Introduction to Deep Learning

Convolutional Neural Networks (2)

Prof. Songhwai Oh

ECE, SNU

STOCHASTIC GRADIENT DESCENT

Gradient Descent

$\mathbf{w} \leftarrow$ any point in the parameter space

loop until convergence **do**

  for each w_i **in** w **do**

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$$

α: step size or learning rate.

$$\begin{aligned}
\frac{\partial}{\partial w_i} Loss(\mathbf{w}) &= \frac{\partial}{\partial w_i} (y - h_{\mathbf{w}}(x))^2 \\
&= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i} (y - h_{\mathbf{w}}(x)) \\
&= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i} (y - (w_1 x + w_0))
\end{aligned}$$

$$\frac{\partial}{\partial w_0} Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x))$$

$$\frac{\partial}{\partial w_1} Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)) \times x$$

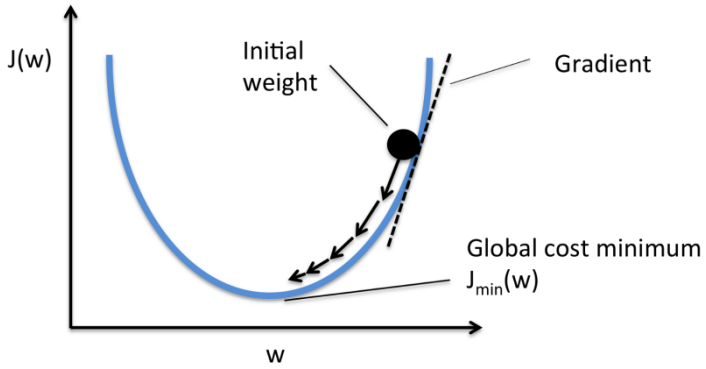
$$w_0 \leftarrow w_0 + \alpha\,(y - h_{\mathbf{w}}(x))\,; \quad w_1 \leftarrow w_1 + \alpha\,(y - h_{\mathbf{w}}(x)) \times x$$

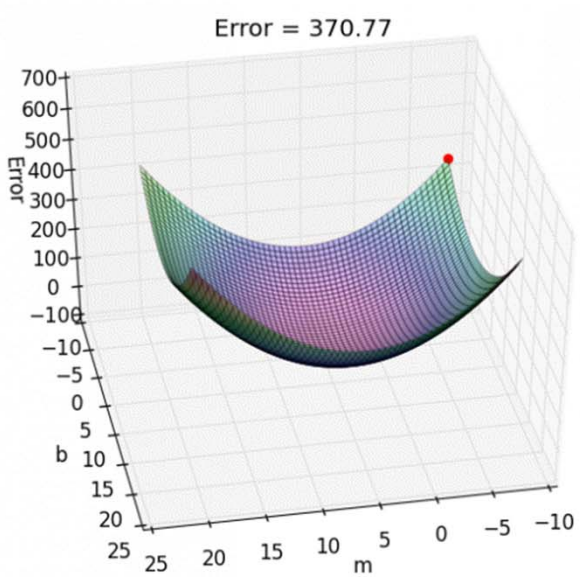
Batch gradient descent: (Steepest descent)

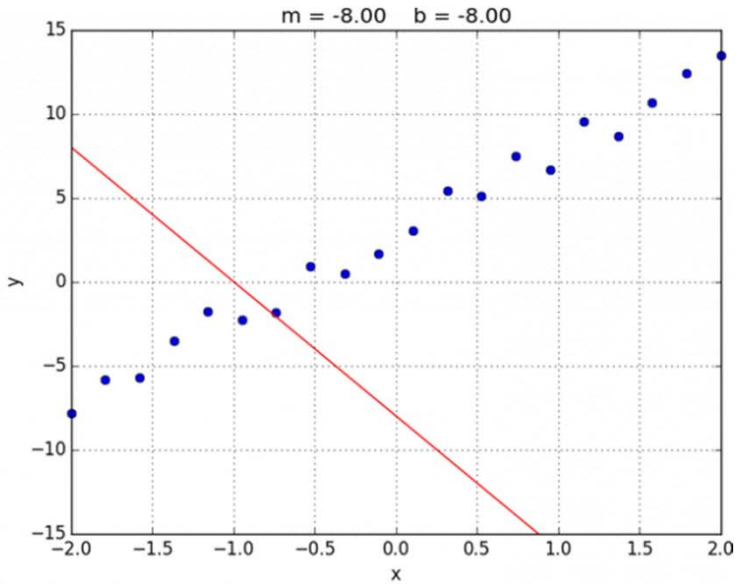
$$w_0 \leftarrow w_0 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j))\,; \quad w_1 \leftarrow w_1 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j)) \times x_j$$

Stochastic gradient descent: processes one data point at a time

Gradient Descent







Source: https://hackernoon.com/dl03-gradient-descent-719aff91c7d6?gi=7c0ea3c7fc76
https://alykhantejani.github.io/images/gradient_descent_line_graph.gif

Loss Functions

- t: target value
- y: network output

- **Cross entropy loss** (or log loss) [t, y: distribution]

$$Loss = \sum_{k=1}^{K} -t_k \log y_k$$

- **Hinge loss** [$t = \pm 1$, for classification]

$$Loss = \max(0, 1 - t \cdot y)$$

- **Huber loss** [for regression]

$$Loss = \begin{cases} \frac{1}{2}(t-y)^2 & \text{if} \quad |t-y| < \delta \\ \delta|t-y| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases}$$

- **Kullback-Leibler (KL) divergence** [t, y: distribution]

$$Loss = \sum_{k=1}^{K} t_k \log\left(\frac{t_k}{y_k}\right)$$

- **Mean absolute error (MAE)** (or L1 loss)

$$Loss = \frac{1}{N}\sum_{n=1}^{N}|t_n - y_n|$$

- **Mean squared error (MSE)** (or L2 loss)

$$Loss = \frac{1}{N}\sum_{n=1}^{N}(t_n - y_n)^2$$

Momentum

- Stochastic gradient descent can be slow on a plateau
- **Momentum method** (Polyak, 1964)
  - Accelerate gradient descent using the momentum (speed) of previous gradients

$$v_{t+1} = \alpha v_t - \eta \nabla_w Loss$$
$$w_{t+1} = w_t + v_{t+1} = w_t - \eta \nabla_w Loss + \alpha v_t$$

  - v_t: velocity
  - w_t: parameter
  - α: decay rate (≤ 1)
  - η: learning rate

RMSprop

- RMSProp: Root Mean Square Propagation

$$v_{t+1} = \gamma v_t + (1-\gamma)(\nabla_w Loss)^2$$
$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_{t+1}}} \nabla_w Loss$$

  - v_t: moving average of magnitudes of previous gradients
  - w_t: parameter
  - γ: forgetting factor ($0 \leq \gamma < 1$)
  - η: learning rate
  - Elementwise squaring and square-rooting
  - Works well under online and non-stationary settings

ADAM

- **ADAM**: Adaptive Moment Estimation

$$m_{t+1} = \beta_1 m_t + (1-\beta_1)\nabla_w Loss$$
$$v_{t+1} = \beta_2 v_t + (1-\beta_2)(\nabla_w Loss)^2$$
$$\hat{m} = \frac{m_{t+1}}{1-\beta_1{}^{t+1}} \qquad \hat{v} = \frac{v_{t+1}}{1-\beta_2{}^{t+1}}$$
$$w_{t+1} = w_t - \eta \frac{\hat{m}}{\sqrt{\hat{v}}+\epsilon}$$

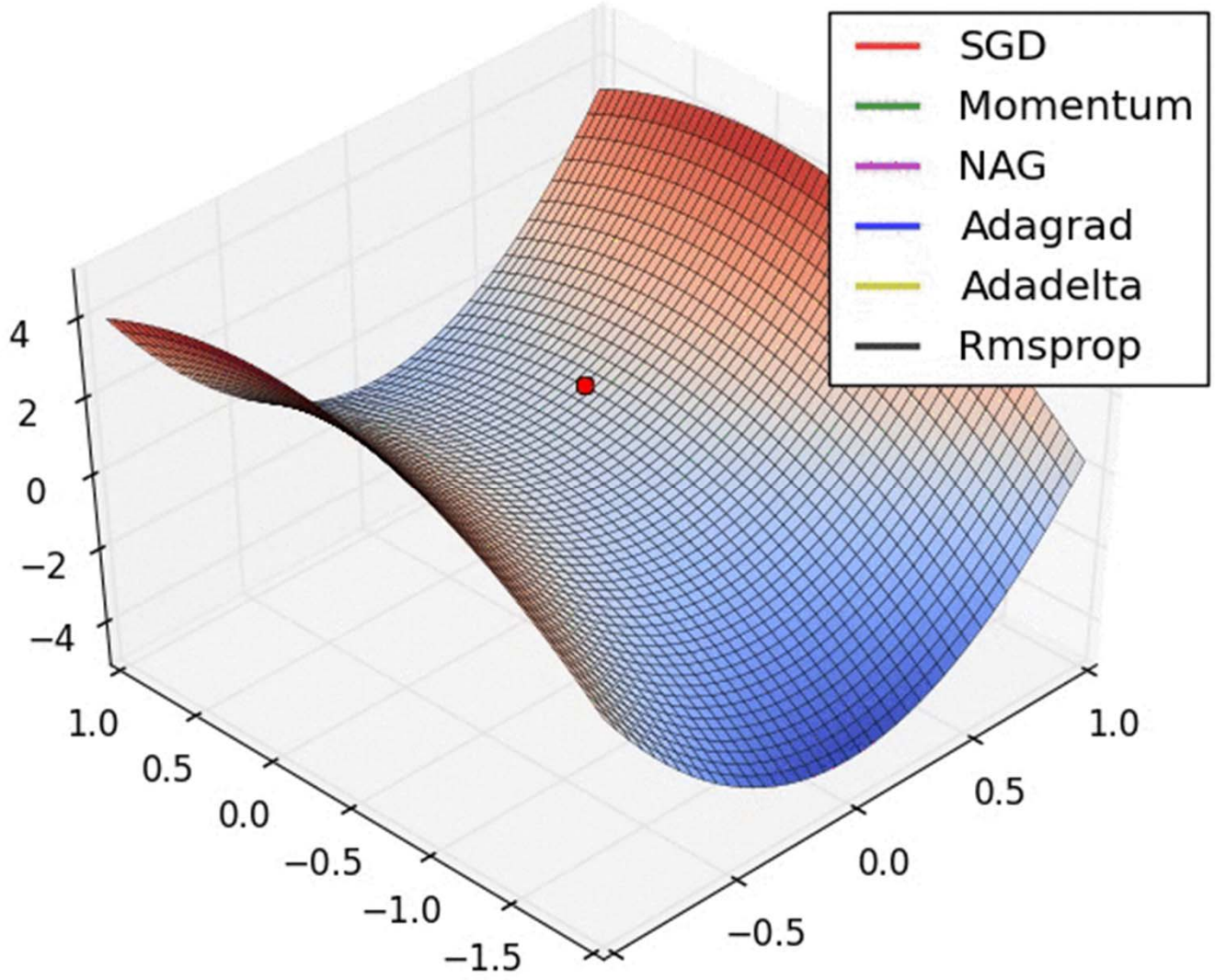
* Combines AdaGrad and RMSprop

Initialization bias correction

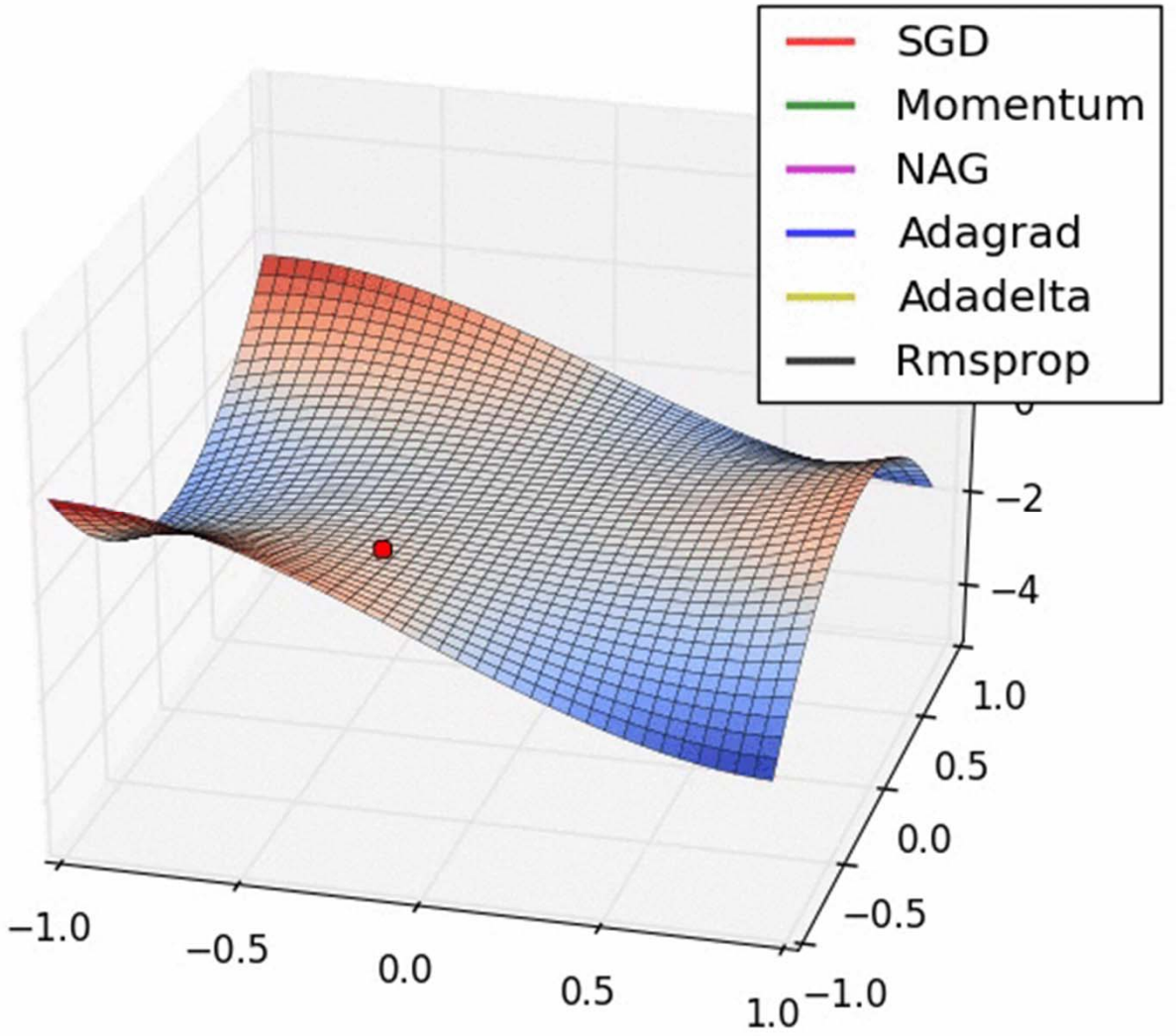
- m_t: moving average of previous gradients
- v_t: moving average of magnitudes of previous gradients
- w_t: parameter
- β_1, β_2: forgetting factor ($\beta_1, \beta_2 \in [0,1)$)
- η: learning rate
- ϵ: a small number
- Elementwise squaring and square-rooting

Diederik P. Kingma, Jimmy Ba, "**Adam: A method for Stochastic Optimization**," ICLR 2015.

Valley

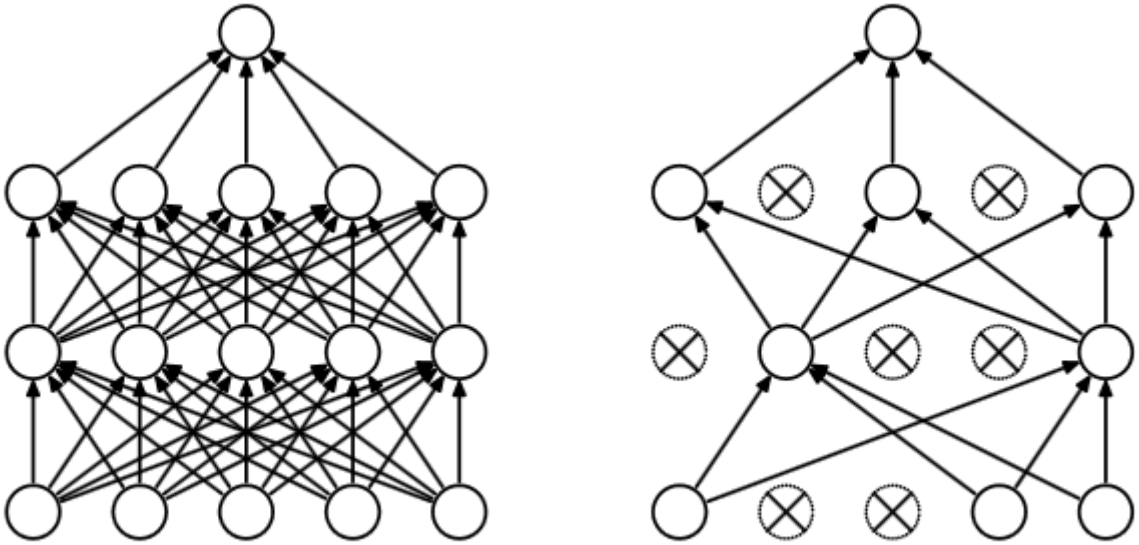

SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
4
2
0
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.5
1.0
0.5
0.0
−0.5

Saddle Point



DROPOUT, BATCH NORMALIZATION

Dropout



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

- A regularization method to address the overfitting problem
- Randomly drop units during training
- Each unit is retained with probability p (hyperparameter), independent of other units

Srivastava, Hinton, Krizhevsky, Sutskever, Salakhutdinov. "**Dropout: a simple way to prevent neural networks from overfitting**." *Journal of Machine Learning Research* 15, no. 1 (2014): 1929-1958.

Batch Normalization

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

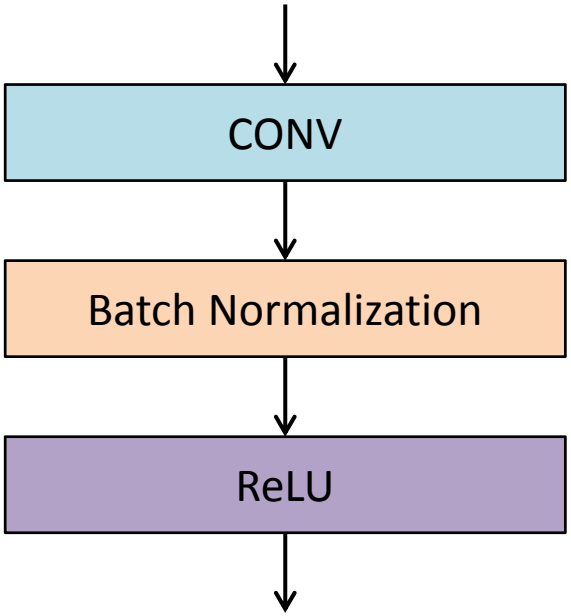
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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



- To reduce internal **covariate shift** (changes in input distribution)
- Faster convergence (requires only 7% of training steps)
- Applied before a nonlinear activation unit by normalizing each dimension
- During testing, fixed mean and variance (estimated during training) are used

Ioffe, Szegedy, "**Batch normalization: Accelerating deep network training by reducing internal covariate shift**," ICML, 2015.

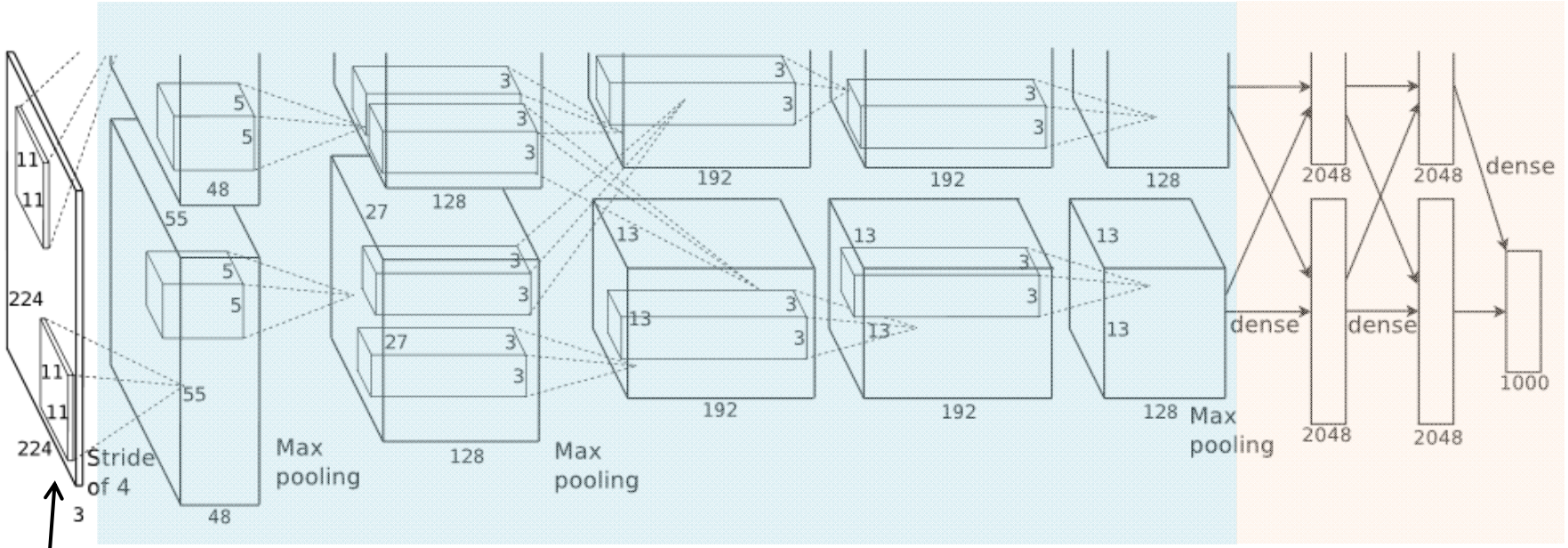
Accelerating BN Networks

- Increase learning rate (x5, x30)
- Remove dropout
  - BN can address the overfitting problem similar to dropout
- Reduce the weight of L2 regularization (x5)
- Accelerate the learning rate decay (x6)
- Remove local response normalization
- Shuffle training examples more thoroughly
- Reduce the photometric distortions

Ioffe, Szegedy, "**Batch normalization: Accelerating deep network training by reducing internal covariate shift**," ICML, 2015.

POPULAR CNN ARCHITECTURES

AlexNet (2012)



5 convolutional layers

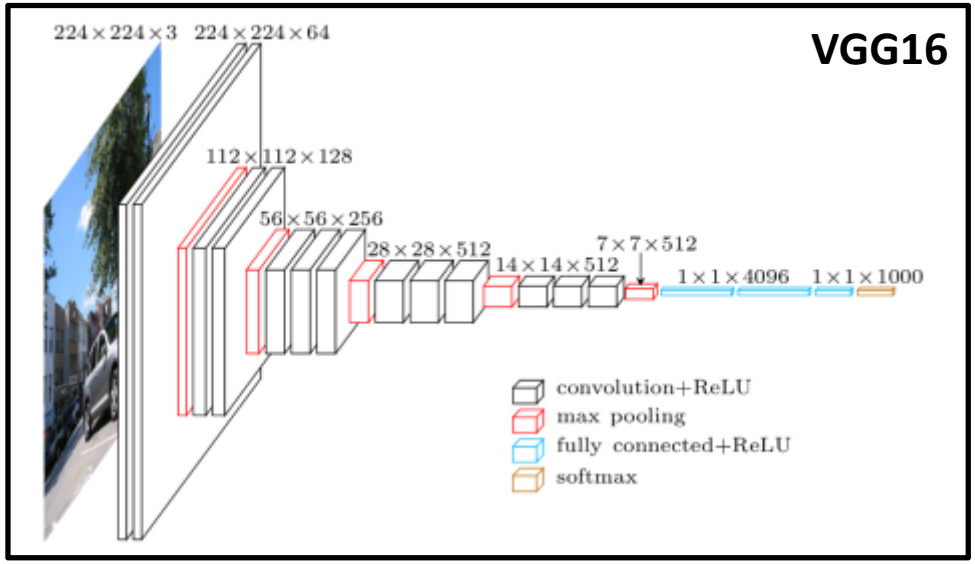
3 fully connected layers

Learned 11x11x3 filters

Key ideas:

- Rectified Linear Unit (ReLU): an activation function
- GPU implementation (2 GPUs)
- Local response normalization, Overlapping pooling
- Data augmentation, Dropout (p=0.5)
- 62M parameters

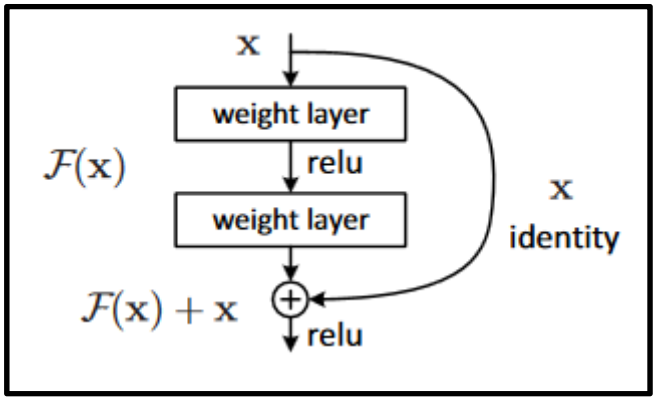
VGGNet (2015)



Source: https://www.cs.toronto.edu/~frossard/post/vgg16/

- Increasing depth with 3x3 convolution filters (with stride 1)
  - More discriminative (compared to larger receptive fields).
  - Less number of parameters
- 1x1 convolution filters
- Dropout (p=0.5)
- VGG16, VGG19
  - 138M parameters for VGG16

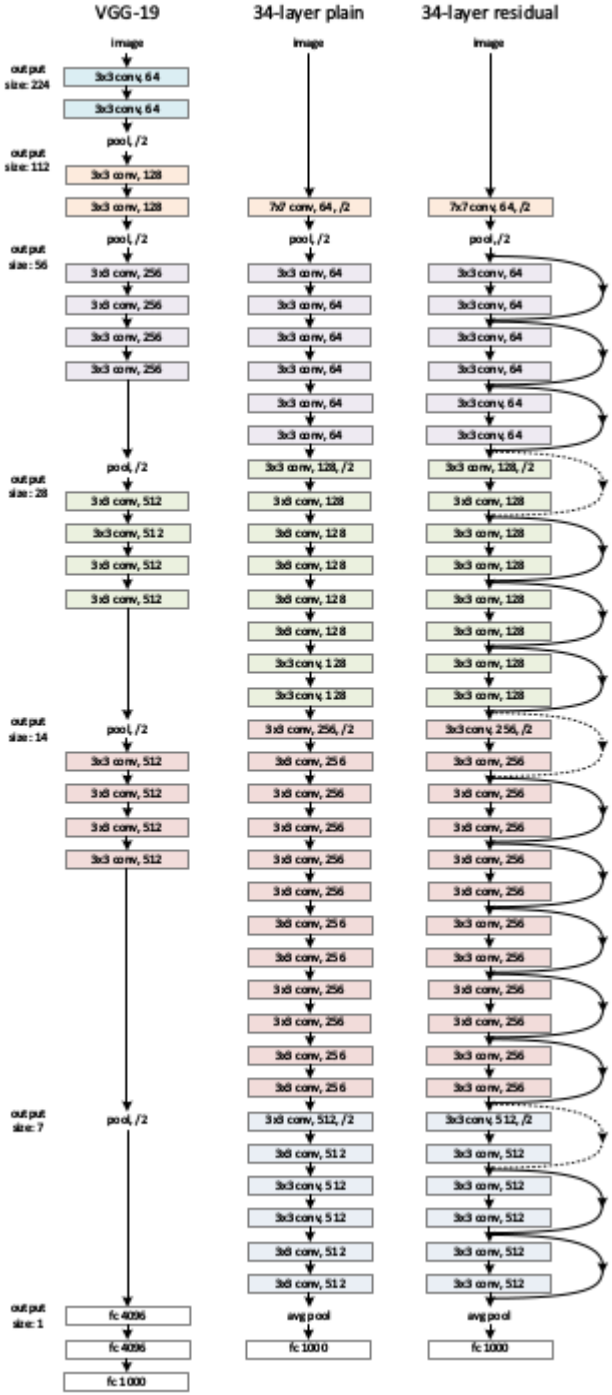
ResNet (2016)



Residual Learning

$\mathcal{F}(x) = H(x) - x$

ResNet-18
ResNet-34
ResNet-50
ResNet-101 (40M)
ResNet-152
ResNet-1202

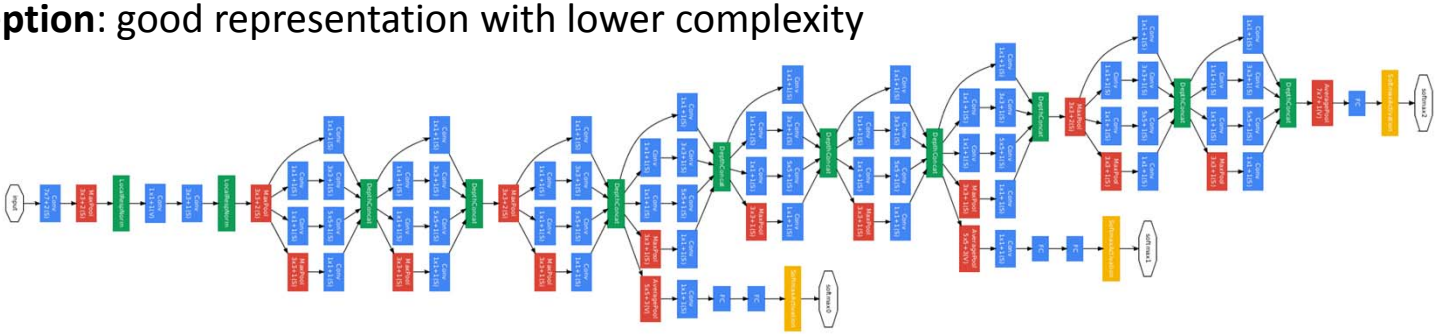


- Degradation problem with deep networks
  - Accuracy saturates as the depth increases
- Solution
  - Instead of learning a mapping $H(x)$ directly, learn the residual function $\mathcal{F}(x)$ instead
  - E.g., learning an identity function
- Shortcut connections

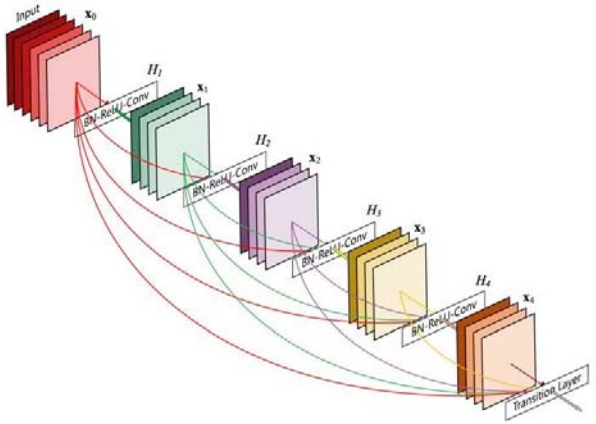
Other Architectures

GoogLeNet (2015, 22 layers, 5M, inception)

- **Inception**: good representation with lower complexity

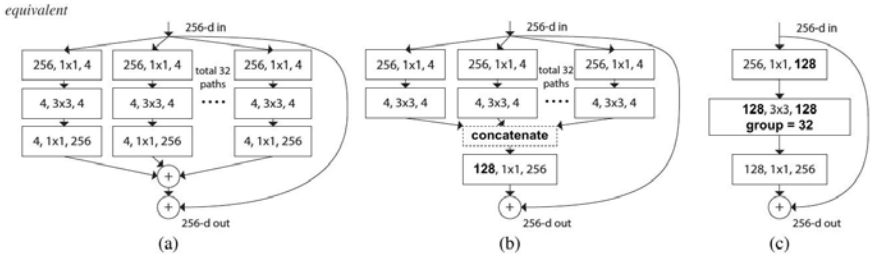


DenseNet (2017, 264 layers, 70M)



ResNeXt

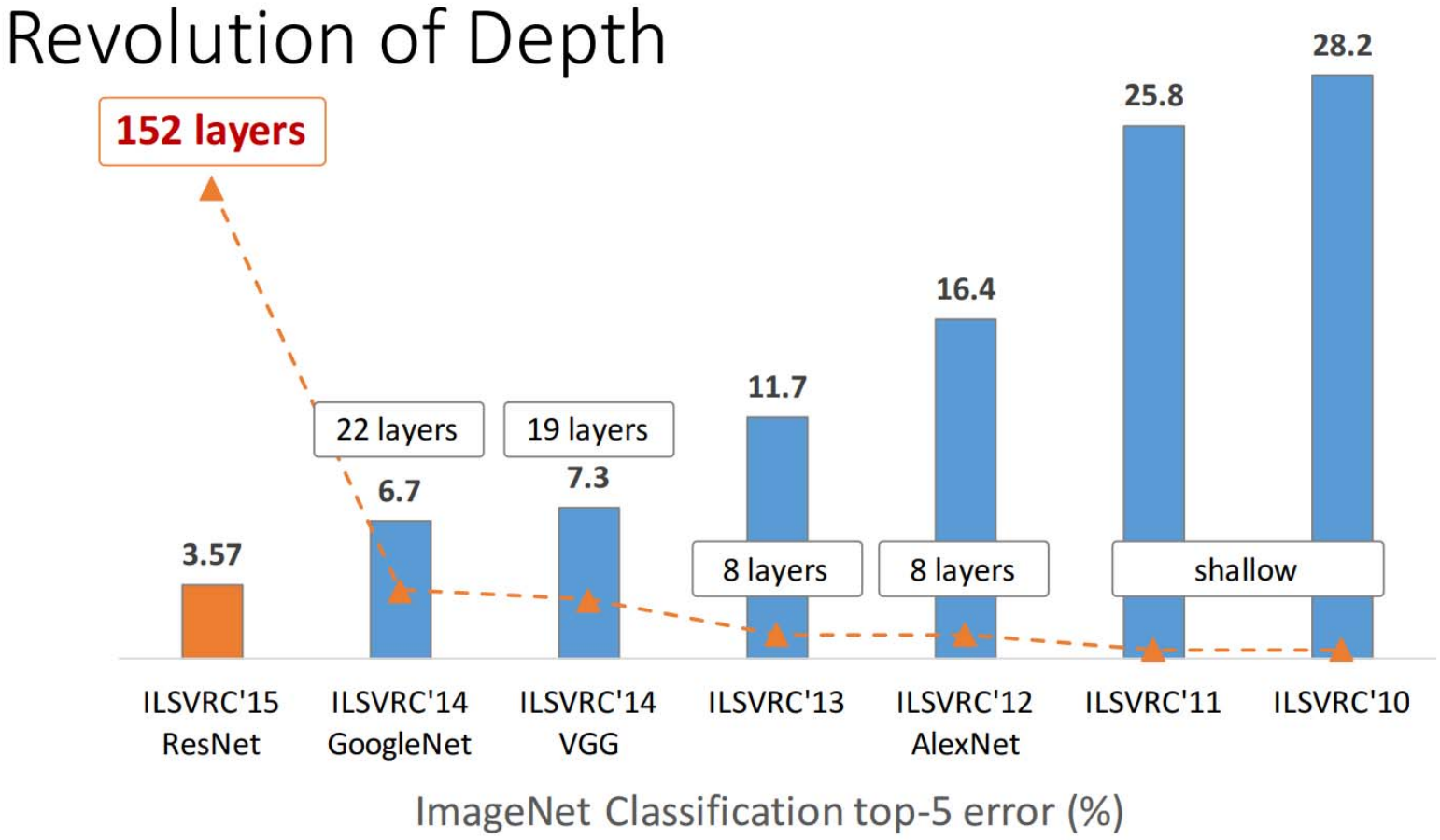
- Multiple paths with the same topology



WRN: Wide Residual Networks

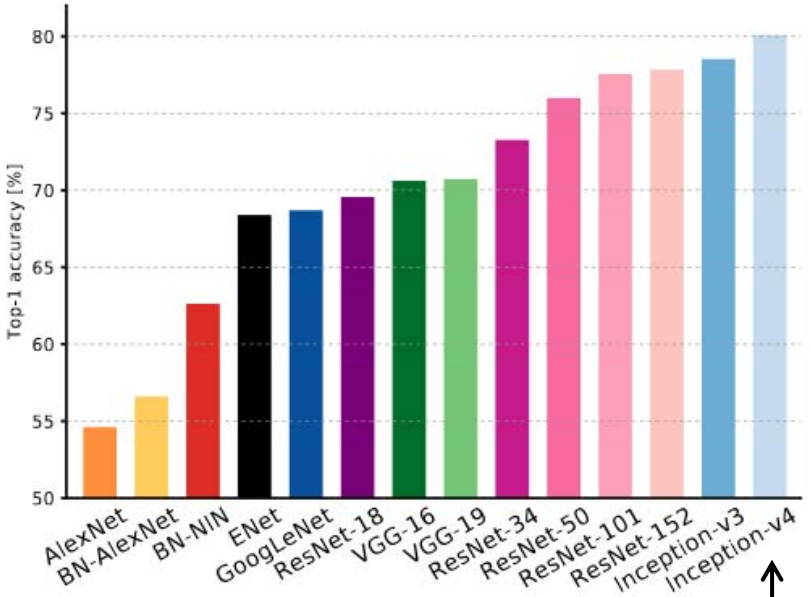
- increase width, not depth for computation efficiency

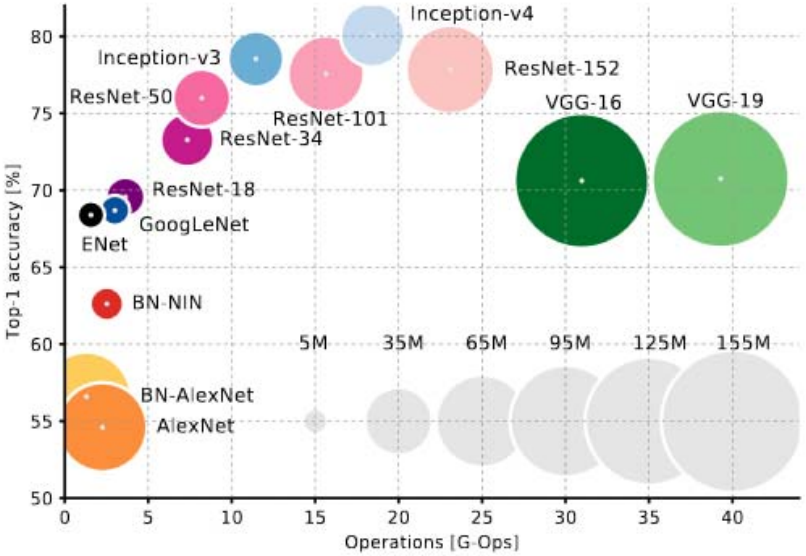
ImageNet Challenge



K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. CVPR, 2016.

Comparison





Inception-v4 = ResNet + Inception

Canziani, Paszke, Culurciello, "**An analysis of deep neural network models for practical applications**," *arXiv preprint arXiv:1605.07678*, 2016.

Wrap Up

- Stochastic gradient descent
  - Momentum
  - RMSprop
  - ADAM
- Dropout
- Batch normalization
- CNN Architectures
  - AlexNet
  - VGGnet
  - ResNet