CMSE 890-002: Mathematics of Deep Learning, MSU, Spring 2020

Lecture 11: Logistic Regression & Neurons

February 5, 2020

Lecturer: Matthew Hirn

Kernels allow us to implicitly use a nonlinear dictionary $\Phi(x)$ without having to specify $\Phi(x)$, but the dictionary is not learned. The kernel $K(x,x')$ specifies a unique representation $\Phi(x)$, and as the above calculation (20) shows the learning aspect specifies the parameters of the model, but not the representation. Deep learning takes an alternate approach, in which the features themselves are parameterized, and these parameters are learned. Let $\theta = (\boldsymbol{\theta}, \mathbf{w}) \in \mathbb{R}^n$ be the parameters of a deep network in which the last (output) layer computes a linear regression. Then, drawing an analogy with (18) we can write this network as:

$$f(x;\theta) = \langle \Phi(x;\boldsymbol{\theta}), \mathbf{w} \rangle ,$$

where the vector $\boldsymbol{\theta}$ parameterizes the representation $\Phi(x;\boldsymbol{\theta})$ and the vector $\mathbf{w}$ gives the weights of the linear regression over this representation. Thus when fitting the optimal $\widehat{\theta}$ to training data, we not only learn how to combine the features via $\mathbf{w}$, but we learn the features themselves through the parameter vector $\boldsymbol{\theta}$. This will allow for very powerful learning algorithms, but because of the features dependence on the parameters $\boldsymbol{\theta}$ it has made understanding why deep learning works a much more difficult problem than standard dictionary or kernel approaches. Nevertheless, some progress has been made and new results are arriving every month. In the rest of this course we will explore some of these results.

5.2 Logistic regression and nonlinearity

Logistic regression is a learning algorithm for categorical classes (i.e., a finite number of classes), that adds a nonlinear function to linear models in order change the output into a probability. Given a new test point $x \in \mathcal{X}$, it does not make a hard decision about which class x belongs to, but rather, for each possible class, gives the probability of x belonging to that class. It is a very simple version of a neural network.

The way logistic regression does this is with the *sigmoid function*, which is defined as:

$$\sigma(z) = \frac{1}{1+e^{-z}}, \quad z \in \mathbb{R}.$$

Suppose now that $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{0,1\}$, i.e., our data points are d-dimensional vectors and we are working on a binary classification problem. Logistic regression defines a parameterized hypothesis space $\mathcal{P}$ of conditional probability density functions $p_{Y|X}(y \mid x, \theta)$, where $\theta \in \mathbb{R}^d$

and

$$p(y = 1 \mid x, \theta) = \sigma(\langle x, \theta \rangle) = \frac{1}{1 + e^{-\langle x, \theta \rangle}}$$
$$p(y = 0 \mid x, \theta) = 1 - p(y = 1 \mid x, \theta)\,.$$

To solve for the parameters θ, we can use maximum likelihood estimation (MLE) from Section 1.3, or maximum a posteriori (MAP) from Section 3.2. To keep things simple, we explain with MLE. Suppose we are given a training set $T = \{(x_i, y_i)\}_{i=1}^N$ in which the x_i's are sampled independently from $p_X(x)$. Recall that the MLE model is:

$$\begin{aligned}
\widehat{\theta} = \arg\max_{\theta \in \mathbb{R}^d} p_{X,Y}(T \mid \theta) &= \arg\max_{\theta \in \mathbb{R}^d} \prod_{i=1}^N p(y_i \mid x_i, \theta) \\
&= \arg\max_{\theta \in \mathbb{R}^d} \prod_{i=1}^N [\sigma(\langle x_i, \theta \rangle)]^{y_i} [1 - \sigma(\langle x_i, \theta \rangle)]^{1 - y_i} \\
&= \arg\max_{\theta \in \mathbb{R}^d} \sum_{i=1}^N \left[y_i \log \sigma(\langle x_i, \theta \rangle) + (1 - y_i) \log(1 - \sigma(\langle x_i, \theta \rangle)) \right]\,.
\end{aligned}$$

Unlike linear regression, $\widehat{\theta}$ cannot be solved for analytically and in closed form. However, one can compute the gradient of

$$\mathcal{L}(\theta) = -\sum_{i=1}^N \left[y_i \log \sigma(\langle x_i, \theta \rangle) + (1 - y_i) \log(1 - \sigma(\langle x_i, \theta \rangle)) \right]$$

analytically, which allows for efficient minimization. Indeed,

$$\nabla_\theta \mathcal{L}(\theta) = \sum_{i=1}^N [\sigma(\langle x_i, \theta \rangle) - y_i] x_i\,.$$

Logistic regression can be extended to more than two classes; the resulting algorithm is called *softmax regression*. Suppose $\mathcal{Y} = \{1, \ldots, M\}$, i.e., we have M classes. For each class, we learn weights $\theta_m \in \mathbb{R}^d$, $1 \leq m \leq M$. Softmax regression then assigns probabilities as:

$$p\left(y = m_0 \mid x, \{\theta_m\}_{m=1}^M\right) = \frac{e^{-\langle x, \theta_{m_0} \rangle}}{\sum_{m=1}^M e^{-\langle x, \theta_m \rangle}}\,.$$

Softmax regression is often used in the last layer of a neural network trainied for classification, but the relevance goes further. Logistic regression and softmax regression show the importance of nonlinearity, and in particular, having a nonlinear function of a linear transformation. Neural networks build upon this by cascading may successive alternations of linear (or affine) functions and nonlinear functions, such as the sigmoid.

6 What is an artificial neural network?

Recall the logistic regression classifier from Section 5.2:

$$x \mapsto \frac{1}{1+e^{-\langle x,\theta\rangle}}\,,$$

which consisted of mapping a data point $x \in \mathcal{X} = \mathbb{R}^d$ into $\mathbb{R}$ via $x \mapsto \langle x, \theta\rangle$, and then from $\mathbb{R}$ again into $\mathbb{R}$ via the nonlinear sigmoid function $\sigma(z) = 1/(1+e^{-z})$ with $z = \langle x,\theta\rangle$. A neural network generalizes the logistic regression function by defining a mathematical *neuron*[11] as

$$\eta(x) = \sigma(\langle x, w\rangle + b)\,, \quad w \in \mathbb{R}^d\,,\, b \in \mathbb{R}\,, \tag{21}$$

and where $\sigma : \mathbb{R} \to \mathbb{R}$ is a nonlinear function, e.g., the sigmoid function, but there are other possibilities, including:

- Perceptron: $\sigma(z) = 0$ if $z \leq 0$ and $\sigma(z) = 1$ if $z > 0$
- Sigmoid: $\sigma(z) = 1/(1+e^{-z})$
- Hyperbolic tangent: $\sigma(z) = \tanh(z)$
- Rectified linear unit (ReLU): $\sigma(z) = \max(0, z)$
- Leaky ReLU: $\sigma(z) = \max(az, z)$ with $0 \leq a < 1$.
- Smoothed versions of (leaky) ReLU to eliminate the point of non-differentiability at $z = 0$
- Absolute value (or modulus if $z \in \mathbb{C}$): $\sigma(z) = |z|$[12]

Figure 21 plots some of these nonlinear functions and Figure 22 depicts the a single mathematical neuron.

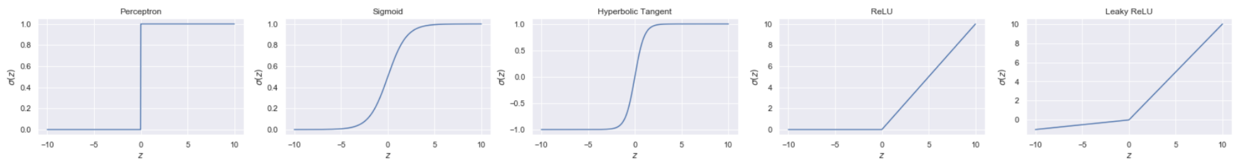

Figure 21: *Examples of nonlinear activation functions $\sigma(z)$. From left to right: perceptron, sigmoid, hyperbolic tangent, rectified linear unit (ReLU), leaky ReLU with $a = 0.10$.*

[11] Yes I know η does not correspond to "n," but it still looks like it...

[12] Thanks to Anna Little for reminding me to put this one.

A

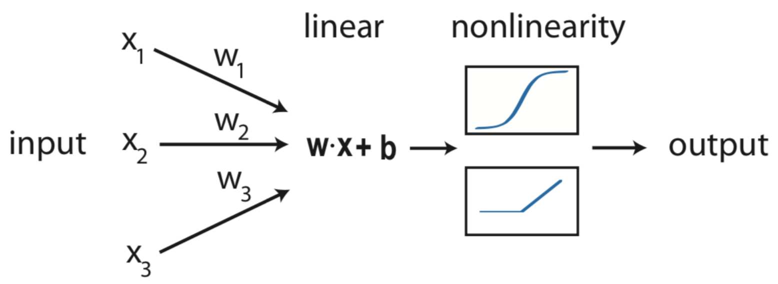

Figure 22: *An illustration of a single mathematical neuron, defined in equation* (21)*. Here the notation is a bit different than the text, as* $x = (x_1, x_2, x_3) \in \mathbb{R}^3$*, not three training points, and* $w = (w_1, w_2, w_3) \in \mathbb{R}^3$*. Figure taken from [5].*

References

[1] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations*, 2015.

[2] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Conference on Computer Vision and Pattern Recognition*, 2009.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. 2012.

[4] Larry Greenemeier. AI versus AI: Self-taught AlphaGo Zero vanquishes its predecessor. *Scientific American*, October 18, 2017.

[5] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre G.R. Day, Clint Richardson, Charles K. Fisher, and David J. Schwab. A high-bias, low-variance introduction to Machine Learning for physicists. arXiv:1803.08823, 2018.

[6] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer-Verlag New York, 2nd edition, 2009.

[7] J. Hartlap, P. Simon, and P. Schneider. Why your model parameter confidences might be too optimistic - unbiased estimation of the inverse covariance matrix. *Astronomy and Astrophysics*, 464(1):399–404, 2007.

[8] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Adaptive Computation and Machine Learning. The MIT Press, 2002.

[9] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[10] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference for Learning Representations*, San Diego, CA, USA, 2015.