Introduction:

Bias vs variance: a perfect decision $y = f(x)$; $\hat{y}$ train label, Bias - dis between average model and theoretical best. Variance - variability with different traing samples.

Simple model: high Bias, low variance. Hyper-para: # of samples, feature dim d. Cross-validation: partition data into N subset, every time train on $N-1$,

Complex model: low Bias, high variance. regularization: ↓ some parameter. validate on N_i, rotate all N options, choose best configuration, Retrain

make use of small date set, but expensive.

$$MSE = E[(\hat{y}_0 - f(x_0))^2] = E[(\hat{y}_0 - E[\hat{y}_0] + E[\hat{y}_0] - f(x_0))^2]$$
$$= E[(\hat{y}_0 - E[\hat{y}_0])^2] + E[(E[\hat{y}_0] - f(x_0))^2] + 2E[(\hat{y}_0 - E[\hat{y}_0])(E[\hat{y}_0] - f(x_0))]$$
$$= E[(\hat{y}_0 - E[\hat{y}_0])^2] + (E[\hat{y}_0] - f(x_0))^2 \quad \text{for } E[\hat{y}_0 - E[\hat{y}_0]] = 0.$$
$$= Var(\hat{y}_0) + Bias^2(\hat{y}_0)$$

Regression:

linear if $f(x, w) = w^T x$, extend to non-linear $f(x; w) = w^T h(x)$, $h(x)$ non-linear.

OLS: total error $= \sum_i (y_i - \hat{y}_i)^2 = \sum_i (y_i - \sum_k w_k x_{ik})^2$, $\hat{y}$ prediction.

Matrix notation: $(Hw - Y)^T (Hw - Y)$, H: $N \times k$, w: $k \times 1$, Y: $N \times 1$

close form solution: $L = (Hw - Y)^T (Hw - Y)$, L scalar, target $\frac{\partial L}{\partial w} = 0$

$tr(L) = L \Rightarrow \frac{\partial L}{\partial w} = \frac{\partial L}{\partial w} tr((Hw - Y)^T (Hw - Y)) = \frac{\partial L}{\partial w} tr(w^T H^T H w - w^T H^T Y - Y^T H w + Y^T Y)$

$\frac{\partial}{\partial w} tr(w^T H^T H w) = (H^T H w) + (w^T H^T H)^T$

$\frac{\partial}{\partial w} tr(w^T H^T Y) = \frac{\partial}{\partial w} tr(Y^T H w) = (Y^T H)^T$

$\frac{\partial}{\partial w} tr(Y^T Y) = 0$

$\frac{\partial L}{\partial w} = H^T H w + H^T H w - 2H^T Y = 0$

$H^T H w = H^T Y \Rightarrow w = (H^T H)^{-1} H^T Y$.

property: $tr(AB) = tr(BA)$.

$tr(ABC) = tr(CAB) \cdots$ 循环

$\partial\, tr(XB) = \partial\, tr(BX) = B^T$

$\partial\, tr(X^T B) = B$

$tr(A) = tr(A^T)$

$\partial\, tr(AXBX^T) = AXB + A^T X B^T$

$\partial\, tr(A^T X B^T) = AB$

Regularization: large parameter cause overfit.

Ridge: $\hat{w}_{ridge} = \arg\min_w \sum_i (y_i - (w_0 + \sum_{k=1}^K w_k h_k(x_i)))^2 + \lambda \sum_{k=1}^K w_k^2 = \arg\min_w (Hw - Y)^T (Hw - Y) + \lambda w^T I w$

$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial w} tr(w^T H^T H w - w^T H^T Y - Y^T H w + Y^T Y + \lambda w^T I w)$

$= 2H^T H w - 2H^T Y + 2\lambda I w = 0$.

$(H^T H + \lambda I) w = H^T Y$

$w = (H^T H + \lambda I)^{-1} H^T Y$.

LASSO: $\hat{w}_{lasso} = \dots + \lambda \sum_{k=1}^K |w_k|$

push more weigh to 0.

allow feature selection.

no differentiable and close form.

CNN:

sigmoid: $f(u) = \frac{1}{1 + e^{-u}}$, $\frac{df(u)}{du} = f(1 - f)$ (MLP)

tanh: $\frac{e^x - e^{-x}}{e^x + e^{-x}}$ (CNN)

diff MLP and CNN:

use convolution (weight sharing), pooling (subsampling), no feature extraction as pre-processing, use tanh.

data argmentation: Flip, Random crop, color distortion, Rotation.

padding: 边缘扩展, output size $= \frac{\text{input} + 2\text{pad} - \text{kernel}}{\text{stride}} + 1$

stride: step.

pooling: max, average, global.

sigmoid: 导数 < 1/4, vanishing gradient, use Relu as activation, 导数为 1

softmax: $\frac{e^{s_i}}{\sum_j^c e^{s_j}}$

Cross-entropy loss: $-\sum_i^c t_i \log(f(s)_i)$

drop-out

Receptive Field: region of input space affects a particular unit of the network.

Dilated conv: enlarge the receptive field. dilation = 2, 隔1作为 input

1x1 conv: cross channel information aggregation, feature projection, dim reduction

Depthwise conv: each channel has its own kernel, parameter reduced, but no cross channel information (combine w/ conv)

group conv: split channels into group. still need 1x1 conv. (ResNext)

DPM: sliding window. binary classification, HOG as feature. Linear SVMs.

R-CNN: extract region (selective search), get feature (CNN), Classify (SVM). Mask R-CNN: mask prediction, ROI align.

dis: 候选框重复计算，候选框缩放到固定尺寸，畸变，多阶段训练，分步，內存占用大

Fast R-CNN: One-stage train, no svm. no warp/resize, fast. RPN.

but region proposals are not efficient.

ROI pooling: quantization two times.

ROI align: sampling, Bilinear interpolation, pooling.

Attention: equip a neural network with ability to focus on a subset of input. $a = f_\phi(x)$, $g = a \odot z$.

spatial attention: 在 HxW 平面上学习权重, 同一位置每个通道相同.

channel attention: 在深度 C 上学习权重 (SE block).

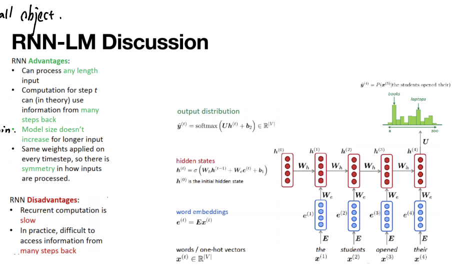
way to improve performance: extra training date; augmentation strategies, change backbone architecture. modify classification and regression method, proper loss function.

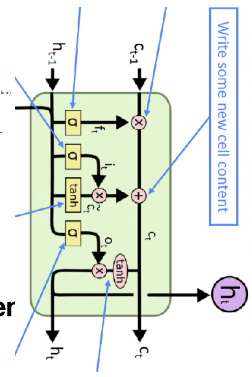
Application: image classifaction, detection and tracking. Pose estimation, segmentation. image reconstruction. 3D and localization,

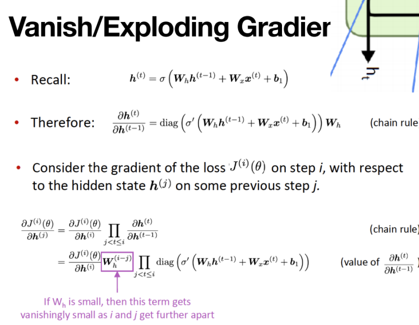
YOLO: use stride to decrease size, no pooling and full-connect, fast, simple algorithm, larger scale, bad at small object.

SSD: multi box and multi scale.

NMS: algorithm to choose the best bounding box.







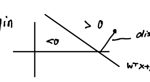
SVM:

Optimal margin classifier

: $h_\theta(x) = g(\theta^T x)$, $\theta^T x \ge 0$, Predict "1" $\Rightarrow$ if $y^{(i)} = 1$, we hope $\theta^T x \gg 0$

geometric margin (data linearly seperate)

label $y \in \{-1, 1\}$, h as output, $g(z) = \begin{cases} 1 & \text{if } z \ge 0 \\ -1 & \text{otherwise} \end{cases}$, $h_{w,b}(x) = g(w^T x + b)$ no constraint $x_0 = 1$



$\hat{r}$: functional margin of a hyper plane by (w, b) w.r.t. train sample is $\hat{r}^{(i)} = y^{(i)} (w^T x^{(i)} + b)$, hope $\hat{r}^{(i)} \gg 0$

normalize: $(w, b) \Rightarrow (\frac{w}{\|w\|}, \frac{b}{\|b\|})$ to avoid cheating.

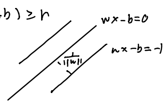
r: geometric margin of hyper plane (w, b) wrt $(x^{(i)}, y^{(i)})$ is $r^{(i)} = \frac{(w^T x^{(i)} + b) \cdot y^{(i)}}{\|w\|}$ distance.

geo margin $= \frac{\text{functional margin}}{\|w\|}$

optimal margin clasifier chose w, b to max r.

$\max_{r, w, b} r$, s.t. $\frac{y^{(i)} (w^T x^{(i)} + b)}{\|w\|} \ge r$, $i = 1, 2, \dots, n$. $\Rightarrow \min_{w,b} \|w\|^2$ s.t. $y^{(i)} (w^T x^{(i)} + b) \ge r$

if chose $\|w\| = \frac{1}{r}$, $\Rightarrow \max \frac{1}{\|w\|}$, s.t. $y^{(i)} (w^T x^{(i)} + b) \ge 1$.



suppose $w = \sum_{i=1}^m \alpha_i y^{(i)} x^{(i)}$. $\Rightarrow \min_{w,b} \frac{1}{2} \|w\|^2$ s.t. $y^{(i)} (w^T x^{(i)} + b) \ge 1$,

$\Rightarrow$ as $\min \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y^{(i)} y^{(j)} x^{(i)T} x^{(j)}$, $y^{(i)} (\sum_j \alpha_j y^{(j)} x^{(j)T} x^{(i)} + b) \ge 1$

Simplify to $\max \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle$, s.t. $\alpha_i \ge 0$, $\sum_i y^{(i)} \alpha_i = 0$. Dual optimization.

solve α_i, b, compute $h_{w,b}(x) = g(w^T x + b) = g((\sum_i \alpha_i y^{(i)} x^{(i)})^T x + b) = g(\sum_i \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b)$.

Kernel trick: write algorithm in terms of $\langle x^{(i)}, x^{(j)} \rangle$, mapping from $x \to \phi(x)$. high dim, Find $k(x, z) = \phi(x)^T \phi(z)$, replace $\langle x, z \rangle$ with $k(x, z)$

$J(\theta) = C [\sum_{i=1}^m y^{(i)} [\text{cost}_1(\theta^T x_i)] + (1 - y^{(i)}) [\text{cost}_0(\theta^T x_i)]] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$, $C = \frac{1}{\lambda}$ if $y = 1$, want $\theta^T x > 1$; $y = 0$, want $\theta^T x < -1$

support vector $y^{(i)} (w \cdot x^{(i)} + b) = 1$

for non-linearly $\min_{w,b} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^m \varepsilon_i$ s.t. $y^{(i)} (w \cdot x^{(i)} - b) \ge 1 - \varepsilon_i$, $\varepsilon_i \ge 0$. as slack variable

目标: max margin, min mis-classified.

lagrange duality:

NLP:

Vector Semantics: words meaning characterized by words frequently appear close-by. distributional semantics.

Bag of words: word counts, document-term matrix. / word-word matrix: window of k nearby words PPMI $(w_1, w_2) = \max(\log_2 \frac{P(w_1, w_2)}{P(w_1) P(w_2)}, 0)$

word embedding: prediction based. cbow / skip-gram.

cbow: v_w, w center world, u_w w context word; v vector for word w; $P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$ center c, context o; $P(w_c | w_{o_1}, \dots, w_{o_m}) = \frac{\exp(\frac{1}{2m} u_c^T (v_{o_1} + \dots + v_{o_{2m}}))}{\sum_{i \in V} \exp(\frac{1}{2m} u_i^T (v_{o_1} + \dots + v_{o_{2m}}))}$, $\log P(w_c | w_o) = u_c^T \bar{v}_o - \log(\sum_{i \in V} \exp(u_i^T \bar{v}_o))$

$\frac{\partial \log P(w_c | w_o)}{\partial v_{oi}} = \frac{1}{2m} (u_c - \sum_{j \in V} \frac{\exp(u_j^T \bar{v}_o)}{\sum_{i \in V} \exp(u_i^T \bar{v}_o)} u_j) = \frac{1}{2m} (u_c - \sum_{j \in V} P(w_j | w_o) u_j)$

Fast training, efficient usage of statistic / primarily used to capture word similarity, disproportionate importance given to large counts) word2vec

scales with corpus size, inefficient usage of statistics / generate improved performance on other tasks. capture complex patterns beyond word similarity.

word vec: NN; input one-hot. linear hidden, softmax output.

skip-gram: given a specific word, pre nearby word. CBOW: give context, pre target word. CBOW: learn to pre word by context, skip-g: pre context by center word.

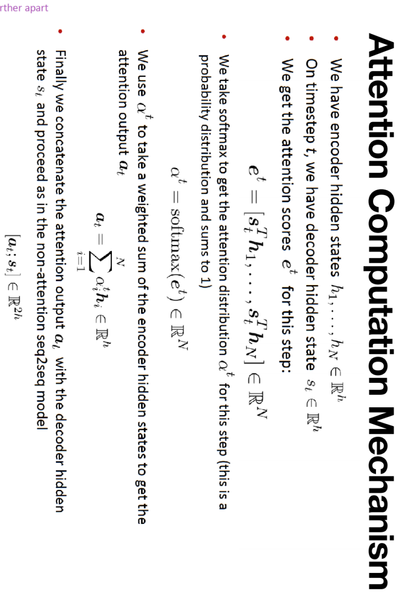
cbow faster, skip-g work well with small training data / huge vocab size. operation in softmax $O(V)$. use Hierarchical softmax / negative sampling.

glove: Ratio of co-occurence probabilities can encode meaning components. only ratio, not counts. $w_i \cdot w_j = \log P(i|j)$

Hierarchical softmax: tree, $\log|V|$. parameter $|V-1|$ nodes. $P(w_1 | w_c) = \sigma(u_{n(w_1,1)}^T v_c) \cdot \sigma(-u_{n(w_3,2)}^T v_c) \cdot \sigma(u_{n(w_3,3)}^T v_c)$

negative sampling: $J_\theta = -h^T v'_w + \log \sum_{w_i \in V} \exp(h^T v'_{w_i})$ $\nabla_\theta J_\theta = \nabla_\theta \mathcal{E}(w) - \sum_{w_i \in V} P(w_i) \nabla_\theta \mathcal{E}(w_i)$

task: language modeling, predict what words come next.



Unsupervised learning:

hierarchical clustering: 1. Agglomerative (iteratively group samples, good for different linkages, but expensive)

2. Divisive clustering: Iteratively divide samples (k means), better decisions for small number of clusters)

Agglomerative: Produce a family of clusterings represented by a dendrogram.

define closest: closest pair (single-link). Farthest pair (complete-link). Average of all pairs, Ward's method (min variance).

k-means: pick k random as cluster centers $C_1, \dots C_K$; Alternate: Assign each x^i to a C^j closest to it, update each C^j. Stop when no points change.

local assignment $L(\{x^i\}, \{a^i\}, \{c^k\}) = \sum_i dist(x^i, c^{a^i})$ as optimization

k-means not deterministic

Initialization: initial matter. : multiple random starts / divisive clustering

GMM: Soft assignment., allow cluster with different size, overlap cluster.

$p(x|\theta) = \sum_{j=1}^{k} P_j N(x; \mu_j, \Sigma_j)$, $q_{ij} = \frac{P_j N(x_i; \mu_j, \Sigma_j)}{\sum_{j'}^{k} P_{j'} N(x_i; \mu_{j'}, \Sigma_{j'})}$ (E step), $\mu_j = \frac{\sum_i q_{ij} x_i}{\sum_i q_{ij}}$, $\Sigma_j = \frac{\sum_i q_{ij}(x_i-\mu_j)(x_i-\mu_j)^T}{\sum_i q_{ij}}$, $P_j = \frac{1}{m}\sum_i q_{ij}$ (M-step).

P_j is the probability of class j occurs, q_{ij} is probability of $x^{(i)}$ belong to class j, μ_j as mean, Σ_j as covariance.

Evaluation: compare to labeled data; plug into some application. (cluster purity), Rand index).

PCA: feature Projections

linear projection: a basis $-(u_1, u_2 \dots u_k)$ length 1, orthonormal basis., $\hat{x}^i = \bar{x} + \sum_{j=1}^{k} z_j^i u_j$, $z_j^i = (x^i - \bar{x})^T u_j$

$x^i = (x_1^i, \dots, x_n^i)$, $i = 1, \dots, m.$, $\hat{x}^i = \bar{x} + \sum_{j=1}^{k} z_j^i u_j$, $z_j^i = (x^i - \bar{x})^T u_j$, $err_k = \sum_{i=1}^{m} \|x^i - \hat{x}^i\|^2$

algorithm: $X \in m \times n$. subtract mean from each row of X., compute $\Sigma = \frac{1}{m} X_c^T X$, find eigenvectors and values of Σ.

Principal components: k eigenvectors with highest eigenvalues.

Neural Network:

a perceptron: $h_{w,b}(x) = f(W^T x) = f(\sum_{i=1}^{d} w_i x_i + b)$. $\tanh f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ sigmoid $f(z) = \frac{1}{1+e^{-z}}$

relu $f(z) = \max(0, z)$, softplus $f(z) = \ln(1+e^z)$

$a_1^{(2)} = f(W_{11}^{(1)} x_1 + W_{12}^{(1)} x_2 + W_{13}^{(1)} x_3 + b_1^{(1)})$. $a_i^{(l)}$ i^{th} neuron at 2^{nd} layer. $W_{ij}^{(l)}$ weight from i^{th} neuron in layer $(l-1)$ to j^{th} neuron in layer l.

Forward: $z^{(2)} = W^{(1)} x + b^{(1)}$. $a^{(2)} = f(z^{(2)}) \dots$, a as activation

Back propagation: $J(W,b;x,y) = \frac{1}{2}\|h_{w,b}(x) - y\|^2$, chain rule, ex. neuron i at l^{th} layer: $\delta_i^{(l)} = \frac{\partial J(w,b)}{\partial z_i^{(l)}} = (a_i^{(l)} - y_i) f'(z_i^{(l)})$, $\frac{\partial z_i^{(l+1)}}{\partial W_{ij}^{(l)}} = a_j^{(l)}$

$\Rightarrow \frac{\partial J(W,b)}{\partial W_{ij}^{(l)}} = a_j^{(l)} \cdot \delta_i^{(l+1)}$, $\frac{\partial J(W,b)}{\partial b^{(l)}} = \delta_i^{(l+1)}$

$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b)$, $b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W,b)$. $\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b) = a_j^{(l)} \delta_i^{(l+1)}$,

$\frac{\partial}{\partial b_i^{(l)}} J(W,b) = \delta_i^{(l+1)}$, $\delta_i^{(l)} = (\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)}) f'(z_i^{(l)})$

1. $\delta^{(l)} = (a_i^{(l)} - y_i) f'(z^{(l)})$, $\delta^{(k)} = W^{(k)T} \delta^{(k+1)} \cdot f'(z^{(k)})$.

2. $\frac{\partial J}{\partial W_{jr}^{(k)}} = a_j^{(k)} \cdot \delta_i^{(k+1)}$, $a^{(1)} = x$, $k = l-1, l-2, \dots 1$; $\frac{\partial J}{\partial b^{(k)}} = \delta^{(k+1)}$.

2. $\frac{\partial J}{\partial W^k} = \delta^{(k+1)} a^{(k)T}$, $\frac{\partial J}{\partial b^{(k)}} = \delta^{(k+1)}$

Choose number of hidden layers (1 or >1): use cross entropy loss for classification

regularization to avoid overfitting.

Gradient checking., ill-conditioned optimization: progress much slower at some direction.

SGD with Momentum: Exponential moving average over subsequent gradients to get more stable directions of descent.

Ada: automatically scaling learning rate for each parameter., for sparse feature, smaller adjustment; bad: , learning rate ↓ to fast as accumulating over all previous gradients., non-convex problem (deep learning), bad convergence.

RMS Prop: moving average over squared gradients, better scaling.

Adam: Exponential moving average of both gradients and squared g, renormalize to remove small bias, use epsilon to avoid 0 divisor.

Decision Trees:

we prefer smaller tree has more examples at each leaf

NP-complete problem. hard. use a greedy heuristic, start from a empty tree, split on next best attribute (feature), recurse. $\arg\max IG(x_i)$

split to choose good attribute.

use counts at leaves to define probability distributions, so we can measure uncertainty

good split if we are more certain about classification after split. (Deterministic good all true or all false / uniform distribution bad)

Entropy: $H(Y) = -\sum_{i=1}^{k} P(Y=y_i) \log_2 P(Y=y_i)$. more uncertainty, more entropy.

Conditional entropy: $H(Y|X) = -\sum_{j=1}^{v} P(x=x_j) \sum_{i=1}^{k} P(Y=y_i|X=x_j) \log_2 P(Y=y_i|X=x_j)$

Information gain: $IG(x) = H(Y) - H(Y|X)$

Standard tree will overfit as no learning bias. (Fixed depth, Fixed number of leaves. validation set for pruning / - first grow a big tree, prune back by optimizing on the held-out set)

no free lunch: dis for decision tree: greedy learning, high variance: small change in data have large effect on tree structure.

solution: $f(x) = \frac{1}{M}\sum_{m=1}^{M} f_m(x)$, Bagging: train $f_m(x)$ on random subsets of training data.

random Forests: train $f_m(x)$ on randomly chosen subsets of input feature.

bagging: bootstrap aggregating., bootstrap: resampling observed data size, same size.

Random forests: de-correlate the tree., it's easy to achieve 0% error rate on training.

randomly restrict the features used in each split.

GNN: Average neighbor messages and apply a neural network.

$h_v^0 = x_v$ init'l layer 0 embeddings are equal to node features

$h_v^k = \sigma(W_k \sum_{u \in N(v)} \frac{h_u^{k-1}}{|N(v)|} + B_k h_v^{k-1})$ $\forall k > 0$, $Z_v = h_v^k$ as output after k-layers aggregation.

GCN $h_v^k = \sigma(W_k \sum_{u \in N(v) \cup v} \frac{h_u^{k-1}}{\sqrt{|N(u)||N(v)|}})$ $H^{(k+1)} = \sigma(D^{-\frac{1}{2}} \hat{A} D^{-\frac{1}{2}} H^{(k)} W_k)$, $\hat{A} = A + I$, $D_{ii} = \sum_j A_{ij}$

Logistic Regression

$h_\theta(x) = g(\theta^T x)$. $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$, learn θ; $h_\theta(x)$ as estimated probability that y=1 given input x. Decision rule: $\theta^T x \geq 0$, y=1; $\theta^T x < 0$, y=0

$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$, $Cost = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$, not convex

So $Cost(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y=1 \\ -\log(1-h_\theta(x)) & \text{if } y=0 \end{cases}$ $\Rightarrow J_\theta = -y\log(h_\theta(x)) - (1-y)\log(1-h_\theta(x)) \Rightarrow J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$

min $J(\theta)$, $\theta_j \leftarrow \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$, min cost is equal to max $h_\theta(x)$ when y=1 and min $h_\theta(x)$ when y=0.

MLE: $P(x^{(i)}, y^{(i)}) = [h_\theta(x^{(i)})]^{y^{(i)}} [1 - h_\theta(x^{(i)})]^{1-y^{(i)}}$

Cross entropy $(P, Q) = H(P, Q) = \sum P(x) \log \frac{1}{Q(x)}$

KL divergence: $D_{KL}(P\|Q) = -\sum_{x} P(x) \log(\frac{Q(x)}{P(x)})$ $KL(P\|Q) \geq 0$, $=0$ only $Q = P$

$J(\theta) = \min_\theta \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$, parameter $\theta = (\theta_0, \theta_1)$, start with some guess of θ (ex. $\theta_0 = \theta_1 = 0$)

Repeat until convergence. $\theta_j \leftarrow \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$. $\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{N}\sum_{i=1}^{N}(h(x^{(i)}) - y^{(i)})$, $\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{N}\sum_{i=1}^{N}(h(x^{(i)}) - y^{(i)}) x^{(i)}$.

$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{\partial}{\partial \theta_j}[y^{(i)}\log(g(\theta^T x)) + (1-y^{(i)})\log(1-h(x^{(i)}))] = (y \frac{1}{g(\theta^T x)} - (1-y)\frac{1}{1-g(\theta^T x)}) \frac{\partial}{\partial \theta_j} g(\theta^T x) = (y \frac{1}{g(\theta^T x)} - (1-y)\frac{1}{1-g(\theta^T x)}) g(\theta^T x)(1-g(\theta^T x)) \frac{\partial}{\partial \theta_j}\theta^T x = (y(1-g(\theta^T x)) - (1-y)g(\theta^T x)) x_j = (y - h_\theta(x)) x_j$

Batch GD: compute based on whole dataset. (Time-consuming, large memory)

Stochastic mini Batch: divide N to $k^* m$ as k mini batches. k term as 1 epoch.

$a = y^T A x$ a scalar, $\frac{\partial a}{\partial x} = y^T A$, $\frac{\partial a}{\partial y} = x^T A^T$; $\alpha = x^T A x$, $\frac{\partial \alpha}{\partial x} = x^T(A + A^T)$; $J(\theta) = \frac{1}{2m}(X\theta - Y)^T(X\theta - Y) = ((X\theta)^T - Y^T)(X\theta - Y) = \theta^T X^T X \theta - 2Y^T X\theta + Y^T Y$

$\frac{\partial J(\theta)}{\theta} = 2X^T X\theta - 2X^T Y$. $\frac{\partial Ax}{\partial x} = A^T$ $\theta = (X^T X)^{-1} X^T Y$

Active learning: the ability to ask queries about the world based upon the past queries and response.

world —query→ / ←response— Active Learner —output→ Model or Classifier.

Step: Start from a pool of unlabeled data (U); pick a few points randomly and get labels by oracle; Repeat: Fit a classifier to labels seen so far, pick best to get label.

best? (closest to boundary)? most uncertain? most likely to ↓ overall uncertainty).

Stream-based Selective sampling: using when obtain unlabeled instance is cheap., query can first be sampled from actual distribution, learned decide whether request label.

as stream / sequential active learning.

pool-Based Sampling: assume small set of labeled data and large pool of unlabeled U, learner supplied with set of unlabeled examples to selects queries.

query Strategy frameworks:

uncertainty sampling: queries the instances least certain how to label (close to boundary), 3 strategies: ① least confident, ② margin sampling, ③ entropy.

① uncertainty $x^*_{LC} = \arg\max_x 1 - P_\theta(\hat{y}|x)$ ② $x^*_M = \arg\min_x P_\theta(\hat{y}_1|x) - P_\theta(\hat{y}_2|x)$ ③ $x^*_H = \arg\max_x -\sum_i P_\theta(y_i|x)\log P_\theta(y_i|x)$.

Query-By-Committee: maintain a committee of models trained on labeled set L, but competing each other, each votes for label of query candidates. pick the most disagreement one., goal is min version space.

1) $x^*_{VE} = \arg\max_x -\sum_i \frac{V(y_i)}{C}\log\frac{V(y_i)}{C}$, $V(y_i)$ as number of votes. C is committee size.. Vote entropy.

2) $x^*_{KL} = \arg\max_x \frac{1}{C}\sum_{c=1}^{C} D(P_{\theta^{(c)}}\|P_C)$, $D(P_{\theta^{(c)}}\|P_C) = \sum_i P_{\theta^{(c)}}(y_i|x)\log\frac{P_{\theta^{(c)}}(y_i|x)}{P_C(y_i|x)}$. $\theta^{(c)}$ as a particular model. Kullback-leibler divergence.

expected Model Change: select instance impart the greatest change to current model if we knew its label.

expected gradient length (EGL), gradient-based training.

$x^*_{EGL} = \arg\max_x \sum_i P_\theta(y_i|x)\|\nabla \ell_\theta(\mathcal{L} \cup \langle x, y_i\rangle)\|$, $\nabla \ell_\theta(\mathcal{L})$ gradient of objective function ℓ wrt model parameter θ, $\nabla \ell_\theta(\mathcal{L} \cup \langle x, y_i\rangle)$ the new gradient be obtain by add $\langle x, y\rangle$ to L

expected error reduction: query instance most reduce error. Measure how much its generalization error is likely to be reduced.

$x^*_{0/1} = \arg\min_x \sum_i P_\theta(y_i|x)(\sum_{u=1}^{U} 1 - P_{\theta^{+\langle x, y_i\rangle}}(\hat{y}|x^{(u)}))$, $\theta^{+\langle x, y_i\rangle}$ as new model after $\langle x, y_i\rangle$ added and retrain.

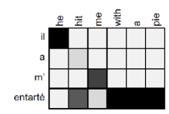
estimate expected future error of model trained using $\mathcal{L} \cup \langle x, y\rangle$ on remaining unlabeled U, query the instance with min expected future error.

Density-weighted methods: query the most "informative" and "representative" (inhabit dense regions of input space)

$x^*_{ID} = \arg\max_x \phi_A(x) \times (\frac{1}{U}\sum_{u=1}^{U} \text{sim}(x, x^{(u)}))^\beta$. $\phi_A(x)$ informativeness of x based on some base query strategy A., β controls relative importance.

Attention Advantages

- Attention significantly improves NMT performance
  - It's very useful to allow decoder to focus on certain parts of the source
- Attention solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with vanishing gradient problem
  - Provides shortcut to faraway states
- Attention provides some interpretability
  - By inspecting attention distribution, we can see what the decoder was focusing on
  - We get (soft) alignment for free!
  - This is cool because we never explicitly trained an alignment system
  - The network just learned alignment by itself



$I(X;Y) = H(X) - H(X|Y)$

$H(X) = -\sum_x P_X(x)\log P_X(x)$

$H(X|Y) = \sum_y P_Y(y)(-\sum_x P_{X|Y}(x|y) \cdot \log(P_{X|Y}(x|y)))$

Gated Recurrent Units

- Proposed by Cho et al. in 2014 as a simpler alternative to the LSTM.
- On each timestep t we have input $x^{(t)}$ and hidden state $h^{(t)}$ (no cell state).

Update gate: controls what parts of hidden state are updated vs preserved

$u^{(t)} = \sigma(W_u h^{(t-1)} + U_u x^{(t)} + b_u)$

Reset gate: controls what parts of previous hidden state are used to compute new content

$r^{(t)} = \sigma(W_r h^{(t-1)} + U_r x^{(t)} + b_r)$

New hidden state content: reset gate selects useful parts of prev hidden state. Use this and current input to compute new hidden content.

$\tilde{h}^{(t)} = \tanh(W_h(r^{(t)} \circ h^{(t-1)}) + U_h x^{(t)} + b_h)$

Hidden state: update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

$h^{(t)} = (1 - u^{(t)}) \circ h^{(t-1)} + u^{(t)} \circ \tilde{h}^{(t)}$

How does this solve vanishing gradient? Like LSTM, GRU makes it easier to retain info long-term (e.g. by setting update gate to 0)