Vanishing gradients

Glenn Bruns
CSUMB

Much material in this deck from Géron, Hands-on Machine Learning with Scikit-Learn and TensorFlow

Learning outcomes

After this lecture you should be able to:

- diagnose and address the vanishing and exploding gradients problem

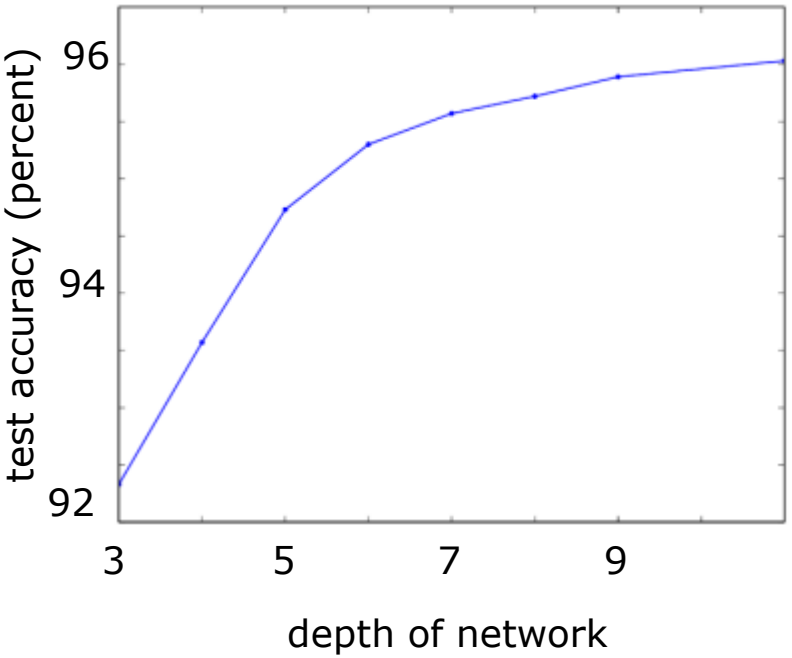
Deep neural nets

"Deep networks … can model complex functions using exponentially fewer neurons than shallow nets" (Géron)

"Empirically, greater depth does seem to result in better generalization for a wide variety of tasks" (Goodfellow et al)

"Deeper networks are often able to use far fewer units per layer and have far fewer parameters … but they also tend to be harder to optimize" (Goodfellow et al)

effect of depth on task of finding multidigit numbers in photos of addresses



You may need to train a net with 10 or more layers, and hundreds of neurons per layer

Problems in training deep neural nets

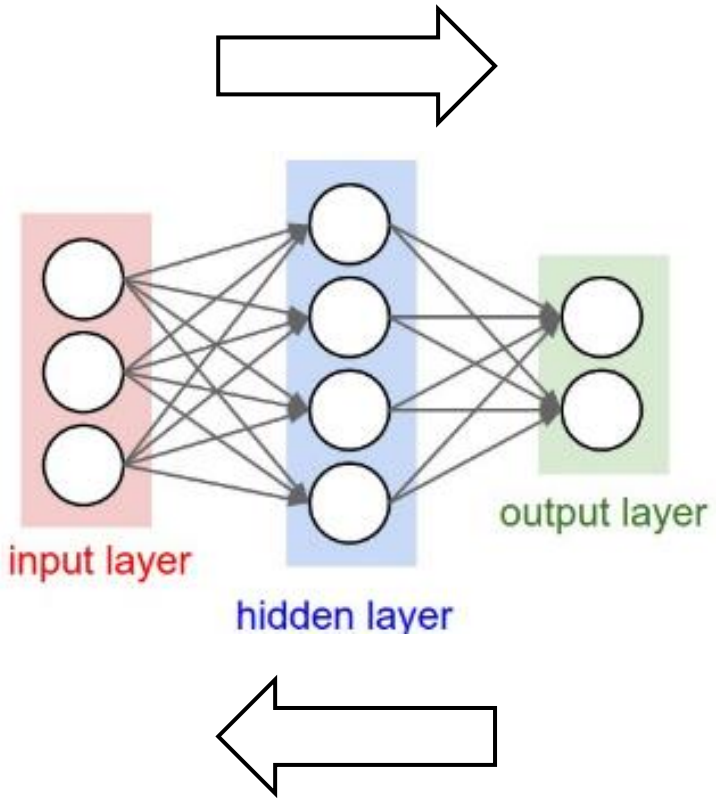
1. vanishing gradients, exploding gradients
   - makes lower layers hard to train
2. training can take a long time
3. with millions of parameters, overfitting is a big risk

Vanishing/Exploding gradients

In backpropagation:

- a forward pass computed outputs from inputs
- a backward pass computes gradients

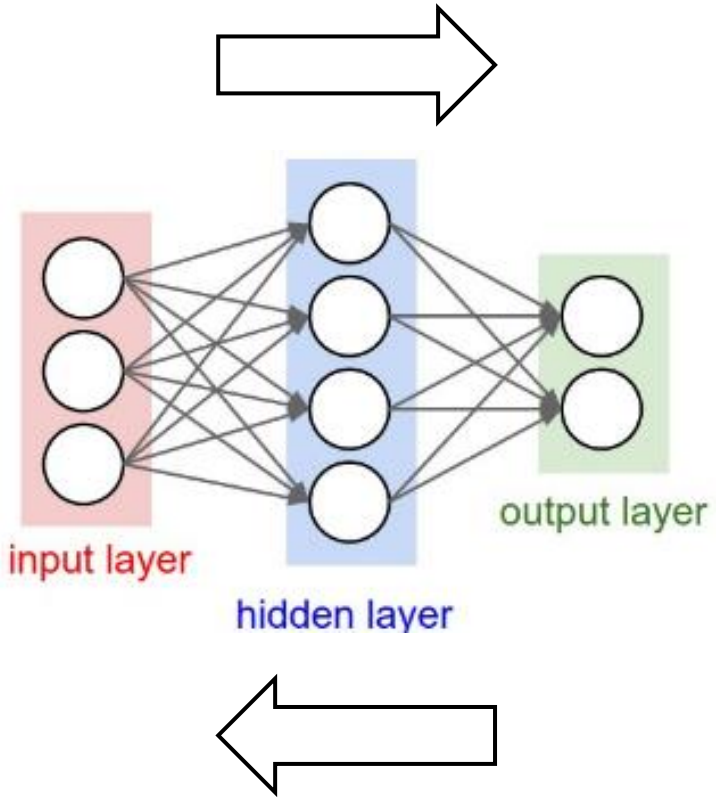
Gradients are then used to update parameters in a gradient descent step



Vanishing/Exploding gradients (cont'd.)

Vanishing gradients: gradients get smaller and smaller during backward pass

Exploding gradients: gradients get larger and larger during backward pass



Causes of the problem

"Understanding the Difficulty of Training Deep Feedforward Neural Networks" (Glorot and Bengio, 2010), diagnosed the issue.

1. Use of the sigmoid activation function
2. Method used to initialize network parameters
   - random initialization with mean 0, std dev 1

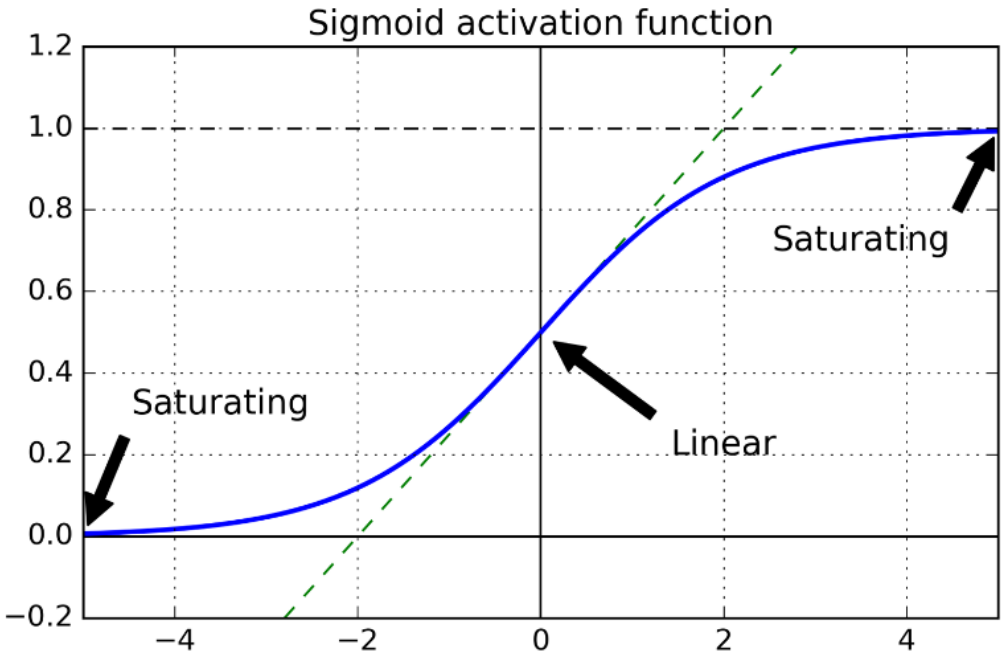
Idea:

- with 1 and 2 above, the outputs of a layer have greater variance than the inputs of a layer
- activation function then saturates in top layers

Saturating sigmoid activation function

When the function is saturated, the slope is almost zero.

No gradient to propagate back through the network.



Solution idea 1: better initialization

Xavier initialization for sigmoid activation:

Normal dist. with mean 0, and:

$$\sigma = \sqrt{\frac{2}{n_{inputs} + n_{outputs}}}$$

or

Uniform dist. in [-r, r], with

$$r = \sqrt{\frac{6}{n_{inputs} + n_{outputs}}}$$

Suggested initializations for other activation functions:

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)	$r = \sqrt{2}\sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = \sqrt{2}\sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$

He initialization

Idea 2: dump the sigmoid activation

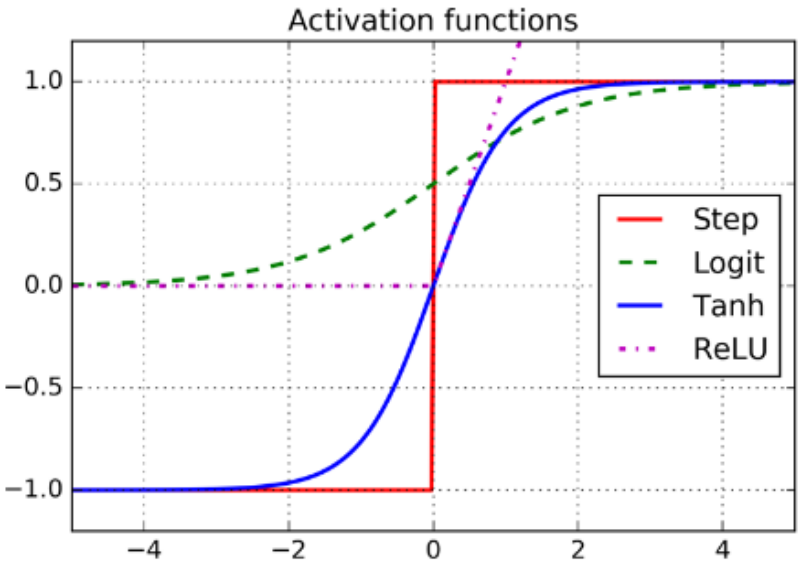
Replacement candidate 1:
ReLU activation function

pros:

- doesn't saturate for positive values
- fast to compute

cons:

- dying ReLU problem: many neurons in network stop outputting anything other than 0
- if weighted sum of neuron's inputs is negative, ReLU gives 0
- once this happens, it tends to stay that way

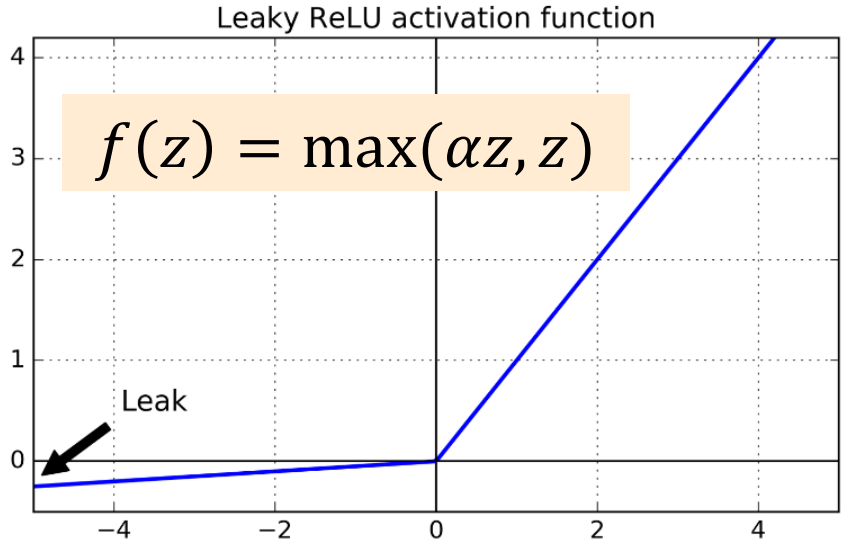


Dump the sigmoid (cont'd.)

Replacement candidate 2: leaky ReLU

pros:

- unlike a ReLU, it can never "die"
- a recent paper claims it always outperforms plain ReLU



Other cousin candidates:

- randomized leaky ReLU
  - α is picked randomly during training
- parametric leaky ReLU
  - α is learned during training

α is a hyperparameter; typically set to 0.01

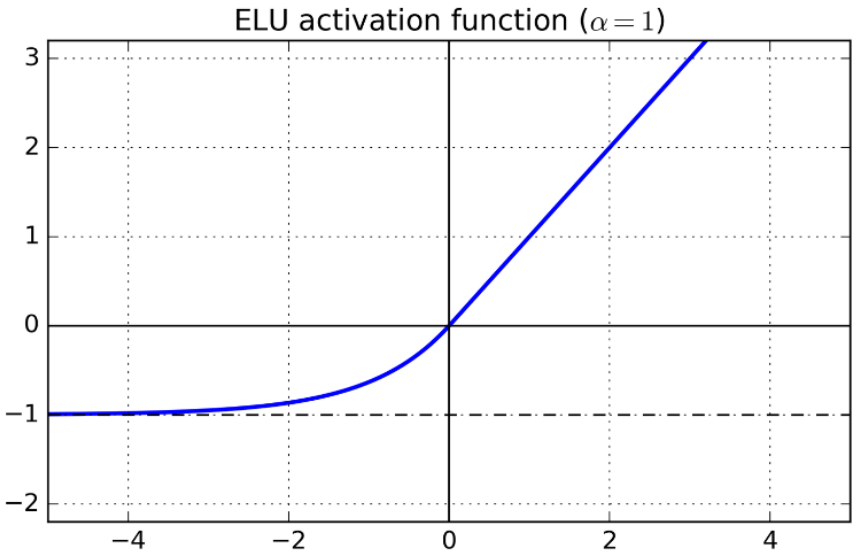
values of up to 0.2 may work even better

Dump the sigmoid (cont'd.)

Replacement candidate 3: exponential linear unit (ELU)

pros:

- outperformed all ReLU variants in a recent study
- training time reduced; performance better
- function is smooth everywhere, helping with gradient descent
- negative value when z < 0, avoids vanishing gradients
- non-zero gradient when z < 0, avoids dying units



$$\mathrm{ELU}_\alpha(z) = \begin{cases} \alpha(\exp(z) - 1) & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$

hyperparameter usually set to 1

cons: slower to compute than ReLU and its variants

Dumping the sigmoid in TensorFlow

ELU: (when calling fully_connected)

```
hidden1 = fully_connected(X, n_hidden1, activation_fn=tf.nn.elu)
```

leaky ReLU: (do-it-yourself)

```
def leaky_relu(z, name=None):
    return tf.maximum(0.01 * z, z, name=name)

hidden1 = fully_connected(X, n_hidden1, activation_fn=leaky_relu)
```

Solution idea 3: Batch normalization

He initialization plus better activation function especially helps with vanishing/exploding gradients during beginning of training.

A newer technique (2015) is batch normalization.

Idea:

- add operation just before activation function
- this operation lets the model learn the optimal scale and mean of the inputs for each layer
- inventors claim many benefits, including reduction in vanishing gradients, and better performance

See textbook for using batch normalization in TensorFlow

Solution idea 4: Gradient clipping

Addresses the exploding gradients problem.

Idea: clip gradients during backpropagation so they never exceed a threshold.

Geron says batch normalization is now mostly preferred to gradient clipping.

See textbook for using gradient clipping in TensorFlow

Summary

What is the vanishing and exploding gradients problem?

How to address it?

- Xavier and He initialization of network parameters
- Nonsaturating activation functions
- Batch normalization
- Gradient clipping