ML Reading Group

Introduction to Neural Networks - Part II

Learning Error Vs. Generalization Error

["Train faster, generalize better", Hardt, 2016]

Consider regression problem. We are given a sample

$$S = ((x_1, y_1), \dots, (x_n, y_n))$$

$$y_i = f(x_i), \quad x_1, \dots, x_n \text{ i.i.d.} \sim p_X$$

f is the function that we want to learn.

Our estimate is the function $\hat{f}_S$ ($\hat{f}$ depends on the sample S) and our performance metric is L_2 loss: Then, the RISK is

$$R = \mathbb{E}_{\substack{x \sim p_X \\ S}} \left[|f(x) - \hat{f}_S(x)|^2 \right]$$

(expected value with respect to the sample S and to the new value x.)

Since we do not know the true f, we cannot compute the risk.

However, we can compute the empirical risk

$$R_{emp}(\hat{f}) = \mathbb{E}_S \left[\frac{1}{n} \sum_{i=1}^{n} |f(x_i) - \hat{f}_S(x_i)|^2 \right].$$

$$R(\hat{f}) \leq \underbrace{|R(\hat{f}) - R_{emp}(\hat{f})|}_{\text{(expected) generalization error}} + \underbrace{R_{emp}(\hat{f})}_{\text{(expected) learning error}}$$

The aim of the paper by Hardt is to provide upper bounds on the generalization error.

Formally, **Stability ⇒ Generalization** means the following.

Definition [ε-uniform stability] We say that the estimate $\hat{f}$ is ε-uniformly stable if for all data sets S, S' s.t. S and S' differ in at most one example, we have

$$\sup_x \left| \hat{f}_S(x) - \hat{f}_{S'}(x) \right| \leq \varepsilon .$$

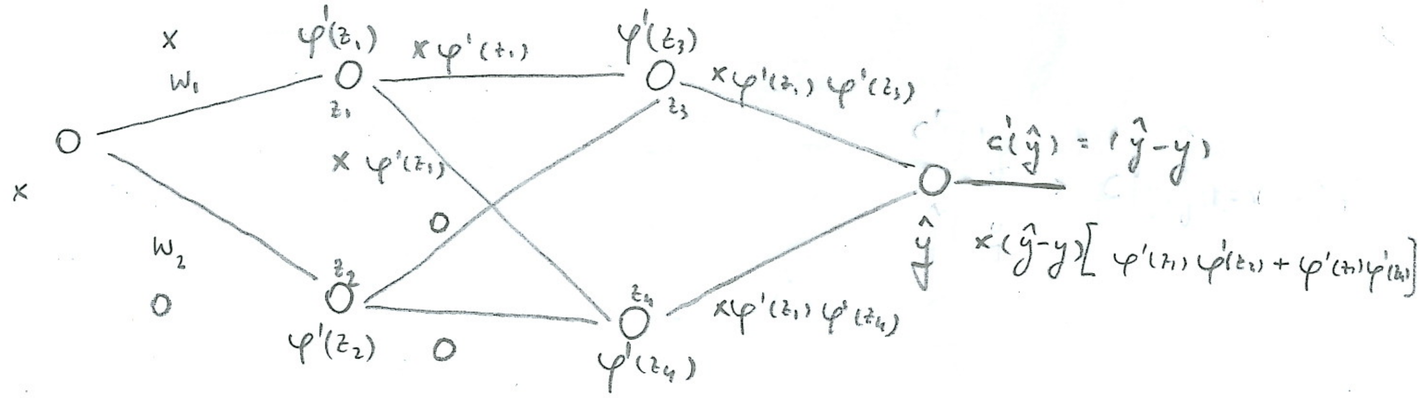
Theorem ["Stability and generalization", Elisseeff, 2002] Let $\hat{f}$ be ε-uniformly stable. Then,

$$\left| R - R_{emp} \right| \leq \varepsilon .$$

If an algorithm is uniformly stable, then its generalization error is small.

Backpropagation Algorithm

Toy Example



- all remaining weights are 1
- all biases are 0
- activation function at all hidden nodes is $\varphi(\cdot)$
- activation function at the output is $\varphi(x) = x$.
- cost function at the output is $C(\hat{y}) = \frac{1}{2}(\hat{y}-y)^2$

The most natural thing to do: "Forwardpropagation Algorithm"
It is a message-passing algorithm where messages flow on the edges of the network from input to output.

1) Propagate input and store derivatives at operating point.

Suppose we want to compute $\frac{\partial C}{\partial w_1}$.

2) Initialization: initialize the message on the edge corresponding to w_1 with its input. All edges of the same layer are initialized to 0.

3) Message passing rule. The output of each node is the sum of the inputs times the value stored in the node. The output is propagated to all outgoing edges.
Stop when reach the output.

$$\frac{\partial C}{\partial w_1} = (\hat{y}-y)\left[\varphi'(z_1)\varphi'(z_3) + \varphi'(z_1)\varphi'(z_4)\right]x$$

So far so good... Why then backpropagation?

DOUBLE-CHECK

$$C(w_1) = \frac{1}{2}(\hat{y}-y)^2$$

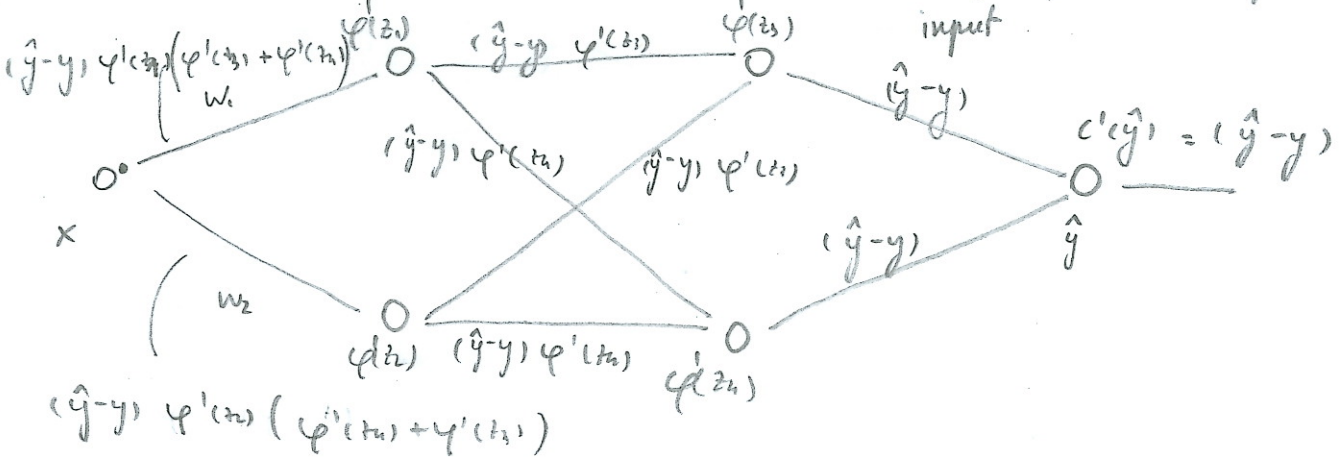
$\hat{y} = \varphi(z_3) + \varphi(z_4)$, $z_3 = \varphi(z_1) + \varphi(z_2)$, $z_1 = w_1 x$

The problem is that if now you want to compute $\frac{\partial C}{\partial w_2}$ you have to repeat 2) and 3) again. (4)

We would like to obtain all derivatives in 1 shot.

Back propagation :

Message passing algorithm where messages flow on the edges of the net from output to input



1) Same as before

2) Initialization : start from output and initialize edges to the value in the output node

3) Message passing rule. The input of each node is the sum of the outputs times the value stored in the node. The input is propagated to all incoming edges.

Stop when reach the input of the edge you are interested in.

Results.

$$\frac{\partial C}{\partial w_1} = (\hat{y}-y)\,\varphi'(z_2)\,(\varphi'(z_3)+\varphi'(z_4))\,x$$

$$\frac{\partial C}{\partial w_2} = (\hat{y}-y)\,\varphi'(z_2)\,(\varphi'(z_3)+\varphi'(z_4))\,x$$

Some Practical Tricks

→ Preprocess your data

1) 0 - mean and 1 - variance

2) Dimensionality reduction (e.g., PCA).

Principal Component Analysis

- Compute empirical covariance matrix of the data

$$S = \sum_{i=1}^{n} x(i) \; x^T(i)$$

$x(i) \in \mathbb{R}^{d \times 1}$, $x^T(i) \in \mathbb{R}^{1 \times d}$

- Compute eigenvalues and eigenvectors of S

- Project data onto d' directions (eigenvectors) associated to the largest eigenvalues. Usually ppl want to retain 75% - 90% of energy i.e. d' is chosen s.t. the sum of the d' largest eigenvalues is = to 75% - 90% of the sum of all eigenvalues.

→ Different cost function (cross entropy instead of MSE)

Advantage: avoids learning slow-down.

$$C(y, \hat{y}) = -\frac{1}{n} \sum_{i,k} y_i(k) \ln \hat{y}_i(k) + (1 - y_i(k)) \ln(1 - \hat{y}_i(k))$$

KL distance from $y(k)$ to $\hat{y}(k)$.

→ Softmax - different activation function at the output layer

So far $z_j^L = \sum_k w_{jk}^L h_k^{L-1} + b_j^L$ ...

$h_j^L = \varphi(z_j^L)$

Softmax layer means that

$$h_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}}$$

- Now $\sum_j h_j^L = 1$ and they can be interpreted as probabilities

Typically softmax is associated with log-likelihood cost function

$$C \equiv -\ln h_y^L$$

Perfect classification means $h_y^L = 1$ and $h_i^L = 0$ for $i \neq y$

so $C \equiv 0$ and it makes sense!

- With log-likehood cost function also no learning slowdown (as with cross-entropy cost function + sigmoid layer)

→ Split training set into actual training + validation set.

Use actual training to learn the weights and validation to test the results. This is useful to avoid overfitting and to choose the hyperparameters of the network (resp. size of gradient descent, # hidden nodes, # epochs for which one has to learn).

→ Regularization.

L_2 regularized - cross entropy

$$C = \overbrace{-\frac{1}{n}\sum_{i,x}\left(y_i(x)\ln\hat{y}_i(x) + (1-y_i(x))\ln(1-\hat{y}_i(x))\right)}^{C_0} + \frac{\lambda}{2n}\sum_w w^2$$

regularization parameter

L_1 regularization

$$C = C_0 + \frac{\lambda}{n}\sum_w |w|$$

→ Artificial expansion of training data. Pick all MNIST training images and

(i) rotate them by $5°, 10°, 15°, -5°, -10°, -15°$

(ii) translate them by 1 pixel on rx, ex, up, down.

→ Weight initialization

Initialize the weights that are given as input to some neuron with $N(0, 1/n_{in})$ where n_{in} is the number of input weights of the neuron.

In this way, the neuron will have an input with variance 1 and will not immediately saturate with good probability.

$$\sum_{k=1}^{n_{in}} w_{jk} \sim N(0,1)$$

$$\sim N\left(0, \frac{1}{n_{in}}\right)$$