Chapter 11. Training Deep Neural Nets

Difficulties of large deep neural network:

- *Vanishing gradients problem* (or *exploding gradients problem*) makes lower layers very hard to train.
- Training is extremely slow.
- Risk overfitting.

Vanishing/Exploding Gradients Problems

Vanishing gradients problem: gradients often get smaller and smaller as the algorithm progresses down to the lower layers. As a result, the Gradient Descent update leaves the lower layer connection weights virtually unchanged, and training never converges to a good solution.

Exploding gradient problem: the gradients can grow bigger and bigger, so many layers get insanely large weight updates and the algorithm diverges. (mostly encountered in recurrent neural networks)

More generally, deep neural networks suffer from unstable gradients; different layers may learn at widely different speeds.

"Understanding the Difficulty of Training Deep Feedforward Neural Networks" by Xavier Glorot and Yoshua Bengio found a few suspects, including the combination of the popular logistic sigmoid activation function and the weight initialization technique that was most popular at the time, namely random initialization using a normal distribution with a mean of 0 and a standard deviation of 1. In short, they showed that with this activation function and this initialization scheme, the variance of the outputs of each layer is much greater than the variance of its inputs.

Xavier and He Initialization

For the signal to flow properly, the variance of the outputs of each layer is need to be equal to the variance of its inputs, and we also need the gradients to have equal variance before and after flowing through a layer in the reverse direction. The connection weights must be initialized randomly as

Xavier (Glorot) initialization (when using the logistic activation function)

> Normal distribution with mean 0 and standard deviation $\sigma = \sqrt{\frac{2}{n_{in}+n_{out}}}$
>
> or a uniform distribution between $-r$ and $+r$, with $r = \sqrt{\frac{6}{n_{in}+n_{out}}}$

where n_{in} and n_{out} are the number of input and output connections for the layer whose weights are being initialized (also called *fan-in* and *fan-out*).

When $n_{in} \approx n_{out}$,

$$\sigma = \frac{1}{\sqrt{n}} \quad or \quad r = \frac{\sqrt{3}}{\sqrt{n}}$$

The initialization strategy for the ReLU activation function (and its variants, including the ELU activation) is sometimes called *He initialization*.

Activation function	Uniform distribution [–r, r]	Normal distribution
Logistic	$r = \sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = \sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$
Hyperbolic tangent	$r = 4\sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = 4\sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$
ReLU (and its variants)	$r = \sqrt{2}\sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = \sqrt{2}\sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$

Table 11-1. Initialization parameters for each type of activation function

Use He initialization

In [2]:
```
import tensorflow as tf

tf.reset_default_graph()

n_inputs = 28 * 28  # MNIST
n_hidden1 = 300

X = tf.placeholder(tf.float32, shape=(None, n_inputs), name="X")

he_init = tf.variance_scaling_initializer()
hidden1 = tf.layers.dense(X, n_hidden1, activation=tf.nn.relu,
                          kernel_initializer=he_init, name="hidden1")
```

NOTE

> He initialization considers only the fan-in, not the average between fan-in and fan-out like in Xavier initialization. This is also the default for the `variance_scaling_initializer()` function, but you can change this by setting the argument `mode="FAN_AVG"` .

Nonsaturating Activation Functions

ReLU activation function is much better than sigmoid activation function, because it does not saturate for positive values (and also because it is quite fast to compute).