MLPR Revision

Mathematical

Multivariate Gaussians

1. Central Limit Theorem: adding together many random outcomes is approximately Gaussian distributed
   - bounded mean and variance
   - constrained values
   - convergence only close to the mean
2. Covariance matrix: $\text{cov}(\mathbf{x}) = E[\mathbf{x}\mathbf{x}^\top] - E[\mathbf{x}]E[\mathbf{x}]^\top$, $\text{cov}[\mathbf{x}]_{ij} = E[x_i x_j] - E[x_i]E[x_j]$.
   - symmetric & (strictly) positive definite.
   - if semi-definite, $|\Sigma|$ will be zero, the pdf will go wrong, or transformed into a Dirac delta function
3. transforming and rotating: (connected with the factorization of Σ)
   - D independent $\mathcal{N}(0,1)$ draws $\mathbf{x}$, $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, I)$.
   - $\mathbf{y} = A\mathbf{x}$, $\mathbf{y} \sim \mathcal{N}(\mathbf{0}, AA^\top)$, $\Sigma = AA^\top$.
   - $\mathbf{z} = \mathbf{y} + \mu$, $\mathbf{z} \sim \mathcal{N}(\mu, AA^\top)$.

   $\mathcal{N}(\mathbf{z}; \mu, \Sigma) = \frac{1}{|\Sigma|^{1/2}(2\pi)^{D/2}} e^{-\frac{1}{2}(\mathbf{z}-\mu)^\top \Sigma^{-1}(\mathbf{z}-\mu)}$.

Evaluating & choosing methods from the zoo of probabilities

1. Baselines, simple possible algorithm run for comparison
2. training/test sets
   - training set: cannot use performance to pick among models
   - validation set: evaluate to select among models (usually variants of the same model, with different choices or hyperparameters) (not on test sets!)
   - test set: only used for reporting error when generalizing to new data

   splitting up data: *error bar* to see if there's enough for test/validation data

   should never look up to test/valid set (seldom cases can look at inputs)
3. What to report:

   test error. (maybe train/validation error (to see underfitting/overfitting))
   - true error: $\text{Generalization error} = E_{p(\mathbf{x},y)}[L(y, f(\mathbf{x}))] = \int L(y, f(\mathbf{x}))\, p(\mathbf{x}, y) d\mathbf{x} d\mathbf{y}$.
   - estimation of the error: $\text{Average test error} = \frac{1}{M}\sum_{m=1}^{M} L(y^{(m)}, f(\mathbf{x}^{(m)}))$, $\quad \mathbf{x}^{(m)}, y^{(m)} \sim p(\mathbf{x}, y)$.

     Monte Carlo estimate (of the generalization error): **unbiased** if $(\mathbf{x}, y)$ (test set item) is really sampled from real distribution.

     distinguish between **error** and **cost**: error shouldn't include regularization term!
   - **error bar** (for the error): $\mu = \bar{x} \pm \hat{\sigma}/\sqrt{N}$, meaning the true mean can be somewhere in between the error bar (66.7% for one standard error, 95% for two standard error), better than just stating $\bar{x}$.
     - unbiased estimator (for true mean μ): $\bar{x}$.
     - standard error $\hat{\sigma}/\sqrt{N}$, $\hat{\sigma}$ can be the standard deviance of any specific term (like mean)
     - suitable for analyzing uncertainty for "every term", not only mean, but also variance (like $\hat{\sigma}$ for variance would need running the whole sampling N times)

   models with low error mean & low standard error are better.
4. K-fold cross-validation
   - for small datasets
   - split into K parts, each model fit K times, with each of the K parts as the validation set, the other K-1 as training set.
   - choose the model with lowest (average) validation error
5. Overfitting
   - not enough basis functions: underfitting
   - **overfitting**:

     typical behavior: extreme weights with large magnitudes, unreasonably close to noisy observations

     (arbitrary model not trustable, generalize badly; data may carry noise themselves)

     solution: start from simple models, average similar features, or discourage unreasonable fits
   - **regularization**: penalizing extreme solutions

     L2 regularization: penalizing sum of square weights in cost function, for linear regression:
     $E_\lambda(\mathbf{w}; \mathbf{y}, \Phi) = (\mathbf{y} - \Phi\mathbf{w})^\top(\mathbf{y} - \Phi\mathbf{w}) + \lambda\mathbf{w}^\top\mathbf{w} = (\tilde{\mathbf{y}} - \tilde{\Phi}\mathbf{w})^\top(\tilde{\mathbf{y}} - \tilde{\Phi}\mathbf{w}) = E(\mathbf{w}; \tilde{\mathbf{y}}, \tilde{\Phi})$, same fit.

     $\tilde{\mathbf{y}} = \begin{bmatrix} \mathbf{y} \\ \mathbf{0}_K \end{bmatrix} \quad \tilde{\Phi} = \begin{bmatrix} \Phi \\ \sqrt{\lambda} I_K \end{bmatrix}$.

- note 1: before regularization, should better normalize (in case of large function values causing weight with large weights drew back due to regularization). or add not regularized bias term.
- note 2: no hard-rule. Validation error larger than training error may indicates overfitting, but it's the generalization performance itself matters.

Regression

matching fitted scalar real-valued functions $f(\mathbf{x})$ from a real-valued input vector $\mathbf{x}$ to real-valued observations or targets y.

Linear regression

1. affine function: $f(\mathbf{x};\mathbf{w},b)=\mathbf{w}^\top\mathbf{x}+b$
   - inputs (features) $\mathbf{x}$, weights $\mathbf{w}$, bias/offset b
2. fitting: training set with N input-output pairs $\{\mathbf{x}^{(n)},y^{(n)}\}_{n=1}^N$
   - $\mathbf{f}=X\mathbf{w}+b$
   - design matrix X: each row gives features for one input vector

     $$\tilde{X}=\begin{bmatrix}\phi(\mathbf{x}^{(1)})^\top & 1\\ \phi(\mathbf{x}^{(2)})^\top & 1\\ \vdots & \vdots\\ \phi(\mathbf{x}^{(N)})^\top & 1\end{bmatrix}=\begin{bmatrix}\phi_1(\mathbf{x}^{(1)}) & \phi_2(\mathbf{x}^{(1)}) & \cdots & \phi_K(\mathbf{x}^{(1)}) & 1\\ \phi_1(\mathbf{x}^{(2)}) & \phi_2(\mathbf{x}^{(2)}) & \cdots & \phi_K(\mathbf{x}^{(2)}) & 1\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ \phi_1(\mathbf{x}^{(N)}) & \phi_2(\mathbf{x}^{(N)}) & \cdots & \phi_K(\mathbf{x}^{(N)}) & 1\end{bmatrix}.$$

     $\phi_k(\mathbf{x})$ is basis function of any kind (a dimension itself, polynomial, RBF, sigmoid... 1 corresponds to bias b in $\tilde{\mathbf{w}}$)

     for each data point, $f(\mathbf{x})=\mathbf{w}^\top\phi(\mathbf{x})$.
     - RBF (radial basis function): $\exp\left(-(\mathbf{x}-\mathbf{c})^\top(\mathbf{x}-\mathbf{c})/h^2\right)$, bell-shaped, center $\mathbf{c}$, bandwidth h; change the function close to a reference point
     - logistic-sigmoid function: $\sigma(\mathbf{v}^\top\mathbf{x}+b)=\frac{1}{1+\exp(-(\mathbf{v}^\top\mathbf{x}+b))}$, s-shaped, steepness $\mathbf{v}$, offset b; change the function in half of the space
   - optimizing: least-square fitting
     - residual: $\mathbf{r}=\mathbf{y}-\mathbf{f}$, cost (square error): $\sum_{n=1}^N[y^{(n)}-f(\mathbf{x}^{(n)};\mathbf{w},b)]^2=(\mathbf{y}-\mathbf{f})^\top(\mathbf{y}-\mathbf{f})$.
     - gradients: $\nabla_{\mathbf{w}}\mathbf{r}^\top\mathbf{r}=\mathbf{0}$, normal equation solution (closed form): $\mathbf{w}=(X^\top X)^{-1}X^\top\mathbf{y}$.

       note: if $N<D$, then there's more than one solution for $\mathbf{w}$, and $X^\top X$ will not be full rank to be invertible.

Bayesian linear regression

the point of Bayesian methods is it can represent a whole probability distribution over possible outputs for each input location.

Given the probabilistic model, can use Bayesian reasoning to make predictions. able to specify the uncertainty about parameters and predictions.

probabilistic model for regression: (starting with prior Gaussian + likelihood Gaussian)

- $p(y|\mathbf{x},\mathbf{w})=\mathcal{N}(y;f(\mathbf{x};\mathbf{w}),\sigma_y^2)$, $f(\mathbf{x};\mathbf{w})=\mathbf{w}^\top\phi(\mathbf{x})$.
- prior: $p(\mathbf{w})=\mathcal{N}(\mathbf{w};\mathbf{w}_0,V_0)$.
- likelihood: $p(\mathbf{y}|\mathbf{w},X)=\Pi_n\mathcal{N}(y^{(n)};\mathbf{w}^T\phi(\mathbf{x}^{(n)}),\sigma_y^2)$.
- posterior: $p(\mathbf{w}|D)\propto p(\mathbf{w})p(\mathbf{y}|\mathbf{w},X)$ (Bayes rule). Given conjugate prior, $p(\mathbf{w}|D)=\mathcal{N}(\mathbf{w};\mathbf{w}_N,V_N)$, still Gaussian.
  - $V_N=\sigma_y^2(\sigma_y^2V_0^{-1}+\Phi^\top\Phi)^{-1}$, $\mathbf{w}_N=V_NV_0^{-1}\mathbf{w_0}+\frac{1}{\sigma_y^2}V_N\Phi^\top\mathbf{y}$.
- posterior prediction distribution:
  - generic method: $p(y|\mathbf{x},D)=\int p(y,\mathbf{w}|\mathbf{x},D)d\mathbf{w}=\int p(y|\mathbf{x},\mathbf{w})p(\mathbf{w}|D)d\mathbf{w}$. i.e. the new expectation of y given $\mathbf{x}$ on the posterior $\mathbf{w}$ distribution.
  - simplified method given predictive distribution is also Gaussian (only need mean & variance):

    transforming the two successive gaussians into sum of two gaussians:
    $\mathbf{y}=f(\mathbf{x})+\nu=\mathbf{x}^\top\mathbf{w}+\nu\quad f\sim\mathcal{N}(\mathbf{x}^\top\mathbf{w}_N,\mathbf{x}^\top V_N\mathbf{x})\quad\nu\sim\mathcal{N}(0,\sigma_y^2)$.

  $\mathbf{y}(\text{predict})\sim\mathcal{N}(\mathbf{x}^\top\mathbf{w}_N,\mathbf{x}^\top V_N\mathbf{x}+\sigma_y^2)$.
- decision making: finding the optimal guess based on the **loss function** and the **posterior distribution**.

  expected loss: $c=E_{p(y|\mathbf{x},D)}[L(y,\hat{y})]=\int L(y,\hat{y})p(y|\mathbf{x},D)dy$.

  optimal guess of $\hat{y}$: $\frac{\partial c}{\partial\hat{y}}=0$. In the above setting, $\hat{y}=E_{p(y|\mathbf{x},D)}[y]$, i.e. mean of posterior prediction.
- model choice:
  - marginal likelihoods (for scoring different models): $p(\mathbf{y}|X,\mathcal{M})=\int p(\mathbf{y}|X,\mathbf{w},\mathcal{M})p(\mathbf{w}|\mathcal{M})d\mathbf{w}$, favoring simple models for well-behaved data.
  - hyperparameters: maximize their marginal likelihood $p(\mathbf{y}|X,\sigma_w,\sigma_y)$.

  note: problems caused by not suitable model complexity:
  - too simple: posterior will be sharply peaked around the least bad fit, because normalized, we can still be very confident, but actually bad, with strong correlated residuals.

- too complicated: most weights will not change after few observations, predictions nearly same as under prior, learn very slow.

necessity: under this setting, the results corresponds to using a L2 regularized model (can be proved with $\mathbf{w}_N$), but:

- warn us to manually intervene, or gather relevant data;
- we do not only need point estimate generally;
- in general case, loss functions are asymmetric.

Gaussian Processes (kernelized BLR)

represent functions as infinite-dimensional vectors, and define a multi-variate gaussian on it. Can model complex functions, represent uncertainty of functions

- probabilistic model for gaussian process:
  - $y_i \sim \mathcal{N}(f_i, \sigma_y^2),\ f_i = f(\mathbf{x}^{(i)}), f \sim \mathcal{GP}$
    - function values at every input locations, $\tilde{\mathbf{f}}$. at training locations (we observe), $\mathbf{f}$; at testing locations (we predict), $\mathbf{f}_*$.
    - training locations $X = \{\mathbf{x}^{(i)}\}$; testing locations $X_* = \{\mathbf{x}^{(*,i)}\}$.
  - prior $p(\mathbf{f}) = \mathcal{N}(\mathbf{f}; \mathbf{0}, K)$, $K_{ij} = \text{cov}[\tilde{f}_i, \tilde{f}_j] = k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$.
    - kernel functions:
      - kernel trick:

        Bayesian linear regression kernel: $k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \sigma_w^2 \mathbf{x}^{(i)\top}\mathbf{x}^{(j)} + \sigma_b^2$, covariance matrix is not full rank, GP can be solved in finite observations. $k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \sigma_w^2 \boldsymbol{\phi}(\mathbf{x}^{(i)})^\top \boldsymbol{\phi}(\mathbf{x}^{(j)}) + \sigma_b^2$ can increase amount of features. As long as observations are always less than basis functions, the model can be updated infinitely. **Replacing an inner product of features with a simple kernel function, corresponding to a large or infinite set of basis functions** is known as the kernel trick. All the positive definite kernel functions correspond to infinite feature space.
      - e.g. RBF, squared-exponential, exponentiated-quadratic, ...
      - GP also known as *kernelized* Bayesian linear regression.
      - kernels can be combined in various ways (linear combination of positive definite kernels is also positive definite)
      - kernels can give functions with qualitatively different properties, including smoothness (oscillation), periodicity, linear correlation...
      - kernels usually have parameters (can be tuned).
  - joint distribution of observations $\mathbf{y}$ and test location values $\mathbf{f}_*$: (marginal of the prior over the whole function)
    $p\left(\begin{bmatrix}\mathbf{y}\\ \mathbf{f}_*\end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix}\mathbf{y}\\ \mathbf{f}_*\end{bmatrix};\ \mathbf{0},\ \begin{bmatrix}K(X,X)+\sigma_y^2 I & K(X,X_*)\\ K(X_*,X) & K(X_*,X_*)\end{bmatrix}\right)$, $K(X,Y)_{i,j} = k(\mathbf{x}^{(i)}, \mathbf{y}^{(j)})$.
  - posterior: $p(\mathbf{f}_*|\mathbf{y}) \sim \mathcal{N}(\bar{\mathbf{f}}_*, \text{cov}(\mathbf{f}_*))$, (marginal of the joint distribution over the whole function, again a gaussian process)
    - $\text{mean}, \bar{\mathbf{f}}_* = K(X_*,X)(K(X,X)+\sigma_y^2 I)^{-1}\mathbf{y}$.
    - $\text{cov}(\mathbf{f}_*) = K(X_*,X_*) - K(X_*,X)(K(X,X)+\sigma_y^2 I)^{-1}K(X,X_*)$.
  - hyperparameters
    - take RBF $k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \sigma_f^2 \exp\left(-\frac{1}{2}\Sigma_{d=1}^D (x_d^{(i)} - x_d^{(j)})^2 / l_d^2\right)$ as an example.

      lengthscale/bandwidth l: larger, smoother; smaller, more oscillative.

      function/signal variance σ_f^2: larger, oscillate with greater amplitude, sensitive to x's change.
    - learning hyperparameters: maximum likelihood (maximize $p(\mathbf{y}|X,\theta)$ before training); grid search; gradient-based optimizers. need to prevent overfitting meantime:

      > sometimes regularize log noise variance $\log \sigma_y^2$.
      >
      > or use fully Bayesian approach to treat hyperparameters as distributions also, then do the integration.
  - complexity: scale poorly on large datasets.
    - according to the upper equation, mostly $O(N^3)$ for the $(K(X,X)+\sigma_y^2 I)^{-1}$ term (with Cholesky).
    - computing kernel matrix $O(DN^2)$, and uses $O(N^2)$ memory.
  - conclusion: the things we are updating are the **GP itself**.
    - prior: over training (as well as testing), see the "joint distribution".
    - observations: over training locations.
    - posterior: over testing locations.
    - cannot represent all functions with strictly positive definite kernel, such as monotonic ones.

Classification

$\{\mathbf{x}^{(n)}, y^{(n)}\}$: $\mathbf{x}^{(n)}$ same as regression, labels y belongs to a discrete set of categories. more like "splitting the input vector space with decision boundaries".

question: why not regression? approaches for classification will usually generalize better.

- least square objective does not return the most useful function for constructing a classification rule.
- fitted functions will usually extend outside $f \in [0,1]$, hard to take the probabilistic interpretation seriously.

One-hot encoding

for discrete input features / classes, encoding them as integers is not ok (would assume book feature vector is the *average* of vector for a crime novel and a fantasy novel will be about sport, not reasonable!)

- one-hot encoding (one-of-K encoding)

  using binary vector, like $\mathbf{y} = [0\ 1\ 0\ 0\ \ldots]^\top$.
- multiclass classification: each input has only one label
- multilabel classification: each input has more than one label

Generative models

fitting a simple model (distribution) for each class, modeling the generating process of the data in it.

- simple gaussian generative classifier: (could use other distribution as well)
  - likelihood term $P(\mathbf{x}|y=k) = \mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k)$: fit mean & covariance of Gaussian k to the mean & covariance of the set of feature vectors with label k;
  - prior term $P(y=k)$: the proportion of data with label k;
  - Bayes' rule + normalization, $P(y=k|\mathbf{x})$.

  note: sensitive to how the data were sampled in the particular application.
- Naive Bayes: a special case of generative classifiers with Naive Bayes assumption.
  - Naive Bayes assumption: assumption that features are independent given the class.
  - examples:
    - gaussians: $P(\mathbf{x}|y=k) = \mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k)$, when Σ_k is diagonal, features are independent to each other, $P(\mathbf{x}|y=k) = \Pi_d \mathcal{N}(x_d; \mu_{d,k}, \sigma^2_{d,k})$.
    - binary features: $P(\mathbf{x}|y=k, \theta) = \Pi_d P(x_d|y=k, \theta) = \Pi_d \theta_{d,k}^{x_d}(1-\theta_{d,k})^{1-x_d}$.

  note: distinguish between "Bayesian methods" & "Naive Bayes classifiers"!

Discriminative models

directly model (separate) the classes/labels, instead of focusing the generative process.

- logistic regression
  - $f(\mathbf{x}; \mathbf{w}) = \sigma(\mathbf{w}^\top \mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^\top \mathbf{x}}}$, forcing the output to lie in $[0, 1]$ with sigmoid function, to match the probability interpretation.
  - optimizing: **maximum likelihood fitting**, (given the interpretation of the function as a *probability*)
    - loss (negative log likelihood, NLL): $L(\mathbf{w}) = \Pi_{n=1}^N P(y^{(n)}|\mathbf{x}^{(n)}, \mathbf{w})$, $\mathrm{NLL} = -\log L(\mathbf{w}) == -\Sigma_{n=1}^N \log\left(\sigma(z^{(n)}\mathbf{w}^\top \mathbf{x}^{(n)})\right) \quad z^{(n)} = (2y^{(n)} - 1) \in \{-1, +1\}$. can have regularizer added.
    - gradients: $\nabla_{\mathbf{w}} \mathrm{NLL} = -\Sigma_{n=1}^N (1-\sigma_n) z^{(n)} \mathbf{x}^{(n)} = \mathbf{0}$.
- robust logistic regression (include a label corruption)
  - $P(y=1|\mathbf{x}, \mathbf{w}, m) = \begin{cases} \sigma(\mathbf{w}^\top \mathbf{x}) & m=1 \\ \frac{1}{2} & m=0. \end{cases}$, $P(m|\epsilon) = \mathrm{Bernoulli}(m;\, 1-\epsilon) = \begin{cases} 1-\epsilon & m=1 \\ \epsilon & m=0 \end{cases}$.
  - $P(y=1|\mathbf{x}, \mathbf{w}, \epsilon) = \Sigma_{m\in\{0,1\}} P(y=1|\mathbf{x}, \mathbf{w}, m) P(m|\epsilon) = (1-\epsilon)\sigma(\mathbf{w}^\top \mathbf{x}) + \epsilon\frac{1}{2}$,
  - optimizing: same
    - gradients: $\nabla_{\mathbf{w}} \log P(z^{(n)}|\mathbf{x}^{(n)}, \mathbf{w}) = \frac{1}{1+\frac{1}{2}\frac{\epsilon}{1-\epsilon}\frac{1}{\sigma_n}} \nabla_{\mathbf{w}} \log \sigma_n$, not convex anymore! (recover original model at $\epsilon = 0$)
  - setting ϵ: by hand, grid search, or gradient-based optimization with $\mathbf{w}$ (remember reparameterization).
- softmax regression, generalization to "multiclass classification"
  - $P(y_k = 1|\mathbf{x}, W) = f_k(\mathbf{x}; W) \quad f_k = \frac{s_k}{\Sigma_{k'} s_{k'}} = \frac{e^{(\mathbf{w}^{(k)})^\top \mathbf{x}}}{\Sigma_{k'} e^{(\mathbf{w}^{(k')})^\top \mathbf{x}}}$. i.e. $\mathbf{f}(\mathbf{x}; W) = \mathrm{softmax}(W\mathbf{x}) \quad W = \{\mathbf{w}^{(k)}\}_{k=1}^K$.
  - when $K = 2$, can be simplified to logistic regression.
  - optimizing: least squares or maximum likelihood ($\log 0$ not preferred) both consistent estimation of the parameters.
    - log-probability: $\log P(y_c = 1|\mathbf{x}, W)$.

      $(\mathbf{w}^{(1)} - \mathbf{w}^{(2)})$ turn into one. (if fitting two, can add regularizer or set one to 0 to make the solution unique)
    - better use stochastic gradient ascent or batch gradient ascent (iterative) to optimize it: $\nabla_{\mathbf{w}^{(k)}} \log f_c = (y_k - f_k)\mathbf{x}$.
- Bayesian logistic regression

  probabilistic model for logistic regression:
  - $f(\mathbf{x}; \mathbf{w}) = \sigma(\mathbf{w}^\top \mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^\top \mathbf{x}}}$.
  - likelihood: $P(\mathcal{D}|\mathbf{w}) = P(\mathbf{y}|X, \mathbf{w}) = \Pi_n \sigma(z^{(n)}\mathbf{w}^\top \mathbf{x}^{(n)})$.
  - prior: $p(\mathbf{w}) = \mathcal{N}(\mathbf{0}, \sigma_w^2 I)$.
  - posterior: $p(\mathbf{w}|D) = \frac{P(\mathcal{D}|\mathbf{w})p(\mathbf{w})}{P(\mathcal{D})} \propto P(\mathcal{D}|\mathbf{w})p(\mathbf{w}) \quad P(\mathcal{D}) = \int P(\mathcal{D}|\mathbf{w}) d\mathbf{w}$. (can be approximated)
  - **Laplace approximation**: approximation method for the normalization:

starting with posterior, as a Gaussian $p(\mathbf{w}|\mathcal{D}) \approx \mathcal{N}(\mathbf{w}; \mathbf{w}^*, H^{-1})$.

energy function: $E(\mathbf{w}) = -\log p(\mathbf{w}, \mathcal{D})$.

- **MAP** (maximize a posterior) estimate (for the "most probable" weights): if with a Gaussian prior, equivalent to MLE method with an L2 regularized estimate:

  $\mathbf{w}^* = \text{argmin}_{\mathbf{w}} E(\mathbf{w})$, (coincide with $\text{argmax}_{\mathbf{w}} \left[\log P(\mathcal{D}|\mathbf{w}) - \frac{1}{2\sigma_w^2}\mathbf{w}^\top\mathbf{w}\right]$).

- **curvature**:

  $H = \frac{\partial^2 E(w)}{\partial w^2}|_{w=w^*}$, $H_{ij} = \frac{\partial^2 E(\mathbf{w})}{\partial w_i \partial w_j}|_{\mathbf{w}=\mathbf{w}^*}$.

normalizer based on the approximated Gaussian posterior: $P(\mathcal{D}) \approx p(\mathbf{w}^*, \mathcal{D})|2\pi H^{-1}|^{1/2}$.

note: posterior less uncertain, more tightly peaked, more like gaussian.

drawbacks: doesn't always work well, asymmetric, multi-modal, ...

- **Variational methods**

  fit a target distribution (posterior) by defining *an optimization problem*:

  - a family of possible distributions $q(\mathbf{w}, \alpha)$. (here consider the Gaussian case, $\alpha = \{\mathbf{m}, V\}$)
  - a variational cost function, which describes the discrepancy between $q(\mathbf{w}; \alpha)$ and the target distribution (posterior)

  optimize the variational parameters α.

  KL divergence: $D_{\text{KL}}(p||q) = \int p(\mathbf{z}) \log \frac{p(\mathbf{z})}{q(\mathbf{z})} d\mathbf{z}$.

  - non-negative, only zero when two distributions are identical
  - not symmetric!

  minimizing: -ELBO + loglikelihood, $D_{KL}(q||p) \geq 0 \quad \log P(\mathcal{D}) \geq -J(q)$, maximizing ELBO. Dominated by SGD.

  Comparison between two approximation method:

  - Laplace
    - straight forward to apply, but not flexible
    - is a local approximation, not suitable to multi-modal ..
  - Variational methods
    - swift, can be applied to different models
    - traditionally harder to apply, high complexity optimization
    - sensitive to the distribution family chosen

- posterior prediction distribution: $p(y|\mathbf{x}, D) = \int p(y, \mathbf{w}|\mathbf{x}, D) d\mathbf{w} = \int p(y|\mathbf{x}, \mathbf{w}) p(\mathbf{w}|D) d\mathbf{w}$. (can't evaluate in closed form)

  approximation: $p(y = 1|\mathbf{x}, \mathcal{D}) \approx \int \sigma(\mathbf{w}^\top\mathbf{x}) \mathcal{N}(\mathbf{w}; \mathbf{w}^*, H^{-1}) d\mathbf{w} = E_{\mathcal{N}(\mathbf{w};\mathbf{w}^*,H^{-1})}[\sigma(\mathbf{w}^\top\mathbf{x})] = E_{\mathcal{N}(a;\mathbf{w}^{*\top}\mathbf{x},\mathbf{x}^\top H^{-1}\mathbf{x})}[\sigma(a)]$

  .

  - **Monte Carlo approximations** (no need of gaussian constraints):

    $E_{p(\mathbf{w}|\mathcal{D})}[\sigma(\mathbf{w}^\top\mathbf{x})] \approx \frac{1}{S}\Sigma_{s=1}^S \sigma({\mathbf{w}^{(s)}}^\top\mathbf{x}) \quad \mathbf{w}^{(s)} \sim p(\mathbf{w}|\mathcal{D})$.

  - **importance sampling** (no need of gaussian constraints):

    - prediction: (from posterior)

      $p(y = 1|\mathbf{x}, \mathcal{D}) = \int \sigma(\mathbf{w}^\top\mathbf{x}) p(\mathbf{w}|\mathcal{D}) \frac{q(\mathbf{w})}{q(\mathbf{w})} d\mathbf{w} = E_{q(\mathbf{w})}[\sigma(\mathbf{w}^\top\mathbf{x}) \frac{p(\mathbf{w}|\mathcal{D})}{q(\mathbf{w})}] \approx \frac{1}{S}\Sigma_{s=1}^S \sigma({\mathbf{w}^{(s)}}^\top\mathbf{x}) \frac{p(\mathbf{w}^{(s)}|\mathcal{D})}{q(\mathbf{w}^{(s)})} \quad \mathbf{w}^{(s)} \sim q(\mathbf{w})$

      , importance weight: $r^{(s)} = \frac{p(\mathbf{w}^{(s)}|\mathcal{D})}{q(\mathbf{w}^{(s)})}$.

    - denominator $P(\mathcal{D})$: (from prior)

      $P(\mathcal{D}) = \int P(\mathcal{D}|\mathbf{w}) p(\mathbf{w}) \frac{q(\mathbf{w})}{q(\mathbf{w})} d\mathbf{w} = E_{q(\mathbf{w})}[P(\mathcal{D}|\mathbf{w}^{(s)}) \frac{p(\mathbf{w})}{q(\mathbf{w})}] \approx \frac{1}{S}\Sigma_{s=1}^S P(\mathcal{D}|\mathbf{w}^{(s)}) \frac{p(\mathbf{w}^{(s)})}{q(\mathbf{w}^{(s)})} \quad \mathbf{w}^{(s)} \sim q(\mathbf{w})$.

    - continue separate the posterior by substituting the denominator:

      set $\tilde{r}^{(s)} = \frac{P(\mathcal{D}|\mathbf{w}^{(s)})p(\mathbf{w}^{(s)})}{q(\mathbf{w}^{(s)})}$, thus $P(\mathcal{D}) = \frac{1}{S}\Sigma_{s=1}^S \tilde{r}^{(s)}$.

      set $r^{(s)} = \frac{\tilde{r}^{(s)}}{\frac{1}{S}\Sigma_{s=1}^S \tilde{r}^{(s)}}$, thus

      $p(y = 1|\mathbf{x}, \mathcal{D}) = \frac{1}{S}\Sigma_{s=1}^S \sigma({\mathbf{w}^{(s)}}^\top\mathbf{x}) \frac{P(\mathcal{D}|\mathbf{w}^{(s)})p(\mathbf{w}^{(s)})/P(\mathcal{D})}{q(\mathbf{w}^{(s)})} = \frac{1}{S}\Sigma_{s=1}^S \sigma({\mathbf{w}^{(s)}}^\top\mathbf{x}) \frac{\tilde{r}^{(s)}}{\frac{1}{S}\Sigma_{s=1}^S \tilde{r}^{(s)}} = \frac{1}{S}\Sigma_{s=1}^S \sigma({\mathbf{w}^{(s)}}^\top\mathbf{x}) r^{(s)} \quad \mathbf{w}^{(s)} \sim q$

      , then there's no need of calculating the non-closed-form posterior.

    - note: the choose of $q(\mathbf{w})$. importance sampling can work far beyond logistic regression.

Optimizing

Convexity

$C(\alpha\mathbf{w} + (1-\alpha)\mathbf{w}') \ \leq \ \alpha C(\mathbf{w}) + (1-\alpha)C(\mathbf{w}'), \quad \text{for any } \mathbf{w}, \mathbf{w}', 0 \leq \alpha \leq 1.$

strictly convex, the function only have one unique minimum; otherwise, connected convex set.

- logistic regression, loss is a convex function of $\mathbf{w}$;
- logistic regression, square loss is not a convex function of $\mathbf{w}$.

gradient-based methods

- closed form solution (see each method)
- Iterative methods:

  generic method, iteratively improve initial guess of model parameters with a cost function and its gradient vectors.

  - **stochastic gradient descent (SGD)**: one data point, one update. Fast, but noisy.

    each example gives a crude one-example Monte Carlo approximation of the gradient sum.

  - traditional batch steepest descent: whole dataset, one update. Smooth, but slow.
- about gradients
  - finite differences: for checking the correctness of your gradients;
  - automatic differentiation: the number of operations for computing derivatives should be *similar* to one function evaluation.

Reparameterization

transforming constrained-range variables to unconstrained-range ones.

- positive data (or to avoid numerical overflow, make maths convenient), take $\log()$
- $(0,1)$ data, use $\text{logit}()$
- $\exp()$ to make numbers positive
- sigmoid $\sigma()$ to make numbers lie in $(0,1)$

Neural Networks

- definition: $\mathbf{h}^{(l)} = g^{(l)}(W^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)})$.
  - unit: logistic units (activation function), hidden units, visible units (position in network)
  - layer: hidden layer, visible layer
  - non-linear function, activation (input of non-linear function) (notice the gradient of the function!)
- some architectures: skip connections & residual layer (RNN, LSTM, GRU), embedding vectors, bags and sequences of items (pooling, attention), sequence2sequence (Transformer, ConvNet)
- optimization:
  - initialization: random, considering input sizes, specific application.
  - local optima: usually use iterative gradient-based optimizer, non-convex wrt all parameters (easily showed by switching weight matrix's rows or columns)
  - regularization
    - early stopping: periodically monitor performance on a validation set, stop training if not getting better
    - regularized cost function
    - adding gaussian noise to inputs/layers
    - drop-out
  - backpropagation
    - chain rules: $\frac{\partial f}{\partial u_i} = \Sigma_{d=1}^{D} \frac{\partial f}{\partial x_d} \frac{\partial x_d}{\partial u_i}$.
    - forward-mode differentiation: derivative of every variable in the DAG wrt a scalar **input**, $\dot{\theta} = \frac{\partial \theta}{\partial u}$.

      can be done alongside the function computation, never much more expensive than the original function, only a constant factor of extra memory required.
    - reverse-mode differentiation: derivative of a scalar **output** variable wrt to every other variables in the DAG, $\bar{\theta} = \frac{\partial z}{\partial \theta}$.

      traverse the graph in reverse after computing the function, using stored values of intermediate quantities. **similar complexity to function computation**, use more memory than forward mode.
    - array-based operations: $C = AB \quad \bar{A} = \bar{C}B^\top$, $c = \mathbf{x}^\top A\mathbf{x} \quad \bar{\mathbf{x}} = 2\bar{c}A\mathbf{x}$. match the dimensions to remember!

Learning low-dimensional representations

- Autoencoders:
  - a neural network representing a vector-valued function, when fitted well, approximately returns the input, $\mathbf{f}(\mathbf{x}) \approx \mathbf{x}$.
  - dimensionality reduction: form "bottleneck" to create a lossy compressor.

    $\mathbf{h} = g^{(1)}(W^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$.

    $\mathbf{f} = g^{(2)}(W^{(2)}\mathbf{h} + \mathbf{b}^{(2)})$.

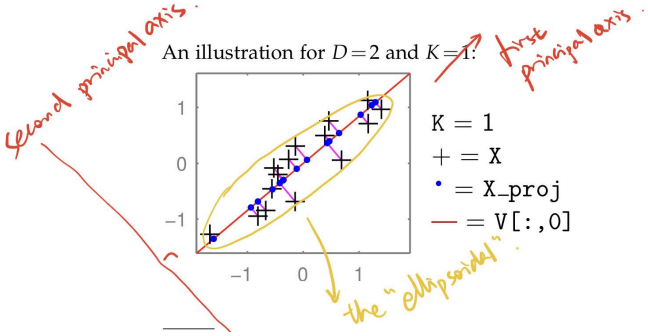
    encode, decode, reconstruct. $\mathbf{h}$ contains most of the information of the input.
  - denoising and sparse AE: form high dimensional hidden layer, give features easier to separate.
    - denoising AE: introduce noises by randomly setting some of the features in input to 0, learn more robust representation
    - sparse AE: only allow a small fraction of hidden units activated.
  - data processing inequality: at test time, transforming an individual input with an AE can't add information about it.
- Principal Components Analysis (PCA), a linear autoencoder

$\mathbf{h} = V^\top(\mathbf{x} - \bar{\mathbf{x}})$

$\mathbf{f} = V\mathbf{h} + \bar{\mathbf{x}}$

- features:
  - shared V, no non-linear activations
  - solutions for different hidden layer size K is *nested*. (increasing K, new hidden layer includes hidden layer solutions with smaller size)
  - solution unique
  - parameters can be found with:

    covariance of points $\Sigma = Q\Lambda Q^\top$ describe an "ellipsoidal", eigen vectors point along its axes, longest axes corresponding to the largest eigen value. set columns of V to the K eigen vectors ($Q[:, :k]$) associated with the largest K eigen values.



- note: pre-processing matters, because of error can scale with data units. Given features with similar importance, common to standardize all features, with unit std.
- **aim**: does not always work well.
  - get a sensible lower-dimension representation, used in downstream tasks;
  - interpret each principal component, understand data.
- **advantage**:
  - unsupervise model, no need of labels
  - variance maximization (of data in new axes)
  - mathematically solvable, simple
  - nested hidden representation solution, size controllable
- **disadvantage**
  - can only capture linear relationships
  - sensitive to outliers (in covariance calculation)
  - limited to gaussians
  - features not necessarily interpretable

- probabilistic PCA

  $\mathbf{x} \sim \mathcal{N}(\mu, WW^\top + \sigma^2 I)$.

- truncated Singular Value Decomposition (SVD)

  $X \approx USV^\top$, a more general decomposition method for not square matrix.

  U is a $(N \times K)$ orthogonal matrix, S is a $(K \times K)$ diagonal matrix, $V^\top$ is a $(K \times D)$ orthogonal matrix.

  - solution unique
  - rows of U give K-dimensional embedding of rows of X, gives eigen vectors of $XX^\top$;
  - rows of V give K-dimensional embedding of columns of X, gives eigen vectors of $X^\top X$.
  - When $K < \min(N, D)$, $USV^\top = X_k$ is the (rank K) best low-rank approximation of a matrix, as measured by square error. (actually "truncated" the first K important rank of the original matrix)
  - When $N = K$, and data centered with mean 0 (covariance matrix symmetric), then SVD turns to eigen decomposition (PCA), then $U = V$ gives eigen vectors of X^2, i.e. eigen vectors of X; S includes the former K eigen values.