CSCI 5525 Machine Learning Fall 2019

Lecture 8: Kernels and Neural Networks

Sep 26 2019

Lecturer: Steven Wu *Scribe: Steven Wu*

1 Kernels: Feature Expansion

Let us revisit the idea of Gaussian kernel.

Gaussian kernel. For any parameter $\sigma > 0$, consider a feature expansion such that

$$\phi(x)^\intercal \phi(x') = \exp\left(-\frac{\|x - x'\|_2^2}{2\sigma^2}\right)$$

The similarity measure $K(x, x') = \phi(x)^\intercal \phi(x')$ defined above is called RBF kernel or Gaussian kernel. So what is a kernel in general?

Definition 1.1. A *kernel function* $K\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a symmetric function such that for any $x_1, x_2, \ldots, x_n \in \mathcal{X}$, the $n \times n$ *Gram matrix* G with each (i, j)-th entry $G_{ij} = K(x_i, x_j)$, is positive semidefinite, that is, for any vector $c \in \mathbb{R}^n$, $c^\intercal K c \geq 0$.

For any kernel K, there is a feature expansion mapping $\phi\colon \mathcal{X} \to \mathbb{H}$ such that

$$\phi(x)^\intercal \phi(x') = K(x, x')$$

where $\mathbb{H}$ is a Hilbert space called the *Reproducing Kernel Hilbert Space (RKHS)* corresponding to K. You could read more about it here.

In the homework, you will learn how to build new kernels out of existing kernels, for example by rescaling an exisiting kernels or taking convex combination of existing kernels.

Now let us return to *Kernel SVM* and replace the product $\phi(x_i)^\intercal \phi(x_j)$ with $K(x_i, x_j)$:

$$\max_{\alpha, \lambda} \sum_i \lambda_i - \frac{1}{2} \sum_{i,j \in [n]} \lambda_i \lambda_j y_i y_j K(x_i, x_j)$$

$$\text{such that for all } i: \qquad 0 \leq \lambda_i \leq C$$

How do we represent the underlying linear predictor now? If we are forced to write down $\hat{\mathbf{w}}$, this will be an infinite-dimensional object. Well, it turns out that we only need to remember the support vector examples, since the prediction can be derived as follows:

$$\phi(x)^\intercal \hat{\mathbf{w}} = \sum_{i=1}^n y_i \lambda_i^* \phi(x)^\intercal \phi(x_i) = \sum_{i=1}^n y_i \lambda_i^* K(x, x_i)$$

This requires iterating the support vector examples (x_i, y_i) along with their associated dual variables λ_i^* to actually get a prediction.

Applying kernel to ridge regression

We can also apply the idea of kernels to ridge regression. This is sometimes called kernelized ridge regression. Recall the notations of design matrix and response vector in linear regression:

$$A = \begin{bmatrix} \leftarrow x_1^\intercal \rightarrow \\ \vdots \\ \leftarrow x_n^\intercal \rightarrow \end{bmatrix} \qquad \mathbf{b} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

The ridge regression solution is given by

$$\hat{\mathbf{w}} = (A^\intercal A + \lambda I)^{-1} A^\intercal \mathbf{b}. \tag{1}$$

Here is a neat trick in linear algebra. Let P be an $N \times M$ matrix while Q be an $M \times N$ matrix:

$$(PQ + I_N)^{-1} P = P(QP + I_M)^{-1}$$

Now if we have $P = (1/\lambda)A^\intercal$ and $Q = A$, then

$$(A^\intercal A + \lambda I_d)^{-1} A^\intercal = A^\intercal (AA^\intercal + \lambda I_n) = A^\intercal (G + \lambda I_n)$$

where G is a $n \times n$ matrix with $G_{ij} = x_i^\intercal x_j$. It follows that

$$\hat{w} = A^\intercal \underbrace{(G + \lambda I_n)\mathbf{b}}_{\mathbf{v}}$$

$$A^\intercal \mathbf{v} = \sum_{i=1}^{n} \mathbf{v}_i x_i$$

Thus, for any new feature vector x, the prediction will be

$$x^\intercal \hat{\mathbf{w}} = \sum_{i=1}^{n} \mathbf{v}_i x^\intercal x_i$$

Now if we replace each x_i with $\phi(x_i)$, then G is the Gram matrix, each entry $G_{ij} = K(x_i, x_j)$, which will allow us to compute $\mathbf{v}$. Then during prediction time

$$\phi(x)^\intercal \hat{\mathbf{w}} = \sum_{i=1}^{n} \mathbf{v}_i \phi(x)^\intercal \phi(x_i)$$

While we can potentially obtain richer representation of the feature space, the downside is that we need to store a lot of information for making predictions.

2 Beyond linear functions

Now that we know how to build predictors using linear functions, how can we use similar ideas to build more powerful predictors?

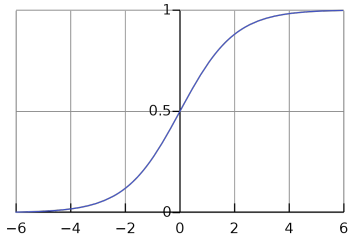

Figure 1: Logistic function $\frac{1}{1+\exp(-w^\intercal x)}$

Warmup: composition of linear functions.

- First linear transformation: $x \to W_1 x + b_1$
- Second linear transformation: $x \to W_2(W_1 x + b_1) + b_2$
- . . . . . .
- L-th linear transformation: $x \to W_L(\dots(W_1 x + b_1)\dots) + b_L$

Question: do we gain anything?
Well, not quite. Observe that

$$W_L(\dots(W_1 x + b_1)\dots) + b_L = Wx + b,$$

where $w = W_L \dots W_1$ and $b = b_L + W_L b_{L-1} + \dots + W_L \dots W_2 b_1$. This is just another linear function.

3 Neural Network Non-linear activation

Example. Recall that in the lecture of logistic regression, we introduce a probability model

$$\mathbf{Pr}[Y = 1 \mid X = x] = \frac{1}{1 + \exp(-w^\intercal x)} \equiv \sigma(w^\intercal x)$$

where σ is the *logistic* or *sigmoid* function. See Figure 1. Now consider a vector-valued version that applies the logisitc function coordinate wise: $f_i(z) = \sigma(W_i z + b_i)$. This gives the most basic neural network.

$$x \to (f_L \circ \dots \circ f_1)(x), \quad \text{where} \quad f_i(z) = \sigma(W_i z + b_i).$$

Here we call the $\{W_i\}_{i=1}^L$ the *weights*, and $\{b_i\}_{i=1}^L$ the *biases*.

More generally, given a collection of *activation* (or *nonlinearities, transfer, link*) functions $\{\sigma_i\}_{i=1}^L$, weights, and biases, we can write down a basic form of a neural network:

$$F(x, \theta) = \sigma_L\left(W_L(\dots W_2 \sigma_1(W_1 x + b_1) + b_2 \dots) + b_L\right)$$

where θ denotes the set of parameters $W_1, \dots, W_L, b_1, \dots, b_L$.

Choices of activation functions.

- Indicator or threshold: $z \to \mathbf{1}[z \geq 0]$.
- Sigmoid or logistic: $z \to \frac{1}{1+\exp(-z)}$.
- Hyperbolic tangent: $z \to \tanh(z)$.
- Rectificed linear unit (ReLU): $z \to \max\{0, z\}$. Variants include Leaky ReLU and ELU. These are the most popular choices now since the AlexNet paper [1].
- Identity: $z \to z$. This is often used in the last layer when we evaluate the loss.

Learning pipeline.

- Split the data into traning and validation datasets.
- **Hyperparameters:** pick a class of functions for the networks (or architecture), the function $F(\cdot, \cdot)$.
- **ERM:** on training data set $\{(x_i, y_i)\}$, we pick a loss function ℓ (e.g. cross entropy loss function, square loss) and perform *empirical risk minimization*:

$$\arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, F(x_i, \theta)) =$$

- Choose the architecture with the lowest validation error.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.