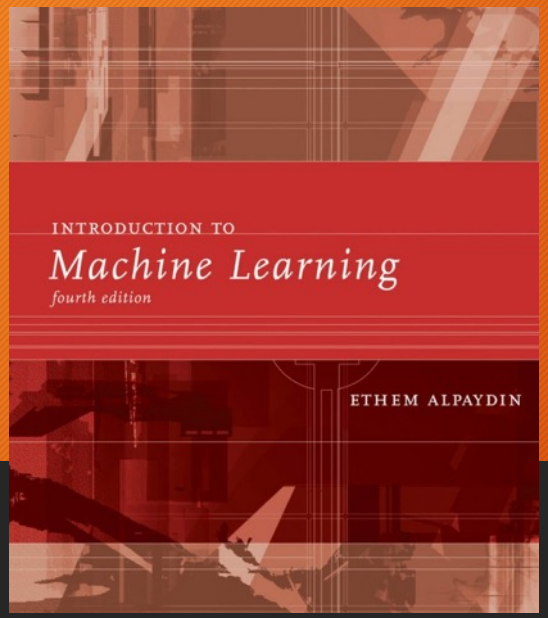


Lecture Slides for

INTRODUCTION TO Machine Learning 4e

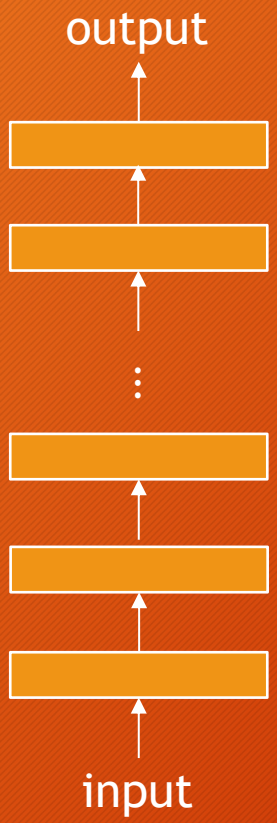
ETHEM ALPAYDIN
The MIT Press, 2020

ethem.alpaydin@gmail.com

CHAPTER 12:

Deep Learning

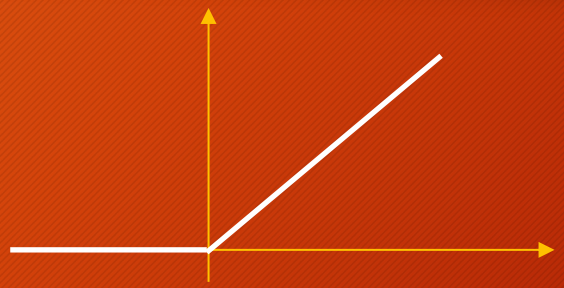
Deep Neural Networks



- Many hidden layers
- End-to-end training
- Learn increasingly abstract representations with minimal human contribution
- Basis functions calculated from simpler basis functions

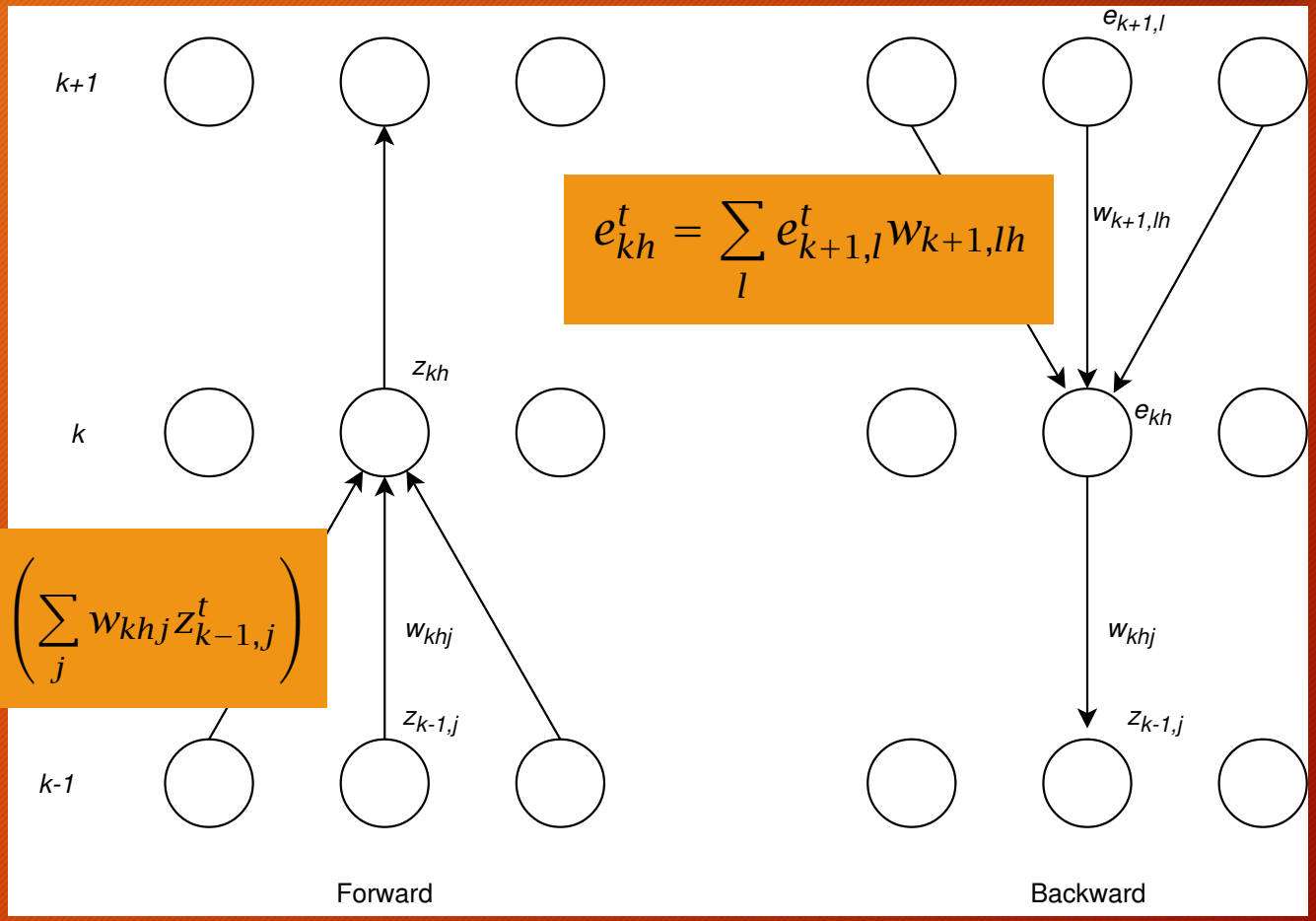
Rectified Linear Unit (ReLU)

$$\mathrm{ReLU}(a) = \begin{cases} a & \text{if } a > 0 \\ 0 & \text{otherwise} \end{cases}$$



- Does not saturate for large *a*
- Leads to a sparse representation
- No learning for *a*<0, be careful with initialization
- Leaky ReLU

Generalizing Backpropagation



$$z_{kh}^t = f\left(\sum_j w_{khj} z_{k-1,j}^t\right)$$

$$e_{kh}^t = \sum_l e_{k+1,l}^t w_{k+1,lh}$$

Improving Training Convergence

- Momentum. At each update, also add a fraction of the average of past gradients:

$$
\begin{aligned}
s_i^t &= \alpha s_i^{t-1} + (1-\alpha)\frac{\partial E^t}{\partial w_i} \\
\Delta w_i^t &= -\eta s_i^t
\end{aligned}
$$

Adaptive Learning Factor

- RMSprop. Make update inversely proportional to the sum of past gradients, so, update more when gradient is small and less where it is large.

$$\Delta w_i^t = -\frac{\eta}{\sqrt{r_i^t}} \frac{\partial E^t}{\partial w_i}$$

where r_i is the accumulated past gradient,

$$r_i^t = \rho r_i^{t-1} + (1-\rho) \left| \frac{\partial E^t}{\partial w_i} \right|^2$$

ADAM: Adaptive Learning Factor w/ Momentum

$$s_i^t = \alpha s_i^{t-1} + (1-\alpha)\frac{\partial E^t}{\partial w_i}$$

$$r_i^t = \rho r_i^{t-1} + (1-\rho)\left|\frac{\partial E^t}{\partial w_i}\right|^2$$

$$\Delta w_i^t = -\eta \frac{\tilde{s}_i^t}{\sqrt{\tilde{r}_i^t}}$$

Initially both *s* and *r* terms are 0, so we bias-correct them (both α and ρ are <1, so they get smaller as *t* gets large):

$$\tilde{s}_i^t = \frac{s_i^t}{1-\alpha^t} \text{ and } \tilde{r}_i^t = \frac{r_i^t}{1-\rho^t}$$

Batch Normalization

- Z-normalize hidden unit values (m and s are mean and stdev over the current minibatch):

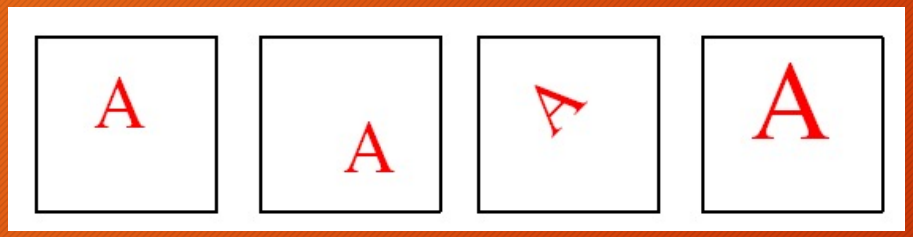
$$\tilde{a}_j = \frac{a_j - m_j}{s_j}$$

- Can then rescale (γ and β are also learned):

$$\hat{a}_j = \gamma_j \tilde{a}_j + \beta_j$$

Regularization: Hints

- Invariance to translation, rotation, size



(Abu-Mostafa, 1995)

- Virtual examples
- Augmented error: $E' = E + \lambda_h E_h$

  If $\boldsymbol{x}'$ and $\boldsymbol{x}$ should be the "same":

  $$E_h = [g(x|\theta) - g(x'|\theta)]^2$$

Regularization: Weight Decay

- If a weight is 0, we have a simpler model.
- Initially all weights are close to 0 and they are moved away as learning proceeds.
- Early stopping: Stop before too many weights are updated.
- Weight decay: Add a term that penalizes non-zero weights:

$$E' = E + \frac{\lambda}{2}\sum_i w_i^2$$

$$\Delta w_i = -\eta\frac{\partial E}{\partial w_i} - \lambda' w_i$$

Regularization: Bayesian Perspective

- Consider weights w_i as random vars, prior $p(w_i)$

$$p(\mathbf{w}\mid\mathcal{X})=\frac{p(\mathcal{X}\mid\mathbf{w})p(\mathbf{w})}{p(\mathcal{X})} \qquad \hat{\mathbf{w}}_{MAP}=\arg\max_{\mathbf{w}} \log p(\mathbf{w}\mid\mathcal{X})$$

$$\log p(\mathbf{w}\mid\mathcal{X})=\log p(\mathcal{X}\mid\mathbf{w})+\log p(\mathbf{w})+C$$

$$p(\mathbf{w})=\prod_i p(w_i) \text{ where } p(w_i)=c\cdot\exp\left[-\frac{w_i^2}{2(1/2\lambda)}\right]$$

$$E'=E+\lambda\|\mathbf{w}\|^2$$

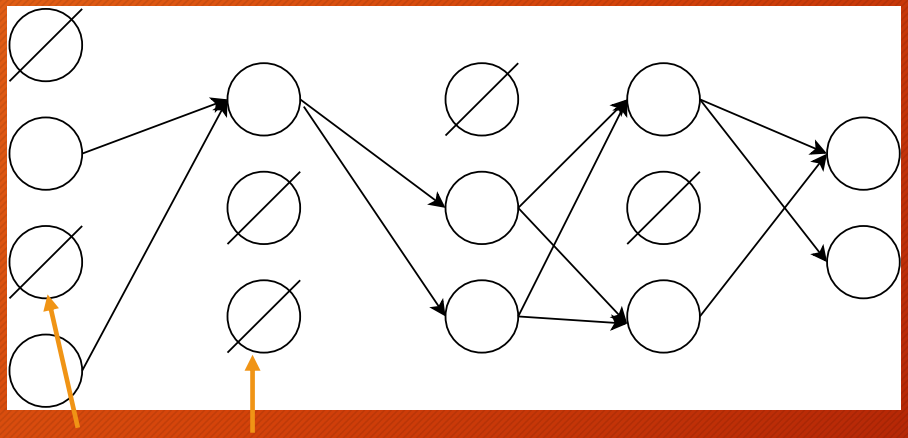
- Weight decay, ridge regression, regularization

  cost=data-misfit + λ complexity

  More about Bayesian methods in chapter 15

Regularization: Dropout

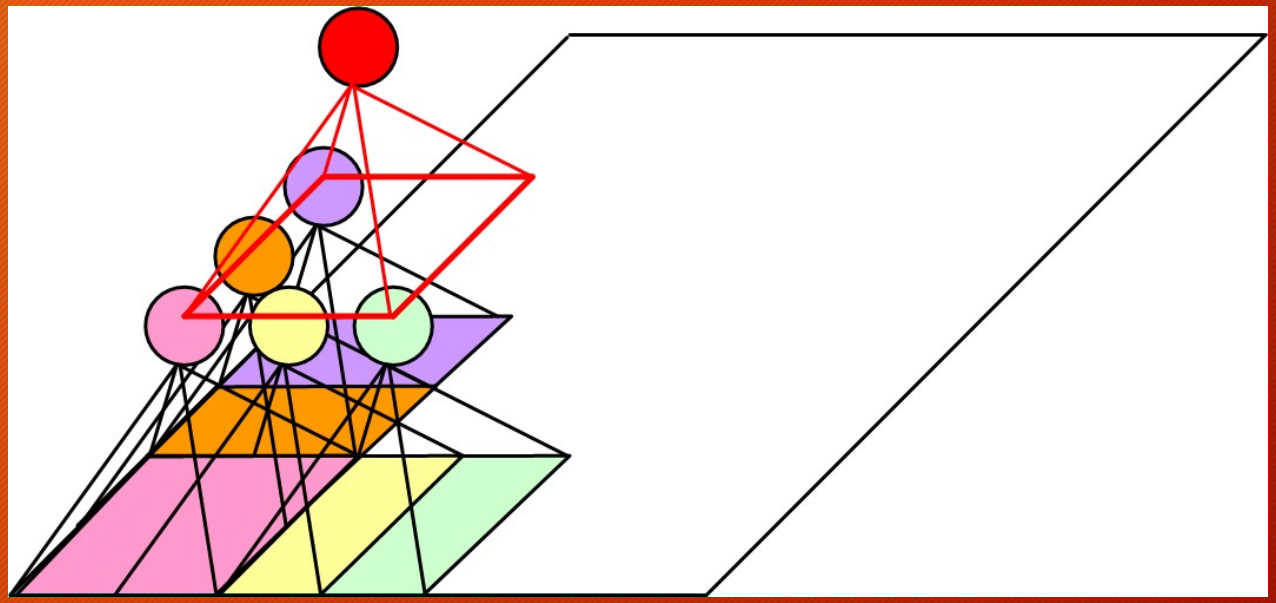
- Adding noise to inputs/hidden unit values/weights has a regularizing effect
- What if we drop them out completely?



Input or hidden unit dropped out (output set to 0) with probability *p* in each minibatch

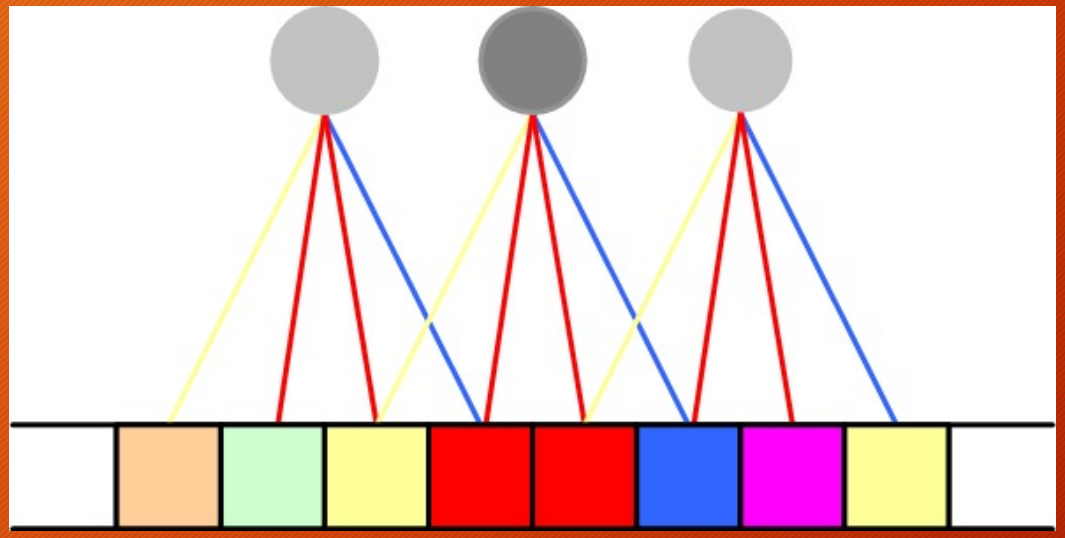
Regularization: Convolutions

- Each unit is connected to a small set of units in the preceding layer. In images, this corresponds to a local patch (LeCun et al 1989).



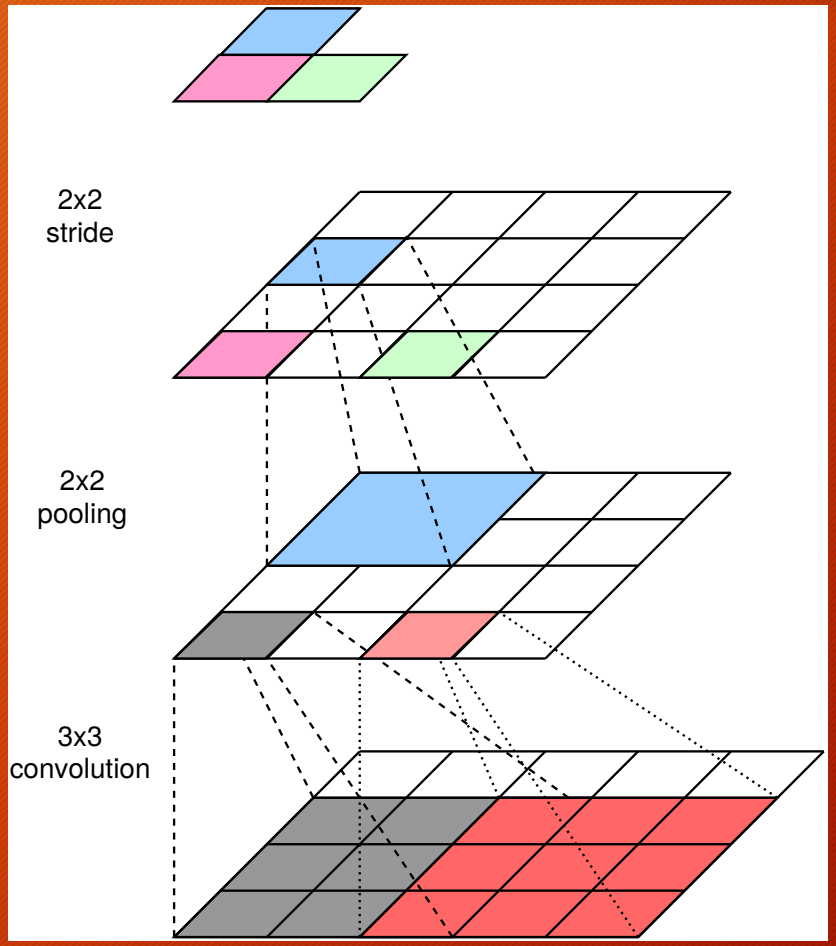
Regularization: Weight Sharing

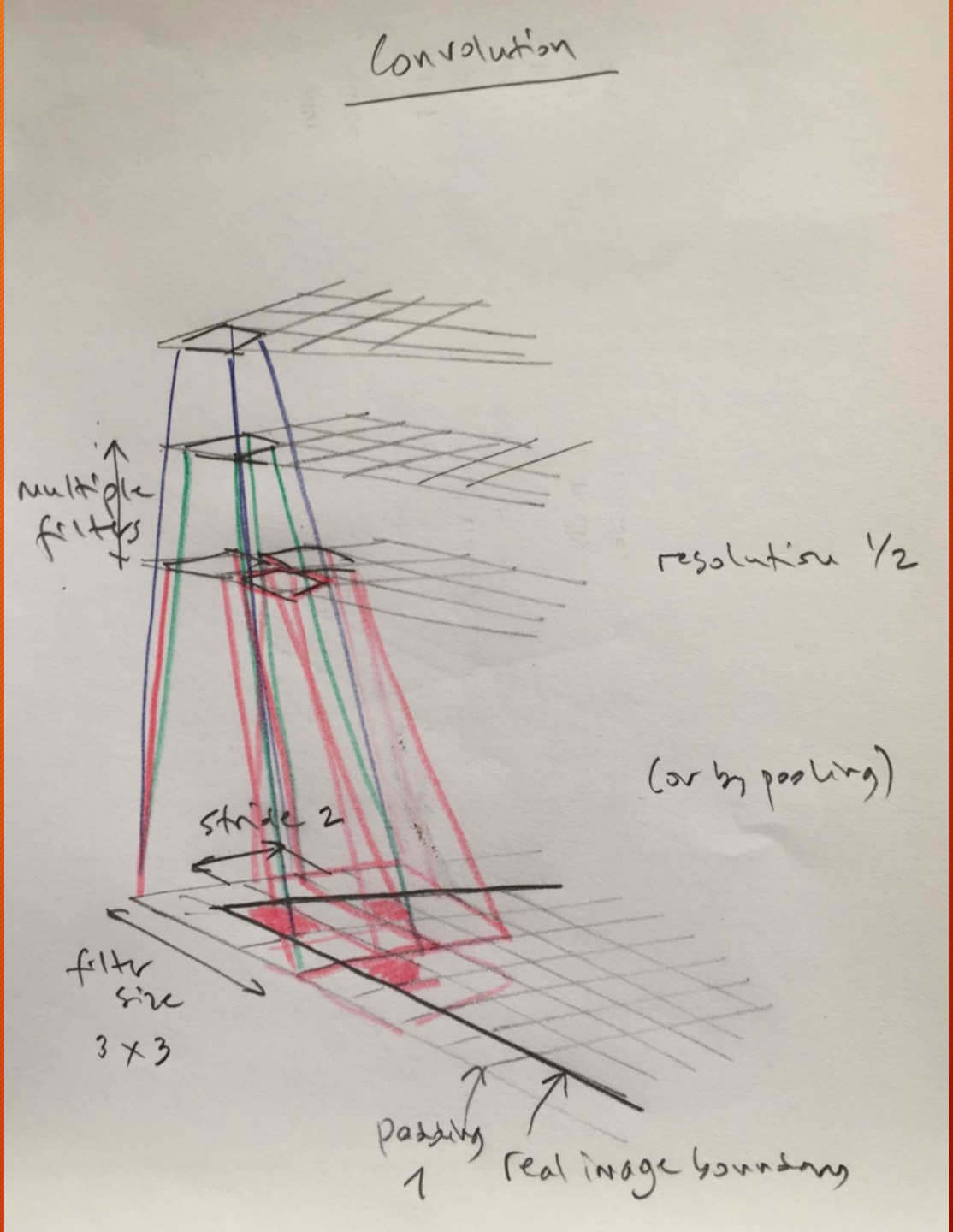
The same weights are used in different locations



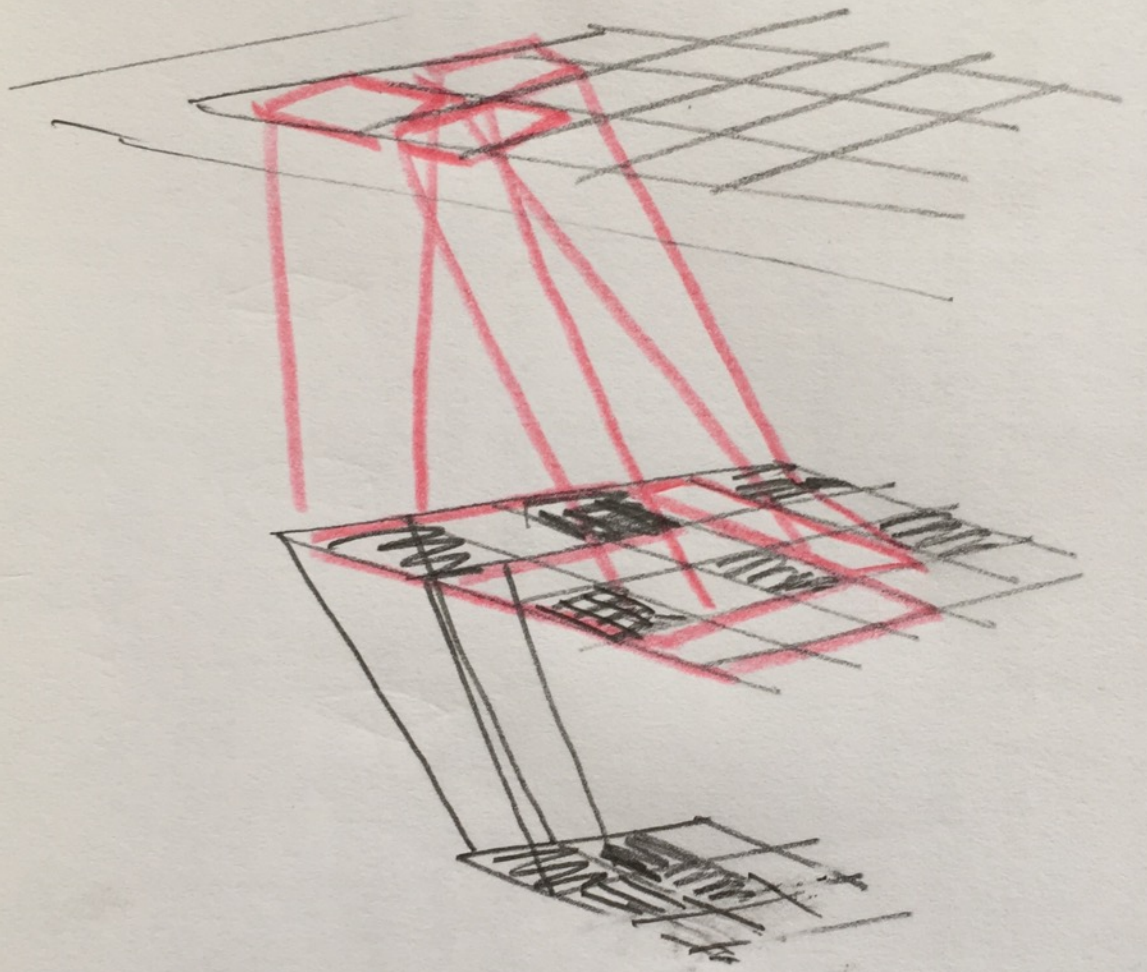
1d example with 1x4 convolutions and weight sharing

Multiple Convolutional Layers



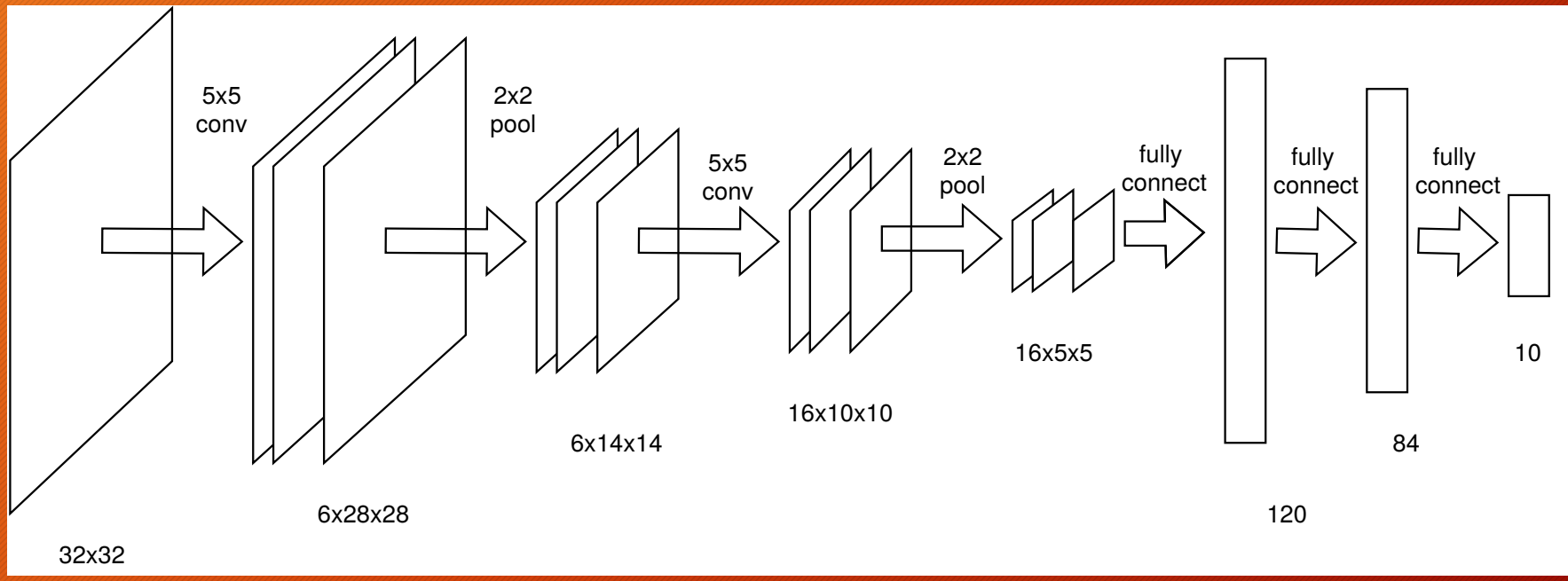

Convolution
multiple filters
resolution 1/2
(or by pooling)
stride 2
filter size
3 x 3
padding
1
real image boundary

Deconvolution



Resolution 2

LeNet-5 (LeCun et al 1998)



Has approximately 60K weights and trained on 60K examples (MNIST data set)

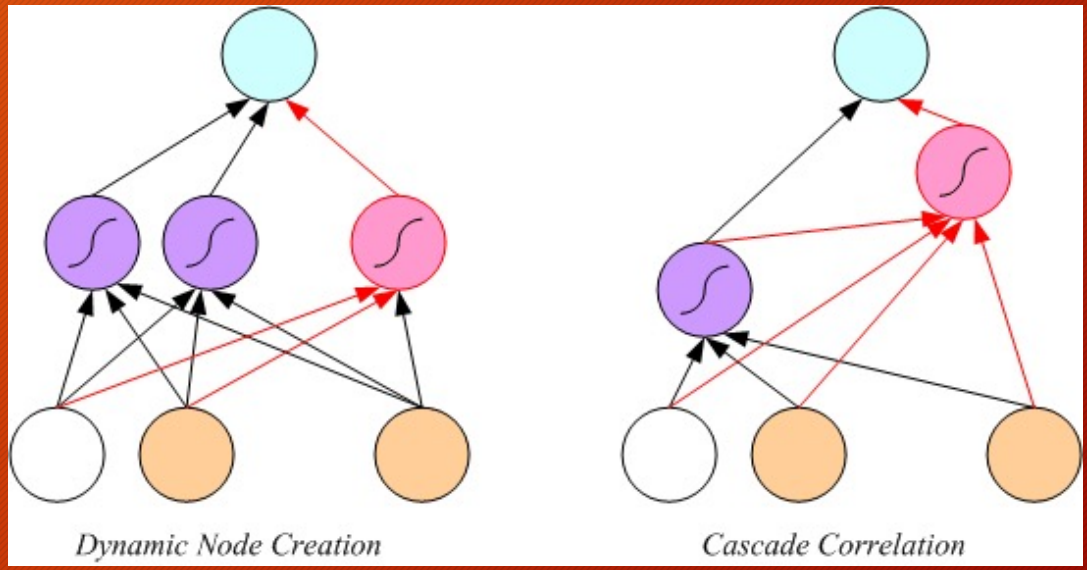
Tuning the Network Size

- Destructive
  Weight decay:

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} - \lambda w_i$$

$$E' = E + \frac{\lambda}{2} \sum_i w_i^2$$

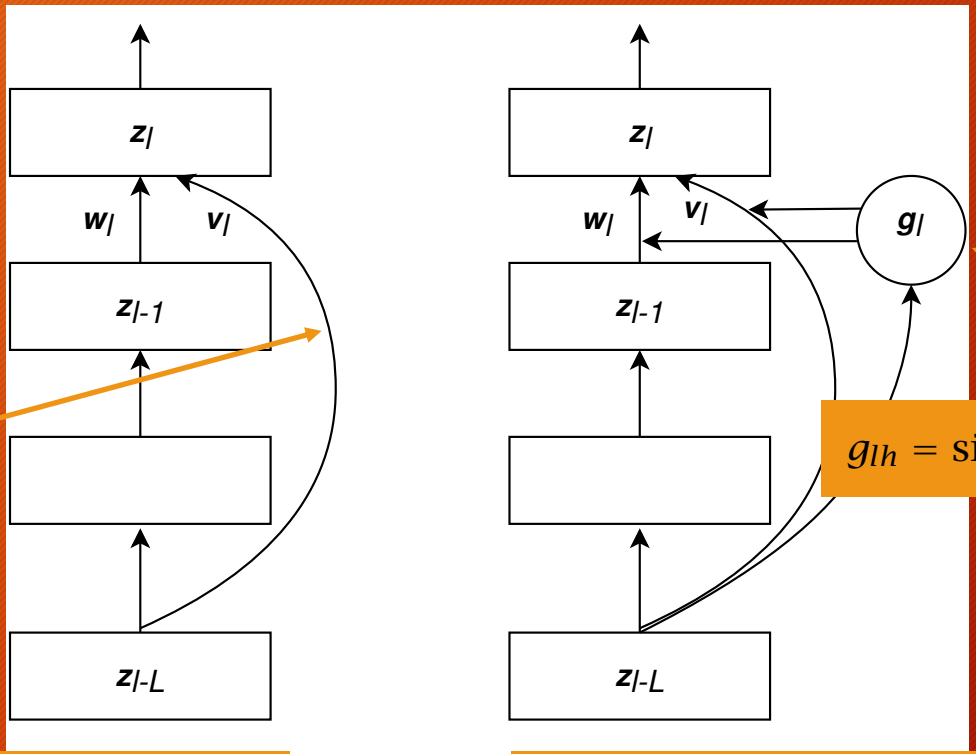
- Constructive
  Growing networks



(Ash, 1989)

(Fahlman and Lebiere, 1989)

Skip Connections



Skip connection

Gating unit

$$g_{lh} = \text{sigmoid}\left(\boldsymbol{a}_{lh}^T \boldsymbol{z}_{l-L} + a_{lh0}\right)$$

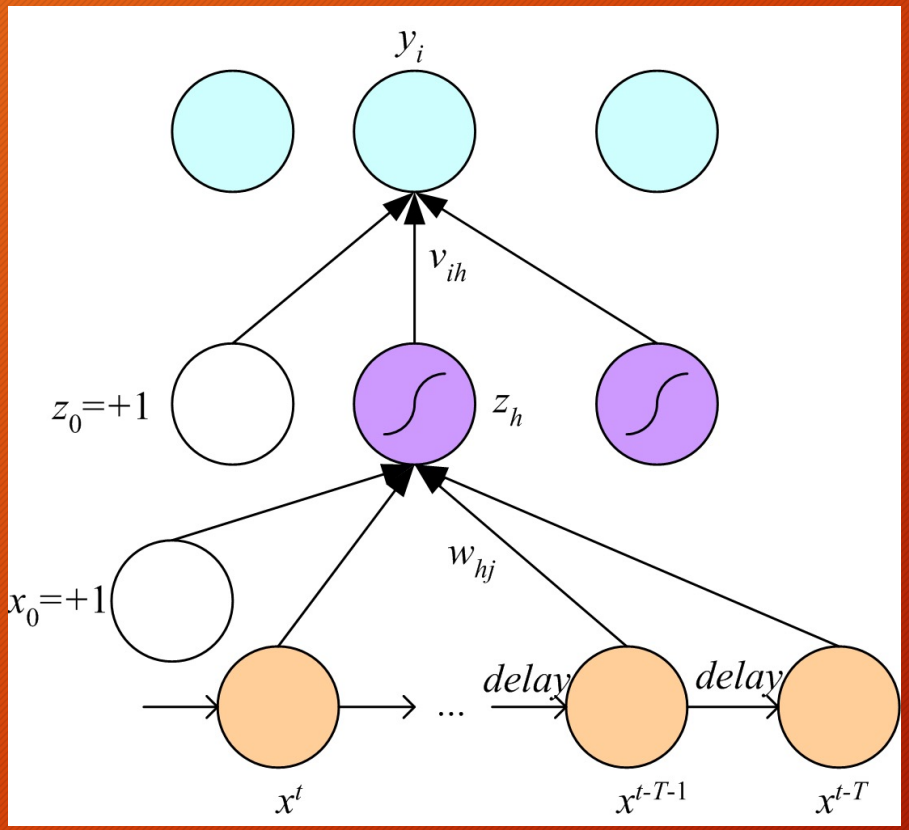
$$z_{lh} = f\left(\sum_i w_{l,h,i} z_{l-1,i} + \sum_j v_{l,h,j} z_{l-L,j}\right)$$

$$z_{lh} = f\left(g_{lh} \sum_i w_{lhi} z_{l-1,i} + (1 - g_{lh}) \sum_j v_{lhj} z_{l-L,j}\right)$$

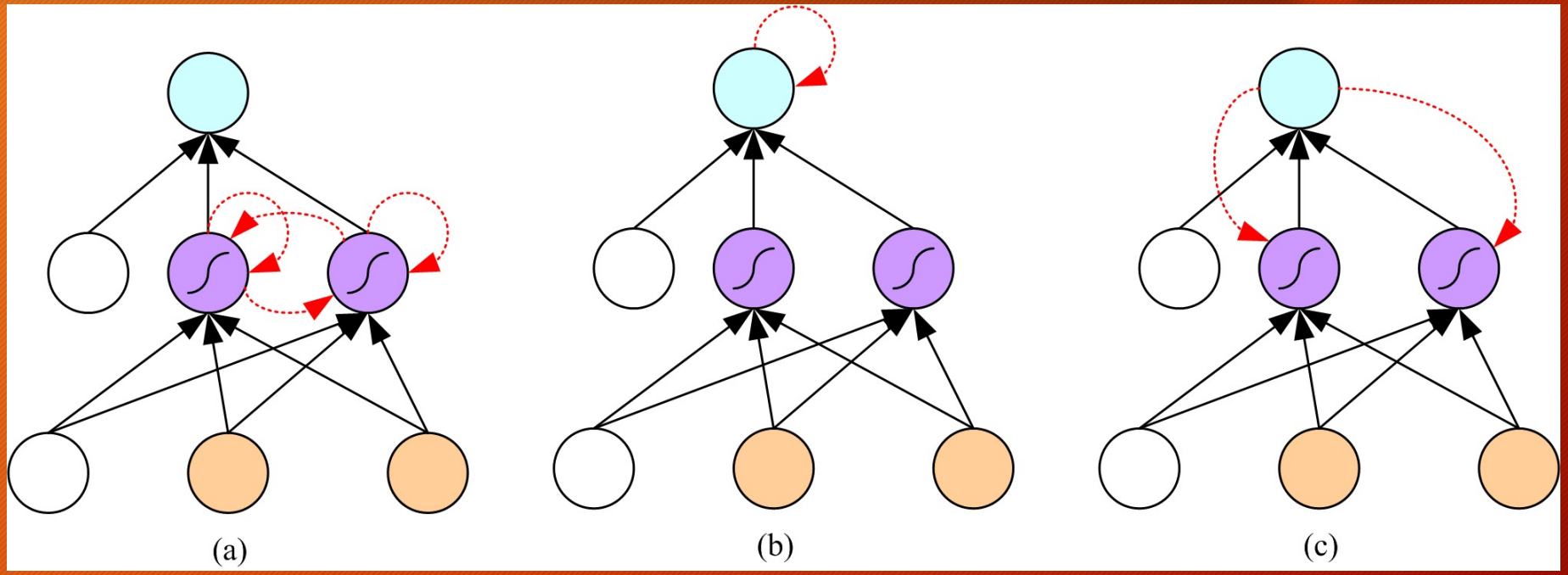
Learning Time

- Applications:
  - Sequence recognition: Speech recognition
  - Sequence reproduction: Time-series prediction
  - Sequence association
- Network architectures
  - Time-delay networks (Waibel et al., 1989)
  - Recurrent networks (Rumelhart et al., 1986)

Time-Delay Neural Networks



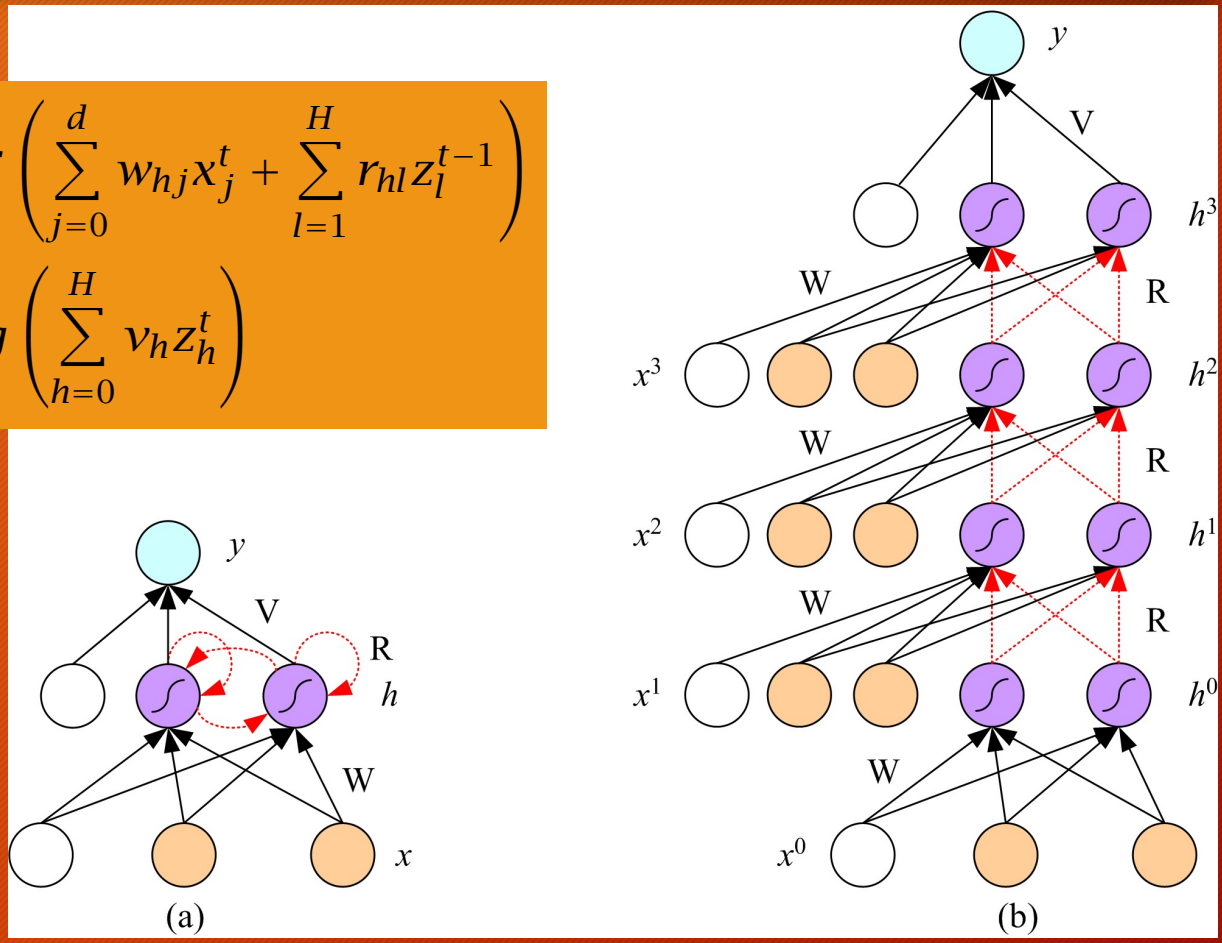
Recurrent Networks



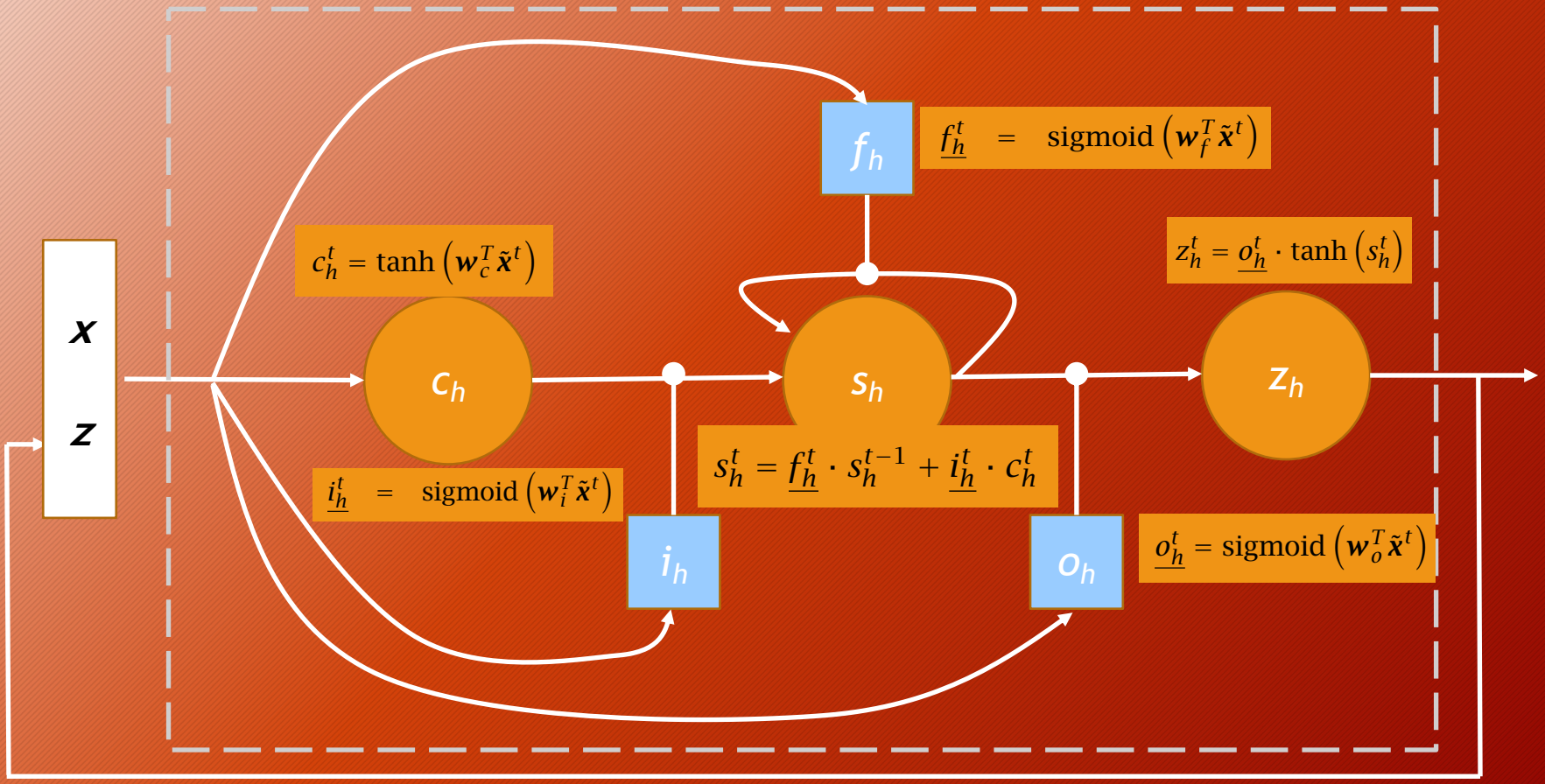
Unfolding in Time

$$z_h^t = f\left(\sum_{j=0}^{d} w_{hj} x_j^t + \sum_{l=1}^{H} r_{hl} z_l^{t-1}\right)$$

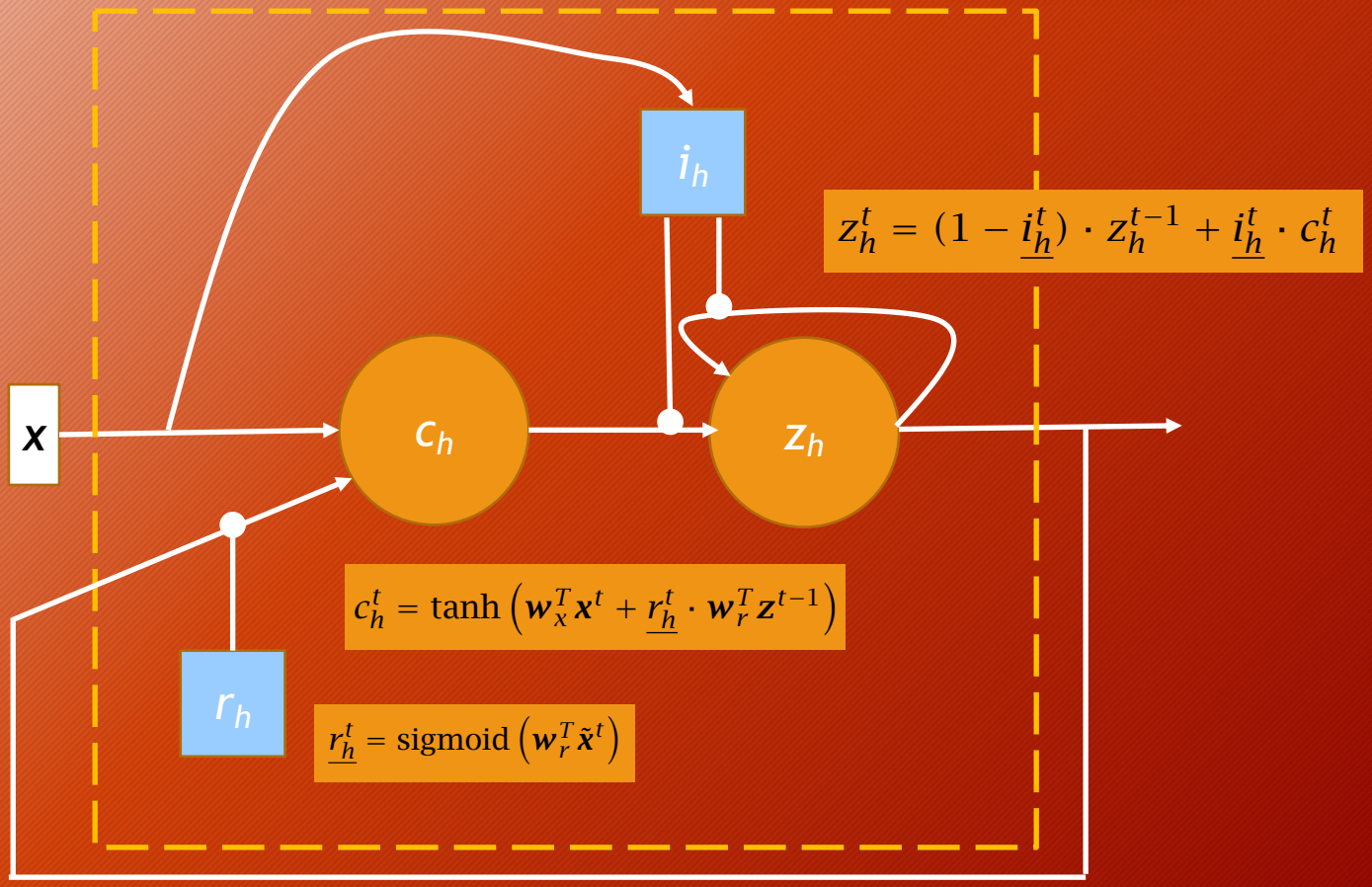
$$y^t = g\left(\sum_{h=0}^{H} v_h z_h^t\right)$$



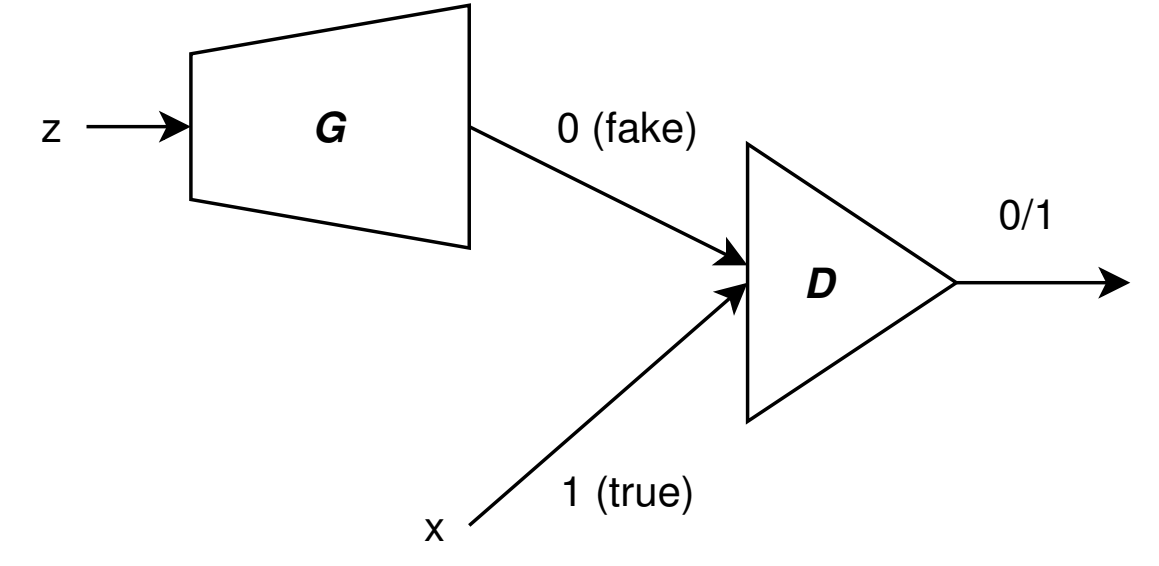
Long Short-Term Memory (LSTM)



Gated Recurrent Unit (GRU)



Generative Adversarial Networks



$$\sum_t \log D(\boldsymbol{x}^t) + \sum_{\boldsymbol{Z} \sim p(\boldsymbol{Z})} \log\left(1 - D(G(\boldsymbol{z}))\right)$$

Likelihood-based

$$\sum_t E[D(\boldsymbol{x}^t)] - \sum_{\boldsymbol{Z} \sim p(\boldsymbol{Z})} E[D(G(\boldsymbol{z}))]$$

Wasserstein loss