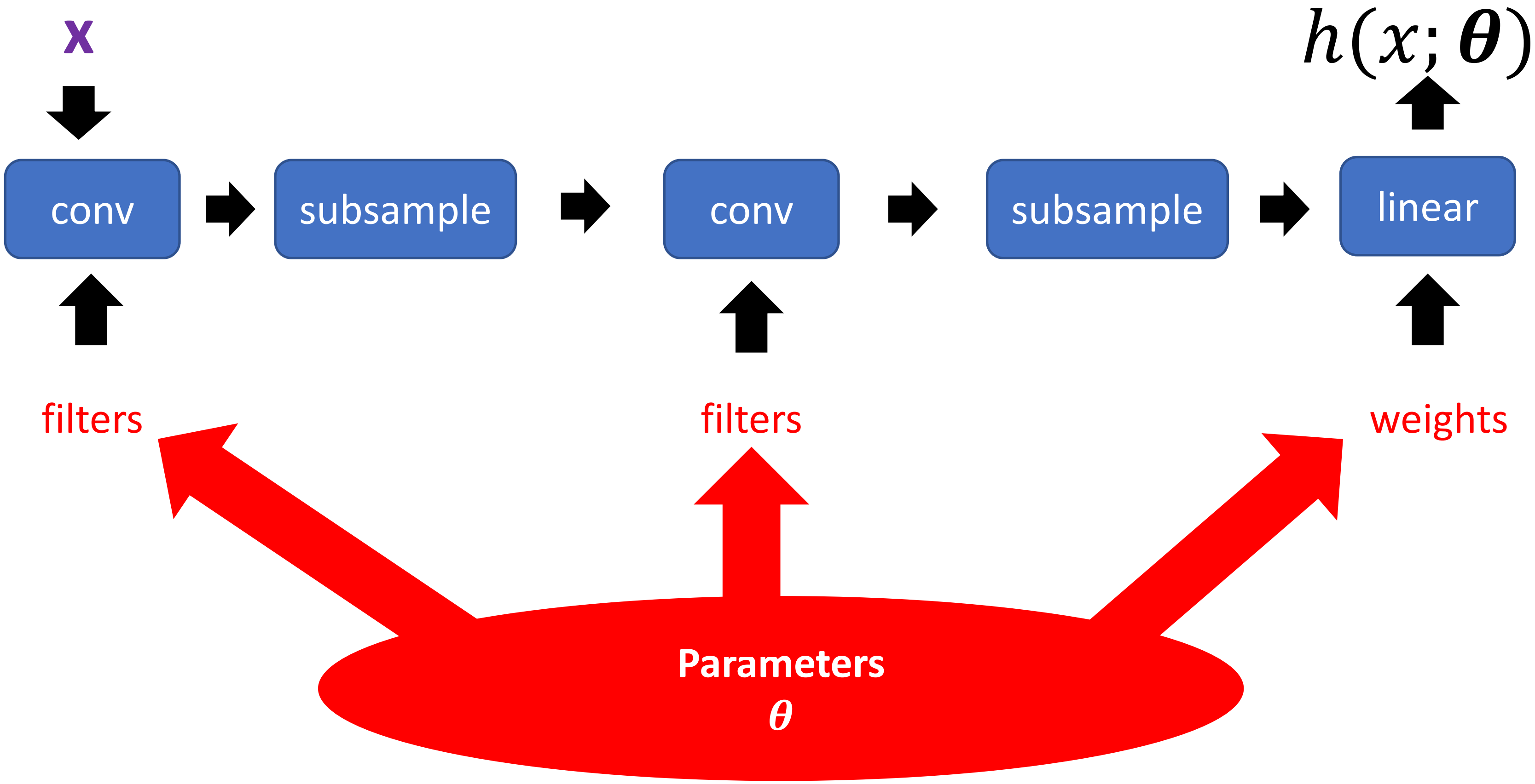
Backpropagation

Why backpropagation

- Neural networks are sequences of parametrized functions



Why backpropagation

- Neural networks are sequences of parametrized functions
- Parameters need to be set by minimizing some loss function

$$\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^{N} L(\boxed{h(x_i; \boldsymbol{\theta})}, y_i)$$

Convolutional network

Why backpropagation

- Neural networks are sequences of parametrized functions
- Parameters need to be set by minimizing some loss function
- Minimization through gradient descent requires computing the gradient

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \lambda \frac{1}{N} \sum_{i=1}^{N} \nabla L(h(x_i; \boldsymbol{\theta}), y_i)$$

Why backpropagation

- Neural networks are sequences of parametrized functions
- Parameters need to be set by minimizing some loss function
- Minimization through gradient descent requires computing the gradient

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \lambda \frac{1}{N} \sum_{i=1}^{N} \nabla L(h(x_i; \boldsymbol{\theta}), y_i)$$

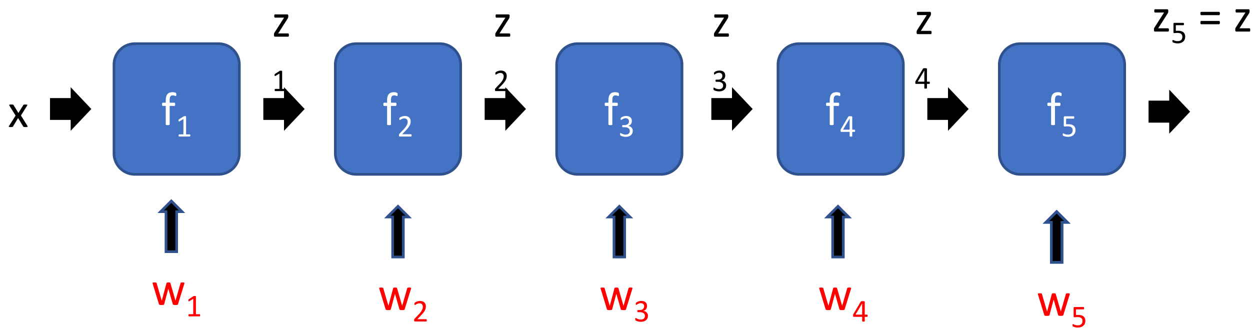
$$z = h(x; \boldsymbol{\theta})$$

$$\nabla_{\boldsymbol{\theta}} L(z, y) = \frac{\partial L(z, y)}{\partial z} \frac{\partial z}{\partial \boldsymbol{\theta}}$$

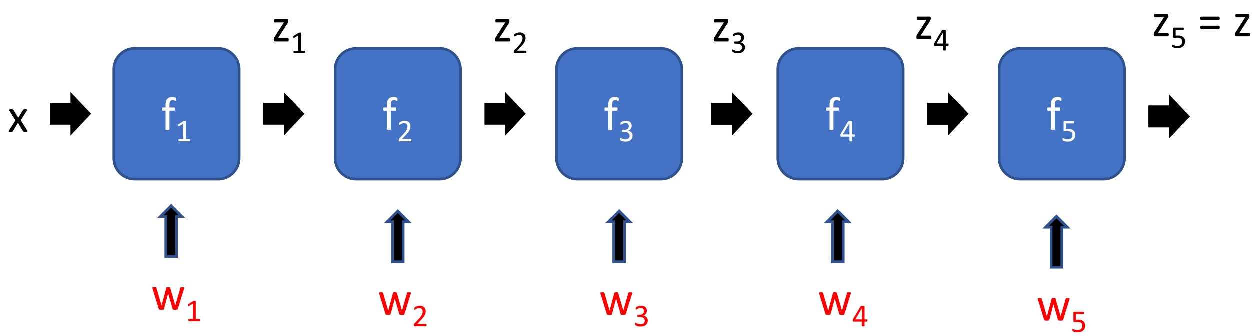
Why backpropagation

- Neural networks are sequences of parametrized functions
- Parameters need to be set by minimizing some loss function
- Minimization through gradient descent requires computing the gradient
- **Backpropagation**: way to compute gradient $\frac{\partial z}{\partial \boldsymbol{\theta}}$

The gradient of convnets

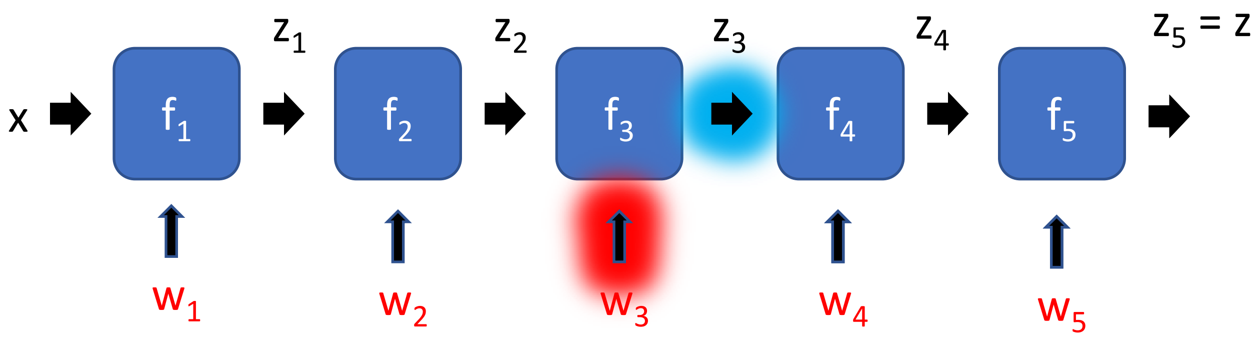


The gradient of convnets



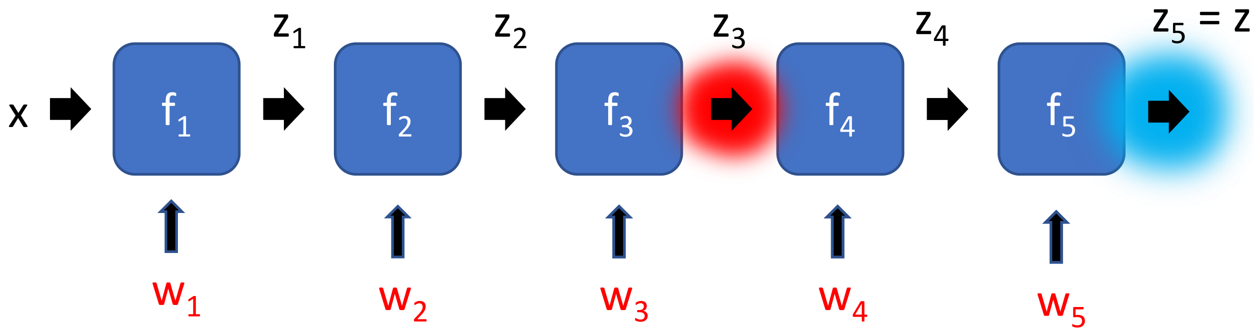
$$\frac{\partial z}{\partial w_3}$$

The gradient of convnets



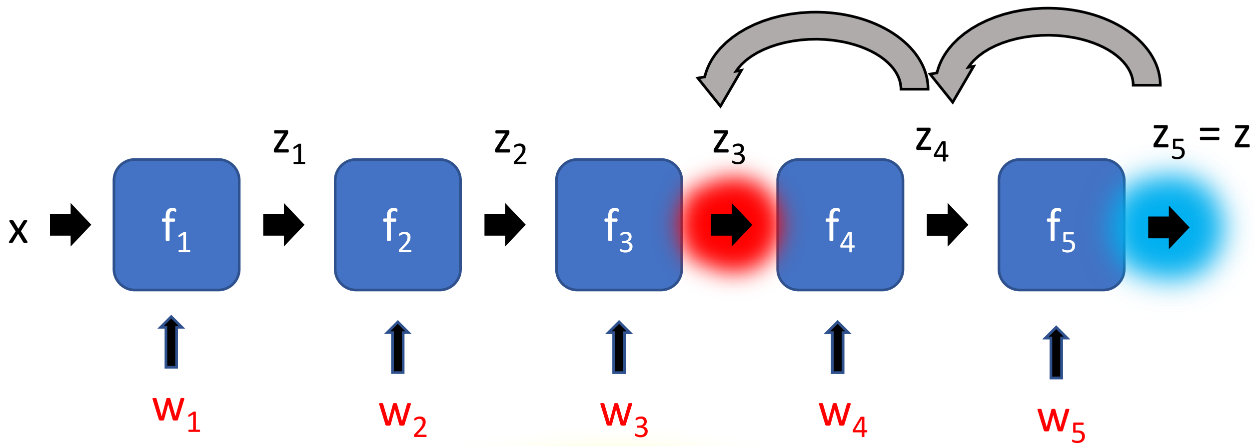
$$\frac{\partial z}{\partial w_3} = \frac{\partial z}{\partial z_3}\frac{\partial z_3}{\partial w_3}$$

The gradient of convnets



$$\frac{\partial z}{\partial w_3} = \frac{\partial z}{\partial z_3} \frac{\partial z_3}{\partial w_3}$$

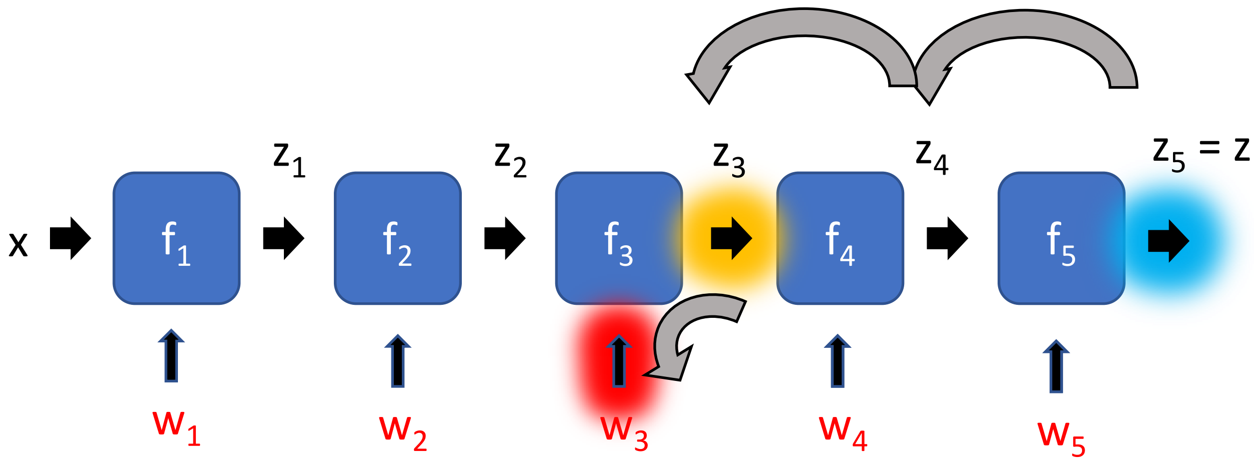
The gradient of convnets



$$\frac{\partial z}{\partial z_3} = \frac{\partial z}{\partial z_4}\frac{\partial z_4}{\partial z_3}$$

$$\frac{\partial z}{\partial w_3} = \frac{\partial z}{\partial z_3}\frac{\partial z_3}{\partial w_3}$$

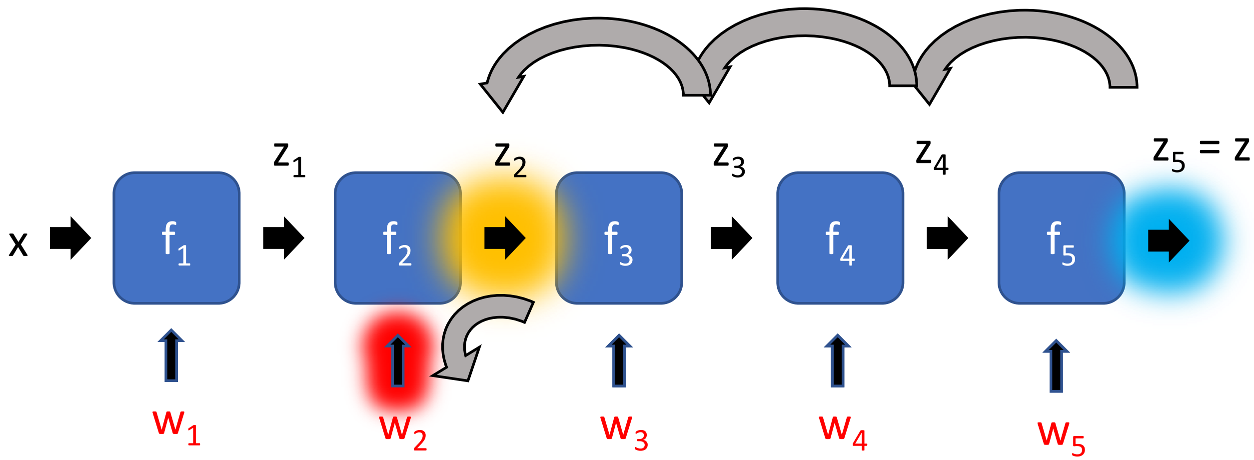
The gradient of convnets



$$\frac{\partial z}{\partial z_3} = \frac{\partial z}{\partial z_4}\frac{\partial z_4}{\partial z_3}$$

$$\frac{\partial z}{\partial w_3} = \frac{\partial z}{\partial z_3}\frac{\partial z_3}{\partial w_3}$$

The gradient of convnets

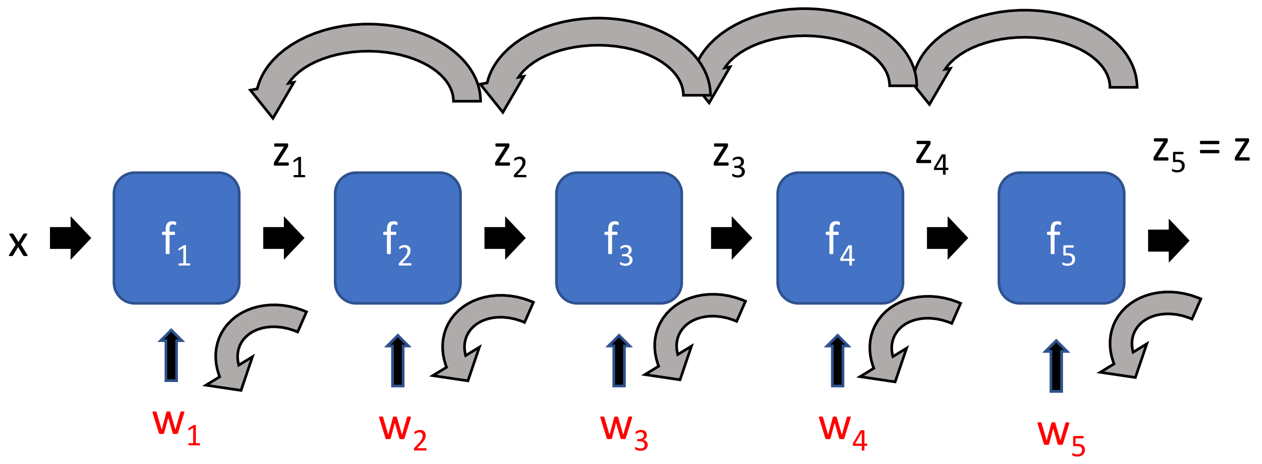


$$\frac{\partial z}{\partial z_2} = \frac{\partial z}{\partial z_3}\frac{\partial z_3}{\partial z_2}$$

$$\frac{\partial z}{\partial w_2} = \frac{\partial z}{\partial z_2}\frac{\partial z_2}{\partial w_2}$$

Recurrence going backward!!

The gradient of convnets



Backpropagation

Backpropagation for a sequence of functions

$$z_i = f_i(z_{i-1}, w_i) \qquad z_0 = x \qquad z = z_n$$

- Assume we can compute partial derivatives of each function

$$\frac{\partial z_i}{\partial z_{i-1}} = \frac{\partial f_i(z_{i-1}, w_i)}{\partial z_{i-1}} \qquad \frac{\partial z_i}{\partial w_i} = \frac{\partial f_i(z_{i-1}, w_i)}{\partial w_i}$$

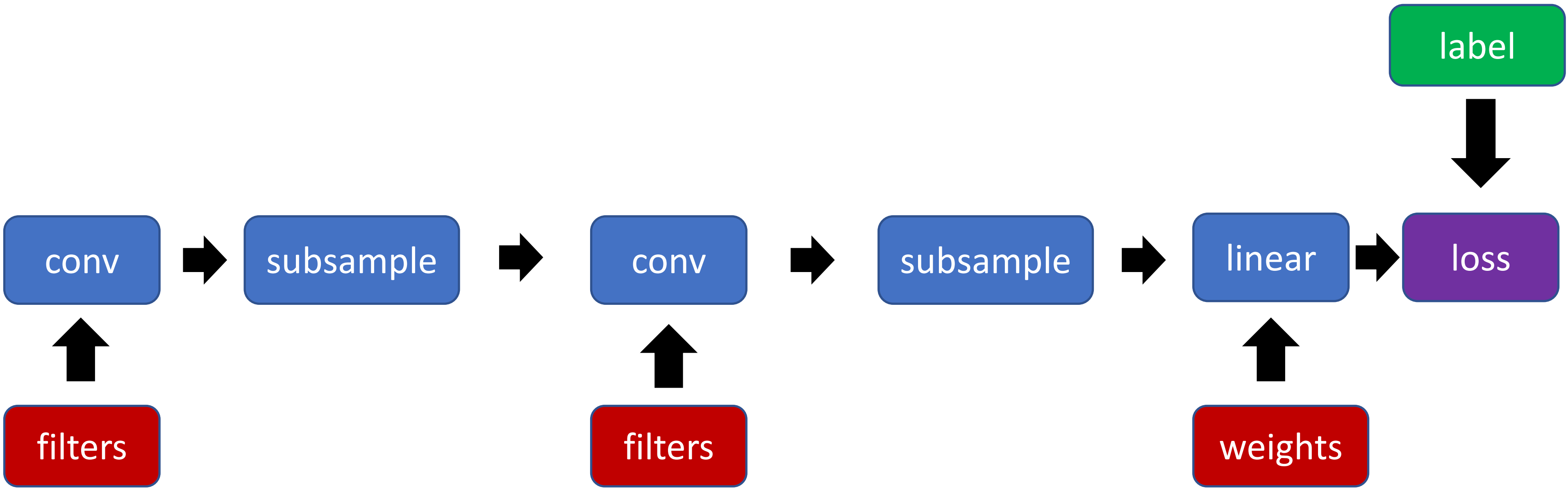
- Use $g(z_i)$ to store gradient of z w.r.t z_i, $g(w_i)$ for w_i
- Calculate $g(z_i)$ by iterating backwards

$$g(z_n) = \frac{\partial z}{\partial z_n} = 1 \qquad g(z_{i-1}) = \frac{\partial z}{\partial z_i}\frac{\partial z_i}{\partial z_{i-1}} = g(z_i)\frac{\partial z_i}{\partial z_{i-1}}$$

- Use $g(z_i)$ to compute gradient of parameters

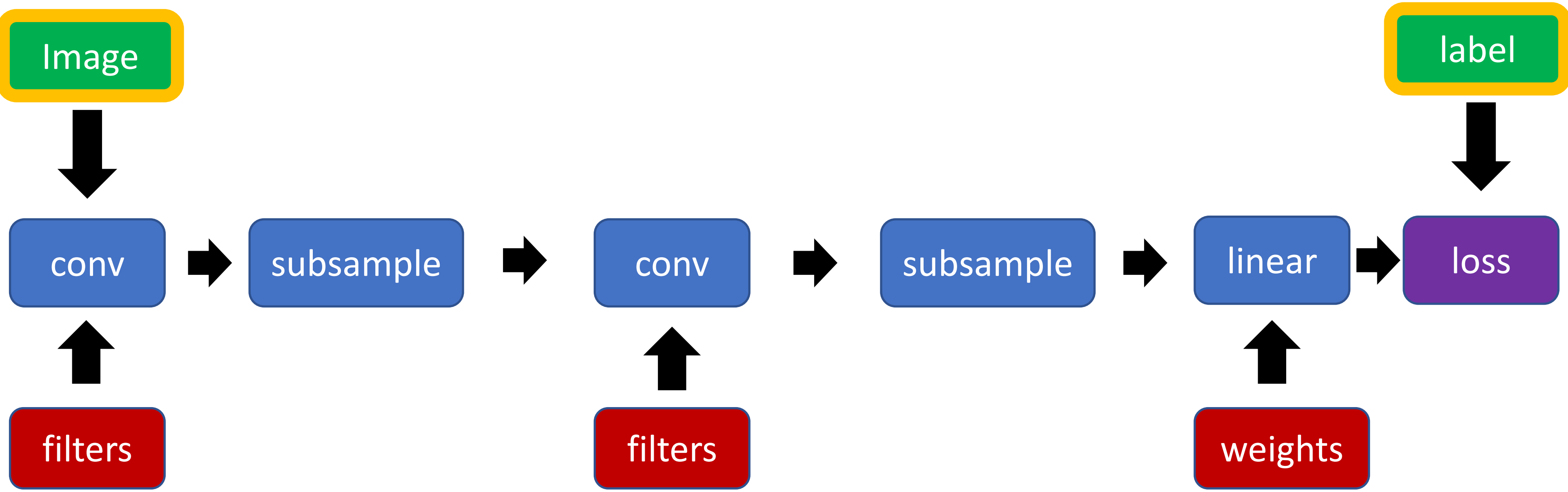
$$g(w_i) = \frac{\partial z}{\partial z_i}\frac{\partial z_i}{\partial w_i} = g(z_i)\frac{\partial z_i}{\partial w_i}$$

Loss as a function



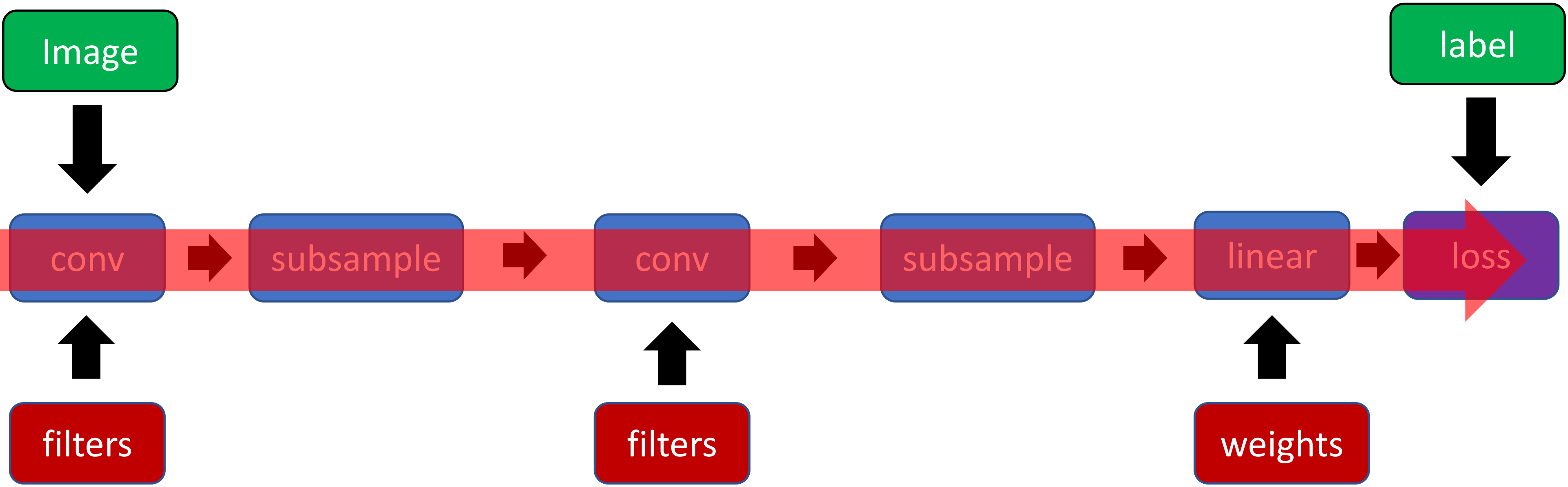
Putting it all together: SGD training of ConvNets

1. Sample image and label



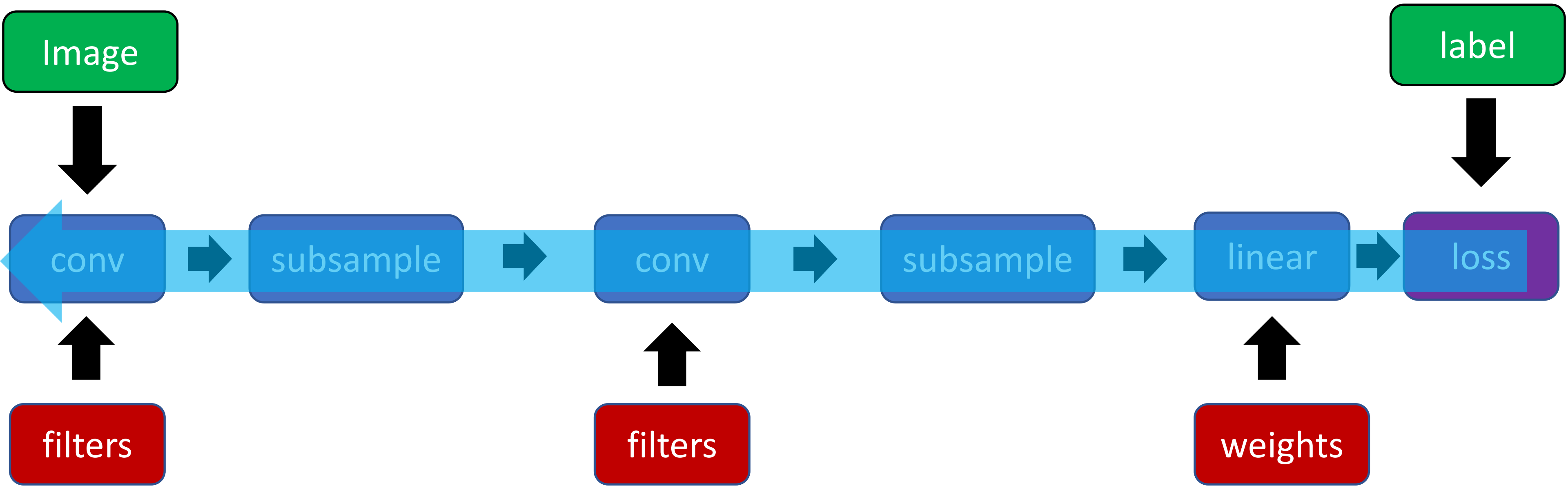
Putting it all together: SGD training of ConvNets

1. Sample image and label
2. Pass image through network to get loss (forward)



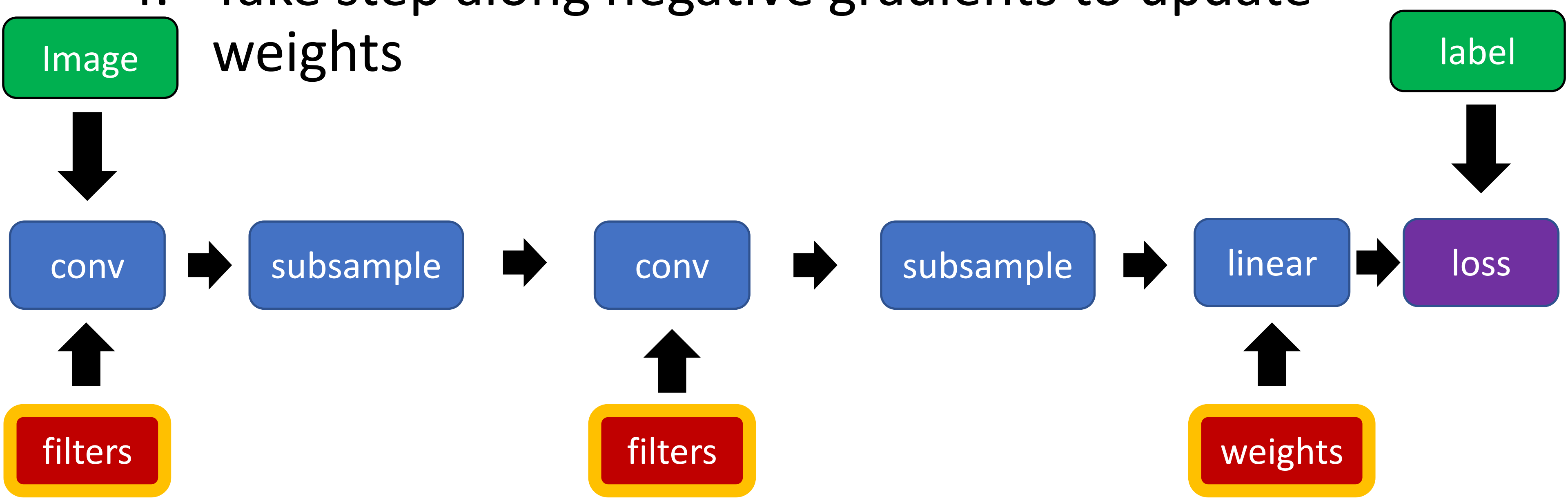
Putting it all together: SGD training of ConvNets

1. Sample image and label
2. Pass image through network to get loss (forward)
3. Backpropagate to get gradients (backward)



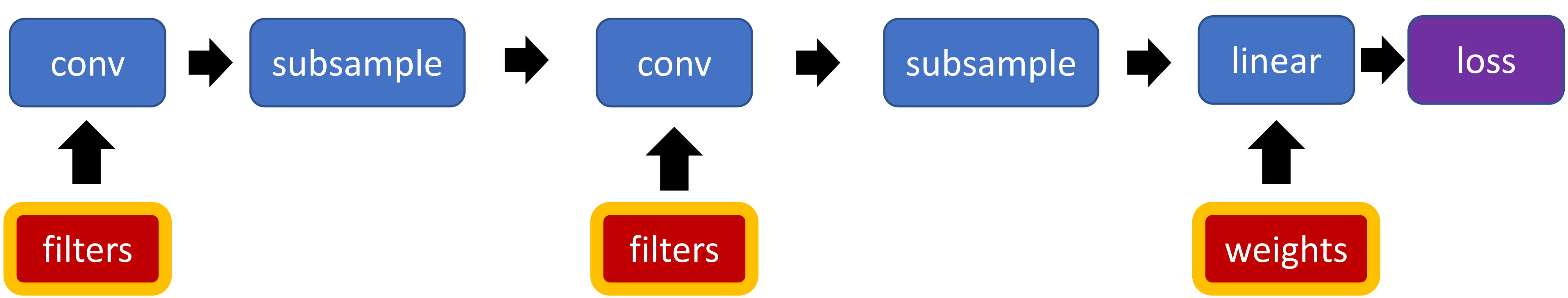
Putting it all together: SGD training of ConvNets

1. Sample image and label
2. Pass image through network to get loss (forward)
3. Backpropagate to get gradients (backward)
4. Take step along negative gradients to update weights



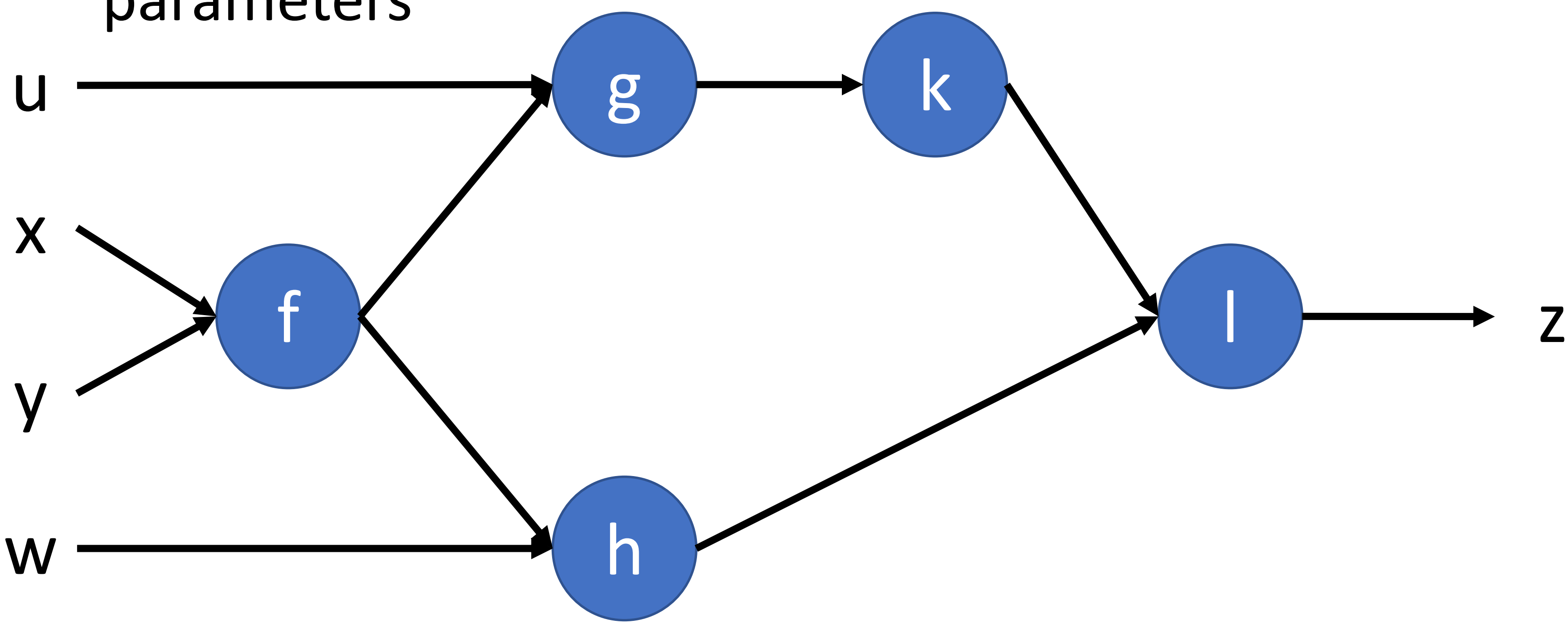
Putting it all together: SGD training of ConvNets

1. Sample image and label
2. Pass image through network to get loss (forward)
3. Backpropagate to get gradients (backward)
4. Take step along negative gradients to update weights
5. Repeat!



Beyond sequences: computation graphs

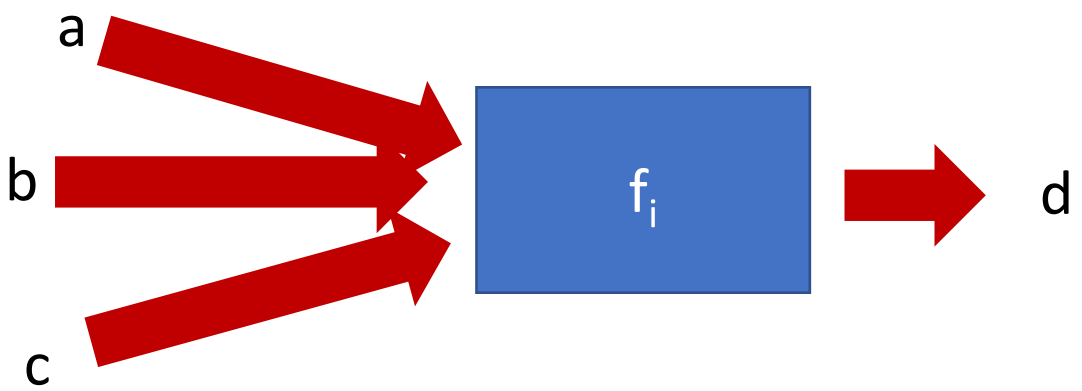
- Arbitrary *graphs* of functions
- No distinction between intermediate outputs and parameters



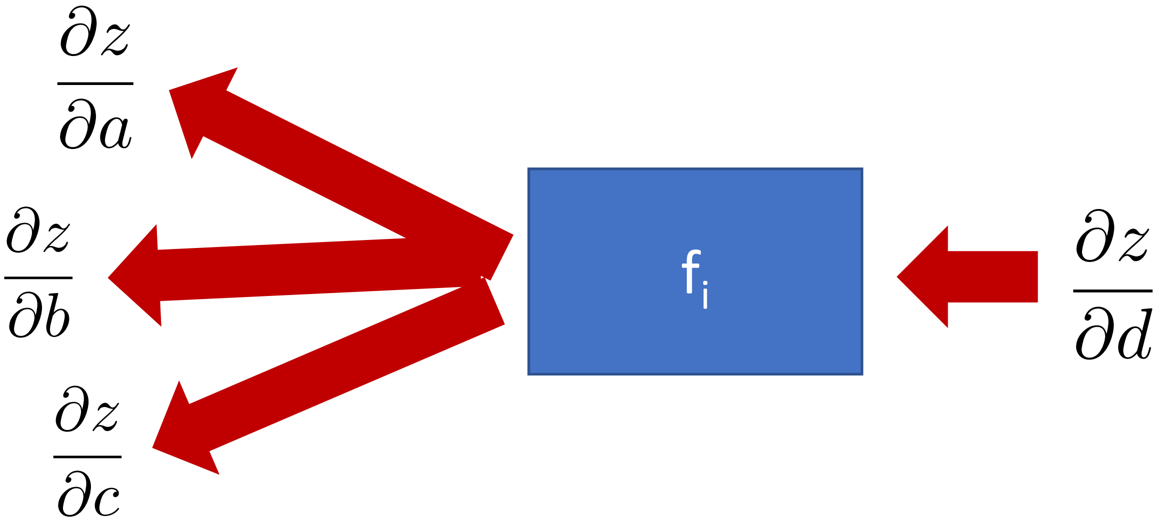
Computation graph - Functions

- Each node implements two functions
  - A “forward”
    - Computes output given input
  - A “backward”
    - Computes derivative of z w.r.t input, given derivative of z w.r.t output

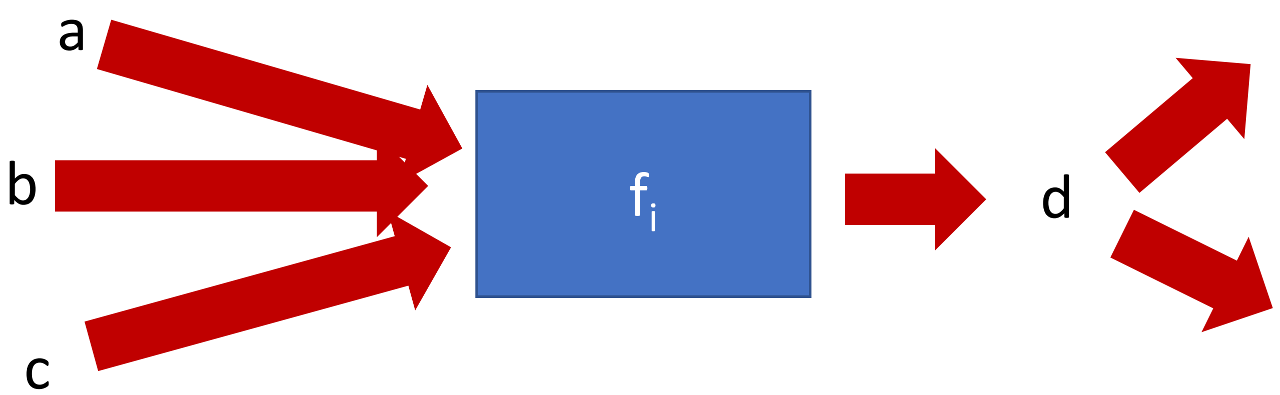
Computation graphs



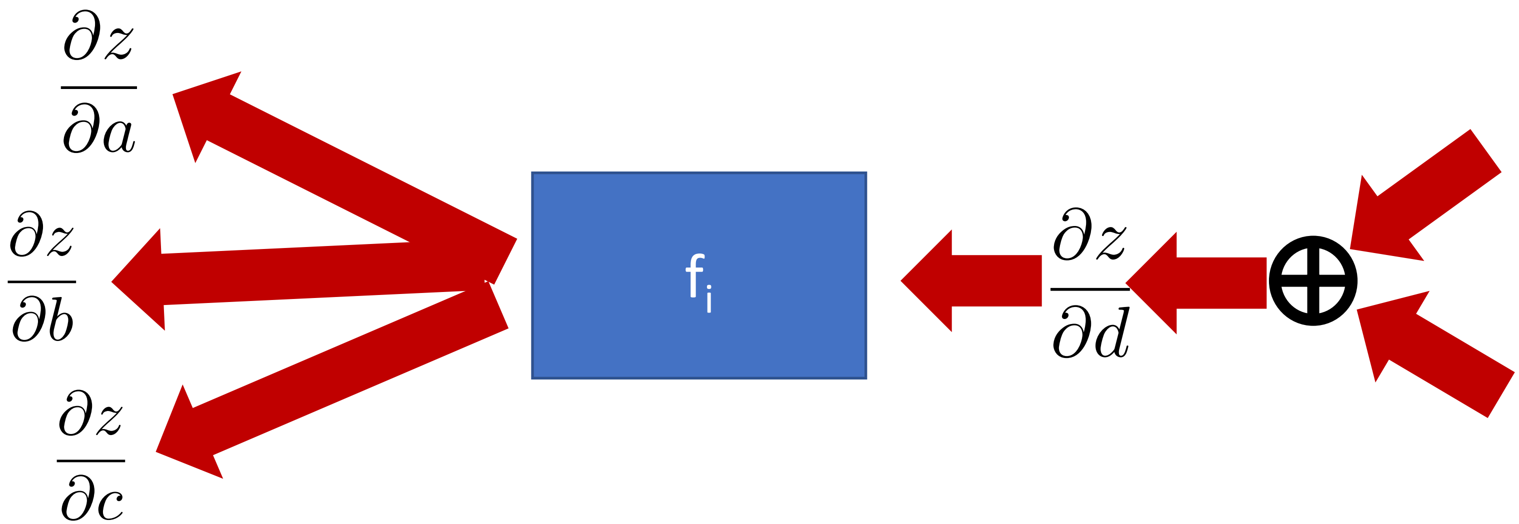
Computation graphs



Computation graphs



Computation graphs



Neural network frameworks

Stochastic gradient descent

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \lambda \underbrace{\frac{1}{K}\sum_{k=1}^{K} \nabla L(h(x_{i_k};\boldsymbol{\theta}^{(t)}), y_{i_k})}_{\text{Noisy!}}$$

Momentum

- *Average* multiple gradient steps
- Use *exponential averaging*

$$\mathbf{g}^{(t)} \leftarrow \frac{1}{K}\sum_{k=1}^{K} \nabla L(h(x_{i_k};\boldsymbol{\theta}^{(t)}), y_{i_k})$$

$$\mathbf{p}^{(t)} \leftarrow \mu \mathbf{g}^{(t)} + (1-\mu)\mathbf{p}^{(t-1)}$$

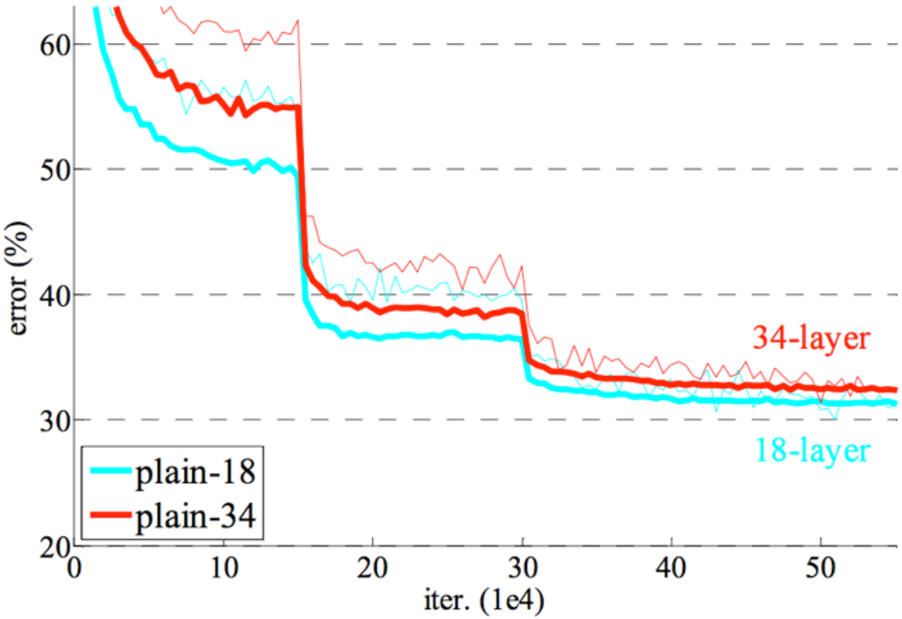
$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \lambda \mathbf{p}^{(t)}$$

Weight decay

- Add $-\alpha\boldsymbol{\theta}^{(t)}$ to the gradient
- Prevents $\boldsymbol{\theta}$ from growing to infinity
- Equivalent to L2 regularization of weights

Learning rate decay

- Large step size / learning rate
    - Faster convergence initially
    - Bouncing around at the end because of noisy gradients
- Learning rate must be decreased over time
- Usually done in steps



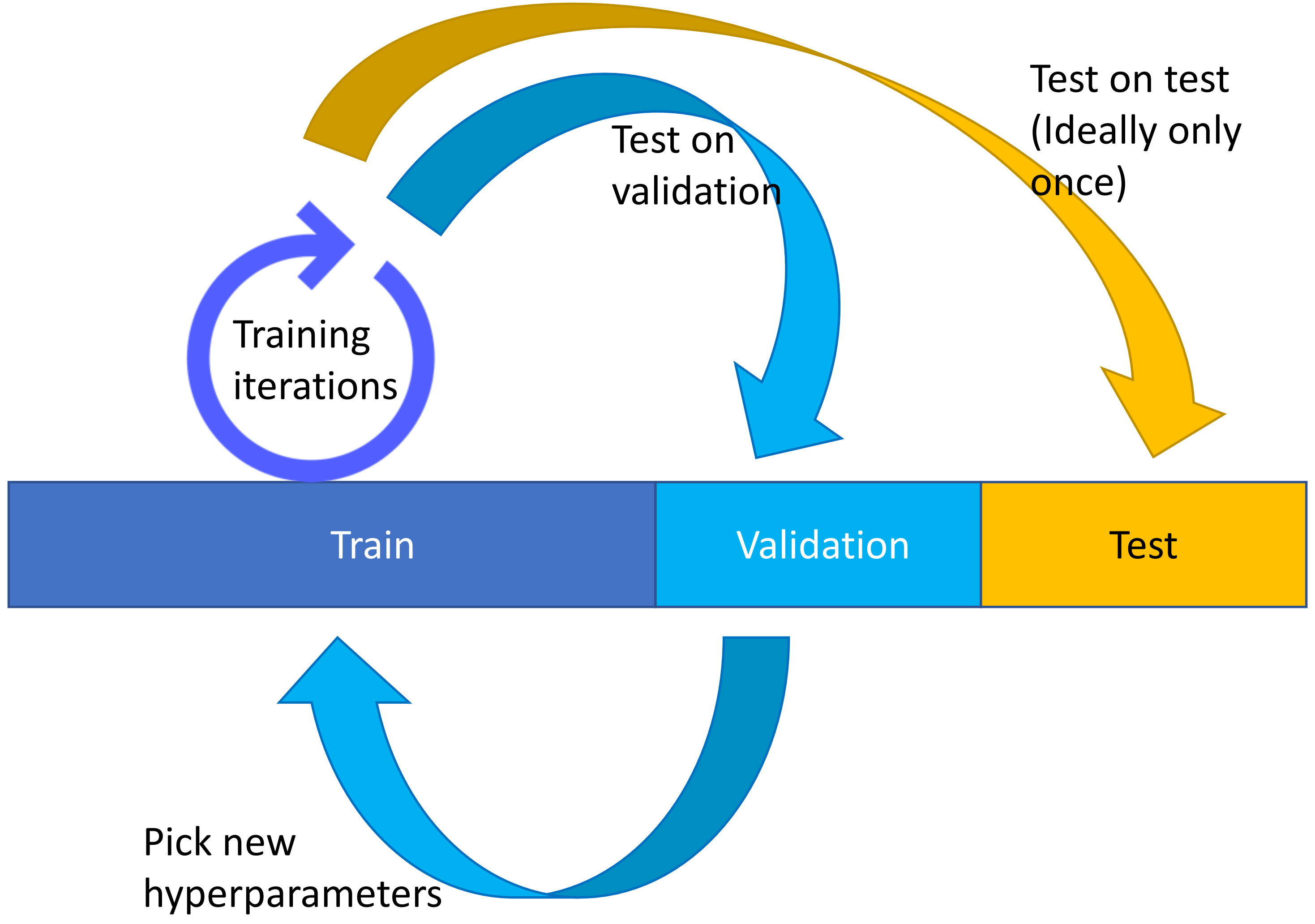
Convolutional network training

- Initialize network
- Sample *minibatch* of images
- Forward pass to compute loss
- Backpropagate loss to compute gradient
- Combine gradient with momentum and weight decay
- Take step according to current learning rate

Setting hyperparameters

- How do we find a hyperparameter setting that works?
- Try it!
  - Train on train
  - Test on ~~test~~ validation
- Picking hyperparameters that work for test = Overfitting on test set

Setting hyperparameters



Vagaries of optimization

- Non-convex
  - Local optima
  - Sensitivity to initialization
- Vanishing / exploding gradients

$$\frac{\partial z}{\partial z_i} = \frac{\partial z}{\partial z_{n-1}} \frac{\partial z_{n-1}}{\partial z_{n-2}} \cdots \frac{\partial z_{i+1}}{\partial z_i}$$

  - If each term is (much) greater than 1 → *explosion of gradients*
  - If each term is (much) less than 1 → *vanishing gradients*

Image Classification

How to do machine learning

- Create training / validation sets
- Identify loss functions
- Choose hypothesis class
- Find best hypothesis by minimizing training loss

How to do machine learning

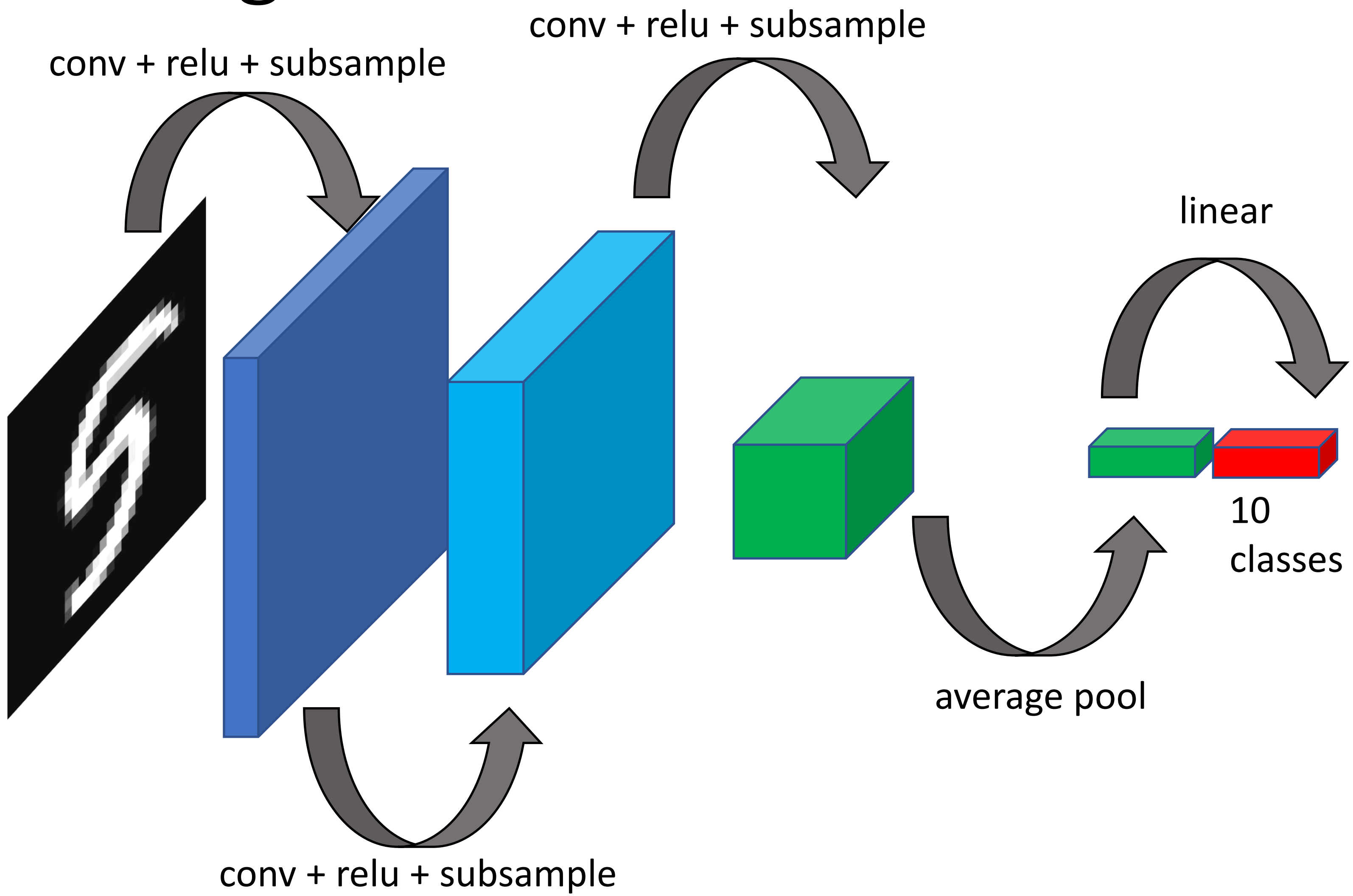
- Create training / validation sets
- Identify loss functions
- Choose hypothesis class
- Find best hypothesis by minimizing training loss

Multiclass classification!!

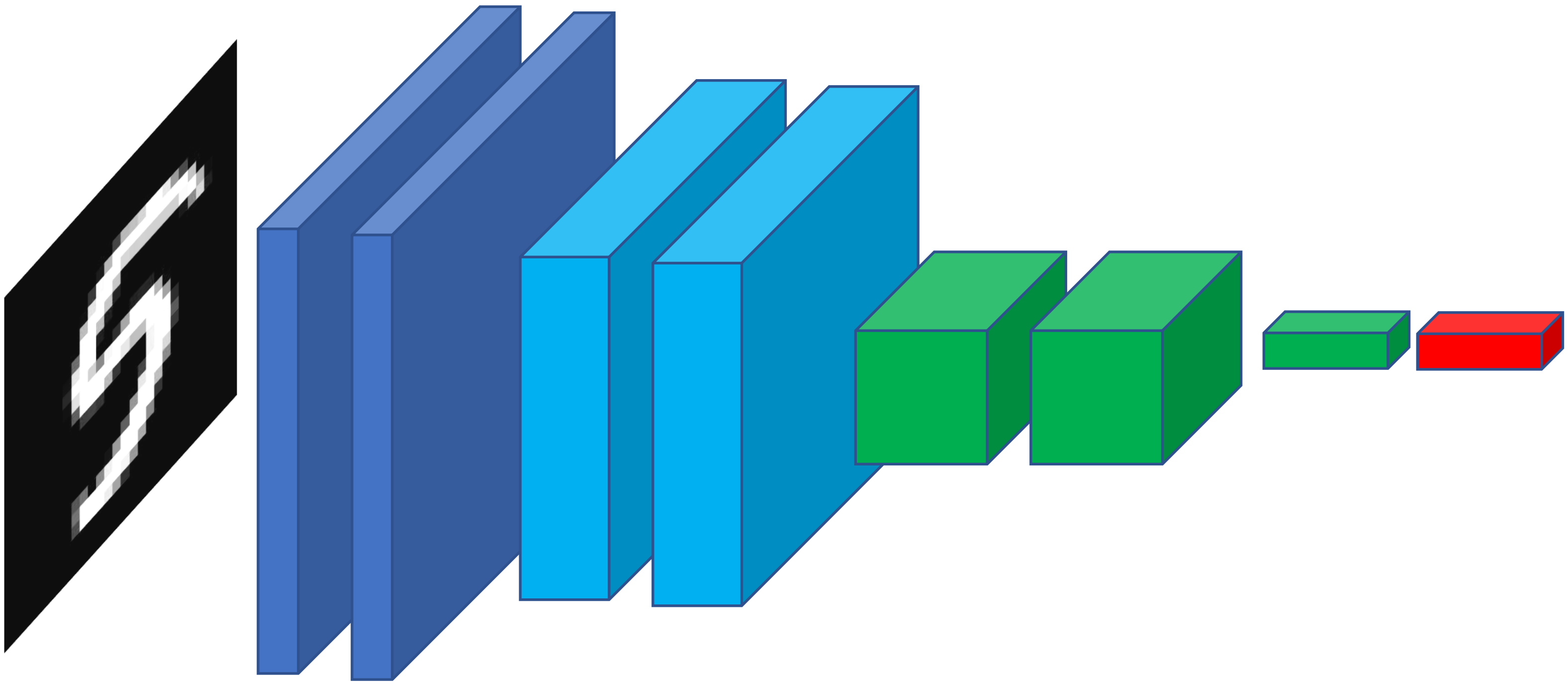
$$h(x) = \mathbf{s} \qquad \hat{p}(y = k|x) \propto e^{s_k} \qquad \hat{p}(y = k|x) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

$$L(h(x), y) = -\log \hat{p}(y|x)$$

Building a convolutional network



Building a convolutional network



MNIST Classification

Method	Error rate (%)
Linear classifier over pixels	12

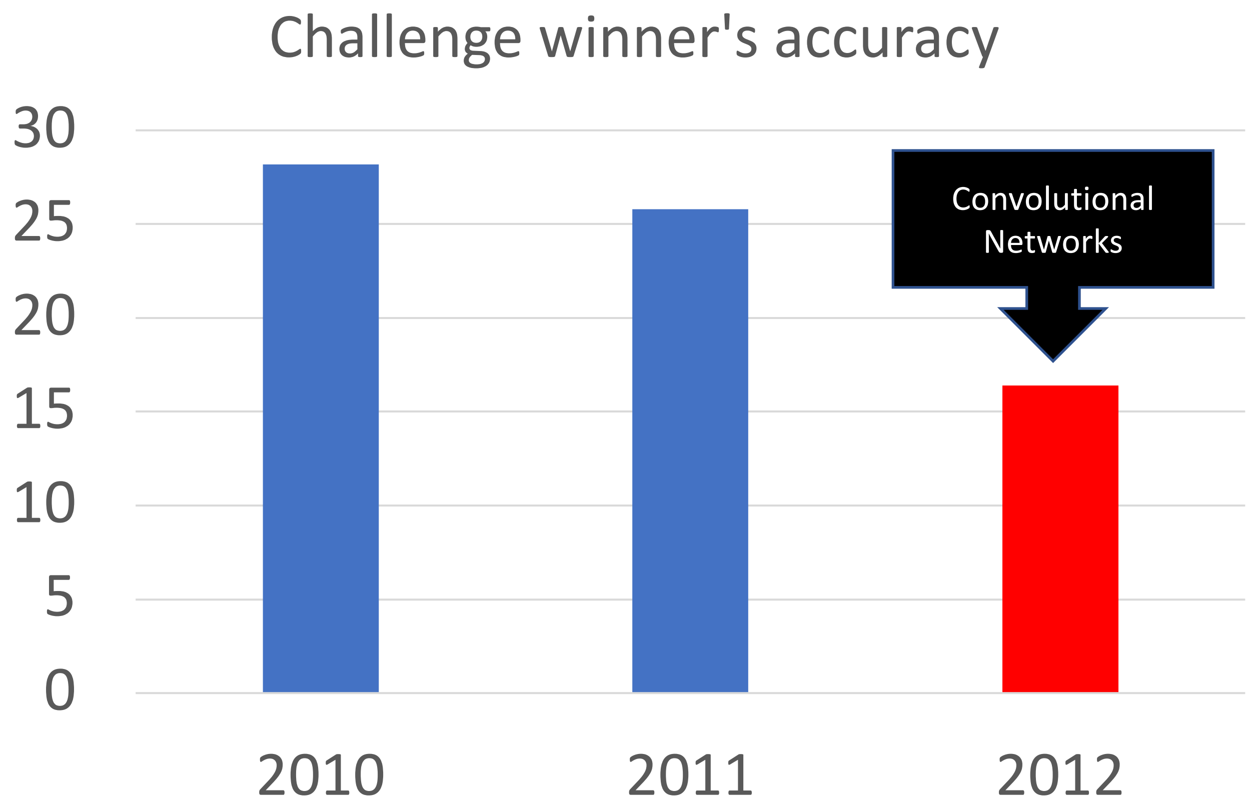
ImageNet

- 1000 categories
- ~1000 instances per category

Olga Russakovsky*, Jia Deng*, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. (* = equal contribution) **ImageNet Large Scale Visual Recognition Challenge**. *International Journal of Computer Vision*, 2015.

ImageNet

- Top-5 error: algorithm makes 5 predictions, true label must be in top 5
- Useful for incomplete labelings


Challenge winner's accuracy
30
25
20
15
10
5
0
Convolutional Networks
2010
2011
2012