

CNNs, RNNs, and interpretability

Recitation 3

MIT - 6.802 / 6.874 / 20.390 / 20.490 / HST.506 - Spring 2019

Sachit Saksena

Structured neural networks: structured data

Example 1 (unstructured): $\mathcal{X} = \mathbb{R}^n$, gene expression values for n genes, order of features (genes) meaningless

Structured neural networks : structured data

Example 1 (unstructured): $\mathcal{X} = \mathbb{R}^n$, gene expression values for n genes, order of features (genes) meaningless

Example 2 (1D structure): $\mathcal{X} = \mathbb{R}^{n \times t}$, gene expression values for n genes on t time points

Example 3 (1D structure): $\mathcal{X} = \{0, 1\}^{m \times n}$, biological sequences (DNA, RNA, protein sequence) of length m and alphabet size n , e.g., $n = 4$ (A, C, G, T)

Example 4 (1D structure): Natural language

Structured neural networks : structured data

Example 1 (unstructured): $\mathcal{X} = \mathbb{R}^n$, gene expression values for n genes, order of features (genes) meaningless

Example 2 (1D structure): $\mathcal{X} = \mathbb{R}^{n \times t}$, gene expression values for n genes on t time points

Example 3 (1D structure): $\mathcal{X} = \{0, 1\}^{m \times n}$, biological sequences (DNA, RNA, protein sequence) of length m and alphabet size n , e.g., $n = 4$ (A, C, G, T)

Example 4 (1D structure): Natural language

Example 5 (2D structure): $\mathcal{X} = [0, 1]^{28 \times 28}$, MNIST images

Example 6 (3D structure): $\mathcal{X} = [0, 1]^{m \times n \times p}$, voxels (volumetric pixels) from imaging methods such as CT and MRI

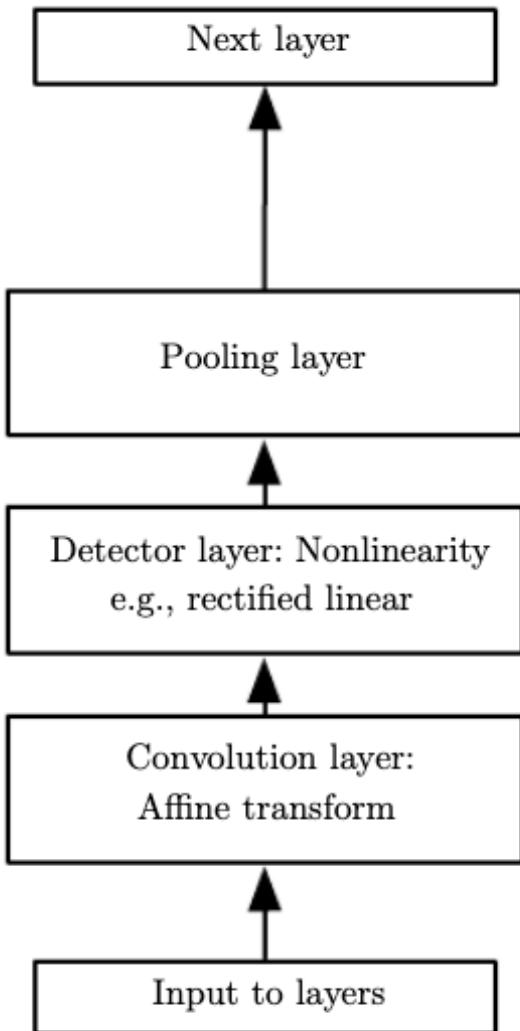
Structured neural networks : structured data

Spatial locality: The set of pixels we will have to take into consideration to find a particular object will be near one another in the image.

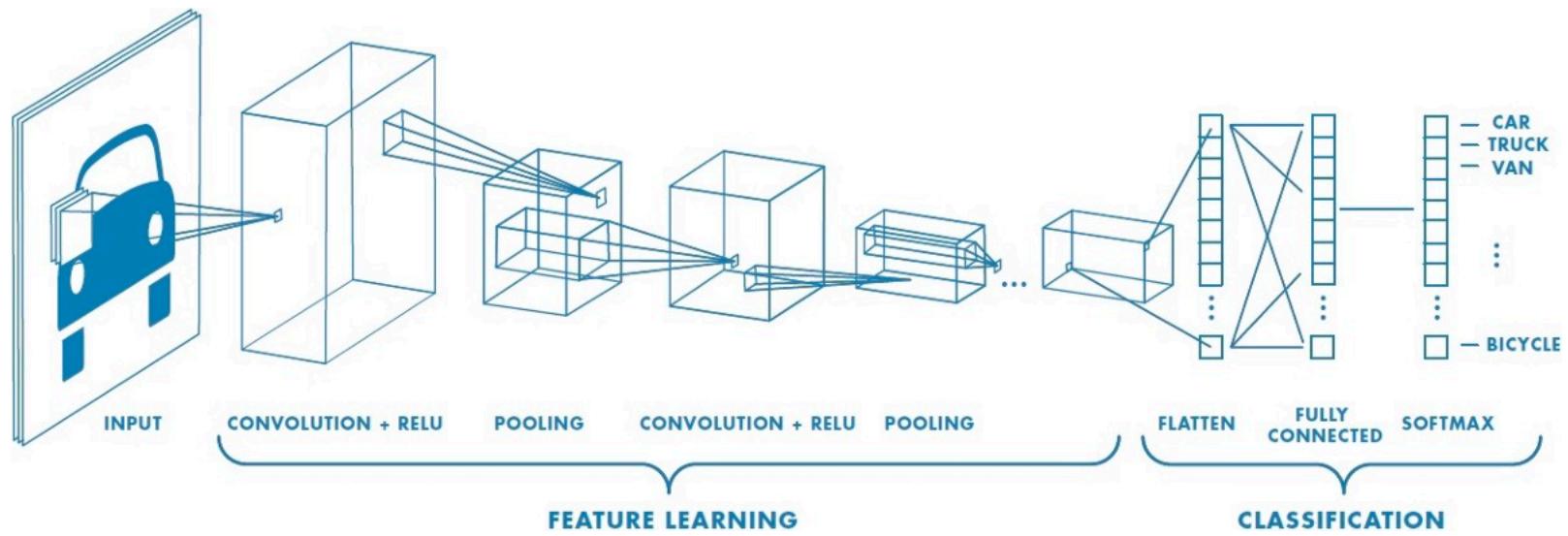
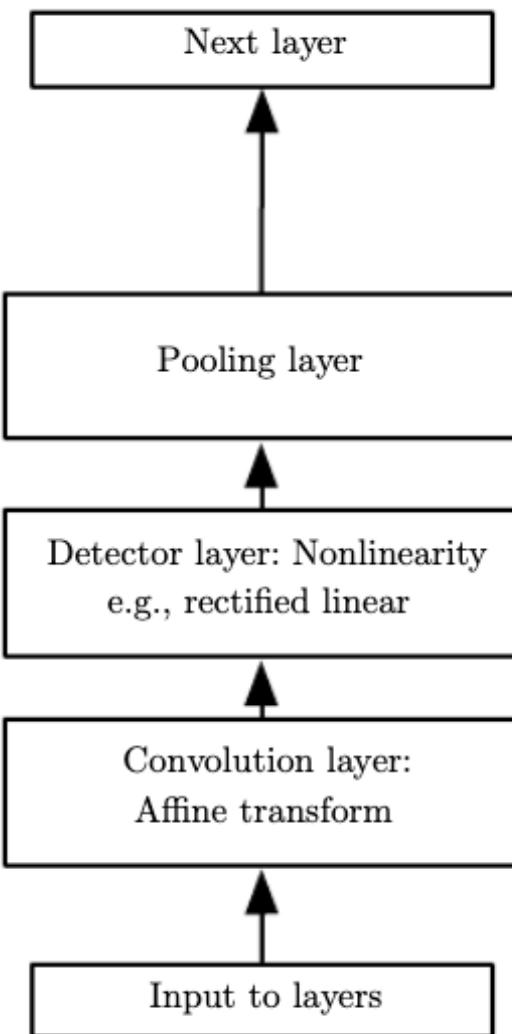
Translational invariance: A particular object is identifiable regardless of position in an image.

Hierarchical features: High-level features are composed of low-level features.

Convolutional neural networks: overview



Convolutional neural networks: overview



Convolutional neural networks: convolution operation

Imagine a position x taken at a time point t , which are both continuous,

$$x(t)$$

Suppose this measurement is noisy. To obtain a less noisy estimate of the position, we want a weighted average of recent measurements. Using a weighting function $w(a)$, where a is the age of the measurement,

$$s(t) = \int x(a)w(t-a)da = (x * w)(t)$$

input
convolutional kernel/filter

Generally, sensor inputs are not continuous and most signal processing tasks rely on discretized data—data provided at regular intervals. In machine learning, input is generally a multi-dimensional array of data and the kernel is a multi-dimensional array of parameters:

$$S(i, j) = (W * I)(i, j) = \sum_m \sum_n I(i + m, j + n)W(m, n)$$

This operation also exhibits translational equivariance. Let g be a function mapping one image function to another image function that shifts inputs to the right by one,

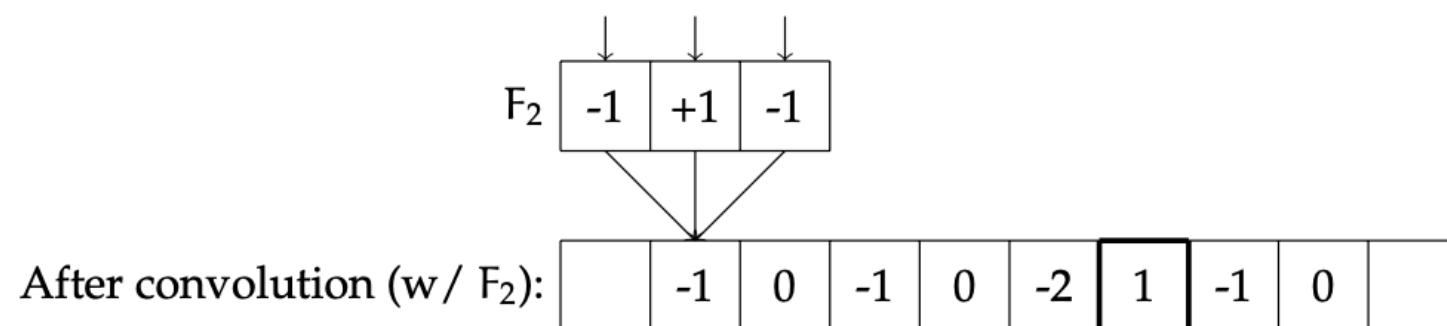
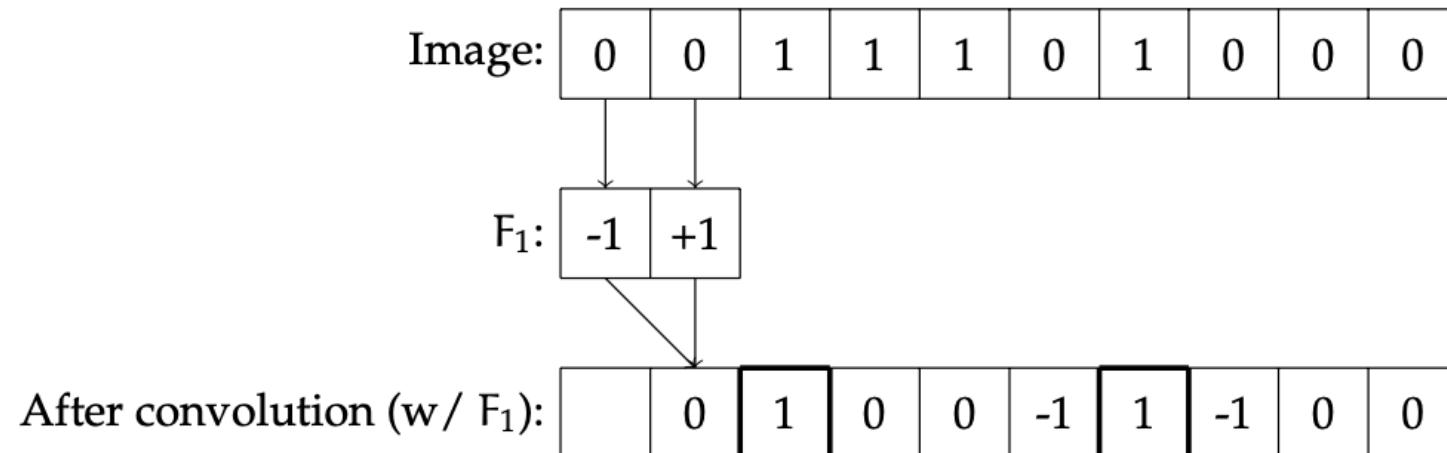
$$I'(i, j) = g(I(i, j)) = I(i - 1, j)$$

If we apply the transformation to I , then apply a convolution, the output is equivalent to applying a convolution to I' then applied the transformation g to the output,

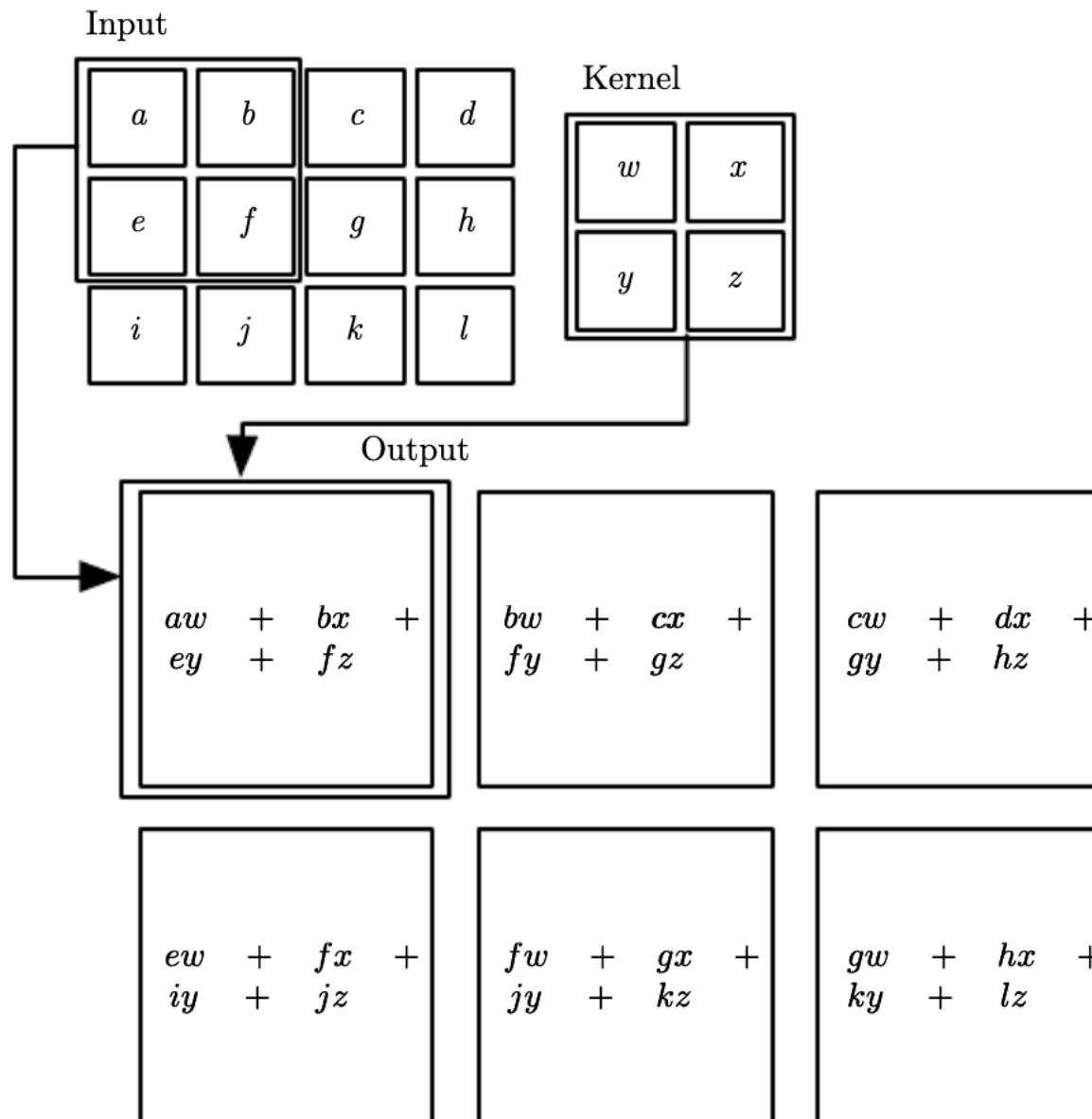
$$(W * I')(i, j) = g((W * I)(i, j))$$

So, a filter applied over an input pattern that is shifted in an image will result in the same convolutional output as the same translational transformation applied after the convolutional—translational equivariance.

Convolutional neural networks: 1D convolution



Convolutional neural networks: 2D convolution



Convolutional neural networks: 2D convolution

1	0	1
0	1	0
1	0	1

Filter / Kernel

1 <small>x1</small>	1 <small>x0</small>	1 <small>x1</small>	0	0
0 <small>x0</small>	1 <small>x1</small>	1 <small>x0</small>	1	0
0 <small>x1</small>	0 <small>x0</small>	1 <small>x1</small>	1	1
0	0	1	1	0
0	1	1	0	0

Image

Convolutional neural networks: 2D convolution

1	0	1
0	1	0
1	0	1

Filter / Kernel

1 <small>x1</small>	1 <small>x0</small>	1 <small>x1</small>	0	0
0 <small>x0</small>	1 <small>x1</small>	1 <small>x0</small>	1	0
0 <small>x1</small>	0 <small>x0</small>	1 <small>x1</small>	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved Feature

Convolutional neural networks: multi-channel inputs

0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...

Input Channel #3 (Blue)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1



308

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2



-498

0	1	1
0	1	0
1	-1	1

Kernel Channel #3



164

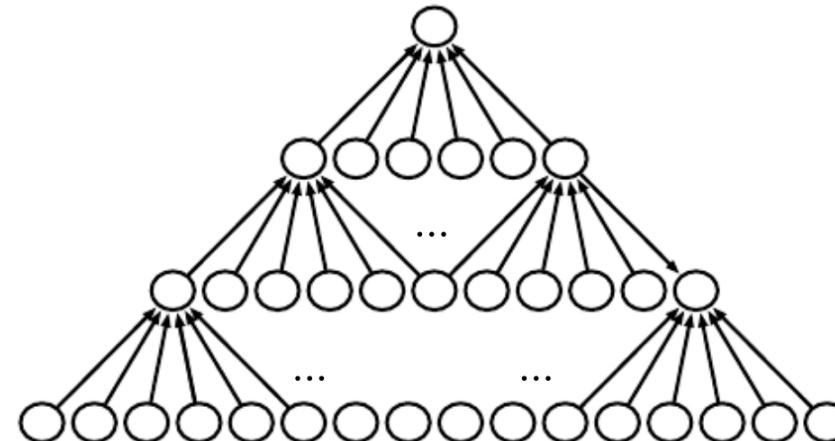
$$+ 1 = -25$$

$$\uparrow \text{Bias} = 1$$

-25				...
				...
				...
				...
...

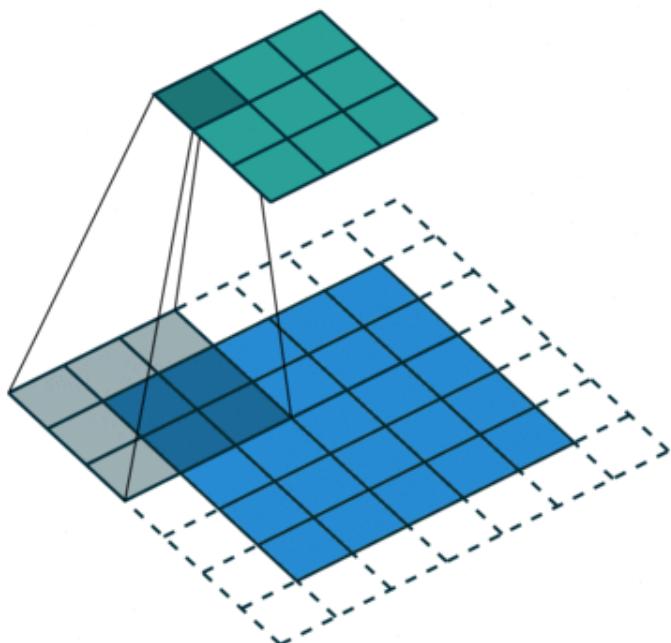
Convolutional neural networks: padding

Why zero-pad?

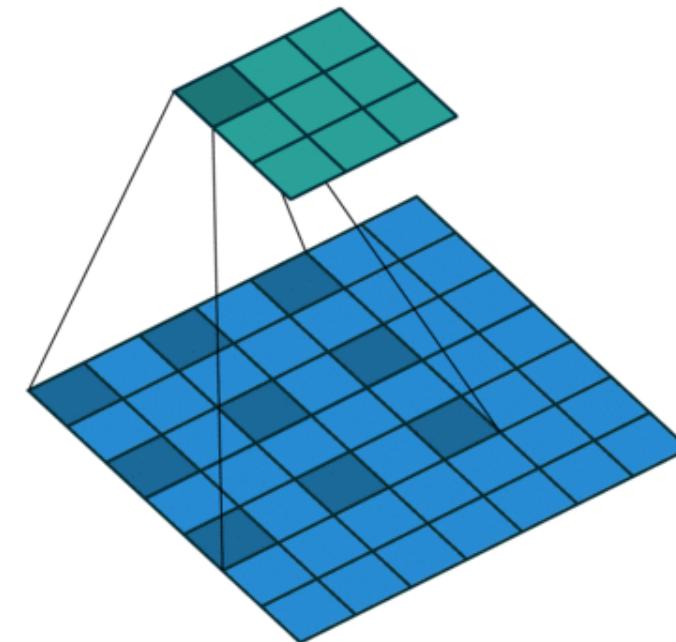


Convolutional neural networks: 2D convolution

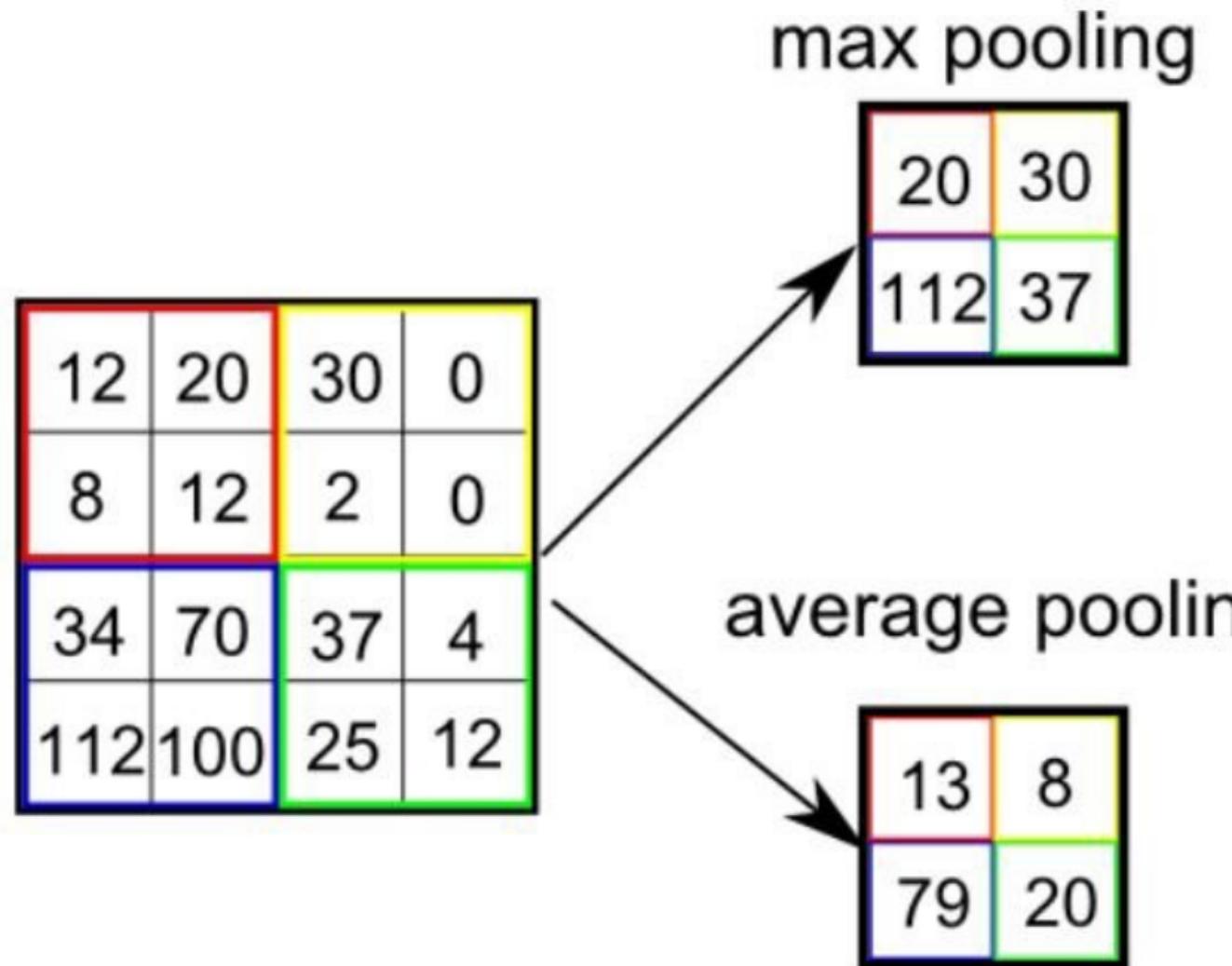
Standard convolution



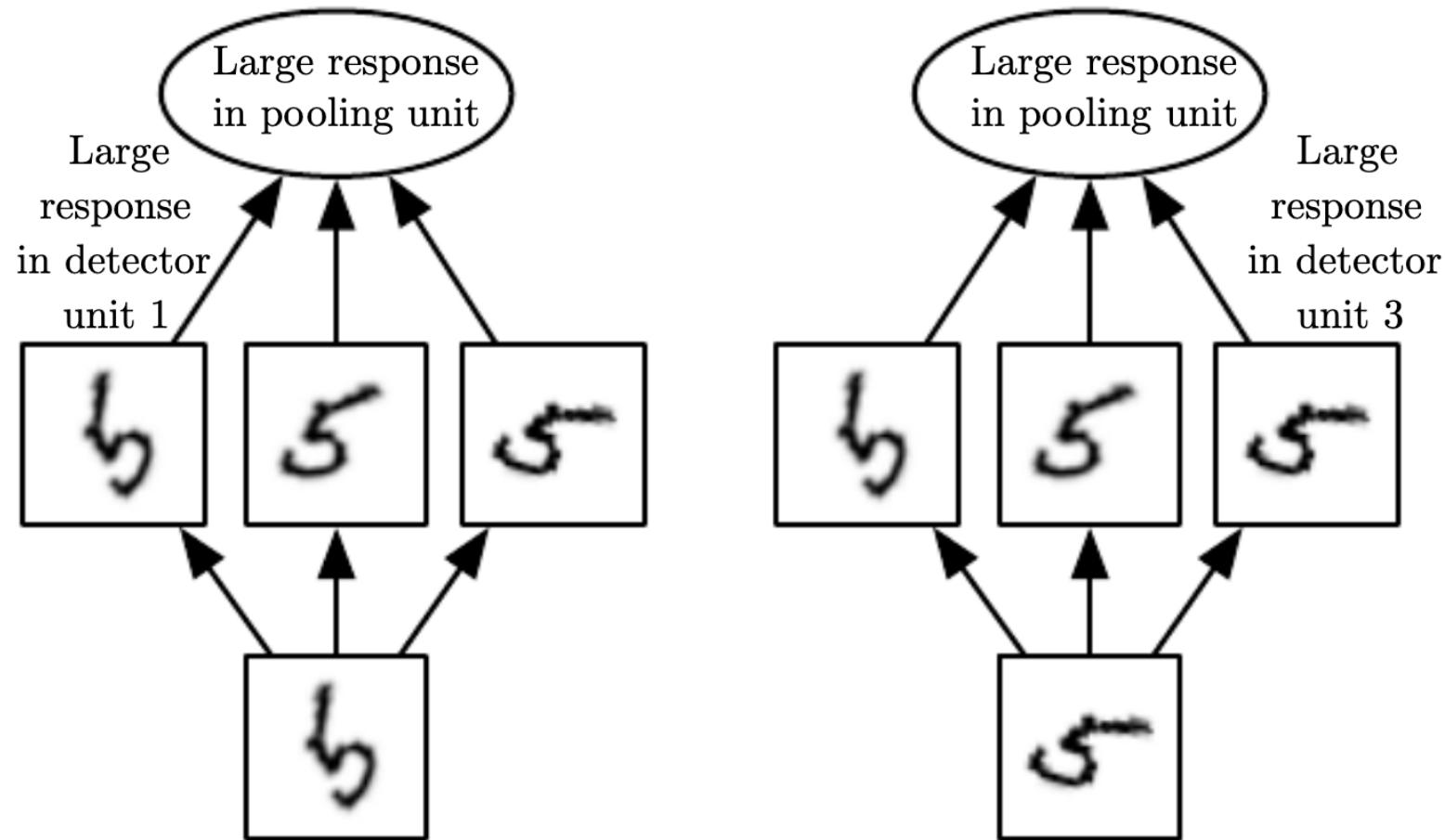
Dilated convolution



Convolutional neural networks: pooling



Convolutional neural networks: pooling

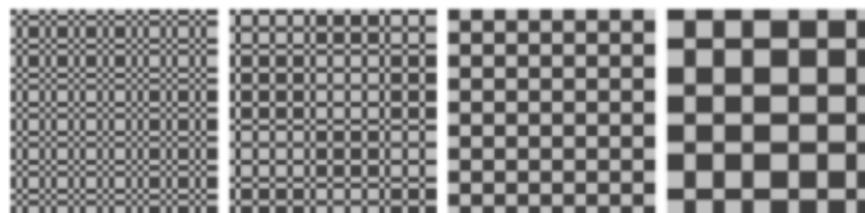


Convolutional neural networks: pooling

Fractional Max-Pooling (*Graham, 2014*)

<https://arxiv.org/pdf/1412.6071.pdf>

- Reduce input dimension by constant alpha
- Max pooling on random tiles of size 2x2, 2x1, 1x2, 1x1
- Overlapping or non-overlapping
- Better generalization

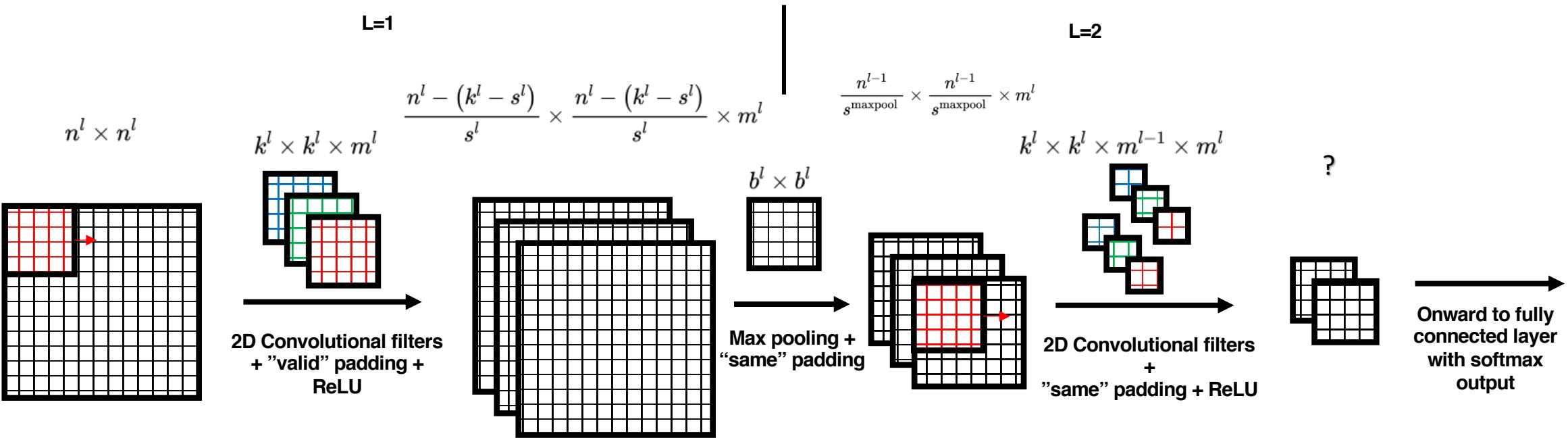


Striving for Simplicity: The All Convolutional Net (*Springenberg et al, 2015*)

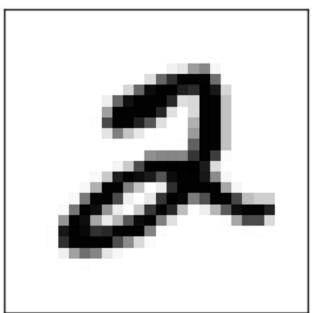
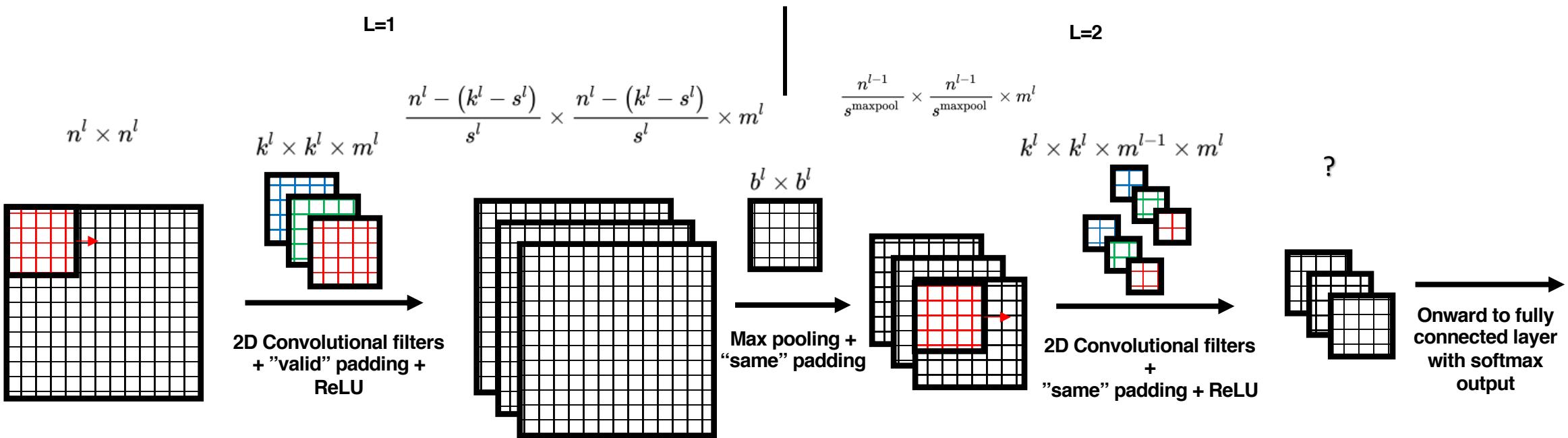
<https://arxiv.org/pdf/1412.6806.pdf>

- No pooling layer
- Convolution with bigger step size instead
- Similar performance

Convolutional neural networks: tracking dimensions



Convolutional neural networks: tracking dimensions

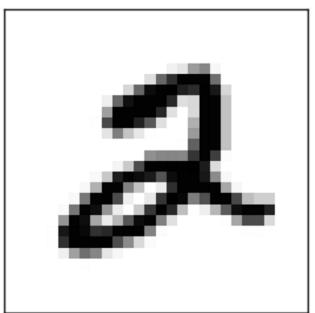
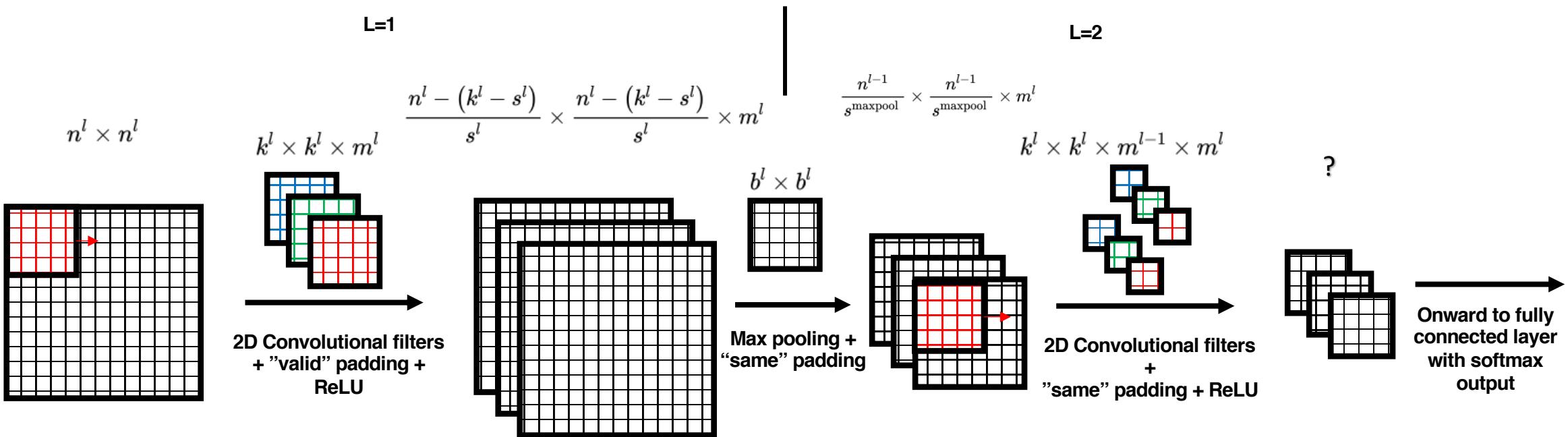


28 x 28 x 1

$$\begin{aligned} k^l &= 5 \\ s^l &= 1 \\ m^l &= 3 \end{aligned}$$

→ 24 x 24 x 3

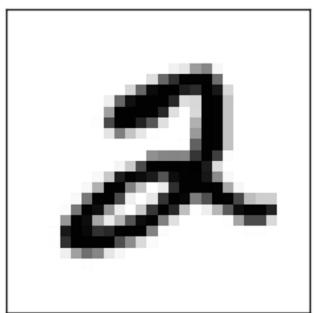
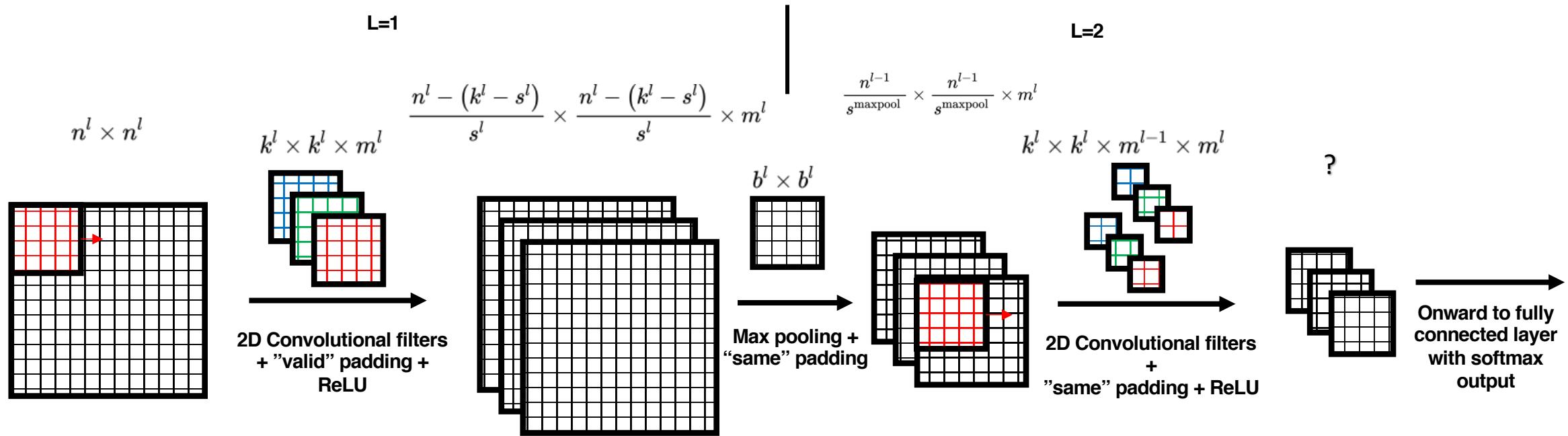
Convolutional neural networks: tracking dimensions



28 x 28 x 1

$$\begin{aligned} k^l &= 5 \\ s^l &= 1 \\ m^l &= 3 \end{aligned} \quad \rightarrow 24 \times 24 \times 3 \quad \rightarrow \quad \begin{aligned} b^l &= 2 \\ s^{\text{maxpool}} &= 2 \end{aligned} \quad \rightarrow 12 \times 12 \times 3$$

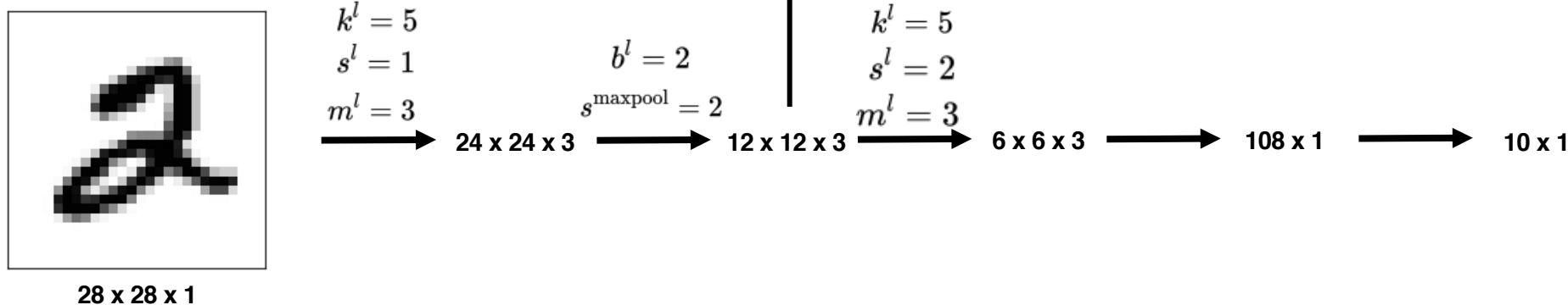
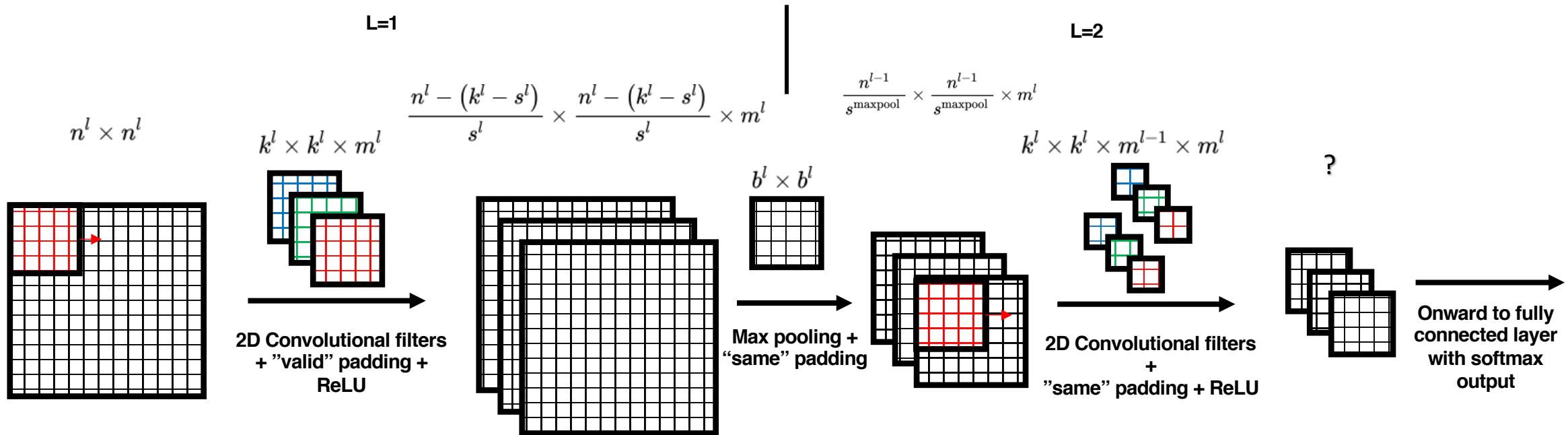
Convolutional neural networks: tracking dimensions



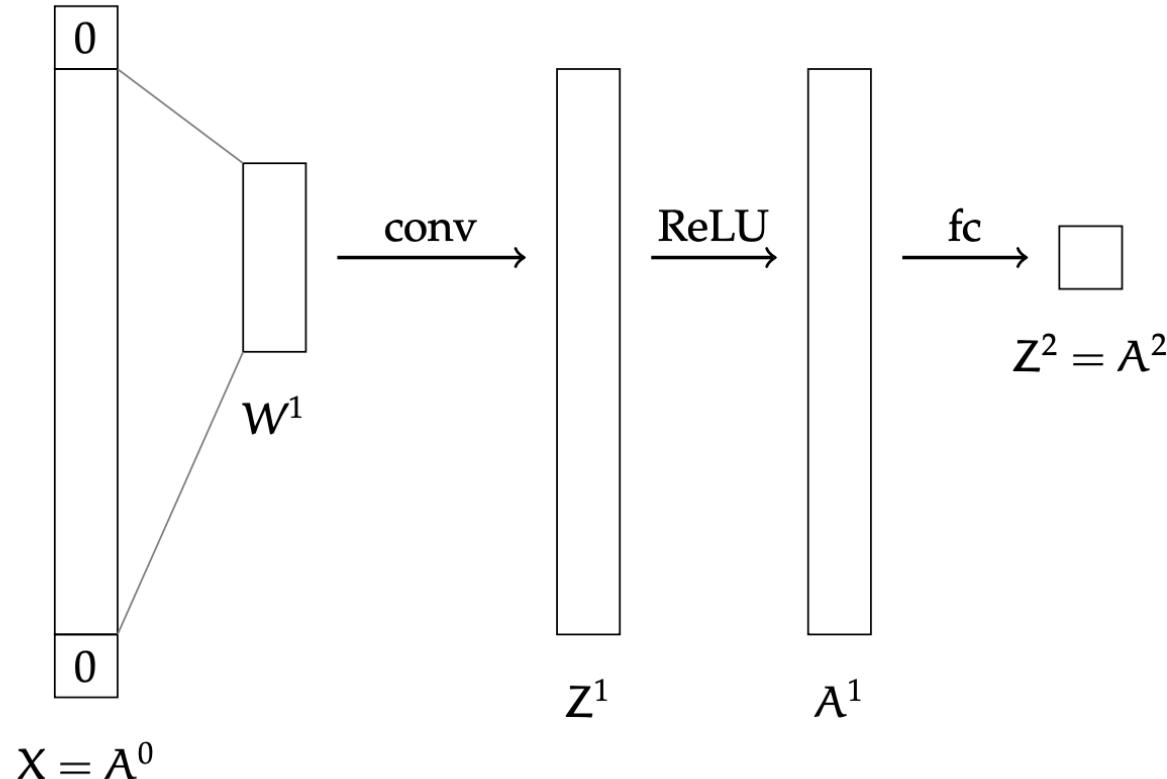
28 x 28 x 1

$$\begin{array}{c} k^l = 5 \\ s^l = 1 \\ m^l = 3 \end{array} \rightarrow 24 \times 24 \times 3 \rightarrow b^l = 2 \quad s^{\text{maxpool}} = 2 \rightarrow 12 \times 12 \times 3 \rightarrow \begin{array}{c} k^l = 5 \\ s^l = 2 \\ m^l = 3 \end{array} \rightarrow 6 \times 6 \times 3$$

Convolutional neural networks: tracking dimensions

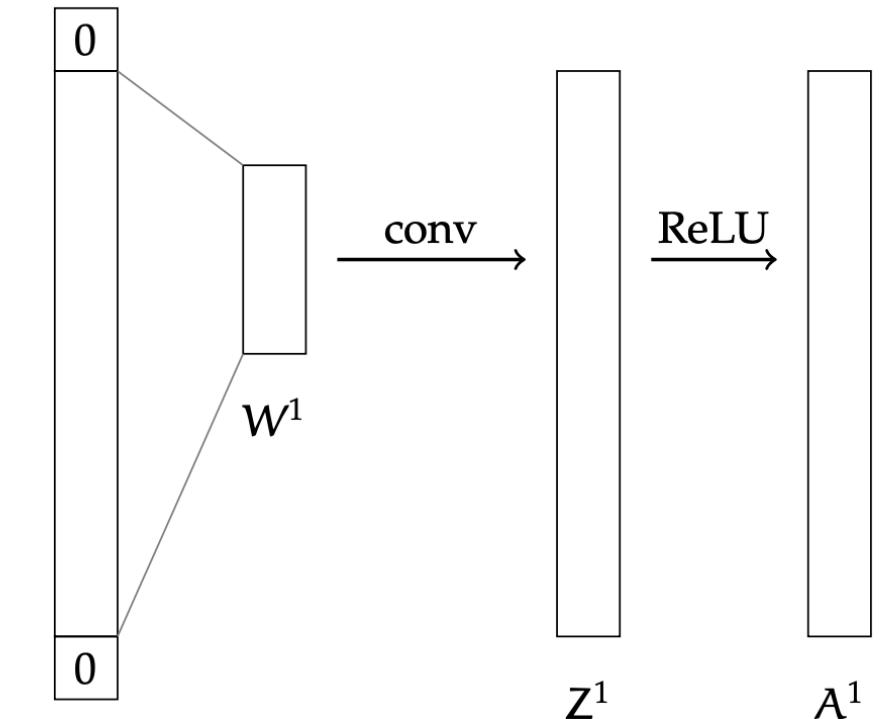


Convolutional neural networks: 1D forward pass



$$\begin{aligned}Z_i^1 &= W^1 \cdot A_{[i-k/2]:i+k/2]}^0 \\A^1 &= \text{ReLU}(Z^1) \\A^2 &= W^{2^T} A^1 \\L(A^2, y) &= (A^2 - y)^2\end{aligned}$$

Convolutional neural networks: 1D backprop



$$X = A^0$$

This should be a $k \times 1$ vector of the gradient

$$\frac{\partial \text{loss}}{\partial W^1} = \frac{\partial Z^1}{\partial W^1} \frac{\partial A^1}{\partial Z^1} \frac{\partial \text{loss}}{\partial A^1}$$

This is an $n \times 1$ vector

$$\frac{\partial Z^1}{\partial W^1} = \begin{pmatrix} 0 & 0 & \dots & a_n^{n-k/2} \\ 0 & a_0^1 & \dots & a_n^{n-k/2} \\ a_0^0 & a_1^1 & \dots & a_n^{n-k/2} \\ a_0^1 & a_1^2 & \dots & a_n^{n-k/2} \\ \vdots & \vdots & \ddots & \vdots \\ a_0^k & a_1^k & \dots & a_n^{n+k/2} \end{pmatrix}$$

This is not exactly accurate but you get the point

$$\frac{\partial A_i^1}{\partial Z_i^1} = \begin{cases} 1 & \text{if } Z_i^1 > 0 \\ 0 & \text{otherwise} \end{cases}$$

This is an $n \times n$ matrix

Structured neural networks : structured data

Example 1 (unstructured): $\mathcal{X} = \mathbb{R}^n$, gene expression values for n genes, order of features (genes) meaningless

Example 2 (1D structure): $\mathcal{X} = \mathbb{R}^{n \times t}$, gene expression values for n genes on t time points

Example 3 (1D structure): $\mathcal{X} = \{0, 1\}^{m \times n}$, biological sequences (DNA, RNA, protein sequence) of length m and alphabet size n , e.g., $n = 4$ (A, C, G, T)

Example 4 (1D structure): Natural language

Example 5 (2D structure): $\mathcal{X} = [0, 1]^{28 \times 28}$, MNIST images

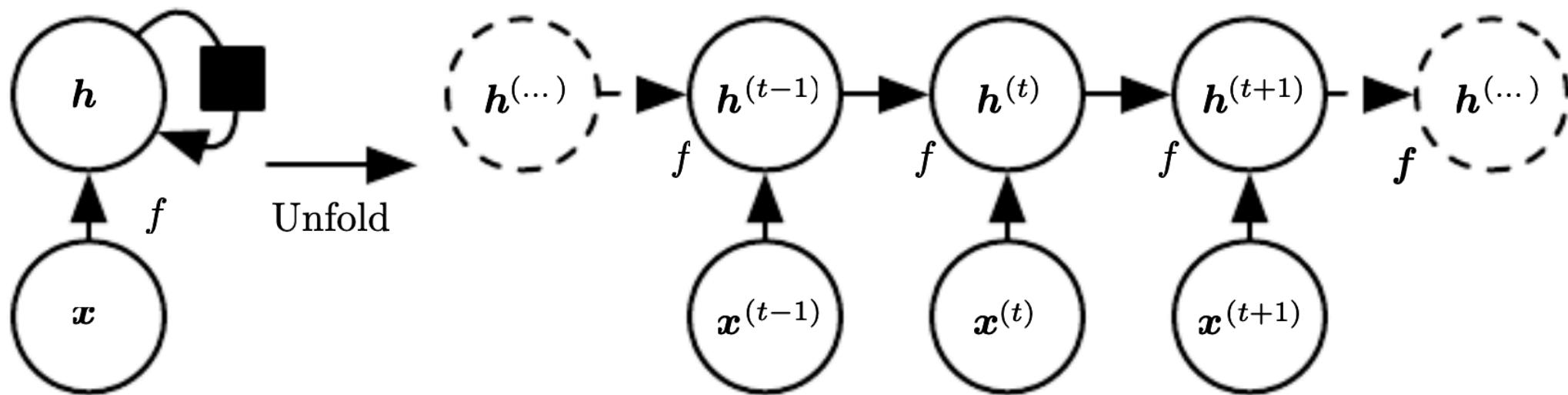
Example 6 (3D structure): $\mathcal{X} = [0, 1]^{m \times n \times p}$, voxels (volumetric pixels) from imaging methods such as CT and MRI

Example 7 (4D structure): $\mathcal{X} = [0, 1]^{m \times n \times p \times t}$, doxels (dynamic voxels) a sequence of voxel data, e.g., time series of CT and MRI scans

Recurrent neural networks: state machine

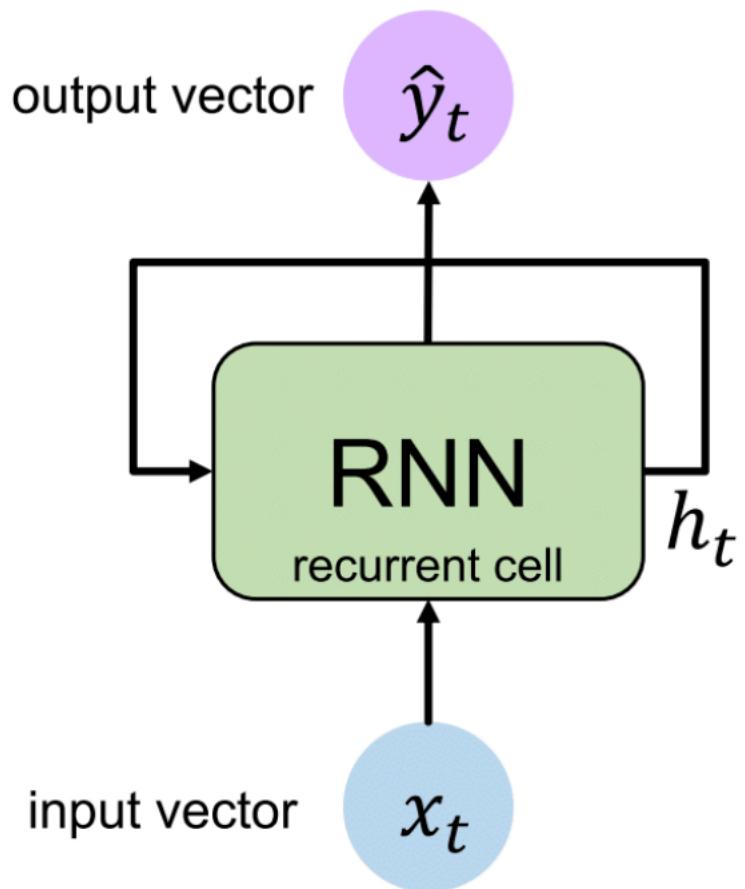
Recurrence relation w/ external input

$$\mathbf{s}^{(t)} = f(\mathbf{s}^{(t-1)}, \mathbf{x}^{(t)}; \boldsymbol{\theta})$$



Recurrent neural networks: back propagation through time

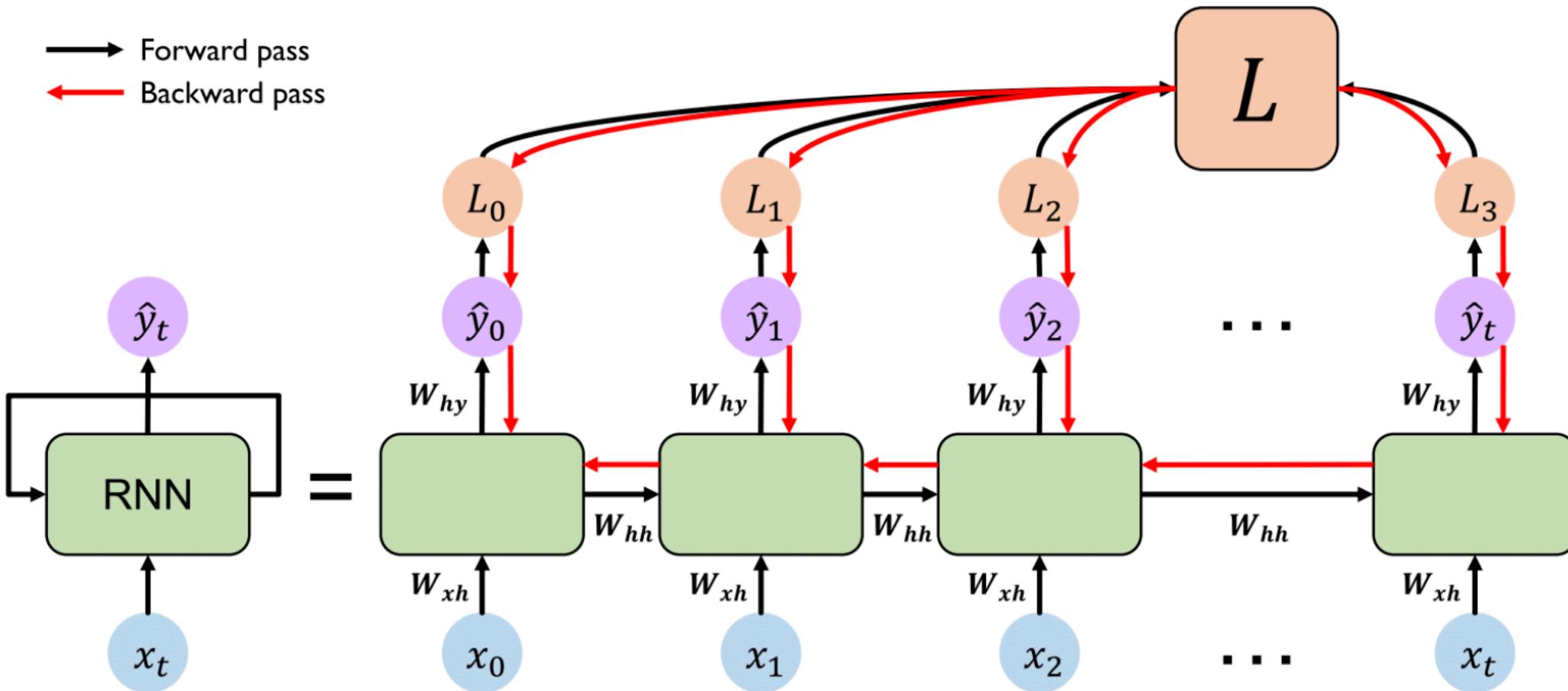
Idea: Learn sequential / temporal relationships



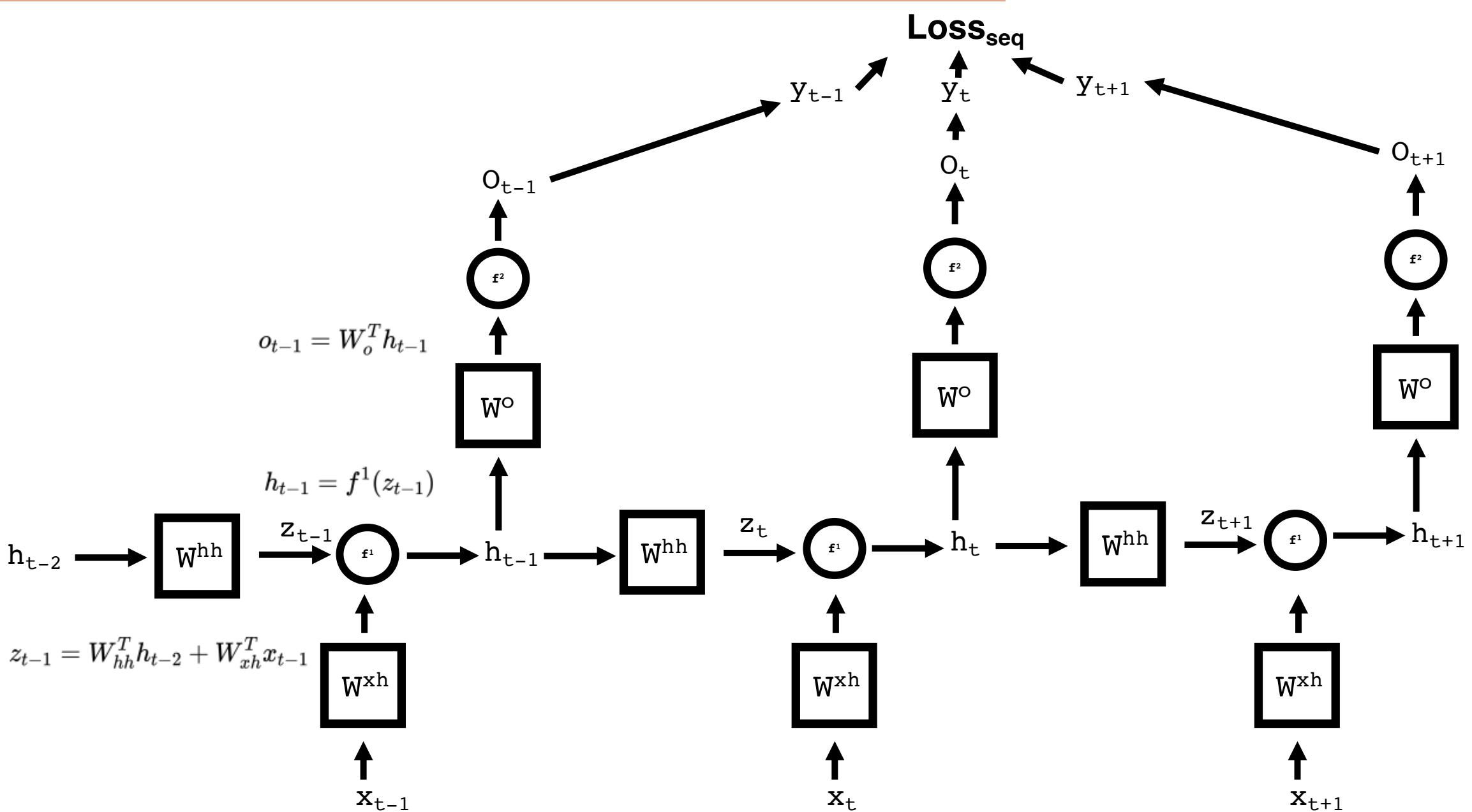
Apply a **recurrence relation** at every time step to process a sequence:

Note: the same function and set of parameters are used at every time step

Recurrent neural networks: back propagation through time



Recurrent neural networks: forward pass



Recurrent neural networks: back propagation through time

For both of the non-output weights (i.e W_{hh}, W_{xh}), we need to find the following,

$$\frac{d\text{Loss}_{\text{seq}}}{dW} = \sum_{u=1}^n \frac{d\text{Loss}_{\text{elt}}(o_u, y_u)}{dW}$$

Now computing the total derivative as the sum of the partials contributing to the gradient with respect to the weight,

$$\begin{aligned} &= \sum_{u=1}^n \sum_{t=1}^n \frac{\partial \text{Loss}_{\text{elt}}(o_u, y_u)}{\partial h_t} \cdot \frac{\partial h_t}{\partial W} \\ &= \sum_{t=1}^n \frac{\partial h_t}{\partial W} \cdot \sum_{u=t}^n \frac{\partial \text{Loss}_{\text{elt}}(o_u, y_u)}{\partial h_t} \end{aligned}$$

Because h_t is only affected by the losses after it, we can rewrite the equation as,

$$= \sum_{t=1}^n \frac{\partial h_t}{\partial W} \cdot \left(\frac{\partial \text{Loss}_{\text{elt}}(o_t, y_t)}{\partial h_t} + \underbrace{\sum_{u=t+1}^n \frac{\partial \text{Loss}_{\text{elt}}(o_u, y_u)}{\partial h_t}}_{\delta^h_t} \right)$$

You can see the second term in the parentheses reflects the gradient of all future losses with respect to the current state. We need to work out how to compute that.

$$F_t = \sum_{u=t+1}^n \text{Loss}_{\text{elt}}(o_u, y_u)$$

Working backwards from the last stage, where the gradient of future losses with respect to the final state is 0,

$$\begin{aligned} \frac{\partial F_{t-1}}{\partial h_{t-1}} &= \frac{\partial}{\partial h_{t-1}} \sum_{u=t}^n \text{Loss}_{\text{elt}}(o_u, y_u) \\ &= \frac{\partial h_t}{\partial h_{t-1}} \cdot \frac{\partial}{\partial h_t} \sum_{u=t}^n \text{Loss}_{\text{elt}}(o_u, y_u) \\ &= \frac{\partial h_t}{\partial h_{t-1}} \cdot \frac{\partial}{\partial h_t} \left[\text{Loss}_{\text{elt}}(o_t, y_t) + \sum_{u=t+1}^n \text{Loss}_{\text{elt}}(o_u, y_u) \right] \\ &= \frac{\partial h_t}{\partial h_{t-1}} \cdot \left[\frac{\partial \text{Loss}_{\text{elt}}(o_t, y_t)}{\partial h_t} + \frac{\partial F_t}{\partial h_t} \right] \end{aligned}$$

We are able to compute the terms above by chain rule,

$$\begin{aligned} \frac{\partial \text{Loss}_{\text{elt}}(o_t, y_t)}{\partial h_t} &= \frac{\partial o_t}{\partial h_t} \cdot \frac{\partial \text{Loss}_{\text{elt}}(o_t, y_t)}{\partial o_t} \\ \frac{\partial h_t}{\partial h_{t-1}} &= \frac{\partial z_t}{\partial h_{t-1}} \cdot \frac{\partial h_t}{\partial z_t} \end{aligned}$$

Now we have all the elements for a weight update, shown below:

$$\frac{d\text{Loss}_{\text{seq}}}{dW} \leftarrow \frac{d\text{Loss}_{\text{seq}}}{dW} + \frac{\partial F_{t-1}}{\partial W} = \frac{d\text{Loss}_{\text{seq}}}{dW} + \frac{\partial z_t}{\partial W} \frac{\partial h_t}{\partial z_t} \left(\frac{\partial o_t}{\partial h_t} \frac{\partial \text{Loss}_{\text{elt}}(h_t, y_t)}{\partial o_t} + \frac{\partial F_t}{\partial h_t} \right)$$

The gradient of W_O is much simpler because it doesn't rely on the future losses, so it ends up looking a lot like error backprop. It would be good practice to try to work it out yourself (using a loss like MSE or NLL)!!

Recurrent neural networks: exploding/vanishing gradients

For both of the non-output weights (i.e W_{hh}, W_{xh}), we need to find the following,

$$\frac{d\text{Loss}_{\text{seq}}}{dW} = \sum_{u=1}^n \frac{d\text{Loss}_{\text{elt}}(o_u, y_u)}{dW}$$

Now computing the total derivative as the sum of the partials contributing to the gradient with respect to the weight,

$$\begin{aligned} &= \sum_{u=1}^n \sum_{t=1}^n \frac{\partial \text{Loss}_{\text{elt}}(o_u, y_u)}{\partial h_t} \cdot \frac{\partial h_t}{\partial W} \\ &= \sum_{t=1}^n \frac{\partial h_t}{\partial W} \cdot \sum_{u=t}^n \frac{\partial \text{Loss}_{\text{elt}}(o_u, y_u)}{\partial h_t} \end{aligned}$$

Because h_t is only affected by the losses after it, we can rewrite the equation as,

$$= \sum_{t=1}^n \frac{\partial h_t}{\partial W} \cdot \left(\frac{\partial \text{Loss}_{\text{elt}}(o_t, y_t)}{\partial h_t} + \underbrace{\sum_{u=t+1}^n \frac{\partial \text{Loss}_{\text{elt}}(o_u, y_u)}{\partial h_t}}_{\delta^h_t} \right)$$

You can see the second term in the parentheses reflects the gradient of all future losses with respect to the current state. We need to work out how to compute that.

$$F_t = \sum_{u=t+1}^n \text{Loss}_{\text{elt}}(o_u, y_u)$$

Working backwards from the last stage, where the gradient of future losses with respect to the final state is 0,

$$\begin{aligned} \frac{\partial F_{t-1}}{\partial h_{t-1}} &= \frac{\partial}{\partial h_{t-1}} \sum_{u=t}^n \text{Loss}_{\text{elt}}(o_u, y_u) \\ &= \frac{\partial h_t}{\partial h_{t-1}} \cdot \frac{\partial}{\partial h_t} \sum_{u=t}^n \text{Loss}_{\text{elt}}(o_u, y_u) \\ &= \frac{\partial h_t}{\partial h_{t-1}} \cdot \frac{\partial}{\partial h_t} \left[\text{Loss}_{\text{elt}}(o_t, y_t) + \sum_{u=t+1}^n \text{Loss}_{\text{elt}}(o_u, y_u) \right] \\ &= \frac{\partial h_t}{\partial h_{t-1}} \cdot \left[\frac{\partial \text{Loss}_{\text{elt}}(o_t, y_t)}{\partial h_t} + \frac{\partial F_t}{\partial h_t} \right] \end{aligned}$$

We are able to compute the terms above by chain rule,

$$\frac{\partial \text{Loss}_{\text{elt}}(o_t, y_t)}{\partial h_t} = \frac{\partial o_t}{\partial h_t} \cdot \frac{\partial \text{Loss}_{\text{elt}}(o_t, y_t)}{\partial o_t}$$

$$\frac{\partial h_t}{\partial h_{t-1}} = \frac{\partial z_t}{\partial h_{t-1}} \cdot \frac{\partial h_t}{\partial z_t}$$

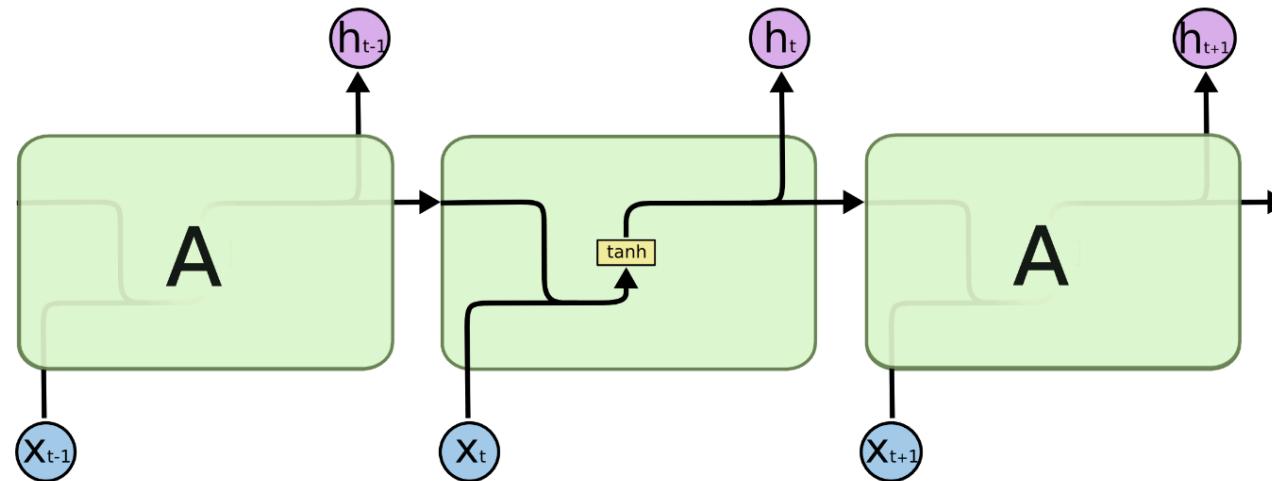
Now we have all the elements for a weight update, shown below:

$$\frac{d\text{Loss}_{\text{seq}}}{dW} \leftarrow \frac{d\text{Loss}_{\text{seq}}}{dW} + \frac{\partial F_{t-1}}{\partial W} = \frac{d\text{Loss}_{\text{seq}}}{dW} + \frac{\partial z_t}{\partial W} \frac{\partial h_t}{\partial z_t} \left(\frac{\partial o_t}{\partial h_t} \frac{\partial \text{Loss}_{\text{elt}}(h_t, y_t)}{\partial o_t} + \frac{\partial F_t}{\partial h_t} \right)$$

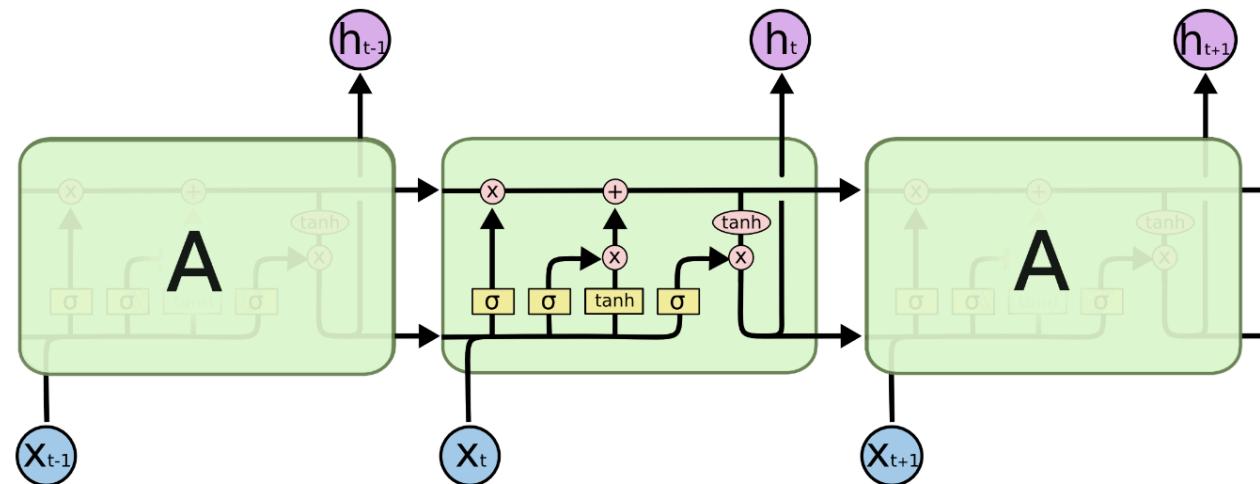
The gradient of W_O is much simpler because it doesn't rely on the future losses, so it ends up looking a lot like error backprop. It would be good practice to try to work it out yourself (using a loss like MSE or NLL)!

Recurrent neural networks: LSTM

Standard RNN has a single layer per “cell state”

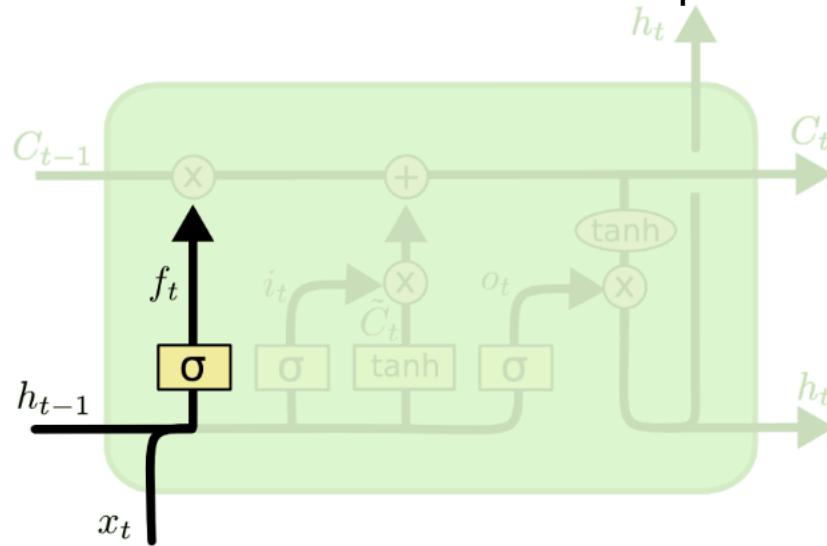


LSTM has multiple layers per “cell state” acting as interacting “gates”



Recurrent neural networks: LSTM

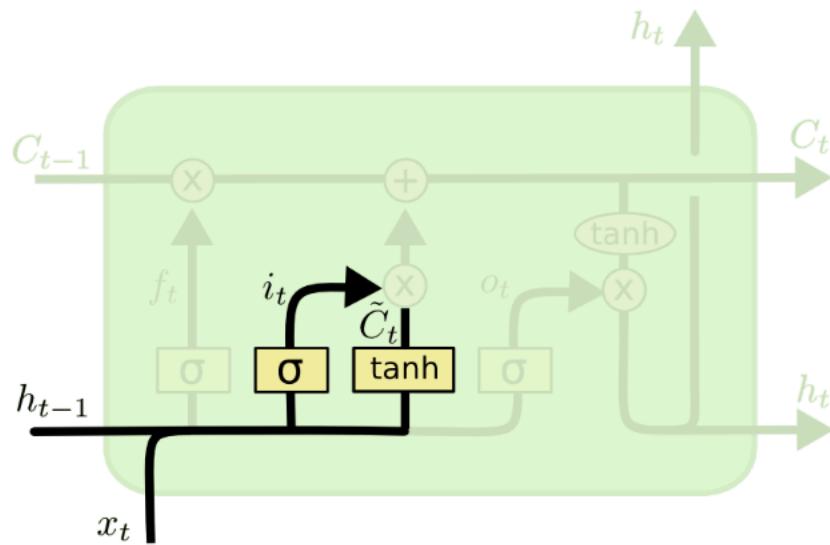
In the “forget gate”, the LSTM decides what information to retain from the old state based on new information in x by assigning a number between 0 and 1 for each unit of the pre-activation output.



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Recurrent neural networks: LSTM

First, a sigmoid layer called the “input gate layer” decides which values we’ll update with new external information. Next, a tanh layer creates a vector of new candidate values, \tilde{C}_t that could be added to the state.

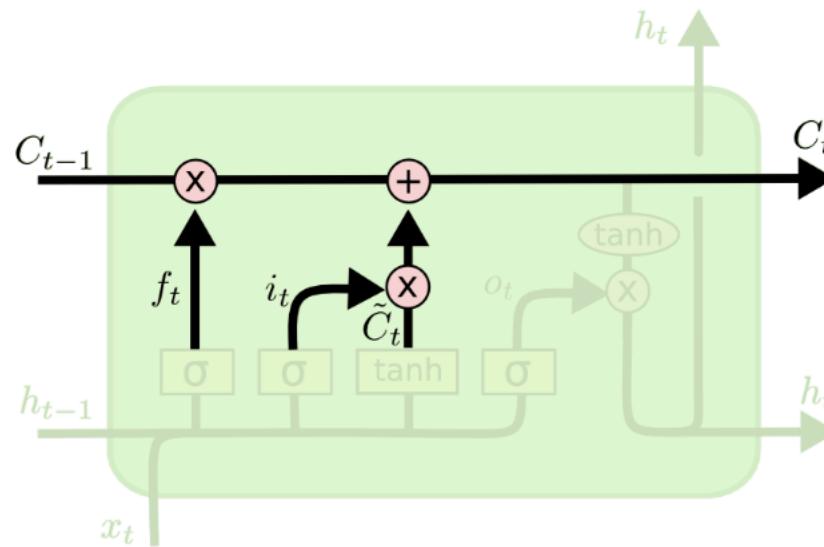


$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Recurrent neural networks: LSTM

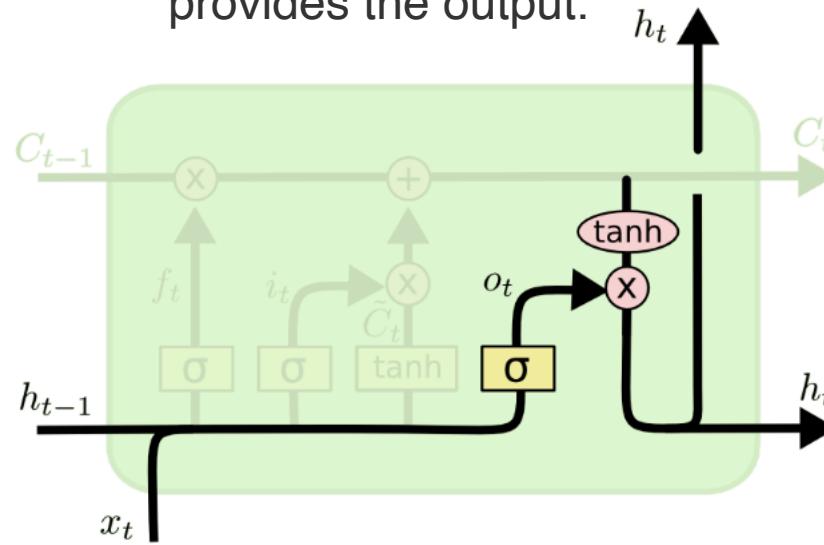
Now, we want to update “cell state” with the outputs of each “gate”. First, we multiply the old state (part of the running state vector) by the forget gate output and add the sum of the input gate and candidate value vector.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Recurrent neural networks: LSTM

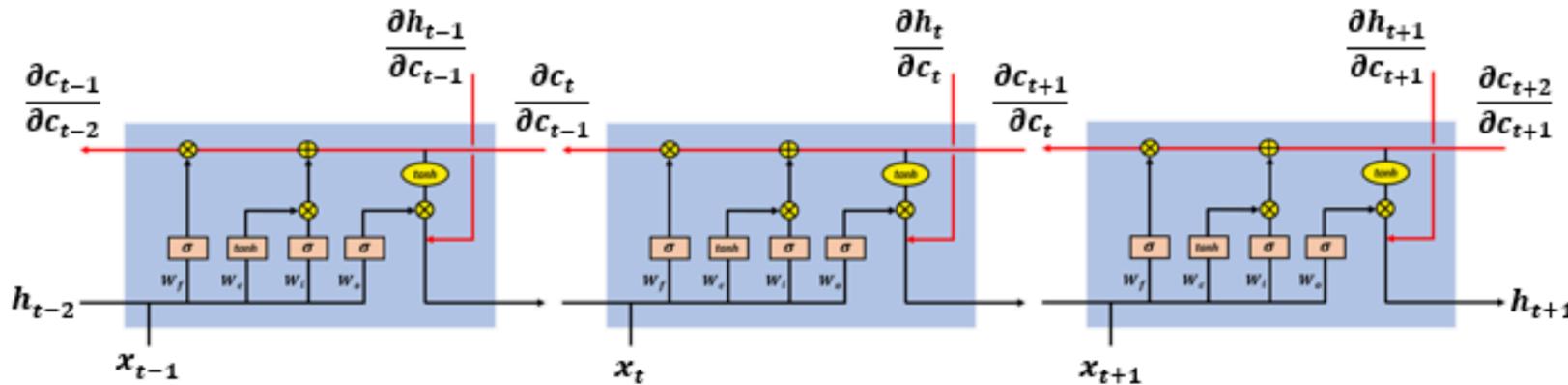
The last piece is the “output” gate. Passing the output of the last cell state and the input to this cell state through a sigmoid activation picks what to output and then multiplying with the tanh activation of the calculated cell states from the previous gates provides the output.



$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

Recurrent neural networks: why do LSTMs prevent gradient problems?



Breaking up the derivative of each state, we can see that the gradient passed between states is additive:

$$A_t = \sigma'(W_f \cdot [h_{t-1}, x_t]) \cdot W_f \cdot o_{t-1} \otimes \tanh'(c_{t-1}) \cdot c_{t-1}$$

$$B_t = f_t$$

$$C_t = \sigma'(W_i \cdot [h_{t-1}, x_t]) \cdot W_i \cdot o_{t-1} \otimes \tanh'(c_{t-1}) \cdot \tilde{c}_t$$

$$D_t = \sigma'(W_c \cdot [h_{t-1}, x_t]) \cdot W_c \cdot o_{t-1} \otimes \tanh'(c_{t-1}) \cdot i_t$$

Giving the gradient of the current state with respect to the previous state stored in the running memory,

$$\frac{\partial c_t}{\partial c_{t-1}} = A_t + B_t + C_t + D_t$$

Now the chained computation of future losses resulting in exploding/vanishing gradients is modified as follows:

$$\frac{\partial F_{t-1}}{h_{t-1}} = \frac{\partial h_t}{\partial c_t} \cdot \frac{\partial c_t}{\partial c_{t-1}} \cdot \left[\frac{\partial \text{Loss}_{\text{ett}}(o_t, y_t)}{\partial h_t} + \frac{\partial F_t}{\partial h_t} \right]$$

The additive term takes advantage of the variety of activation functions to control gradient update coefficients.

Next week: Gene regulation and chromatin accessibility

**The week after: GANs, model uncertainty, and model
interpretability**

Model interpretability: gradient-based methods

For the j th feature of i th datapoint in a dataset, the feature's contribution to the model's pre-activation output is,

$$W_j^{(i)} x_j^{(i)}$$

From our understanding of gradient learning and backpropagation, we can represent the weight above as,

$$W_j^{(i)} = \frac{\partial Z_i}{\partial x_j^{(i)}}$$

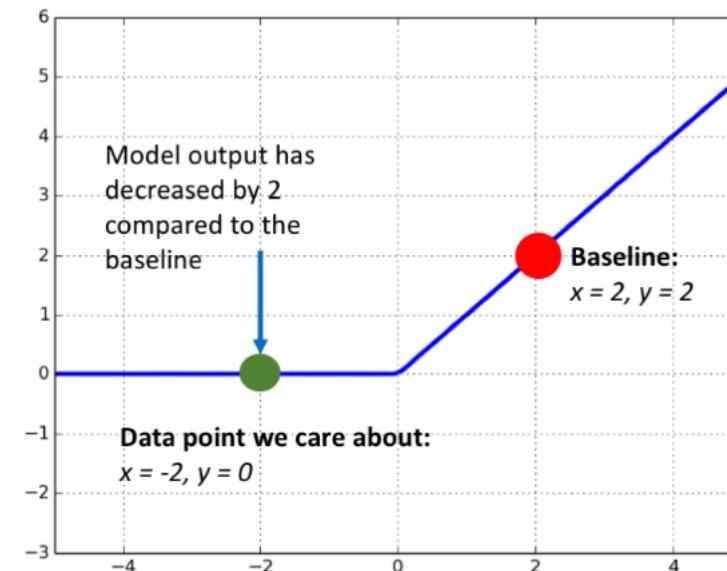
In other words, the weight assigned to the j th feature tells us the gradient of that feature with respect to the pre-activation output (it is trivial to do this for a post-activation output as well). During backprop, we are computing these gradients, and we can now use various methods to leverage gradients for mapping feature contributions.

There is one problem: feature importance is *relative*. To solve this issue, we need a baseline to compare to. For MNIST, a black background is suitable.



With other inputs, picking a baseline is not so intuitive. And in the context of a neural network, you can have *saturation* of a feature gradient. In a ReLU, this is clear, since the output is changing but the gradient is 0.

$$y = \text{ReLU}(x) = \max(0, x)$$



Integrated gradients and DeepLIFT are two approaches to using gradient-based methods and mitigating saturation and numerical problems with computing infinitesimal differences instead of finite differences.

Model interpretability: saliency maps

Visualizing saliency

The procedure is related to back-propagation, but in this case the optimization is performed with respect to the input image, while the weights are fixed to those found during the training stage.

More concretely,

$$S_c(I) \approx W^T I + b$$

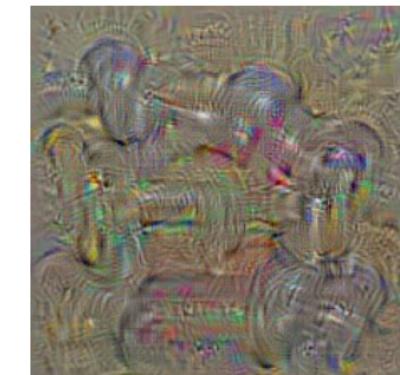
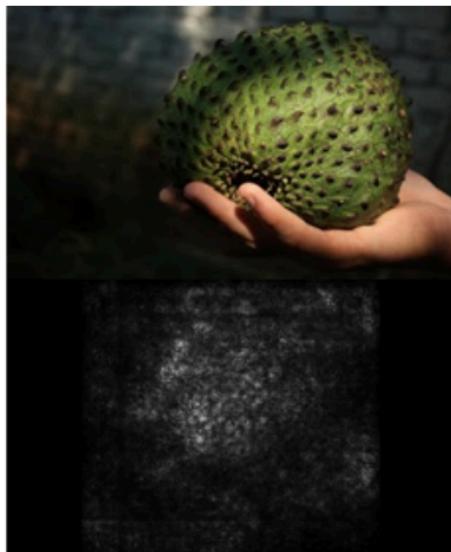
Where,

$$W = \frac{\partial S_c}{\partial I} \Big|_{I_i}$$

Optimization by gradient ascent

More formally, let $S_c(I)$ be the score of the class c , computed by the classification layer of the ConvNet for an image I . We would like to find (via backprop) an L2-regularised image, such that the score S_c is high,

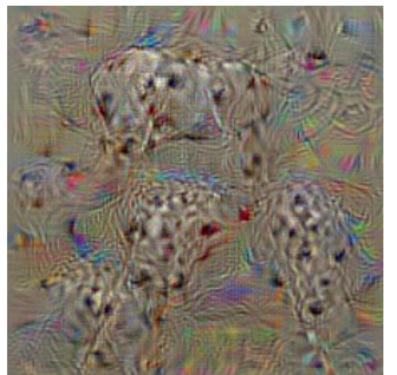
$$\arg \max_I S_c(I) - \lambda \|I\|_2^2$$



dumbbell

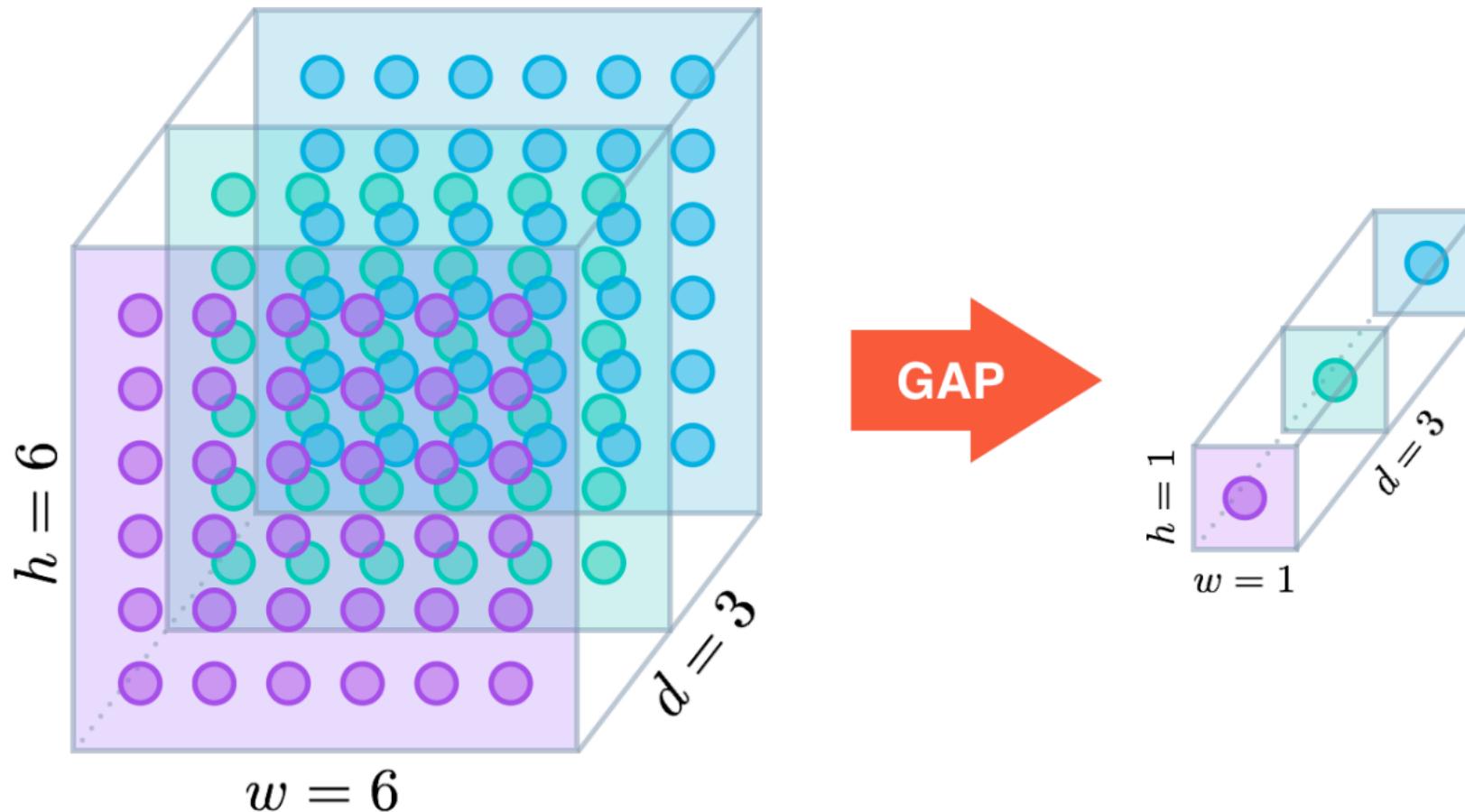


cup



dalmatian

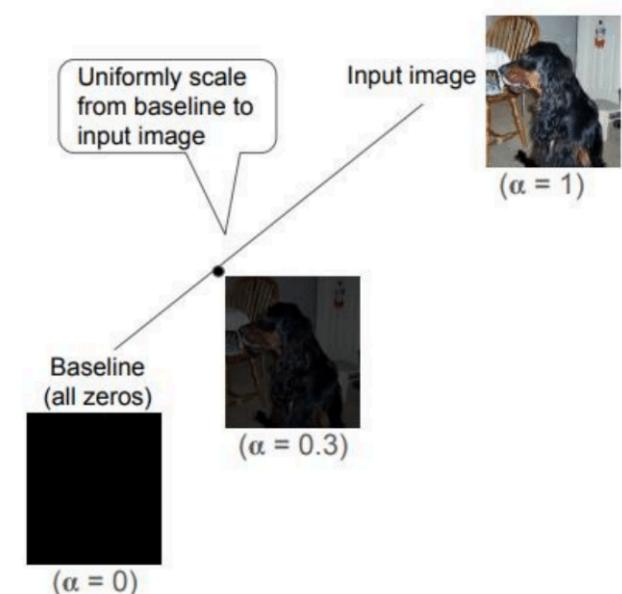
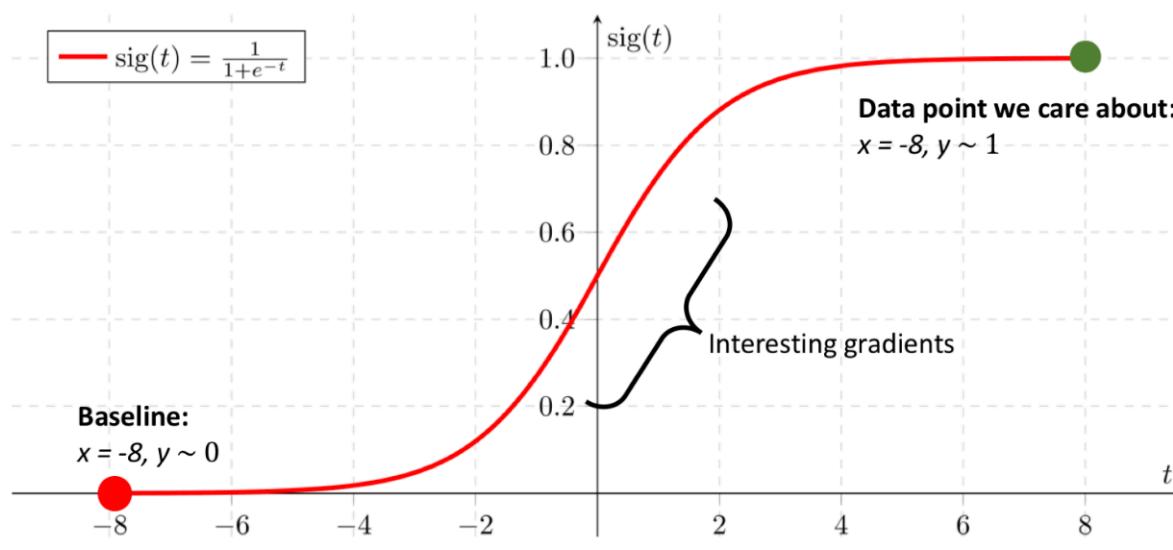
Model interpretability: class activation mapping



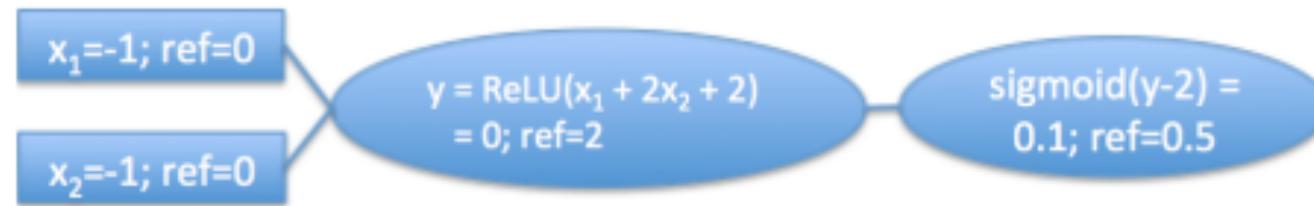
Model interpretability: integrated gradients

$$\text{IntegratedGrads}_i(x) ::= (x_i - x'_i) \times \int_{\alpha=0}^1 \frac{\partial F(x' + \alpha \times (x - x'))}{\partial x_i} d\alpha$$

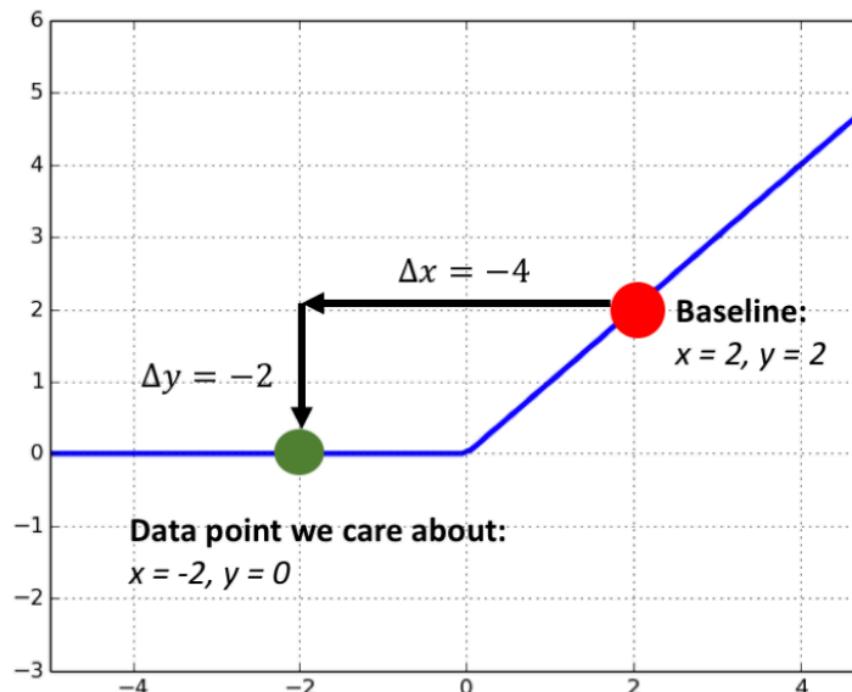
$$\text{Integrated Grads}_i^{\text{approx}}(x) ::= (x_i - x'_i) \times \sum_{k=1}^m \frac{\partial F(x' + \frac{k}{m} \times (x - x'))}{\partial x_i} \times \frac{1}{m}$$



Model interpretability: deeplift



$$y = \text{ReLU}(x) = \max(0, x)$$



1. Calculating the slope

$$\frac{\Delta y}{\Delta x} = \frac{-2}{-4} = 0.5$$

2. Finding the feature importance

$$\Delta x \times \frac{\Delta y}{\Delta x} = -4 \times 0.5 = -2$$