9.5 deep learning

Tuesday, March 10, 2020 10:52 AM

Formalizing the classification generalization problem

Given a probability distribution D over an instance space X, we receive a training set S of points drawn ind. at random from D, and want to predict well on new points drawn from D.

Let $c^* \subseteq X$ be a target concept (e.g. spam emails).
We wish to produce a hypothesis $h \subseteq X$, s.t. the symmetric difference $h \Delta c^*$ is minimized.

Define the true error of h $\text{err}_D(h) = \mu(h \Delta c^*)$, where μ is prob. mass according to D.

And the training error of h $\text{err}_S(h) = \frac{|S \cap (h \Delta c^*)|}{|S|}$

Unfortunately, minimizing $\text{err}_S(h)$ may not minimize $\text{err}_D(h)$ because of overfitting.

Instead, we turn to a restricted class of hypotheses $\mathcal{H} \subseteq 2^X$, which can be interpreted as set systems $(X, \mathcal{H})$.

When can we hope to generalize and not overfit? One condition is when sample size is large compared to VC-dimension.

Let $\mathcal{H}' = \{h \Delta c^* \mid h \in \mathcal{H}\}$ be the collection of error regions of hypotheses in $\mathcal{H}$.

Lemma: $\mathcal{H}$ and $\mathcal{H}'$ have the same VC-dim and shatter function.

proof. For $A \subseteq X$ with $|A| = n$ and $|\{A \cap h \mid h \in \mathcal{H}\}| = \pi_{\mathcal{H}}(n)$, let $A' \subseteq X$ be defined by $A' = A \Delta c^*$. Then

$$\{A \cap h \mid h \in \mathcal{H}\} = \{A \cap (c^* \Delta c^* \Delta h) \mid h \in \mathcal{H}\} = \{A \cap (c^* \Delta h') \mid h' \in \mathcal{H}'\}$$
$$= \{(A \cap c^*) \Delta (A \cap h') \mid h' \in \mathcal{H}'\}$$

But if $A \Delta B = A' \Delta B$, $A = A'$. Thus,
$|\{(A \cap c^*) \Delta (A \cap h') \mid h' \in \mathcal{H}'\}| = |\{A \cap h' \mid h' = \mathcal{H}'\}| \leq \pi_{\mathcal{H}'}(n)$.

Repeating the argument with A' maximally shatterable by $\pi_{\mathcal{H}'}(n)$, we prove the claim. ☑

Thm 5.15 (sample bound): For any class $\mathcal{H}$ and distribution D, if

Thm 5.15 (sample bound): For any class $\mathcal{H}$ and distribution D, if a training sample S is drawn from D of size

$$n \geq \frac{2}{\varepsilon}\left[\log\left(2\pi_{\mathcal{H}}(2n)\right) + \log\frac{1}{\delta}\right],$$

then w.p. $\geq 1-\delta$, every $h \in \mathcal{H}$ with true error $err_D(h) \geq \varepsilon$ has $err_S(h) > 0$. Equivalently, every $h \in \mathcal{H}$ with training error $err_S(h) = 0$ has $err_D(h) < \varepsilon$.

proof. Apply Thm 5.14 to $\mathcal{H}' = \{h \Delta c^* \mid h \in \mathcal{H}\}$.

Thm 5.16 (growth function uniform convergence)

For any class $\mathcal{H}$ and distribution D, if a training sample S is drawn from D of size

$$n \geq \frac{8}{\varepsilon^2}\left[\ln\left(2\pi_{\mathcal{H}}(2n)\right) + \ln\frac{1}{\delta}\right],$$

then w.p. $1-\delta$, every $h \in \mathcal{H}$ will have $\left|err_S(h) - err_D(h)\right| \leq \varepsilon$.

proof. Similar to 5.14 and 5.15, and applying Chernoff-Hoeffding bounds.

i.e. if your sample is big enough, sample error $\approx$ true error.

Corollary 5.17 (from 5.15) For any class $\mathcal{H}$ and distribution D, a training sample S of size

$$O\left(\frac{1}{\varepsilon}\left[VC(\mathcal{H})\log\frac{1}{\varepsilon} + \log\frac{1}{\delta}\right]\right)$$

is sufficient to ensure w.p. $1-\delta$ that every $h \in \mathcal{H}$ with true error $err_D(h) \geq \varepsilon$ has training error $err_S(h) > 0$.

Equivalently, every $h \in \mathcal{H}$ with $err_S(h) = 0$ has $err_D(h) < \varepsilon$.

VC-dim is one measure of the complexity of a set system, which allows proving generalization guarantees. There are others, such as Shannon entropy and Rademacher complexity (how well a concept class $\mathcal{H}$ can fit random noise).

These types of guarantees give us hope that we can train a ML algorithm on a small sample of data and then make useful predictions elsewhere.

Our discussion of VC-dim. tells us that if our concept can be captured by a hypothesis in $\mathcal{H}$ to within some small error, then we can sample enough to find such a classifier. But how do we choose a hypothesis family or optimize to find the right hypothesis within a family?

Deep learning

Consider a classification function $f(\vec{x}) = y$, $\vec{x} \in \mathbb{R}^d$, $y \in \mathbb{R}$ is a label.

We want to approximate f via a chain

$f_0(x) = f_k \circ f_{k-1} \circ f_{k-2} \circ \cdots \circ f_1(x)$, where each f_i is drawn from some simpler family of functions, and $f_0 \sim f$.

We are given a set of training examples $\vec{x}_1, \vec{x}_2, \ldots$ with corresponding labels $y_1, y_2, \ldots$,

and we want to minimize the error $\sum_i (f_0(\vec{x}_i) - y_i)^2$. (squared error)

Or, if instead of having a single label y_i, we have a prob. dist. $\vec{p}_i$, and we produce a prob. dist. $\vec{q}_i = f_0(\vec{x}_i)$,

we often optimize for cross-entropy $\sum_i H(\vec{p}_i, \vec{q}_i)$,

where $H(\vec{p}, \vec{q}) = -\sum_{y \in \mathcal{Y}} p(y) \log q(y)$.

(Recall KL divergence (Kullback-Leibler) $D_{KL}(\vec{p} \| \vec{q}) = -\sum_{y \in \mathcal{Y}} p(y) \log q(y) + \sum_{y \in \mathcal{Y}} p(y) \log p(y)$.)

Major questions

- Architecture – what form should f_i's take
- Loss function – squared error, cross-entropy, etc.
- [illegible]

- Loss function - squared error, cross-entropy, etc.
- Optimization - (stochastic) gradient descent

Architecture

Consider starting with linear separators, or half-spaces $\{\vec{x} \mid \vec{w}^T \vec{x} \geq t\}$, since those are well-understood and easy to compute.

$f_i : \mathbb{R}^m \to \{0,1\}$

$f_i(\vec{x}) = \mathbb{1}_{\{\vec{x} \mid \vec{w}^T \vec{x} \geq b\}}$ for some weight vector $\vec{w}$ and constant b

We can stack a bunch of these together to get

$f_i : \mathbb{R}^m \to \{0,1\}^n$ by $f_{i,j}(\vec{x}) = \mathbb{1}_{\{\vec{x} \mid \vec{w}_j^T \vec{x} \geq b_j\}}$.

But this is bad for chaining together, since then $f_{i+1} : \{0,1\}^n \to \dots$

So let's just remove the indicator variable.

$\Rightarrow f_i : \mathbb{R}^m \to \mathbb{R}^n$ s.t. $f_{i,j}(\vec{x}) = \vec{w}_j^T \vec{x} + b_j$

or $f_i(\vec{x}) = \begin{bmatrix} w_1^T \\ \vdots \\ w_n^T \end{bmatrix} \vec{x} + \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} \equiv W\vec{x} + \vec{b}$.

Our domains and ranges now line up, but we have another problem

$$f_2(f_1(\vec{x})) = W_2(W_1\vec{x} + \vec{b}_1) + \vec{b}_2 = \underbrace{W_2 W_1 \vec{x}}_{W\vec{x}} + \underbrace{W_2\vec{b}_1 + \vec{b}_2}_{\vec{b}}$$

Chaining together affine functions is no more expressive than a single affine function.

When we briefly covered ML earlier, I motivated nonlinearities by a kernel function to transform data that is not linearly separable to data that is. This time, my motivation is just to avoid this collapse in expressivity.

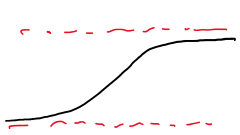
Try: $f_i : \mathbb{R}^m \to \mathbb{R}^n$ s.t. $f_i(\vec{x}) = \sigma(W\vec{x} + \vec{b})$,

where σ is some simple nonlinear function.

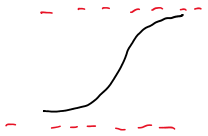
$(x \mapsto x^2)$

Where σ is some simple nonlinear function.

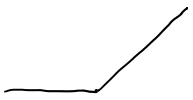
e.g. $\sigma(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ $\quad \frac{\partial \sigma}{\partial x} = 1 - \left(\frac{e^x - e^{-x}}{e^x + e^{-x}}\right)^2$



e.g. $\sigma(x) = \text{sigmoid}(x) = \frac{1}{1+e^{-x}}$ $\quad \frac{\partial \sigma}{\partial x} = \frac{e^{-x}}{(1+e^{-x})^2}$



e.g. $\sigma(x) = \text{ReLU}(x) = \max(0, x)$ $\quad \frac{\partial \sigma}{\partial x} = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$



Differentiability is important, but it turns out most other characteristics aren't really, and you can prove that even small nonlinearities allow $f_0 = f_k(f_{k-1}(\cdots f_1(x)\cdots))$ to approximate any function.

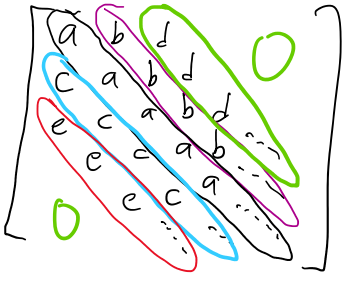
Softmax: We threw out the indicator function in intermediate levels, but we still want something like that in the end for categorical data. So often

$f_k(\vec{x}) = \text{softmax}(\vec{x}) = \frac{1}{|e^{\vec{x}}|_1} e^{\vec{x}}$, which gives a probability distribution over labels.

Reducing degrees of freedom

Full weight matrices have mn df, which may lead to overfitting (if e.g. the VC-dim is too high). A lot of deep learning is putting additional constraints on the matrix.

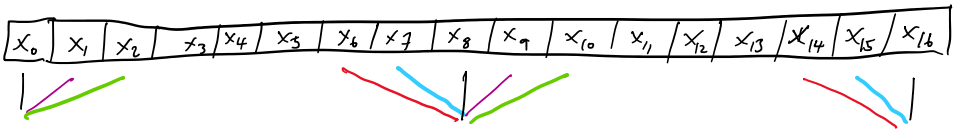
Ex. $W =$



A Toeplitz matrix has $2n-1$ df, as it is constant along every descending diagonal.

We can further reduce df by setting everything to 0 away from a central band.

Let's draw this out this matrix applied to a vector:

$\vec{x} =$ 

$\vec{y} = y_0 \quad \cdots \quad y_8 \quad \cdots \quad y_{16}$

$\to y_2$ $\qquad$ y_{10} $\qquad$ y_{18}

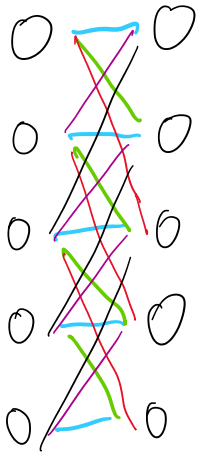
Let's shift $y_i \to y_{i+2}$ to make notation easier

i.e. $y_i = \sum_{m=-\infty}^{\infty} \alpha_m x_{i-m}$ where $\vec{\alpha} = [d \;\; b \;\; a \;\; c \;\; e]$

$$y_2 = d x_2 + b x_1 + a x_0 + c x_{-1} + e x_{-2}$$

Thus, we are applying a convolution to $\vec{x}$.

Can also rewrite in network form, to get a "convolutional neural net":



Note that the edges here share weights.

Very us