Contents

1 Optimization for Machine Learning, Elad Hazan

`http://www.cs.princeton.edu/~ehazan/tutorial/SimonsTutorial.htm`

The paradigm is as follows: we have a machine we want to train on inputs, like images to classify. The distribution is over vectors in $\mathbb{R}^n$, and the output is a label $b = f_\theta(a)$ where θ are the parameters. The behavior of the function is set by the parameters.

We care about training the machine *efficiently* and so that it *generalizes*.

We will cover

1. Learning as mathematical optimization
2. Regularization
3. Gradient descent++ (Frank-Wolfe, acceleration, variance reduction...)

1.1 Math optimization

The input is a function $f : K \to \mathbb{R}$ for $K \subseteq \mathbb{R}^d$. The output is a minimizer $x \in K$ such that $f(x) \leq f(y)$ for all $y \in K$.

How can we access f? We assume we can access values and derivatives. We don't have to work in the oracle model; sometimes we know the function (e.g., a polynomial). Even in the non-oracle model, the problem can easily be NP-hard.

Learning is the same as optimization over data (a.k.a. empirical risk minimization) of the function

$$\text{argmin}_{x \in \mathbb{R}^d} \frac{1}{m} \sum_{i=1}^{m} \ell_i(x, a_i, b_i) + R(x)$$

where m is the number of examples, (a, b) are pairs of (feature, label), d is the dimension, and x is the parameter. Training means fitting the parameters of the model.

For example, in linear classification, we try to find a hyperplane which separates points of one class from points of another class. Given a sample $S = \{(a_1, b_1), \ldots, (a_m, b_m)\}$ where $b_i \in \{\pm 1\}$, find a hyperplane (WLOG through the origin) minimizing the number of mistakes:

$$\text{argmin}_{\|x\| \leq 1} \left| \left\{ i : \text{sign}(x^T a_i) \neq b_i \right\} \right|.$$

We can put it in the form we had before

$$\text{argmin}_{\|x\|\leq 1} \frac{1}{m}\sum_{i=1}^{m} \ell(x,a,b), \quad \ell(x,a_i,b_i) = \begin{cases} 1, & x^T a \neq b \\ 0, & x^T a = b. \end{cases}$$

This is an example of how to convert a learning problem to an optimization problem. This simple problem is already NP-hard.

Why? The sum of sign functions can be any piecewise constant function (with finitely many pieces), which can be complicated.

Is there a local property that ensures global optimality? Yes, convexity.

Definition 1.1: A continuous function $f : \mathbb{R}^d \to \mathbb{R}$ is **convex** iff

$$f\left(\frac{1}{2}(x+y)\right) \leq \frac{1}{2}f(x) + \frac{1}{2}f(y),$$

or equivalently (for differentiable functions),

$$f(y) \geq f(x) + \nabla f(x)^T (y-x).$$

Informally the function looks like a smiley.

Similarly we can define convex sets.

Definition 1.2: A closed set K is **convex** iff $\frac{1}{2}(x+y) \in K$ whenever $x, y \in K$.

To make the linear (or kernel) classification problem tractable, we consider convex relaxations such as the following loss functions $\ell(x, a_i, b_i)$.

1. Ridge/linear regression $(x^T a_i - b_i)^2$
2. SVM (the most popular method a decade ago) $\max\{0, 1 - b_i x^T a_i\}$.
3. Logistic $\ln(1 + e^{-b_i x^T a_i})$.

We have cast learning as mathematical optimization and argued convexity is algorithmically important. Next we cover algorithms.

1.1.1 Gradient descent

The most naive method is gradient descent. It is a local algorithm where we move in the direction of steepest descent.

Algorithm 1.3 (Gradient descent)**:**

$$-[\nabla f(x)]_i := -\frac{\partial}{\partial x_i} f(x) \tag{1}$$

$$y_{t+1} \leftarrow x_t - \eta \nabla f(x_t) \tag{2}$$

$$x_{t+1} = \text{argmin}_{x\in K} |y_{t+1} - x|. \tag{3}$$

Note the second step is necessary if we are optimizing over a subset K of $\mathbb{R}^n$; we have to project back to the set.

Theorem 1.4. *For step size* $\eta = \frac{D}{G\sqrt{T}}$

$$f\left(\frac{1}{T}\sum_t x_t\right) \leq \min_{x^* \in K} f(x^*) + \frac{DG}{\sqrt{T}}$$

where G is upper bound on norm of gradients and D is the diameter of the constraint set:

$$\forall t, \quad \|\nabla f(x_t)\| \leq G \tag{4}$$
$$\forall x, y \in K, \quad \|x - y\| \leq D. \tag{5}$$

Proof. We make 2 observations.

$$|x^* - y_{t+1}|^2 = |x^* - x_t|^2 - 2\eta \nabla f(x_t)(x_t - x^*) + \eta^2 |\nabla f(x_t)|^2 \tag{6}$$
$$|x^* - x_{t+1}|^2 \leq |x^* - y_{t+1}|^2 \tag{7}$$

The second follows from the Pythagorean theorem because the angle made at x_{t+1} is obtuse.

Combining these results and telescoping.

$$|x^* - x_{t+1}|^2 \leq |x^* - x_t|^2 - 2\eta \nabla f(x_t)(x_t - x^*) + G^2 \tag{8}$$
$$f\left(\frac{1}{T}\sum_t x_t\right) - f(x^*) \leq \frac{1}{T}\sum_t \left[f\left(\sum_t x_t\right) - f(x^*)\right] \qquad \text{convexity} \tag{9}$$
$$\leq \frac{1}{T}\sum_t \nabla f(x_t)(x_t - x^*) \tag{10}$$
$$\leq \frac{1}{T}\sum_t \frac{1}{2\eta}(|x^* - x_{t+1}|^2 - |x^* - x_t|^2) + \frac{\eta}{2}G^2 \tag{11}$$
$$\leq \frac{1}{T2\eta}D^2 + \frac{\eta}{2}G^2 \leq \frac{DG}{\sqrt{T}} \tag{12}$$

with our choice of η. □

Note we showed that the average of the points converges. Under more conditions we can show the last point converges. There are examples in the stochastic setting where the average but not the last point converges.

Thus, to get ε-approximate solution, apply GD $O\left(\frac{1}{\varepsilon^2}\right)$ times.

1.1.2 Online/stochastic gradient descent and ERM

This is not suited to what we are looking for. For ERM problems we want

$$\text{argmin}_{x \in \mathbb{R}^d} \frac{1}{m}\sum_{i=1}^m \ell_i(x, a_i, b_i) + R(x)$$

The gradient depends on *all* data, which is inefficient, and we haven't considered generalization.

We consider simultaneous optimization and generalization. We have to recall that our data came from a distribution. We use the statistical (PAC) learning model.

- Nature chooses iid from a distribution D over $A \times B = \{(a, b)\}$.
- The learner chooses a hypothesis $h \in H$.
- There is a loss function ℓ, such as $\ell(h, (a, b)) = (h(a) - b)^2$.
- The error is $err(h) = \mathbb{E}_{a,b\sim D}[\ell(h, (a, b))]$.

Definition 1.5: We say that the hypothesis class H of functions $X \to Y$ if **learnable** if for all $\varepsilon, \delta > 0$ there exists an algorithm such that after seeing m examples, for $m = \text{poly}(\delta, \varepsilon, \dim(H))$, it finds h such that w.p. $1 - \delta$,

$$err(h) \leq \min_{h^* \in H} err(h^*) + \varepsilon.$$

A more powerful setting is online learning in games.

1. Player picks $h_t \in H$.
2. Adversary chooses $(a_t, b_t) \in A$.
3. Loss function is ℓ.

There is no distribution. The aim is to minimize

$$\frac{1}{T}\left[\sum_t \ell(h_t, (a_t, b_t)) - \min_{h^* \in \mathcal{H}} \sum_t \ell(h^*, (a_t, b_t))\right].$$

We say that H is **learnable** in this setting if this quantity (the regret) approaches 0 as $T \to \infty$. This is not a priori clear that it can be done. Note your algorithm is allowed to change, and you compare to a fixed h^*. Vanishing regret in this setting implies generalization in the PAC setting; it is strictly more general.

From this point onwards, we consider $f_t(x) = \ell(x, a_t, b_t)$, the loss for one example.

Can we minimize regret efficiently?

There is a natural analogue of gradient descent, online gradient descent. The only difference is that we take a step with respect to the gradient of the *current* (possibly adversarially chosen!) function f_t.

Algorithm 1.6 (Online/stochastic gradient descent)**:**

$$y_{t+1} = x_t - \eta \nabla f_t(x_t) \tag{13}$$
$$x_{t+1} = \text{argmin}_{x \in K} |y_{t+1} - x|. \tag{14}$$

This may look strange at first glance: we take a step using f_t even though we may never see this function again!

(Notes: We assume we can always see the gradient. There are extensions that work even if you only see the loss, the bandit optimization problem.) .

Theorem 1.7 (Zinkevich)**.** *The regret of online gradient descent is*

$$\sum_t f_t(x_t) - \sum_t f_t(x^*) = O(\sqrt{T}).$$

The proof is similar to the previous proof. The only difference is that the gradient function is different at each step.

Proof. We make the same 2 observations.

$$|x^* - y_{t+1}|^2 = |x^* - x_t|^2 - 2\eta \underbrace{\nabla f_t(x_t)}_{:=\nabla_t}(x_t - x^*) + \eta^2 |\nabla f_t(x_t)|^2 \tag{15}$$

$$|x^* - x_{t+1}|^2 \leq |x^* - y_{t+1}|^2 \tag{16}$$

Combining these results and telescoping,

$$|x^* - x_{t+1}|^2 \leq |x^* - x_t|^2 - 2\eta \nabla_t (x_t - x^*) + \eta^2 \|\nabla_t\|^2 \tag{17}$$

$$f\left(\sum_t x_t\right) - f(x^*) \leq \sum_t \nabla_t (x_t - x^*) \tag{18}$$

$$\leq \sum_t \frac{1}{2\eta}(|x^* - x_{t+1}|^2 - |x^* - x_t|^2) + \frac{\eta}{2}\sum_t \|\nabla_t\|^2 \tag{19}$$

$$\leq \frac{1}{\eta}|x_1 - x^*|^2 + \eta T G \leq DG\sqrt{T} \tag{20}$$

□

This is tight. For the lower bound, take $K = [-1, 1]$, $f_1(x) = x$, $f_2(x) = -x$. The expected loss is 0, and the regret compared to either $-1, 1$ is on the order of the variance.

$$\mathbb{E}|\#1's - \#-1's| = \Omega(\sqrt{T}).$$

This gives rise to the most important problem in optimization for ML, stochastic gradient descent. The learning problem is

$$\text{argmin}_{x \in \mathbb{R}^d} F(x)$$

where

$$F(x) = \mathop{\mathbb{E}}_{(a_i, b_i)}[\ell(x, a_i, b_i)].$$

Use online gradient descent where at each step we take a random example $f_t(x) = \ell_i(x, a_i, b_i)$.

We have proved that

$$\frac{1}{T}\sum_t \nabla_t^T x_t \leq \min_{x^* \in K} \frac{1}{T}\sum_t \nabla_t^T x^* + \frac{DG}{\sqrt{T}}.$$

Taking expectation we achieve the same bound as in gradient descent. If m is the number of examples, we've moved from $O\left(\frac{d}{\varepsilon^2}\right)$ rather than $O\left(\frac{md}{\varepsilon^2}\right)$ steps for ε generalization error with slight degradation because we only get a result in expectation.

$$\mathbb{E}\left[F\left(\frac{1}{T}\sum_t x_t\right) - \min_{x^* \in K} F(x^*)\right] \leq \mathbb{E}\left(\frac{1}{T}\sum_t \nabla_t^T (x_t - x^*)\right) \leq \frac{DG}{\sqrt{T}}.$$

(This is easier than the perceptron algorithm which requires finding a misclassified point at each step.)

1.2 Regularization

What is regularization and why do it? Statistical learning theory (Occam's razor) says that the number of examples needed to learn a hypothesis depends on the "dimension" which depending on the setting, could be

- VC dimension
- fat-shattering dimension
- Rademacher width
- margin/norm of linear/kernel classifier.

An expressing hypothesis class can overfit.

In PAC theory, regularization reduces complexity of the hypothessis.

In the regret minimization framework, regularization helps stability.

(Regularization also helps for the purpose of optimization—adding a convex function can distort the function to become convex. We don't consider this here. Note this may hurt generalization.)

We are trying to minimize regret (loss compared to best in hindsight). The most natural is to take

$$x_t = \text{argmin}_{x \in K} \sum_{i=1}^{t-1} f_i(x).$$

This doesn't work: consider the $\pm x$ example, when the adversary always chooses the opposite. This is called fictitious play in economics. This fails because of instability, the loss function can vary wildly each step.

This modification provably works (Kalai-Vempala 2005):

$$x'_t = \text{argmin}_{x \in K} \sum_{i=1}^{t} f_i(x) = x_{t+1}.$$

If $x_t \approx x_{t+1}$ we get a regret bound. However, the instability $|x_t - x_{t+1}|$ can be large.

We alter this further: pick the best point in hindsight, but add a term to make it stable.

Algorithm 1.8 (Follow The Regularized Leader)**:**

$$x_t = \text{argmin}_{x \in K} \sum_{i=1}^{t-1} \nabla_t^T x + \frac{1}{\eta} R(x)$$

where $R(x)$ is a strongly convex function.

Adding R ensures stability.

Theorem 1.9. *FTRL achieves regret*

$$\nabla_t^T (x_t - x_{t+1}) = O(\eta).$$

In economics this is called smooth fictitious play. There is a "center of mass" effect that pulls you towards a solution.

What to choose for R? The most obvious is $R(x) = \frac{1}{2}\|x\|^2$. For linear cost functions we recover gradient descent (π_K is projection to K)

$$x_t = \text{argmin}_{x \in K} \sum_{i=1}^{t-1} \nabla f_i(x)^T x + \frac{1}{\eta} R(x) = \pi_K(-\eta \sum_{i=1}^{t-1} \nabla f_i(x_i)).$$

There are many interesting algorithms you can recover by this paradigm, for example multiplicative weights. Take $f_t(x) = c_t^T x$ where c_t is vector of losses, $R(x) = \sum_i x_i \ln x_i$ the negative entropy. Then

$$x_t = \exp(-\eta \sum_{i=1}^{t-1} c_i)/Z_t$$

where Z_t is a normalization constant.

1.3 Gradient descent++

We cover AdaGrad, variance reduction, ande acceleration/momentum.

1.3.1 AdaGrad

What regularization should we choose? If we have a generalized linear model, then OGD update is inefficient if we have sparse data because it treats all coordinates the same way. Sparse data is common. Adaptive regularization copes with sparse data.

Which regularization to pick? The idea of AdaGrad is to treat choosing the R as a learning problem itself. Consider the family of regularizations

$$R(x) = \|x\|_A^2, \quad A \succeq 0, \quad \text{Tr}(A) = d.$$

This is finding the best regret in hindsight for a matrix optimization problem.

Algorithm 1.10 (AdaGrad)**:**

$$G_t = \text{diag}\left(\sum_{i=1}^{t} \nabla f_i(x) \nabla f_i(x_i)^T\right) \tag{21}$$

$$y_{t+1} = x_t - \eta G_t^{-\frac{1}{2}} \nabla f_t(x_t) \tag{22}$$

$$x_{t+1} = \text{argmin}_{x \in K} (y_{t+1} - x)^T G_t (y_{t+1} - x). \tag{23}$$

Theorem 1.11. *AdaGrad gives regret bound*

$$O\left(\sum_i \sqrt{\sum_t \nabla_{t,i}^2}\right).$$

This regret bound can be $\sqrt{d}$ better than SGD. The $\frac{1}{\sqrt{T}}$ in SGD was tight, but the constants can be improved by AdaGrad.

1.3.2 Variance reduction

Variance reduction uses special ERM structure and is very effective for smooth and convex functions.

Acceleration/momentum works for smooth convex functions only; it is used in general purpose optimization since the 80's.

Definition 1.12: If $0 \prec \alpha I \preceq \nabla^2 f(x) \preceq \beta I$, then the condition number if $\gamma = \frac{\beta}{\alpha}$. We say that f is α-strongly convex if the first inequality holds, β-smooth if the second inequality holds.

A convex function always satisfies this with some $\alpha \geq 0$. We can also talk about smoothness for non-convex functions:

$$-\beta I \preceq \nabla^2 f(x) \preceq \beta I.$$

Why do we care? Well-conditioned functions exhibit faster optimization; they can be optimized in polynomial time. Gradient descent is not polytime per se. Polytime means a bound logarithmical in approximation.

Smoothness is important in second-order methods

The loss function in ridge and logistic regression are strongly convex and smooth.

For smooth functions, a gradient step causes decrease in function value proportional to the value of the gradient. Taking $\eta = \frac{1}{2\beta}$,

$$f(x_{t+1}) - f(x_t) \leq -\nabla_t(x_{t+1} - x_t) + \beta|x_t - x_{t+1}|^2 \tag{24}$$

$$= -(\eta + \beta\eta^2)|\nabla_t|^2 = -\frac{1}{4\beta}|\nabla_t|^2. \tag{25}$$

Lemma 1.13 (Gradient descent lemma)**.** *For β-smooth functions, $f(x_{t+1}) - f(x_t) \leq -\frac{1}{4\beta}|\nabla_t|$.*

For M-bounded functions, what happens when we take many steps?

$$-2M \leq f(x_T) - f(x_1) \leq \sum_t [f(x_{t+1}) - f(x_t)] \leq -\frac{1}{4\beta}\sum_i |\nabla_i|^2.$$

There exists t for which

$$|\nabla_t|^2 \leq \frac{8M\beta}{T}.$$

1. Note we didn't use convexity. This is pretty much the only thing we can say for nonconvex functions.

2. Note for convex functions, we get a quadratic improvement: for $T = \Omega\left(\frac{1}{\varepsilon}\right)$ we get $|\nabla_t|^2 \leq \varepsilon$.

For nonconvex optimization, we can't hope for global optimality, but we can hope for local optimality (gradient vanishes).

This is nice but not practical for ML. We're taking full gradient steps; we need to go over the entire data set.

Let's look at the stochastic version. Take a step in direction $\widetilde{\nabla}_t$ where $\mathbb{E}\widetilde{\nabla}_t = \nabla_t$.

$$\mathbb{E}[f(x_{t+1}) - f(x_t)] \leq \mathbb{E}[-\nabla_t(x_{t+1} - x_t) + \beta|x_t - x_{t+1}|^2] \tag{26}$$
$$\leq \mathbb{E}[-\widetilde{\nabla}_t \cdot \eta\nabla_t + \beta|\widetilde{\nabla}_t|^2] \tag{27}$$
$$= -\eta\nabla_t^2 + \eta^2\beta\mathbb{E}|\widetilde{\nabla}_t|^2 \tag{28}$$
$$= -\eta\nabla_t^2 + \eta^2\beta(\nabla_t^2 + \mathrm{Var}(\widetilde{\nabla}_t)). \tag{29}$$

Theorem 1.14. *For gradient descent for β-smooth, M-bounded functions, for $T = O\left(\frac{M\beta}{\varepsilon^2}\right)$, there exists $t \leq T$, $|\nabla_t|^2 \leq \varepsilon$.*

In practice, we take minibatches: take 10 or 100 examples instead of 1. This decreases variance, and is amenable to computer architecture.

In theory, tune step size according to these parameters. In practice, take a logarithmic scale, and test those step sizes.

Consider a hybrid model: sometimes compute the full gradient and sometimes take only the gradient of 1 example. This interpolates between GD and SGD.

Estimator combines both to create a random variable with lower variance.

Algorithm 1.15 (SVRG)**:**

$$x_{t+1} = x_t - \eta[\widetilde{\nabla}f(x_t) - \widetilde{\nabla}f(x_0) + \nabla f(x_0)].$$

Every so often, compute the full gradient and restart at new x_0.

Theorem 1.16 (Schmidt, LeRoux, Bach 12; Johnson, Zhang 13, Mahdavi, Zhang, Jin 13)**.** *Variance reduction for γ-well-conditioned functions produces an ε-approximate solution in*

$$O\left((m+\gamma)d\ln\left(\frac{1}{\varepsilon}\right)\right)$$

γ should be interpreted in $\frac{1}{\varepsilon}$ beacuse a naturally occuring function is not strongly convex (ex. hinge loss), but we can artificially add one with γ behaving like $\frac{1}{\varepsilon}$—this reduces the problem of optimizing a general convex function to optimizing a well-conditioned function.

There is a decoupling of m and $\frac{1}{\varepsilon}$ as compared to SGD.

1.3.3 Acceleration/momentum

This is a breakthrough by Nesterov, 1983. For certain optimization problems this gives the optimal number of steps. In practice it helps but not by much. Combining everything gives in theory the best known running time for first order methods,

$$O\left((m+\sqrt{\gamma m})d\ln\left(\frac{1}{\varepsilon}\right)\right).$$

This is tight (Woodworth, Srebro, 2015).

SVRG is in practice very effective for convex optimization.

Now we move from first-order to second order methods.

1.3.4 Second-order methods

Gradient descient moves in direction of steepest descent. It doesn't take into account the curvature of the function. One way to correct is to use local curvature; normalize the gradient according to the local norm.

Take a second-order approximation and optimize that. Think of GD as taking first-order Taylor approximation.

Algorithm 1.17 (Newton method)**:**

$$x_{t+1} = x_t - \eta[\nabla^2 f(x)]^{-1}\nabla f(x).$$

This is solving linear equations corresponding to the Taylor approximation of the function.

For non-convex function this can move to ∞. The solution is to solve a quadratic approximation in a local area (the trust region).

This is not used in practice because inversion takes d^3 time per iteration, but recently there have been advances.

To try to speed up the Newton direction computation:

- Spielman-Teng 2004: solve diagonally dominant systems of equations in linear time.
- Approximate by low-rank matrices and invert by Sherman-Morrison, etc. This is still d^2 time.
- Stochastic Newton (Linear-time second-order stochastic algorithm, LiSSA): Let $\tilde{n} = [\nabla^2 f(x)]^{-1}\nabla f(x)$ be the Newton direction.

  Use the structure of the ML problem. For simplicity, suppose the loss is rank-1.

  $$\text{argmin}_x \mathop{\mathbb{E}}_i[\ell(x^T a_i, b_i) + \frac{1}{2}|x|^2].$$

  This is an unbiased estimator of the Hessian,

  $$\widetilde{\nabla^2} = a_i a_i^T \cdot \ell'(x^T a_i, b_i) + l, \quad i \sim U[1, \ldots, m].$$

  Clearly $E[\widetilde{\nabla^2}] = \nabla^2 f$, but $\mathbb{E}[\widetilde{\nabla^2}^{-1}] \neq (\nabla^2 f)^{-1}$. It's not clear how to get an unbiased estimator of the Newton direction!

  We circumvent the Hessian estimation. 3 steps:

  1. Represent Hessian inverse as infinite series $\nabla^{-2} = \sum_{i=0}^{\infty}(l - \nabla^2)^i$.
  2. Sample from the infinite series (Hessian-gradient product), once:

     $$[\nabla^2 f]^{-1}\nabla f = \mathbb{E}_{i\sim\mathbb{N}}(1 - \nabla^2 f)^i \nabla f \frac{1}{\mathbb{P}(i)}$$

     (The distribution depends on the condition number, ex. take the uniform distribution up to the condition number.)

3. Estimate the Hessian power by taking examples,

$$\mathop{\mathbb{E}}_{i\in\mathbb{N},k\sim[i]}\left[\prod_{k=1}^{i}(1-\nabla^2 f_k)\nabla f\frac{1}{\mathbb{P}(i)}\right].$$

This only uses vector-vector products.

We get unbiased estimate in linear time.

Algorithm 1.18 (LiSSA)**:** Use estimator $\widetilde{\nabla^{-2}f\nabla f}$ as above, where we compute full (or large batch) gradient ∇f. Move in the direction $\widetilde{\nabla^{-2}f\nabla f}$.

Theorem 1.19 (Agarwal, Bullins, Hazan 15)**.** *The running time is*

$$O\left(dm\ln\left(\frac{1}{\varepsilon}\right)+\sqrt{\gamma}dd\ln\left(\frac{1}{\varepsilon}\right)\right).$$

This is faster than first-order methods and seems to be tight (Arjevani, Shamir 16).

Can you do variance reduction with the Hessian estimator? This is an open question people are working on.

LiSSA works better BFGS methods.

For ML we cannot tolerate anything with running time superlinear.

1.3.5 Optimization with constraints

We talk about constrained optimization. One example is matrix completion. The (i, j) entry in the table is the rating of user i for movie j; we want to complete missing entries. Assume the true matrix is low-rank. The convex relaxation is bounded trace.

The trace norm is sum of singular values. The projection step is difficult, cubic time. Optimizing a linear function over this set is easy though.

Thus we do not want to project. To solve $\min_{x\in K} f(x)$, f smooth, convex, assuming linear optimization over K is easy, use

Algorithm 1.20 (Franke-Wolfe)**:**

$$v_t = \operatorname{argmin}_{x\in K}\nabla f(x_t)^T x \tag{30}$$

$$x_{t+1} = x_t + \eta_t(v_t - x_t). \tag{31}$$

This is same spirit as GD but different. This has been used a lot in stochastic optimization.

2 Submodularity and ML: Theory and Applications, Stefanie Jegelka and Andreas Krause

Consider a ground set V; let $F : 2^V \to \mathbb{R}$ be a function on the power set. Assume $F(\phi) = 0$ and we have a black-box oracle to evaluate F.

What does this have to do with ML?

- V are variables to observe, $F(S)$ is information obtained from observing them
- V is the seed nodes in a network, $F(S)$ is the spread of information
- V is collection of images, sentences, etc. $F(S)$ is a measure of representation (ex. how well they describe/summarize the data). This includes dictionary learning, matrix approximation, object detection...

In these examples, we want $\max_S F(S)$, where F could be coverage, spread, diversity, etc.

We could also want to do the opposite, maximize coherence, smoothness, $\min_S F(S)$.

- V are data points and $F(S)$ is coherence/separation.
- V is pixels in an image and $F(S)$ is coherence/matching (for picking out an object from an image)
- V are coordinates (variables) and $F(S)$ is coherence.

Many functions can be represented as optimizing a set function. We need some additional structure that makes this doable.

Convex functions

- occur in many models, and are often the only nontrivial property that can be stated in general.
- is preserved under many operations and transformations
- have sufficient structure for theory
- allow efficient minimization.

In the discrete world, submodular set-functions share the above four properties.

We'll define submodularity, and talk about minimization, maximization, and advanced topics.

2.1 What is submodularity?

Submodularity is defined by marginal/diminishing gains.

Definition 2.1: $F : 2^V \to \mathbb{R}$ is **submodular** if for all $A \subseteq B$ and $s \notin B$,

$$F(A \cup s) - F(A) \geq F(B \cup s) - F(B).$$

For example, the more sensors I have, the less information I gain from adding another one. Here we view F as a utility function; we can also view consider F as a cost function, in which it represents economies of scale. Ex. The more you buy, the less an extra item costs.

Another way to represent the submodular property is by the union-intersection property: for all $S, T \subseteq V$,

$$F(S) + F(T) \geq F(S \cup T) + F(S \cap T).$$

Where does this submodularity property actually come up?

1. A modular function is one such that $F(S) = \sum_{e \in S} w(e)$ for some weight function w on V. Here, for $e \notin A$,
$$F(A \cup e) - F(A) = w(e).$$
F is both submodular and supermodular.

2. Coverage function: For example, for V all possible sensor locations, F is the area covered by all sensors,
$$F(S) = \left| \bigcup_{v \in S} \text{area}(S) \right|.$$
Networks coverage is a stochastic version of coverage.

3. Mutual information: There is a random variable (ex. temperature) X_i at all locations. Observations at adjacent locations are not independent. We can observe Y_i, which are the variables X_i plus noise. Select some of these observations. How much do I reduce uncertainty about latent variables Y by observing some of the X's?

   The objective is the difference between the uncertainty about Y before sensing and the uncertainty about Y after sensing, which is the mutual information.
$$F(A) = H(Y) - H(Y|X_A) = I(Y; X_A).$$

4. Entropy: Consider rv's $X_1, \ldots, X_n$, $F(S) = H(X_S)$ the joint entropy of variables indexed by S. Entropy only shrinks if you condition on more variables,
$$H(A \cup e) - H(A) = H(X_e|X_A) \leq H(X_e|X_B) = H(B \cup e) - H(B)$$
if $A \subseteq B$.

   If $X_i, i \in S$ are statistically independent, H is modular/linear on S. Submodular allows some degree of dependence.

5. Linear independence: V is a set of column vectors and $F(S) =$ rank of the matrix formed by columns in S.

6. Graph cuts $F(S) = \sum_{u \in S, v \notin S} w_{uv}$. The minimum cut problem tries to minimize this.

   Consider the cut function on the graph of 2 nodes u, v with an edge between them. Consider the union-intersection property. The only nontrivial inequality is the inequality
$$F(\{u\}) + F(\{v\}) \geq F(\{u, v\}) + F(\phi).$$
   This is true because $2w_{u,v} \geq 0$.

   For an arbitrary graph we get a sum of functions like this, so graph cut is submodular.

7. Distributions can be log-submodular or log-supermodular, sub/supermodular function that is exponentiated.

(a) A log-supermodular distribution satisfies $P(S) \propto \exp(-F(S))$. Equivalently,

$$P(S)P(T) \leq P(S \cup T)P(S \cap T).$$

This means positive associations are allowed. For example, ferromagnetic Ising model/conditional random field.

An application is image segmentation. A set of pixels are more likely to be an object if they are positively correlated. Adjacent functions are more likely to take the same label than different labels; there is a penalty when there is a difference.

What is the benefit of submodular functions here? Finding the mode of a supermodular distribution corresponds to minimizing a submodular function. We can approximate partition functions.

(b) Log-submodular distributions have

$$P(S)P(T) \geq P(S \cup T)P(S \cap T).$$

Examples are determinantal point processes and volume sampling $P(S) \propto \text{Vol}(\{v_i\}_{i \in S})$, which prefers more linearly independent vectors. A determinantal point process has $P(S) \propto \det(L_S)$, the submatrix formed by rows and columns with indices in S.

Submodular functions arise in graph, game, matroid, information theory, stochastic processes, machine learning, information theory, and electrical networks.

Does submodularity correspond to discrete convexity or concavity? A bit of both.

- They are like convex functions because you can make it into a convex function (convex relaxation) and optimize that . There is a duality theory.
- On the other hand, diminishing returns corresponds to shrinking derivatives which corresponds to a concave function.

  For example, consider $F(S) = g(|S|)$. This is submodular iff g is concave. Taking this further, I could take $g(\sum_{i \in S} w_i) = g(\sum_i w_i x_i)$ where x is the indicator for S.

Stacking submodular functions we get a deep submodular function. It looks like a deep net! The function $F(x)$ defined by

$$z_l^1 = g_l^1(\sum_i w_{l,i}^1 x_i) \tag{32}$$

$$z_l^k = g_l^k(\sum_j w_{l,j}^k z_j^{k-1}) \tag{33}$$

$$F(S) = \sum_l z_l^K. \tag{34}$$

is submodular if the g_l^k are concave and increasing and weights are nonnegative. (Unlike in a neural net we are not optimizing over the w's but over the x's.)

More generally we can define submodular functions on lattices.

Definition 2.2: A **submodular function** on a lattice satisfies

$$f(x) + f(y) \geq f(x \vee y) + f(x \wedge y).$$

On a lattice, diminishing returns is stronger than submodularity.
Many optimization results generalize to this setting.
Let's look at computational problems.

2.2 Submodular minimization

How can we find $\min_{S \subseteq V} F(S)$? We use a relaxation,

$$\min_{x \in \{0,1\}^n} F(x) \to \min_{x \in [0,1]^n} f(x).$$

How do we do this? What function f interpolates F?

One natural thing to do is to define f as the expectation of a rv. How do do this in a way such that f has nice properties? Do threshold rounding.

Definition 2.3: The **Lovász extension** of F is

$$f(x) := \mathop{\mathbb{E}}_{\theta \sim x}[F(S_\theta)].$$

where $\theta \in [0, 1]$ is sampled uniformly, and $S_\theta = \{e : x_e \geq \theta\}$.

For example, for $x = [0.5, 0.8]$,

$$\mathbb{P}(\{a, b\}) = 0.5 \tag{35}$$
$$\mathbb{P}(\{b\}) = 0.3 \tag{36}$$
$$\mathbb{P}(\phi) = 0.2, \tag{37}$$

then $f(x) = 0.5F(\{a, b\}) + 0.3F(\{b\})$.

(???) Two examples:

1. $F(S) = \max\{|S|, 1\}$. Then $f(x) = 0.8 \max_i x_i = \|x\|_\infty$. This is the l^∞ norm of the vector.
2. For $F(S) = \text{cut}(S)$, $f(x) = |x_a - x_b|$. This is the total variation function.

Theorem 2.4. *The Lovász extension is convex iff F is submodular.*

We prove that if F is submodular, than the Lovász extension is convex.

Proof sketch. If F is submodular, we can write it as a pointwise max of convex functions.

$$f(x) = \max_{y \in B_F} y^T x.$$

Here B_F is the base polytope.

Definition 2.5: The **submodular polyhedron** is

$$P_F := \left\{ y \in \mathbb{R}^n : \sum_{a \in A} y_a \leq F(A) \text{ for all } A \subseteq V \right\}.$$

The **base polytope** is

$$B_F = \left\{ y \in B_F : \sum_{a \in V} y_a = F(V) \right\}.$$

Note that there are an exponential number of inequalities defining P_F. □

Examples are

- probability simplex
- spanning tree polytope (convex hull of spanning tree indicator variables)
- permutahedron (convex hull of permutation matrices)

How can we do linear optimization over the base polytope? There are exponentially many constraints one for each subset.

But these polytopes are so nice that greedy algorithm works (Edmonds 1971).

Algorithm 2.6 (Greedy algorithm for linear optimization over base polytope)**:**

1. Sort cost vector $x_{\pi(1)} \geq x_{\pi(2)} \geq \cdots$.
2. This gives sets $S_i = \{\pi(1), \ldots, \pi(i)\}$.
3. Set $y_{\pi(i)} = F(S_i) - F(S_{i-1})$ (marginal gains).

We can do this optimization in $n \ln n$ time, the time to sort.

This implies we can compute Lovasz extension and subgradients of Lovasz extension.

A piecewise linear function is not differentiable at corner points, but there are many linear functions that lower bounds and touch at the point of discontinuity. These slopes are the subgradients. We can do gradient descent with subgradients instead. We have to be more careful with step sizes but it works.

Now we put things together. How do we go back to the discrete solution? (The solution could be fractional.)

Algorithm 2.7 (Submodular optimization)**:**

1. Relax to a convex optimization problem. Solve it using e.g. the ellipsoid algorithm.
2. The relaxation is exact. Pick elements with positive coordinates $S^* = \{e : x_e^* > 0\}$.

This solves submodular minimization in polynomial time.

There are different optimization algorithms we can apply.

- Ellipsoid method
- Subgradient method
- minimum-norm point/Fujishige-Wolfe algorithm

Another approach is combinatorial methods. Ex. Min-cut has a polytime combinatorial algorithm. Solve the dual of the minimization problem and use network flow algorithms.

The minimum-norm point/Fujishige-Wolfe algorithm uses a different relaxation,

$$\min_x f(x) + \frac{1}{2}\|x\|^2 .$$

where f is the Lovász extension. This solves a parametric series of problems

$$\min_{S \subseteq V} F(S) + \alpha|S|$$

for all α, in particular $\alpha = 1$. Thresholding at α gives the optimal solution x^* at α.

The dual problem is the minimum norm point of the base polytope

$$\min_{y \in B_F} \|y\|^2 .$$

It suffices to solve this.

Computing whether we are inside or outside the polytope, and hence projecting to it, is difficult. Instead, rely on the fact that we can do efficient linear optimization. Use the Frank-Wolfe algorithm.

At each step, find the best point along the segment joining the current point to the point that solves a linear program,

$$s^t \in \text{argmax}_{s \in B_f} \left\langle -\nabla g(y^t), s \right\rangle .$$

The min-norm point algorithm converges the fastest.

There are applications to structured sparsity, decomposition and parallel algorithms, variational inference...

For sparsity: Relax the subset selection problem using the Lovász extension. This is like going from l^0 to l^1.

2.3 Submodular maximization

This exploits concavity-like properties.

Now we want $\max F(S)$, often subject to some constraints.

Many such problems are motivated by optimal information gathering, like telling robots where to go to collect measurements, choosing a subset of variables to do experiments on,... Here $F(S)$ represents information.

Another application is data summarization. Select a subset of images that are a representative sample. This could be for exploratory data analysis. To train deep learning model on a large data set, what if we could just select a representative subset and train it on that subset.

Definition 2.8: A set function f is monotone if whenever $S \subseteq T$ then $F(S) \leq F(T)$.

A non-example is the graph-cut function.

Unconstrained maximization of monotone submodular functions is trivial: take the entire set. It makes sense to consider constraints like cardinality $|S| \leq k$.

This is NP-hard so we look for approximation algorithms. The simplest algorithm is greedy.

Algorithm 2.9 (Greedy algorithm)**:** Let $S_0 = \phi$. For $i = 0, \ldots, k-1$,

$$e^* = \text{argmax}_{e \in V \setminus S_i} F(S_i \cup \{e\}) \tag{38}$$

$$S_{i+1} = S_i \cup \{e^*\}. \tag{39}$$

This looks like a discrete analogue of gradient descent.

In practice the greedy algorithm does close to optimal. One can prove the following.

Theorem 2.10 (Nemhauser, Wolsey, Fisher 78)**.** *Let F be (nonnegative) monotone submodular, S_k be the solution of the greedy algorithm. Then*

$$F(S_k) \geq \left(1 - \frac{1}{e}\right) F(S^*).$$

In general, no polytime algorithm can do better.

Proof. Consider $F(S_l)$ as a function of l. (Side note: this is concave from submodularity.) Look at the gaps at each step of the algorithm $\Delta_i = OPT_k - F(S_i)$. The key lemma is the rate equation.

Lemma 2.11 (Rate equation)**.**

$$\max_e F(S_i \cup \{e\}) - F(S_i) \geq \frac{1}{k} \Delta_i.$$

This relates the marginal gain at the ith step to the gap.

This implies $\Delta_{i+1} \leq \left(1 - \frac{1}{k}\right) \Delta_i$. Recursively,

$$\Delta_l \leq \left(1 - \frac{1}{k}\right)^l OPT_k.$$

and so

$$F(S_l) \geq (1 - e^{-l/k}) OPT_k.$$

□

This is tight: You can match this with cover functions.

2.3.1 Greedy++ algorithms

Now we talk about "greedy++" algorithms.

1. What if I have more complex constraints?
2. Greedy algorithms take time $O(nk)$. What if n, k are large?
3. What if the function is not monotone?

2.3.2 Complex constraints

A more complex constraint is a budget,

$$\sum_{e \in S} c(e) \leq B.$$

For example, it's more expensive to place sensors at certain locations, we have to summarize in 140 characters...

We can

1. run the greedy algorithm, or
2. run a modified greedy algorithm using the benefit-cost ratio.

$$e^* = \text{argmax}_e \frac{F(S_i \cup \{e\}) - F(S_i)}{c(e)}.$$

We can construct instances where either does arbitrarily badly, but if we pick the better of the two outputs, we can always get an approximation factor of $1 - \frac{1}{\sqrt{e}}$.

There are algorithms that are more general and achieve $1 - \frac{1}{e}$.

We relax from a discrete to a continuous function.

$$\max_{S \in I} F(S) \to \max_{x \in \text{conv}(I)} f_M(x).$$

We will round to get back the discrete solution.

The first attempt is to use the Lovász extension: But it is convex, and we can't maximize it.

Instead, use the multilinear extension: sample an item e with probability x_e (each is Bernoulli random variable)

$$f_M(x) = \mathop{\mathbb{E}}_{S \sim x} [F(S)] \tag{40}$$

This is neither convex nor concave (it is convex in some directions, and concave in others).

Unlike the Lovász extension, the multilinear extension can in general only be approximated by sampling so is slower unless you have special structure.

Use the continuous greedy algorithm on f_M.

The feasible set is some polytope. Do a Frank-Wolfe-like algorithm.

Algorithm 2.12 (Continuous greedy algorithm)**:** Start at 0. Look at the gradient of the multilinear extension at that point. Solve the linear program in the direction of the gradient. Increment by a step side in that direction.

$$v_{t+1} = \text{argmax}_{x \in P} x^T g_t \tag{41}$$

$$x_{t+1} = x_t + \gamma v_{t+1}. \tag{42}$$

Take $\gamma = \frac{1}{T}$.

(N.B. it is v_{t+1}, not $v_{t+1} - x_t$: move along the segment parallel to the segment joining the origin to v_{t+1}.)

This requires the polytope to be a downward closed polytope: for $x \in P$, $[0, x_1] \times \cdots \times [0, x_n] \subseteq P$.

The continuous greedy algorithm always exploits concave directions.

This also works for the more general class of monotone continuous DR-submodular functions, which could be nonconvex functions!

We need to do rounding (omitted).

2.3.3 Faster algorithms

How can we leverage parallelism? A natural idea is to parallelize the greedy selection. This requires communication after every element has been selected.

An idea is to partition the dataset, find the solution for each, and then merge the solutions together, and run greedy again. Each could pick $\frac{k}{m}$. This doesn't work well.

A simple modification: each selects a feasible solution of size k, merge to get a solution of size km, and then run the greedy algorithm on that set. Without more assumptions, this doesn't do well, giving $\frac{1}{\min\{\sqrt{k}, m\}}$ approximation.

Instead consider the best among the $m + 1$ sets: the final greedy solution and one of the sets chosen in the first step. Then if the partition is random, in expectation we get a $\frac{1}{2}\left(1 - \frac{1}{e}\right)$ approximation.

There is also work on accelerating the sequential algorithm, filtering/streaming/multi-stage algorithms, and distributed algorithms.

2.3.4 Non-monotone maximization

Non-monotone maximization is generally inapproximable unless F is nonnegative—it is NP-hard even to know the sign of the optimal solution.

For unconstrained maximization, the double greedy algorithm gives the optimal $\frac{1}{2}$ approximation.

For constrained maximization: for cardinality constraints, use the randomized greedy algorithm.

2.4 Advanced topics

2.4.1 Semigradient methods

We want to optimize $F(S)$ with some constraints. In some cases, this is very hard for submodular F, but easy or well approximable for modular $F(S) = \sum_{a \in S} w_a$.

Examples: solving modular optimization problems over trees, matchings, cuts, paths is tractable. Replacing by submodular functions, the resulting problem is difficult. So we approximate the submodular function by a modular function.

Start with initial guess S, and repeat: approximate F by modular function F, and optimize that.

Similar to convex functions, submodular functions have subdifferentials (Fujishige). A subdifferential at X satisfies

$$m(S) \leq F(S) \text{ for all } S \subseteq V \text{ and } m(X) = F(X).$$

They also have superdifferentials (Iyer, Jegelka, Bilmes).

The semigradients (sub/superdifferentials) have a polyhedral structure.

You can recover results for SFMax (submodular function maximization) if you choose the subgradients suitably. However, this generalization allows us to handle more complex constraints. Guarantees depend on the **curvature** κ, which measures how far the function is from modular:

$$\kappa = 1 - \min_{j \in V} \frac{F(V) - F(V \backslash \{j\})}{F(\{j\})}.$$

(Compare the smallest possible gain from adding j, with the value of j alone.) Guarantees are on the order of $\frac{1}{1-\kappa}$.

There are nice applications in computer vision: segmentation, informative path planning.

Can we do inference in the resulting distributions $P(S) = \frac{1}{Z} \exp(\pm F(S))$? (For log-sub/supermodular distributions the signs are $+/-$, respectively.) The key challenge is to compute the normalizing constant $Z = \sum_S \exp(\pm F(S))$. This is #P-hard.

The gradient perspective is useful here. The idea is variational inference. Elements from the sub/superdifferentials bound F,

$$x(A) \leq F(A) \leq y(A).$$

Compute the partition function for the upper and lower bounds,

$$\sum_{A \subseteq V} \exp(x(A)) \leq \sum_{A \subseteq V} \exp(F(A)) \leq \sum_{A \subseteq V} \exp(y(A))$$

with the inequalities reversed if we replace F by $-F$, y by $-y$. The upper and lower bounds break into products. We can optimize over these upper and lower bounds by exploiting structure of the sub/superdifferentials.

An application is semantic segmentation. Solving submodular optimization finds the mode, the MAP. We can also compute the marginal entropy for each pixel.

2.4.2 Interactive optimization

We generalize subset selection problems to adaptive problems. Suppose we are a vet treating a puppie. Depending on the heart rate, we might take a ECG, where we might take a blood sample or take a fMRI, etc.

The setting: Pick an element, observe something about that element, and then depending on that element, decide which next one to pick. Is there a notion of submodularity for sequential decision tasks?

Given

- items $V = [n]$

- random variables X_i, $i \in V$ taking values in O.
- objective $f : 2^V \times O^V \to \mathbb{R}$.

We want a policy π that maps observation x_A to next item. The value is

$$F(\pi) = \sum_{x_V} P(x_V) f(\pi(x_V), x_V).$$

The policy could take exponential space to write, so we can restrict to e.g. trees of size k. Are there sufficient conditions for greedy algorithm to work? Generalize the notion of marginal gain, the conditional expected benefit of adding item s,

$$\Delta(s|x_A) = \mathbb{E}\left[f(A \cup \{s\}, x_V) - f(A, x_V)|x_A\right].$$

What's the natural greedy algorithm? The adaptive greedy policy.

Definition 2.13: The value function satisfies **adaptive monotonicity** and **adaptive submodularity** if

$$\Delta(s|x_A) \geq 0$$

and

$$\Delta(s|x_A) \geq \Delta(s|x_B) \text{ for } x_A \preceq x_B,$$

respectively.

Submodularity says it's always better to take an action earlier than later.

We also get $1 - \frac{1}{e}$ approximation.

People also try to use submodular optimization to solve non-submodular problems, like applying convex methods to nonconvex problems.

3 Reinforcement learning, Emma Brunskill

In reinforcement learning, an agent interacts with the environment, and gets back a reward and next state which is uses to determine what the next action it takes will be. It learns to maximize expected reward.

How the world (stochastic environment) works is initially unknown.

Why care about RL?

- Learning to make good sequences of decisions under uncertainty is a critical part of autonomy and intelligence.
- There are many applications: robotics, consumer modeling, education (tutoring)...

Why is RL different from ML/AI planning? Standard AI planning assumes we know how the world works. We also think about sequences of actions, but because we don't know how the world works, there are more issues we have to tackle.

We have to deal with

1. generalization: The number of states could be enormous.
2. exploration: how to gather data.
3. delayed consequences: the decisions we make now could have consequences a lot later.

RL has been separate from the active learningn community, but I think there's a lot of overlap.

3.1 Standard RL setting

Most work has focused on Markov decision processes

Definition 3.1: A **Markov decision process (MDP)** has

- set of states S
- set of actions A
- stochastic transition/dynamics model $T(s, a, s')$, the probability of reaching s' after action a in state s
- reward model $R(s, a)$ (or $R(s)$ or $R(s, a, s')$)—random variable depending on state and action
- γ discount factor: how much to value immediate reward vs. future reward

A policy for a MDP is a function $\pi : S \to A$.

The γ-discounted reward is $R = \sum_{t=0}^{\infty} \gamma^t R_t$ where R_t is the reward at the next step. The Q-function $Q : S \times A \to \mathbb{R}$ for a policy π is

$$Q^\pi(x, a) = \mathbb{E}[R | s_0 = x, a_0 = a, \pi \text{ is followed after the first step}] \tag{43}$$
$$Q^*(x, a) = \max_\pi (Q^\pi(x, a)) \tag{44}$$

The Bellman equation is

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q^*(s', a').$$

The current state is sufficient to make the next decision.

MDPs may sound like a restricted model but you can always cheat by putting the whole history into the state space.

We can use the Bellman equation as an iterative algorithm.

Algorithm 3.2 (Value iteration)**:**

$$Q_0(s, a) = 0 \tag{45}$$
$$Q_{i+1}(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q_t(s', a') \tag{46}$$

The update is a contraction so it converges in the tabular case (where we keep track of every value $Q(s,a)$). We can recover the best policy from the Q-function by taking $\text{argmax}_a Q(s,a)$.

Algorithm 3.3 (Policy iteration)**:**

$$Q_0(s,a) = 0 \tag{47}$$
$$\pi_t(s) = \text{argmax}_a Q_t(s,a) \tag{48}$$
$$Q_{t+1}(s,a) = Q^{\pi_t}(s,a). \tag{49}$$

We can break all of RL into 3 different camps: value function, policy, model. There are algorithms combine these different approaches.

In model-based RL, use the experience to estimate model T and R, ex. with ML estimate of model parameters given observed (s_t, a_t, r_t, s_{t+1}) tuples. Use estimated models to estimate Q and Q to estimate π.

This uses data efficiently but is computationally expensive.

(Is this robust? If the models are close the Q-values are close. Do the errors compound? Yes.)

An alternative, more feasible approach is Q-learning, which is model free.

Algorithm 3.4 (Q-learning)**:** Initialize $Q(s,a)$ for all (s,a) pairs. On observing (s_t, a_t, r_t, s_{t+1}),

- Calculate TD-error

$$\delta(s_t, a_t) = r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t).$$

- Update

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha\delta(s_t, a_t).$$

Here we're not picking the action.

This is computationally cheap. This only propagates experience one step, but this can be fixed by e.g. replay.

We can use all our information and recompute from scratch or just do local updates. Local updates are more computationally efficient but less sample efficient.

In policy search, directly search the policy space for $\text{argmax}_\pi V^\pi$. Parameterize policy and do stochastic gradient descent.

3.2 Exploration

We start off not knowing how the (stochastic) world works. Given some experience (data), we can estimate how the world works, or the Q-values or policies.

But the information that we've seen so far can be misleading, or we haven't seen everything. In particular, we may want to explore: try things that aren't seen to be optimal yet. Eventually, though, we want to exploit: gather high reward.

The agent is responsible for gathering data to learn how the world works.

Key ideas in exploration vs. exploitation algorithms are:

1. Act greedily mostly, sometimes randomly.
2. Be optimistic in the face of uncertainty.
3. Maintain a distribution over worlds.
   (a) Sample a world and act if the sample was the real world, or
   (b) Choose action reasoning about full distribution

Each gives a different algorithm.

3.3 Multi-arm bandit

Let's first cover a simpler model, the multi-arm bandit. There is no state.

Definition 3.5: The **multi-arm bandit** problem consists of the following.

- A is a set of arms
- R_a is unknown probability distribution of reward r if you pull arm a
- At each step t, the agent selects an arm a and gets $r_t \sim R_a$.
- The goal is to maximize cumulative reward over t pulls.

If we knew the best arm we could just choose that—but we don't.

Algorithm 3.6 (ε-greedy exploration)**:**

- Given t observations so far, estimate the mean reward for each action $\widetilde{\mu}_a$.
- W.p. $1-\varepsilon$ select $\operatorname{argmax}_a \widetilde{\mu}_a$.
- W.p. ε select random arm.

This only assumes rewards are bounded.
The second idea is optimism under uncertainty.

Algorithm 3.7 (UCB)**:** Estimate upper confidence bound on value (using concentration inequalities). Using Hoeffding,

$$\overline{\mu}_a = \widetilde{\mu}_a + \sqrt{\frac{2\ln t}{N_t(a)}}.$$

where $N_t(a)$ is the number of times a was pulled. Select $\operatorname{argmax}_a \overline{\mu}_a$.

(In the beginning pull everything once. You can't use this algorithm for huge action spaces.)

Either the arm you pull is reall the best arm and you get large reward, or you get information that allows you to revise your estimate.

We could be Bayesian: have a distribution over worlds. Maintain posterior distribution over reward distribution.

One approach to this is probability matching/Thompson sampling.

Choose arm with probability it is optimal

$$\pi(a|h_t) = \mathbb{P}(\mu_a > \mu_{a'}, \forall a' \neq a).$$

This may be hard to compute analytically, but we can estimate it by sampling.

Algorithm 3.8 (Thompson sampling)**:**

$$\pi(a|h_t) = \mathbb{P}(\mu_a > \mu_{a'}, \forall a' \neq a) = \mathop{\mathbb{E}}_{p(R|h_t)}[a - \text{argmax}_a \, \mu_a].$$

?? Sample a reward distribution given posterior, compute the mean given the sampled R, and select the action with highest mean for that sample.

This works as long as it is cheap to represent the posterior and we can update it easily. Often use the conjugate distribution: if the reward is Bernoulli, use a beta distribution for the prior/posterior.

What is the Bayes-optimal algorithm, that gets the most reward achievable? Suppose we act for H steps. We want to select the arm to maximize cumulative expected reward over H steps given the prior. This directly reasons about the value of exploration and information. This is the best we can hope to do; it optimally balances exploration and exploitation.

This view transforms a learning problem into a planning (sequential decision-making) problem.

The hidden/latent state is static; it is parameters of the reward distribution. There is a posterior/belief state that is a probability over arm distribution parameters given the history.

Thus, an optimal policy for POMDP planning gives a Bayes-optimal solution for bandit learning. However, it is generally computationally intractable to do POMDP planning for continuous states exactly.

We can use sparse sampling of Monte Carlo Tree Search to approximately plan continuous-state MDP or POMDP. But the algorithms don't have provable guarantees.

For RL, the same ideas apply, except we compute Q-values rather than the total rewards.

For optimism in the face of uncertainty, compute upper confidence bound over Q values, and given state s, select $\text{argmax}_a \, Q(s, a)$. (UCRL algorithm)

To maintain a distribution over worlds, we have to approximate.

3.4 Evaluating an RL algorithm

Which should be pick?

When I say performance, I'm talking about the rewards, or the amount of data required to find the (near) optimal policy.

Other people have been focused on computational efficiency.

We can evaluate performance by empirical, convergence, asymptotic optimality, in the probability approximately correct model, regret, Bayes-optimality, and by comparing within

a class of algorithms. Often algorithms with theoretical guarantees don't do so well empirically; we can tune constants of other a;gorithms to do better.

Convergence: do we converge to a single estimate of the Q-function? Depending on the domain, this is often nontrivial; Q-function estimate may end up oscillating.

Asymptotic optimality:

- Greedy in the limit of infinite exploration (decrease rate of exporation): Q-learning converges to Q^*. (Watkins and Dayan 1992)
- In the limit of infinite exploration model-based RL, we also converge to Q^* (Littman 1996)

This is unsatisfactory because it is an asymptotic result.

Definition 3.9: RL algorithm A is **PAC-MDP** if on all but N steps the algorithm's non-stationary policy A_t satisfies

$$V^{A_t}(s_t) \geq V^*(s_t) - \varepsilon$$

with probability $1 - \delta$ where $N = \text{poly}(|S|, |A|, \frac{1}{\varepsilon}, \frac{1}{\delta}, \frac{1}{1-\gamma})$.

γ is the discount factor; think of $\frac{1}{1-\gamma}$ as the horizon.

A popular approach for establishing algorithm is PAC RL.

RL algorithm is greedy if it maintains a Q-value estimate Q_t and selects actions by $\text{argmax}_a\, Q_t(s, a)$.

Define a known MDP M_K related to the real MDP M: Given an input Q, K a set of known (s, a) pairs,

- for all $s, a \in K$, use the true transition and reward model M parameters,
- for all $s, a \notin K$, use self-loop and set reward to input $Q(s, a)$.

This is like optimism under uncertainty: imagine that places that are unknown have good reward.

Let $A(\varepsilon, \delta)$ be any greedfy learning RL algorithm such that with probability $\geq 1 - \delta$,

- accuracy $V_t^{\pi_t}(s_t) - V_{M_{K_t}}^{\pi_t}(s_t) \leq \varepsilon$
- Optimisim $Q(s, a) \geq Q^*(s, a) - \varepsilon$
- bounded learning complexity: total number of updates to Q estimates and visites to unknown (s, a) is bounded in terms of...

A popular approach for establishing algorithm is PAC RL: Divide horizon into epsidoes, during each episode either don't update Q¡ or update Q and visit unknown (s, a) pair, use threashold to set pair as known. Pig principle ensures tha we con only visit (s, a) a finite number of times until it becomes known.

How do we ensure...

- accuracy? Use the simulation lemma: if two MDPs are close in parameters in max norm, then thei value of a policy will be close in max norm.
- optimism? Combine simulation lemma and use opt estimate of rewards for all unknown (s, a) pairs.

PAC is good because it has finite sample complexity: we bound the total number of mistakes with high probability. It's bad because experimentally, we end up exploring much longer.

Early PAC bounds are like $N = \widetilde{O}\left(\frac{|S|^2|A|}{\varepsilon^3}\right)(1-\gamma)^6$. For $|S| = 10, |A| = 10, \varepsilon = 0.1, \gamma = 0.9$, we need $N = 10^{12}$ samples.

In the episodic case (act for H steps, and then reset) we have a bound tight up to $|S|$. The upper bound is $N = \widetilde{O}\left(\frac{|S|^2|A|H^3}{\varepsilon^2}\right)$ (δ hidden in log term) and the lower bound is $N = \widetilde{O}\left(\frac{|S||A|H^3}{\varepsilon^2}\right)$. There are analogous bounds for episodic, nonstationary dynamics. Key insights include a more refined definition of knownness depending on the probability of visiting (s, a) pairs (based on Tor, Hutter 2012), and bound $V^*\mathbb{P}(s'|s, a)$ instead of each separately.

In RL, samples are not iid, so we can't apply standard concentration.

Another approach is to think about regret. Total regret for an algorithm A is

$$\Delta(M, A, s, T) = \max_{\pi} \left[\mathbb{E}\left(\sum_{t=1}^{T} r_t\right)\right] - \sum_{t=1}^{T} r_t.$$

Note a subtlety: your policy and the optimal policy π may be evaluated along different sequences of states, unlike in PAC.

UCRL2 is a optimism under uncertainty algorithm. The diameter is the maximum over all state pairs of the expected number of timesteps to go from one state to another under a policy you choose.

The UCRL2 algorithm gives an upper bound and an expected regret upper bound

$$\Delta(M, UCRL2, s, T) \leq 34D|S|\sqrt{|A|T\ln\left(\frac{T}{\delta}\right)} \tag{50}$$

$$\mathbb{E}[\Delta(M, UCRL2, s, T)] \leq O\left(\frac{D^2|S||A|\ln T}{g}\right) \tag{51}$$

where $g = Q^*(s, a^*) - \max_{a' \neq a^*} Q^*(s, a')$ is the maximum gap betweeen the average reward of the best policy and best non-optimal policy for any state.

There is also an algorithm called REGAL (Bartlett).

We can also define Bayesian regret for RL. The PSRL expected regret (Bayesian with respect to distribution of MDPs) is $\mathbb{E}[\Delta(T, PSRL)] \leq O(\tau|S|\sqrt{|A|T\ln(|S||A|T)})$. This often does better than UCRL2. Note this is still linear in the state space.

Consider contextual bandits—close to the RL setting. At each time step get a context x that doesn't depend on our actions. The reward $r_t \sim R_{ax}$ depends on the action and context.

We want to do some generalization/sharing across our state or action space. Can we make different assumptions on R_{ax} to make this tractable? For example, it's linear.

We can reduce to supervised learning. We get computationally efficient (polylog in $|\Pi|$, size of set of policies),

$$\Delta(PolicyE, T) \leq O\left(\sqrt{|A|T \ln |\Pi|}\right).$$

Assume the class is expressive enough to have the optimal policy (but often it will have to be exponential, $|A|^{|X|}$).

3.5 Current and future work

3.5.1 Off policy policy evaluation

The past data is gathered using one or more behavior policies π_b, but we want to estimate the performance on a different π_a.

In an educational game: what level do we give the students at each time to keep them engaged and learning? We used off-policy evaluation to find a policy with 30% higher engagement. (Human selection did worse than random; in complex state spaces, human intuition can fail.)

The state space is a bunch of features of the gameplay. We don't assume it's Markov.

More details: we have historical data $R_j = \sum_i r_{ij}$, assume Markov. We want to

- Estimate MDP model and compute Q of π_e, or
- do model-free learning: just compute Q of π_e.

This gives a low variance estimator of policy perforamce, but it is biased and and inconsistent.

Use Importance Sampling: reweight distribution by probability of observing the history from the evaluation policy versus from the behavior policy. This is unbiased and strongly consistent by is a high variance estimator.

$$\frac{1}{N}\sum_{j=1}^{N}\frac{\mathbb{P}(H_j|\pi_e)}{\mathbb{P}(H_j|\pi_b)}r_j$$

Can we get the best of two worlds? Dudik et al. 2011 did this for bandits, Jiang and Li 2015 did this for RL.

In MAGIC we (Thomas, Brunskill 2016) blend IS-based and model-based estimator to directly minimize the MSE.

$$MSE(x) = Bias\left(\sum_{j=1}^{\infty} x_j g^{(j)}(\pi_e|D)\right)^2 + \mathrm{Var}\left(\sum_{j=0}^{\infty} x_j g^{(j)}(\pi_e|D)\right)$$

To compute the bias we need to know the true value. We do a conservative estimate of the value. We may overtrust the model sometimes. Empirically this leads to orders of magnitudes lower error.

Other variants:

$$\Delta Q(x,a) = \sum_{t\geq 0}\gamma^t\left(\prod_{s=1}^{t} c_s\right)\underbrace{\left(r_t + \gamma \mathop{\mathbb{E}}_{\pi} Q(x_{t+1},\cdot) - Q(x_t,a_t)\right)}_{\delta_t}.$$

- In IS the trace coefficient is $c_s = \frac{\pi(a_s|x_s)}{\mu(a_s|x_s)}$, which has high variance.
- In $Q^\pi(\lambda)$, discount large trajectories $c_s = \lambda$, but this is not safe (off-policy).
- In $TB(\lambda)$, take $c_s = \lambda\pi(a_s|x_s)$, but this not efficient.
- In Retrace(λ), take $c_s = \lambda \min\left(1, \frac{\pi(a_s|x_s)}{\mu(a_s|x_s)}\right)$. Idea is to cut high variance returns by terminating long sequencea of returns, dealing with the fact that this may throw away a lot of data. This is better but still struggles for long histories.

Long horizons, where we get the reward only at the end, is still a challenge. IS estimators have high variance. Retrace will cut (ignore long horizon reward), MAGIC will reduce to model-based estimator. An open problem is to get off-policy, unbiased, low variance estimators for long-horizon delayed reward problems.

3.5.2 Safety and risk sensitivity

We want confidence bounds on expected value learned from finite data, so we can guarantee the expected return of a new policy is better.

We want to compute more than expected return: distribution of returns, conditional value at risk.

People have focused mostly on batch learning from a past set of data, but we want safety during online reinforcement learning. See Moldovan and Abbeel 2012.

Deep RL scales up to large domains. DeepMind has results in Atari competitive with humans, and beat one of the world masters in Go. They use Monte Carlo Tree Search and use RL to estimate the value function.

Why has this happened over the last few years? Historically much of RL has used tabular representations, or linear combination of features (with limited expressive power, and requires feature selection), and dynamic Bayes nets which scales poorly. Fitted Value Iteration can use any regressor to represent value function, but many common function approximators are not non-expanders: the estimated Q-function may not converge. It can oscillate and fail to converge.

In practice, a neural network to represent the Q-value of policy can work really well. We have enormous data, NNs are a powerful function approximator, and various algorithmic modifications can improve stability. They have ≈ 13 layers deep.

Deep RL is exploding. Theory is very limited but it works well in practice.

The initial deep RL algorithms used simple ε-greedy exploration. In some games, exploration needs to be more targeted. This is also not sample efficient.

How to do generalization and exploration at the same time? Cluster state-action dynamics until evidence shows that it should be separate. This speeds up learning and still converges to optimal policy (Mandel, Liu, Brunskill, Popovis 2016). We can view this as posterior sampling over state representation as well as model parameters, and also relates to infinite POMDP's.

How to represent uncertainty with deep RL to support exploration? (Osband, Blundell, Pritzel, Van Roy 2016) The state representation to represent optimal policy may not be

Markov. We might need a less compact representation to learn a policy vs. represent it. Aggregating states may lose Markovness.

Sample efficiency and representation: typically the formal sample efficiency scales as specifed state space, but we would like it to scale with the necessary state space. For factored MDP RL, we can get PAC RL to scale in the in-degree of necessary features, not max in-degree of all features. (Guo, Brunskill, in preparation.)

Open question: is this true (empirically) for deep RL?

Often only 1 frame is needed to represent the policy in Atari.

Other promising directions:

1. contextual MDPs (Krishnamurthy, Langford, Agarwal, Jiang, Schapire 2016a/b) (Best-in-class approach)

2. Temporal abstraction (actions, policies that take place over multiple timesteps). Ex. learning skill of going through a door; in-game transfer.

3. Human-in-the-loop RL (Mandoel, Liu, Brunskill, Popovic 2017). You can change the action space—humans create new actions to add into the system.

4 Interactive Learning of Classifiers and Other Structures

4.1 What is interactive learning?

Labeled data is expensive. Interactive learning hopes to reduce the amount of labeling necessary. Some examples:

1. Active learning: machine queries a few labels adaptively.

2. Explanation-based learning. In addition to lables the human gives an explanation in the form of relevant features.

   Ex. highlight words that are predictive of the label in document classification.

   ex. click on a part of the picture that explains the difference between the prediction and the actual label.

   what is the benefit these interactions give in addition to the labels? there's inherent ambiguity in the feedback

3. Interactions for unsupervised learning.

   For high dimensional datasets there are so many ways you can cluster it. There's no way an unsupervised algorithm knows which clustering you want.

   Show a random snapshot of the clustering of a few points; a human gives feedback, ex. which 2 points should be together that aren't.

   Machine has a clustering C of data X and wants feedback. Show human the restriction of C to $O(1)$ points from X.

4. Teaching: a human chooses maximally informative examples.

   In the other settings, a machine has to choose the informative examples.

Questions:

1. Efficient interaction algorithms: how much interaction is needed to learn?
2. Interaction versus computational complexity: situations where interaction circumvents computational hardness.
3. Modes of interaction: what kinds of interactions are easy and pleasant for human and provide reliable feedback?¿ Does it help to have a don't know option?
4. Communication gap between human and machine: How to bridge this?

4.2 Query learning of classifiers

Typical heuristics for active learning:

- Start with a pool of unlabeled data.
- Pick a few points at random and get their labels.
- Repeat: fix a classifier to the labels so far. Query the unlabelled point that is closest to the boundary, or most uncertain, or most likely to decrease overall uncertainty.

This seems sensible, how to analyze? The statistical learning model: there is an unknown, underlying distribution $\mathbb{P}$ on the (data, label) space, a hypothesis class H of candidate classifiers, and the target is h^* making fewest errors. Choose h_n after seeing n examples. We'd like $h_n \to h^*$ as rapidly as possible.

You get a distribution which looks less and less like the underlying distribution though: there is underlying bias going on.

Consider the data lying on a line. We're looking for a threshold classifier.

Even with infinitely many labels you could converge to the wrong thing, a classifier with greater error than the best achievable, which is not consistent.

The main problem is biased sampling: how can you ensure consistence? We have to be careful of misplaced confidence. We have to be aware of confidence, do some exploration.

Is there a generic fix to uncertainty-based heuristics?

What kind of benefits do we expect to get over random sampling?

Suppose the ground truth classifier is a threshold functions on the real line. In supervised learning, for error $\leq \varepsilon$, we need $\approx \frac{1}{\varepsilon}$ labeled points. In unsupervised learning, binary search just needs $\lg\left(\frac{1}{\varepsilon}\right)$ labels, giving an exponential improvement in label complexity. What about other hypothesis classes?

For supervised learning of a hypothesis class of VC dimension d, we need about $\frac{d}{\varepsilon}$ labeled points. Then there are $\leq \left(\frac{d}{\varepsilon}\right)^d$ ways to classify these using H (Sauer's lemma). If we ask

queries that cut this space in half each time, then just $d \ln\left(\frac{d}{\varepsilon}\right)$ are needed. This is the dream situation. [1]

But halving queries might not exist: there are examples where each query could be highly biased. We can pick whatever comes closest (greedy approach). Many variants have been investigated. Query by committee: do something similar with Bayesian prior.

Three types of active learning:

1. Mellow active learning
2. Margin-based active learning
3.

4.2.1 Mellow active learning

Cohn, Atlas, Ladner 90's.

Separable data streams in.

Algorithm 4.1 (Mellow active learner)**:** Let H_1 be the hypothesis class. Repeat for $t = 1, 2, \ldots,$

- receive unlabeled $x_t \in X$
- If there is *any* disagreement within H about x_t's lable query label y_t and set $H_{t+1} = \{h \in H_t : h(x_t) = y_t\}$. Else $H_{t+1} = H_t$.

Only ask for labels for points in region of disagreement. This is the most conservative learner.

There is no need to explicitly maintain H.

This ends up with a label for everything: either the machine asked for the label, or was 100% sure and labeled it itself. There is no sampling bias.

The label complexity can be upper-bounded in terms of the VC dimension d of H and the disagreement coefficient θ which depends on H and on the distribution $\mathbb{P}$. To achieve misclassification error ε with constant probability, suffices to have number of labels

$$O\left(\theta d \ln\left(\frac{d}{\varepsilon}\right)\right).$$

The intuition: Let $\mathbb{P}$ be the underlying distribution on the input space X.

After t points are seen, H_t consists of classifiers with error at most $\approx \Delta_t := \frac{d}{t}$. Let $DIS(H_t) \subseteq X$ be the part of the input space on which there is disagreement within H_t Any point outside $DIS(H_t)$ is not queried.

The disagreement coefficient θ satisfies $\mathbb{P}(DIS(H_t)) \leq \theta \Delta_t$.

[1] In teaching, you just need 2 examples: choose examples that are just to the left and right of the threshold.

The expected number of queries is

$$\sum_{t=1}^{T} \mathbb{P}(DIS(H_t)) \leq \theta \sum_{t=1}^{T} = \theta \sum_{t=1}^{T} \frac{d}{t} \leq \Delta_t \approx \theta d \ln T.$$

Defining the disagreement coefficient requires us to get into the geometry of the hypothesis class. The induced pseudo-metric on hypotheses is

$$d(h, h') = \mathbb{P}[h(X) \neq h'(X)].$$

The ball is $B(h, r) = \{h' \in H : d(h, h') < \varepsilon\}$.

The disagreement region of anyu set of candidate hyp $V \subseteq H$ is $DIS(V) = \{x \in X : \exists h, h' \in V \text{ s.t. } h(x)$
Need only consider $V = B(h^*, r)$, h^* target hypothesis,

$$\theta = \sup_r \frac{\mathbb{P}[DIS(B(h^*, r))]}{r}.$$

For thresholds for $\mathbb{R}$, $\theta = 2$.

For linear separators, $\theta \leq \sqrt{d}$.

4.2.2 Margin-based active learning

(Balcan-Long)

Consider algorithms based on actual heuristics, like querying points lying close to the boundary. Query points that are within an ε margin of the boundary, and reduce ε over time.

Algorithm 4.2 (Margin-based active learning)**:** For example, say all $\|x\| = 1$, $t = 1, 2, \ldots$, let w_t be the classifier based on data so far. Randomly choose points with $|x \cdot w_t| \leq m_t$, and query their labels. $\{m_t\}$ is a schedule of margins that decreases to 0.

Later on, you need more examples because most of the examples are useless. Never get to the point where you need $\frac{d}{\varepsilon}$ examples to halve the error. Cut out the regions you're sure about. Operate in the region where the error rate is constant.

4.2.3 Active annotation

The schemes we covered before are fine-tuned to the classifier. If you decide to change the classifier, you have to get a different set of examples each time.

Input:

- finite set of data points $\{x_1, \ldots, x_n\}$ each of which has an associated label y_i that is initially missing.
- parameters $0 < \delta, \varepsilon < 1$.
- access to oracle that can supply any label y_i.

Output $\widehat{y}_i$ such that w.p. $\geq 1-\delta$, at most ε fraction of these labels are incorrect

$$\sum_i \mathbb{1}(y_i \neq \widehat{y}_i) \leq \varepsilon n.$$

The goal is to minimize calls to the oracle.

For example, the input is a neighborhood graph G whose nodes are the data points x. Each node has an unknown label. The goal is to find the cut-edges in this graph that separate two labels.

Algorithm 4.3 (S^2 algorithm, Dasarthy-Nowak-Zhu)**:** Keep going until budget runs out.

- If there exist labeled nodes of opposite polarity that are connected in G, find the shortest path connecting nodes of opposite labels, and query its midpoint. Else Choose a random point and query.
- Remove any newly -revealed cut edges from the graph G.

Reduce amount of effort required by human to train a machine. Also applies to minimize experimental budget.

Often adaptive sampling leads to complicating complexity, including scalability, computation, fragility to modeling assumptions. This is why we haven't seen active learning used so much in practice.

Does interactivity always help? Not always, but it has promise in many applications.

Solution: identify specific ML models and apps that have demonstrated real-world successes. We go back to multi-armed bandits.

We focus on theoretical foundations of basic and linear bandit algorithms with human-machine interaction applications.

Why? The analysis and bounds are very tight (even in terms of constant factors), so we have theoretically-sound algorithms that are as aggressive as possible.

In the stochastic bandit problem, there are n arms, the reward distributions p_i are unknown. Each step select arm $i_t \in [n]$ and draw $x_{i_t,t} \sim p_{i_t}$ independently from past.

Examples: arms are ads, action is display an add, reward is clicks. Help scientists adapt select exper to det which genes involved in disease. Arms are genes/proteins.

Let $\mu_1 > \cdots \geq \mu_n$ be the expected rewards of each arm, $\Delta_i = \mu_1 - \mu_i$ the "gaps", $x_{it} \sim p_i$ indepedent random reward from arm i at tie t.

$$\widehat{\mu}_{i,t} = \frac{1}{t_i} \sum_{j=1}^{t_i} x_{ij}$$

Assume bounded, subgaussian rewards. Then for all i and all t, we get a confidence bound on μ_i.

The regret is

$$R_T = T \max_i \mu_i - \mathbb{E} \sum_{t=1}^{T} x_{i_t,t} = \sum_{i=1}^{n} \Delta_i \mathbb{E} T_i.$$

Sample arms to minimize $\mathbb{P}(\mu_{\widehat{i}} \neq \max_i \mu_i)$.

Eventually stop sampling suboptimal arms as long as there is some gap between the top and second best action.

Regret is $R_T = O(\sum_{i \geq 2} \frac{\ln T}{\Delta_i})$. Select top few.

For any i and all t w.p. $\geq 1 - \delta$,

$$\widehat{\mu}_{i,t} - 2\sqrt{\frac{\ln \ln \left(\frac{t}{\delta}\right)}{t}} \leq \mu_i \leq \widehat{\mu}_{i,t} + 2\sqrt{\frac{\ln \ln \left(\frac{t}{\delta}\right)}{t}}.$$

This is an adaptive algorithm. How does it compare to the best non-adaptive algorithm?

We have to sample every arm as many times as the second arm, $\sum_i T_i = \widetilde{O}(n\Delta_2^{-2})$. We could be a factor of n off.

Every week $n \approx 5000$ captions are submitted to the New Yorker. Crowdsource contest to volunteers who rate captions. The goal is to identify funniest caption.

Over time, focus on the better captions. Little gap between theory and practice. There is a 5x improvement in sample size.

There is a feature vector associated with each arm $x_i \in \mathbb{R}^d$. The reward model is $y_i = \langle x_i, \theta^* \rangle + \varepsilon_t$. Try to construct a confidence set so θ^* is always in that confidence set. Form the least-squares/ridge regression estimate of θ.

$$\widehat{\theta}_t \approx \theta^* + (x_t^T X_t + \lambda I)^{-1} X_t^T \varepsilon_t$$

Using martingale techniques get a confidence ellipsoid.

To select the arm to sample

$$x_{i_t} = \text{argmax}_{x_{1:n}} \left\langle x_i, \widehat{\theta}_t \right\rangle + \sqrt{\beta_t} \sqrt{x_t^T V x_t} ...$$

The first term is the exploit term; the second is the explore term.

Interactive image search problem: User gives feedback on images from the Zappos50K datset (shoes). From deep learning, shoes are represented by $\mathbb{R}^{1000}$ feature vectors. Linear bandits retrieve 2-3 times as many relevant items.

Now we talk about systems and apps. It's difficult to even do an experiment! There are many practical challenges. Researchers are not easily able to test, theoreticians may be unaware of real-world challenges with interesting math solutions; practitioners are not able to apply interactive learning methods.

Two groups working on this are Microsoft's Multiworld Testing Decision Service, and NEXT at UW-Madison.

5 Deep learning for robotics, Sergey Levine

In my work, I design an algorithm and take it from the foundations to a real-life application. Robots include arms, mobile robots, drones...

Deep learning allowed end-to-end learning for computer vision.

5–10 years ago, the image pipeline was the following.

- Extract features, e.g. HOG

- Extract mid-level features (DPM)
- Apply a classifier, e.g. SVM.

Deep learning still does these things, but feature selection is automated. The kind of features we learn are somewhat generic and particular suited to the classification task.

How do we use this output to make a decision about what to do next? Note the decisions will affect what it sees next, so there is a loop (RL). There are two parts to robotics, perception and action. We can close this loop, the sensorimotor loop: train the entire model together.

Richard Dawkins "When a man throws a ball high in the air and catches it again. he behaves as if he had solved a set of differential equations in predicting the trajectory of the ball... at some subconscious level, something functionally equivalent to the mathematical calculations is going on."

McLeod and Dienes: But if we couple perception and control, we can shortcut the pipeline and do better.

(Ex. for pilots: Look for a spot of dust, and see if it's stationary with respect to the other airplane; if it is you are on a collision course.)

A person opens a door better than a robot now with less sensory information.

Standard robotic control involves observations, state estimation (vision), modeling and prediction, motion planning, low-level controller (PD), motor torques... We want to do this end-to-end.

Inputs from sensors are input into learned model, and it directly outputs the actuator commands. In reality there are many challenges that makes it more difficult than the $n+1$th application of deep learning. Often the supervision is indirect (though for self-driving cars you can have more direct supervision), and actions have consequences! Data is not iid, and actions will affect what you observe.

Why should we care? It allows us to make robots that do useful things in the real world, and understand intelligence. We can build other interactive systems like interactive personal assistants, smart power systems...

5.1 Formalisms

Let o be the observation and output be u. We want to learn the policy $\pi_\theta(u_t|o_t)$.

Let x be the state. In the probabilistic graphical model, the state obeys the Markov property. The observation in general does not.

5.2 Imitation learning

Consider o is the view from a camera on the windshield and u is the steering command. The simplest thing is to get a human to drive around to collect data, train with supervised learning to get $\pi_\theta(u_t|o_t)$.

Does this work? No. The intuitive explanation: Because of error, the expected trajectory will be different from the training trajectory, and once it is off the trajectory it can make bigger mistakes. Mistakes can accumulate quadratically.

Why did that self-driving car work? There are 3 cameras, not 1. Their network only processes 1 image at a time. There is a camera facing foward, labeled with steering command, and cameras facing left/right labeled with the steering command with small offset to right/left. This ensures stability: the algorithm can now cope with deviations.

Take the optimal controller, ask the controller to compensate for small perturbations to the states, and use those to augment the dataset. This tends to work better.

Can we make this work more often? Analyze this from a probabilistic standpoint. Why does the training diverge from the expected trajectory? Consider $p_{data}(o_t)$. $\pi_\theta(u_t|o_t)$ sees data from $p_{\pi_\theta}(o_t)$. The test distribution is different from the training distribution.

Instead of being clever about $p_{\pi_\theta}(o_t)$ be clever about $p_{data}(o_t)$.

Use DAgger: dataset aggregation.

Collect training data from $p_{\pi_\theta}(o_t)$ instead of $p_{data}(o_t)$, by running $\pi_\theta(u_t|o_t)$.

1. Train $\pi_\theta(u_t|o_t)$.
2. Run $\pi_\theta(u_t|o_t)$ to get dataset $D_\pi = \{o_1, \ldots, o_M\}$.
3. Ask human to label D_π with actions u_t.
4. Aggregate $D \leftarrow D \cup D_\pi$.

This will converge.

What are the problems?

We have ask a human for the labels. Can we get a computer to supply those labels.

Summary: Imitation learning is often insufficient by itself because of distribution mismatch. It sometimes works well with hacks (ex. left/right views from car).

Distribution mismatch does not seem to be the whole story. Imitation often works without Dagger. Can we think about how stability factors in?

Imitation is more than copying an action someone took. It can be more high-level—copying someone's intentions. When you infer intention and attempt to find a behavior that satisfies it, you are combining your observation of someone else and your own experience. Then you don't have to rely on the demonstration giving you all the data. This is a crucial ingredient of true imitation learning!

5.3 Imitation without a human

We want $\min_{u_{1:T}} \sum_{t=1}^{T} c(x_t, u_t)$ such that $x_t = f(x_{t-1}, u_{t-1})$.

A classical method is to use trajectory optimization. Keep substituting to get a single optimization problem, or use sequential quadratic programming, etc. You can differentiate through via backpropagation and optimize. In practice it helps to use a 2nd order method because f is applied many times.

We can write a probabilistic version, $x_{t+1} = f(x_t, u_t)$, $x_{t+1} \sim p(x_{t+1}|x_t, u_t)$ such as $N(f(x_t, u_t), \Sigma)$. We can build a simple stochastic policy such as $p(u_t|x_t) = N(K_t x_t + k_t, \Sigma_{u_t})$. There is an $\mathbb{E}$ in the optimization problem now.

Can we use something like this in the context of an algorithm like DAgger?

Another problem with DAgger: you also need to run π_θ. If you have a bad policy, results can be bad (car driving unsafely).

PLATO (policy learning with adaptive trajectory optimization):

1. Train $\pi_\theta(u_t|o_t)$ from human data.
2. Run $\widehat{\pi}(u_t|o_t)$ to get dataset $D_\pi = \{o_1, \ldots, o_M\}$.
3. Ask human to label D_π with actions u_t.
4. Aggregate $D \leftarrow D \cup D_\pi$.

Here

$$\widehat{\pi}(u_t|x_t) = \text{argmin}_{\widehat{\pi}} \sum_{t'=t}^{T} \mathop{\mathbb{E}}_{\widehat{\pi}}[c(x_{t'}, u_{t'})] + \lambda D_{KL}(\widehat{\pi}(u_t|x_t)||\pi_\theta(u_t|o_t)).$$

Replan at each step. Replanning is model predictive control (MPC). π_θ is control from images, while $\widehat{\pi}$ is control from state. We need some way to obtain the state at training time to solve this optimal control problem with respect to states. (This is a strong assumption.) We assume $p(x_{t+1}|x_t, u_t)$ is known but $p(o_t|x_t)$ is unknown. This is realistic for e.g. autonomous cars: physics are relatively simple; images are complicated.

Safety comes from: You're not ever running π_θ until it's fully trained.

There is also the rare events problem which this says nothing about.

Example: flying quadrucopter. If the policy is not very good (has high cost, ex. leads you towards an obstacle), the cost function becomes large.

This assumes knowing the true state. In the future we want to relax that assumption.

It would be nice to not require knowledge of dynamics. We can do trajectory optimization with unknown dynamics. We need derivatives $\frac{df}{dx_t}, \frac{df}{du_t}$ (linearization of system).

Fit a plane to data and hope it's close enough. There's a trick to make this work. If dynamics are $p(x_{t+1}|x_t, u_t) = N(f(x_t, u_t), \Sigma)$, fit (after each time step) $f(x_t, u_t) \approx A_t x_t + B_t u_t$, $A_t = \frac{df}{dx_t}, B_t = \frac{df}{du_t}$. we may go into region where linearization is wrong. We can add the constraint $D_{KL}(p(\tau), \widetilde{p}(\tau)) \leq \varepsilon$. We have a heuristic that adjusts ε.

How to combine with policy learning?

Guided policy search. $\min_\theta \mathbb{E}_{\pi_\theta}[c(\tau)]$ is equivalent to $\min_{\theta, p(\tau)} \mathbb{E}_p\, c(\tau)$ such that $\pi_\theta(u_t|o(x_t)) = p(u_t|x_t)$ for all t, x_t, u_t. This constraint is equivalent to $D_t(\pi_\theta, p) = 0$ for all t.

Write the Lagrangian and use dual gradient descent.

$$\mathcal{L}(\theta, p, \lambda) = \mathop{\mathbb{E}}_{p}[c(\tau)] + \sum_{t=1}^{T} \lambda_t D_t(\pi_\theta, p).$$

Optimize $\mathcal{L}$ wrt p, wrt θ, update λ with subgradient descent, and repeat.

Combine trajectory-centric RL with supervised learning, which tries to generalize.

The data is used to fit dynamics and used to train the NN.

For imitating optimal control, is there any difference from standard imitation learning? We can change the behavior of the "teacher" programmatically.

Can the policy help optimal control rather than the other way around?

5.4 Reinforcement learning

We consider model-free algorithms; directly optimize policies.

First we cover policy gradient, then make it practical with function approximators like deep NN.

We want

$$\min_\theta \underbrace{\mathop{\mathbb{E}}_{\tau\sim p_\theta(\tau)}[c(\tau)]}_{J(\theta)},$$

where

$$p_\theta(\tau) = p_\theta(x_1, u_1, \ldots, x_T, u_T) = p(x_1)\prod_{t=1}^{T} p(x_{t+1}|x,u)\pi_\theta(u_t|x_t).$$

Compute the gradient $\nabla_\theta J(\theta) = \int [\nabla_\theta p_\theta(\tau)]c(\tau)\,d\tau$. Use the trick $\nabla_\theta p_\theta(\tau) = p_\theta(\tau)\nabla_\theta \ln p_\theta(\tau)$. Get

$$\nabla_\theta \ln p_\theta(\tau) = \sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(u_t|x_t).$$

The dynamics go away! (Williams 1992)

For example, $\pi_\theta(u_t, x_t) = N(f_{NN}(x_t), \Sigma)$. In the REINFORCE algorithm, sample $\{r^i\}$ from $\pi_\theta(u_t|x_t)$, estimate

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\left(\sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(u_t|x_t)\right)\left(\sum_{t=1}^{T} c(x_t,u_t) - b\right)\right],$$

made gradient step.

Here b is a baseline. We use the identity

$$\mathbb{E}[\nabla \ln p(y)b] = \int p(y)\nabla \ln p(y)b = \int \nabla p(y)b = b\nabla \int p(y) = 0$$

b doesn't change the mean but reduces variance. A good, not optimal choice is $b = \mathbb{E}\left[\sum_t c(x_t,u_t)\right]$.

Using the average cost baseline, a good/bad policy becomes more/less likely.

This is often not good enough with large policy classes like NN, because of high variance. (Make b depend on state and even action.) There is poor conditioning (so use higher order methods, natural gradient), it is very hard to choose step size (trust region policy optimization, TRPO).

We can rewrite

$$\nabla_\theta J(\theta) = \sum_{t=1}^{T} \mathbb{E}\left[\nabla_\theta \ln \pi_\theta(u_t|x_t)\left(\sum_{t=1}^{T} c(x_t,u_t) - b\right)\right]$$

Using knowledge of causality (action at later times don't affect previous rewards) we get

$$\nabla_\theta J(\theta) = \sum_{t=1}^{T} \mathbb{E}\left[\nabla_\theta \ln \pi_\theta(u_t|x_t)\underbrace{\mathbb{E}\left(\sum_{t'=t}^{T} c(x_t,u_t) - b\right)}_{Q^{\pi_\theta}}\right]$$

Consider the toal remaining cost of executing $\pi_\theta(u_t|x_t)$.

We can fit a function approximator to Q. We want

$$\widehat{Q}^\pi_\phi(x_t, u_t) \approx \mathbb{E}\left[\sum_{t'=t}^{T} c(x_{t'}, u_{t'})\right],$$

fit by regression: This gives the actor-critic method.

Challenges include usual problems with policy gradient, bias/variance tradeoff. FA has lower variance but higher bias. We can combine Monte Carlo and function approximation: generalized advantage estimtion.

This can solve bipedal running.

There are also online actor-critic methods. Make one decision, add to duffer, sample minibatch from buffer, estimate $\nabla_\theta J(\theta)$, update θ.

Deep deterministic policy gradient (DDPG) is effective.

You can also just directly use the Q-algorithm to make decisions. Instead of estimating the Q-function of the current policy, you can take the best action (Q-learning). π could be $\propto \exp(-\widehat{Q}_\phi(x_t, u_t))$ or ε-greedy.

Q-learning for deep RL: make one decision with $u \sim \pi_\theta(u|x)$, add to buffer D. Sample minibatch from D to fit $\widehat{Q}_\phi$. Choose $\pi(u|x)$ based on $\widehat{Q}_\phi$, e.g. $\pi(u|x) \propto \exp(-\widehat{Q}_\phi(x, u))$.

For continuous Q-learning, you can choose a representation for your Q function that makes the minimization easier. Output a Q-function quadratic in the action.

$$\widehat{Q}_\phi(x, u) = \frac{1}{2}(u - \mu(x))^T P(x)(u - \mu(x)) + V(x).$$

Tradeoffs between imitation learning and RL:

Sample complexity: FA like NN tend to be data-hungry; this is a major challenge. Training on Atari (DQN) would take 100 hours if real-time; for the bipedal task it would be 50 hours (GAE). DDPG/NAF take 4-5 hours to learn basic manipulation.

Model based methods are more efficient: time-varying linear models take 3 minutes for real world manipulation, GPS with vision takes 30-40 minutes for real-world visuomotor policies.

What do we need to get the same degree of generalization? For supervised learning, we need computation, algorithms, data. For learning sensorimotor skills, we have computations, sort-of algorithms, but where do we get data?

We can scale up via parallel operating robots training together.

Emergent behavior for grasper: When the object is soft, it's more effective to just puncture it in the middle, if it's heavy, grasp in middle, if it's flat, estimate prob of success, until find one with good probability of success.

Can we learn the cost via visual features? ("learn what success means")

Challenges include sample complexity, safety, scalability, supervision.

Exploration is an important question. Part of why in door-opening, we used Q-learning is because it does better in exploration than model-fitting approach. Random exploration comes from using Boltzmann-style policy. A model-based approach depends on how good the model is. More explicit exploration can do a long way. One family that's effective are

generative models. Use some measure of the surprise of model. Add exploration bonus to encourage visiting those more.

Adding noise can help: learn to correct mistakes.

What's a good place to do theory? In imitation learning, we want some set of assumption for which was can say something concrete. We can use that as guidance for data collection.

What is the ideal theorem you would like to see? In imitation learning, if your expret behaves according to some policy which satisfies some properties (stay in constrained region, etc.) then imitation learning has bounded loss, regret.

Imitation learning is a class of techniques. There are different properties it can have. DAgger requires more data.

Why are robots acting slowly? If you are using a control algorithm that assumes some model of system dynamics and it's not good, if you move slowly it will be less wrong.

Model-based, free seem different fundamentally. How to close the gap? One classical approach is the Dyna algorithm: fit a model of dynamics to the data, generate artificial rollouts from the model.

6 Nonparametric Bayesian methods, Tamara Broderick and Michael Jordan

This is different than many other ML topics. There is a lot of theory but much of it is open: not clear what the problems are.

Theorems rate, limiting distribution. Bernstein-von Mises, stronger than optimization where you don't know the limiting coefficients.

But that's the statistical; theory. The algorithms here are not the t. math structure that leads to certin classes of models. Think of components of model, put together with prof. Stochastic processes, links to combi, linear algebra, analysis.

Levy-Kitchine.

Linear functional plus regularization

Then apply standard algorithms

Combinatorial problems. richer than Gaussian models

not algorithm, analysis of algorithm. Combo, stoch processes, random measure.

What does nonparametric Bayes mean?

1. Bayes's Rule says

$$\mathbb{P}\left(\text{parameters—data}\right) \propto \mathbb{P}\left(\text{data—parameters}\right) \mathbb{P}\left(\text{parameters}\right)$$

   We want to understand the generative model for the data so we can apply Bayes's Theorem.

2. Nonparametric: not saying we don't have parameters, but have growing number of parameters, infinite latent parameter

   - We can read article after article on Wikipedia. No matter how many you've read there's always more topics to explore. We may be under the belief that no matter how many we've read, there are more.

- Species discovery. In the past week, a bunch of new species were discovered.
- Exercises
- As you analyze a social networks you find more friend groups.
- Loooking images, you find more unique objects.
- More ancestral groups.
- More health issues as you examine more of population
- Density estimation from more data points.

We want more parameters as we examine more data to express more nuance.

I'll focus on a particular model, clustering. The NPBayes model is the Dirichlet process. Big questions:

- Why NPBayes?
- What does a growing/infinite number of parameters really mean (in NPBayes)?
- Why is NPBayes challenging but practical?

Note this is a stationary model that allows

6.1 Clustering

In a finite Gaussian mixture model with $K = 2$ clusters we have parameters

- means $\mu_k \sim N(\mu_0, \Sigma_0)$ iid
- Proportions $\rho_1 \sim Beta(a_1, a_2)$, $\rho_2 = 1 - \rho_1$. This is conjugate with respect to the categorical distribution. We make convenient distribution. (In research we do want to ask whether these are appropriate.)
- Assignments $z_n \sim Categorical(\rho_1, \rho_2)$
- The data are generated by
$$x_n \sim N(\mu_{z_n}, \Sigma).$$

(When do you stop putting priors on things? We assume μ_0 is known.)

The beta distribution is

$$Beta(\rho_1|a_1, a_2) = \frac{\Gamma(a_1 + a_2)}{\Gamma(a_1)\Gamma(a_2)}\rho_1^{a_1-1}(1 - \rho_1)^{a_2-1}.$$

Intuition:

- For $(\rho_1, \rho_2) \sim Beta(a, a)$ for a small, we get extremes: most of the mass is on one or the other.
- For a large, we get more even distributions.

- For $Beta(a_1, a_2)$, $a_1 < a_2$, ρ_2 has more mass. (On average, $\frac{a_2}{a_1+a_2}$.)

For k clusters,

- means $\mu_k \sim N(\mu_0, \Sigma_0)$ iid
- Proportions $\rho \sim Dirichlet(a_{1:K})$. This is the generalization of the beta distribution.
- Assignments $z \sim Categorical(\rho_{1:K})$
- The data are generated by
$$x_n \sim N(\mu_{z_n}, \Sigma).$$

The Dirichlet distribution is

$$Dirichlet(\rho_{1:K}, a_{1:K}) = \frac{\Gamma(\sum kKa_k)}{\prod_{k=1}^K \Gamma(a_k)} \sum_{k=1}^K \rho_k^{a_k - 1}.$$

Dirichlet(1,1,1,1) is uniform over the probability simplex. ρ_1 is Beta(1,3).

- If all a are small, most mass is on one of the ρ's.
- If all a are large, mass is approximately evenly distributed.

So far $K \ll N$. We assumed we were seeing all the clusters. What if this is not true?

The frequentist view: Given clustering, I'm trying to find model. Bayes: Given model, what do data look like. In Bayes inf, let's make a story about how to generate data given parameters. Bayes's Theorem turns that around, probability of parameters given data. Think species sampling. Go to island, don't know about its biology, start collecting organisms (cf. Good-Turing). There are many species we haven't sampled yet. Similar with topic modeling, we don't think we've found them all. There are probably empty clusters, clusters that have no observed data points.

We make a distinction between components and clusters—number of components actually represented in the data.

Ex. consider $K = 1000$. Draw ρ from the Dirichlet distribution, represent it as a partition of $[0, 1]$ $(0, \rho_1, \rho_1 + \rho_2, \ldots)$. The number of component seen after n draws is approximate $\ln n$.

Order clusters by order of appearance. First component we see is the first cluster. Second point either joins the first cluster or starts the second cluster. The number of clusters realized is the maximum cluster number.

The number of clusters is random. It grows with the size of the data but stops at K. Sometimes this is fine, but often it is difficult to choose a finite K in advance. We run the risk of being wrong, running up against K, or having K too large and having too many parameters. Choosing a particular K can be problematic. We will choose $K = \infty$, but how can we represent this? We only represent what we need.

Let's choose $K = \infty$. Does this recover the right K? If you have a finite K, the prior goes to ∞. But in the posterior you can concentrate at that K. There are conditions where you get consistency.

We're doing something that is stationary. The model doesn't change in time. Cf. coalesence in genetics.

At first glance, $K = \infty$ seems like a bad idea. Can we even do this? What are the properties of this model? Can we actually do inference on this? What are the properties of this model? How to generate $K = \infty$ strictly positive frequencies that sum to 1?

Observe that to generate $\rho_{1:K} \sim Dirichlet(a_{1:K})$, we can generate with a sequence of beta draws,

$$\rho_1 \sim Beta(a_1, \sum_{k=1}^{K} a_k - a_1) \perp \frac{(\rho_2, \ldots, \rho_K)}{1 - \rho_1} \sim Dirichlet(a_1, \ldots, a_K).$$

This gives a recursion. This viewpoint is called "stick-breaking."

To get an infinity of variables summing to 1, we can just not stop!

$$V_1 \sim Beta(a_1, b_1) \qquad \rho_1 = V_1 \tag{52}$$

$$V_2 \sim Beta(a_2, b_2) \qquad \rho_2 = (1 - V_1)V_2 \tag{53}$$

$$\ldots \tag{54}$$

$$V_k \sim Beta(a_k, b_k) \qquad \rho_k = \left[\prod_{j=1}^{k-1}(1 - V_j)\right] V_k. \tag{55}$$

For simplicity, we take $a_k = 1, b_k = \alpha > 0$, the Dirichlet process stick-breaking, a.k.a. the Griffiths-Engen-McCloskey (GEM) distribution. (You can show w.p. 1 the ρ's will sum to 1.)

(This is a rich math tool. but why is this a good model for reality? We want a growing number of clusters. Cf. Day 1 of statistics: Gaussian. Our model gives heavy tails, not power laws. To get power laws do something more. You need a math framework to start. The first step is a baby step.)

You can a distribution on α's, etc. Where do you put your energy in richness. A lot are nuisance parameters.

GEM comes out of genetics.

GEM(α): For α smaller, we get more unequal sizes.

Note we lose the order. Integrate out via the Chinese Restaurant Process. It's a size-biased ordering, order based on order of appearance.

Sort: Poisson-Dirichlet distribution. Given the distribution, sample from them. Now you get something that tends to go down. Somewhere there is a permutation that is factored out.

The Dirichlet process mixture model:

1. $\rho = (\rho_1, \rho_2, \ldots) \sim GEM(\alpha)$.
2. $\mu_k \sim N(\mu_0, \Sigma_0)$, $k = 1, 2, \ldots$. Equivalently, take ρ_i mass on μ_i:

$$G = \sum_{k=1}^{\infty} \rho_k \delta_{\mu_k} \stackrel{d}{=} DP(\alpha, N(\mu_0, \Sigma_0)).$$

This is what's called the Dirichlet process.

3. $z_n \sim Categorical(\rho)$ iid

4. $\mu_n^* = \mu_{z_n}$, i.e., $\mu_n^* \sim G$.

How do we make draws? On-demand. Draw a random point in $[0, 1]$, then keep drawing ρ's and stop when we're past that point. For each ρ drawn, draw the corresponding mean. Draw the datapoint around the cluster to which it's assigned. Everything is finite because it's on demand.

Answers to questions:

- Why NPBayes? Learn more from more data.
- What does a growing/infinite number of parameters really mean (in NPBayes)? Components vs. clusters; latent vs. realized.
- Why is NPBayes challenging but practical? Infinite-dimensional parameter, but finitely many realized.

Typical approaches are to integrate out the infinite parameter or truncate the infinite parameter.

6.2 Chinese restaurant process

[2]

Consider the distribution

$$G = \sum_{k=1}^{\infty} \pi_k \delta_{\phi_k}.$$

Here $\pi \sim GEM(\alpha)$ is a sequence of scalars that sums to 1 with probability 1. Here $\phi_k \sim G_0$ are drawn iid from some base distribution, for example, Gaussians. It could be any distribution; for example, G_0 could be a distribution from a function space, or any separable space Θ (Hilbert space, Banach space). We say that $G \sim DP(\alpha, G_0)$.

The height π is given by a separate process independent of G_0. A draw G looks like a picket fence, and is an infinite object. G is a random measure; given a set A we can talk about $G(A)$:

$$G(A) = \sum_k \pi_k \delta_{\phi_k}(A) = \sum_{k:\phi_k \in A} \pi_k.$$

cf. A Ising model is a fixed graph of fixed weights. If we have a distribution on the weights, it's the spin-glass model.

G is a stochastic process, an indexed collection $(X_t)_t$ of random variables. For a fixed A, $G(A)$ is a random variable. Drawing G again, we get different values of $G(A)$.

$\{G(A) : A \in \mathcal{A}\}$ is the collection of random variables, $\mathcal{A}$, the set of measurable sets, is the index set.

See Jim Pitman, St. Flour lecture notes, which covers the probability and combinatorics.

[2] In Brownian motion, the times of the maximum excursion points are GEM. It's a deep mathematical theory.

Using the formalism of stochastic processes, we can talk about algebra of stochastic proceesses, hierarchies of stochastic processes, marginalization (ex. we can try to infer whether two data points are in the same cluster). In Bayes's Theorem

$$\mathbb{P}(G|X) \propto \mathbb{P}(X|G)\mathbb{P}(G).$$

G is a stochastic process.

Now given $G \sim DP(\alpha, G_0)$, draw

$$\theta_i|G \sim G \text{ iid} \tag{56}$$

in the classical Bayesian way. We can instantiate a finite number of ϕ_k's, or integrate out (Chinese restaurant process), or slice sampling, which adaptively adjusts the truncation which samples exactly from the posterior. There are other rich representations, marginalization and augmentation.

We can run Gibbs sampling, etc, and get k-means algorithm. (Cf. EM vs. annealing. Initialization for Bayesian things.) Spectral algorithms occur at the level G. At a higher level, the completely random measure level, you can do other things.

(The model is too expressive, gives all levy processes on separable spaces. We carve it down to make it less expressive and more computationally tractable.)

The Chinese restaurant process gives a distribution on partitions $\pi_{[N]}$ of $[N]$. (Think of $\pi_{[N]}$ as a set of subsets.) How do we specify the probabilities? Define the Chinese restaurant. There are an infinite number of tables. The first customer sits at the first table. The second person joins the first table with probability $\frac{1}{1+\alpha}$ or starts a new one. There is preferential attachment:

$$\mathbb{P}\left(\text{customer } n+1 \text{ joins table } c\right) = \begin{cases} \frac{|c|}{\alpha+n}, & \text{if } c \in \pi_{[n]} \\ \frac{\alpha}{\alpha+n}, & \text{otherwise.} \end{cases}$$

Let's talk about the closely related model, the Pólya urn. When you draw a ball, put the ball back with another ball of the same color. Consider a variation with a designated black ball: if you draw black, generate a new color and put it in. (This is also called the Hockey urn.) This is exactly the same as the Chinese restaurant process (each table is a color).

What is the probability after 6 people have come in we have the following partition?

$$\mathbb{P}(\{1,2,5\},\{3,4\},\{6\}) = \frac{\alpha}{\alpha}\left(\frac{1}{\alpha+1}\right)\left(\frac{\alpha}{\alpha+2}\right)\left(\frac{1}{\alpha+3}\right)\left(\frac{2}{\alpha+4}\right)\left(\frac{\alpha}{\alpha+5}\right)$$

A critical fact is that this is exchangeable, invariant under permutation. EPPF: exchangeable partition probability function. (Kingman, 60's characterized urn models with EPPF.)

$$\mathbb{P}(\pi_{[N]}) = \frac{\alpha^K}{\alpha^{\overline{N}}} \prod_{c \in \pi_{[N]}} (|C|-1)!$$

where $\alpha^{\overline{N}} = \alpha(\alpha+1)\cdots(\alpha+N-1)$ is the rising power. there are K moments where a new table was started, giving α^K. The denominators always have product $\alpha^{\overline{N}}$. For the product, given a table, look at the 2nd to $|C|$th person joining the table. The ordering is gone.

There is more general exchangability on groups, involing representation theory.

Theorem 6.1 (De Finetti)**.** *Let $(\theta_1, \theta_2, \ldots)$ be random variables. Suppose that scrambling them gives the same probability: For any permutation π, $A_k \in \Theta$,*

$$\mathbb{P}(\theta_1 \in A_1, \ldots, \theta_N \in A_N) = \mathbb{P}(\theta_{\pi(1)} \in A_1, \ldots, \theta_{\pi(N)} \in A_N,$$

then there exists a probability measure λ on the G's such that

$$\mathbb{P}(\theta_1 \in A_1, \ldots, \theta_N \in A_N) \int \prod_{i=1}^{n} G(A_i) \mathbb{P}(\lambda G).$$

The converse also holds.

This is often called the fundamental theorem of Bayesian analysis. There is G so that once you draw G, the A_i's are independent.

A sequence of 1's and 0's where someone didn't tell you the probability is exchangeable. Here $\prod_{i=1}^{n} G(A_i)$ would be the Bernoullis, $P(\lambda G)$ is the distribution on the Bernoullis. Choosing Beta gives a particular one. For iid, $\mathbb{P}(\lambda G)$ is a delta function.

If you have the representation, then you have exchangeability because it is invariant under permutation. The other direction is nontrivial. If you restrict to Euclidean spaces this is false.

Polya urn is exchangeable in the De Finetti sense.

Now given G, pick $\theta_i | G \sim G$ iid. This follows the De Finetti theorem recipe with $G = \sum_{k=1}^{\infty} \pi_k \delta_{\phi_k}$.

Take ratio of probabilities. Start with arbitrary allocation of people. Take Peter and pull him out. Should I put him at the same or not? We can assume he was the last person to arrive.

Why logarithm? The expectation of indicator variables is

$$\mathbb{E}\mathbb{1} = \sum_{n=1}^{N} \frac{\alpha}{\alpha + n} \sim \alpha \ln N.$$

How is exchangeability/de Finetti useful algorithmically? Each data point has its own parameters. Pull someone out, look at other parameters. Sample from an existing group or put it in another. This gives a Gibbs sampler!

How heavy is the tail? Not too heavy. If I want to get power laws, use the extension that generates them, the Pitman-Yor model. There is a discount parameter. Consider the Chinese restaurant process, but with discount parameter γ,

$$\mathbb{P}\left(\text{customer } n+1 \text{ joins } c | \pi_{[n+1]}\right) = \begin{cases} \frac{|C| - \gamma}{\alpha + n}, & C \in \pi_{[n]} \\ \frac{\alpha + \gamma k_n}{\alpha + n}, & \text{otherwise} \end{cases}$$

This gets more tables occupied. The G will not be the GEM but something else.

$$\text{CRM} \xrightarrow{\text{augmentation}} C \xrightarrow{\text{marginalization}} \text{CRP(Polya)}.$$

The CRM is the zoo that generates the C's.

We can prove statistical guarantees of algorithms, not CS-type theorems.

Topic models: not parametric LDA, better way with these tools, nothing known theoretically.

Math art in constructing G. Beta has 2 parameters. $Beta(\alpha_1, \alpha_2)$ can give Pitman-Yor. Still in stick-breaking family.

Broader class of EPPFs are Gibbs type pdfs.

I don't want to assume iid, I assume exchangability. This is more realistic.

6.3 Going back

Can we go the other way: if we start with the Pólya urn, can we go back to the stick-breaking process? Consider a special case of the Pólya urn where you have binarize the variables: 1 means you sit at the first table, 0 means you sit at any other table. Consider the probability everyone sits at the first table,

$$\mathbb{P}(Z_1 = 1, \ldots, Z_N = 1) = \frac{1}{\alpha+1}\frac{2}{\alpha+2}\cdots\frac{N}{\alpha+N} = \frac{\Gamma(N+1)}{(\alpha+1)^{\overline{N}}}.$$

This looks like a moment. Let's return to De Finetti. Guess the distribution that recovers the moments. Guess the beta distribution. We have

$$\int \eta^N \frac{1}{B(1,\alpha)}(1-\eta)^{\alpha-1}\,d\eta = \frac{\Gamma(N+1)}{(\alpha+1)^{\overline{N}}} = \mathbb{P}(Z_1 = 1, \ldots, Z_N = 1)$$

This is beta-Bernoulli, Bernoulli mixed by beta. (It has heavier tails.)

This is the Nth moment of the random variable.

The probability of sitting at the first table or not is given by $B(1,\alpha)$. Now let's forget about the people who sat at the first table. Take the 2nd table, look at everyone who came after the first person sitting there. Again it's $B(1,\alpha)$. This breaks off another part of the stick.

Let Θ be the underlying space. Take a partition $A_1, \ldots, A_n$. Put a random measure on it. Get a random variable associated with each region. Get

$$(G(A_1), \ldots, G(A_k)).$$

This random vector has a distribution. They add up to 1 and is nonnegative. It comes from a Dirichlet distribution

$$(G(A_1), \ldots, G(A_k)) \sim Dir(\alpha G_0(A_1), \alpha G_0(A_2), \ldots, \alpha G_0(A_k)).$$

Dirichlet process has Dirichlet marginals.

We'll prove this as a consequence of what we did. This was taken as the definition in the first paper; they used the Kolmogorov Theorem to define the Dirichlet process as the process that had these marginals, so the process is defined existentially. (This isn't actually quite right: Kolmogorov requires consistent finite-dimensional distributions plus a topological constraint, which doesn't quite work out.)

We will define a Gamma process. If we have k gamma variables, divide and normalize to get the Dirichlet distribution. This is one definition of Dirichlet. We define a gamma process which has gamma marginals.

We define a completely random measure. In general $G(A_i), G(A_j)$ are dependent. (Everything sums to 1 so there is a weak negative correlation.)'Let's define a stochastic process such that $G(A_i), G(A_j)$ are independent. If we have independent we can divide and conquer.

Definition 6.2: A **completely random measure** is such that for any partition $\bigsqcup A_i$, $G(A_i)$ are independent.

A Poisson point process is a set-valued stochastic process, such that the number of points in a set is a Poisson random variable, $Pois(\mu(A))$. Pick a total number of points, then throw at random number in the space. A nonhomogeneous Poisson process has μ nonuniform, $Pois\left(\int_A \mu(dx)\right)$.

We can convert a measure into a set-valued function by putting delta functions at each point. This gives $G(A) = \sum_{k=0}^{\infty} \delta_{\phi_k}(A)$. A Poisson random measure is a completely random measure.

Recommended: E. C.inlar is the Poisson-focused probability book.

How do we get beyond the Poisson? We have an amazing construction due to Kingman.

Take $N = \sum_{k=0}^{\infty} \delta_{\phi_k}$ where $N(A) \sim Pois(\mu(A))$. Let $\Theta \otimes \mathbb{R}^+$. Put a non-homogeneous Poisson process on the product space, $\mu = G_0 \otimes \nu$, $\mu(A \otimes E) = G_0(A)\nu(E)$.

From this Poisson random measure, we get points $\{(\phi_k, w_k)\}$. One defines location, the other defines height.

$$G = \sum_{k=1}^{\infty} w_k \delta_k.$$

In CRP you sit at one table. What if tables are not categories but features, so you want to be able to sit at multiple tables? This is a feature model. Dirichlet doesn't deliver that. The beta process does.

This is the only way to get completely random measures.

Cayman: thin book on Poisson process, weekend read. There are many characterization. The most beautiful is the most spare one.

Levy processes are Brownian motion plus jumps. If we are in 1 dimension, these would be Levy processes.

Definition 6.3: The **gamma process** $G \sim GaP(\alpha, \beta)$ is the completely random measure

$$\mu(d\phi, d\omega) = G_0(d\phi)\alpha w^{-1} e^{-\beta w}\, dw.$$

This integrates to ∞, which is what you want. If it was finite, the number of atoms would be finite; I want infinite sums; I want some part of the mass to generate an infinite number of atoms.

We show $G(A_i) \sim Gam(\alpha_i, \beta_i)$ are independently gamma, so we can normalize to get Dirichlet process. How do you show something has gamma marginals? I show that the moments match up. Use the moment generating function to do it all at once.

How do you calculate mgf of stochastic processes? That is the Levy-Khintchine Theorem. Probabilists use this all the time.

The gamma distribution is

$$Gamma(a, b) = \frac{b}{\Gamma(a)} z^{a-1} e^{-bz}.$$

It has $\mathbb{E}X = \frac{a}{b}$ We have

$$G(A) = \sum_k w_k \delta_{\phi_k}(A) \tag{57}$$
$$= \int w \mathbb{1}_A(\phi) N(d\phi, dw) \tag{58}$$
$$= \int f(w, \phi) N(d\phi, dw). \tag{59}$$

Integral a function against a random measure, get a number for each N, But N is random so this is a random variable.

For $\xi = (\phi, w)$, we want $\int f(\xi) N(d\xi)$. Consider first

$$f(\xi) = c\mathbb{1}(\xi) \tag{60}$$
$$\mathbb{E}(\cdot) = \mathbb{E} \int c\mathbb{1}_C(\xi) N(\xi) \tag{61}$$
$$= c\mathbb{E}N(c) \tag{62}$$
$$= c\mu(C). \tag{63}$$

For sums of indicators

$$f(\xi) = \sum c_j \mathbb{1}_{C_j}(\xi) \tag{64}$$
$$\mathbb{E}(\cdot) = \sum_j c_j \mathbb{1}(\xi) N(\xi) \tag{65}$$
$$= \sum_j c_j \mathbb{E}N(c_j) \tag{66}$$
$$= \sum) j c_j \mu(C_j) \tag{67}$$
$$= \int f(\xi) \mu(\xi). \tag{68}$$

We get this is true for arbitrary f by monotone convergence theorem.

We took the integral of a deterministic function against stochastic process, $\int f(\xi) N(d\xi)$. The end result is replacing N with μ.

Levy-Khintchine tells us how to get mgf for stochastic processes. Similar to the above, we get the following.

Theorem 6.4 (Levy-Khintchine)**.**

$$\mathbb{E}e^{-t\int f(\xi)N(d\xi)} = \exp\left(-\int (1 - e^{-tf(\xi)})\mu(d\xi)\right).$$

This is a Lapalce transform. Why do we get $1 - e^{-tf(\xi)}$? It's the mgf of Poisson. You're integrating the mgf of a Poisson.

Let's return to the gamma process.

$$\mathbb{E}e^{-tG(A)} = \mathbb{E}e^{-\int_\Theta \int_0^\infty (1 - e^{-tw\mathbb{1}_A(\phi)}) G_0(d\phi)\nu(dw)}$$

where $\nu(d\omega) = \alpha w^{-1} e^{-\beta w} dw$. This gives

$$= \exp(-\alpha G_0(A) \int_0^\infty (1 - e^{-t\omega}) w^{-1} e^{-\beta a}\, dw) = \left(\frac{\beta}{\beta + t}\right)^{\alpha G_0(A)}.$$

which is the gamma mgf.

We can design other stochastic processes by replacing $\nu(d\omega)$ with something else like $w^{-1}(1-w)^{a-1}\, dw$.

Didn't get to cover: Hierarchical Dirichlet processes, etc. G_0 is from a Dirichlet process. Multiple clustering: each document cluster words, but we want to unify documents among corpus. Prove things like: rate is $\ln \ln n$.

Book: Aad van der Vaart, Holland: Theory of Bayesian nonparametrics. `http://www.math.leidenuniv.nl/~avdvaart/BNP/BNP.pdf`

Tamara: conjugacy theory for exponential family-based random measures.

6.4 Applications

`http://www.tamarabroderick.com`

What do we do with these models besides clustering? One application: we have different datasets, we share power among them.

In probabilistic models for graphs, we capture rich relationships, coherent uncertainties, and prior information.

Example models are stochastic block model, mixed membership stochastic block model, infinite relational models, and many more (Lloyd 2012).

How do these models capture streaming data?

Are we capturing the right growth properties in these graphs?

One simple property is **projectivity**: adding more data doesn't change distribution of earlier data. Then these models and many more are misspecified because they are too dense. (The number of edges as a function of the number of nodes it quadratic, too many. There are many real-life models where we don't see this.)

We give a new framework for sparse graphs. This is a Bayesian nonparametric framework.

We will come back to clustering and get some characterization theorems.

There is also concurrent and independent work by Crane and Dempsey.

Consider a sequence of graphs. At each step we add some nodes and edges, but never subtract anything. Imagine this sequence has countably infinite steps.

A particular interesting case is the number of nodes going to ∞. A graph sequence is **dense** if

$$\#E(G_n) \geq c \#V(G_n)^2.$$

It is **sparse** if

$$\#E(G_n) \in o(\#V(G_n)^2).$$

All the graph models we saw before can't satisfy this. (why not?)

Add one node at each step; label by the step where it comes in. Whenever a node comes in, it connects to all edges it will ever connect to. Assume we have some distribution over this graph sequence. A key property is exchangeability. We don't have data points.

The probability doesn't change if we label the nodes differently. (Hoover 1979) We call this node exchangeability. How might we generate an exchangeable distribution for graphs.

Consider a function $W : [0,1]^2 \to [0,1]$ that is symmetric about $x = y$ (a graphon). Generate a graph as follows. For each node choose a number x_i in $[0,1]$. For (i,j) put an edge with probability $W(x_i, x_j)$. This generates a node-exchangeable graph.

Theorem 6.5 (Aldous, Hoover)**.** *Every node-exchangeable graph has a graphon representation.*

We have

$$\mathbb{E}[\#E(G_n)] = \mathbb{E}\left[\binom{n}{2}\frac{1}{2}\int_0^1\int_0^1 W(x,y)\,dx\,dy\right] \sim cn^2.$$

Every node-exchangeable graph sequence is dense (or empty) almost surely.

Intuition: to a given node, all other nodes look the same. See survey by Orbanz, Roy 2015.

Here is an alternative idea. Label the edges at the step it's added. Bring in a node when it's part of an edge that's been instantiated. This is like emails (sender and receiver). Exchangeability: The probability doesn't change if we change the order of the edges. (Ex. the order of the emails doesn't matter.)

Theorem 6.6 (CCB)**.** *A wide class of edge-exchangeable graph models yields sparse graph sequences.*

Our goal:

1. characterization theorem for edge-exchangeable graphs.
2. sparsity theorem for edge-exchangeable graphs.

Along the way we use examples from clustering.

We want clusters to be meaningful (cat, dog, mouse pictures, etc.). We could represent this in a matrix-like form. There is no ordering on columns. There is something missing in this representation: what if some pictures have multiple animals? We want feature allocation: each datapoint can belong to multiple groups (feature). We have the same exchangeability assumption.

Graphs are a specific subclass: each data point belongs to 2 columns. Columns are labels. The groups are the vertices. Exchangeability of rows is not edge exchangeability. We allow multigraphs.

This looks like clustering.

An exchangeable probability function for a clustering is a symmetric function of the size of the clusters

$$p(S_{N,1}, \ldots, S_{N,K}).$$

This is the exchangeable partition probability function. Pitman showed every exchangeable clustering has an EPPF.

For feature allocation, making $S_{N,K}$ the size of (number of datapoints in) the Kth feature, I can define the exchangeable feature probability function (EFPF)

$$p(N; S_{N,1}, \ldots, S_{N,K}).$$

Many but not all exchangeable feature allocations have an exchangeable feature probability function.

Suppose each row is an edge, each vertex is a column. $S_{N,K}$ is the degree of the Kth vertex. This is an exchangeable vertex probability function.

Does every edge-exchangeable graph have an EVPF? No, here is a counterexample: Suppose we have 4 vertices. Every time, choose iid an edge 12, 23, 34, or 41. If I had an EVPP, then 12, 34 and 23, 41 would have the same probability. But the probabilities are p_1p_2 and p_3p_4. This is also a proof that we don't always have EFPR's.

Here is a way to generate exchangeable clusters. Randomly paritition $[0, 1]$. For each data point, it belongs to the cluster corresponding to the interval it falls in. When fall in same interval, fall in the same cluster.

Theorem 6.7 (Kingman)**.** *A clustering is exchangeable iff it has a "Kingman paintbox" representation.*

This paintbox representation is exactly the De Finetti mixing measure. Dirichlet process is one example.

We can have more general subsets that overlap. Now a variable can belong to multiple subsets so can have multiple features.

Theorem 6.8. *A feature allocation is exchangeable iff it has a feature paintbox representation.*

(You can also allow dust: you could never see the cluster again.)

We make the constraint that every slice has exactly 2 nodes. How do you generate a usable form of this as a stochastic process? Use independence assumptions.

Theorem 6.9. *A graph sequence is edge-exchange iff it has a graph paintbox.*

This extends to hypergraphs. You can add self-edges and skips.

If you don't want repeated edges this is not the model for you. You will get an infinite number of repeated edges! You can also include dust: people communicate once and never again.

Is there another natural model that prohibits repeated edges?

Picture: (Edge-exchangeable $\iff$ graph paintbox) $\supseteq$ EVertexPF

How to prove sparsity?

We need the number of nodes to go to infinity. We need a countable infinity of latent nodes.

The graph frequency model/vertex popularity model.

- Draw a rate w_i for each vertex i.
- Draw $\{i, j\}$ with probability $\propto w_iw_j$.

Suppose $w_i \sim$PoissonPointProcess(ν) where ν is regularly varying.

$$\int_x^1 \nu(dw) \sim x^{-\alpha} l(x^{-1}), x \to 0, \quad \forall c > 0 \lim_{x\to\infty} \frac{l(cx)}{l(x)} = 1.$$

Then $|V_n| = \Theta(n^\alpha l(n))$ and $|E_n| = \Theta(n)$ a.s.

It remains to fill out the Venn diagram.

Graph frequency models are a subset of edge-exchangeable models. (Ex. the 4-vertex model is in the difference.) In fact, EVertex PF are the same as graph frequency models!

What's next: characterize all sparse, edge-exchangeable graphs. Characterize different types of power laws (edges, triangles, degree distributions, etc.).

Can this incorporate community structure? Vertex popularity. You can also do within graph paintbox.

Only certain subsets are exchangable (within communities)? You can get away from sparse assumption.

Do they model real world graphs? Can we simulate and get desirable behaviors? Can we put models together? There are many other properties beyond sparsity you can be interested in.

7 Deep learning, Ruslan Salakhutdinov

Deep learning algorithms are very data-hungry. They are hierarchical models.

The impact of deep learning has been strong and surprising: speech recognition, computer vision, recommender systems, language understanding, drug discovery and medical image analysis.

On the Reuters dataset, it discovers topics in an unsupervised way. For caption generation for images, the nearest neighbor sentence isn't very useful, but a neural language model does reasonably well.

Supervised learning is the most successful. I'll cover recent optimization and regularization techniques. Then I'll talk about unsupervised (deep generative) models which doens't work as well. Finally I'll give open questions.

7.1 Supervised learning

Traditional approaches extract features from the data and feed them into the learning algorithm. Vision has many techniques for finding the right features. Ten years ago, the main problem was finding the right features. Audio is similar. In representation learning, we want to automatically learn these features/representations.

7.1.1 Definition of neural networks

The neuron pre-activation and output activation are

$$a(x) = b + \sum_i w_i x_i = b + w^T x \tag{69}$$

$$h(x) = g(a(x)) \tag{70}$$

where g is some activation function. Popular activation functions are sigmoids $\frac{1}{1+e^{-a}}$, $\tanh(a)$.

One of the most successful ones (Hinton) is $ReLU(a) = \max(0, a)$, which tends to produce sparse activity. The gradient is a linear function. The network is easier to optimize. The only reason the system becomes nonlinear is that you shut down some units. (It is piecewise linear.) Computing a linear function is faster than computing sigmoid or tanh.

In multilayer neural net,

$$a^{(k)} = b^{(k)} + W^{(k)} h^{(k-1)}(x^{(k-1)}) \tag{71}$$

$$x^{(k)} = g(a^{(k)}). \tag{72}$$

Combining layers you can construct complex decision boundaries.

Theorem 7.1 (Hornik's universal approximation theorem)**.** *A single hidden layer neural network with linear output can approximate any continuous function arbitrary well, given enough hidden units.*

Empirical risk minimiation is

$$\text{argmin}_\theta \frac{1}{T} \sum_t l(f(x^{(t)}; \theta), y^{(t)}) + \lambda \Omega(\theta).$$

For NN the regularizer plays a critical role. Recent regularization techniques is what made these models so successful. People used regularization like L^2. They work but are not sufficient. Techniques like dropout are important.

Use stochastic gradient descent (SGD). Initialie $\theta = \{W^{(1)}, \ldots, b^{(L+1)}\}$,

$$\Delta = -\nabla_\theta l(f(x^{(t)}; \theta), y^{(t)}) - \lambda \nabla_\theta \Omega(\theta) \tag{73}$$

$$\theta \leftarrow \theta + \alpha \Delta. \tag{74}$$

The backpropagation (chain rule) algorithm computes these gradients. There are good software packages. It's hard to beat SGD.

With these models, we can't find global optima. In practice, local optima is not a big problem even though there are tons of local optima.This is a good theory question.

Model selection:

- Train model on training set
- For model selection use validation set
- Estimate generalization performance using test set.

We need hyperparameter search: hidden layer size, learning rate, number of iterations/epochs, etc. They play a critical role.

Stop training when validation set error increases.

Generating data using 1 hidden layer, you want to train using more units (overparameterize). Even for a latent space of 1 dimension, optimization is hard so you want to overparameterize.

Improvements:

- Make updates based on minibatch of examples.
- Momentum: use exponential average of previous gradients.

$$\overline{\nabla}_\theta^{(t)} = \nabla_\theta l(f(x^{(t)}), y^{(t)}) + \beta \overline{\nabla}_\theta^{(t-1)}.$$

  (Poor man's approximation to diagonals of Hessian.)

Every layer is learning a distributed representation. Units in a layer are not mutually exclusive. Units do not necessarily correspond to a partitioning.

Clustering or nearest neighbors partitions the space. There are parameters for each region and the number of regions grows linearly with number of parameters.

For distributed representations (RBM, factor model, PCS, sparse coding, deep models), the number of realizable sign assignments grows faster: it grows polynomially (n.b. not exponential) in the number of hyperplanes.

NN gains inspiration from visual cortex.

Let's get to more controversial things. Why is training hard? Hypotheses:

- Hard optimization problem (underfitting): vanishing gradient problem, saturated units block gradient propagation.
- Overfitting: we are exploring a space of complex functions, and deep nets have lots of parameters. We could be in a high variance/low bias situation.

  Fitting using a linear function has low variance and high bias. Fitting using a complicated function has high variance, because we might fit very different functions to slightly different datasets, but low bias, because we can get close. The best function trades off between these two.

For large-scale practical problems, you have to use both better optimization and better regularization. The best performing systems are where you build a very large model and regularize.

There is good theoretical foundation for unsupervised pre-training (Jeff Hinton, variational lower bound). You are actually optimizing some objective function.

Initialize hidden layers using unsupervised learning: force network to represent latent structure of input distribution. Why is one image a character and another not? This is harder than supervised learning (classification) so we expect less overfitting.

There is a class of networks called autoencoders. They try to reproduce the input at the output layer. (For linear maps, this recovers PCA.)

How does pre-training work? Use a greedy layer-wise procedure. Train one layer at a time with unsupervised criterion. Fix parameters of previous hidden layers. Previous layers can be used as feature extraction. Make sure the parameters can reconstruct the inputs. Then add output layer and train whole network in supervised fashion (fine tuning).

You can enforce by bottleneck, sparsity, etc.

Why train one layer at a time? It's an easier optimization problem.

7.1.2 Regularization

Another regularization is stochastic dropout. Cripple neural network by removing hidden units stochastically.

1. Each hidden unit is set to 0 with probability 0.5 (in forward propagation). (0.5 typically works well.)

$$h^k(x) = g(a^k(x)) \odot m^k \tag{75}$$

   where each entry of m^k is Bernoulli(0.5).

2. Hidden units cannot co-adapt to other units.

3. Hidden units must be more generally useful.

The units tend to be more uncorrelated.

At test time, replace masks by their expectations, ex. the constant 0.5 if you keep with probability 0.5. (Or: stochastically sample forward passes.)

This beats regular propagation on many datasets.

Ensemble: this can be viewed as a geometric average of an exponential number of networks.

Anytime you inject noise, you tend to do better. Sometimes people inject Gaussian noise. That helps as well.

There is a beautiful technique called batch normalization that has become standard. Normalizing the inputs speed up training. Batch normalization tries to do this at the level of the hidden layers (Ioffe, Szegedy 2014). Each unit's pre-activation is normalized (subtract mean and divide by standard deviation). During training, mean and standard deviation are computed for each minibatch. Backprop takes into account the normalization.

Algorithm 7.2 (Batch normalization)**:** For B the minibatch,

$$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \tag{76}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \tag{77}$$

$$\widehat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}} \tag{78}$$

$$y_i = \gamma\widehat{x}_i + \beta = BN_{\gamma,\beta}(x_i) \tag{79}$$

(The last step is a scale and shift: for every neuron, learn parameters γ, β; linear transformation with parameters γ, β adapt to non-linear activation function. (The extra parameters are not a big deal because most of the parameters come from the edges, not nodes.)

This seems to be approximating the diagonal of Hessian but not quite. From a theoretical perspective this is just a hack, but it works remarkably well. You're computing these based on the batch! Batches are typically size 256. Depends on your GPU.

SGD with momentum, batch-normalization and dropout usually works well; it's hard to beat this.

Pick learning rate by running on subset of data: start large, divide by 2 until loss does not diverge; decay learning rate by ≈ 100 by end of training.

Visualize: Visualize features; see that features are uncorrelated and have high variance. Training is bad when many hidden units ignore the input and/or exhibit strong correlations.

Why normalize the pre-activation?

7.1.3 Computer vision

Given an image, classify it. Use convolution neural networks. Do convolution, max pooling $(2 \times 2, 3 \times 3)$ to get translational invariances. One layer has multiple maps.

This is used in OCR, pedestrian detection, object detection...

The ImageNet dataset has 1.2 million images and 1000 classes. Important breakthrough was AlexNet, deep convolutional nets (8 layers) for vision (Krizhevsky, Sutskever, Hinton). This halved the error of existing systems.

Manual tuning of features is now replaced with tuning of architecture. Use grid search (need lots of GPUs). Smarter strategies include random search, Bayesian optimization.

AlexNet has 8 layers total. If we remove top fully connected layer 7, drop ≈ 16 million parameters, lose 1.1% performance..

GoogLeNet has 24 layers, "inception module". Multiple filter scales at each layer. Use dimensionality reduction to keep computation down. Width ranges from 256 to 1024. Can remove FC layers on top completely. Number of parameters reduced to 5 million. 6.7% top-5 validation error on Imagenet.

Residual networks have many more layers. The trick is skip-connections, $F(x) + x$.

ImageNet has 50 kinds of mushrooms, dog breeds. When you train on these, and you want to train on people, the features are still useful. There is interesting transferability!

7.2 Unsupervised learning: learning deep generative models

There are

- non-probabilistic models: sparse coding, autoencoders, etc.
- probabilistic (generative) models with explicit density $p(x)$
  - tractable models: fully observed belief nets, NADE, PixelRNN
  - non-tractable models: Boltzmann models, variational autoencoders, Helmholtz machines
- Generative adversarial networks, moment matching netwrks implicitly model density. These are game-theoretic.

7.3 Basic building blocks

Sparse coding objective: given set of input data vectors $\{x_i\}$ learn a dictionary of bases $\{\phi_K\}$ such that

$$x_n = \sum_i a_{ni}\phi_i.$$

with a sparse. We want each image patch to be represented by a sparse linear combination of basis vectors.

Training is

$$\min_{a,\phi} \sum_{n=1}^{N} \left\| x_n - \sum_{k=1}^{K} a_{nk}\phi_k \right\|_2^2 + \lambda \sum_{n=1}^{N}\sum_{k=1}^{K} |a_{nk}|.$$

Do alternating minimization.

Going from sparse features to x' is an explicit linear decoding; encoding is nonlinear and implcit.

Autoencoder: Given input image, encode to feature representation, decode to get back input image. (Note: From a theorist's viewpoint, the names are flipped from coding theory.) The details of what goes inside the encoder and decoder matter: we need constraints to not just learn the identity.

If both hidden and output layers are linear, and you use squared error, you recover PCA. You can think of autoencoders as a nonlinear extension.

A denoising autoencoder: Idea is that representation should be robust to introduction of noise. Randomly assign subset of inputs to 0 with probability ν. This is similar to dropout on the input layer. Alternatively add Gaussian additive noise. This is like trying to get back to the data manifold.

Predictive sparse decomposition:

$$\min_{D,W,z} \underbrace{\|Dz - x\|_2^2 + \lambda|z|_1}_{\text{decoder}} + \underbrace{\|\sigma(Wx) - z\|_2^2}_{\text{encoder}}.$$

For stacked autoencoders, do greedy layer-wise learning. Remove the decoding and use feed-forward part for supervised training.

We can compress images 28×28 to 30-dimensional with deep autoencoder. Olivetti face patches: It removes glasses, mustaches.

Information retrieval: a nonlinear model preserves more clustering structure.

For binary codes, we can encode input into binary space. Searching in binary space is a much easier problem. Map documents into 20-D binary space. You can also search large image database using binary codes.

7.3.1 Deep generative models

Given 25000 characters from 50 alphabets around the world, generate characters. You can also given half of image, sample what the other half looks like. Markov chain Monte Carlo generates different characters.

We can model conditional probabilities

$$p_{\text{model}}(x) = p_m(x_1) \prod_{i=2}^{n} p_m(x_i | x_1, \ldots, x_{i-1})$$

and get good images. (Pixel CNN, RNN.) What representation are these nets using?

RBM is a Markov random field with stochastic binary visible variables $v \in \{0,1\}^D$, hidden variables $h \in \{0,1\}^F$, bipartite conndtions

$$P_\theta(v,h) = \frac{1}{Z(\theta)} \exp\left(\sum_{i=1}^{D} \sum_{j=1}^{F} W_{ij} v_i h_j + \sum_{i=1}^{D} v_i b_i + \sum_{j=1}^{F} a_j h_j \right).$$

Ising model with bipartite architecture.

We get sparse representations. Inference in this model is basically exact.

We are matching sufficient statistics. $\mathbb{E}_{P_{data}}[v_i h_j]$ is easy to compute, $\mathbb{E}_{P_\theta}[v_i h_j]$ is difficult to compute because there are exponentially many configuration. This comes from $\frac{\partial}{\partial W_{ij}} \ln Z(\theta)$.

People use MCMC.

Sample $\mathbb{P}(h|v), \mathbb{P}(v|h)$ alternately. Instead of running to ∞, run one time. This is called contrastic divergence: Then udpate model parameters

$$\Delta W_{ij} = \mathop{\mathbb{E}}_{P_{data}}[v_i h_j] - \mathop{\mathbb{E}}_{P_1}[v_i h_j].$$

Shapes distribution locally around data point, making it more probable.

You can model real-valued data $v \in \mathbb{R}^D$,

RBM for word count: You can apply this to modeling count data: replicated softmax model. On documents, you find some version of topics.

We can use this for collaborative filtering; it finds "genre". We can infer states of hidden variables exactly, unlike in Bayesian models where inference can be expensive.

Another intution: these models are also called product models. Marginalizing over hidden variables gives a prduct of experts

$$P_\theta(v) = \sum_h P_\theta(v,h) = \frac{1}{Z(\theta)} \prod_i \exp(b_i v_i) \prod_j \left(1 + \exp\left(a_j + \sum_i W_{ij} v_i\right)\right)$$

Topics government, corruption, and mafia gives high probability to "Silvio Berlusconi". This is very different from mixture-based models, where you choose a component and generate a particular word from that component.

Deep belief networks: these are older models, 2005. They gave rise to many deep learning models. The model is a hybrid model, undirected graphical model at top with directed model going down (sigmoid belief network). This is because of a specific training procedure.

$$\mathbb{P}(v, h^1, h^2, h^3) = \mathbb{P}(v|h^1)\mathbb{P}(h^1|h^2)\mathbb{P}(h^2, h^3).$$

This model is a map; proper inference is very hard. There is an approximate inference network, some NN model that infers the parameters/hidden states, approximation to posterior.

DBN layer-wise training:

- learn RBM with input layer v and hidden layer h¿
- Treat inferred values $Q(h^1|v) = P(h^1|v)$ as the data for training 2nd layer RBM.

1-layer RBM with W_1 is equivalent to 2-layer DBN with $W_2 = W_1^T$ undirected on top. Greedy training improves the variational lower bound. For any approximating distribution $Q(h^1|v)$, we can lower bound with variational bou

$$\log P_\theta(v) \geq \sum_{h^1} Q(h^1|v) \left[\log P(h^1) + \log P(v|h^1)\right] + H(Q(h^1|v)).$$

Training 2nd layer RBM improves the term in brackets.

Convolution extensions of DBN: the first layer is learning edges; 2nd layer pick up parts; 3rd layers pick up faces.

Message: original paper on DBN: justification for variational inference is nice. Adding additional layers is solving an optimization problem.

Comparison: DBN has an interesting structure: undirected followed by sigmoid belief networks. DBM (Deep Boltzmann machine) is an undirected graphical model.

In DBM, to compute the value of a given variable, we need to take into account variables both above and below; it's not just feedforwards, so we need some approximate inference. (For DBN, inference is feedforward.)

$$\frac{1}{Z(\theta)} \sum_{h^1,h^2,h^3} \exp\left[v^T W^1 h^1 + h^{1T} W^2 h^2 + h^{2T} W^3 h^3\right]$$

By making your model more expressive, inference becomes more difficult. $\mathbb{P}_\theta(h^1|v)$ is intractable; it doesn't factor.

The first term tries to put probability around observed data, $\mathbb{E}_{P_{data}}[vh^{1T}]$. The second term $\mathbb{E}_{P_\theta}[vh^{1T}]$ makes sure there is little mass at other locations.

$$\frac{\partial}{\partial \ln P_\theta(v)} W^1 = \mathbb{E}_{P_{data}}[vh^{1T}] - \mathbb{E}_{P_\theta}[vh^{1T}].$$

Use variational inference for first term, stochastic (MCMC) approximation to estimate the second term.

Many previous approaches were not successful for learning general Boltzmann machines with hidden variables.

First we need to infer the distribution over hidden variables. I can simulate from the model and see what's being produced, $P_{model}(h, v)$ gives data-independent $EE_{P_{model}}[vh^T]$. From approximate conditional $P_{data}(h|v)$, try to fit data-dependent $\mathbb{E}_{P_{data}}[vh^T]$ to this.

In general, you expect the posterior to be unimodal. This suggests that for posterior inference use mean-field theory (variational inference). For model simulation, use MCMC (stochastic approximation).

Define a Markov chain by updating θ_t, x_t sequentially where $x = \{v, h^1, h^2\}$.

- Generate $x_t \sim T_{\theta_t}(x_t \leftarrow x_{t-1})$ by simulating from Markov chain leaving P_{θ_t} invariant.
- Update θ_t by replacing intractable $\mathbb{E}_{P_{\theta_t}}[vh^T]$ with point estimate $[v_t h_t^T]$.

(Younes, Probability Theory 1989, Robbins Munro 1957.) In practice simulate Markov chains in parallel.

The update rule decomposes into the true gradient and the perturbation term

$$\theta_{t+1} = \theta_t + \alpha_t \left(\mathop{\mathbb{E}}_{P_{data}} [vh^T] - \mathop{\mathbb{E}}_{P_{\theta_t}} [vh^T] \right) + \alpha_t \left(\mathop{\mathbb{E}}_{P_{\theta_t}} - \frac{1}{M} \sum_{m=1}^{T} v_t^{(m)} h_t^{(m)T} \right).$$

But we don't get an unbiased estimator, because the Markov chain could mix very slowly. As learning rate goes to 0 at a rate slower than mixing of the Markov Chain, we're ok. There are no known bounds on mixing.

For high-dimensional data the probability landscape is highly multimodal. But the transition operator can be any valid transitoin operator: we can construct other Markov chains that jump between modes. There are connections to stochastic approximation and adaptive MCMC.

Suppose we try to approximate $P_\theta(h|v)$ with simpler tractable distribution $Q_\mu(h|v)$. Write

$$\ln P_\theta(v) = \ln \sum_h P_\theta(h, v) \tag{80}$$

$$\geq \sum_h Q_\mu(h|v) \ln \frac{P_\theta(h, v)}{Q_\mu(h|v)} \tag{81}$$

$$= \sum_h Q_\mu(h|v) \underbrace{\ln P_\theta^*(h, v)}_{v^T W^1 h^1 + h^{1T} W^2 h_2 + h^{2T} W^3 h_3} - \ln Z(\theta) + \sum_h Q_\mu(h|v) \ln \frac{1}{Q_\mu(h|v)}. \tag{82}$$

The partition function is constant here. This is a variational lower bound.

$$= \ln P_\theta(v) - KL(Q_\mu(h|v)||P_\theta(h|v)).$$

Mean field: choose a fully factorized distribution $Q_\mu(h|v) = \prod_{j=1}^F q(h_j|v)$ with $q(h_j = 1|v) = \mu_j$. Variational inference: maximize the lower bound wrt variational parameters μ.

If posterior multimodal, select 1 model.

Mean-field gives fast inference, and learning can scale to millions of examples.

Compare real with generated characters. Generated characters are not as precise. Diversity in real data is higher. (Exploration of exponential space: Markov chain can't explore all of space.)

One of the biggest problems is that there's little quantitative evaluation.

These generative models can be used for classification. You can do pattern completion.

Is there a proper way to evaluate generative models? Yes.

We can model multimodel data. The space of text is sparse, discrete, while images are real-valued, dense. We have noisy and missing data. Boltzmann machine models give good text (samples from conditional distribution).

We make a multimodel DBM: bring image and text to some level and model together. Information from words can influence low-level features in images and vice versa.

Failure: crane becomes Obama. The image dataset has lots of Obama pictures, but not animals. Prior takes over because the posterior is too weak.

Instantiate hidden states randomly. There are multiple alternative explanations. This can be useful. Instead of just tagging, get whole distribution of alternative explanations. (Alternative facts!)

There are 25k labeled examples and 1 million unlabeled which helps.

Another class of models is Helmholtz machines/variational autoencoders (VAE). They made a big comeback in the past few years and are popular now because there is a trick that makes them much faster to train.

Helmholtz machines are directed models. The generative process is forwards $\mathbb{P}(h^2|h^3), \mathbb{P}(h^1|h^2), \mathbb{P}(x|h^1)$– sampling and probability evaluation is tractable for each $p(h^l|h^{l+1})$. Approximate inference going to the other way is harder.

There are 2 terms

$$\ln p(x) = \ln \mathop{\mathbb{E}}_{q(h|x)} \left[\frac{p(x,h)}{q(h|x)}\right] \geq \mathop{\mathbb{E}}_{q(h|x)} \left[\ln \frac{p(x,h)}{q(h|x)}\right] = L(x).$$

How do you optimize in terms of p, q? q is the recognition network.

Assume recognition distribution is Gaussian.

$$q(h^l|h^{l-1}, \theta) = N(\mu(h^{l-1}, \theta), \Sigma(h^{l-1}, \theta)).$$

We can express this in terms of auxiliary variable Recognition distribution can be expressed in terms of determinisitc mapping.

How do we compute the gradient? Reparametrize in terms of ε.

$$\nabla_\theta \mathbb{E}_{h\sim q(h|x,\theta)} \left[\ln\left(\frac{p(x,h|\theta)}{q(h|x,\theta)}\right)\right] \tag{83}$$

$$= \nabla_\theta \mathop{\mathbb{E}}_{\varepsilon^1,\ldots\varepsilon^L\sim N(0,I)} \left[\ln \frac{p(x,h(\varepsilon,x,\theta)|\theta)}{q(h(\varepsilon,x,\theta)|x,\theta)}\right] \tag{84}$$

$$= \mathop{\mathbb{E}}_{\varepsilon^1,\ldots\varepsilon^L\sim N(0,I)} \left[\nabla_\theta \ln \frac{p(x,h(\varepsilon,x,\theta)|\theta)}{q(h(\varepsilon,x,\theta)|x,\theta)}\right]. \tag{85}$$

Here h is a deterministic neural net for fixed ε.

Instead of comupting mean first, compute gradient and then mean. We can backpropagate through whole system, through the latent variables! Sample once and backpropagate. Break the system to stochastic and deterministic parts. Instead of single sample, you can draw multiple samples. We can improve VAE by using a importance weighting of log-likelihood $w_i = p(x,h_i)/q(h_i|x)$.

$$L_k(x) = \mathbb{E}_{h\sim q(h|x)} \left[\ln \frac{1}{k}\sum_{i=1}^{k} p(x,h_i)/q(h_i|x)\right] \leq \ln p(x).$$

q is usually a product distribution.

Entropy term makes sure you spread things out.

Autoencoders are not generative model. These are truly generative models.

Using more samples can only improve tightness of the bound. For all k,the lower bounds satisfy

$$\ln p(x) \geq L_{k+1}(x) \geq L_k(x).$$

Compute gradients using reparameterization trick.

The MC estimate of gradient both use $\nabla_\theta w(x, h(\varepsilon_i, x, \theta), \theta)$, but we average differently. some will be better, put more weight on them.

Can we generate images from natural language descriptions? "Stop sign flying in blue skies." There is a generative model and an inference model. "A toilet seat sits open in the grass field."

7.4 Generative adversarial ntework

There is no explicit definition of density for $p(x)$. They work remarkably well.

Set up a game between 2 players, discriminator D and generator G. The discriminator D tries to discriminate between a sample from the data distribution and a sample from the generator G.

Given x sampled from data, D tries to output 1, meaning that it came from the data. The discriminator tries to output 0 for data generated from the generator (fake data). The generator tries to get the discriminator to output 1.

$$\min_G \max_D V(D, G) = \mathop{\mathbb{E}}_{x \sim p_{data}(x)} [\ln D(x)] + \mathop{\mathbb{E}}_{z \sim p_x(z)} [\ln(1 - D(G(z))].$$

The generator pushes down, the discriminator pushes up. The discriminator tries to classify data as being real, and generator samples as being fake. The optimal strategy for discriminator is

$$D(x) = \frac{p_{data}(x)}{p_{data}(x) + p_{model}(x)}.$$

DCGAN combines with convolutional network.

7.5 Model evaluation

Ex. Bedrooms, CIFAR. Is the model just memorizing the samples, or is it generating these images?

It's hard to define distance in pixel space. There's an open question of how to evaluate these models.

It does generate something different from the training set. What's the objective of this generative model?

People have looked at conditional simulations: conditioning on a sketch can you generate a real image? conditioning on a sketch it hasn't seen before, can it generate a real image?

For RBM, the partition function is intractable.

Model A: Pick a training example at random and display it. But this doesn't generalize. Mixture of Bernoullis has test log probability -138nats/digit; RBM is 50 higher.

Suppose two distributions defined on X have probability distribution $p_i(x) = f_i(x)/Z_0$, and $p_t(x) = f_t(x)/Z_t$. We can get unbiased estimate using MC approximation (Under mild condition on support) $Z_t \approx \frac{1}{M} \sum_{m=1}^{M} \frac{f_t(x^m)}{p_i(x^m)}$. But the variance is high.

Annealed importance sampling (AIS): consider a sequence of intermediate distributions $p_0, p_1, \ldots, p_K$ with $p_0 = p_i$ and $p_K = t$. One way is to use geometric averages $p_\beta(x) =$

$f_i(x)^{1-\beta} f_t(x)^{\beta}$. We need to define transition operator $T_k(x'|x)$ that leaves p_k invariant, like Gibbs sampling; this is easy to implement.

Importance sampling between adjacent distribution. Break into pieces and estimate each piece separately.

AIS provides an unbiased estimator $\mathbb{E}[\widehat{Z}_t] = Z_t$. But we are interested in estimating $\ln Z_t$. By Jensen's inequality $\mathbb{E}[\ln \widehat{Z}_t] \leq \ln Z_t$. By Markov's ineq it is very unikely to overestimate $\ln Z_t$. But underestimating $\ln Z_t$ overestimates

$$\ln p(x) = \ln f(x) - \ln Z_t.$$

Failing to estimate $\ln Z$ gets misleadingly good log probability estimates.

Upper bounds for general graphical models are difficult.

Run Markov chain for $T = 1000$ steps and assume that's the equilibrium distribution. Forget about graphical model: unroll the RBM as a deep generative model.

Let $x = \{v, h\}$, v observed, h hidden. Reverse AIS estimator: Generative model $p_f(x_{0:K}) = p_0(x_0) \prod_{k=1}^{K} T_k(x_k|x_{k-1})$. $\mathbb{E}_{q_{rev}}[v]$ will be an underestimate.

Use reverse chain as proposal: $q_r(x_{0:K-1}, h_K|v_t) = p_t(h_K|v_t) \prod_{k=1}^{K} \widetilde{T}_k(x_{k-1}|x_k)$.

The higher the average test set $\ln p$ the better (cf. compression). There is a gap between AIS and RAISE; they give higher and lower bound on the log probabilities. The more intermediate distributions we use the smaller the gap. Sometimes we can get tight bounds.

Decoder-based model transform sample from simple distribution to data manifold, $p(x, z) = p(x|z)p(z)$ where $p(x|z)$ is calculated by deterministic neural network. AIS can be used to properly evaluate decoder-based models.

Ex. if model says horse on a table is somewhat probable (though less probably than just horse), then it generalizes.

I want, when I show a test example the model has never seen. I want the model to believe it's a realistic example.

If I can generate new examples from specific class that I can add to my dataset to improve classification, that's good. This algorithm would be improving something else. It's hard to argue against this measure! To some extent it's been used, but it's not well developed yet. RL use generative models to generate possible futures.

Open problems include unsupervised/transfer/one-shot learning; reasoning, attention and memory; natural language understanding; deep RL.

7.5.1 Natural language

Sequence-to-sequence models. Skip-thought model: predict next and previous sentence. Inductive bias: if surrounding contexts look the same, sentences should be identical.

You can get good results on tasks like semantic relatedness (SemEval).

Best published models use linguistic knowledge. Our model which just looks at a lot of books gets close.

Neural storytelling: generate sentences based on the image. Train on 7000 romantic models.

The syntax looks good. Semantically you make mistake. There is inconsistence (sunrise, sunset). How can you generate coherent pieces of text? Even harder is to evaluate.

Reading comprehension goes beyond standard classification of documents. "Who did what" dataset. Machines get 60% accuracy, people get $95-97\%$. Some better datasets were by Mechanical Turk, like SQuAD. Bidirectional RNN. Multi-hop architecture: for every word, define relationship. Most systems are based on pattern-matching. How do well words in query match words in the document? If there is a match, pass to the next level.

There are many models. How do you search? Questions that require looking at multiple places are much harder.

7.5.2 RL

Can a single network play many games at once? Can we learn new games using knowledge from previous games? Ex. if there are things moving at you, shoot them.

8 Tensor Decompositions for Learning Latent Variable Models, Daniel Hsu

We give learning algorithms (parameter estimation) for latent variable models based on decompositions of moment tensors.

Examples:

1. Summarize a corpus of documents. Documents express one or more thematic topics.

   We want to know what topics are expressed, and how prevalent each topic is in the corpus.

   We can fit a latent variable model, a topic model (like latent Dirichlet allocation).

   Suppose there are K topics, giving distributions over vocab words. A document is a mixture of topics. The word tokens in document are drawn iid from this mixture distribution.

   The learning problem: given corpus of documents and hyperparameters (K), produce estimates of model parameters: For $t \in [K]$,

   - distribution P_t over vocab words
   - weight w_t of topic t in document corpus.

   This would be just counting if each word token x in document is annotated with the source topic $t_x \in [K]$. Then estimating the $\{(P_t, w_t)\}_{t=1}^K$ can be done directly.

   Unfortunately, we don't have such annotation: topics are hidden. The direct approach to estimation is unavailable.

2. Subpopulations in data: Pearson (1894) measured the ratio of forehead-width to body-length for 1000 crabs. It was not a normal distribution.

   Pearson hypothesized that it was 2 populations that are each gaussian. He used the method of moments to fit a mixture of 2 gaussians.

(Later it turned out to be just one species...)

$$H \sim \text{Discrete}(\pi_1, \ldots, \pi_K) \tag{86}$$
$$X|H = t \sim \text{Normal}(\mu_t, \Sigma_t), \quad t \in [K]. \tag{87}$$

We want to learn the mean and covariance for each, without knowing the classes of the data points.

There is no direct estimators for MLE when some variables are hidden. The MLE is

$$\theta_{MLE} := \text{argmax}_{\theta \in \Theta} \ln \mathbb{P}_\theta \left(\text{data}\right).$$

The log-likelihood is not necessarily a concave function of θ. This hasn't stopped anyone in ML or statistics: people use Expectation-Maximization (EM).

For Gaussian mixture models, the MLE is: given $\{x_i\}_{i=1}^n$, find $\{(\mu_t, \Sigma_t, \pi_t)\}_{t=1}^K$ to maximize

$$\sum_{i=1}^n \ln \left(\sum_{t=1}^K \pi_t \frac{1}{\det(\Sigma_t)^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x_i - \mu_i)^T \Sigma_t^{-1} (x_i - \mu_t) \right) \right).$$

This is sensible with restrcitions on $\Sigma_t \succeq \sigma^2 I$, but this is similar to the Euclidean K-means problem which is NP-hard.

Perhaps the instance you get is not worst-case. Look at the perspective of average-case analysis.

Suppose we get an iid sample of size n generated from distributions with unknown parameters $\theta \in \Theta \subseteq \mathbb{R}^p$, where p is the number of parameters.

The task is to produce an estimate $\widehat{\theta}$ of θ such that $\mathbb{E} \left\| \widehat{\theta} - \theta \right\| \to 0$ as $n \to \infty$ (i.e. $\widehat{\theta}$ is consistent). (More precisely, $\mathbb{P}(\left\| \widehat{\theta} - \theta \right\| \leq \varepsilon) \geq 1 - \delta$ in polytime.)

For small models ($K = 2, \pi_t = \frac{1}{2}, \Sigma_t = I$), EM can be consistent, but for larger K, it is easily trapped in local maxima far from global maxima. Practitioners often use EM with many random restarts.

It is hard to learn model parameters even when data is generated by model distribution. There is cryptographic hardness or information-theoretic hardness; you many need $2^{\Omega(K)}$ running time or $2^{\Omega(K)}$ sampling size.

(Why do you want to estimate parameters; why not just get a close distribution? We want to learn something specific about the populations.)

Ways around these barriers:

1. Rule out hard cases with separation conditions. For example, assume

$$\min_{i \neq j} \frac{\left\| \mu_i - \mu_j \right\|^2}{\sigma_i^2 + \sigma_j^2}$$

   is large

2. Structural assumptions: sparsity, anchor words

3. We assume non-degeneracy conditions: μ_i are in general position.

There is a broad class of techniques called method of moments that work.

The high-level idea:

1. Determine function of model parameters θ estimable from observable data, $\mathbb{E}_\theta[f(X)]$. Often third-order moments suffice.

2. Form estimates of moments using data (iid sample)

$$\widehat{\mathbb{E}}[f(X)].$$

3. Approximately solve equations for parameters θ $\mathbb{E}_\theta[f(X)] = \widehat{\mathbb{E}}[f(X)]$. (Algorithms for tensor decomposition).

4. Fine-tune estimated parameters with local optimization.

Model misspecification is an unresolved issue. We don't have a general methodology; now it is ad hoc, guided by examples. It's not clear how to incorporate general prior knowledge, and user feedback.

There are many other models amenable to moment tensor decomposition.

8.1 Topic model for single-topic documents

For simplicity, say there is a single topic for each document. The generative process is $t \sim \text{Discrete}(w_1, \ldots, w_K)$; given t pick L words from P_t. Given iid sample of documents of length L, produce estimates of model parameters $\{(P_t, w_t)\}_{t=1}^K$.

How long must the documents be for identifiability?

- $L = 1$: A random document is $\sim \sum_{t=1}^K w_t P_t$. The parameters are not identifiable.

- $L = 2$: regard P_t as probability vector. The joint distribution of word pairs is given by matrix: A random document is $\sum_{t=1}^K w_t P_t P_t^T$. The parameters are still not identifiable because the matrix decomposition is not unique.

  You can also conclude this based on heuristic from parameter counting.

- $L = 3$: Parameters are identifiable. A random document has three-wise co-ocurrences $\sim \sum_{t=1}^K w_t P_t^{\otimes 3}$,

  Claim 8.1. *If $\{P_t\}_{t=1}^K$ are linearly independent and all $w_t > 0$, then parameters $\{(P_t, w_t)\}_{t=1}^K$ are identifiable from word triples.*

  This is implied by uniqueness of tensor decompositions.

First, background on tensors.

Matrices are tensors of order 2 which can be thought of as bilinear functions $M : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, $M(x, y) = x^T M y = \sum_{i,j} M_{ij} x_i y_j$.

A tensor of order p is a p-linear function $T : \mathbb{R}^d \times \cdots \mathbb{R}^d \to \mathbb{R}$. Describe T by d^p values $T(e_{i_1}, \ldots, e_{i_p})$. Identify T with a multi-index array $T \in \mathbb{R}^{d \times \cdots \times d}$.

Most tensor problems are NP-hard (Hillar, Lim 2013). It is remarkable that we can do anything at all.

Task: Given tensor $T = \sum_{t=1}^K v_t^{\otimes 3}$ with linearly independent components $\{v_t\}_{t=1}^K$, find the components up to scaling.

Jennrich's algorithm is based on collapsing the tensor. Think of $T : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ as $T : \mathbb{R}^d \to \mathbb{R}^{d \times d}$. (Cf. currying in functional programming.)

$$[T(x)]_{j,k} = T(x, e_j, e_k).$$

Collapse 2 different ways.

Algorithm 8.2 (Jennrich)**:** 1. Pick x, y independently and uniformly from $\mathbb{S}^{d-1}$.

2. Compute and return eigenvectors of $T(x)T(y)^+$. with nonzero eigenvalues.

For $T = \sum_{t=1}^K v_t^{\otimes 3}$,

$$T(x) = \sum_{t=1}^K \langle v_t, x \rangle \, v_t v_t^T = V D_x V^T.$$

where $V = [v_1 | \cdots | v_K]$ and $D_x = \operatorname{diag}(\langle v_1, x \rangle, \ldots, \langle v_K, x \rangle)$. By linear independence of $\{v_t\}_{t=1}^K$ and random choice of x, y,

1. V has rank K
2. D_x, D_y are invertible a.s.
3. diagonal entries of $D_x D_y^{-1}$ are distinct a.s.
4. $T(x)T(y)^+ = V(D_x D_y^{-1})V^+$ a.s.

So $\{v_t\}_{t=1}^K$ are the eigenvectors of $T(x)T(y)^+$ with distinct non-zeros eigenvalues. We actually want the v_t to be linear independent enough, well-conditioned.

(Think of Jennrich as randomly combining different slices of tensor cube.)

We can generalize this to the asymmetric case.

For the topic model, the probability of word triples is a third-order tensor

$$T = \sum_{t=1}^K P_t^{\otimes 3} = \sum_{t=1}^K v_t^{\otimes 3}$$

where $v_t = w_t^{\frac{1}{3}} P_t$. $\{v_t\}_{t=1}^K$ linearly independent means $\{P_t\}_{t=1}^K$ are linearly independent and $w_t > 0$. We can recover $\{P_t\}_{t=1}^K$ and then w_t.

In conclusion, parameters of topic model for single-topic documents satisfying linear independence condition can be efficiently recovered from distribution of three-word documents. Two-word documents are not sufficient.

It actually does something reasonable on real data: 300,000 NYT articles, 102660 words, $K = 50$ topics. This is a 4-8 times speed-up over Gibbs sampling for LDA, comparable to FastLDA.

Some issues are accuracy of moment estimates (can more reliably estimate lower-order moments, get distribution-specific sample complexity bounds), robustness of approximate tensor decomposition (can use more error-tolerant decomposition algorithm, also computationally efficient), generality beyond simple topic models.

8.2 Moment decomposition for other models

We will discuss two classical mixture models, and models with multi-view structure.

Consider mixtures of spherical gaussians

$$H \sim \text{Discrete}(\pi_1, \ldots, \pi_K) \tag{88}$$

$$X|H = t \sim \text{Normal}(\mu_t, \sigma_t I_d), \quad t \in [K]. \tag{89}$$

For simplicity, restrict to $\sigma_1 = \cdots = \sigma_K = \sigma$.

The generative process is $X = Y + \sigma Z$ where $\mathbb{P}(Y = \mu_t) = \pi_t$ and $Z \sim N(0, I_d)$ is Gaussian noise. The moments are

$$\mathbb{E}X = \sum_{t=1}^{K} \pi_t \mu_t \tag{90}$$

$$\mathbb{E}(X \otimes X) = \sum_{t=1}^{K} \pi_t \mu_t \otimes \mu_t + \sigma^2 I_d. \tag{91}$$

The span of the top K eigenvectors of $EX \otimes X$ contains $\{\mu_t\}_{t=1}^K$.

Quantify separation by the number of standard deviations between component means

$$sep : - \min_{i \neq j} \frac{\|\mu_i - \mu_j\|}{\sigma}.$$

Distance-based clustering (EM) works when $sep \gtrsim d^{\frac{1}{4}}$. Problem becomes K-dimensional via PCA when $sep \gtrsim K^{\frac{1}{4}}$, $K \leq d$.

Third-order moments identify the mixture distribution when $\{\mu_t\}_{t=1}^K$ are linearly independent. Separation may be arbitrarily close to 0. People have also looked at general Gaussian and no minimum separation, but require $\Omega(K)$th order moments (exponential complexity).

The third-order moment tensor is

$$\mathbb{E}(X^{\otimes 3}) = \mathbb{E}(\{Y+\sigma Z\}^{\otimes 3}) = \mathbb{E}(Y^{\otimes 3}) + \sigma^2 \mathbb{E}(Y \otimes Z \otimes Z + Z \otimes Y \otimes Z + Z \otimes Z \otimes Y) = \sum_{t=1}^{K} \pi_t \mu_t^{\otimes 3} + \tau(\sigma^2, \mu).$$

This by itself does not look useful. But μ, σ^2 are functions of $\mathbb{E}X$ and $\mathbb{E}(X \otimes X)$, so we can estimate τ.

Claim 8.3. *If $\{\mu_t\}_{t=1}^K$ are linearly independent and all $\pi_t > 0$, then $\{(\mu_t, \pi_t)\}_{t=1}^K$ are identifiable from*

$$T := \mathbb{E}(X^{\otimes 3}) - \tau(\sigma^2, \mu) = \sum_{t=1}^{K} \pi_t \mu_t^{\otimes 3}.$$

Then you can use e.g. Jennrich's algorithm to recover $\{(\mu_t, \pi_t)\}_{t=1}^K$ from T.

If not linearly independent, you may need lots of data.

Note linear independence condition on $\{\mu_t\}_{t=1}^K$ requires $K \leq d$.

Many works relax this to learn more mixtures of gaussians. They analyze mixtures of $d^{O(1)}$ gaussians with simple or known covariance via smoothed analysis and $O(1)$ order moments. Ge, Huang, Kakade 2015 generalize to arbitrary unknown covariances. This is the overcomplete case.

Mixtures of linear regressions is another common mixture model

$$H \sim \text{Discrete}(\pi_1, \ldots, \pi_K) X \sim N(\mu, \Sigma) \tag{92}$$

$$Y|H = t, X = x \sim N(\langle \beta_t, x \rangle, \sigma^2) \tag{93}$$

The second-order moments give you a lot of information (assume $X \sim N(0, I_d)$)

$$\mathbb{E}(Y^2 X X^T) = 2\sum_{t=1}^K \pi_t \beta_t \beta_t^T + \left(\sigma^2 + \sum_{t=1}^K \pi_t \|\beta_t\|^2\right) I_d \tag{94}$$

The span of top K eigenvectors contains $\{\beta_t\}_{t=1}^K$. Using Stein's identity, similar approach works for GLMs.

Tensor decomposition can recover parameters $\{(\beta_t, \pi_t)\}_{t=1}^K$.

8.2.1 Multi-view models

In topic model for single-topic documents, there are K topics $\{P_t\}_{t=1}^K$. Pick topic $H = t$ with probability w_t (hidden). Words are $X_1, \ldots, X_L \sim P_H$ iid. The key property $X_1, \ldots, X_L$ are conditionally indepenent given H. Each word token X_i provides a new "view" of hidden variable H.

Blum, Mitchell 1998 use this to justify co-training in semi-supervised learning.

The views can be completely different; what's important is conditional independence.

$$\mathbb{E}(X_1 \otimes X_2 \otimes X_3) = \sum_{t=1}^K \pi_t \mu_t^{(1)} \otimes \mu_t^{(2)} \otimes \mu_t^{(3)}.$$

where $\mu_t^{(i)} = \mathbb{E}[X_i | H = t]$, $\pi_t = \mathbb{P}(H = t)$. Jennrich's algorithm works in the asymmetric case.

Examples of multi-view mixture models are

1. Mixtures of high-dimensional product distributions
2. Hidden Markov models
3. Phylogenetic trees (X_1, X_2, X_3 are genes of 3 extant species, and H is the LCA (lowest common ancestor) of most closely related pair of species)

We never exactly have the moments. We can only estimate moments with empirical moment tensor $\widehat{T}$, and approximately decompose $\widehat{T}$ to get parameter estimate $\widehat{\theta}$. We want algorithms that work for tensors that are close to having this form.

8.3 Error-tolerant algorithms for tensor decompositions

We can estimate $\mathbb{E}[X^{\otimes 3}]$ from iid sample $\{x_i\}_{i=1}^n$, $\widehat{[X^{\otimes 3}]} := \frac{1}{n}\sum_{i=1}^n x_i^{\otimes 3}$. We expect error of order $n^{-\frac{1}{2}}$ in some norm, $\|T\| = \sup_{x,y,z\in\mathbb{S}^{n-1}} T(x,y,z)$ operator norm or $\|T\|_F := \sum T(e_i,e_j,e_k)^2$ Frobenius norm.

We only have $\widehat{T}$, with say $\left\|\widehat{T}-T\right\| \precsim n^{-\frac{1}{2}}$, $T = \sum_{t=1}^K v_t^{\otimes 3}$.

Stability of eigenvectors requires eigenvalue gaps

$$\Delta := \min_{i\neq j}\left|\frac{\langle v_i, x\rangle}{\langle v_i, y\rangle} - \frac{\langle v_j, x\rangle}{\langle v_j, y\rangle}\right|.$$

We need $\left\|\widehat{T}(x)\widehat{T}(y)^+ - T(x)T(y)^+\right\| \ll \Delta$ so $\widehat{T}(x)\widehat{T}(y)^+$ has sufficient eigenvalue gaps. We need $\left\|\widehat{T}-T\right\|_F \ll \frac{1}{\mathrm{poly}(d)}$ for this.

We instead reduce to orthogonal case by considering a different set of moments. In many applications we estimate moments of the form

$$M = \sum_{t=1}^K v_t \otimes v_t \tag{95}$$

$$T = \sum_{t=1}^K \lambda_t v_t \otimes v_t \otimes v_t. \tag{96}$$

Assume v_t linearly independent, λ_t positive. M is positive semidefinite of rank K so determines inner product system on $\mathrm{span}\{v_t\}_{t=1}^K$ such that $\{v_t\}_{t=1}^K$ are orthonormal. The process is **whitening**.

Our goal: given $\widehat{T} \in \mathbb{R}^{d\times d\times d}$ such that $\left\|\widehat{T}-T\right\| \leq \varepsilon$ for some $T = \sum_{t=1}^d \lambda_t v_t^{\otimes 3}$ where $\{v_t\}_{t=1}^d$ are orthonormal and $\lambda_t > 0$, approximately recover $\{(v_t,\lambda_t)\}_{t=1}^d$.

The analogous matrix problem:

1. For $\varepsilon = 0$ this is eigendecompsotion. unique if $\{\lambda_t\}_{t=1}^d$ distinct.
2. For $\varepsilon > 0$ use perturbation theory for eigenvalues (Weyl) and eigenvectors (Davis and Kahan).

First assume $\varepsilon = 0$ so $\widehat{T} = T$. Try to match moments by optimization

$$\{(\widehat{v}_t,\widehat{\lambda}_t)\}_{t=1}^d := \mathrm{argmin}_{\{(x_t,\sigma_t)\}_{t=1}^d}\left\|T - \sum_{t=1}^d \sigma_t x_t^{\otimes 3}\right\|_F^2.$$

Greedy approach is to find best rank 1 approximation, deflate $T := T - \widehat{\lambda}\cdot\widehat{v}^{\otimes 3}$ and repeat. For orthogonal decomposition, this works! (Note the optimization is equivalent to $\widehat{v} = \mathrm{argmin}_{x\in\mathbb{S}^{d-1}} T(x,x,x)$.)

Claim 8.4. *The local maximizers of the function*

$$x \mapsto T(x,x,x) = \sum_{t=1}^d \lambda_t \langle v_t, x\rangle^3$$

are $\{v_t\}_{t=1}^d$ *and* $T(v_t,v_t,v_t) = \lambda_t$.

The algorithm is to use gradient ascent to find each component v_t.

A slightly different algorithm, nicer, is parameter-free fixed-point algorithm, due to De Lathauwer, De Moore, Vandewalle 2000. The first-order (necessary but not sufficeint) optimality condition is $\nabla_x T(x,x,x) = \lambda x$. The partial evaluation of T is

$$\nabla_x T(x,x,x) = 3\sum_{i,j} T_{i,j,k} x_i x_j e_k = 3T(x,x,\cdot).$$

The third-order tensor power iteration is

$$x^{(i+1)} = \frac{T(x^{(i)}, x^{(i)}, \cdot)}{\|T(x^{(i)}, x^{(i)}, \cdot)\|}.$$

This is only in the orthogonal case. This looks like matrix power iteration but has different properties.

For matrix power iteration $x^{(i+1)} = \frac{Mx^{(i)}}{\|Mx^{(i)}\|}$, we require gap $\min_{i \neq 1} 1 - \frac{\lambda_i}{\lambda_1} > 0$ to converge to v_1. For tensor power iteration no gap is required (note the solution is unique even without a gap).

Where we converge depends on the initialization. In matrix power iteration, if $\langle v_1, x^{(0)} \rangle \neq 0$, gap > 0, then converges to v_1. If $t := \text{argmax}_{t'} \lambda_{t'} |\langle v_{t'}, x^{(0)} \rangle|$, it converges to v_t. Matrix power iteration converges at linear rate and tensor power iteration converges at quadratic rate.

Now allow $\varepsilon > 0$.

Claim 8.5. *For $\widehat{v} := \text{argmax}_{x \in \mathbb{S}^{d-1}} \widehat{T}(x,x,x)$,*

$$|\widehat{\lambda} - \lambda_t| \leq \varepsilon, \|\widehat{v} - v_t\| \leq O\left(\frac{\varepsilon}{\lambda_t} + \left(\frac{\varepsilon}{\lambda_t}\right)^2\right)$$

for some $t \in [d]$, $\lambda_t \geq \max_{t'} \lambda_{t'} - 2\varepsilon$.

There are many efficient algorithms for solving this approximately when ε is small enough ($\frac{1}{d}, \frac{1}{\sqrt{d}}$). Think of d as k in the original problem.

We accumulate errors from deflation. (If the λ's are the same, we can just restart with the original tensor, instead of deflating and recursing.)

$$\widehat{T} - \widehat{v}_1^{\otimes 3} = \sum_{t=2}^{d} v_t^{\otimes 3} + E + (v_t^{\otimes 3} - \widehat{v}_1^{\otimes 3}).$$

Error seems to have doubled.

$$\left\| \frac{1}{3} \nabla_x \left[(v_1^{\otimes 3} - \widehat{v}_1^{\otimes 3})(x,x,x) \right] \right\| = \left\| \|v_1, x\|^2 v_1 - \langle \widehat{v}_1, x \rangle^2 \widehat{v}_1 \right\| \tag{97}$$

$$= \langle \widehat{v}_1, x \rangle^2 \tag{98}$$

$$\leq \|v_1 - \widehat{v}_1\|^2 \leq \varepsilon^2. \tag{99}$$

So effect of errors in directions orthogonal to v_1 is $(1+o(1))\varepsilon$ rather than 2ε. Deflation errors have lower-order effect on finding other v_t: analogous statement for deflation with matrices does not hold.

We can also use alternating minimization, but don't know how to analyze it theoretically.

Recap: reduction to nearly orthogonal decomposable tensor permits simple and error-tolerant algorithms. Differences from matrix decomposition (nonlinearity) are crucial.

There are many issues to resolve: handle model misspecification, increase robustness; develop general methodology, incorporate general prior knowledge, incorporate user feedback interactively.

Unlike in some other settings, here the data have to be plausibly generated from the assumed model.

Main use of these methods may be to initialize local search procedure.

See Anandkumar, Ge, Hsu, Telgarsky, Tensor decompositions for learning latent variable models, 2014, and Moitra, Algorithmic aspects of machine learning, 2014 (Chapter 3).

9 Natural language understanding

Language is rich and interesting and makes us human. There are foundational language. We'll talk about language without worrying about what techniques we'll use.

Turing was the first to think about the relationship of language to AI. He was concerned with: can machines think? He came up with the Turing test: if it can convince a human it's human it passes. The Turing test has been criticized but there are 3 important lessons.

1. It's an end-to-end approach: an evaluation that doesn't reference how the task is solved.

2. It's an interactive test, which is more interesting.

3. The end goal is AI—language is a mechanism he uses to assess intelligence.

Language is about meaning and understanding. I want to distinguish "understanding" from "processing".

SHRDLU (1971): Terry Winograd build a system where a human can interact with a computer to move blocks around. The dialogue is quite complex in certain dimensions.

It's unlike systems we have today which are specific in some sense.

But these systems didn't scale up. One year later Terry Winograd thought natural language understanding was a dead end. It was a software engineering problem. Winograd had a flourishing career in human-computer interaction and gave up on this.

The 1990's saw the statistical revolution. We have more computational power and data. Algorithms are parsing sentences on the internet. They changed the problem: dependency parsing (figure out parts of speech and relationships), word vectors.

Here's a way to think about the field in broad terms: breadth vs. depth. SHRDLU is deep but only works in limited domains. Parsing has breadth but is not deep.

There is opportunity for transfer of ideas between ML and NLP. There has been lots of interaction.

- mid-1970's: HMMs for speech recognition. HMM is an example of probabilistic models.

- early 2000's: conditional random fields for part-of-speech tagging. This led to structured prediction: don't predict a single label but entire structure.

- mid 2010's: attention-based sequence-to-sequence models for machine translation. People developed recurrent neural networks with memory in response to more challenging questions like machine translation.

What is the right way to think about natural language understanding?

9.1 Properties of language

I'll first deep-dive into properties of language. We all speak it but don't always think about it outside language. I'll go through several paradigms people use to think about language.

Levels of linguistic analyses are:

- syntax: what is grammatical. No compiler errors
- semantics: what does it mean? No implementation bugs
- pragmatic: what does it do? Implement the right algorithm.

The basic unit is a word, ex. light. There are multi-word expressions with meaning beyond word, behaving like word, ex. light bulb. There is morphology: meaning unit within word: lighten, lightening, relight.

Polysemy: one word has multiple meanings (word senses).

Synonymy: Different words can have the same meaning. Sentences can have the same meaning.

Think about distance metric, similarity.

Hyponymy (is-a), meronomy (has-a). This is useful for entailment.

Compositional semantics has two ideas which often get conflated.

1. Model theory: sentences refer to the world, "block 2 is blue".
2. Compositionality: The meaning of whole is meaning of parts. "The [block left of the red block] is blue."

Why is language difficult?

- Universal and existential quantification: Every, some.
- There is quantifier scope ambiguity: Every non-blue block is next to some blue block.

  Some theorem statements have this property...
- Modality: Block 2 must be blue. Block 1 could be red.
- Luis believes Superman is a hero doesn't mean Lois believes Clark Kent is a hero. You can't substitute in belief.
- Pragmatics: There is meaning that is not literal meaning.

- Conversational implicature: new material suggested (not logically implied) by sentence.

  What on earth has happened to the raost beef? The dog is looking very happy.

  All the non-logical stuff is sometimes said to be in the "pragmatic wastebasket".

- Presupposition: background assumption independent of truth of sentence. "I have stopped eating meat" has the presupposition "I once was eating meat". (An alternative is to define "stop" by saying you can logically stop doing something if you were once doing it.)

  Persuasive people know this and use this for traps.

Semantics is what does it mean literally? Pragmatics is what is the speaker really conveying? There is not a strong line.

Grice (1975) stated the principle that language is cooperative game between speaker and listener.

Implicatures and presuppositions depend on people and context and involves soft inference. There are machine learning opportunities here! Statistical tools are much better for this than logical tools.

What makes things hard? Probability is a currency to measure uncertainty.

- Vagueness: speaker does not specify full information. "I had a late lunch."
- Ambiguity: more than one possible (precise) interpretations: "One morning I shot an elephant in my pajamas. How he got in my pajamas, I don't know." –Groucho Marx
- Uncertainty: The witness was being contumacious. (What does "contumacious" mean?)

SO far we've talked about analyses, lexical semantics, compositional semantics, and challenges. Sometimes language is crisp; sometimes it's loose in all kinds of ways.

Models don't have the same access to the tools humans have to disambiguate. Humans have context; models are impoverished.

We give 3 ways to model semantics.

9.2 Distributional semantics

The new design has ? lines. Lety's try to keep the kitchen ?. CLEAN.

Observation: context can tell us a lot of word meaning. The context is the local window around a word occurrence (for now).

Distributional hypothesis: Semantically similar words occur in similar contexts (Harris 1954). You shall know a world by the companty it keeps (Firth 1957). Contrast with Chomsky's generative grammar (lots of hidden prior structure and no data). This is a ML (data) friendly viewpoint.

The general recipe is:

1. Form a word-context matrix of counts (data)

2. Perform dimensionality reduction (generalize). Get word vectors $\theta_w \in \mathbb{R}^d$.

For latent semantic analysis do SVD: $N \approx \Theta S V^T$.

Skip-gram model with negative sampling (word2vec). The context is words in a window. Do logistic regression with SGD, predict good (w, c) using logistic regression

$$\mathbb{P}(g = 1|w, c) = (1 + \exp(\theta_w \cdot \beta_c))^{-1}.$$

Positives are (w, c) from data, and negatives are (w, c') for irrelevant c' (Use k times more negative data).

There are many other ways to do this. Any type of dimensionality reduction gives something similar. You can make a lot of nice plots (ex. 2d visualization of word vectors). Structure emerges; ex. country names are together, pronouns, days of week... It learns some structure. Words that occur in similar contexts are put close by.

What's missing? Does this give you synonyms? Two words are similar if they occur in the same context. "Cherish" is close to adore, love, admire, embrace, rejoice. Tiger is close to leopard (not a synonym), etc. Good is close to bad (antonym). There are many notions all jumbled up.

Suppose Barack Obama always appears together, a collocation. Using Global context (document-level), then $\theta_{Barack} \approx \theta_{Obama}$ because they occur in the same context. Using local context (neighbors), the context is different, so they are different.

With the premise that semantics is context, use the recipe: form word-context matrix and use dimensionality reduction. This is data-friendly, and not to be underestimated, it captures nuance.

But what is the context? There is no such thing as pure unsupervised learning; representation depends on choice of context. Language is not just text in isolation; context should include world/environment.

Example.

- Cynthia sold the bike for $200.
- The bike sold for $200.

Cynthia and the bike appear in the same context but not in the same way.

9.3 Frame semantics

Distributional semantics is monolithic, no internal structure on word vectors.

Frame semantics: meaning is given by a frame, a stereotypical situation. For example, "sell" brings to mind a commercial transaction: seller, buyer, price.

Two properties of frames:

- prototypical: We don't need to handle all the cases.

  For example, frame of "widow": woman marries one man, man dies. What if woman has 3 husbands, 2 died?

- Profiling: highlight one aspect: sell is seller-centric, buy is buyer-centric. Rob highlights person, steal highlighs good.

Linguistics: case grammar (Fillmore 1968). AI: frames (Minsky).

There are many ways of evoking the frame. Just because Cynthia is a subject of "sold" and the biks is a subject of "sold" doesn't mean they go in the frame the same way.

The task is semantic role labeling; you can come up with ML algorithms. Given a sequence of words (inputs), do

1. frame identification (predicate)

2. argument identification (seller, goods)

Frames are usually verbs but can also be prepositions and nouns.

AMR (abstract meaning representation): More recent (the last 4 years) people thought of a more ambitious representation, normalizes "Cynthia" and "she" are the same thing. "The boy wants to go to New York City" becomes a graph of relationships. Visit, want are frames. This is an interesting structure prediction problem: given a sentence, create a graph.

Frames are a stereotypical situations that provide rich structure for understanding.

Are frames normally done by hand? The construction of what frames and arguments are has always done by hand. How map sentences to frames? Automatically. It's more difficult to learn frames and arguments automatically.

Both distributional semantics (DS) and frame semantics (FS) involve compression/abstraction. Frame semantics exposes more structure and are more tied to an external world, but require more supervision. They are less pure from a data perspective.

Cynthia went to the bike shop yesterday. Cynthia bought the cheapest bike. Frames don't tell you that yesterday means "1/26" and "cheapest" means lowest price.

Can frames provide a better notion of context for distributional semantics? They can provide a better prior. This is what I am looking for. They are an n-ary relation, like a tensor, whereas word2vec is bilexical.

Every utterance we form carries with it the whole history of humankind so it's incredibly complex? Frames do not capture all the meaning. There's no substitute for the whole sentence plus its entire context. Context that's kept needs to be sufficiently rich. DS give you a bit more information, it's looking at all the data.

9.4 Model-theoretical semantics

This is the richest, heaviest.

Every non-blue block is next to some blue block.

Distribution says block is like brick, some is like every

Frame semantics: is next to has two arguments, block and block.

Model-theoretic: gives the two interpretations.

Executable semantic parsing: think of sentences as mapping to computer programs. For example: what is the largest city in Europe by population. Semantic parsing gives:

$$\text{argmax}(\text{Cities} \cap \text{contained by}(\text{Europe}, \text{Population}))$$

which executes to Istanbul. You need a database. Understanding the question is not enough.

"Remind me to buy milk", etc.

In executable semantic parsing: A sentence, semantically parsed gives program, executed gives behavior.

(We restrict to a subset of language which is more logical. We'll talk about ways to rescue this later.)

There's a lot of work which comes out of semantic parsing tradition.

How do we go from sentences to logical forms? cities in Europe to Cities∩ContainedBy(Europe). The meaning should be constructed compositionally. This is almost like a parse tree. "In Europe" becomes "ContainedBy(Europe)".

This is your textbook example. Even for this example, there are so many ways to phrase it: cities in Europe, European cities, cities that are in Europe...

There are many solutions; I'll skip ahead to something recent.

We can use deep learning for neural semantic parsing (Jia, Liang 2016). Run the RNN and output pieces of logical form. From linguistic POV this is insane, not modeling compositional structure, but it's not bad.

We learn semantic composition without predefined grammar. We've blown away inductive biases, but can put them back through data recombination.

What's the capital of Germany? CapitalOf(Germany). What countries border France? Borders(France). If we swap France, Germany, the questions make sense. Throw these in the training set too.