E0 306: Deep Learning: Theory and Practice — 26th February 2019

Scribe Notes
Adversarial Classification

Lecturer: Amit Deshpande — *Scribe*: Neha Jawalkar

1 Adversarial classification

The following sections are based on [1].

1.1 Notation

We consider a binary classification problem over the domain $(\mathcal{X}, \mathcal{Y})$, where $\mathcal{X} = \bigtimes_{i=1}^{n} \mathcal{X}_i$ is the input and $\mathcal{Y} = \{+, -\}$ is the (binary) output. An element of $\mathcal{X}$ is denoted by $X = (X_1, \ldots, X_i, \ldots, X_n)$, where X_i is sampled from $\mathcal{X}_i$. Tuples $(x, +)$ are generated i.i.d from a distribution $P(X \mid +)$ and tuples $(x, -)$ are generated i.i.d from a distribution $P(X \mid -)$, giving us the global distribution $P(X) = P(+)P(X \mid +) + P(-)P(X \mid -)$. The training set $\mathcal{S}$ is a set of (x, y) tuples and so is the test set $\mathcal{T}$. In order to learn a classifier $\mathcal{C}$, adversarial classification is modeled as a game between the classifier, $\mathcal{C}$, and an adversary $\mathcal{A}$.

1.2 The Classifier $\mathcal{C}$

The classifier attempts to learn from the training set $\mathcal{S}$ a function $\mathcal{C} : \mathcal{X} \mapsto \mathcal{Y}$ that generalizes well to the test set $\mathcal{T}$.

The classifier takes the following as input:

1. A utility matrix $\mathcal{U}_C(y_\mathcal{C}; y)$, which quantifies the utility of the classifier in classifying an instance of the class y as $y_\mathcal{C}$.

2. A tuple $V = (V_1, \ldots, V_i, \ldots, V_n)$, where V_i denotes the cost of measuring feature i.

Intuitively, the job of the classifier is to maximize accuracy by correctly classifying instances, but only insofar the cost incurred by boosting accuracy in terms of feature measurement remains reasonable.

1.3 The Adversary A

The job of the adversary is to learn a perturbation function $\mathcal{A} : \mathcal{X} \mapsto \mathcal{X}$, such that perturbing $x \in \mathcal{X}$ to $\mathcal{A}(x) \in \mathcal{X}$ causes the classifier to perform badly on the test set $\mathcal{T}$. The adversary takes the following as inputs:

1. A utility matrix $\mathcal{U}_A(y_\mathcal{C}; y)$, which quantifies the utility of the adversary in making the classifier classify an instance of the class y as $y_\mathcal{C}$.

2. A set of functions $W_i(x_i, x'_i)$, which denote the cost of changing the i^{th} feature from x_i to x'_i. We define the total cost of perturbation as $W(x, x') = \Sigma_i W_i(x_i, x'_i)$.

1.4 The utility functions

We can now define the utility functions of the classifier and the adversary. The classifier tries to correctly classify the perturbed instance $\mathcal{A}(x)$ while avoiding the overhead of excessive measurement. As such, its utility function is given by

$$\mathcal{U}_{\mathcal{C}} = \sum_{(x,y)\in\mathcal{X}\mathcal{Y}} P(x,y)\Big[\mathcal{U}_C(\mathcal{C}(\mathcal{A}(x)),y) - \sum_{X_i\in\mathcal{X}_{\mathcal{C}}(x)} V_i\Big]$$

where $\mathcal{X}_{\mathcal{C}}(x)$ is the set of features that $\mathcal{C}$ decides to measure while classifying x.

Similarly, the adversary tries to perturb the instance x so as to maximize the utility it gains from the perturbation while incurring only a reasonable cost by making said perturbation.

$$\mathcal{U}_{\mathcal{A}} = \sum_{(x,y)\in\mathcal{X}\mathcal{Y}} P(x,y)\Big[\mathcal{U}_A(\mathcal{C}(\mathcal{A}(x)),y) - W(\mathcal{A}(x),x)\Big]$$

Some realizations of the adversarial classification game permit Nash equlibria. In particular, the following holds:

Theorem 1. Consider a classification game with a binary cost model for Adversary, i.e., given a pair of instances x and x_0, Adversary can either change x to x_0 (incurring a unit cost) or it cannot (the cost is infinite). This game always has a Nash equilibrium, which can be found in time polynomial in the number of instances.

1.5 Cost Sensitive Learning

In order to initiate the game, we start by modelling a strategy for $\mathcal{C}$. To formulate this strategy, we assume that $\mathcal{C}$ is oblivious to the presence of $\mathcal{A}$. We introduce here the Naive Bayes model, since it forms the basis for $\mathcal{C}$'s strategy. The Naive Bayes model works as follows: for every input instance x and label y, it estimates $P(y \mid x)$ according to the following formula:

$$P(y \mid x) = \frac{P(y)}{P(x)}P(x \mid y) = \frac{P(y)}{P(x)}\prod_i P(x_i \mid y)$$

where $P(x \mid y) = \prod_i P(x_i \mid y)$ is the Naive Bayes assumption. The predicted class $y_{\mathcal{C}}$ is $\operatorname{argmax}_{y\in\mathcal{Y}} P(y \mid x)$.

We slightly tweak Naive Bayes to absorb $\mathcal{C}$'s utility matrix (and ignore, as the paper does, the costs of measurement V_i). Let us define the conditional utility $U(y_{\mathcal{C}} \mid x)$ as the utility accrued by $\mathcal{C}$ when it assigns $y_{\mathcal{C}}$ to x. It is defined as follows:

$$U(y_{\mathcal{C}} \mid x) = \sum_{y\in\mathcal{Y}} P(y \mid x) U_C(y_{\mathcal{C}}; y)$$

The Bayes optimal prediction for x is then the class $\operatorname{argmax}_{y\in\mathcal{Y}} U(y \mid x)$.

1.6 Adversary Strategy

We start modeling $\mathcal{A}$'s strategy by making an important assumption, namely, that of complete information on $\mathcal{A}$'s part. This means that $\mathcal{A}$ is privy to all of $\mathcal{C}$'s parameters.

From the Naive Bayes assumption, we have that

$$\log \frac{P(+ \mid x)}{P(- \mid x)} = \log \frac{P(+)}{P(-)} + \sum_i \log \frac{P(x_i \mid +)}{P(x_i \mid -)}$$

Let us denote $\log \frac{P(+|x)}{P(-|x)}$ by $LO_{\mathcal{C}}(x)$, where LO stands for 'log odds'. We introduce some more notation, the purpose of which is not immediately obvious. Let $LT(U_{\mathcal{C}}) = \log \frac{U(-,-)-U(+,-)}{U(+,+)-U(-,+)}$. Here LT stands for 'log threshold'. Finally, let $gap(x) = LO_{\mathcal{C}}(x) - LT(U_{\mathcal{C}})$.

Our classifier will classify x as positive if its expected utility from doing so is greater than its expected utility of classifying x as negative. This can be mathematically written as

$$\begin{aligned}
&P(+ \mid x)U(+,+) + P(- \mid x)U(+,-) > P(- \mid x)U(-,-) + P(+ \mid x)U(-,+)\\
\implies &P(+ \mid x)\Big[U(+,+) - U(-,+)\Big] > P(- \mid x)\Big[U(-,-) - U(+,-)\Big]\\
\implies &\frac{P(+ \mid x)}{P(- \mid x)} > \frac{U(-,-) - U(+,-)}{U(+,+) - U(-,+)}\\
\implies &\log \frac{P(+ \mid x)}{P(- \mid x)} > \log \frac{U(-,-) - U(+,-)}{U(+,+) - U(-,+)}\\
\implies &LO_{\mathcal{C}}(x) > LT(U_{\mathcal{C}})\\
\implies &LO_{\mathcal{C}}(x) - LT(U_{\mathcal{C}}) > 0\\
\implies &gap(x) > 0
\end{aligned}$$

Given an instance x that is classified as positive, the adversary accrues additional utility $\Delta U_A = U_A(-,+) - U_A(+,+)$ by modifying x so it is classified as negative. It also accrues a cost of modification given by $W(x, \mathcal{A}(x))$. The adversary modifies x if and only if $\Delta U_A > W(x, \mathcal{A}(x))$. The problem of finding an optimal adversarial strategy thus reduces to finding a minimum cost perturbation of x such that $\mathcal{C}(\mathcal{A}(x)) = -$ and $\Delta U_A - W(x, \mathcal{A}(x)) > 0$.

Let us examine what the adversary must needs do to x to make C change its classification. We start with a positively classified instance x. Since x is positively classified, it must be that $gap(x) > 0$, i.e. $LO_{\mathcal{C}}(x) > LT(U_{\mathcal{C}})$. To make $\mathcal{C}$ classify x as negative, we must perturb x to x' such that $LO_{\mathcal{C}}(x') < LT(U_{\mathcal{C}})$.

$$\begin{aligned}
&LO_{\mathcal{C}}(x') < LT(U_{\mathcal{C}})\\
\implies &LO_{\mathcal{C}}(x') + LO_{\mathcal{C}}(x) - LO_{\mathcal{C}}(x) < LT(U_{\mathcal{C}})\\
\implies &LO_{\mathcal{C}}(x) - LT(U_{\mathcal{C}}) < LO_{\mathcal{C}}(x) - LO_{\mathcal{C}}(x')\\
\implies &gap(x) < LO_{\mathcal{C}}(x) - LO_{\mathcal{C}}(x')\\
\implies &LO_{\mathcal{C}}(x) - LO_{\mathcal{C}}(x') > gap(x)
\end{aligned}$$

Let us examine the left hand side of the above equation, namely, $LO_{\mathcal{C}}(x) - LO_{\mathcal{C}}(x')$.

$$
\begin{aligned}
& LO_{\mathcal{C}}(x) - LO_{\mathcal{C}}(x') \\
=& \log \frac{P(+ \mid x)}{P(- \mid x)} - \log \frac{P(+ \mid x')}{P(- \mid x')} \\
=& \log \frac{P(+)}{P(-)} + \sum_i \log \frac{P(x_i \mid +)}{P(x_i \mid -)} - \log \frac{P(+)}{P(-)} - \sum_i \log \frac{P(x'_i \mid +)}{P(x'_i \mid -)} \\
=& \sum_i \log \frac{P(x_i \mid +)}{P(x_i \mid -)} - \sum_i \log \frac{P(x'_i \mid +)}{P(x'_i \mid -)} \\
=& \sum_i \log \frac{P(x_i \mid +)}{P(x_i \mid -)} - log \frac{P(x'_i \mid +)}{P(x'_i \mid -)}
\end{aligned}
$$

Let $\Delta LO_{i,x'_i} = \log \frac{P(x_i|+)}{P(x_i|-)} - log \frac{P(x'_i|+)}{P(x'_i|-)}$. Thus, the transformation affected by the adversary on x must result in an x' such that

$$
\begin{aligned}
& LO_{\mathcal{C}}(x) - LO_{\mathcal{C}}(x') > gap(x) \\
\implies & \sum_i \Delta LO_{i,x'_i} > gap(x)
\end{aligned}
$$

We formulate this as an integer linear program.

In order to do so, we introduce some additional notation. Let δ_{i,x'_i} be a binary variable which is 1 if x_i has been modified to x'_i and 0 otherwise. Our objective is to find a minimum cost transformation such that $\sum_i \Delta LO_{i,x'_i} > gap(x)$ is satisfied. We can do so by solving the following binary integer linear program:

$$
\min \left\{ \sum_{X_i \in \mathcal{X}_c(x)} \sum_{x'_i \in \mathcal{X}_i} W(x_i, x'_i) \delta_{i,x'_i} \right\} \text{ s.t.}
$$

$$
\sum_{X_i \in \mathcal{X}_c(x)} \sum_{x'_i \in \mathcal{X}_i} \Delta LO_{i,x'_i} \delta_{i,x'_i} \geq gap(x)
$$

$$
\delta_{i,x'_i} \in \{0,1\}, \sum_{x'_i \in \mathcal{X}_i} \delta_{i,x'_i} \leq 1
$$

The δ_{i,x'_i} values encode if x_i has been transformed to x'_i or not.

$\sum_{X_i \in \mathcal{X}_c(x)} \sum_{x'_i \in \mathcal{X}_i} W(x_i, x'_i) \delta_{i,x'_i}$ is merely the cost of transformation recoded in terms of δ_{i,x'_i}.

Similarly, $\sum_{X_i \in \mathcal{X}_c(x)} \sum_{x'_i \in \mathcal{X}_i} \Delta LO_{i,x'_i} \delta_{i,x'_i} \geq gap(x)$ is a reformulation of our previously derived constrained on the transformation from x to x', i.e. $\sum_i \Delta LO_{i,x'_i} > gap(x)$.

$\sum_{x'_i \in \mathcal{X}_i} \delta_{i,x'_i} \leq 1$ ensures that every feature X_i is being mapped to exactly one element of $\mathcal{X}_i$ by the transformation.

The solution of the above linear program is called the Minimum Cost Camouflage (MCC) of x. It is the cost-wise closest neighbour of x that is classified as negative by $\mathcal{C}$.

The adversary's strategy is now obvious: for each point x, first calculate $W(x, MCC(x))$. If $W(x, MCC(x)) > \Delta U_A$, transform x to $MCC(x)$, else keep x the way it is.

We note that this game can continue ad infinitum, with the classifier adapting to the adversary's strategy, the adversary adapting to the classifier's, and so on. However, as was done in class, we focus only on one round of the game, with $\mathcal{C}$ playing Naive Bayes, and $\mathcal{A}$ formulating a best response to it.

2 Intriguing properties of neural networks

The following sections are based on [3].

2.1 The natural basis vectors are not special

A natural way of interpreting the latent code of neural networks is to focus on inputs that maximally activate a given hidden unit. Mathematically, this is the set

$$x'_{e_i} = \underset{x \in \mathcal{I}}{\operatorname{argmax}} \langle \phi(x), e_i \rangle$$

where $\mathcal{I}$ is a set of images and e_i is the i^{th} basis vector.

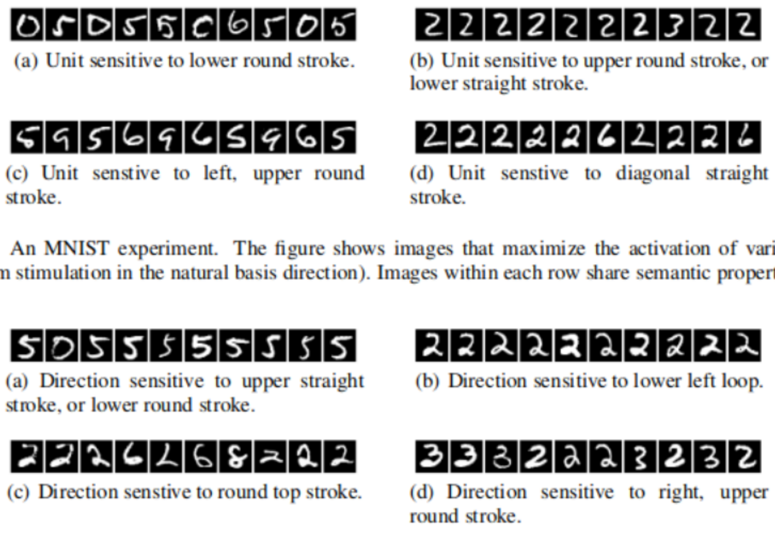
The authors of [3] find that while x'_{e_i} has semantic meaning, it is not special in any way, i.e., *any* random direction $v \in R^n$ gives rise to a x'_v that has interpretable high level semantic properties. This disabuses the notion that the basis vectors have 'special' meaning, and seems to imply that neural networks discover an embedding *space* that is meaningful. Figure 1 shows experiments with MNIST that demonstrate that sets of images maximally activating the natural basis are no more meaningful than sets of images maximally activating a random basis.

2.2 Neural Networks do not learn smooth functions

Neural networks model a probability distribution over the label space conditioned on a given input. The deep stack of non-linear layers has been argued to be a way for networks to encode a non-local generalization prior over the input space. This means that a neural network will assign probabilities to parts of the input space that do not contain any training examples. For instance, a network might label an image x with a label l and also label a rotated version of x, x_{rot} with l, without ever having seen x_{rot}, because it has the same statistical structure as x, despite lying far away from it in pixel space.

Regardless of what networks do to parts of the input space far away from a training example, it is assumed that they behave well locally, i.e., an ϵ-ball around a training example will get assigned the same label as the training example with high probability. This is fair to expect from the learned function, since applying a small perturbation to an image does not often change its class.

A key result of the paper is that for functions learnt by deep neural networks, the above smoothness assumption does not hold. In particular, the authors propose an optimization problem the solution of which, when applied to a hitherto correctly classified image, renders it unrecognisable by a neural network.



(a) Unit sensitive to lower round stroke.

(b) Unit sensitive to upper round stroke, or lower straight stroke.

(c) Unit senstive to left, upper round stroke.

(d) Unit senstive to diagonal straight stroke.

Figure 1: An MNIST experiment. The figure shows images that maximize the activation of various units (maximum stimulation in the natural basis direction). Images within each row share semantic properties.

(a) Direction sensitive to upper straight stroke, or lower round stroke.

(b) Direction sensitive to lower left loop.

(c) Direction senstive to round top stroke.

(d) Direction sensitive to right, upper round stroke.

Figure 2: An MNIST experiment. The figure shows images that maximize the activations in a random direction (maximum stimulation in a random basis). Images within each row share semantic properties.

Figure 1: Images that maximally activate the natural basis and images that maximally activate a random direction. Both sets share high level semantic similarities.

Let $f : R^m \mapsto \{1 \cdots k\}$ be a classifier-learnt mapping from images to a label set. Let $loss_f : R^m \times \{1 \cdots k\} \mapsto R+$ be a continuous loss associated with f. For a given image x^m and target label $l \in \{1 \cdots k\}$, the solution we seek is to the following problem:

$$\begin{aligned} \min \|r\| \ \text{ s.t.} \\ f(x+r) = l \\ x + r \in [0,1]^m \end{aligned}$$

Intuitively, $x+r$ is the closest neighbour of x classified as l by the classifier. Let $D(x,l) = x+r$, where r is a minimum distortion described above. Operationally, we go about finding $D(x,l)$ as follows: we conduct a line search over a scalar c to find the maximum $c > 0$ such that the solution r to the following optimization problem satisfies $f(x+r) = l$.

$$\begin{aligned} \min c|r| + loss_f(x+r,l) \text{ s.t.} \\ x + r \in [0,1]^m \end{aligned}$$

The authors use L-BFGS to solve the above optimization problem. We do not include a description of BFGS or its limited memory analogue here.

Interestingly, in addition to successfully generating barely perceptible distortions that fool the network, the authors found that their distortions generalized across models and also across training sets. This means that their examples were able to fool models with different hyper-parameters (such as a different number of layers), and also models that were trained on training sets disjoint from their own.

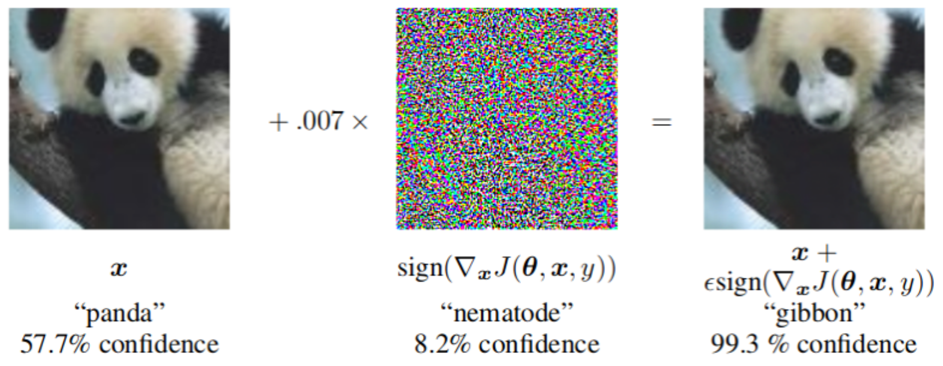

$+ .007 \times$
$=$
$\boldsymbol{x}$
"panda"
57.7% confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

Figure 1: A demonstration of fast adversarial example generation applied to GoogLeNet (Szegedy et al., 2014a) on ImageNet. By adding an imperceptibly small vector whose elements are equal to the sign of the elements of the gradient of the cost function with respect to the input, we can change GoogLeNet's classification of the image. Here our ϵ of .007 corresponds to the magnitude of the smallest bit of an 8 bit image encoding after GoogLeNet's conversion to real numbers.

Figure 2: An example of an image being misclassified by the application of the fast gradient sign method

3 Generating and harnessing adversarial examples

While the ready availability of adversarial examples has been proposed to be an artefact of the extreme non-linearity of deep neural networks, the authors argue that linearity in high dimensions is sufficient to cause a neural network to misbehave in the ϵ-neighborhood of a training example. Consider a perturbation of x by η, such that $\|\eta\|_\infty = \epsilon$. Let the perturbed x be called $\tilde{x}$. Let us examine how the activations of x and $\tilde{x}$ differ with respect to a weight vector w. To do this, we look at the dot product of w and $\tilde{x}$:

$$w^\top \tilde{x} = w^\top x + w^\top \eta$$

The perturbation η causes the activation to grow by $w^\top \eta$. This can be maximized by setting $\eta = \epsilon sign(w)$. If $w \in R^n$, and $\frac{1}{n}\sum_i w_i = m$, then $w^\top \tilde{x}$ will grow by ϵmn. The constraint $\|\eta\|_\infty = \epsilon$ allows us to perturb each dimension of x by ϵ, which means that a small perturbation along each coordinate of a high-dimensional x can induce a large change in the activation $w^\top \tilde{x}$ (courtesy the n in ϵmn).

The authors argue that neural networks are 'linear enough' to be affected by this phenomenon. Activation functions (like the ReLU or the leaky ReLU) are explicitly designed be to linear, and even non-linear activations like the sigmoid are carefully tuned to spend most of their time in the more linear parts of their curves.

Based on the above notion that neural networks are linear enough, the authors propose what they call the 'fast gradient sign method' for generating adversarial examples. Let $J(\theta, x, y)$ be the cost function associated with a classification task. The perturbation η is obtained by linearly approximating J in the neighborhood of θ as follows:

$$\eta = \epsilon sign(\nabla_x J(\theta, x, y))$$

We note that the gradient here is with respect to the input x, and not with respect to the weight vector w. We believe the distinction is important as one more commonly observes that latter, courtesy gradient-based methods for optimizing cost functions. We also note that this gradient can be efficiently computed using backpropogation.

Figure 2 shows an example of an image being misclassified as a consequence of being perturbed by the fast gradient sign method.

3.1 Adversarial Training of Neural Networks

The fast gradient sign method of generating adversarial examples suggests a natural regularizer to avoid overfitting:

$$J(\theta, x, y) = \alpha J(\theta, x, y) + (1 - \alpha) J(\theta, x + \epsilon sign(\nabla_x J(\theta, x, y))$$

. It involves considering loss not only over x, but also over the perturbed version of x generated by the fast gradient sign method. It forces the classifier to not only do well on x, but also to not do badly in the ϵ-neighborhod of x. The authors found $\alpha = 0.5$ to work well in practice.

References

[1] Nilesh Dalvi, Pedro Domingos, Mausam, Sumit Sanghai, and Deepak Verma. Adversarial classification. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 99–108, New York, NY, USA, 2004. ACM.

[2] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572, 2015.

[3] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2014.