CoE202
Fundamentals of Artificial intelligence
<Big Data Analysis and Machine Learning>

Neural Networks Techniques

Prof. Young-Gyu Yoon
School of EE, KAIST

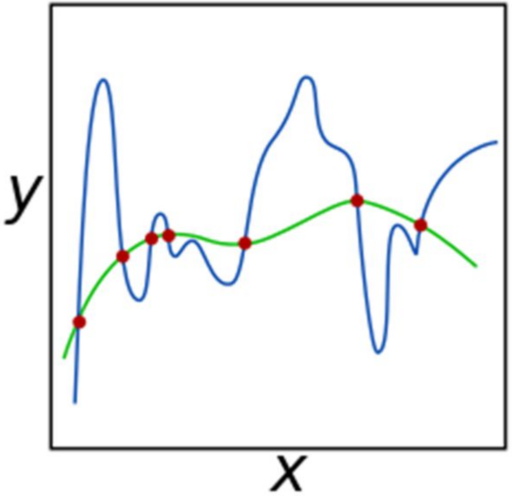
Contents

- Recap
  - Biological vision
  - Convolution operation
  - ConvNet as a special case of NN
  - ConvNet as a generalization of NN
  - Building block of ConvNet

- Regularization methods
- Optimization methods
- NN architectures

Regularization in Optimization

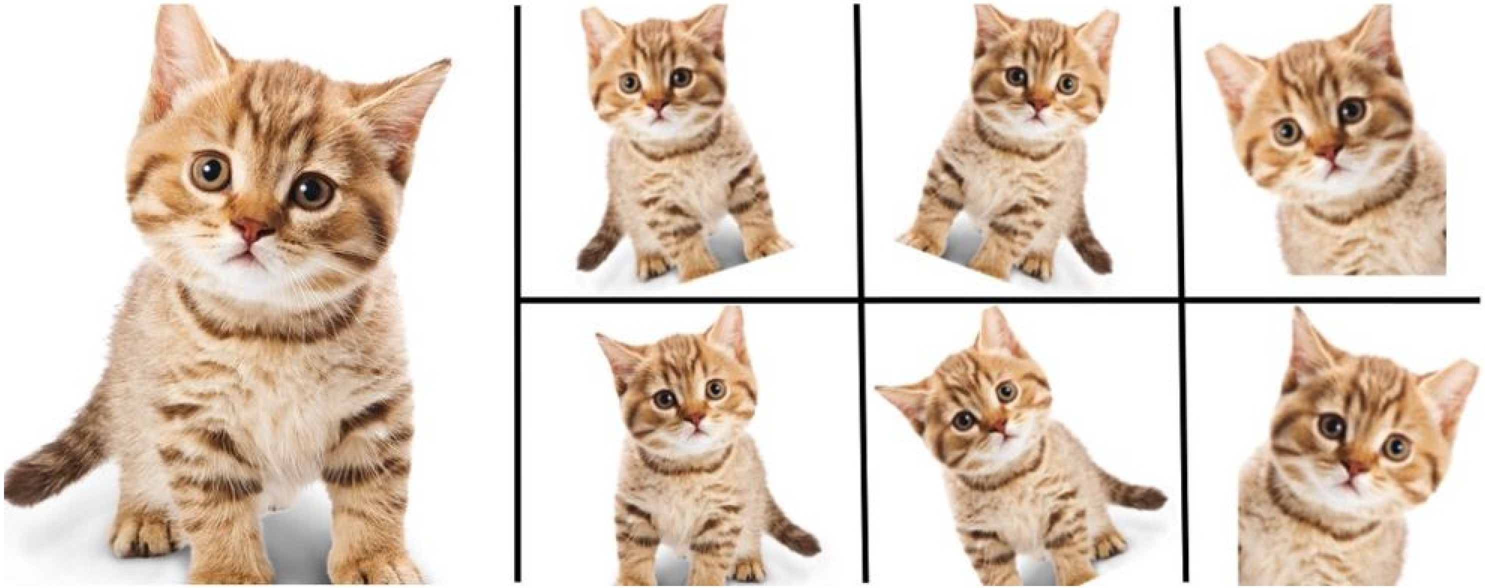
- **Regularization**: is the process of adding information in order to solve an ill-posed problem or to prevent overfitting



Green: with regularization
Blue: without regularization

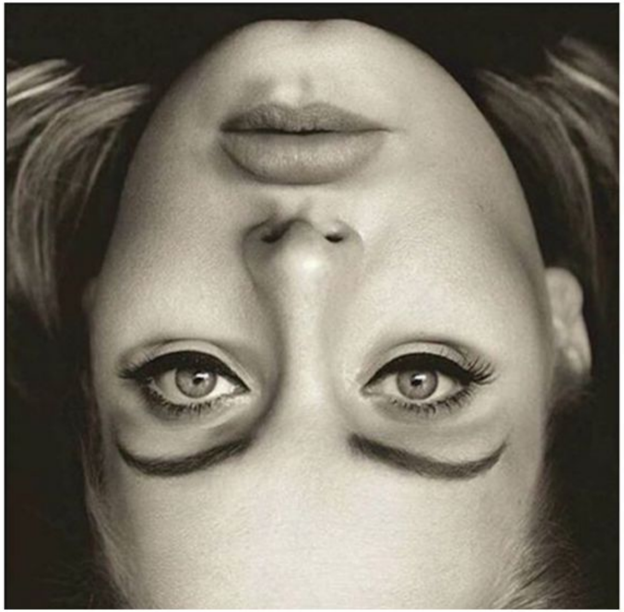
- Modify loss function
- Give constraints to the network
- Decrease network size
- Quit training before the network over-fits
- Add “noise” to data

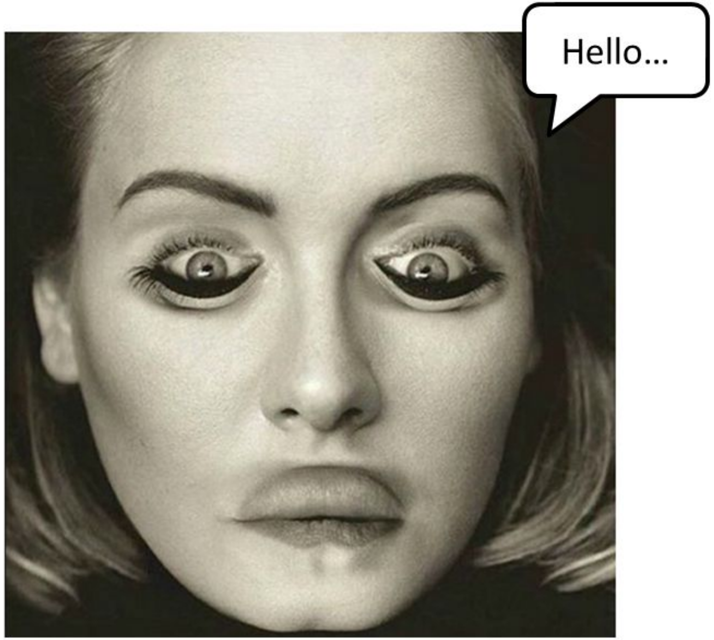
Data augmentation



- Shift, crop, rotate, brightness adjustment (and more)
  - As long as we believe it is realistic enough …

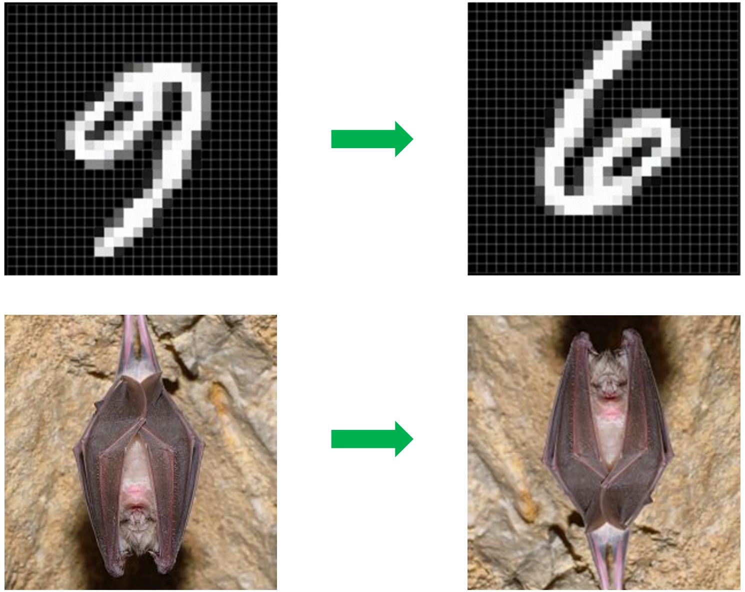
Data augmentation?





- Remember that even human vision is orientation dependent
  - Rotation (and other operations) may or may not alter the content, so we need to think twice before we do it

Data augmentation?



- The examples illustrate why we should think twice and check if the transformation may alter the contents!

L2 (or L1) regularization

- **Problem**
  - While training, the optimizer may tend to increase the absolute value of the parameters
  - This may result in overfitting and/or floating point overflow

- **Solution**: modification of loss function

$$\mathcal{L}_{new} = \mathcal{L}_{origianl} + \boxed{\lambda \mathcal{L}_{L2}}$$

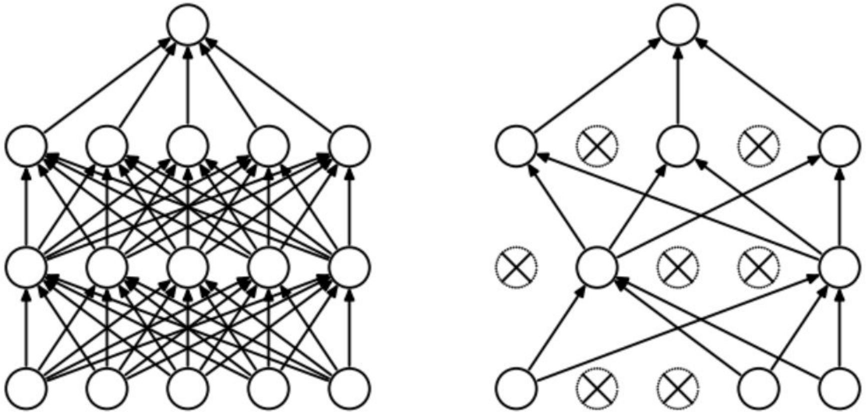
Add another term in our loss function to penalize for large absolute values of parameters

$$\mathcal{L}_{L2} = \sum_i w_i^2$$

$$-\nabla \mathcal{L}_{new} = -\nabla \mathcal{L}_{origianl} - \lambda \nabla \mathcal{L}_{L2}$$

$$-\nabla \mathcal{L}_{L2} = -2 \sum_i w_i$$

Dropout

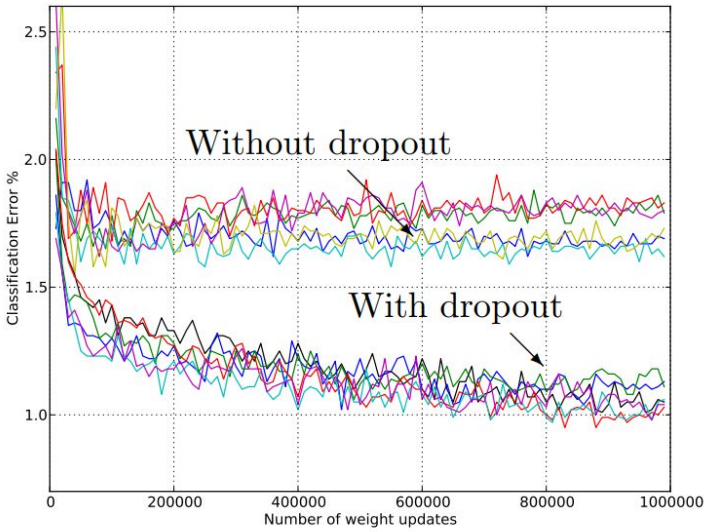


(a) Standard Neural Net (b) After applying dropout.

- A simple technique to prevent overfitting
- (while training) we randomly “sub-sample” part of the neurons in network
- (while testing) we use all of the neurons

Srivastava et al, Dropout: A Simple Way to Prevent Neural Networks from Overfitting (2014)

Dropout



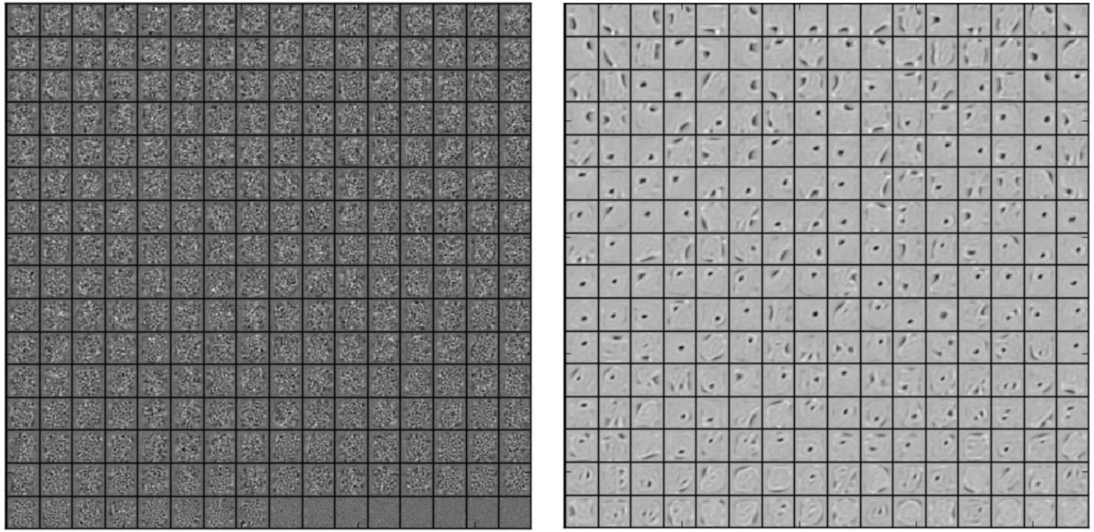
Q) Is this training accuracy or validation/test accuracy?

- **Training**
  - We assign probability p to each neuron (the probability that each neuron will be present or not)
  - Then, we are train the network while randomly sub-sampling it
    - Randomly sub-sampling is based on the probability p
    - We are training a network in a way such that 'a randomly selected part of the network' will be able to solve the task
- **Testing**
  - For validation & testing, we enable all neurons

Srivastava et al, Dropout: A Simple Way to Prevent Neural Networks from Overfitting (2014)

Dropout

7.1 Effect on Features



(a) Without dropout (b) Dropout with $p = 0.5$.

Srivastava et al, Dropout: A Simple Way to Prevent Neural Networks from Overfitting (2014)

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

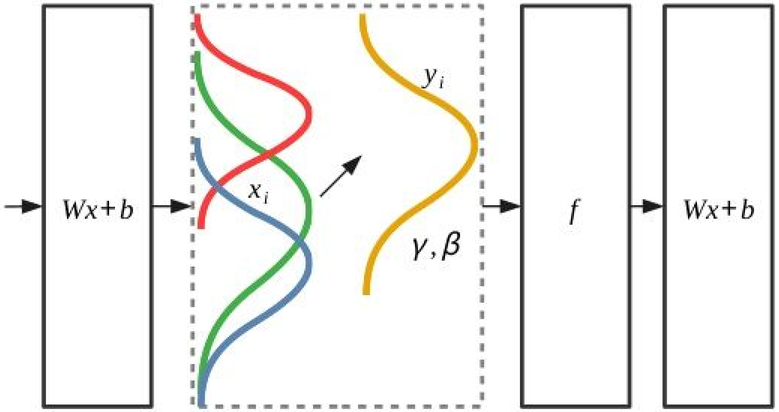
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

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

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

Batch normalization

Ensure the output statistics of a layer are fixed.



S. Loffe and C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift (2015)

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

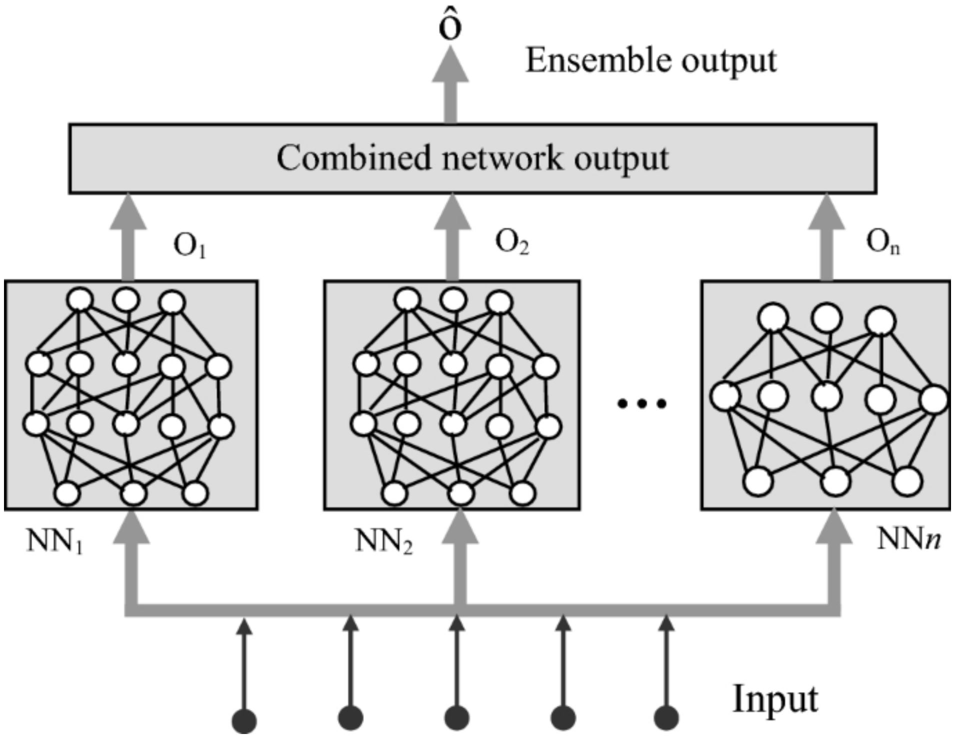
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

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

- **Training**
  - For each mini-batch, the mean and variance is calculated and the data is normalized
  - The normalized data is scaled and shifted (linear transform) with trainable parameters γ and β
- **Testing**
  - For validation & testing, we use mean and variance from the entire dataset (not from a batch)

S. Loffe and C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift (2015)

Model Ensembles



- **Problem**:
  - There always is a bias in our output (especially with high capacity models)

- **Solution**:
  - Train multiple networks and take the averaged output (assuming that the bias will average out)

RMSprop

- **Problem**: $$\theta^{(k+1)} = \theta^{(k)} - \gamma \nabla \mathcal{L}(\theta^{(k)})$$
  - the size of the gradient (used for gradient descent) may vary widely in magnitudes
    → some parameters experience large updates, some do not
    → proper learning rate is different form different parameters

- **Solution**: use adapter learning rates for different parameters
  - divide the gradient (for each parameter) by the square root of the moving average of the squared gradient

$$MA^{(k)} = \beta MA^{(k-1)} + (1-\beta)(\nabla \mathcal{L}(\theta^{(k)}))^2$$

"moving average" of squared gradient

$$\theta^{(k+1)} = \theta^{(k)} - \frac{\gamma}{\sqrt{MA^{(k)}}} \nabla \mathcal{L}(\theta^{(k)})$$

divide by the square root

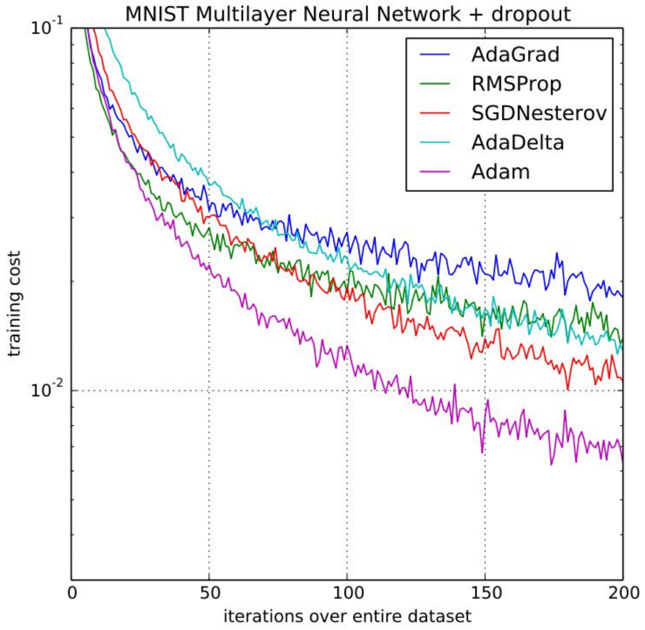
Adam: combine momentum and RMSprop

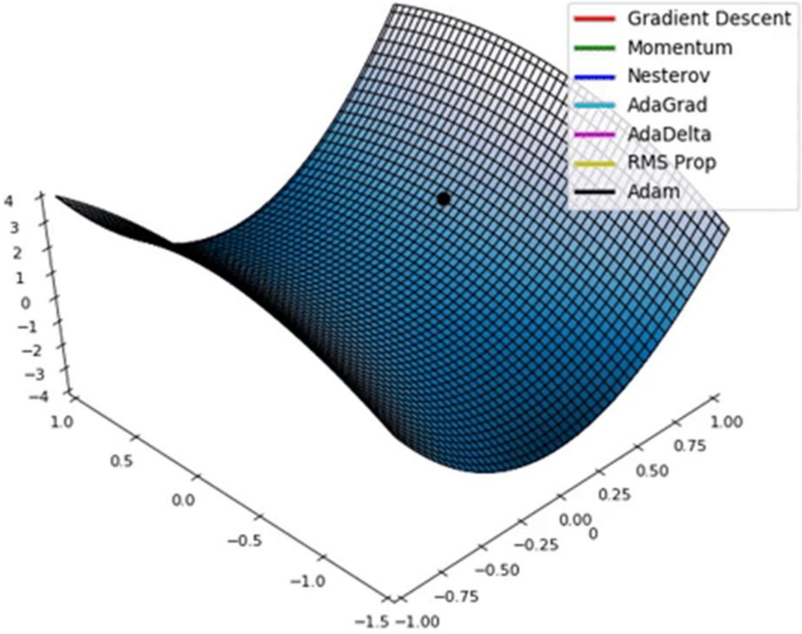
Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

D. P. Kingma, J. Ba, Adam: A Method for Stochastic Optimization (2017)

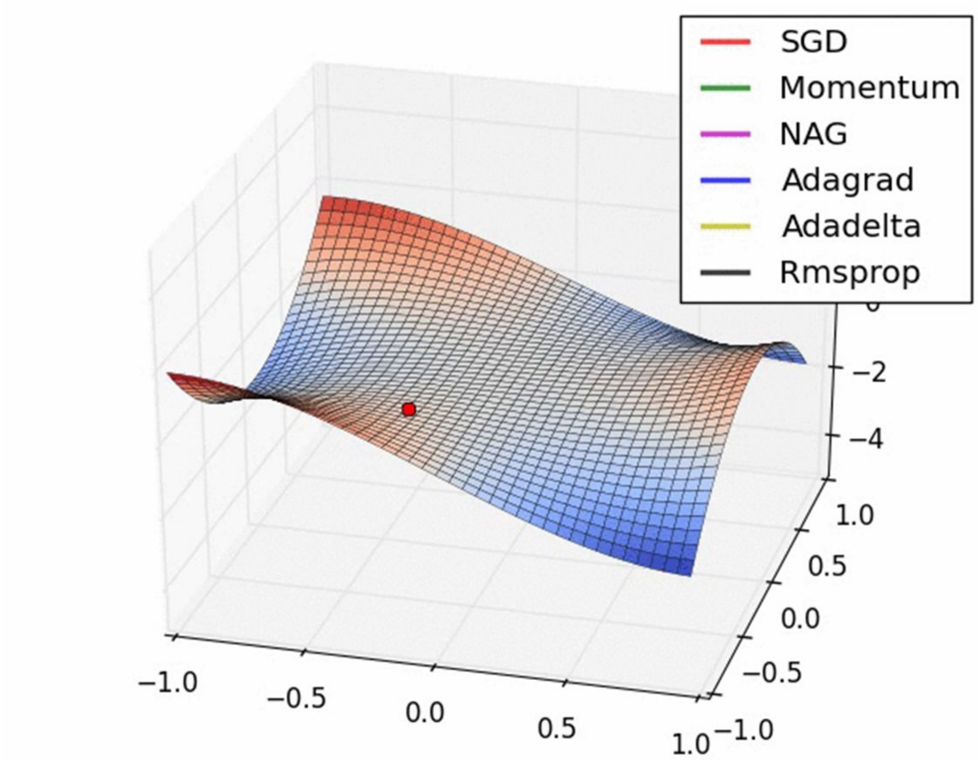
Comparison of optimization methods



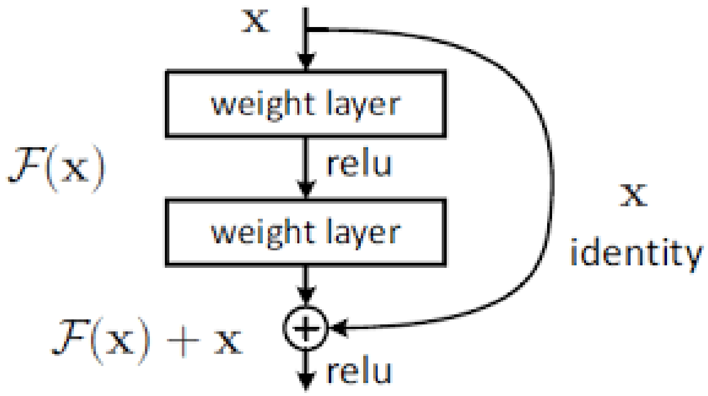


D. P. Kingma, J. Ba, Adam: A Method for Stochastic Optimization (2017)

Comparison of optimization methods



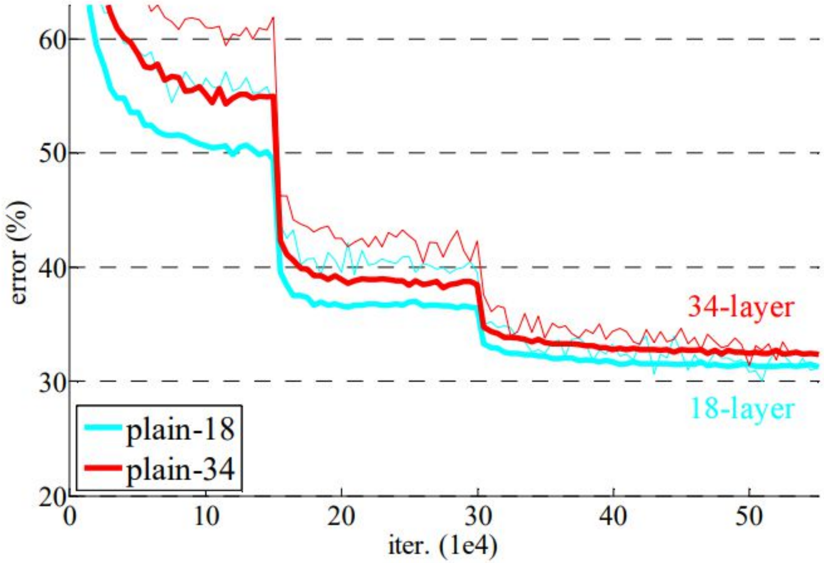
Skip Connection (ResNet)



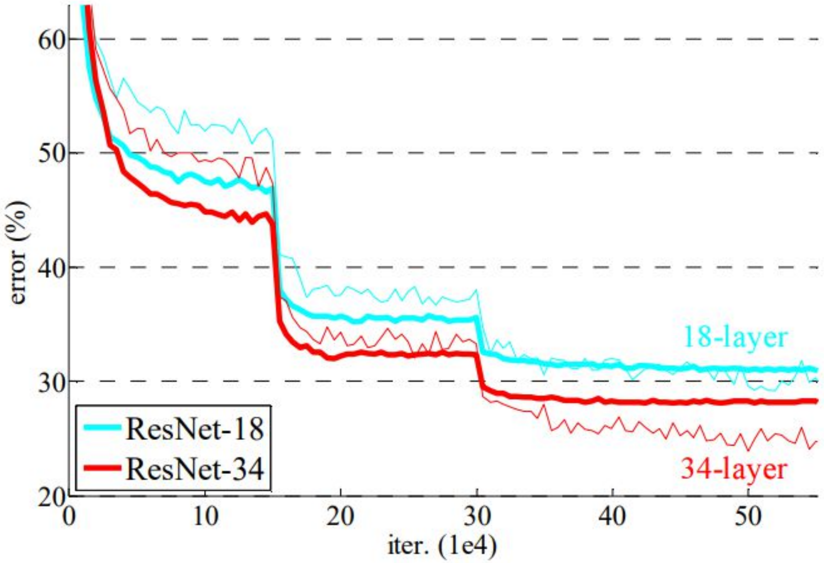
- **Problem**: gradient vanishing
  - The deeper, more the powerful
  - However, a very deep network is difficult to train
  - Back-propagation through very large number of layers needs to be done → it becomes difficult to properly train “front-end” layers
- **Solution**: make skip connections
  - Add direction connection paths in our network, so back-propagation can be done without gradient vanishing problem

K. He et al, Deep Residual Learning for Image Recognition (2015)

Skip Connection (ResNet)



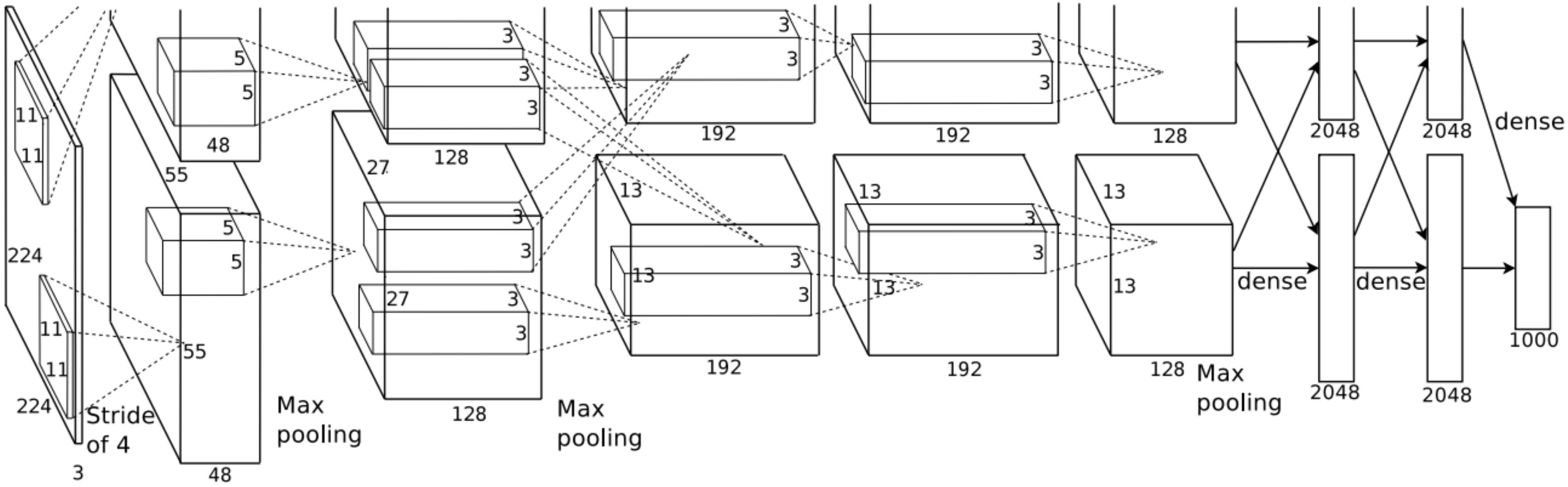
Without skip connections



With skip connections

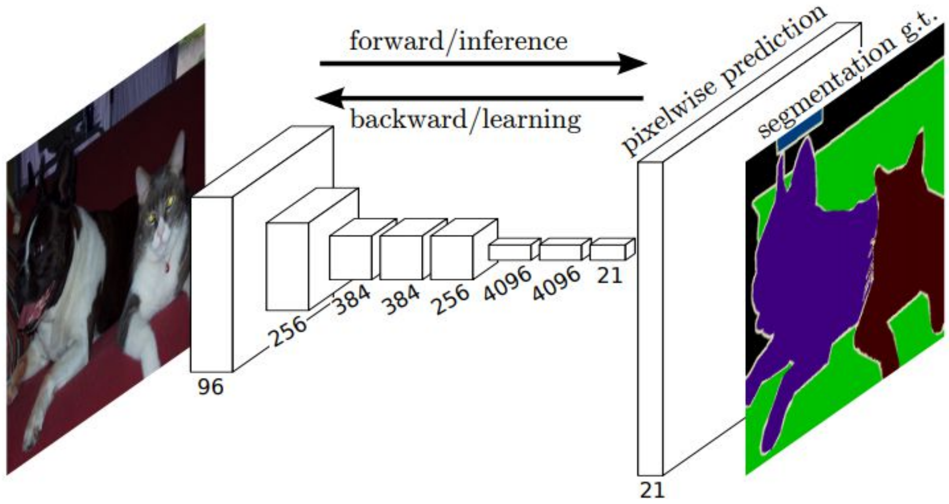
K. He et al, Deep Residual Learning for Image Recognition (2015)

AlexNet: winner of ImageNet challenge 2012



A. Krizhevsky et al, ImageNet Classification with Deep Convolutional Neural Networks (2012)

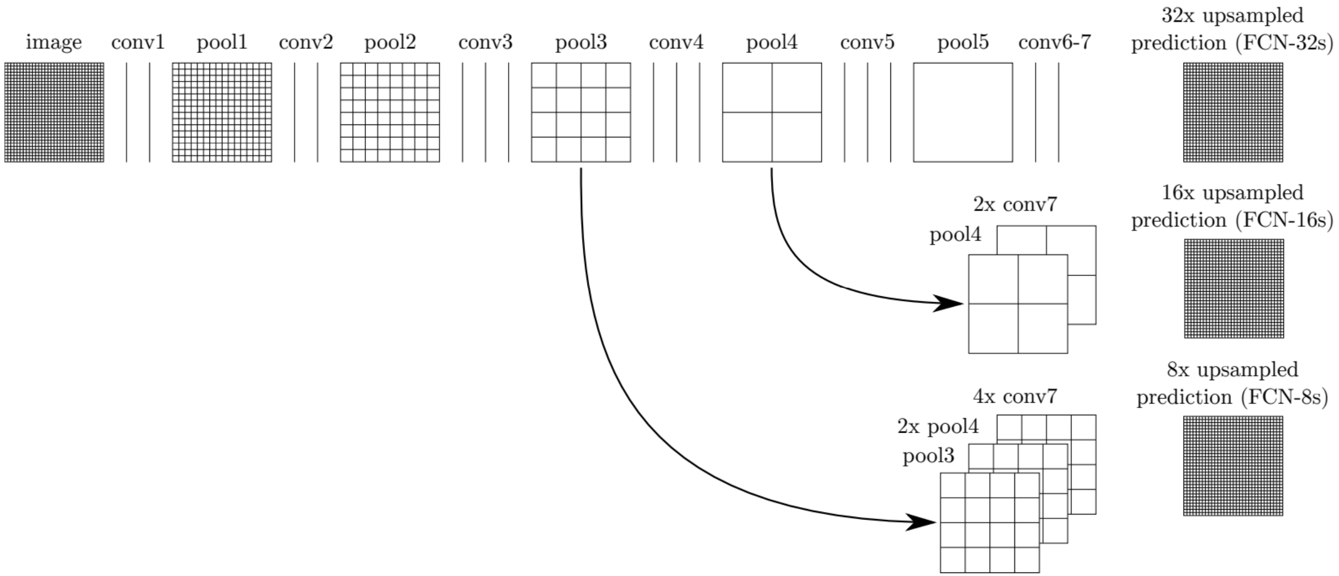
Fully Convolutional Network



- We want more than just image classification → we want to know what's where
- Conceptually speaking, we want X: 512x512x3 → Y: 10x512x512, instead of X: 512x512x3 → Y: 10x1 mapping

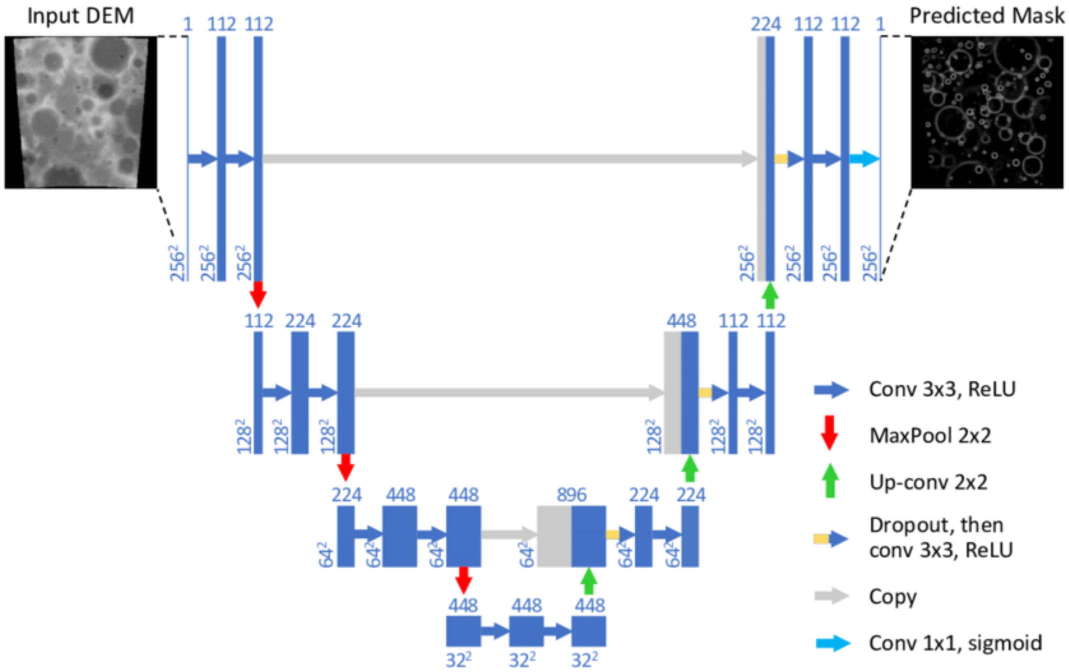
J. Long et al, Fully Convolutional Networks for Semantic Segmentation (2015)

Fully Convolutional Network



- We want more than just image classification → we want to know what's where
- Conceptually speaking, we want X: 512x512x3 →Y: 10x512x512, instead of X: 512x512x3 → Y: 10x1 mapping

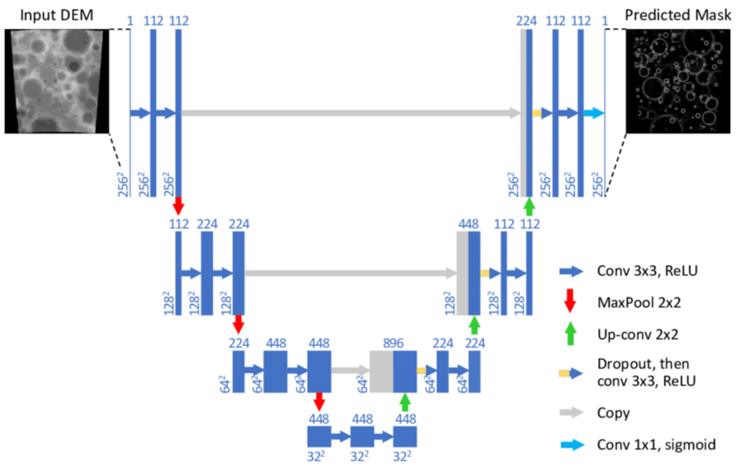
U-net



- Network architecture suited for X: 512x512x3 → Y: 10x512x512

O. Ronneberger et al, U-Net: Convolutional Networks for Biomedical Image Segmentation (2015)

U-net



- **Key properties**
  - **Skip-connection**: high resolution information is directly fed-through
  - **Contraction-and-expansion**: abstraction (high level information extraction) is performed and used for pixel level segmentation
  - **Large receptive field**: convolution operations for image images corresponds to large-kernel convolution for full-sized images
  - **Multi-resolution processing**: information is processed at multiple resolutions

Summary

- Regularization methods
  - Data augmentation
  - L1/L2 regularization
  - Dropout
  - Batch normalization
  - Model ensemble
- Optimization methods
  - RMSprop
  - Adam
- NN architectures
  - ImageNet
  - FCN
  - U-net

References

- Website
  - CS231n course website: https://cs231n.github.io/