Lecture 14

Adaptive y
Conv Nets
Res Nets

Q2 on 3/10 at 4:00pm
LIV-1,2, GD/GA
Sol 5 on Monday 3/9
LV

Can we mimic 2-nd order methods "cheaply"? (T,L)

LIV-2

- "2-nd derivative" is Hessian $\frac{\partial^2 \mathcal{L}}{\partial \mathbb{W}^2}$ a $p \times p$ matrix
- We want to get the benefits of 2-nd order in $\mathcal{O}(p)$ time.[4]

- Let $g \in \mathbb{R}^p$ denote a gradient or stochastic gradient (p is still the number of parameters we are training)
- **Momentum (Heavy ball method)**

$$\mathbb{W}^{t+1} \leftarrow \mathbb{W}^t - \gamma\eta g^t + (1-\gamma)\underbrace{(\mathbb{W}^t - \mathbb{W}^{t-1})}_{\text{previous step}} \tag{27}$$

- (many variations exist, e.g. Nesterov method)

- Adaptive learning rates (coming next)

[4] The factor n or n' is ignored, because it does not affect what we do.

Adaptive Learning Rates (LR) – S,T

η = Learning Rate = Step Size

- Typical for SGD
- The LR adapts over iteration t and parameter w_i
- Goal η "normalized" with a factor that avoids large updates

- $g_i^t = \frac{\partial \mathcal{L}}{\partial w_i}(\mathbb{W}^t)$ ← entry i in $g^t \in \mathbb{R}^p$
- Let $G_{i,t} = \sum_{t'=1}^{t} (g_i^{t'})^2$ = sum of squares of gradients for weight i
- AdaGrad algorithm

$$w_i^{t+1} \leftarrow w_i^t - \frac{\eta}{\sqrt{G_{i,t} + \epsilon}} g_i^t. \quad (28)$$

η_i^t "adaptive" ↘ with t

$\begin{bmatrix} \vdots \\ w_i \\ \vdots \end{bmatrix} = W \in \mathbb{R}^p$

- Problem stepsize can't increase if needed

vector $\begin{bmatrix} g_i^1 \\ g_i^2 \\ \vdots \\ g_i^t \end{bmatrix} = \tilde{g}^t \in \mathbb{R}^t$

$\sqrt{G_{i,t}} = \|\tilde{g}^t\|$

Theory of SGD: $\eta^t \sim \frac{1}{t}$

Adaptive Learning Rates (LR) – **S,T**

- Typical for SGD
- The LR adapts over iteration t and parameter w_i
- Goal η "normalized" with a factor that avoids large updates

- $g_i^t = \frac{\partial \mathcal{L}}{\partial w_i}(\mathbb{W}^t)$
- Let $G_{i,t} = \sum_{t=1}^{t} (g_i^{t'})^2$ = sum of squares of gradients for weight i
- ADAGRAD algorithm

$$w_i^{t+1} \leftarrow w_i^t - \frac{\eta}{\sqrt{G_{i,t} + \epsilon}} g_i^t. \quad (28)$$

$G_{i,t} = G_{i,t-1} + (g_i^t)^2$

- Problem stepsize can't increase if needed
- RMSPROP "momentum" (forgetting) $G_{i,t} = 0.9 G_{i,t-1} + 0.1 (g_i^t)^2$

Adaptive Learning Rates (LR) – S,T

- Typical for SGD
- The LR adapts over iteration t and parameter w_i
- Goal η "normalized" with a factor that avoids large updates

- $g_i^t = \frac{\partial \mathcal{L}}{\partial w_i}(\mathbb{W}^t)$
- Let $G_{i,t} = \sum_{t=1}^{t} (g_i^{t'})^2$ = sum of squares of gradients for weight i
- ADAGRAD algorithm

$$w_i^{t+1} \leftarrow w_i^t - \frac{\eta}{\sqrt{G_{i,t} + \epsilon}} g_i^t. \tag{28}$$

- Problem stepsize can't increase if needed
- RMSPROP "momentum" (forgetting) $G_{i,t} = 0.9 G_{i,t-1} + 0.1 (g_i^t)^2$
- ADAM momentum and RMSPROP
  - Hyperparameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$

same as Heavy Ball

$$m_i^t = \beta_1 m_i^{t-1} + (1 - \beta_1) g_i^t \quad \text{momentum for } g \tag{29}$$

$$v_i^t = \beta_2 v_i^{t-1} + (1 - \beta_2)(g_i^t)^2 \quad \text{RMSPROP, momementum for } g^2 \tag{30}$$

$$\hat{m}_i^t = \frac{m_i^t}{1 - \beta_1^t} \quad \hat{v}_i^t = \frac{v_i^t}{1 - \beta_2^t} \quad \text{decay} \tag{31}$$

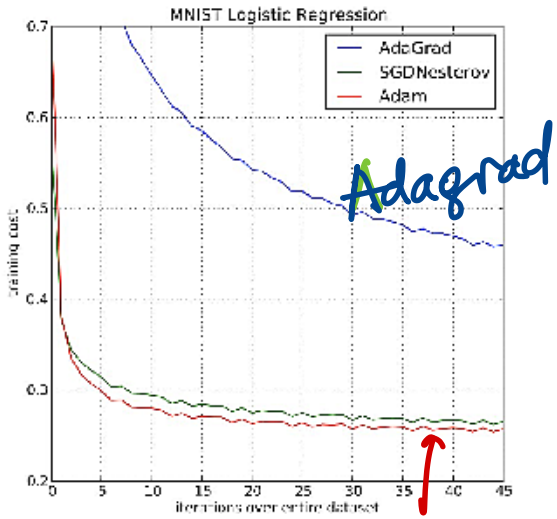
$$w_i^t \leftarrow w_i^{t-1} - \frac{\eta}{\sqrt{\hat{v}_i^t + \epsilon}} \hat{m}_i^t \tag{32}$$

Gist

Comparisons – AdaGrad, RMSProp, ADAM

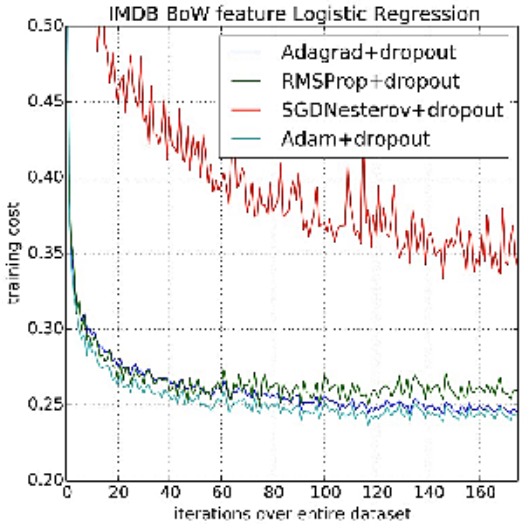
Empirical Comparison

- From Kingma & Ba (ICLR-2015):



Adagrad

Adam



Adagrad
Adam

UNIVERSITY OF WATERLOO

Lecture Notes V – Residual and Convolutional Neural Networks

Marina Meilă
mmp@uwaterloo.ca

With Thanks to Pascal Poupart & Gautam Kamath
Cheriton School of Computer Science
University of Waterloo

February 8, 2026

Convolutional networks (Convnets)

Residual networks (Resnets)

Some (convolutional) neural network breakthroughs

Reading HTF Ch.: 11.3 Neural networks, Murphy Ch.: (16.5 neural nets), Bach Ch.: –, Deep Learning Book (Goodfellow, Bengio, Courville) 6.1-4, ResNet 7.6, ConvNet 9., Autoencoders 14.1, Dive Into Deep Learning 4.1-4.3.

ConvNets – Convolutional Networks

- **discrete convolution** let $f, g : \mathbb{Z} \to \mathbb{R}$
  $\mathbb{Z} =$ all integers

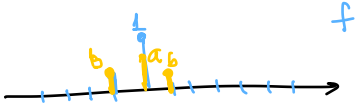
$$(f * g)(t) = \sum_{i \in \mathbb{Z}} f(t-i)g(i) \tag{1}$$

- convolution as **Toeplitz** matrix vector multiplication
- in ConvNets, $\mathbb{Z}$ is replaced by $1 : m$, f is **padded with 0's**
  - g is a (smoothing) kernel
  - i.e. $g(i) = g(-i) > 0$ and $|\operatorname{supp} g| = 2s + 1 \ll m$, $\sum_i g(i) = 1$
- Convolutional layer $f \leftarrow x$ input, $g \leftarrow w$ weights, s output

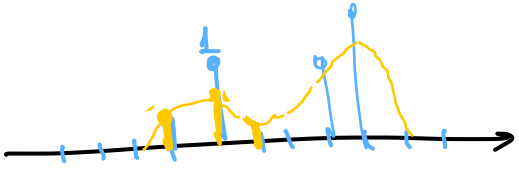
$$s(t) = \sum_{i=t-s}^{t+s} w_i s(t-i) \tag{2}$$

- Pooling
  a symmetric function like max, $\sum$, ...

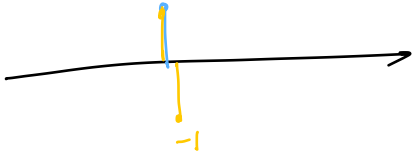
Ex1



f is δ-function



Ex2



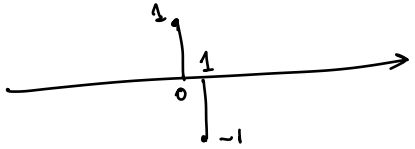
f = step

g = "kernel", "filter"

smoothing

$a + 2b = 1$

$$(f*g)(t) = \sum_{i=-\infty}^{\infty} g(i) f(t-i) = b f(t+1) + b f(t-1) + a f(t)$$



$$(f*g)(t) = f(t) - f(t-1)$$ discrete derivative

$$(f*g)(t) = \sum_{i=-\infty}^{\infty} g(i) f(t-i)$$

$$= f(t) \cdot 1 + f(t-1)(-1) + 0$$

$i=0$ $\quad$ $j=1$

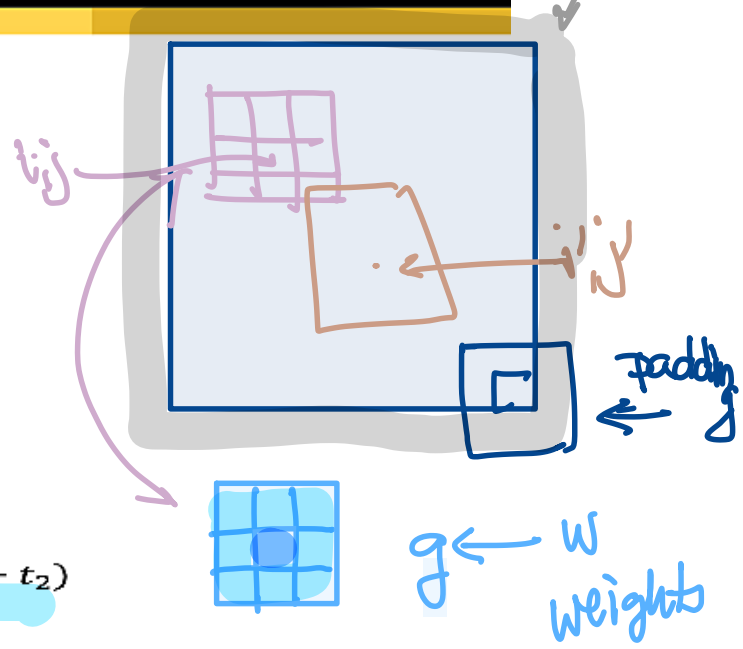
Discrete convolution

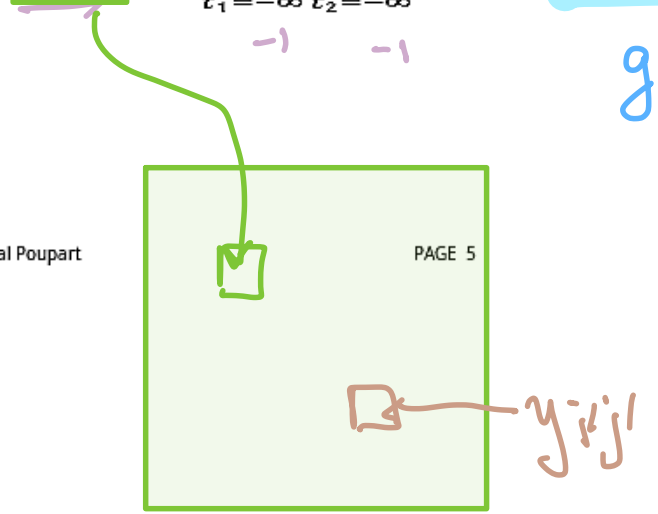
-
- Discrete convolution

$$y(i) = \sum_{t=-\infty}^{\infty} x(t)w(i-t)$$

- Multidimensional convolution

$$y(i,j) = \sum_{t_1=-\infty}^{\infty} \sum_{t_2=-\infty}^{\infty} x(t_1,t_2)w(i-t_1, j-t_2)$$





Example: Edge Detection

Gabor filters

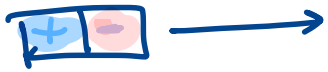
- Consider a grey scale image
- Detect vertical edges: $y(i,j) = x(i,j) - x(i-1,j)$

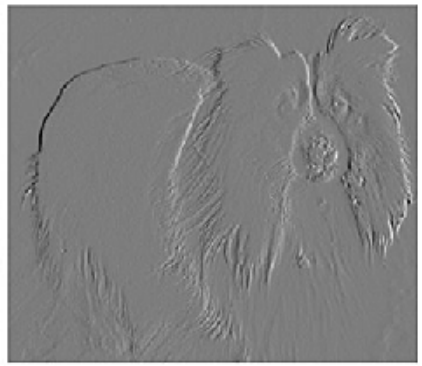
g moved to ij

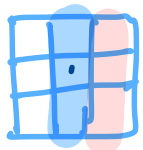
$$w(i - t_1, j - t_2) = \begin{cases} 1 & t_1 = i, t_2 = j \\ -1 & t_1 = i - 1, t_2 = j \\ 0 & \text{otherwise} \end{cases}$$

=

vertical edge

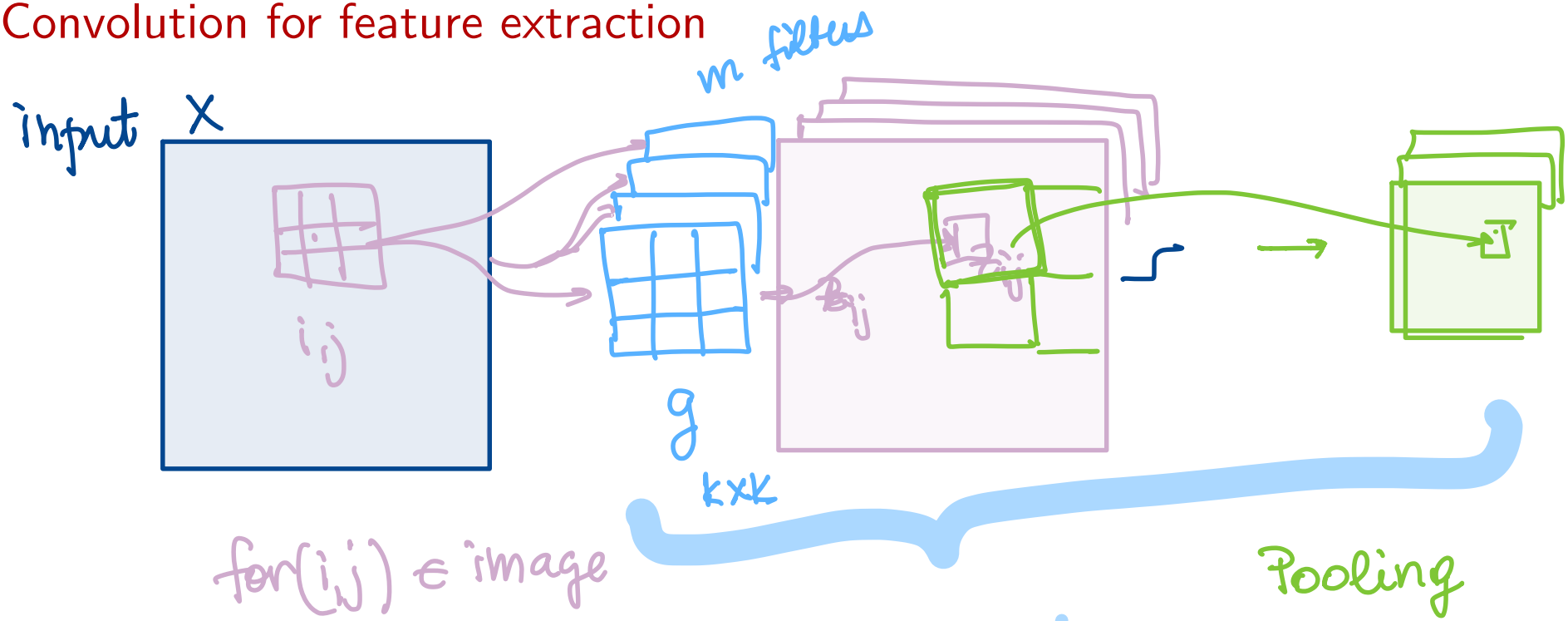






vertical edge with square kernel

Convolution for feature extraction



$$y(\text{patch}) = \max\left(\overset{z}{\text{patch}}\right)$$

sum
prod
...

symmetric function

Conv layer

Pooling

- Pooling: **commutative** mathematical operation that combines several units
- Examples:
  - max, sum, product, average, Euclidean norm, etc.

- Commutative property (order does not matter):

$$\text{Ex.: } \max(a, b) = \max(b, a)$$

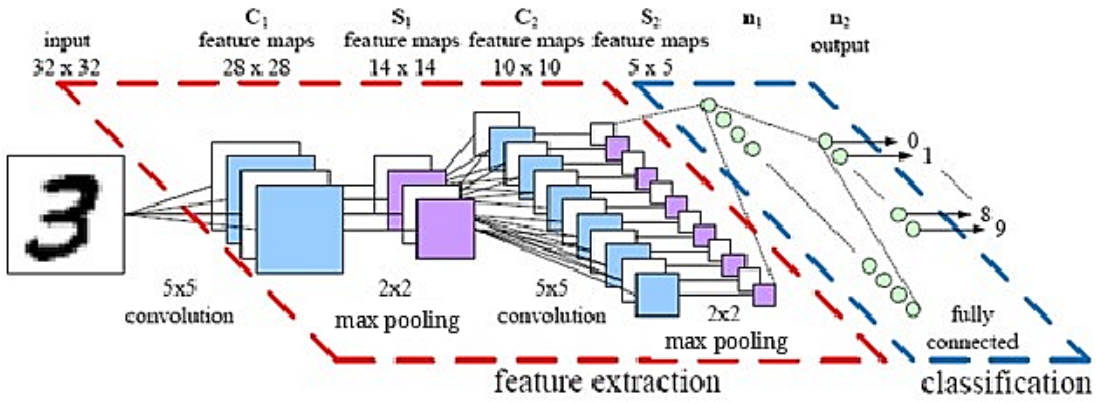
Also called symmetric when > 2 arguments

Convolution Neural Network

- A **convolutional neural network** refers to any network that includes an **alternation of convolution and pooling layers**, where **some of the convolution weights are shared.**
- Architecture:

LeNet ?

Example: Digit Recognition

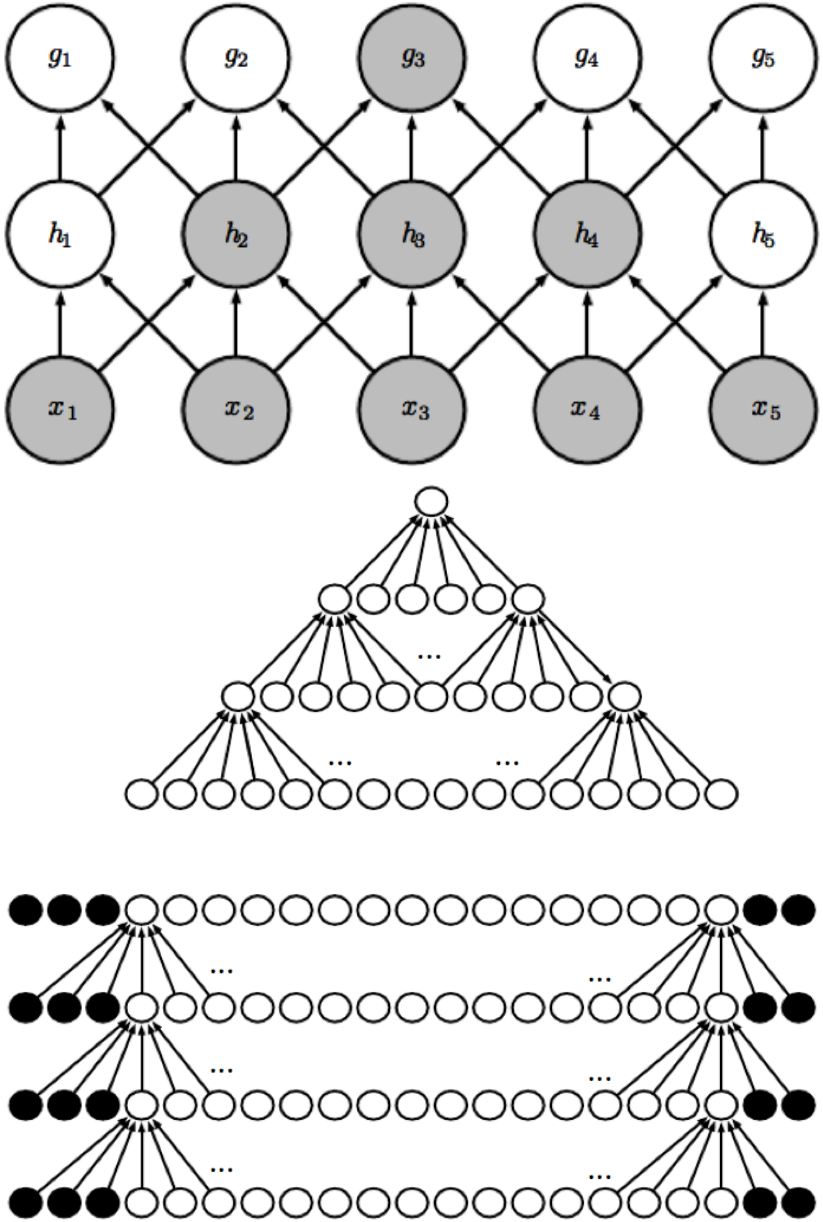


Parameters

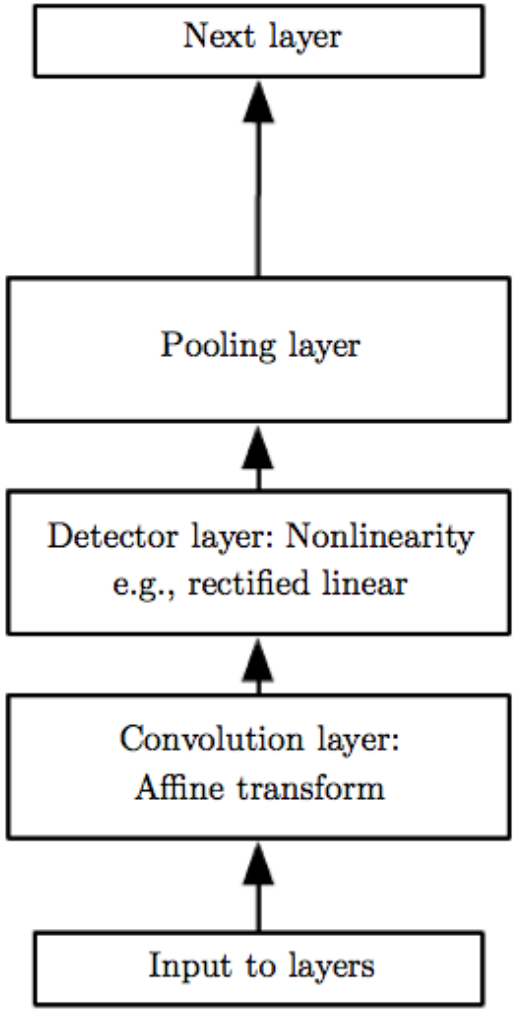
- **# of filters:** integer indicating the # of filters applied to each window.
- **kernel size:** tuple (width, height) indicating the size of the window.
- **Stride:** tuple (horizontal, vertical) indicating the horizontal and vertical shift between each window.
- **Padding:** "valid" or "same". Valid indicates no input padding. Same indicates that the input is padded with a border of zeros to ensure that the output has the same size as the input.

Benefits

- Sparse interactions
  - Fewer connections
- Parameter sharing ← every X ⇒ many patches
  - Fewer weights
- Locally equivariant representation
  - Locally invariant to translations
  - Handle inputs of varying length



from www.deeplearningbook.org Chapter 9



Training

- Convolutional neural networks are trained in the same way as other neural networks
  - E.g., backpropagation
- Weight sharing:
  - Combine gradients of shared weights into a single gradient

Architecture design

- What is the preferred filter size?
- VGG (Visual Geometry Group at Oxford, 2014): stack of small filters is often preferred to a single large filter
  - Fewer parameters
  - Deeper network
- Picture

Resnets – Residual networks

Idea What is the "simplest" input-output function? $f_0(x) = x$

- Hence, a NN layer should learn the difference w.r.t. identity f_0

$$x_{l+1} = B_l \phi(W_l x_l) + x_l \tag{3}$$

Generalization DenseNet

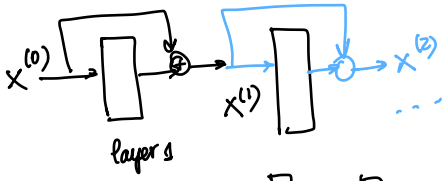
- Layer l gets inputs from $l-1, l-2, \ldots$

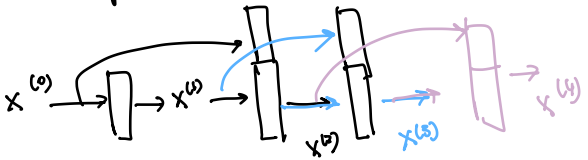
Residual Networks

- Problem: very deep networks can perform worse than shallower networks (due to local optima & other stationary non-optimal points)
- Solution [He et al., 2015]: introduce **residual connections** (a.k.a. skip connections) to make blocks optional
- Picture:

Idea $f(x) = x$ identity

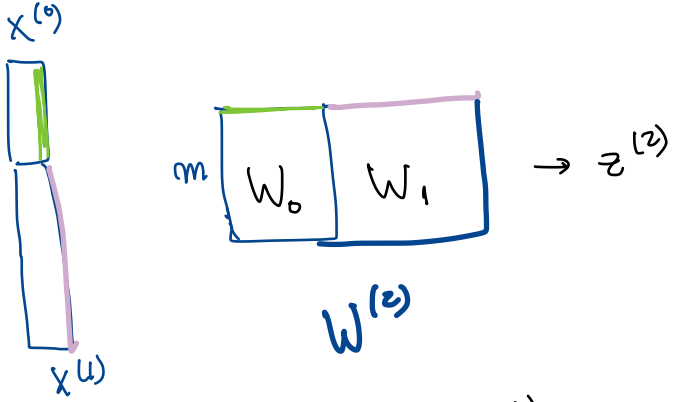
1) $y(x) = x + f(x)$ ← nn

2) 

3) 

layer ℓ : input $x^{(\ell-1)}$, $x^{(\ell-2)}$

4) layer ℓ : input $x^{(\ell-1, \ell-2, \cdots)}$



$$z_i^{(2)} = W_{0i} x^{(0)} + W_{1i} x^{(1)}$$