedge)$

$\Leftrightarrow$ ① $(L,\vee)$ is commutative semigroup.

② $(L,\wedge)$ is commutative semigroup.

③ absorption law.

$a\vee(a\wedge b)=a$

$a\wedge(a\vee b)=a$

$\forall a,b\in L$.

4. 3.3 bounded lattice $(L,\vee,\wedge,0,1)$

$\Leftrightarrow$ $(L,\vee,\wedge)$ is lattice

0 is identity element for $\vee$.

1 is identity element for $\wedge$.

4. module:

R is a ring, left R-module $(M,+,\cdot)$

$\Leftrightarrow$ $(M,+)$ is an abelian group.

$\cdot: R\times M\to M$

satisfying, $\forall r,s\in R$, $x,y\in M$

① $r\cdot(x+y)=r\cdot x+r\cdot y$

② $(r+s)\cdot x=r\cdot x+s\cdot x$

③ $(r\cdot s)\cdot x=r\cdot(s\cdot x)$

④ $1_R\cdot x=x$, 1_R is multiplicative identity of R.

5. algebra A over a field K

$\Leftrightarrow$ A is a vector space over K, with a binary operation $\cdot: A\times A\to A$, satisfying:

$\forall x,y,z\in A$, $a,b\in K$,

① $(x+y)\cdot z=x\cdot z+y\cdot z$

② $x\cdot(y+z)=x\cdot y+x\cdot z$

③ $(ax)\cdot(by)=(ab)(x\cdot y)$

6. algebraic structure is a set with one or more operations defined on it that satisfies a list of axioms.

6.2 examples: groups, rings, fields, lattices, vector spaces, modules, algebras.

4.2 vector space $(V,+,\cdot)$ over a field F.

$\Leftrightarrow$ $(V,+,\cdot)$ is left R-module, where F is the ring.

7. $f: R^m \to R$

$f'(x)=0$, $f''(x)$ positive definite $\Rightarrow$ x is local minimum.

proof: $\forall u$ is a unit direction,

$\frac{\partial^2 f}{\partial u^2} = u^T (f'(x))^T u > 0$

($\because f''$ pd $\Rightarrow (f'')^T$ pd.)

$\because \frac{\partial f}{\partial u} = (f'(x))^T u = 0$.

$\therefore x$ is local minimum along u. (get from 6).

$\because u$ is arbitrary direction

$\therefore x$ is local minimum of f.

7.2. $f'(x)=0$, $f''(x)$ negative definite $\Rightarrow x$ is local maximum.

7.3. $f'(x)=0$, $f''(x)$ symmetric, $\lambda_1>0, \lambda_2<0$ are eigenvalues of $f''(x)$, $\Rightarrow x$ is saddle point.

5. f'' measures curvature.

if $f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2}(x-a)^2 + o((x-a)^2)$

If $f''(a) > 0$,

set $h(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2}(x-a)^2$

if $f'(a)=0$, $h(x)$ is $\cup$,

if $f'(a)<0$, $h(x)$ is $\cap$,

is $f'(a)=0$, $h(x)$ is —

$\because f(x) = h(x) + o((x-a)^2)$

$\therefore f(x)$ can be approximated using $h(x)$.

$\therefore f''(a)$ also measures curvature of f at a.

6. second derivative test

$f: R \to R$, $f'(x)=0, f''(x)>0$

$\Rightarrow x$ is local minimum.

$f'(x)=0, f''(x)<0 \Rightarrow$ x is local maximum.

(we can see that from 5)

$\therefore H = (g'(x))^T$

$f''(x) := (f'(x))'$

$= H^T$

H is called Hessian matrix of f.

$\frac{\partial^2 f}{\partial u^2} := \frac{\partial}{\partial u}\left(\frac{\partial f}{\partial u}\right)$

$= \frac{\partial}{\partial u}(u^T g)$

$= u^T \frac{\partial g}{\partial u}$

$= u^T H u$

$\therefore \frac{\partial^2 f}{\partial u^2} = u^T (f''(x))^T u$

4. Taylor's theorem.

$f: R \to R$, is k times differentiable, at a, $\Rightarrow \exists h_k: R \to R$, such that

$f(x) = f(a) + f'(a)(x-a) + \cdots + \frac{f^{(k)}(a)}{k!}(x-a)^k + h_k(x)(x-a)^k$,

and $\lim_{x \to a} h_k(x) = 0$

Chapter 4.

1. $f(x): R^2 \to R$

$\|u\|=1$, u is a direction, and fixed,

$\Rightarrow \frac{\partial f}{\partial u} = \lim_{\alpha \to 0} \frac{f(x+\alpha u) - f(x)}{\alpha}$

$= u^T \nabla_x f(x) = (f'(x))^T u$

where $f'(x) := \nabla_x f(x) \in R^2$

2. $f(x): R^m \to R^2$, $\|u\|=1$, u is direction

$\therefore f(x) = \begin{pmatrix} f_1(x) \\ f_2(x) \end{pmatrix}$

$\frac{\partial f}{\partial u} := \begin{pmatrix} \frac{\partial f_1}{\partial u} \\ \frac{\partial f_2}{\partial u} \end{pmatrix} = \begin{pmatrix} (\nabla_x f_1)^T u \\ (\nabla_x f_2)^T u \end{pmatrix} = Ju$

where $J := \begin{pmatrix} (\nabla_x f_1)^T \\ (\nabla_x f_2)^T \end{pmatrix}$, is Jacobian of f.

If $f'(x) := J^T \in R^{m \times 2}$

then $\frac{\partial f}{\partial u} = Ju = (f'(x))^T u$

3. $f(x): R^2 \to R$

let $g(x) := f'(x) = \nabla_x f(x)$

$\therefore g(x): R^2 \to R^2$

set H is Jacobian of g,

proof:

$$f(x)=f(x_0)+(x-x_0)f'(x_0)+\frac{1}{2}(x-x_0)^2f''(x_0)+o((x-x_0)^2)$$

let $x=x_0\pm\alpha$, $\because f'(x_0)=0$

$$\therefore f(x_0\pm\alpha)=f(x_0)+\frac{1}{2}\alpha^2f''(x_0)+o(\alpha^2)$$

$$\therefore f(x_0\pm\alpha)-g(x_0\pm\alpha)=\frac{1}{2}\alpha^2(f''(x_0)-g''(x_0))+o(\alpha^2)$$

from 8, we know, $\exists\delta>0$, $\forall\alpha\in O(0,\delta)$, $f(x_0\pm\alpha)>g(x_0\pm\alpha)$.

11. $f:R^m\to R$, f', f'' exist, x_0 is local minimum, $\lambda_1>\lambda_2$

$\Rightarrow f(x_0\pm\alpha v_1)>f(x_0\pm\alpha v_2)$ $\exists\delta>0, \forall\alpha\in O(0,\delta)$,

proof: $\because x_0$ is local minimum.

$\therefore f''$ is symmetric,

λ_1,λ_1 is eigenvalues of f''.

v_1,v_2 is eigenvectors of f''.

$\because \lambda_1>\lambda_2$, $\therefore \frac{\partial^2 f}{\partial v_1^2}>\frac{\partial^2 f}{\partial v_2^2}$

from 10, we get the conclusion.

8. $f(x)=kx+o(x)$, $k>0$

$\Rightarrow \exists\delta>0, \forall x\in(0,\delta)$, we have $f(x)>0$.

proof: from 9, we know, $\exists\delta>0, \forall x\in(0,\delta)$,

$$|o(x)|<\frac{k}{2}|x|$$

$$\therefore o(x)>-\frac{k}{2}x, \forall x\in(0,\delta)$$

$$\therefore f(x)=kx+o(x)>\frac{k}{2}x>0.$$

$\forall x\in(0,\delta)$.

9. $f(x)=o(g(x))$, $x\to a$

$$\Leftrightarrow \lim_{x\to a}\frac{f(x)}{g(x)}=0$$

$\Leftrightarrow \forall\varepsilon>0, \exists\delta>0, \forall x\in O(a,\delta)$,

$$\left|\frac{f(x)}{g(x)}\right|<\varepsilon$$

$\Leftrightarrow \forall\varepsilon>0, \exists\delta>0, \forall x\in O(a,\delta)$,

we have $|f(x)|<\varepsilon|g(x)|$, $g(x)\neq 0$.

10. $f:R\to R$, $g:R\to R$,

at x_0 f', f'', g', g'' exist.

x_0 is local minimum of f, g.

$f''(x_0)>g''(x_0)$, $f(x_0)=g(x_0)$

$\Rightarrow \exists\delta>0, \forall\alpha\in O(0,\delta), f(x_0\pm\alpha)>g(x_0\pm\alpha)$

5. PCA

5.2 PCA finds a representation $z^T = x^T W$, where $Var(z)$ is diagonal.

Here W comes from SVD, $X = U\Sigma W^T$.

proof: $Var(z) = \frac{1}{m-1} z^T z$

$= \frac{1}{m-1}(XW)^T(XW)$

$= \frac{1}{m-1} W^T X^T X W$

$= \frac{1}{m-1} W^T (W\Sigma^2 W^T) W$

$= \frac{1}{m-1}\Sigma^2$

5.3 W in PCA is a rotation of x that aligns the principal axes of variance with the basis of new representation space associated with z.

5.4. PCA is an unsupervised learning algorithm that learns a representation of data. The representation has a lower dimension. The elements of the representation have no linear correlation with each other.

proof: $E[\tilde{\sigma}_m^2]$

$= \frac{m}{m-1} E[\hat{\sigma}_m^2]$

$= \frac{m}{m-1} \cdot \frac{m-1}{m}\sigma^2$

$= \sigma^2$

∴ proved.

3. $\{x^{(1)}, \cdots, x^{(m)}\}$ are points, $x^{(i)} \in R^n, \forall i$,

let $X \in R^{m \times n}$, $X_{i,:} = x^{(i)T}$

if $E(x^{(i)}) = 0, \forall i$,

then unbiased sample covariance matrix is $\frac{1}{m-1}X^T X$.

4. unsupervised learning task is to find "best" representation of data.

4.2 three criteria of a simple representation.

① lower dimensional representation

② sparse representation

③ independent representation.

∵ $EA = \sum_{i=1}^m E(x^{(i)2})$

$= \sum_{i=1}^m (Var(x^{(i)}) + (E(x^{(i)}))^2)$

$= m\sigma^2 + m\mu^2$

$EB = mE(\hat{\mu}^2)$

$= mE\left(\left(\frac{\sum x^{(i)}}{m}\right)^2\right)$

$= \frac{1}{m} E\left((\sum x^{(i)})^2\right)$

$= \frac{1}{m}\left(Var(\sum x^{(i)}) + (E(\sum x^{(i)}))^2\right)$

$= \frac{1}{m}\left(\sum Var(x^{(i)}) + (\sum E(x^{(i)}))^2\right)$

$= \frac{1}{m}(m\sigma^2 + m^2\mu^2)$

$= \sigma^2 + m\mu^2$

∴ $E[\hat{\sigma}_m^2] = \frac{1}{m}E[A-B]$

$= \frac{m-1}{m}\sigma^2$

∴ $\hat{\sigma}_m^2$ is biased.

2.3. unbiased sample variance estimator.

$\tilde{\sigma}_m^2 = \frac{1}{m-1}\sum_{i=1}^m (x^{(i)} - \hat{\mu}_m)^2$

⇒ $\tilde{\sigma}_m^2$ is unbiased.

Chapter 5.

1. A is real symmetric matrix

$f(x) = x^T A x$, $\|x\| = 1$.

⇒ $\min f(x)$ is minimum eigenvalue of A.

$\max f(x)$ is maximum eigenvalue of A.

2. Estimators of variance of Gaussian distribution.

2.2 sample variance of σ^2

$\hat{\sigma}_m^2 = \frac{1}{m}\sum_{i=1}^m (x^{(i)} - \hat{\mu}_m)^2$

where $\hat{\mu}_m = \frac{1}{m}\sum x^{(i)}$ is sample mean

$\{x^{(1)}, \cdots, x^{(m)}\}$ are i.i.d. according to $p(x^{(i)}) = N(x^{(i)}; \mu, \sigma^2), \forall i \in \{1, \cdots, m\}$.

⇒ $\hat{\sigma}_m^2$ is biased estimator.

proof: $E[\hat{\sigma}_m^2] = E\left[\frac{1}{m}\sum_{i=1}^m (x^{(i)} - \hat{\mu}_m)^2\right]$

$= \frac{1}{m} E\left[\sum (x^{(i)2} + \hat{\mu}_m^2 - 2\hat{\mu}_m x^{(i)})\right]$

$= \frac{1}{m} E\left[\sum (x^{(i)})^2 - m\hat{\mu}^2\right]$

$= \frac{1}{m} E(A - B)$