Deep Learning: Lecture 7

Alexander Schönhuth

UU

November 20, 2019

Backpropagation for CNNs

Convolutional Backpropagation

Notation

- In fully connected layers we had [w^l_{jk} is for connecting the k-th neuron in layer $l-1$ with the j-th neuron in layer l]

$$\delta^l_j = \frac{\partial C}{\partial z^l_j} \quad \text{where} \quad z^l_j = \sum_k w^l_{jk} a^{l-1}_k + b^l_j \tag{1}$$

  and $a^l_j = \sigma(z^l_j)$ (where σ is any activation function)

- For sake of simplicity, we assume that there is only one hidden sublayer (corresponding to one feature map, or one channel) at each level of depth, so only one convolutional filter per level of depth required.

Convolutional Backpropagation

Notation

- In fully connected layers we had [w^l_{jk} is for connecting the k-th neuron in layer $l-1$ with the j-th neuron in layer l]

$$\delta^l_j = \frac{\partial C}{\partial z^l_j} \quad \text{where} \quad z^l_j = \sum_k w^l_{jk} a^{l-1}_k + b^l_j \tag{1}$$

  and $a^l_j = \sigma(z^l_j)$ (where σ is any activation function)

- For sake of simplicity, we assume that there is only one hidden sublayer (corresponding to one feature map, or one channel) at each level of depth, so only one convolutional filter per level of depth required.

Convolutional Backpropagation

Notation

- We index neurons using two coordinates, so $z^l_{x,y}$ is the input for the x, y-th neuron of the hidden layer at level of depth l.
- The $M \times M$ filter that connects neurons from level l with neurons at level $l+1$ has weights $w^{l+1}_{ab}, 1 \leq a, b \leq M$
- By applying the convolution operation [and neglecting the exact indexing in the following]

$$z^{l+1}_{x,y} = \sum_a \sum_b w^{l+1}_{ab} \sigma(z^l_{x-a,y-b}) + b^{l+1}_{x,y} \tag{2}$$

Convolutional Backpropagation

Notation

- We index neurons using two coordinates, so $z^l_{x,y}$ is the input for the x, y-th neuron of the hidden layer at level of depth l.
- The $M \times M$ filter that connects neurons from level l with neurons at level $l+1$ has weights $w^{l+1}_{ab}, 1 \leq a, b \leq M$
- By applying the convolution operation [and neglecting the exact indexing in the following]

$$z^{l+1}_{x,y} = \sum_a \sum_b w^{l+1}_{ab} \sigma(z^l_{x-a,y-b}) + b^{l+1}_{x,y} \tag{2}$$

Convolutional Backpropagation

- We compute

$$\delta^l_{x,y} = \frac{\partial C}{\partial z^l_{x,y}} = \sum_{x'} \sum_{y'} \frac{\partial C}{\partial z^{l+1}_{x',y'}} \frac{\partial z^{l+1}_{x',y'}}{\partial z^l_{x,y}} \tag{3}$$

- Moving on, we get

$$\begin{aligned} \frac{\partial C}{\partial z^l_{x,y}} &= \sum_{x'} \sum_{y'} \frac{\partial C}{\partial z^{l+1}_{x',y'}} \frac{\partial z^{l+1}_{x',y'}}{\partial z^l_{x,y}} \\ &= \sum_{x'} \sum_{y'} \delta^{l+1}_{x',y'} \frac{\partial (\sum_a \sum_b w^{l+1}_{ab} \sigma(z^l_{x'-a,y'-b}) + b^{l+1}_{x',y'})}{\partial z^l_{x,y}} \end{aligned} \tag{4}$$

Convolutional Backpropagation

- We compute

$$\delta^l_{x,y} = \frac{\partial C}{\partial z^l_{x,y}} = \sum_{x'} \sum_{y'} \frac{\partial C}{\partial z^{l+1}_{x',y'}} \frac{\partial z^{l+1}_{x',y'}}{\partial z^l_{x,y}} \tag{3}$$

- Moving on, we get

$$\begin{aligned} \frac{\partial C}{\partial z^l_{x,y}} &= \sum_{x'} \sum_{y'} \frac{\partial C}{\partial z^{l+1}_{x',y'}} \frac{\partial z^{l+1}_{x',y'}}{\partial z^l_{x,y}} \\ &= \sum_{x'} \sum_{y'} \delta^{l+1}_{x',y'} \frac{\partial(\sum_a \sum_b w^{l+1}_{ab} \sigma(z^l_{x'-a,y'-b}) + b^{l+1}_{x',y'})}{\partial z^l_{x,y}} \end{aligned} \tag{4}$$

Convolutional Backpropagation

- All terms in (4) where $x \neq x' - a$ or $y \neq y' - b$ are zero, so

$$\sum_{x'} \sum_{y'} \delta^{l+1}_{x',y'} \frac{\partial(\sum_a \sum_b w^{l+1}_{ab} \sigma(z^l_{x'-a,y'-b}) + b^{l+1}_{x',y'})}{\partial z^l_{x,y}} = \sum_{x'} \sum_{y'} \delta^{l+1}_{x',y'} w^{l+1}_{ab} \sigma'(z^l_{x,y}) \quad (5)$$

- Since now $x = x' - a$, $y = y' - b$, we have $a = x' - x$, $b = y' - y$, so

$$\sum_{x'} \sum_{y'} \delta^{l+1}_{x',y'} w^{l+1}_{ab} \sigma'(z^l_{x,y}) = \sum_{x'} \sum_{y'} \delta^{l+1}_{x',y'} w^{l+1}_{x'-x,y'-y} \sigma'(z^l_{x,y}) \quad (6)$$

Convolutional Backpropagation

- All terms in (4) where $x \neq x' - a$ or $y \neq y' - b$ are zero, so

$$\sum_{x'} \sum_{y'} \delta^{l+1}_{x',y'} \frac{\partial(\sum_a \sum_b w^{l+1}_{ab} \sigma(z^l_{x'-a,y'-b}) + b^{l+1}_{x',y'})}{\partial z^l_{x,y}} = \sum_{x'} \sum_{y'} \delta^{l+1}_{x',y'} w^{l+1}_{ab} \sigma'(z^l_{x,y}) \quad (5)$$

- Since now $x = x' - a$, $y = y' - b$, we have $a = x' - x$, $b = y' - y$, so

$$\sum_{x'} \sum_{y'} \delta^{l+1}_{x',y'} w^{l+1}_{ab} \sigma'(z^l_{x,y}) = \sum_{x'} \sum_{y'} \delta^{l+1}_{x',y'} w^{l+1}_{x'-x,y'-y} \sigma'(z^l_{x,y}) \quad (6)$$

Convolutional Backpropagation

- Summary:

$$\delta^l_{x,y} = \sum_{x'} \sum_{y'} \delta^{l+1}_{x',y'} w^{l+1}_{x'-x,y'-y} \sigma'(z^l_{x,y}) \tag{7}$$

- A closer look reveals this as a convolution operation in its own right, applying the filter

$$\sigma'(z^l_{x,y}) \cdot \begin{pmatrix} w_{MM} & \cdots & w_{M1} \\ \vdots & \ddots & \vdots \\ w_{1M} & \cdots & w_{11} \end{pmatrix} \tag{8}$$

to the $l+1$-*th layer of gradients* $\delta^{l+1}_{x',y'}$, for computing values $\delta^l_{x,y}$ of the l-th layer of gradients.

- *Note* that the weights of the original filter have been rotated by 180°
- For further illustrations, see `https://medium.com/@2017csm1006/forward-and-backpropagation-in-convolutional-neural-network-4dfa9`
Note that notation differs (error E there is cost C here, X there is z here, and F are the weights w here, and O is a here, and there is no bias)

Convolutional Backpropagation

- Summary:

$$\delta^l_{x,y} = \sum_{x'} \sum_{y'} \delta^{l+1}_{x',y'} w^{l+1}_{x'-x,y'-y} \sigma'(z^l_{x,y}) \tag{7}$$

- A closer look reveals this as a convolution operation in its own right, applying the filter

$$\sigma'(z^l_{x,y}) \cdot \begin{pmatrix} w_{MM} & \cdots & w_{M1} \\ \vdots & \ddots & \vdots \\ w_{1M} & \cdots & w_{11} \end{pmatrix} \tag{8}$$

  to the $l+1$-*th layer of gradients* $\delta^{l+1}_{x',y'}$, for computing values $\delta^l_{x,y}$ of the l-th layer of gradients.

- *Note* that the weights of the original filter have been rotated by 180°

- For further illustrations, see `https://medium.com/@2017csm1006/forward-and-backpropagation-in-convolutional-neural-network-4dfa9`
  Note that notation differs (error E there is cost C here, X there is z here, and F are the weights w here, and O is a here, and there is no bias)

Convolutional Backpropagation

- Summary:
$$\delta^l_{x,y} = \sum_{x'} \sum_{y'} \delta^{l+1}_{x',y'} w^{l+1}_{x'-x,y'-y} \sigma'(z^l_{x,y}) \tag{7}$$
- A closer look reveals this as a convolution operation in its own right, applying the filter
$$\sigma'(z^l_{x,y}) \cdot \begin{pmatrix} w_{MM} & \cdots & w_{M1} \\ \vdots & \ddots & \vdots \\ w_{1M} & \cdots & w_{11} \end{pmatrix} \tag{8}$$
to the $l+1$-*th layer of gradients* $\delta^{l+1}_{x',y'}$, for computing values $\delta^l_{x,y}$ of the l-th layer of gradients.
- *Note* that the weights of the original filter have been rotated by 180°
- For further illustrations, see `https://medium.com/@2017csm1006/forward-and-backpropagation-in-convolutional-neural-network-4dfa9` Note that notation differs (error E there is cost C here, X there is z here, and F are the weights w here, and O is a here, and there is no bias)

Convolutional Backpropagation

- Summary:

$$\delta^l_{x,y} = \sum_{x'} \sum_{y'} \delta^{l+1}_{x',y'} w^{l+1}_{x'-x,y'-y} \sigma'(z^l_{x,y}) \tag{7}$$

- A closer look reveals this as a convolution operation in its own right, applying the filter

$$\sigma'(z^l_{x,y}) \cdot \begin{pmatrix} w_{MM} & \cdots & w_{M1} \\ \vdots & \ddots & \vdots \\ w_{1M} & \cdots & w_{11} \end{pmatrix} \tag{8}$$

  to the $l+1$-*th layer of gradients* $\delta^{l+1}_{x',y'}$, for computing values $\delta^l_{x,y}$ of the l-th layer of gradients.

- *Note* that the weights of the original filter have been rotated by 180°

- For further illustrations, see `https://medium.com/@2017csm1006/forward-and-backpropagation-in-convolutional-neural-network-4dfa9`
  Note that notation differs (error E there is cost C here, X there is z here, and F are the weights w here, and O is a here, and there is no bias)

Training Variations

Hessian Technique

- As usual, let $\mathbf{w} = (w_1, w_2, \ldots)$ be NN parameters (weights in particular), and C a cost function.
- By *Taylor's theorem*, we can write

$$\begin{aligned} C(w + \delta w) = C(w) + \sum_j \frac{\partial C}{\partial w_j} \delta w_j \\ + \frac{1}{2} \sum_{jk} \delta w_j H_{jk} \delta w_k + \ldots \text{ terms of higher order} \end{aligned} \tag{9}$$

where H is the *Hessian matrix*, given by

$$H_{jk} = \frac{\partial^2 C}{\partial w_j \partial w_k} \tag{10}$$

Hessian Technique

- As usual, let $\mathbf{w} = (w_1, w_2, \ldots)$ be NN parameters (weights in particular), and C a cost function.
- By *Taylor's theorem*, we can write

$$\begin{aligned} C(w + \delta w) = C(w) &+ \sum_j \frac{\partial C}{\partial w_j} \delta w_j \\ &+ \frac{1}{2} \sum_{jk} \delta w_j H_{jk} \delta w_k + \ldots \text{ terms of higher order} \end{aligned} \tag{9}$$

  where H is the *Hessian matrix*, given by

$$H_{jk} = \frac{\partial^2 C}{\partial w_j \partial w_k} \tag{10}$$

Hessian Technique

- Then (12) can further be written as

$$C(w + \delta w) = C(w) + \nabla C \delta w + \frac{1}{2} \delta w^T H \delta w + \ldots \tag{11}$$

- Discarding all terms of order greater than 2, we obtain

$$C(w + \delta w) \approx C(w) + \nabla C \delta w + \frac{1}{2} \delta w^T H \delta w \tag{12}$$

- The right hand side of (12) can be minimized by choosing

$$\delta w = -H^{-1} \nabla C \tag{13}$$

Hessian Technique

- Then (12) can further be written as

$$C(w + \delta w) = C(w) + \nabla C \delta w + \frac{1}{2} \delta w^T H \delta w + \ldots \tag{11}$$

- Discarding all terms of order greater than 2, we obtain

$$C(w + \delta w) \approx C(w) + \nabla C \delta w + \frac{1}{2} \delta w^T H \delta w \tag{12}$$

- The right hand side of (12) can be minimized by choosing

$$\delta w = -H^{-1} \nabla C \tag{13}$$

Hessian Technique

- Then (12) can further be written as

$$C(w+\delta w) = C(w) + \nabla C \delta w + \frac{1}{2}\delta w^T H \delta w + \dots \tag{11}$$

- Discarding all terms of order greater than 2, we obtain

$$C(w+\delta w) \approx C(w) + \nabla C \delta w + \frac{1}{2}\delta w^T H \delta w \tag{12}$$

- The right hand side of (12) can be minimized by choosing

$$\delta w = -H^{-1}\nabla C \tag{13}$$

HESSIAN TECHNIQUE

This suggests the following algorithm for updating weights in a gradient descent like scheme:

1. Choose a starting point **w**
2. Update **w** to $\mathbf{w}' = w - \eta H^{-1} \nabla C$, where H and ∇C are computed at **w**
3. Iterate ... (until appropriate criteria, like convergence, are met)

Hessian Technique

Advantages and Disadvantages

- The Hessian technique takes into account *how fast the gradient changes*
- Theoretical and empirical evidence says that less iterations are needed
- Issue is the size of H, which of size N^2 if N is the number of parameters
- ☞ Note that there could be $N = 10^7$ many parameters
- *Summary*:
  - The Hessian technique can often not be applied because computations are too expensive
  - However, it provided inspiration for other techniques

Hessian Technique

Advantages and Disadvantages

- The Hessian technique takes into account *how fast the gradient changes*
- Theoretical and empirical evidence says that less iterations are needed
- Issue is the size of H, which of size N^2 if N is the number of parameters
- ☞ Note that there could be $N = 10^7$ many parameters
- *Summary*:
  - The Hessian technique can often not be applied because computations are too expensive
  - However, it provided inspiration for other techniques

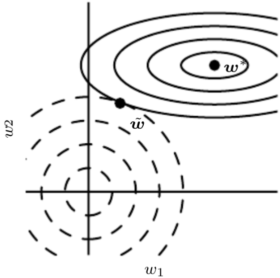
Hessian Technique

Advantages and Disadvantages

- The Hessian technique takes into account *how fast the gradient changes*
- Theoretical and empirical evidence says that less iterations are needed
- Issue is the size of H, which of size N^2 if N is the number of parameters
- ☞ Note that there could be $N = 10^7$ many parameters
- *Summary*:
  - The Hessian technique can often not be applied because computations are too expensive
  - However, it provided inspiration for other techniques

Regularization Revisited

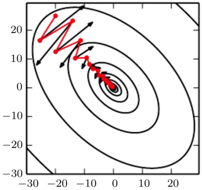
Motivation



- *Reminder*: L2 regularization shrinks weights along Hessian eigenvectors
- The ball then moves as being pulled by the origin $(0, 0)$ in the landscape induced by the eigenvectors of the Hessian

Momentum-based Gradient Descent

Motivation

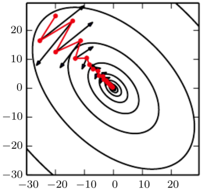


Black: gradients at each step, zig-zagging through the "valley"

- *Motivation*: Going back and forth, without making progress, during (stochastic) gradient descent
- *Reasons*:
  - Poorly conditioned Hessian matrix because of "valleys" (see Figure)
  - Variance between batches
  - High curvature in general
  - Noisy gradients

Momentum-based Gradient Descent

Motivation

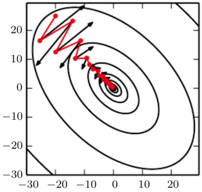


Black: gradients at each step, zig-zagging through the "valley"

- *Motivation*: Going back and forth, without making progress, during (stochastic) gradient descent
- *Reasons*:
  - Poorly conditioned Hessian matrix because of "valleys" (see Figure)
  - Variance between batches
  - High curvature in general
  - Noisy gradients

Momentum-based Gradient Descent

Motivation

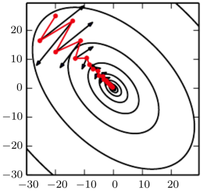


Black: gradients at each step, zig-zagging through the "valley"

- *Motivation*: Going back and forth, without making progress, during (stochastic) gradient descent
- *Reasons*:
  - Poorly conditioned Hessian matrix because of "valleys" (see Figure)
  - Variance between batches
  - High curvature in general
  - Noisy gradients

Momentum-based Gradient Descent

Motivation

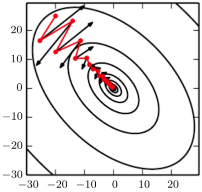


Black: gradients at each step, zig-zagging through the "valley"

- *Motivation*: Going back and forth, without making progress, during (stochastic) gradient descent
- *Reasons*:
  - Poorly conditioned Hessian matrix because of "valleys" (see Figure)
  - Variance between batches
  - High curvature in general
  - Noisy gradients

Momentum-based Gradient Descent

Motivation

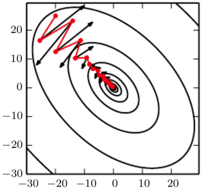


Black: gradients at each step, zig-zagging through the "valley"

- *Motivation*: Going back and forth, without making progress, during (stochastic) gradient descent
- *Reasons*:
  - Poorly conditioned Hessian matrix because of "valleys" (see Figure)
  - Variance between batches
  - High curvature in general
  - Noisy gradients

Momentum-based Gradient Descent

Solution

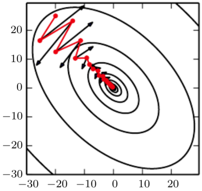


Red: averaged gradients, less zig-zagging

- Keep track of earlier gradients and take the average

Momentum-based Gradient Descent

Solution



Red: averaged gradients, less zig-zagging

- Formally, maintain *velocity* $\mathbf{v}$ in addition to parameters $\mathbf{w}$ themselves
- In each iteration, update
  - velocity, where α is the *momentum hyperparameter*

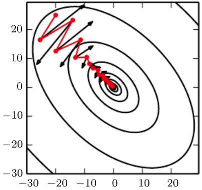
$$\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \nabla_{\mathbf{w}} C \tag{14}$$

  - parameters:

$$\mathbf{w} \leftarrow \mathbf{w} + \mathbf{v} \tag{15}$$

Momentum-based Gradient Descent

Solution

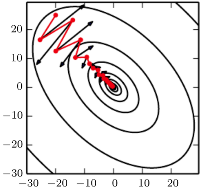


Red: averaged gradients, less zig-zagging

- In momentum based gradient descent, **w** is an exponentially decaying average over gradients
- Variant: *Nesterov's accelerated gradient technique*
- See also `http://neuralnetworksanddeeplearning.com/chap3.html` (Nielsen, chapter 3) for more details

Momentum-based Gradient Descent

Solution



Red: averaged gradients, less zig-zagging

- In momentum based gradient descent, **w** is an exponentially decaying average over gradients
- Variant: *Nesterov's accelerated gradient technique*
- See also `http://neuralnetworksanddeeplearning.com/chap3.html` (Nielsen, chapter 3) for more details

Alternative Optimization

Further Reading

- *Alternative methods*:
  - *Conjugate gradient descent*
  - *BFGS (Broyden-Fletcher-Goldfarb-Shanno) method*
  - *L-BFGS (Limited-memory-BFGS) method*
- *Illustrations / Literature*:
  - Bengio's deep learning book: http://www.deeplearningbook.org/contents/optimization.html
  - "Efficient BackProp", Y. LeCun, L. Bottou, G. Orr, K.-R. Müller, 1998, see http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf
  - "On the importance of initialization and momentum in deep learning", I. Sutskever, J. Martens, G. Dahl and G. Hinton, 2012, http://www.cs.toronto.edu/~hinton/absps/momentum.pdf

Alternative Optimization

Further Reading

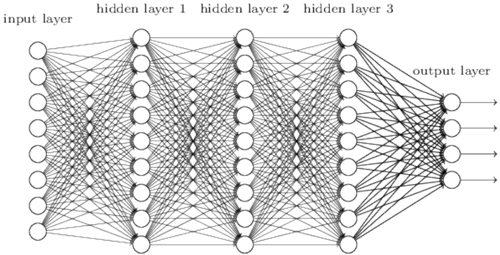
- *Alternative methods*:
  - *Conjugate gradient descent*
  - *BFGS (Broyden-Fletcher-Goldfarb-Shanno) method*
  - *L-BFGS (Limited-memory-BFGS) method*
- *Illustrations / Literature*:
  - Bengio's deep learning book: `http://www.deeplearningbook.org/contents/optimization.html`
  - "Efficient BackProp", Y. LeCun, L. Bottou, G. Orr, K.-R. Müller, 1998, see `http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf`
  - "On the importance of initialization and momentum in deep learning", I. Sutskever, J. Martens, G. Dahl and G. Hinton, 2012, `http://www.cs.toronto.edu/~hinton/absps/momentum.pdf`

The Vanishing Gradient

Why is Deep Learning Tough?

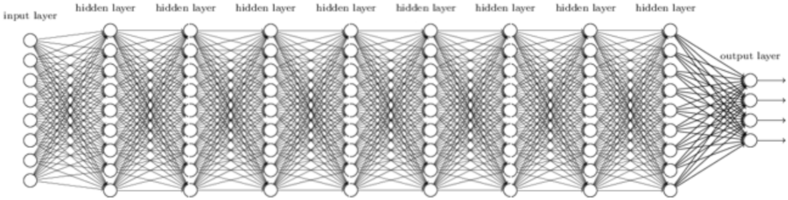
- Deep is supposed to better than shallow
  - Less hidden nodes necessary to approximate the true functional relationship
  - See the "Universal Approximation Theorem" by Montufar, 2014
  - See further "Learning Deep Architectures", Bengio, 2009, `http://www.iro.umontreal.ca/~bengioy/papers/ftml_book.pdf` for a more informal discussion
- *However*: On increasing depth in a naive way, performance usually drops
- What is going wrong?

WHY IS DEEP LEARNING TOUGH?

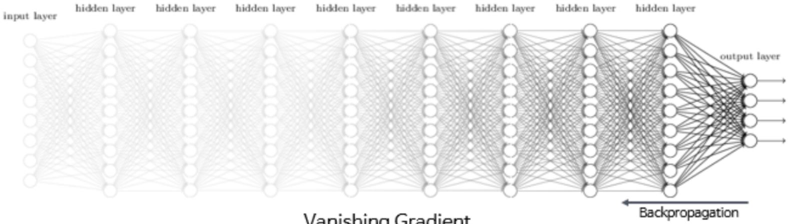


Training Deeper NN's: either the earlier layers (more common; here hidden layer 1) or the later layers (here: hidden layer 3) do not train well

The Vanishing Gradient Problem



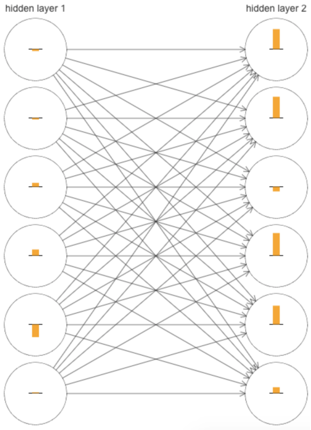
Deep Neural Network



Vanishing Gradient

Most commonly: gradients converge to zero in earlier layers

The Vanishing Gradient Problem



- Changes larger in later hidden layer
- Learning works better in later layers
- Are neurons likely to learn at different rates in different layers in general?

Yellow bars: $\frac{\partial C}{\partial b}$ for each hidden neuron

The Vanishing Gradient Problem

- Let b_j^l be the j-th bias in layer l, and $\frac{\partial C}{\partial b_j^l}$ be the respective partial derivative of the cost C.
- Let
$$\nabla_{\mathbf{b}^l}^{(l)} C := \left(\frac{\partial C}{\partial b_1^l}, \ldots, \frac{\partial C}{\partial b_{d(l)}^l}\right) \tag{16}$$
- Then, in the example from the slide before:
$$||\nabla_{\mathbf{b}}^{(1)} C|| = 0.07 \text{ and } ||\nabla_{\mathbf{b}}^{(2)} C|| = 0.31 \tag{17}$$

The Vanishing Gradient Problem

- Let b^l_j be the j-th bias in layer l, and $\frac{\partial C}{\partial b^l_j}$ be the respective partial derivative of the cost C.
- Let
$$\nabla^{(l)}_{\mathbf{b}^l} C := \left(\frac{\partial C}{\partial b^l_1}, \ldots, \frac{\partial C}{\partial b^l_{d(l)}}\right) \tag{16}$$
- Then, in the example from the slide before:
$$||\nabla^{(1)}_{\mathbf{b}} C|| = 0.07 \;\text{ and }\; ||\nabla^{(2)}_{\mathbf{b}} C|| = 0.31 \tag{17}$$

The Vanishing Gradient Problem

- Then, in this example,

$$||\nabla_{\mathbf{b}}^{(1)}C|| = 0.07 \ \text{ and } \ ||\nabla_{\mathbf{b}}^{(2)}C|| = 0.31 \tag{18}$$

- Formal quantification shows: learning faster in hidden layer 2.
- When running the identical training task (MNIST), we obtain
  - $||\nabla_{\mathbf{b}}^{(1)}C|| = 0.012, ||\nabla_{\mathbf{b}}^{(2)}C|| = 0.06, ||\nabla_{\mathbf{b}}^{(3)}C|| = 0.283$ for three hidden layers
  - $||\nabla_{\mathbf{b}}^{(1)}C|| = 0.003, ||\nabla_{\mathbf{b}}^{(2)}C|| = 0.017, ||\nabla_{\mathbf{b}}^{(3)}C|| = 0.07, ||\nabla_{\mathbf{b}}^{(4)}C|| =$ 0.285 for four hidden layers
  - and so on...

The Vanishing Gradient Problem

- Then, in this example,

$$||\nabla_{\mathbf{b}}^{(1)}C|| = 0.07 \text{ and } ||\nabla_{\mathbf{b}}^{(2)}C|| = 0.31 \tag{18}$$

- Formal quantification shows: learning faster in hidden layer 2.
- When running the identical training task (MNIST), we obtain
  - $||\nabla_{\mathbf{b}}^{(1)}C|| = 0.012, ||\nabla_{\mathbf{b}}^{(2)}C|| = 0.06, ||\nabla_{\mathbf{b}}^{(3)}C|| = 0.283$ for three hidden layers
  - $||\nabla_{\mathbf{b}}^{(1)}C|| = 0.003, ||\nabla_{\mathbf{b}}^{(2)}C|| = 0.017, ||\nabla_{\mathbf{b}}^{(3)}C|| = 0.07, ||\nabla_{\mathbf{b}}^{(4)}C|| = 0.285$ for four hidden layers
  - and so on...

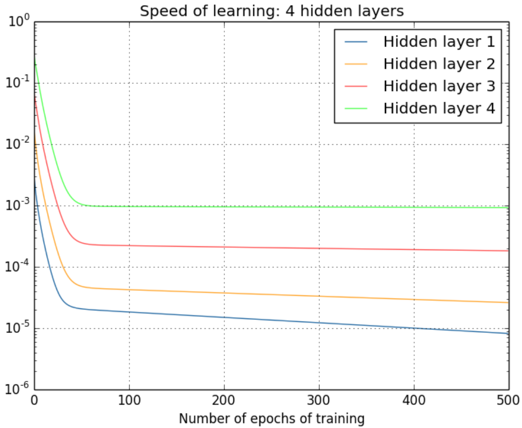
The Vanishing Gradient Problem

- Then, in this example,

$$||\nabla_{\mathbf{b}}^{(1)}C|| = 0.07 \text{ and } ||\nabla_{\mathbf{b}}^{(2)}C|| = 0.31 \tag{18}$$

- Formal quantification shows: learning faster in hidden layer 2.
- When running the identical training task (MNIST), we obtain
  - $||\nabla_{\mathbf{b}}^{(1)}C|| = 0.012, ||\nabla_{\mathbf{b}}^{(2)}C|| = 0.06, ||\nabla_{\mathbf{b}}^{(3)}C|| = 0.283$ for three hidden layers
  - $||\nabla_{\mathbf{b}}^{(1)}C|| = 0.003, ||\nabla_{\mathbf{b}}^{(2)}C|| = 0.017, ||\nabla_{\mathbf{b}}^{(3)}C|| = 0.07, ||\nabla_{\mathbf{b}}^{(4)}C|| =$ 0.285 for four hidden layers
  - and so on...

The Vanishing Gradient Problem



Training speed in [784,30,30,30,30,10]-NN on MNIST

The Vanishing Gradient Problem

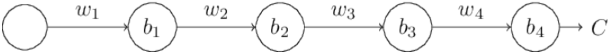
- *Vanishing gradient problem*: Neurons in earlier layers learn more slowly
- *Exploding gradient problem*: Neurons in earlier layers learn faster
- In general, gradients in NN's are unstable across layers
- And: vanishing gradients do not mean that there is nothing left to be learnt
- ☞ Fundamental problem for gradient-based learning in NN's

The Vanishing Gradient Problem

- *Vanishing gradient problem*: Neurons in earlier layers learn more slowly
- *Exploding gradient problem*: Neurons in earlier layers learn faster
- In general, gradients in NN's are unstable across layers
- And: vanishing gradients do not mean that there is nothing left to be learnt
- ☞ Fundamental problem for gradient-based learning in NN's

The Vanishing Gradient Problem

Explanation



Simple NN with 3 hidden layers of one neuron each

Let w_1, w_2, w_3, w_4 be the weights, b_1, b_2, b_3, b_4 be the biases and C the cost. Let all neurons be sigmoid, so the output a_j from the j-the neuron is $\sigma(z_j)$ where $z_j = w_j a_{j-1} + b_j$ is the input of the j-th neuron (notation as usual earlier).

The Vanishing Gradient Problem

Explanation

$$\bigcirc \xrightarrow{w_1} b_1 \xrightarrow{w_2} b_2 \xrightarrow{w_3} b_3 \xrightarrow{w_4} b_4 \rightarrow C$$

Simple NN with 3 hidden layers of one neuron each

Let w_1, w_2, w_3, w_4 be the weights, b_1, b_2, b_3, b_4 be the biases and C the cost. Let all neurons be sigmoid, so the output a_j from the j-the neuron is $\sigma(z_j)$ where $z_j = w_j a_{j-1} + b_j$ is the input of the j-th neuron (notation as usual earlier).

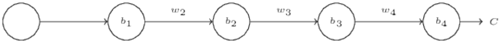
For understanding the Vanishing Gradient Problem, consider $\frac{\partial C}{\partial b_1}$. By repeated application of the backpropagation rules, we see that

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4} \tag{19}$$

The Vanishing Gradient Problem

Explanation

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$



Computing $\frac{\partial C}{\partial b_1}$

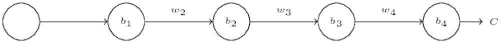
There is an alternative explanation for (19). Let Δ indicate small changes. We know that

$$\frac{\partial C}{\partial b_1} \approx \frac{\Delta C}{\Delta b_1} \tag{20}$$

The Vanishing Gradient Problem

Explanation

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$



Computing $\frac{\partial C}{\partial b_1}$

There is an alternative explanation for (19). Let Δ indicate small changes. We know that

$$\frac{\partial C}{\partial b_1} \approx \frac{\Delta C}{\Delta b_1} \tag{20}$$

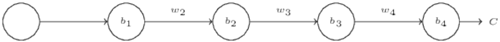
From $a_1 = \sigma(z_1) = \sigma(w_1 a_0 + b_1)$ we further obtain

$$\Delta a_1 \approx \frac{\partial \sigma(w_1 a_0 + b_1)}{\partial b_1} \Delta b_1 = \sigma'(z_1) \Delta b_1 \tag{21}$$

The Vanishing Gradient Problem

Explanation

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$



Computing $\frac{\partial C}{\partial b_1}$

There is an alternative explanation for (19). Let Δ indicate small changes. We know that

$$\frac{\partial C}{\partial b_1} \approx \frac{\Delta C}{\Delta b_1} \tag{20}$$

From $a_1 = \sigma(z_1) = \sigma(w_1 a_0 + b_1)$ we further obtain

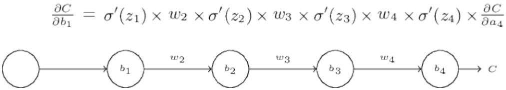
$$\Delta a_1 \approx \frac{\partial \sigma(w_1 a_0 + b_1)}{\partial b_1} \Delta b_1 = \sigma'(z_1) \Delta b_1 \tag{21}$$

further leading to

$$\Delta z_2 \approx \frac{\partial z_2}{\partial a_1} \Delta a_1 = w_2 \Delta a_1 \quad \text{implying} \quad \Delta z_2 \approx \sigma'(z_1) w_2 \Delta b_1 \tag{22}$$

The Vanishing Gradient Problem

Explanation



Computing $\frac{\partial C}{\partial b_1}$

Repeated application of the computations from the slide before eventually yield

$$\Delta C \approx \sigma'(z_1) w_2 \sigma'(z_2) \ldots \sigma'(z_4) \frac{\partial C}{\partial a_4} \Delta b_1 \tag{23}$$

Dividing by b_1 results in the desired expression (19):

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4} \tag{24}$$

The Vanishing Gradient Problem

Explanation

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4} \tag{25}$$

Except from the last term, this is a product of terms of the form

$$w_j \sigma'(z_j) \tag{26}$$

It holds that $0 \leq \sigma'(z_j) \leq 1/4$, while, in practice, when employing standard initialization of weights, typically $|w_j| < 1$, so

$$|w_j \sigma'(z_j)| \leq \frac{1}{4} \tag{27}$$

so in combination

$$\frac{\partial C}{\partial b_1} \leq \sigma'(z_1) (\frac{1}{4})^3 \frac{\partial C}{\partial a_4} \tag{28}$$

THE VANISHING GRADIENT PROBLEM

EXPLANATION

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1)\,\overbrace{w_2\sigma'(z_2)}^{<\frac{1}{4}}\,\overbrace{w_3\sigma'(z_3)}^{<\frac{1}{4}}\,\underbrace{w_4\sigma'(z_4)\frac{\partial C}{\partial a_4}}$$

common terms

$$\frac{\partial C}{\partial b_3} = \sigma'(z_3)\,\overbrace{w_4\sigma'(z_4)\frac{\partial C}{\partial a_4}}$$

Comparing $\frac{\partial C}{\partial b_1}$ with $\frac{\partial C}{\partial b_3}$

So, $\frac{\partial C}{\partial b_1}$ is about a factor of 16 (or more) smaller than $\frac{\partial C}{\partial b_3}$. Similar conclusions are drawn for $\frac{\partial C}{\partial w_j}$.

The Exploding Gradient Problem

The "Exploding Gradient Problem" occurs when

- Weights are too large (say on the order of 100 each)
- Biases b_j are such that $\sigma'(z_j)$ never is small
- *Example*: $b_j = -100 \times a_{j-1}$, so $z_j = 100 \times a_{j-1} - 100 \times a_{j-1} = 0$, implying $\sigma'(z_j) = 1/4$, yielding $w_j\sigma'(z_j) > 20$ as a gradient
- In such situations gradients iteratively explode

The Exploding Gradient Problem

The "Exploding Gradient Problem" occurs when

- Weights are too large (say on the order of 100 each)
- Biases b_j are such that $\sigma'(z_j)$ never is small
- *Example*: $b_j = -100 \times a_{j-1}$, so $z_j = 100 \times a_{j-1} - 100 \times a_{j-1} = 0$, implying $\sigma'(z_j) = 1/4$, yielding $w_j \sigma'(z_j) > 20$ as a gradient
- In such situations gradients iteratively explode

The Exploding Gradient Problem

The "Exploding Gradient Problem" occurs when

- Weights are too large (say on the order of 100 each)
- Biases b_j are such that $\sigma'(z_j)$ never is small
- *Example*: $b_j = -100 \times a_{j-1}$, so $z_j = 100 \times a_{j-1} - 100 \times a_{j-1} = 0$, implying $\sigma'(z_j) = 1/4$, yielding $w_j \sigma'(z_j) > 20$ as a gradient
- In such situations gradients iteratively explode

Gradients are Unstable

- The fundamental problem is that gradients in earlier layers are products of gradients from (all the) later layers.
- If there are many layers, the situation is unstable, unless the gradients are *balanced out*.
- Balancing is very unlikely to happen by chance, so one needs to fix this explicitly.
- Fixing this seems daunting at first glance: when making weights w_j large,

$$\sigma'(z_j) = \sigma'(w_j a_{j-1} + b_j)$$

  will get small.
- *Solutions*:
  - *Rectified Linear Units* instead of sigmoid activation
  - *Batch Normalization* (discussed later in the lecture)

Gradients are Unstable

- The fundamental problem is that gradients in earlier layers are products of gradients from (all the) later layers.
- If there are many layers, the situation is unstable, unless the gradients are *balanced out*.
- Balancing is very unlikely to happen by chance, so one needs to fix this explicitly.
- Fixing this seems daunting at first glance: when making weights w_j large,

  $$\sigma'(z_j) = \sigma'(w_j a_{j-1} + b_j)$$

  will get small.
- *Solutions*:
  - *Rectified Linear Units* instead of sigmoid activation
  - *Batch Normalization* (discussed later in the lecture)

Gradients are Unstable

- The fundamental problem is that gradients in earlier layers are products of gradients from (all the) later layers.
- If there are many layers, the situation is unstable, unless the gradients are *balanced out*.
- Balancing is very unlikely to happen by chance, so one needs to fix this explicitly.
- Fixing this seems daunting at first glance: when making weights w_j large,

$$\sigma'(z_j) = \sigma'(w_j a_{j-1} + b_j)$$

  will get small.

- *Solutions*:
  - *Rectified Linear Units* instead of sigmoid activation
  - *Batch Normalization* (discussed later in the lecture)

Why is Deep Learning Tough?

Literature

There are other issues that prevent easy training of neural networks with deep architectures. For further reading, see for example

- "Understanding the difficulty of training deep feedforward neural networks", X. Glorot, Y. Bengio, 2010, `http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf`

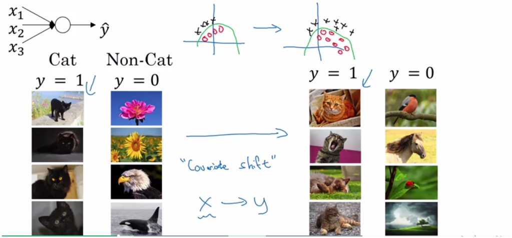
or the earlier

- "Efficient BackProp", Y. LeCun, L. Bottou, G. Orr, K.-R. Müller, 1998, `http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf`
- "On the importance of initialization and momentum in deep learning", I. Sutskever, J. Martens, G. Dahl, G. Hinton, 2013, `http://www.cs.toronto.edu/~hinton/absps/momentum.pdf`

Batch Normalization

Batch Normalization

Motivation

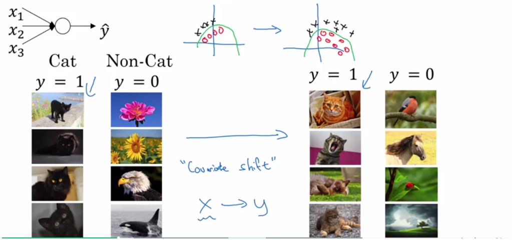


Learning black cats might not help to recognize cats of other colors

- The network might not be able to predict well if presented with examples not present in the training data (batch)
- The function learned can only be guaranteed to predict well in certain areas of feature space

Batch Normalization

Motivation

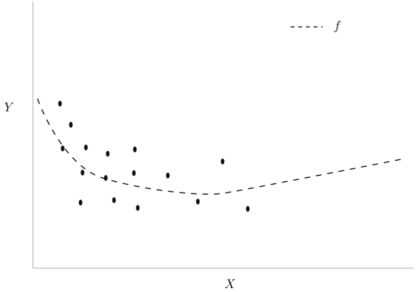


Learning black cats might not help to recognize cats of other colors

- The network might not be able to predict well if presented with examples not present in the training data (batch)
- The function learned can only be guaranteed to predict well in certain areas of feature space

Batch Normalization

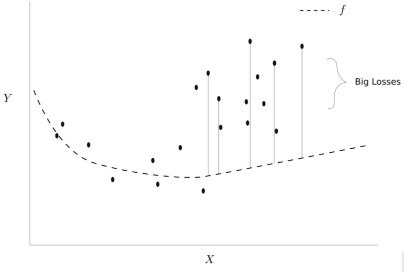
Motivation



- Let f be the function that maps values from layer $l-1$ to layer l
- After updating gradients, values in layer $l-1$ may have been shifted to a region where f does not approximate the true function well
- The effect is referred to as *internal covariate shift* (although this is not necessarily the correct term to describe the effect)
- This slows down training

Batch Normalization

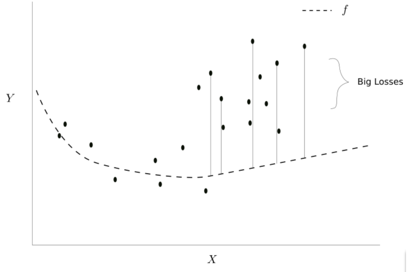
Motivation



- Let f be the function that maps values from layer $l-1$ to layer l
- After updating gradients, values in layer $l-1$ may have been shifted to a region where f does not approximate the true function well
- The effect is referred to as *internal covariate shift* (although this is not necessarily the correct term to describe the effect)
- This slows down training

Batch Normalization

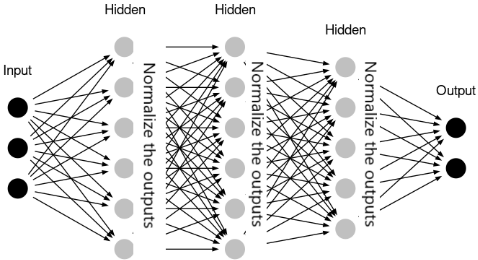
Motivation



- Let f be the function that maps values from layer $l-1$ to layer l
- After updating gradients, values in layer $l-1$ may have been shifted to a region where f does not approximate the true function well
- The effect is referred to as *internal covariate shift* (although this is not necessarily the correct term to describe the effect)
- This slows down training

Batch Normalization

Solution



Batch Normalization: Insert normalization layers between normal-type layers

- After each layer, normalize output values
- There are parameters to be learned for normalization layers
- Parameters for normalization layers can be easily learnt with backpropagation

Batch Normalization

Definition

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

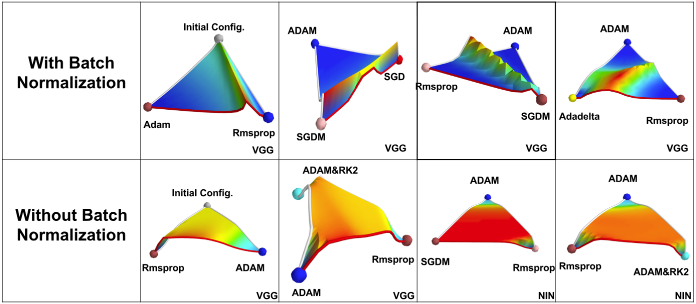
https://arxiv.org/abs/1502.03167

(Ioffe & Szegedy, original paper)

- Compute $\hat{x}_i$ when forwarding training samples
- Learn γ, β during backpropagation

Batch Normalization

Explanation



[From: http://www.aifounded.com/machine-learning/deep-loss]

- Low error regions are larger
- Boundaries are more clearly / sharply defined
- The reshaping of the cost function surface leads to accelerated training

Batch Normalization

Summary Benefits

- *Gradient Vanishing*: Batch Normalization prevents gradients from vanishing
- *Internal Covariate Shift*: controversial debate whether it helps (although it is motivated by it)
- Boundaries of error regions are more clearly / sharply defined
- Reshapes cost function surface: accelerated training

Batch Normalization

Summary Benefits

- *Gradient Vanishing*: Batch Normalization prevents gradients from vanishing
- *Internal Covariate Shift*: controversial debate whether it helps (although it is motivated by it)
- Boundaries of error regions are more clearly / sharply defined
- Reshapes cost function surface: accelerated training

Summary

- Convolutional backpropagation
  - For further illustrations, see `https://medium.com/@2017csm1006/forward-and-backpropagation-in-convolutional-neural-network-`
  - Note that there notation differs (error E there is cost C here, X there is z here, and F are the weights w here, and O is a here, and there is no bias)
- Training variations
  - `http://www.deeplearningbook.org/`, Chapter 8 (corresponding parts)
  - `http://neuralnetworksanddeeplearning.com/`, Chapter 3, "Variations on stochastic gradient descent"
- The vanishing gradient problem
  - `http://neuralnetworksanddeeplearning.com/`, Chapter 5
- Batch normalization
  - See `http://www.deeplearningbook.org/`, 8.7.1
  - See also `http://www.aifounded.com/machine-learning/deep-loss`, for example

Thanks for your attention