Deep Learning

Deep learning

Yann LeCun[1,2], Yoshua Bengio[3] & Geoffrey Hinton[4,5]

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics. Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer. Deep convolutional nets have brought about breakthroughs in processing images, video, speech and audio, whereas recurrent nets have shone light on sequential data such as text and speech.

Multiple Processing Layers

- Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction.

Applications

- Significant Improvement **in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics.**

Backpropagation

- **Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer**

CNN

- **Deep convolutional nets have brought about breakthroughs in processing images, video, speech and audio, whereas recurrent nets have shone light on sequential data such as text and speech.**

Applications in modern Society

- Web searches
- Content filtering on social net- works
- Recommendations on e-commerce websites,
- Consumer products such as cameras and smartphones
- Identify objects in images
- Transcribe speech into text
- Match news items, posts or products with users' interests
- Select relevant results of search

Machine learning System

- Required careful engineering and considerable domain expertise to design a feature extractor
  - Transform the raw data (such as the pixel values of an image) into a suitable internal representation or feature vector
- Detect or classify patterns in the input

Representation learning

- Discover the representations needed for detection or classification

- Deep-learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level

Very complex functions

- Composition of enough transformations
- The key aspect of deep learning is that these layers of features are not designed by human engineers: they are learned from data using a general-purpose learning procedure.

Deep Learning for Artificial Intelligence

- Discover intricate structures in high-dimensional data and is therefore applicable to many domains of science, business and government
- Beating records in image recognition
  has beaten other machine-learning techniques at predicting the activ- ity of potential drug molecules[8], analysing particle accelerator data[9,10], reconstructing brain circuits[11], and predicting the effects of mutations in non-coding DNA on gene expression and disease[12,13]

- Natural language understanding[14], particularly topic classification, sentiment analysis, question answering[15] and language translation
- Very little engineering by hand, so it can easily take advantage of increases in the amount of available computation and data. New learning algorithms and architectures that are currently being developed for deep neural networks will only accelerate this progress

Supervised learning

- A large data set of images of houses, cars, people and pets, each labelled with its category.
- During training, the machine is shown an image and produces an output in the form of a vector of scores, one for each category.
- We want the desired category to have the highest score of all categories, but this is unlikely to happen before training

Objective Function

- An objective function that measures the error (or distance) between the output scores and the desired pattern of scores.
- The machine then modifies its internal adjustable parameters to reduce this error.
- These adjustable parameters, often called weights, are real numbers that can be seen as ‘knobs’ that define the input–output function of the machine.
- In a typical deep-learning system, there may be hundreds of millions of these adjustable weights, and hundreds of millions of labelled examples with which to train the machine.

Training

- To properly adjust the weight vector, the learning algorithm computes a gradient vector that, for each weight, indicates by what amount the error would increase or decrease if the weight were increased by a tiny amount.
- The weight vector is then adjusted in the opposite direction to the gradient vector

- The objective function, averaged over all the training examples, can be seen as a kind of hilly landscape in the high-dimensional space of weight values.
- The negative gradient vector indicates the direction of steepest descent in this landscape, taking it closer to a minimum, where the output error is low on average

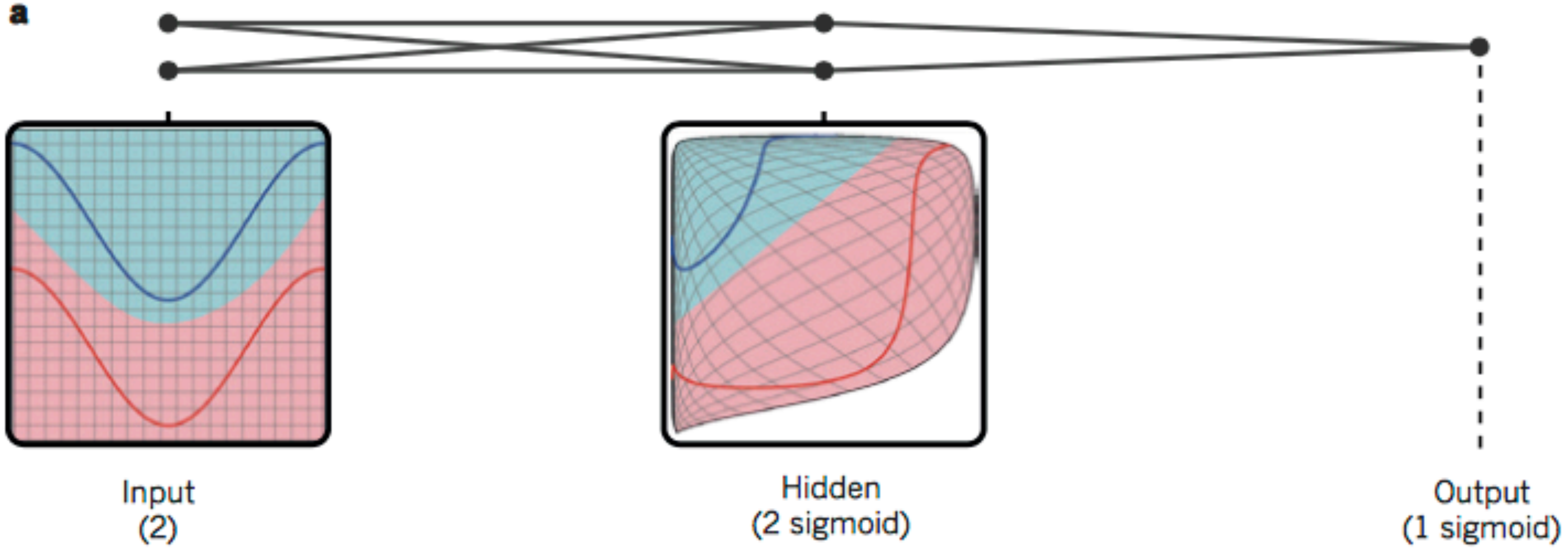
Stochastic gradient descent (SGD)

- This consists of showing the input vector for a few examples, computing the outputs and the errors, computing the average gradient for those examples, and adjusting the weights accordingly.
- The process is repeated for many small sets of examples from the training set until the average of the objective function stops decreasing.
- It is called stochastic because each small set of examples gives a noisy estimate of the average gradient over all examples.
- This simple procedure usually finds a good set of weights surprisingly quickly when compared with far more elaborate optimization techniques

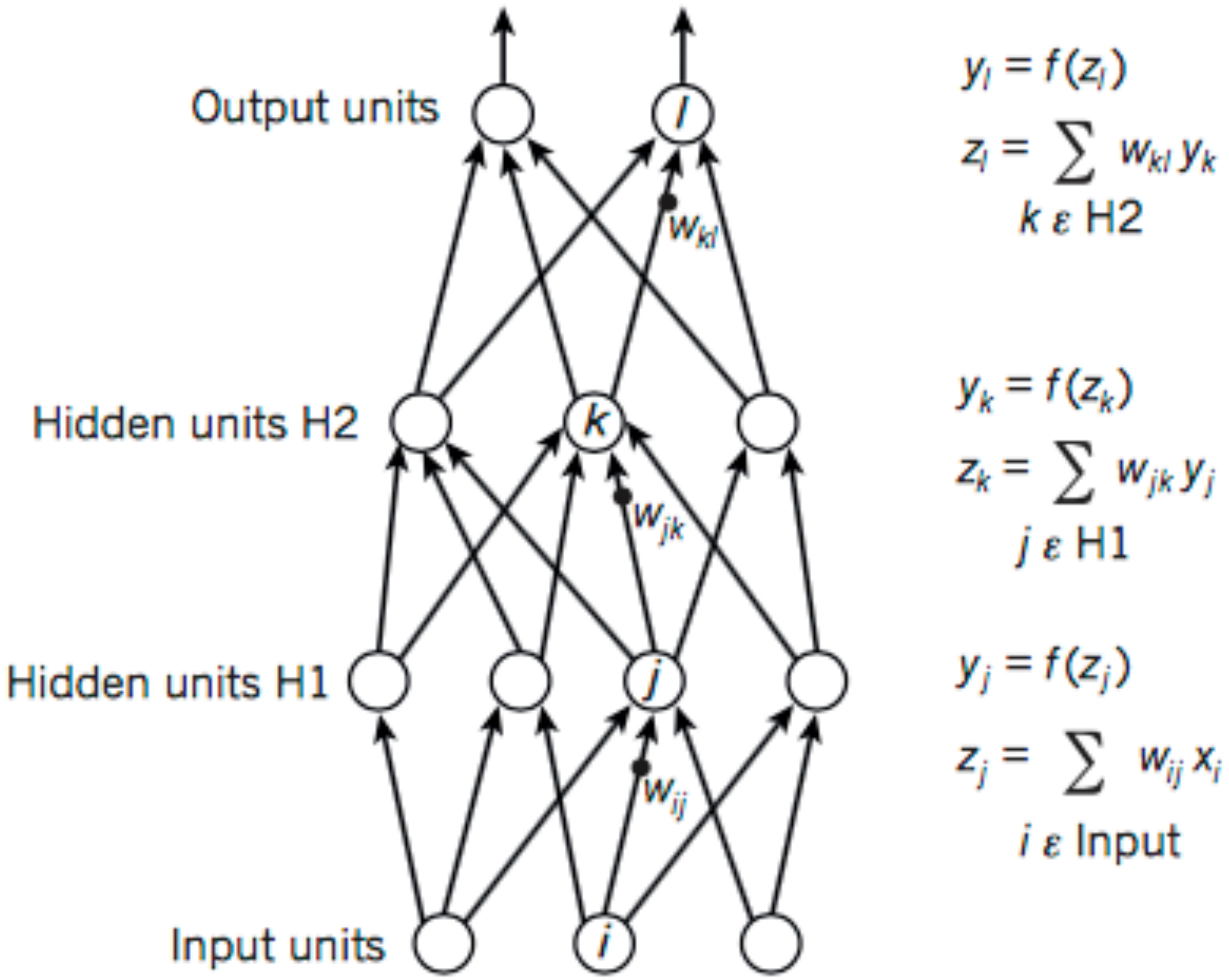
Testing

- After training, the performance of the system is measured on a different set of examples called a test set.
- This serves to test the generalization ability of the machine — its ability to produce sensible answers on new inputs that it has never seen during training.

a



c



Output units
Hidden units H2
Hidden units H1
Input units
l
k
j
i
w_{kl}
w_{jk}
w_{ij}
$y_l = f(z_l)$
$z_l = \sum_{k \,\varepsilon\, \mathrm{H2}} w_{kl}\, y_k$
$y_k = f(z_k)$
$z_k = \sum_{j \,\varepsilon\, \mathrm{H1}} w_{jk}\, y_j$
$y_j = f(z_j)$
$z_j = \sum_{i \,\varepsilon\, \mathrm{Input}} w_{ij}\, x_i$

d

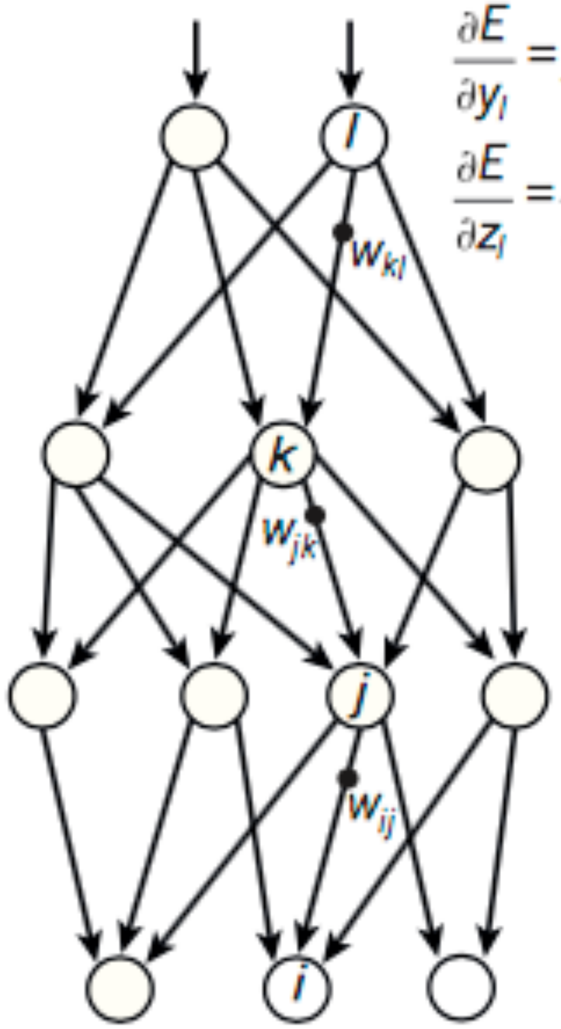
Compare outputs with correct answer to get error derivatives

$$\frac{\partial E}{\partial y_l} = y_l - t_l$$

$$\frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial y_l}\frac{\partial y_l}{\partial z_l}$$

$$\frac{\partial E}{\partial y_k} = \sum_{l\,\varepsilon\,\text{out}} w_{kl}\frac{\partial E}{\partial z_l}$$

$$\frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial z_k}$$



$$\frac{\partial E}{\partial y_j} = \sum_{k\,\varepsilon\,\text{H2}} w_{jk}\frac{\partial E}{\partial z_k}$$

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial z_j}$$

- In the late 1990s, neural nets and backpropagation were largely forsaken by the machine-learning community and ignored by the computer-vision and speech-recognition communities.
- It was widely thought that learning useful, multistage, feature extractors with lit- tle prior knowledge was infeasible.
- In particular, it was commonly thought that simple gradient descent would get trapped in poor local minima — weight configurations for which no small change would reduce the average error.

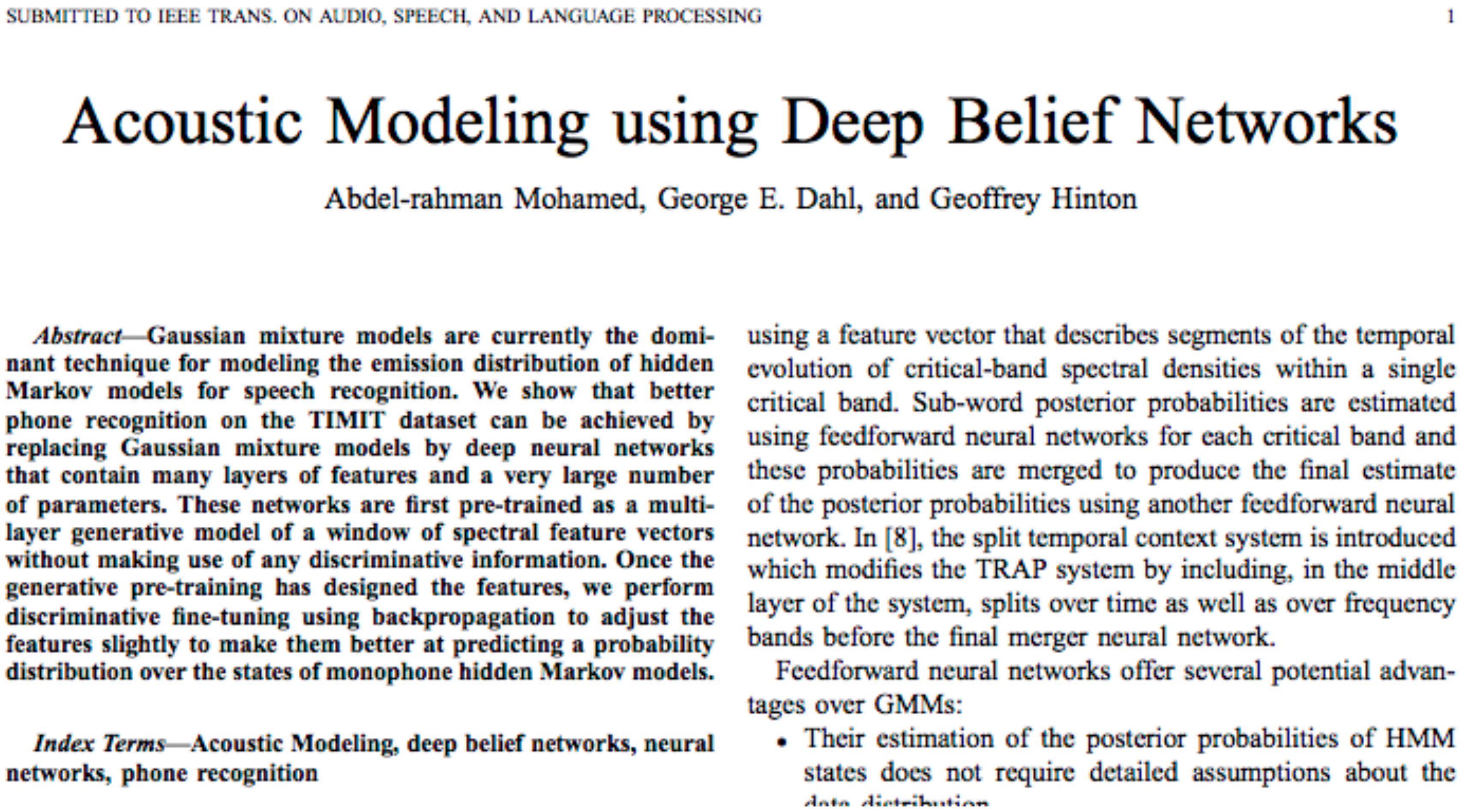
- In practice, poor local minima are rarely a problem with large net- works.
- Regardless of the initial conditions, the system nearly always reaches solutions of very similar quality. Recent theoretical and empirical results strongly suggest that local minima are not a serious issue in general.
- Instead, the landscape is packed with a combinato- rially large number of saddle points where the gradient is zero, and the surface curves up in most dimensions and curves down in the emainder[29,30]. The analysis seems to show that saddle points with only a few downward curving directions are present in very large numbers, but almost all of them have very similar values of the objec- tive function.
- Hence, it does not much matter which of these saddle points the algorithm gets stuck at.

- Interest in deep feedforward networks was revived around 2006 (refs 31–34) by a group of researchers brought together by the Cana- dian Institute for Advanced Research (CIFAR).
- The researchers intro- duced unsupervised learning procedures that could create layers of feature detectors without requiring labelled data. The objective in learning each layer of feature detectors was to be able to reconstruct or model the activities of feature detectors (or raw inputs) in the layer below.
- By 'pre-training' several layers of progressively more complex feature detectors using this reconstruction objective, the weights of a deep network could be initialized to sensible values.
- A final layer of output units could then be added to the top of the network and the whole deep system could be fine-tuned using standard backpropaga- tion [33–35].
- This worked remarkably well for recognizing handwritten digits or for detecting pedestrians, especially when the amount of labelled data was very limited36.

- The first major application of this pre-training approach was in speech recognition, and it was made possible by the advent of fast graphics processing units (GPUs) that were convenient to program[37] and allowed researchers to train networks 10 or 20 times faster.
- In 2009, the approach was used to map short temporal windows of coef- ficients extracted from a sound wave to a set of probabilities for the various fragments of speech that might be represented by the frame in the centre of the window. It achieved record-breaking results on a standard speech recognition benchmark that used a small vocabu- lary[38] and was quickly developed to give record-breaking results on a large vocabulary task[39].

- Mohamed,A.-R.,Dahl,G.E.&Hinton,G. Acoustic modeling using deep belief networks. *IEEE Trans. Audio Speech Lang. Process.* **20,** 14–22 (2012).
- Dahl,G.E.,Yu,D.,Deng,L.&Acero,A.Context-dependent pre-trained deep neural networks for large vocabulary speech recognition. *IEEE Trans. Audio Speech Lang. Process.* **20,** 33–42 (2012).

http://www.cs.toronto.edu/~asamir/papers/speechDBN_jrnl.pdf



Acoustic Modeling using Deep Belief Networks

Abdel-rahman Mohamed, George E. Dahl, and Geoffrey Hinton

***Abstract*—Gaussian mixture models are currently the dominant technique for modeling the emission distribution of hidden Markov models for speech recognition. We show that better phone recognition on the TIMIT dataset can be achieved by replacing Gaussian mixture models by deep neural networks that contain many layers of features and a very large number of parameters. These networks are first pre-trained as a multi-layer generative model of a window of spectral feature vectors without making use of any discriminative information. Once the generative pre-training has designed the features, we perform discriminative fine-tuning using backpropagation to adjust the features slightly to make them better at predicting a probability distribution over the states of monophone hidden Markov models.**

***Index Terms*—Acoustic Modeling, deep belief networks, neural networks, phone recognition**

using a feature vector that describes segments of the temporal evolution of critical-band spectral densities within a single critical band. Sub-word posterior probabilities are estimated using feedforward neural networks for each critical band and these probabilities are merged to produce the final estimate of the posterior probabilities using another feedforward neural network. In [8], the split temporal context system is introduced which modifies the TRAP system by including, in the middle layer of the system, splits over time as well as over frequency bands before the final merger neural network.

Feedforward neural networks offer several potential advantages over GMMs:

- Their estimation of the posterior probabilities of HMM states does not require detailed assumptions about the

```
for l = 1 : numbatches
    batch = x(kk((l - 1) * opts.batchsize + 1 : l * opts.batchsize), :);

    v1 = batch;
    h1 = sigmrnd(repmat(rbm.c', opts.batchsize, 1) + v1 * rbm.W');
    v2 = sigmrnd(repmat(rbm.b', opts.batchsize, 1) + h1 * rbm.W);
    h2 = sigm(repmat(rbm.c', opts.batchsize, 1) + v2 * rbm.W');

    c1 = h1' * v1;
    c2 = h2' * v2;

    rbm.vW = rbm.momentum * rbm.vW + rbm.alpha * (c1 - c2)     / opts.batchsize;
    rbm.vb = rbm.momentum * rbm.vb + rbm.alpha * sum(v1 - v2)' / opts.batchsize;
    rbm.vc = rbm.momentum * rbm.vc + rbm.alpha * sum(h1 - h2)' / opts.batchsize;

    rbm.W = rbm.W + rbm.vW;
    rbm.b = rbm.b + rbm.vb;
    rbm.c = rbm.c + rbm.vc;

    err = err + sum(sum((v1 - v2) .^ 2)) / opts.batchsize;
end
```

$$p(h_j = 1 \mid v, \theta) = sigmoid(c_j + \sum_i w_{ij} v_i)$$

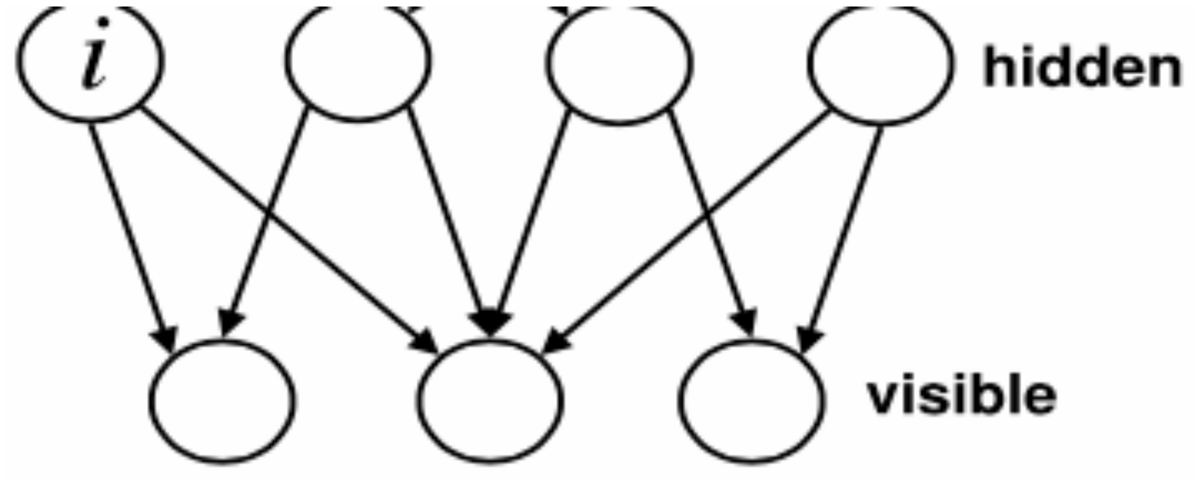
$$p(v_i = 1 \mid h, \theta) = sigmoid(b_i + \sum_j w_{ij} h_j)$$

v1 : input

h1 : generated hidden state given v1 and W

v2 : reconstructed state of visible units

h2 : inferred hidden state

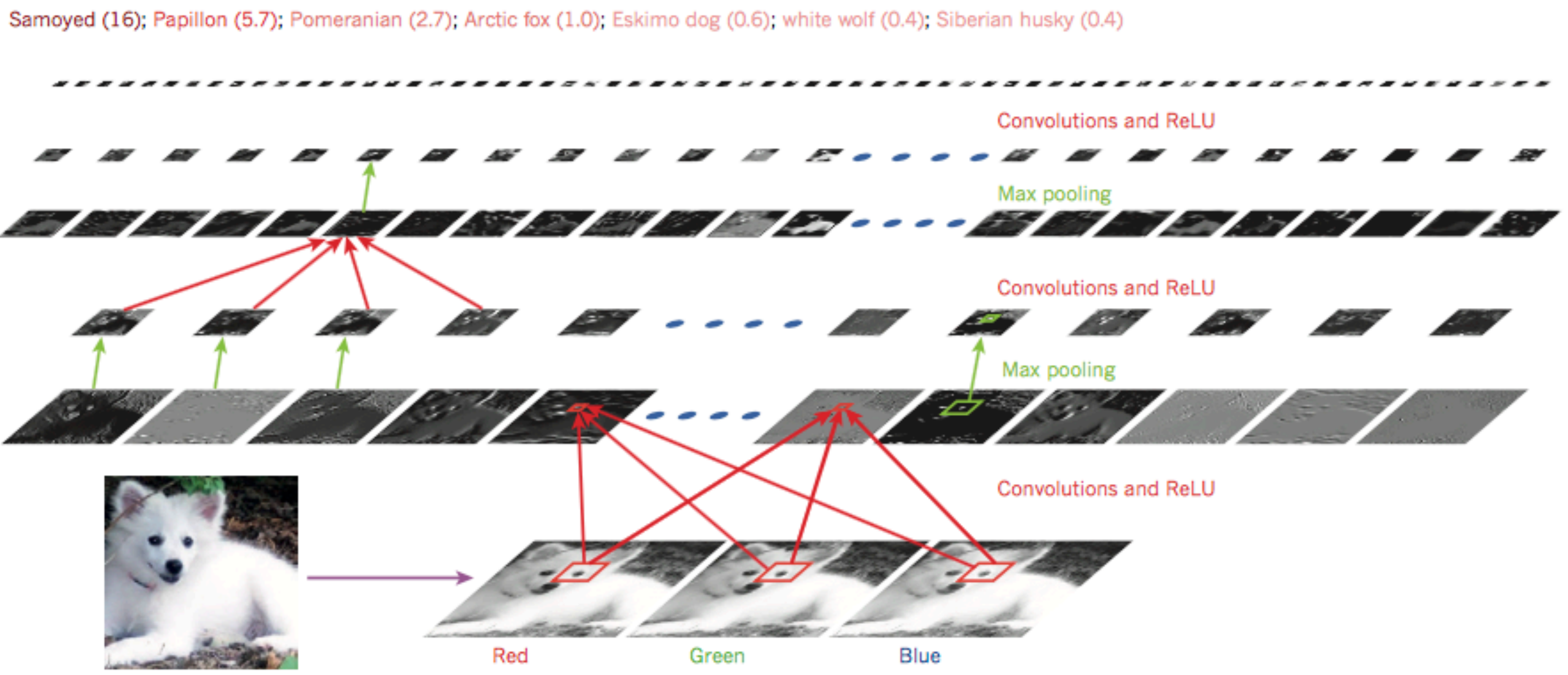


The tied weights mean that the process of inferring $\mathbf{h}^{(2)}$ from $\mathbf{h}^{(1)}$ is the same as the process of generating $\mathbf{v}$ from $\mathbf{h}^{(1)}$. Consequently, $\mathbf{h}^{(2)}$ can be viewed as a noisy but unbiased estimate of the probabilities for the visible units predicted by $\mathbf{h}^{(1)}$. Similarly $\mathbf{h}^{(3)}$ can be viewed as a noisy estimate of the probabilities for the units in the first hidden layer predicted by $\mathbf{h}^{(2)}$. We can use these two facts and equation 9 to get an unbiased estimate of the sum of the derivatives for the first two layers of weights. This gives the following learning rule which is known as "contrastive divergence" [10]:

$$\begin{aligned}\Delta w_{ij} &\propto \left\langle h_j^{(1)}(v_i - h_i^{(2)}) + h_i^{(2)}(h_j^{(1)} - h_j^{(3)})\right\rangle \\ &\propto \langle v_i h_j^{(1)}\rangle - \langle h_i^{(2)} h_j^{(3)}\rangle \qquad (11)\end{aligned}$$

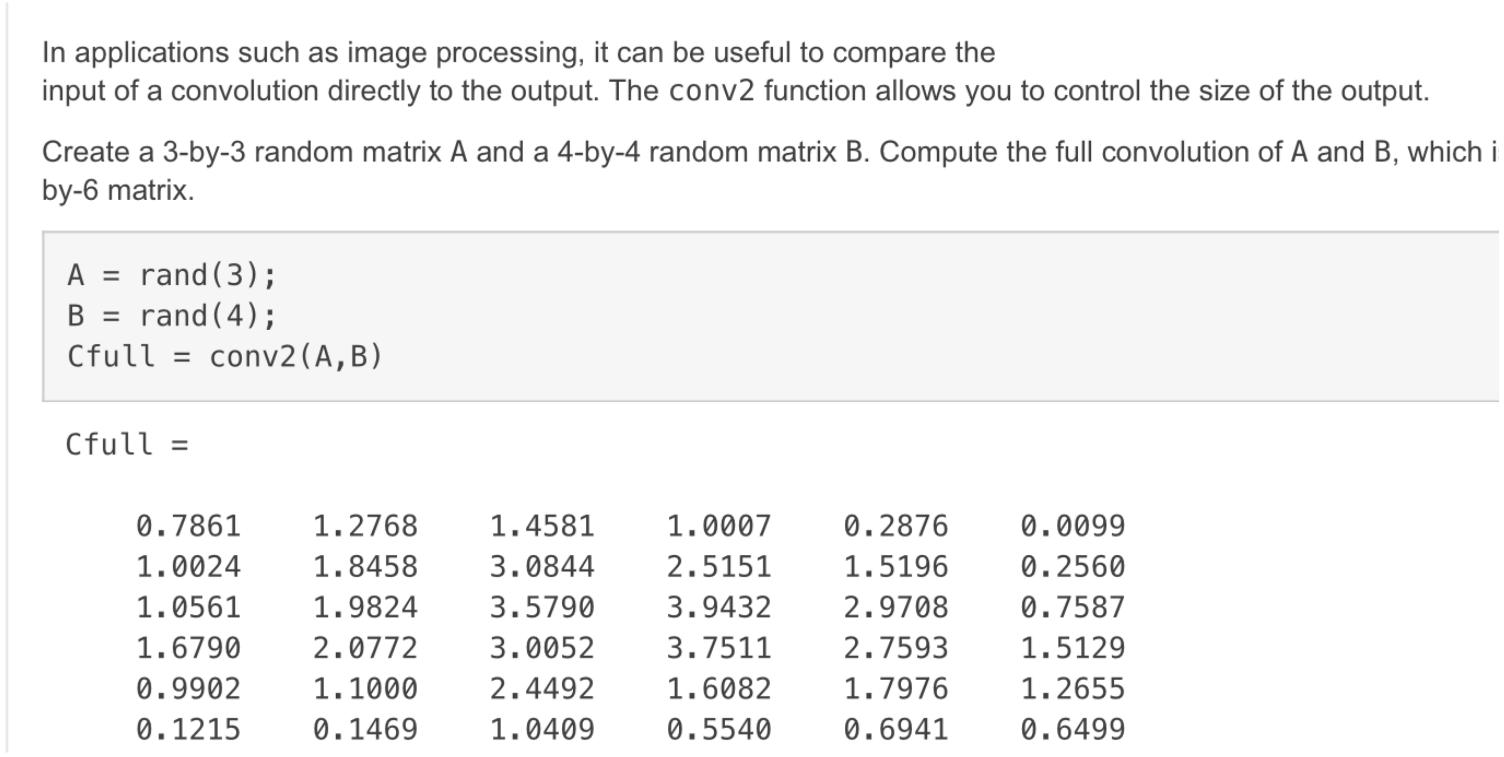
where the angle brackets denote expectations over the training data (or a representative mini-batch).

- By 2012, versions of the deep net from 2009 were being developed by many of the major speech groups[6] and were already being deployed in Android phones. For smaller data sets, unsupervised pre-training helps to prevent overfitting[40], leading to significantly better generalization when the number of labelled examples is small, or in a transfer setting where we have lots of examples for some ‘source’ tasks but very few for some ‘target’ tasks. Once deep learning had been rehabilitated, it turned out that the pre-training stage was only needed for small data sets.


Samoyed (16); Papillon (5.7); Pomeranian (2.7); Arctic fox (1.0); Eskimo dog (0.6); white wolf (0.4); Siberian husky (0.4)
Convolutions and ReLU
Max pooling
Convolutions and ReLU
Max pooling
Convolutions and ReLU
Red
Green
Blue

matlab conv2 & convn

https://www.mathworks.com/help/matlab/ref/conv2.html?requestedDomain=www.mathworks.com&requestedDomain=www.mathworks.com



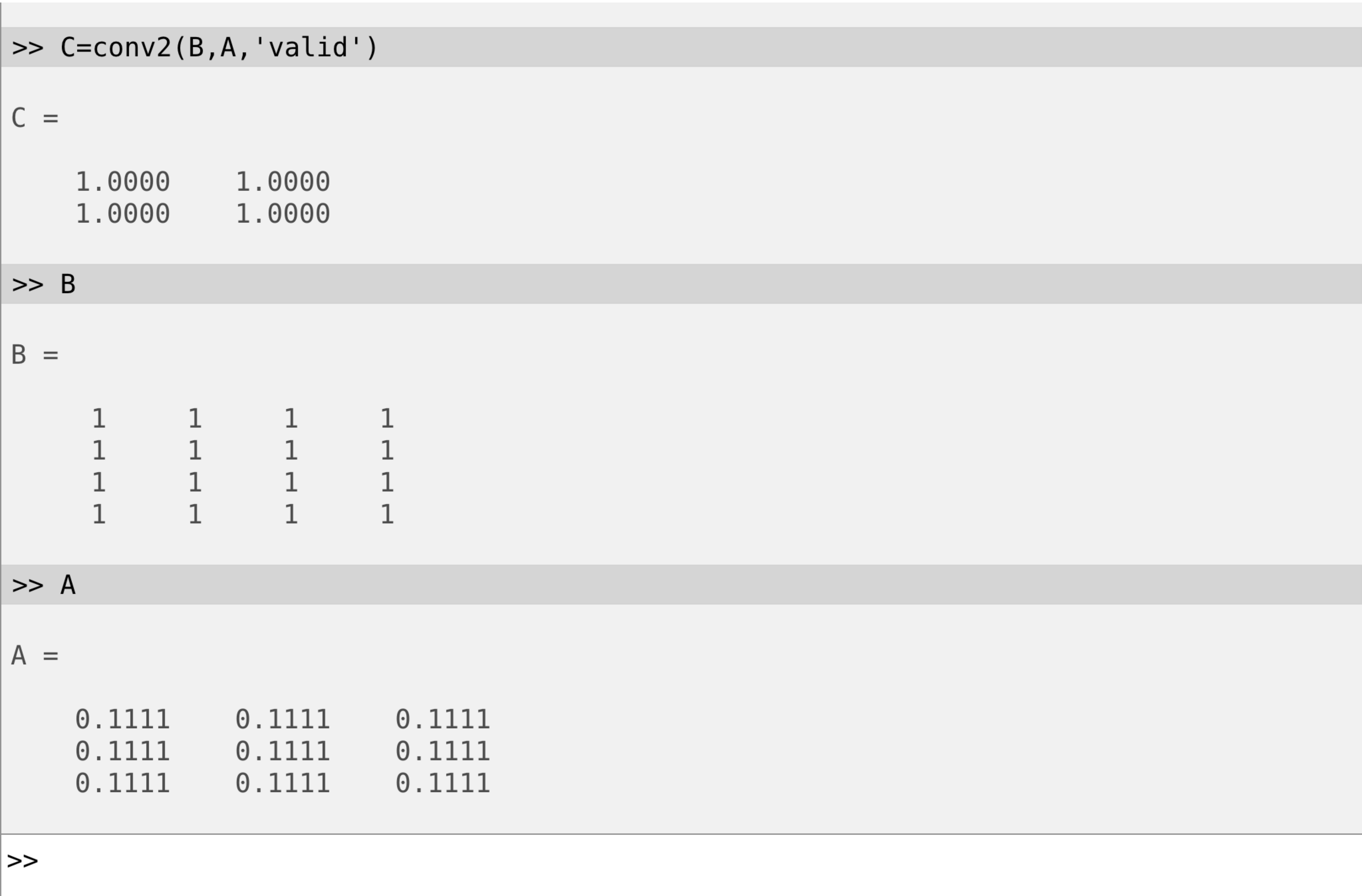
In applications such as image processing, it can be useful to compare the input of a convolution directly to the output. The conv2 function allows you to control the size of the output.

Create a 3-by-3 random matrix A and a 4-by-4 random matrix B. Compute the full convolution of A and B, which i
by-6 matrix.

```
A = rand(3);
B = rand(4);
Cfull = conv2(A,B)
```

```
Cfull =

    0.7861    1.2768    1.4581    1.0007    0.2876    0.0099
    1.0024    1.8458    3.0844    2.5151    1.5196    0.2560
    1.0561    1.9824    3.5790    3.9432    2.9708    0.7587
    1.6790    2.0772    3.0052    3.7511    2.7593    1.5129
    0.9902    1.1000    2.4492    1.6082    1.7976    1.2655
    0.1215    0.1469    1.0409    0.5540    0.6941    0.6499
```

```
B =

     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1

>> C=conv2(A,B)

C =

    0.1111    0.2222    0.3333    0.3333    0.2222    0.1111
    0.2222    0.4444    0.6667    0.6667    0.4444    0.2222
    0.3333    0.6667    1.0000    1.0000    0.6667    0.3333
    0.3333    0.6667    1.0000    1.0000    0.6667    0.3333
    0.2222    0.4444    0.6667    0.6667    0.4444    0.2222
    0.1111    0.2222    0.3333    0.3333    0.2222    0.1111

>> A

A =

    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111
```



```
>> C=conv2(B,A,'valid')

C =

    1.0000    1.0000
    1.0000    1.0000

>> B

B =

     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1

>> A

A =

    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111

>>
```

3D convolution

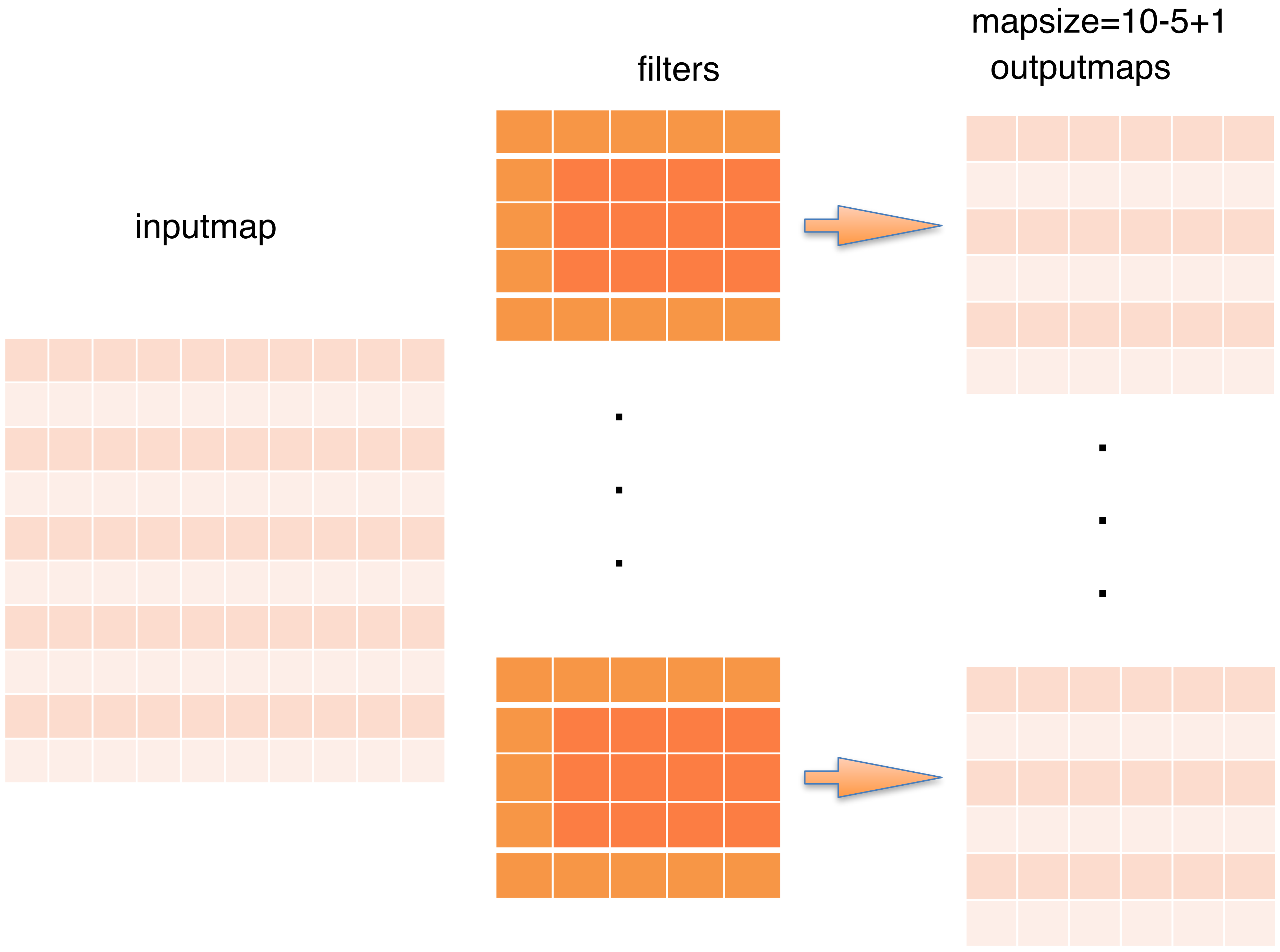
```
>> A=ones(3,3,3)
B=ones(4,4,4)
A=A/27
C=convn(B,A,'valid')
```

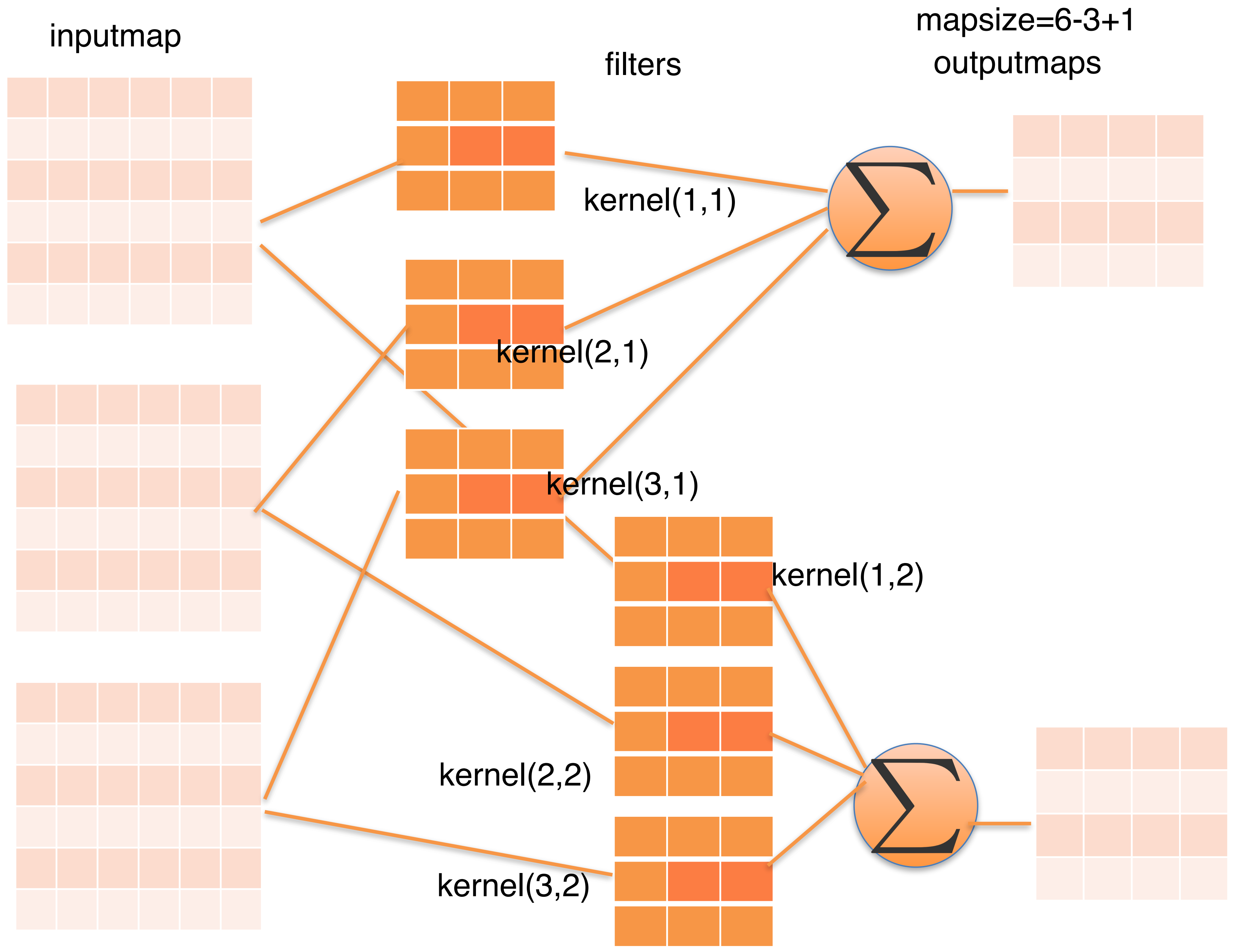
- **Figure2lInside a convolutional network.** The outputs(not the filters) of each layer (horizontally) of a typical convolutional network architecture applied to the image of a Samoyed dog (bottom left; and RGB (red, green, blue) inputs, bottom right). Each rectangular image is a feature map corresponding to the output for one of the learned features, detected at each of the image positions. Information flows bottom up, with lower-level features acting as oriented edge detectors, and a score is computed for each image class in output. ReLU, rectified linear unit.

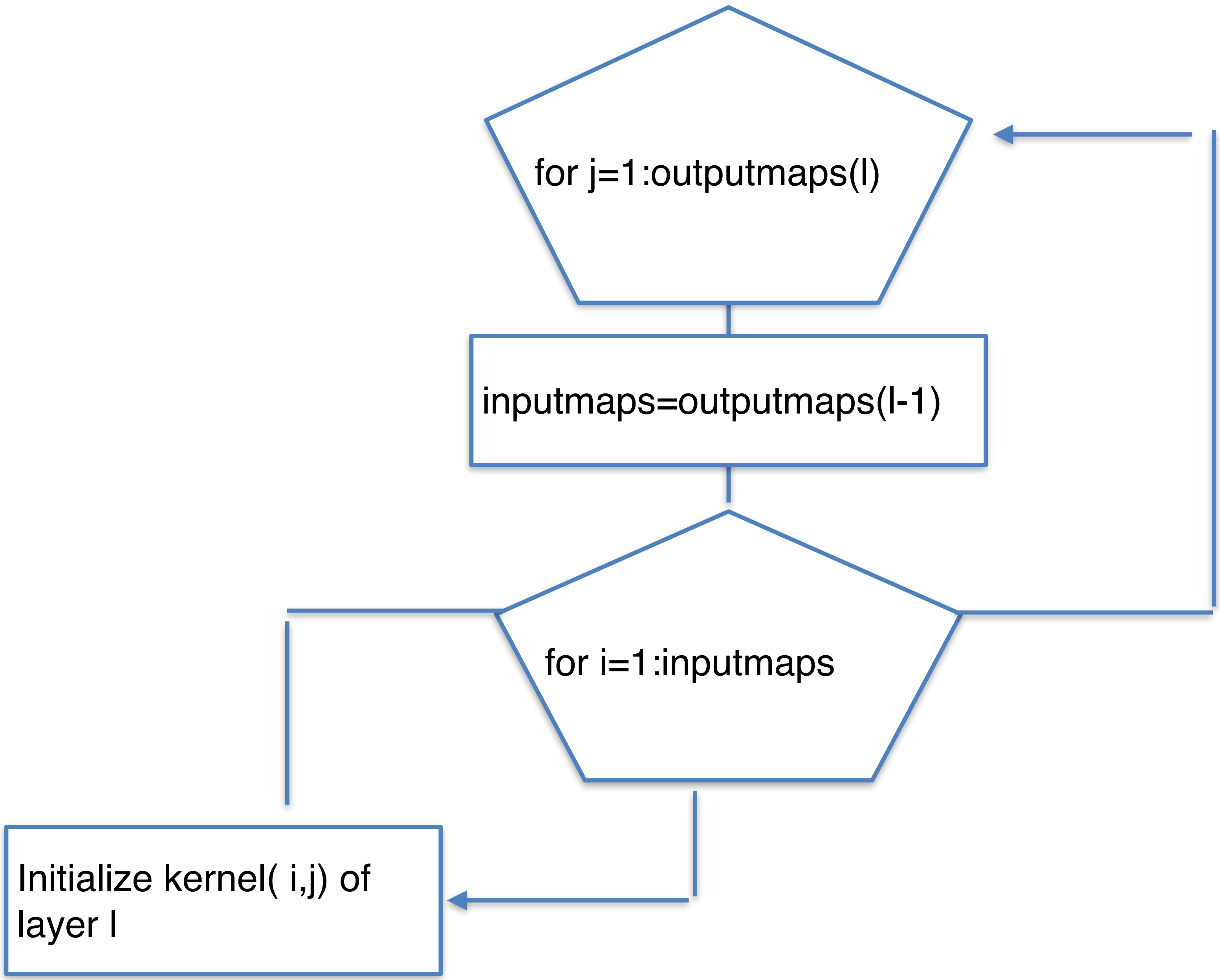
CNN for multiple arrays

- A colour image composed of three 2D arrays containing pixel intensities in the three colour channels.
- Many data modalities are in the form of multiple arrays: 1D for signals and sequences, including language; 2D for images or audio spectrograms; and 3D for video or volumetric images.

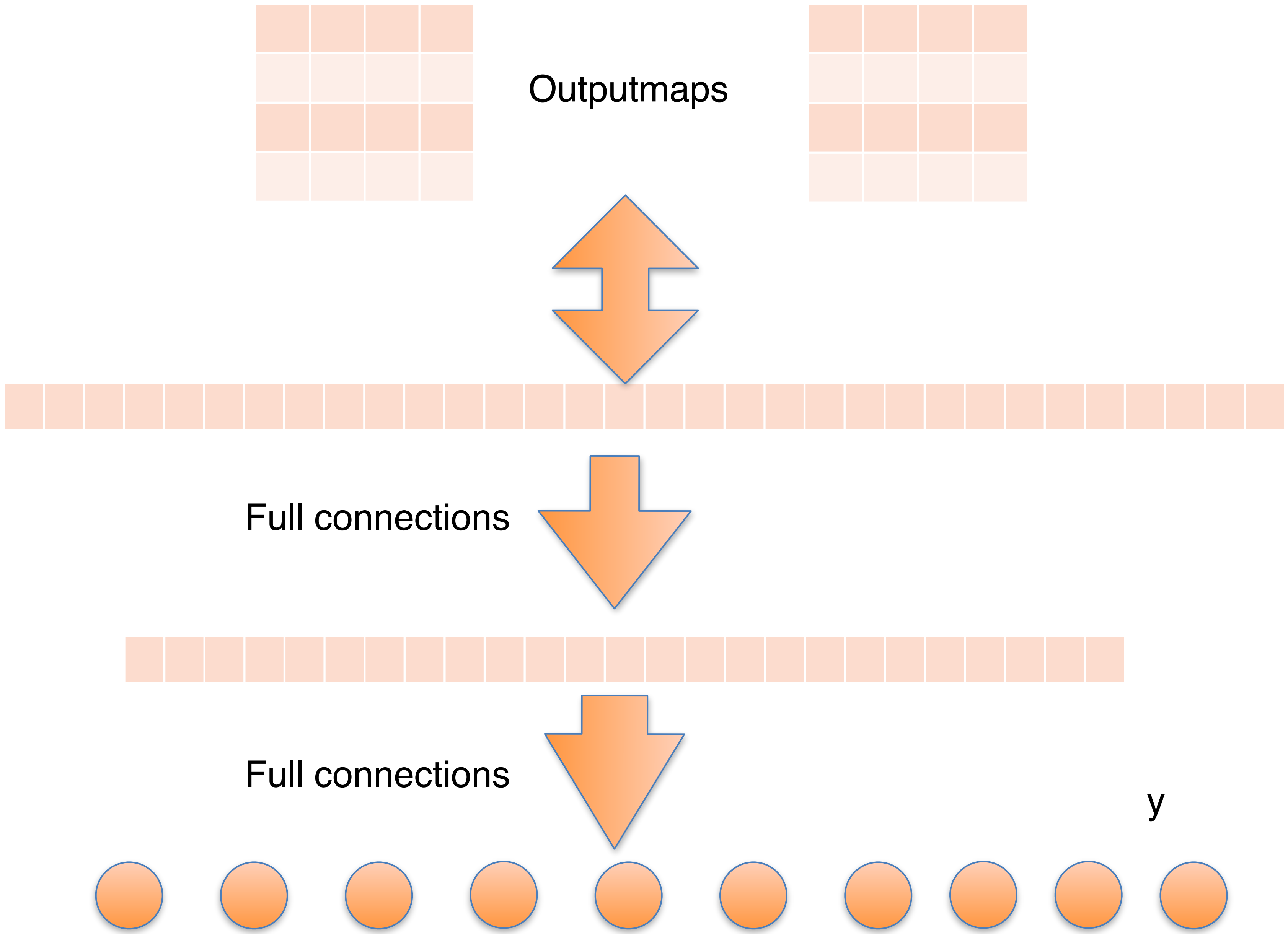
```matlab
    inputmaps = 1;
    mapsize = size(squeeze(x(:, :, 1)));

    for l = 1 : numel(net.layers)   %  layer
        if strcmp(net.layers{l}.type, 's')
            mapsize = mapsize / net.layers{l}.scale;
            assert(all(floor(mapsize)==mapsize), ['Layer ' num2str(l) ' size must be integer. Actual: ' num2str(mapsize)]);
            for j = 1 : inputmaps
                net.layers{l}.b{j} = 0;
            end
        end
        if strcmp(net.layers{l}.type, 'c')
            mapsize = mapsize - net.layers{l}.kernelsize + 1;
            fan_out = net.layers{l}.outputmaps * net.layers{l}.kernelsize ^ 2;
            for j = 1 : net.layers{l}.outputmaps  %  output map
                fan_in = inputmaps * net.layers{l}.kernelsize ^ 2;
                for i = 1 : inputmaps  %  input map
                    net.layers{l}.k{i}{j} = (rand(net.layers{l}.kernelsize) - 0.5) * 2 * sqrt(6 / (fan_in + fan_out));
                end
                net.layers{l}.b{j} = 0;
            end
            inputmaps = net.layers{l}.outputmaps;
        end
    end
```


mapsize=10-5+1
outputmaps
filters
inputmap


inputmap
filters
mapsize=6-3+1
outputmaps
kernel(1,1)
kernel(2,1)
kernel(3,1)
kernel(1,2)
kernel(2,2)
kernel(3,2)
Σ
Σ

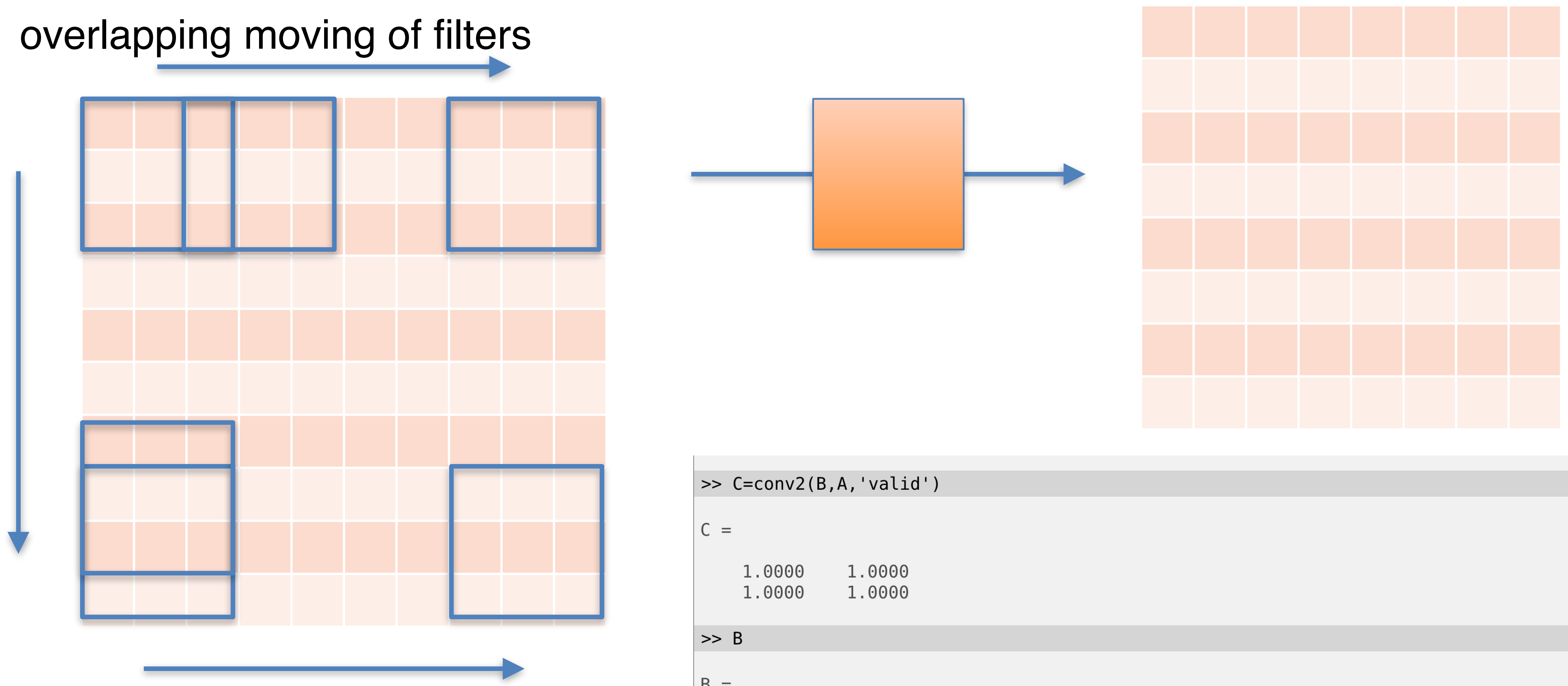

for j=1:outputmaps(l)
inputmaps=outputmaps(l-1)
for i=1:inputmaps
Initialize kernel(i,j) of layer l

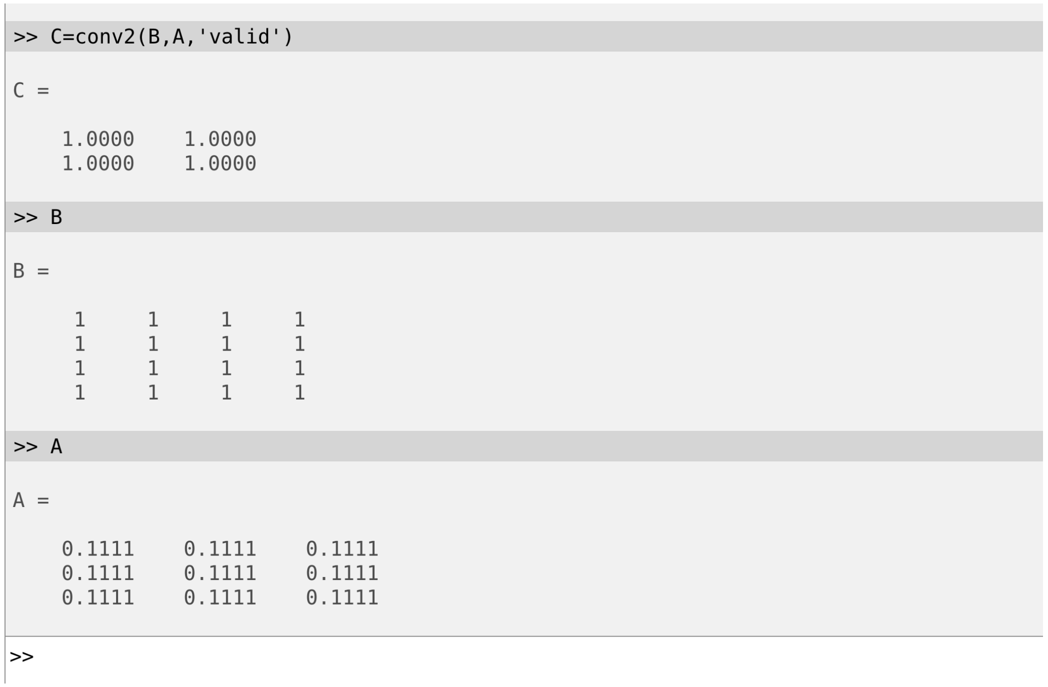
```matlab
    % 'onum' is the number of labels, that's why it is calculated using size(y, 1). If you have 20 labels so the output of the network will be 20 neurons.
    % 'fvnum' is the number of output neurons at the last layer, the layer just before the output layer.
    % 'ffb' is the biases of the output neurons.
    % 'ffW' is the weights between the last layer and the output neurons. Note that the last layer is fully connected to the output layer, that's why the size of the weights is (onum * fvnum)
    fvnum = prod(mapsize) * inputmaps;
    onum = size(y, 1);

    net.ffb = zeros(onum, 1);
    net.ffW = (rand(onum, fvnum) - 0.5) * 2 * sqrt(6 / (onum + fvnum));
end

```


Outputmaps
Full connections
Full connections
y

$$c3, s2, c2, s2$$

2D Convolution

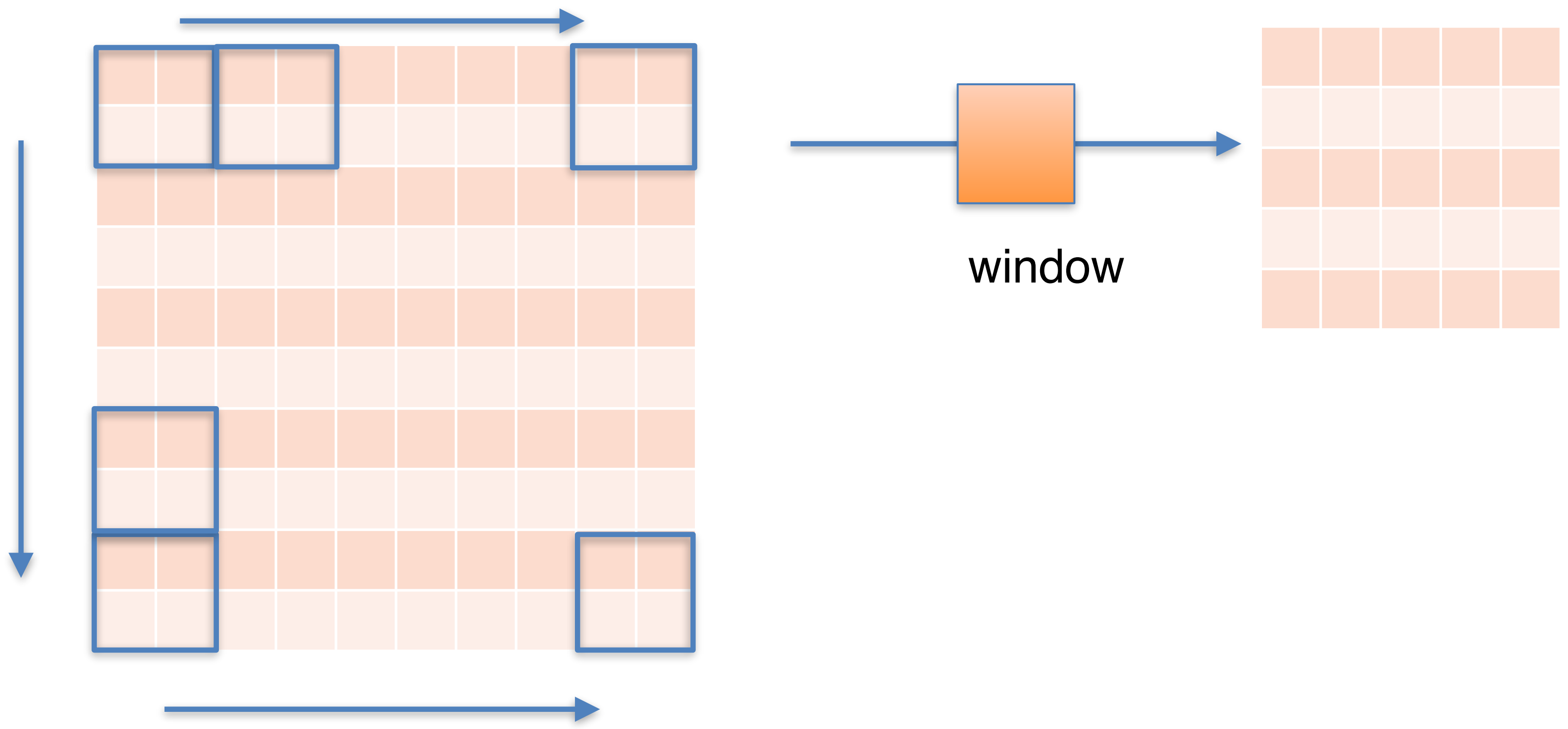


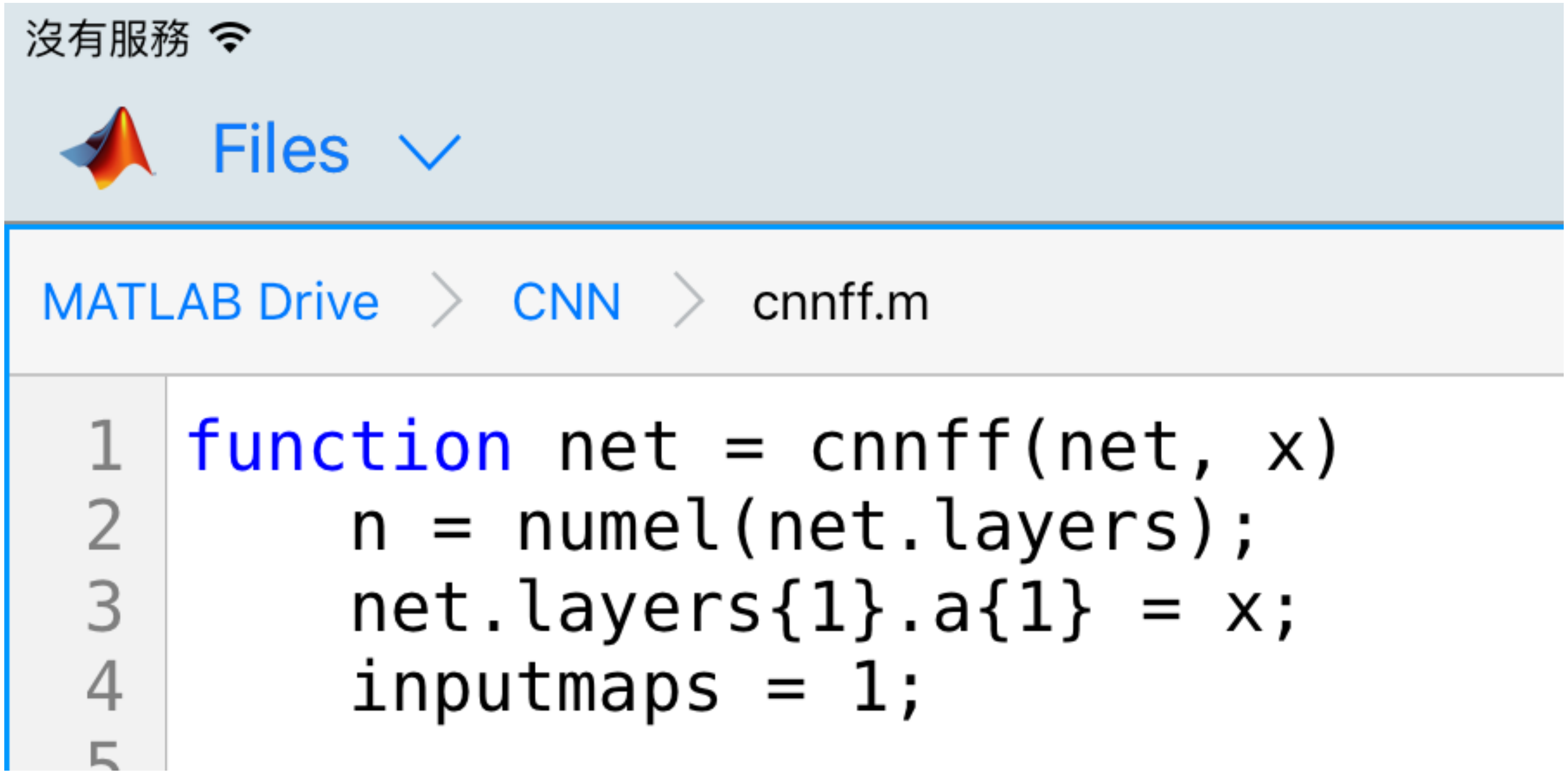


```
>> C=conv2(B,A,'valid')

C =

    1.0000    1.0000
    1.0000    1.0000

>> B

B =

     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1

>> A

A =

    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111

>>
```

Down Sampling

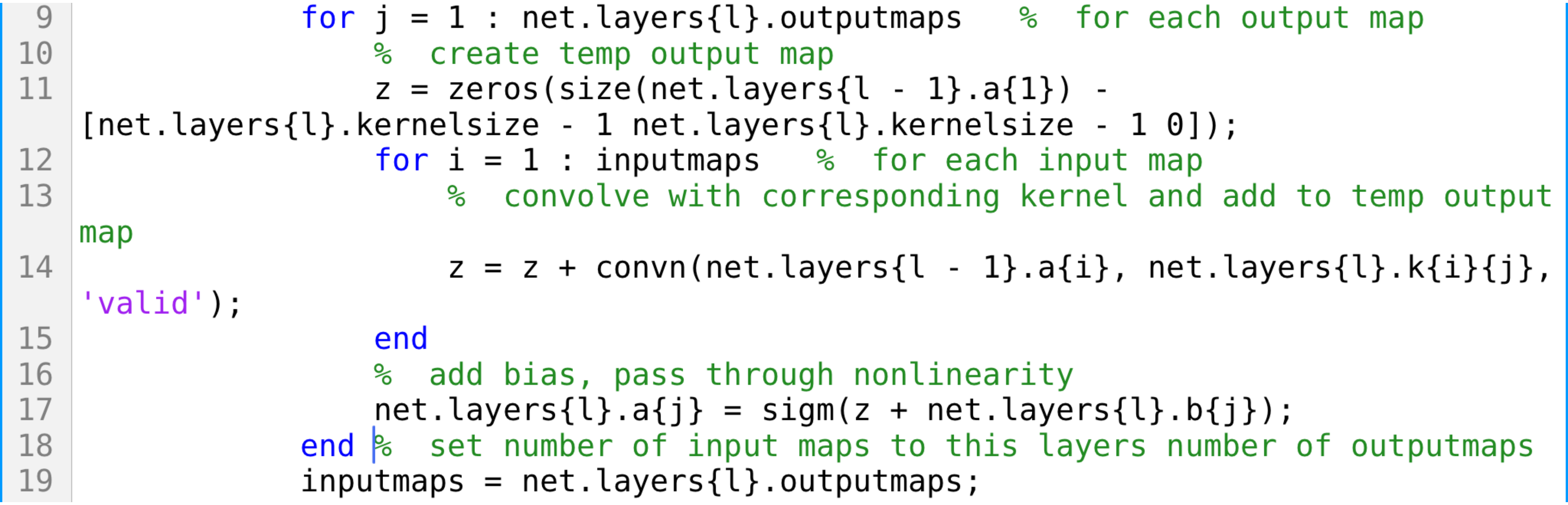
Non-overlapping moving sampling





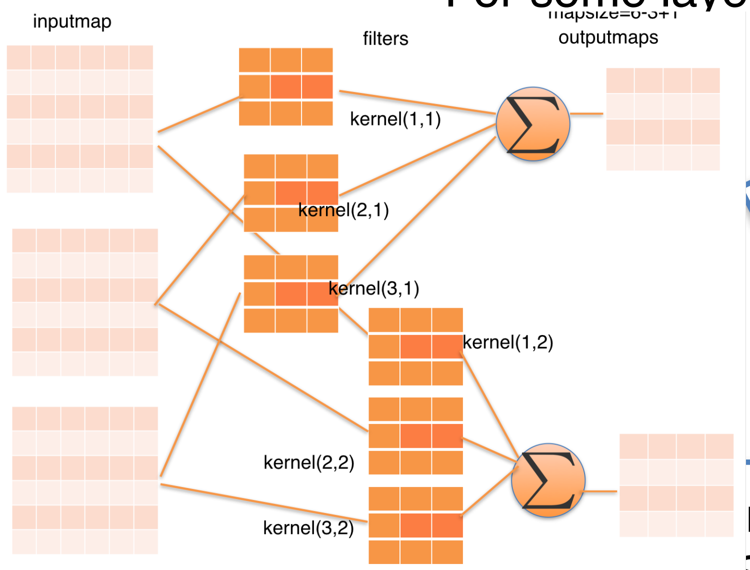
```
function net = cnnff(net, x)
    n = numel(net.layers);
    net.layers{1}.a{1} = x;
    inputmaps = 1;
```

```matlab
    for l = 2 : n   %  for each layer
        if strcmp(net.layers{l}.type, 'c')
            %  !!below can probably be handled by insane matrix operations
            for j = 1 : net.layers{l}.outputmaps   %  for each output map
                %  create temp output map
                z = zeros(size(net.layers{l - 1}.a{1}) - [net.layers{l}.kernelsize - 1 net.layers{l}.kernelsize - 1 0]);
                for i = 1 : inputmaps   %  for each input map
                    %  convolve with corresponding kernel and add to temp output map
                    z = z + convn(net.layers{l - 1}.a{i}, net.layers{l}.k{i}{j}, 'valid');
                end
                %  add bias, pass through nonlinearity
                net.layers{l}.a{j} = sigm(z + net.layers{l}.b{j});
            end %  set number of input maps to this layers number of outputmaps
            inputmaps = net.layers{l}.outputmaps;
        elseif strcmp(net.layers{l}.type, 's')%  downsample
            for j = 1 : inputmaps
                z = convn(net.layers{l - 1}.a{j}, ones(net.layers{l}.scale) / (net.layers{l}.scale ^ 2), 'valid');   %  !! replace with variable
                net.layers{l}.a{j} = z(1 : net.layers{l}.scale : end, 1 : net.layers{l}.scale : end, :);
            end
        end
    end
```

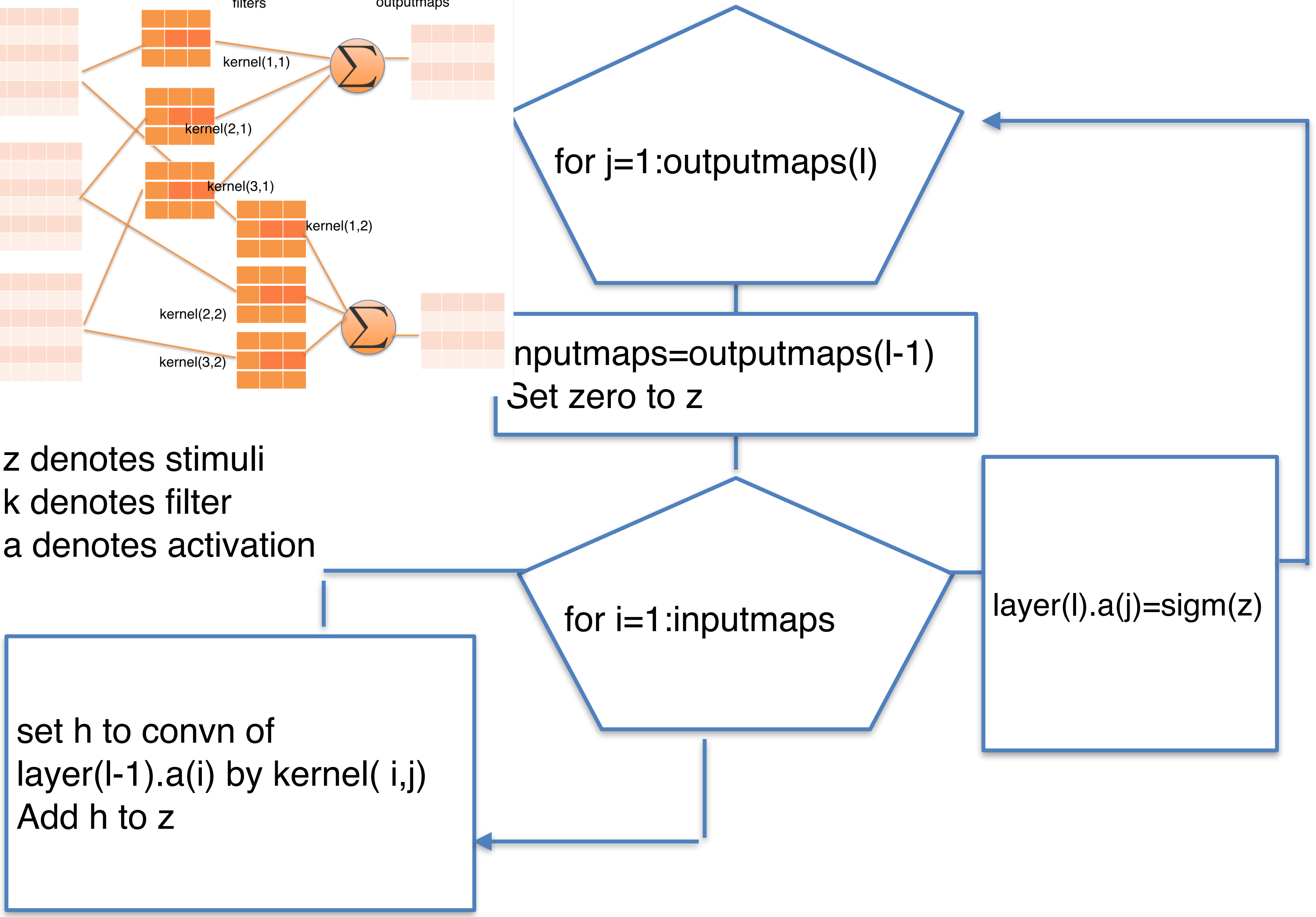


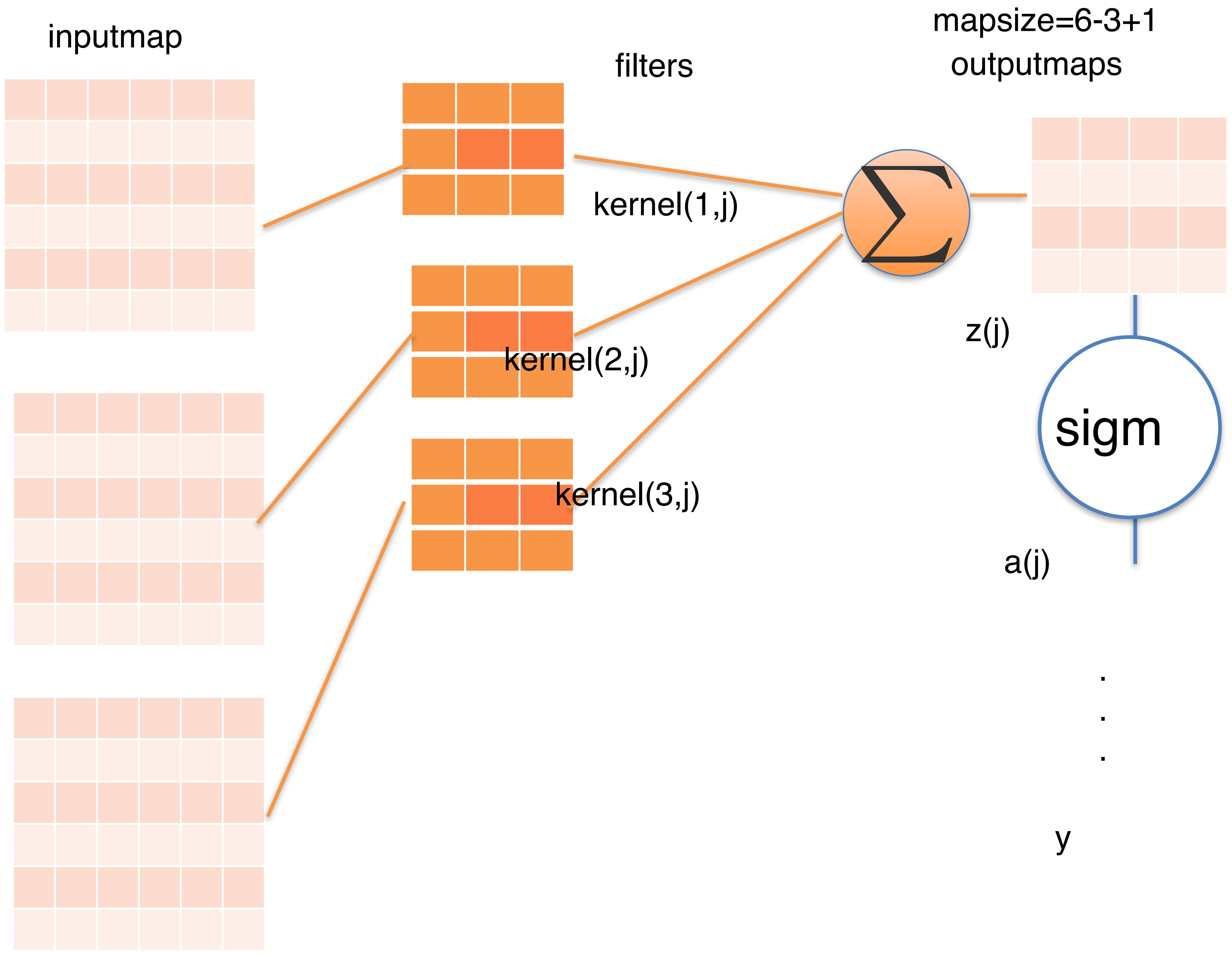
```
            for j = 1 : net.layers{l}.outputmaps   %  for each output map
                %  create temp output map
                z = zeros(size(net.layers{l - 1}.a{1}) - [net.layers{l}.kernelsize - 1 net.layers{l}.kernelsize - 1 0]);
                for i = 1 : inputmaps   %  for each input map
                    %  convolve with corresponding kernel and add to temp output map
                    z = z + convn(net.layers{l - 1}.a{i}, net.layers{l}.k{i}{j}, 'valid');
                end
                %  add bias, pass through nonlinearity
                net.layers{l}.a{j} = sigm(z + net.layers{l}.b{j});
            end %  set number of input maps to this layers number of outputmaps
            inputmaps = net.layers{l}.outputmaps;
```

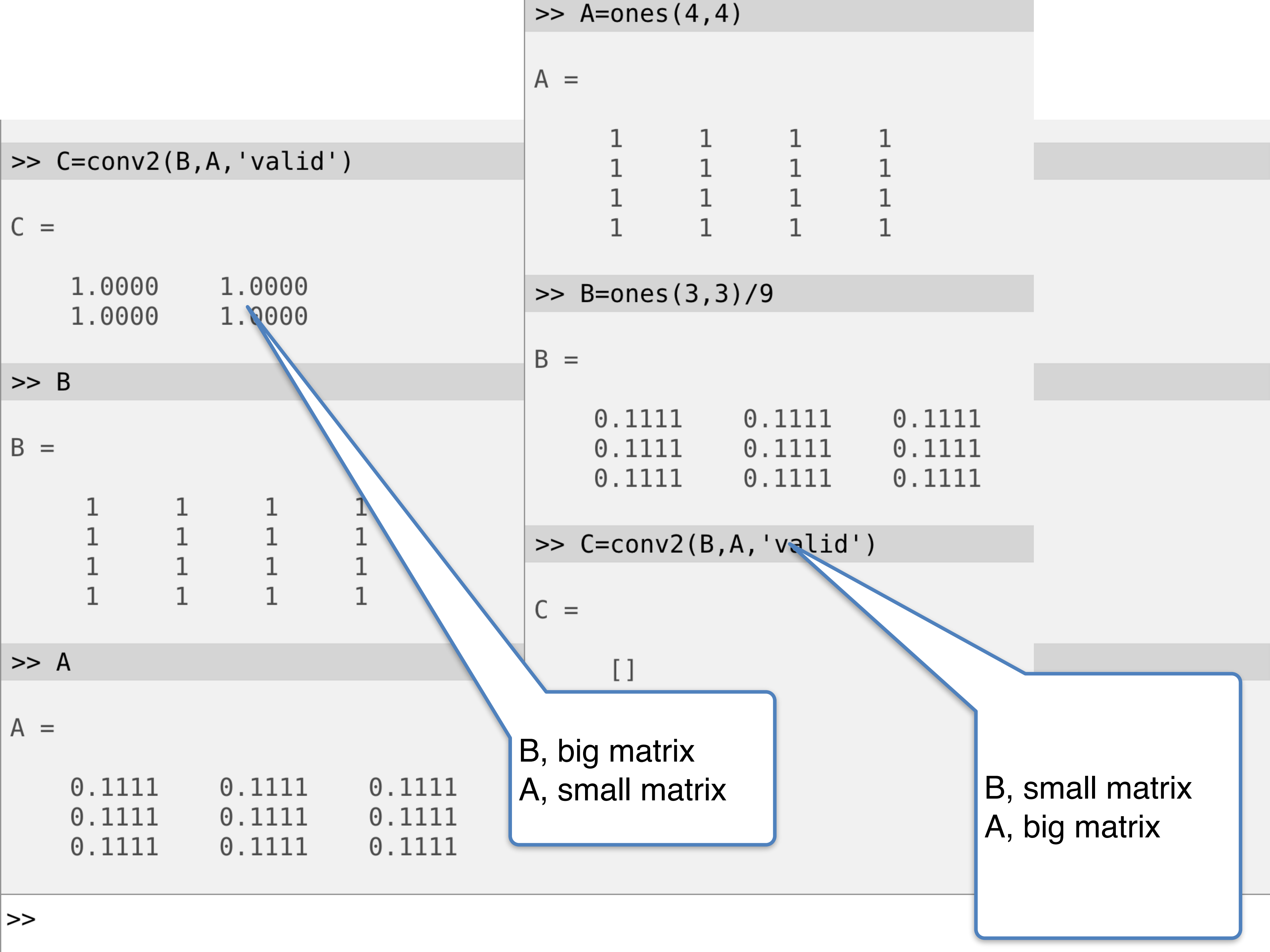
For some layer l



z denotes stimuli
k denotes filter
a denotes activation




inputmap
filters
mapsize=6-3+1
outputmaps
kernel(1,j)
kernel(2,j)
kernel(3,j)
z(j)
sigm
a(j)
.
.
.
y



```
>> C=conv2(B,A,'valid')

C =

    1.0000    1.0000
    1.0000    1.0000

>> B

B =

     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1

>> A

A =

    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111

>>
```

```
>> A=ones(4,4)

A =

     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1

>> B=ones(3,3)/9

B =

    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111

>> C=conv2(B,A,'valid')

C =

     []
```

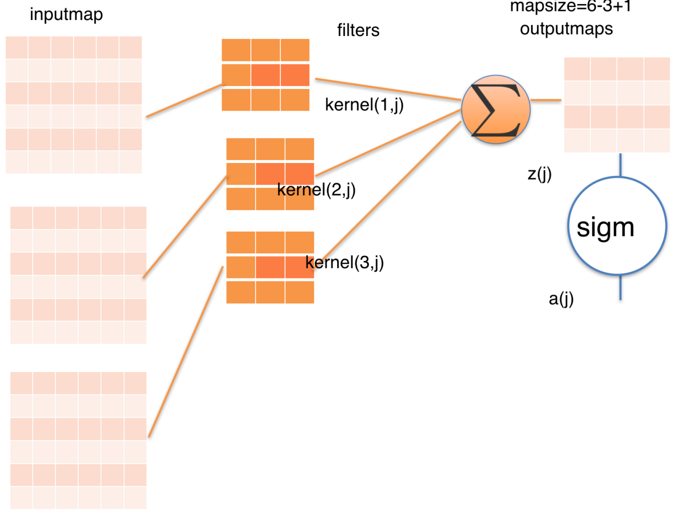
```
>> B=ones(3,3)/9

B =

    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111

>> C=conv2(B,A,'valid')

C =

     []

>> C=conv2(B,A,'full')

C =

    0.1111    0.2222    0.3333    0.3333    0.2222    0.1111
    0.2222    0.4444    0.6667    0.6667    0.4444    0.2222
    0.3333    0.6667    1.0000    1.0000    0.6667    0.3333
    0.3333    0.6667    1.0000    1.0000    0.6667    0.3333
    0.2222    0.4444    0.6667    0.6667    0.4444    0.2222
    0.1111    0.2222    0.3333    0.3333    0.2222    0.1111
```

y denotes output of CNN

$$\frac{dy}{da(j)}, \frac{dy}{dz(j)}, \frac{dy}{dk(i,j)}$$

1. Derivative of output with respect to activation ?
2. Derivative of output with respect to stimuli?
3. Derivative of output with respect to kernels?

for some layer l



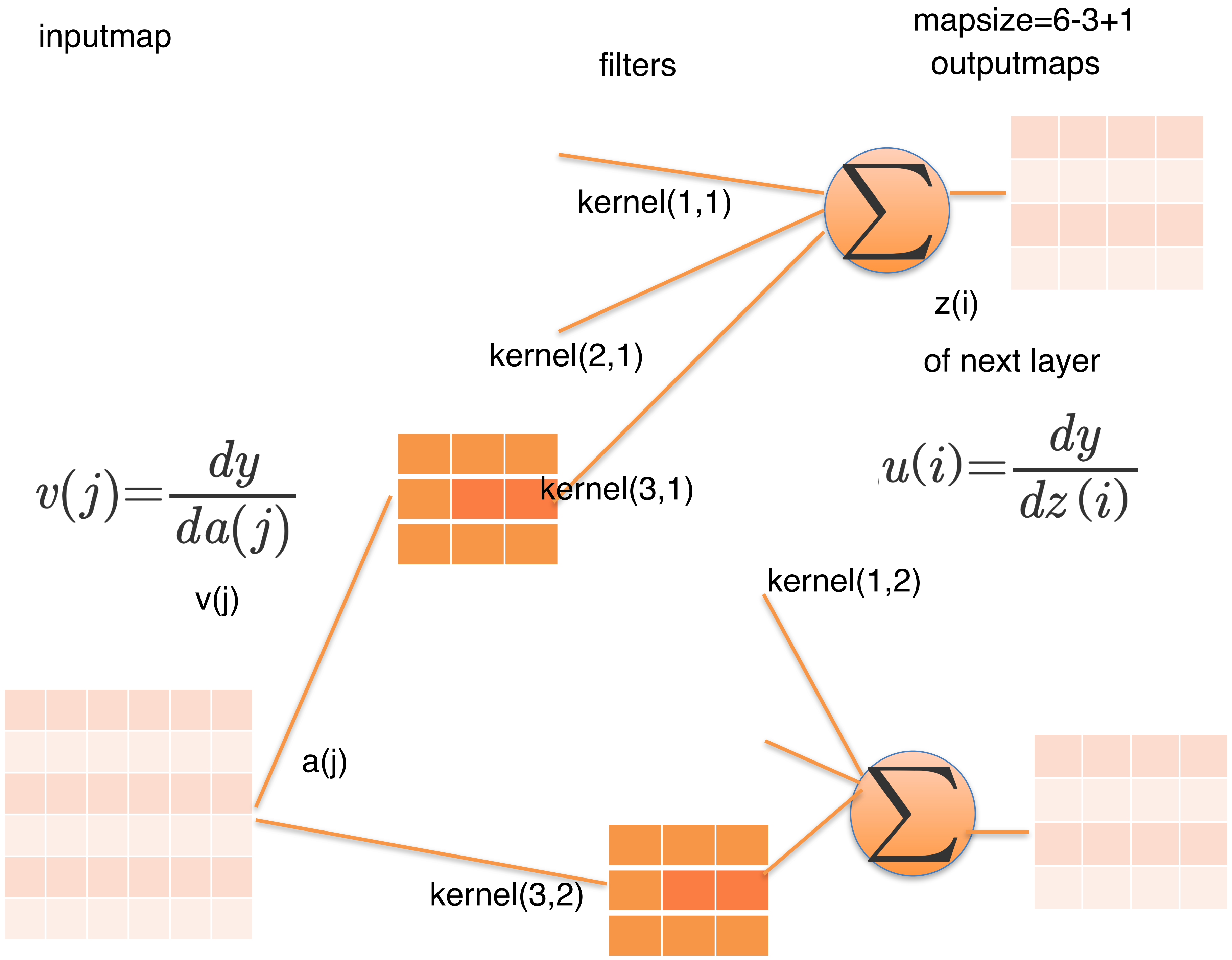
$$a(j)=sigm(z(j))$$

$$\frac{dy}{dz(j)}=\frac{dy}{da(j)}\frac{da(j)}{dz(j)}$$

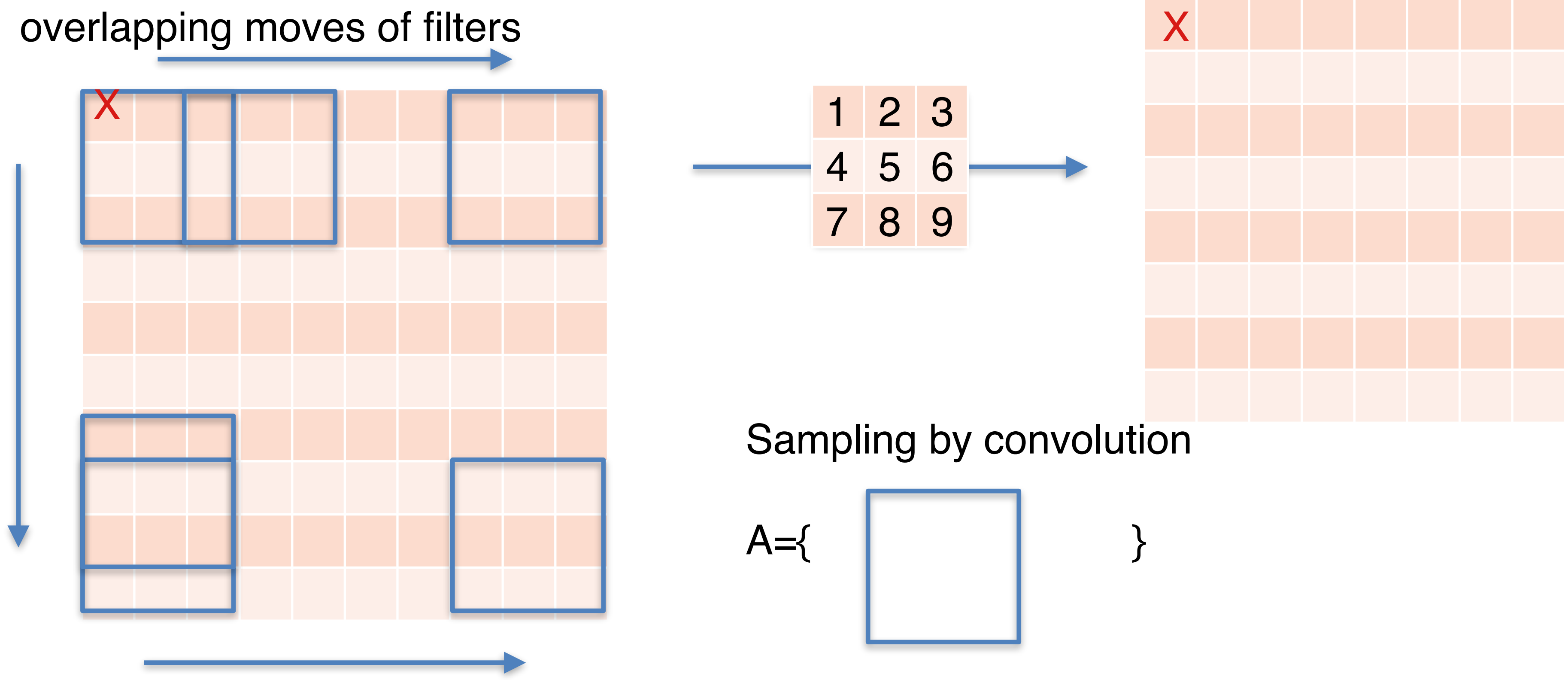
u(j)

The first term is denoted by v(j)

derivative of sigmoid


inputmap
filters
mapsize=6-3+1
outputmaps
kernel(1,1)
kernel(2,1)
kernel(3,1)
z(i)
of next layer
$v(j)=\frac{dy}{da(j)}$
$u(i)=\frac{dy}{dz(i)}$
v(j)
kernel(1,2)
a(j)
kernel(3,2)

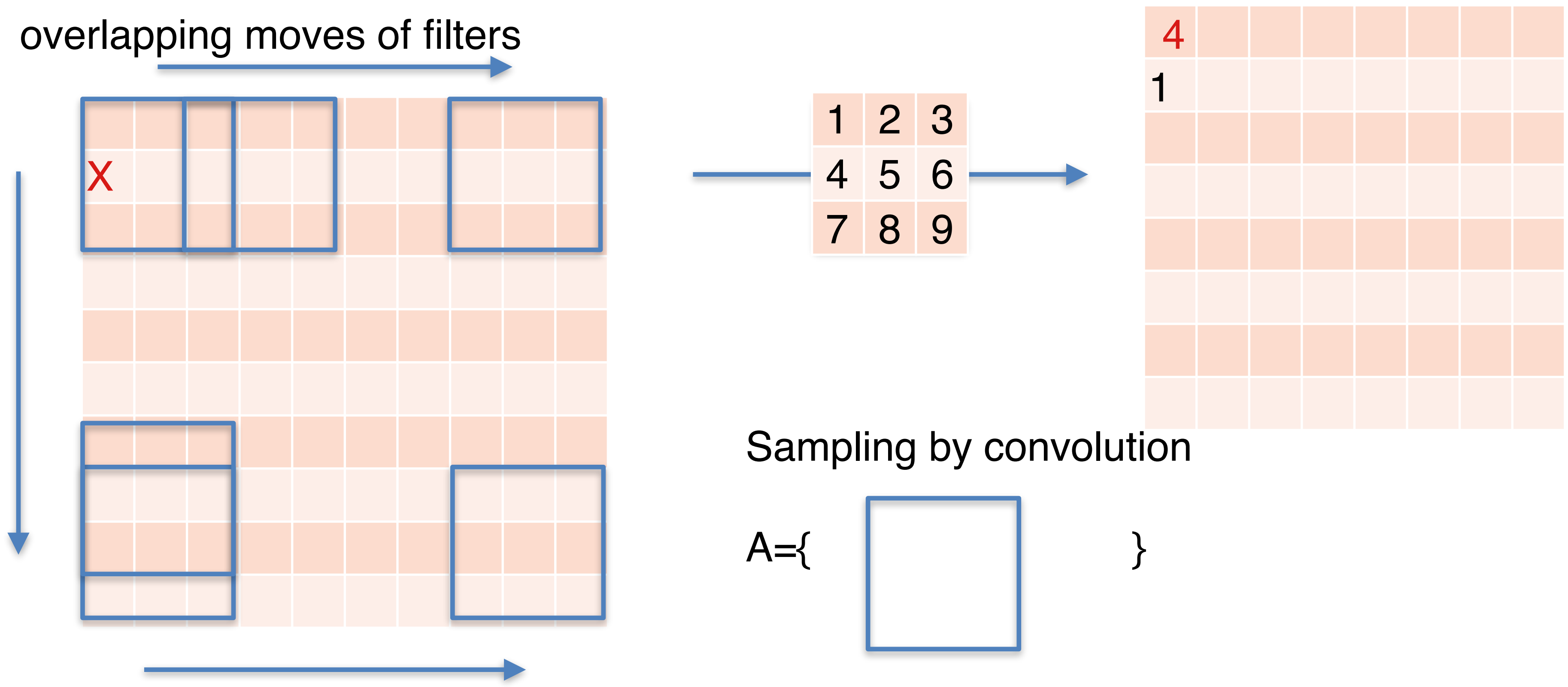
2D Convolution



Sampling by convolution

Position x on the activation map contributes to only position X of the resulting map through k(1)

2D Convolution

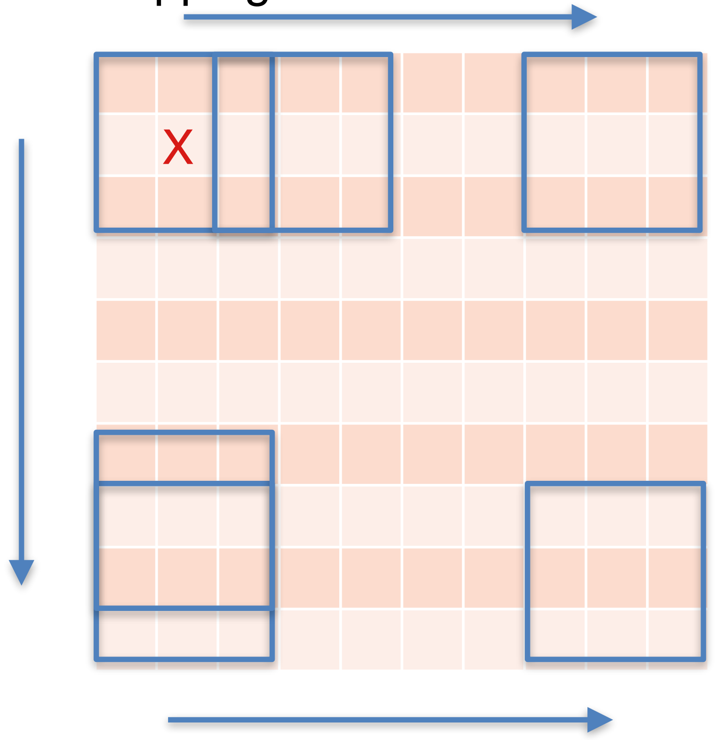


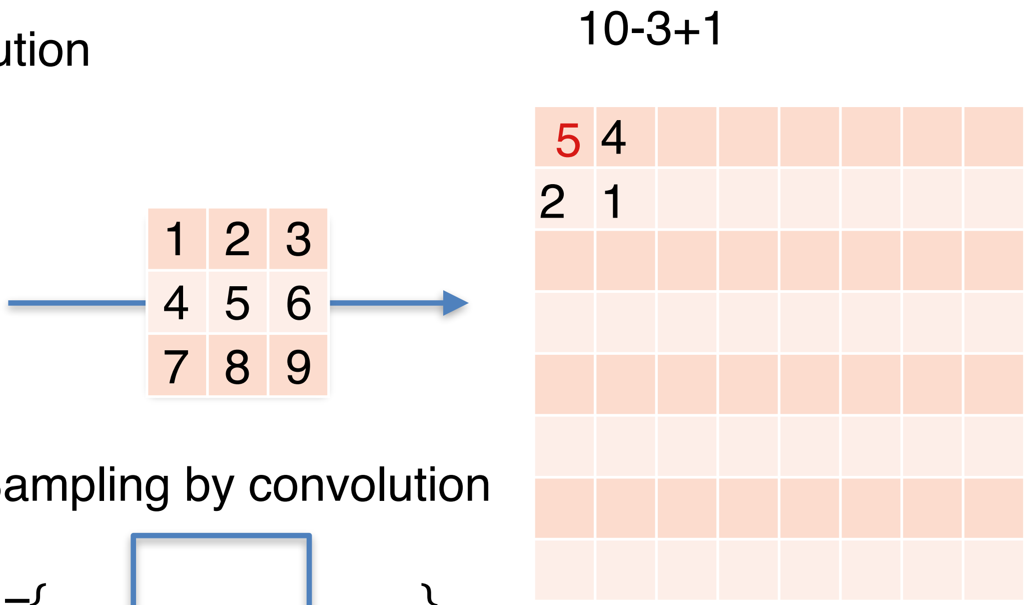
Sampling by convolution

Position x on the activation map contributes to only positions with label 4 and 1 through k(4) and k(1) respectively

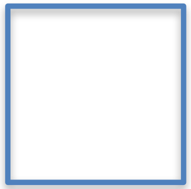
2D Convolution

overlapping moves of filters





Sampling by convolution

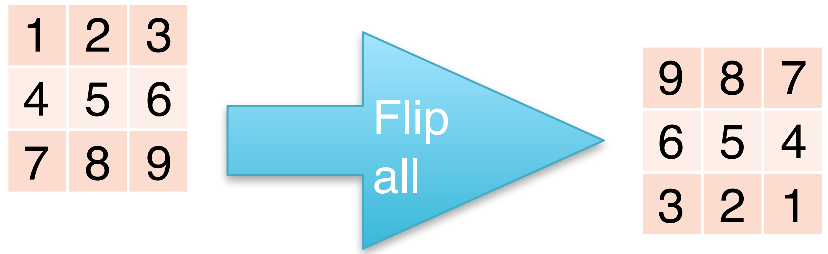
A={  }

.There are four validly convoluted samples that contain position X

.Position x on the activation map contributes to only positions labeled 1,2,4 and 5 through k(

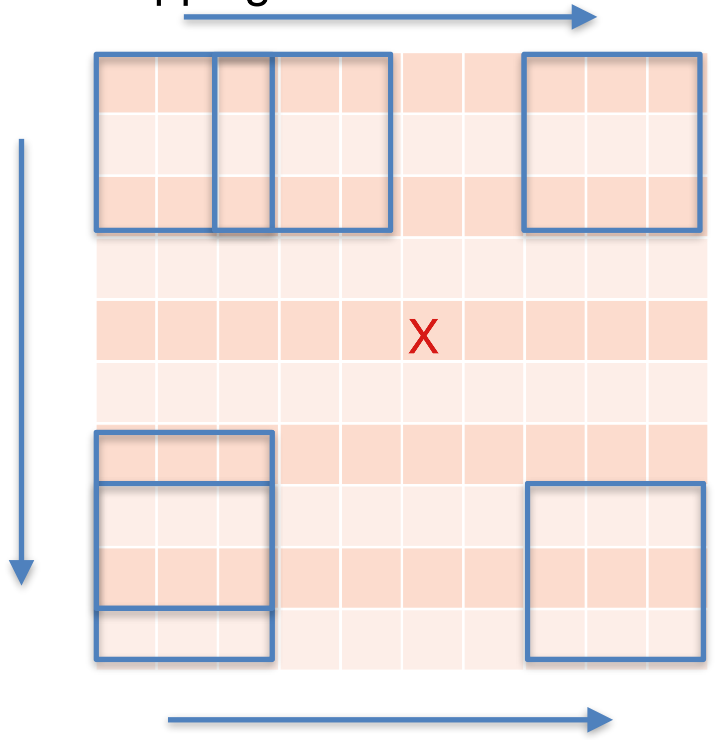
5	4
2	1

)

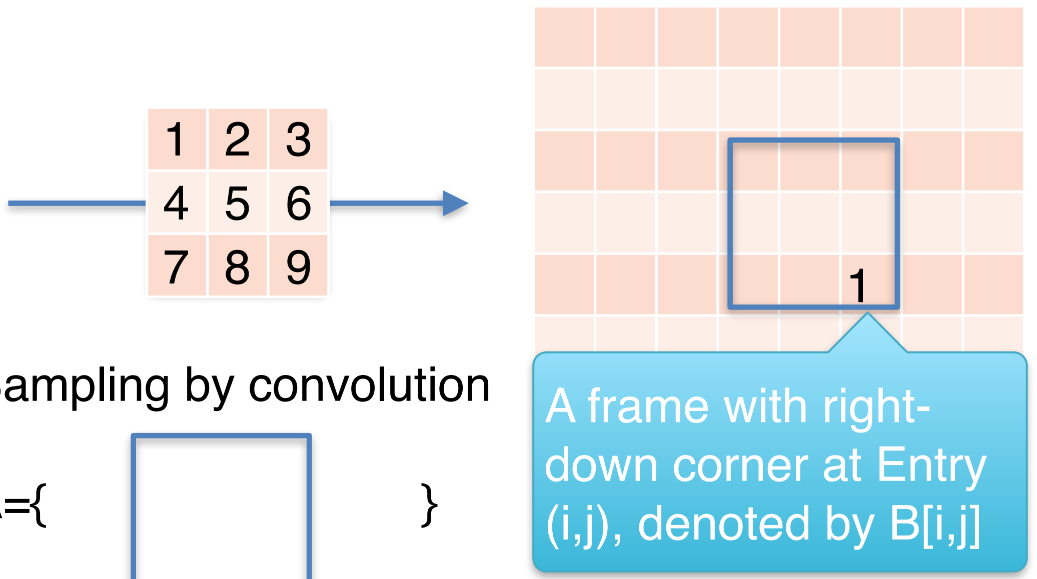


2D Convolution

overlapping moves of filters



10-3+1


1 2 3
4 5 6
7 8 9
1
A frame with right-down corner at Entry (i,j), denoted by B[i,j]

Sampling by convolution

A={ $\boxed{}$ }

.There are 9 validly convoluted samples that contain position X at entry (i,j)

$$\begin{matrix}1&2&3\\4&5&6\\7&8&9\end{matrix} \xrightarrow{\text{Flip all}} \begin{matrix}9&8&7\\6&5&4\\3&2&1\end{matrix}$$

.Position x on the activation map contributes to positions labeled with 1-9 through k($\begin{matrix}9&8&7\\6&5&4\\3&2&1\end{matrix}$)

Indices of layers are omitted. It is observed that a[i,j] contributes to the frame B[i,j] through flipall(K)

9	8	7
6	5	4
3	2	1

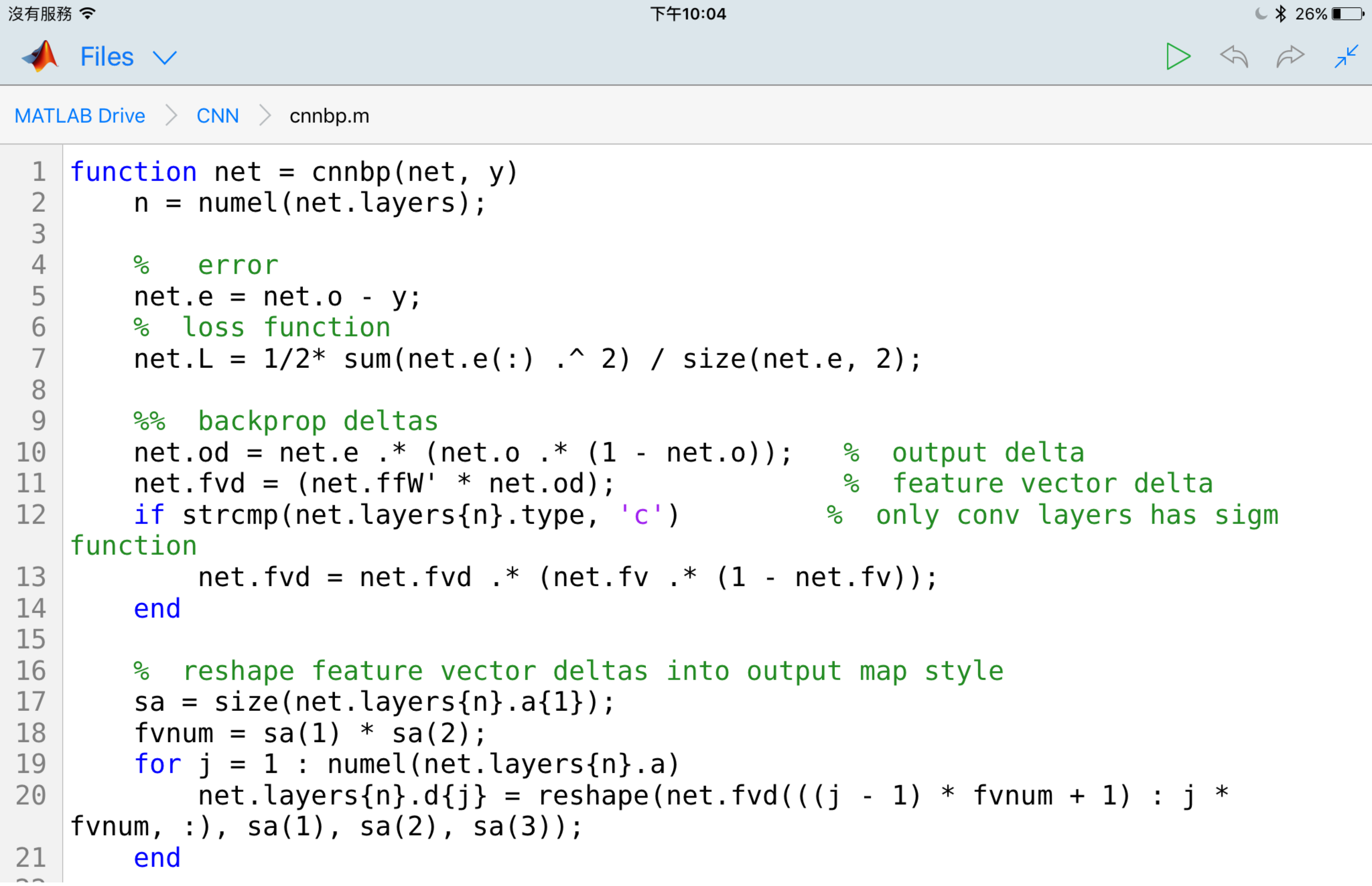
$$\frac{dy}{da[i,j]} = \frac{dy}{dB[i,j]} \frac{dB(ij)}{da[i,j]}$$

$$= \frac{dy}{dB[i,j]} flipall(k)$$

The gradient of output with respect to frame B[i,j], which is denoted by V[i,j]

EQ2

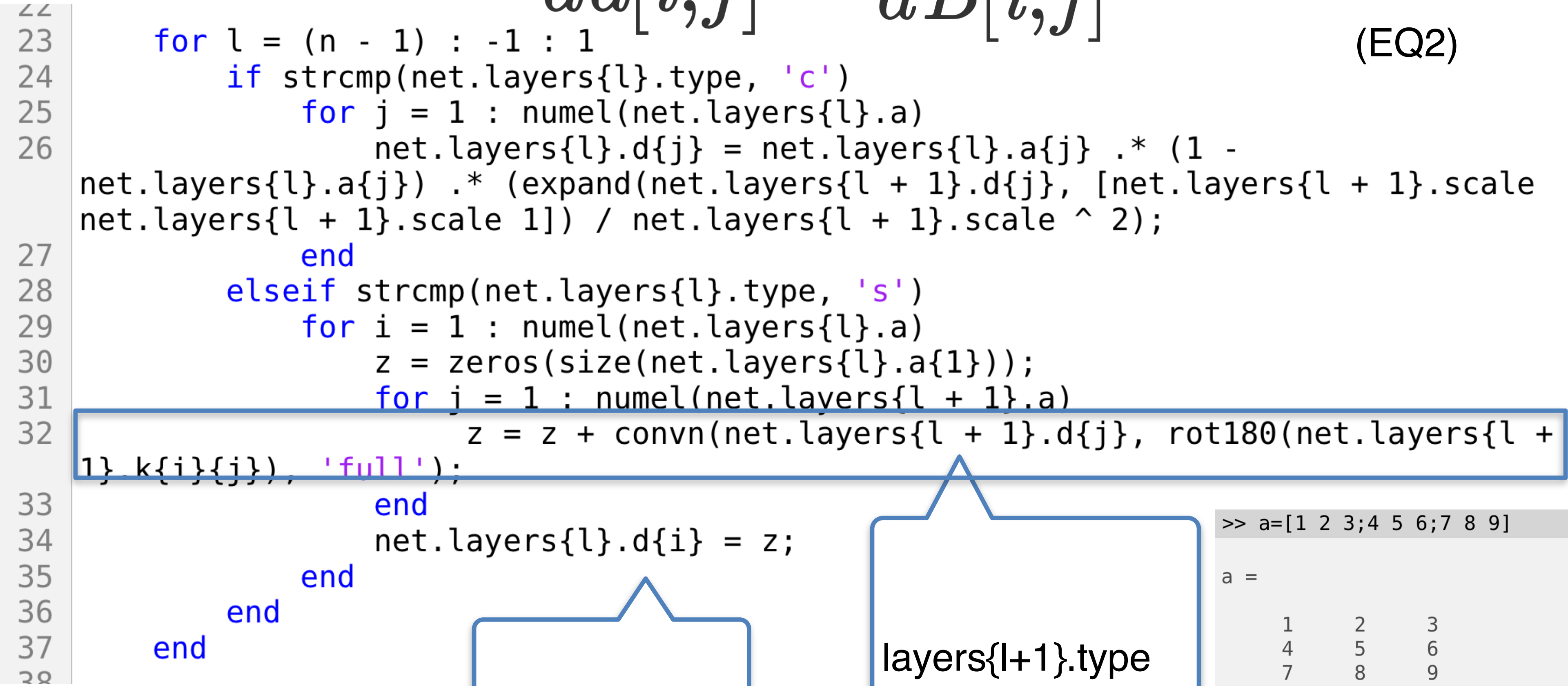
```

    %  concatenate all end layer feature maps into vector
    net.fv = [];
    for j = 1 : numel(net.layers{n}.a)
        sa = size(net.layers{n}.a{j});
        net.fv = [net.fv; reshape(net.layers{n}.a{j}, sa(1) * sa(2), sa(3))];
    end
    %  feedforward into output perceptrons
    net.o = sigm(net.ffW * net.fv + repmat(net.ffb, 1, size(net.fv, 2)));

end
```



```matlab
function net = cnnbp(net, y)
    n = numel(net.layers);

    %   error
    net.e = net.o - y;
    %  loss function
    net.L = 1/2* sum(net.e(:) .^ 2) / size(net.e, 2);

    %%  backprop deltas
    net.od = net.e .* (net.o .* (1 - net.o));   %  output delta
    net.fvd = (net.ffW' * net.od);              %  feature vector delta
    if strcmp(net.layers{n}.type, 'c')         %  only conv layers has sigm function
        net.fvd = net.fvd .* (net.fv .* (1 - net.fv));
    end

    %  reshape feature vector deltas into output map style
    sa = size(net.layers{n}.a{1});
    fvnum = sa(1) * sa(2);
    for j = 1 : numel(net.layers{n}.a)
        net.layers{n}.d{j} = reshape(net.fvd(((j - 1) * fvnum + 1) : j * fvnum, :), sa(1), sa(2), sa(3));
    end
```

$$\frac{dy}{da[i,j]} = \frac{dy}{dB[i,j]} flipall(k)$$

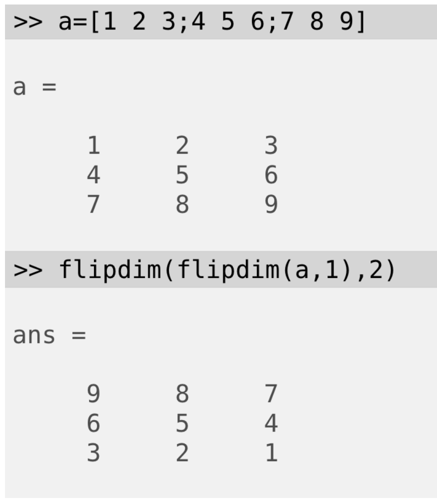
(EQ2)



```
    for l = (n - 1) : -1 : 1
        if strcmp(net.layers{l}.type, 'c')
            for j = 1 : numel(net.layers{l}.a)
                net.layers{l}.d{j} = net.layers{l}.a{j} .* (1 - net.layers{l}.a{j}) .* (expand(net.layers{l + 1}.d{j}, [net.layers{l + 1}.scale net.layers{l + 1}.scale 1]) / net.layers{l + 1}.scale ^ 2);
            end
        elseif strcmp(net.layers{l}.type, 's')
            for i = 1 : numel(net.layers{l}.a)
                z = zeros(size(net.layers{l}.a{1}));
                for j = 1 : numel(net.layers{l + 1}.a)
                     z = z + convn(net.layers{l + 1}.d{j}, rot180(net.layers{l + 1}.k{i}{j}), 'full');
                end
                net.layers{l}.d{i} = z;
            end
        end
    end
```

Size?

layers{l+1}.type
'c'



```
>> a=[1 2 3;4 5 6;7 8 9]

a =

     1     2     3
     4     5     6
     7     8     9

>> flipdim(flipdim(a,1),2)

ans =

     9     8     7
     6     5     4
     3     2     1
```

Size=10-3+1

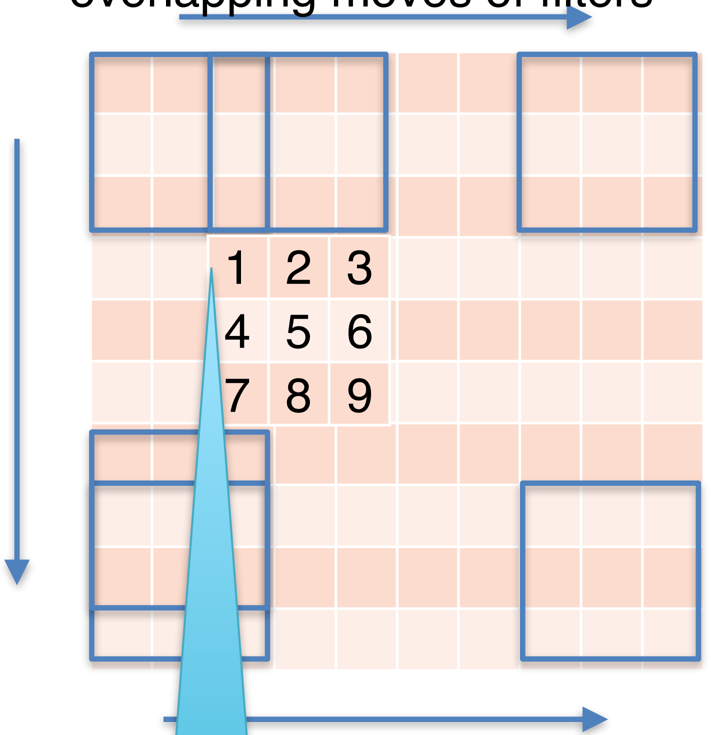
$$a\{n\}$$

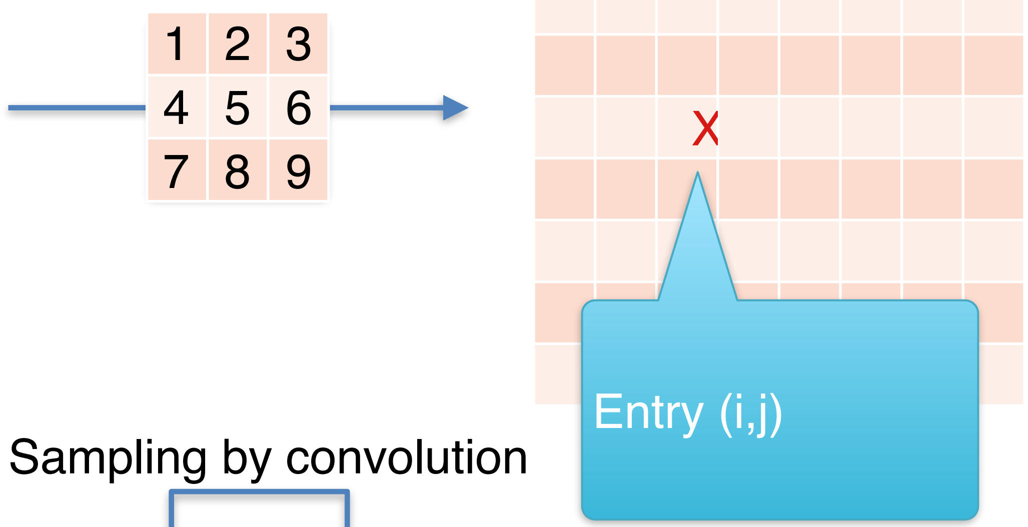
2D Convolution

$$K\{n\}$$

$$h\{n+1\}$$

overlapping moves of filters





Entry (i,j)

Sampling by convolution

A={ $\boxed{}$ }

$$h\{n+1\}[i,j]$$

A frame with it left-upper corner at Entry (i,j) is weighted by the kernel

$$h\{n+1\}=a\{n\}^{*}k\{n\}$$

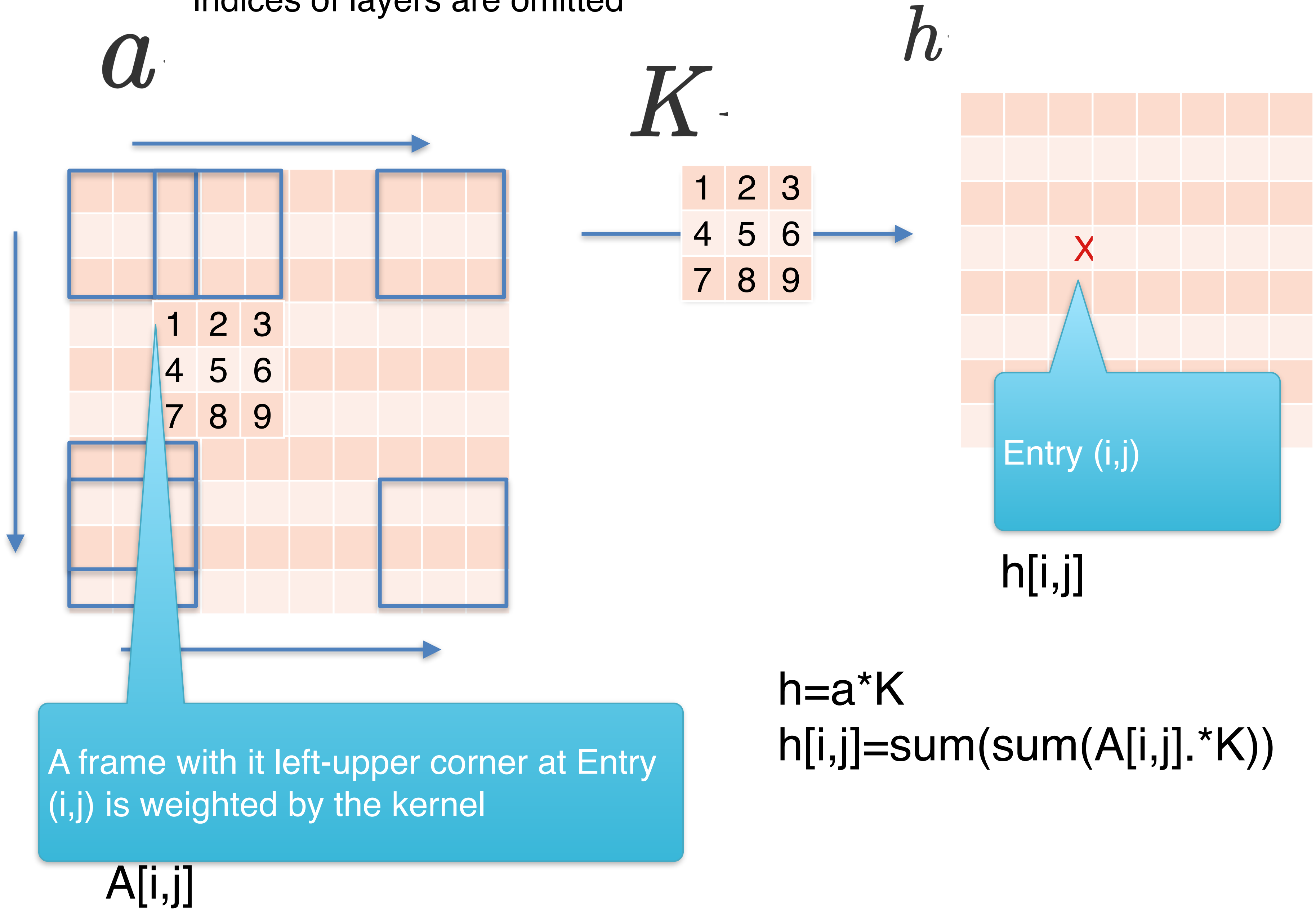
$$a\{n\}\_s[i,j]$$

$$h\{n+1\}=a\{n\}^{*}k\{n\}$$

$$h\{n+1\}[i,j]=a\{n\}\_s[i,j]*k\{n\}$$

$$\frac{dy}{dk\{n\}}=\sum_{i,j}\frac{dy}{dh\{n+1\}[i,j]}\,dh\frac{\{n+1\}}{dk\{n\}}[i,j]$$

Indices of layers are omitted


a
K
h
1 2 3
4 5 6
7 8 9
X
Entry (i,j)
h[i,j]
A frame with it left-upper corner at Entry (i,j) is weighted by the kernel
A[i,j]

h=a*K

h[i,j]=sum(sum(A[i,j].*K))

```
h=a*K
h[i,j]=sum(sum(A[i,j].*K))
```

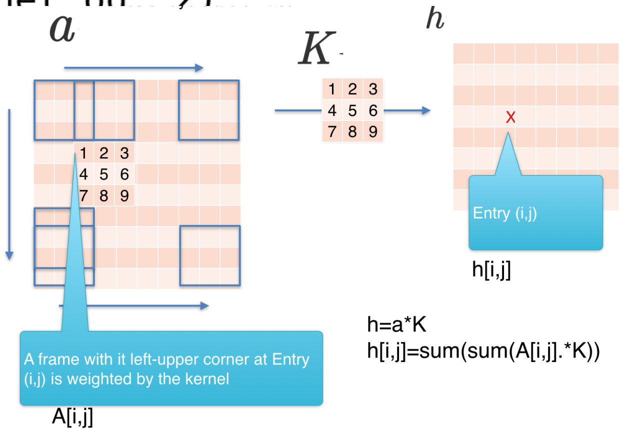
$$\frac{dy}{dK} = \sum_{ij} \frac{dy}{dh[i,j]} \frac{dh[i,j]}{dK}$$

$$= \sum_{ij} v[i,j] A[i,j] \qquad \text{(EQ1)}$$

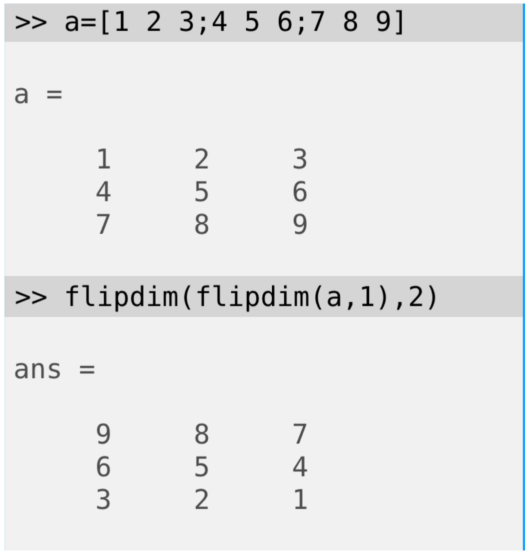
$$\frac{dy}{dK} = \sum_{ij} v[i,j]A[i,j] \quad \text{(EQ1)}$$

```
    %%  calc gradients
    for l = 2 : n
        if strcmp(net.layers{l}.type, 'c')
            for j = 1 : numel(net.layers{l}.a)
                for i = 1 : numel(net.layers{l - 1}.a)
                    net.layers{l}.dk{i}{j} = convn(flipall(net.layers{l - 1}.a{i}), net.layers{l}.d{j}, 'valid') / size(net.layers{l}.d{j}, 3);
                end
                net.layers{l}.db{j} = sum(net.layers{l}.d{j}(:)) / size(net.layers{l}.d{j}, 3);
            end
        end
    end
    net.dffW = net.od * (net.fv)' / size(net.od, 2);
    net.dffb = mean(net.od, 2);

    function X = rot
        X = flipdim(
    end
end
```

Size of dk?


a
K
h
1 2 3
4 5 6
7 8 9
X
Entry (i,j)
h[i,j]
A frame with it left-upper corner at Entry (i,j) is weighted by the kernel
A[i,j]

h=a*K
h[i,j]=sum(sum(A[i,j].*K))



```
>> a=[1 2 3;4 5 6;7 8 9]

a =

     1     2     3
     4     5     6
     7     8     9

>> flipdim(flipdim(a,1),2)

ans =

     9     8     7
     6     5     4
     3     2     1
```

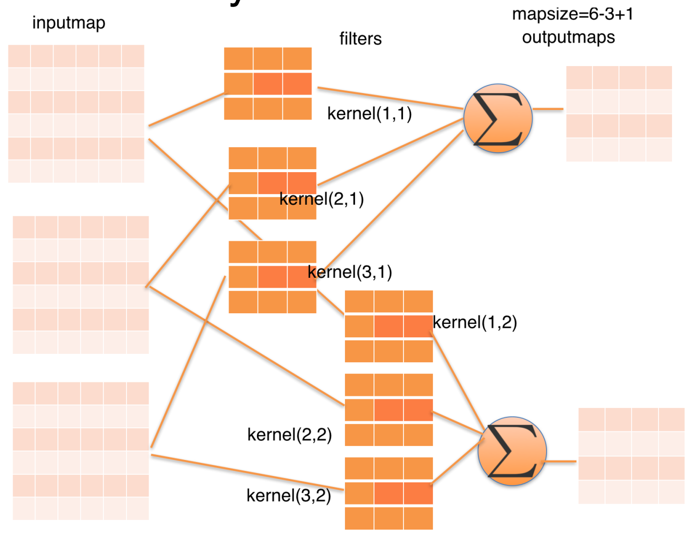
Four key ideas behind ConvNets

- local connections
- shared weights
- Pooling
- the use of many layers

Architecture

- A series of stages
- Units in a convolu- tional layer are organized in feature maps, within which each unit is connected to local patches in the feature maps of the previous layer through a set of weights called a filter bank.
- The result of this local weighted sum is then passed through a non-linearity such as a ReLU
- All units in a feature map share the same filter bank. Differ- ent feature maps in a layer use different filter banks

- Convolutional layers
  - the role of the convolutional layer is to detect local con- junctions of features from the previous layer
- Pooling layers.
  - the role of the pooling layer is to merge semantically similar features into one



- if a motif can appear in one part of the image, it could appear anywhere, hence the idea of units at different locations sharing the same weights and detecting the same pattern in different parts of the array
- Mathemati- cally, the filtering operation performed by a feature map is a discrete convolution, hence the name.

- in array data such as images, local groups of values are often highly correlated, forming distinctive local motifs that are easily detected.
- the local statistics of images and other signals are invariant to location

- A typical pooling unit computes the maximum of a local patch of units in one feature map (or in a few feature maps). Neighbouring pooling units take input from patches that are shifted by more than one row or column, thereby reducing the dimension of the representation and creating an invariance to small shifts and dis- tortions.

- Two or three stages of convolution, non-linearity and pool- ing are stacked, followed by more convolutional and fully-connected layers. Backpropagating gradients through a ConvNet is as simple as through a regular deep network, allowing all the weights in all the filter banks to be trained.

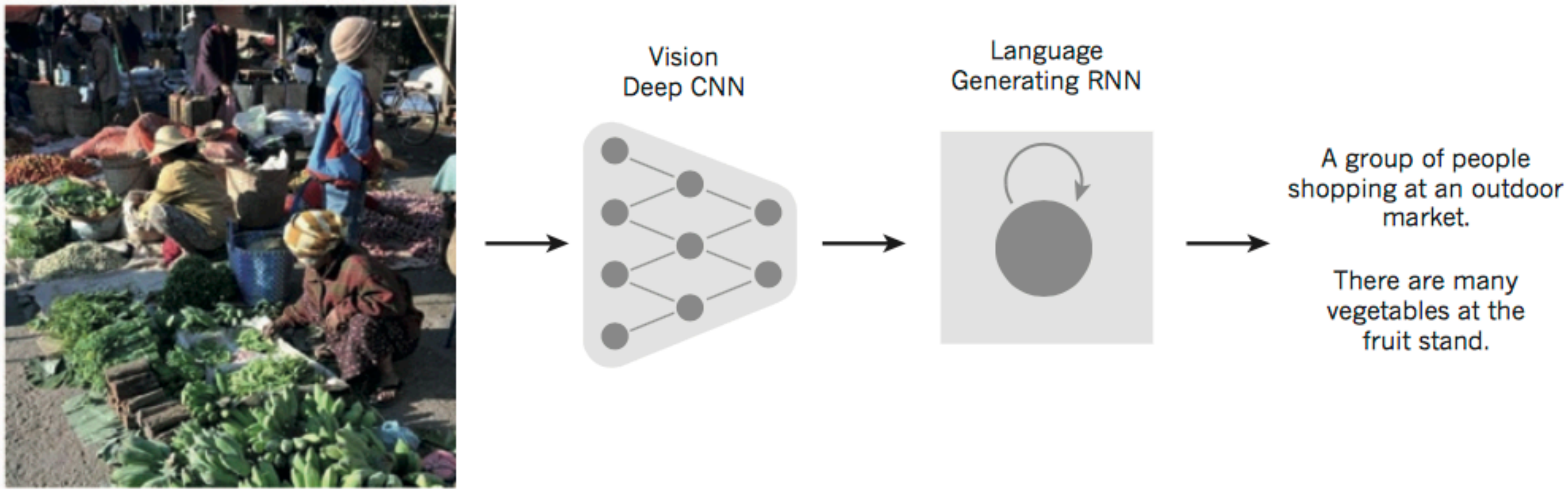
- Deep neural networks exploit the property that many natural sig- nals are compositional hierarchies, in which higher-level features are obtained by composing lower-level ones. In images, local combi- nations of edges form motifs, motifs assemble into parts, and parts form objects.
- Similar hierarchies exist in speech and text from sounds to phones, phonemes, syllables, words and sentences.
- The pooling allows representations to vary very little when elements in the previ- ous layer vary in position and appearance.

simple cells and complex cells in visual neuroscience

- LGN–V1–V2–V4–IT hierarchy in the visual cortex
- ConvNets have their roots in the neocognitron46, the architecture of which was somewhat similar, but did not have an end-to-end supervised-learning algorithm such as backpropagation
- A primitive 1D ConvNet called a time-delay neural net was used for the recognition of phonemes and simple words

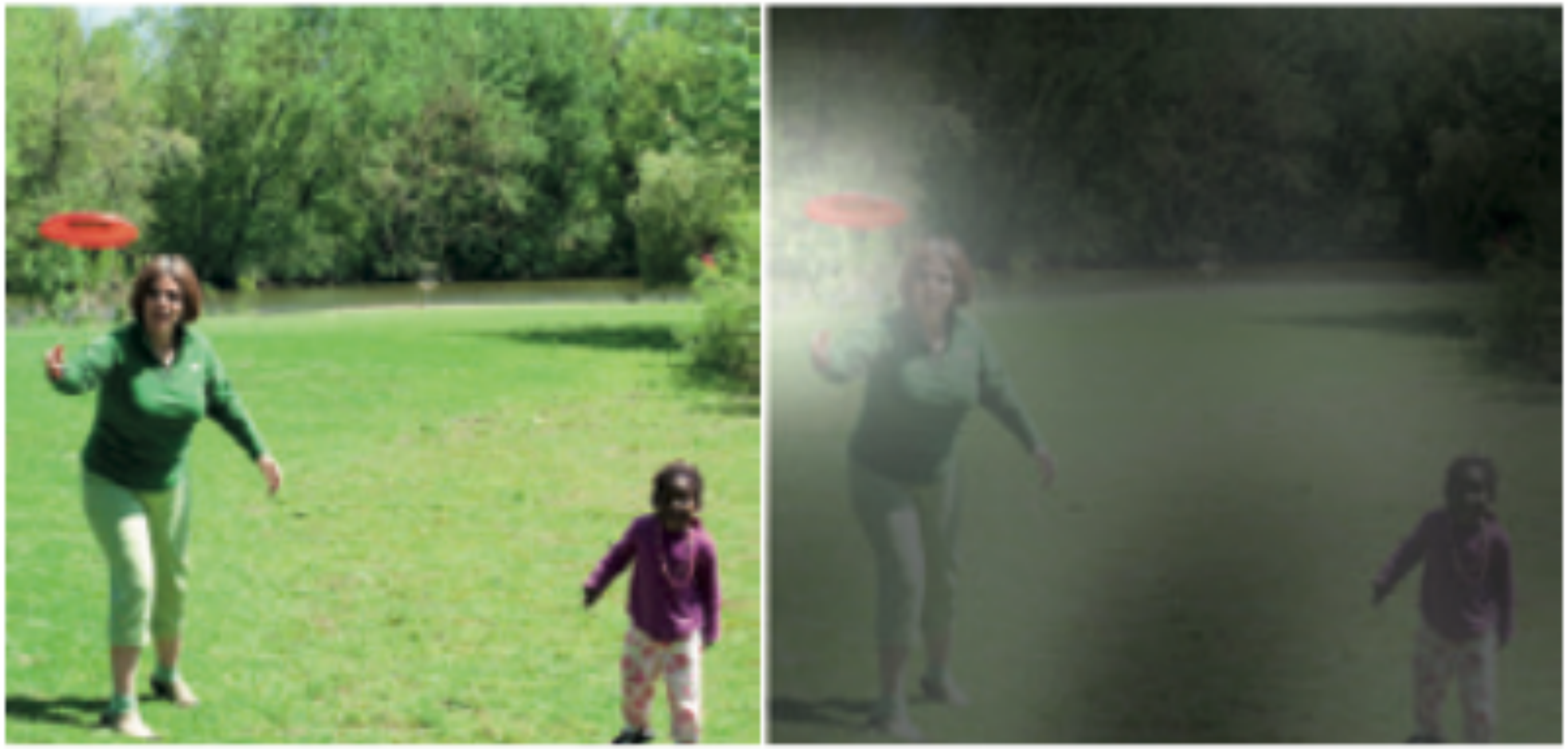
IMAGE UNDERSTANDING WITH DEEP CONVOLUTIONAL NETWORKS

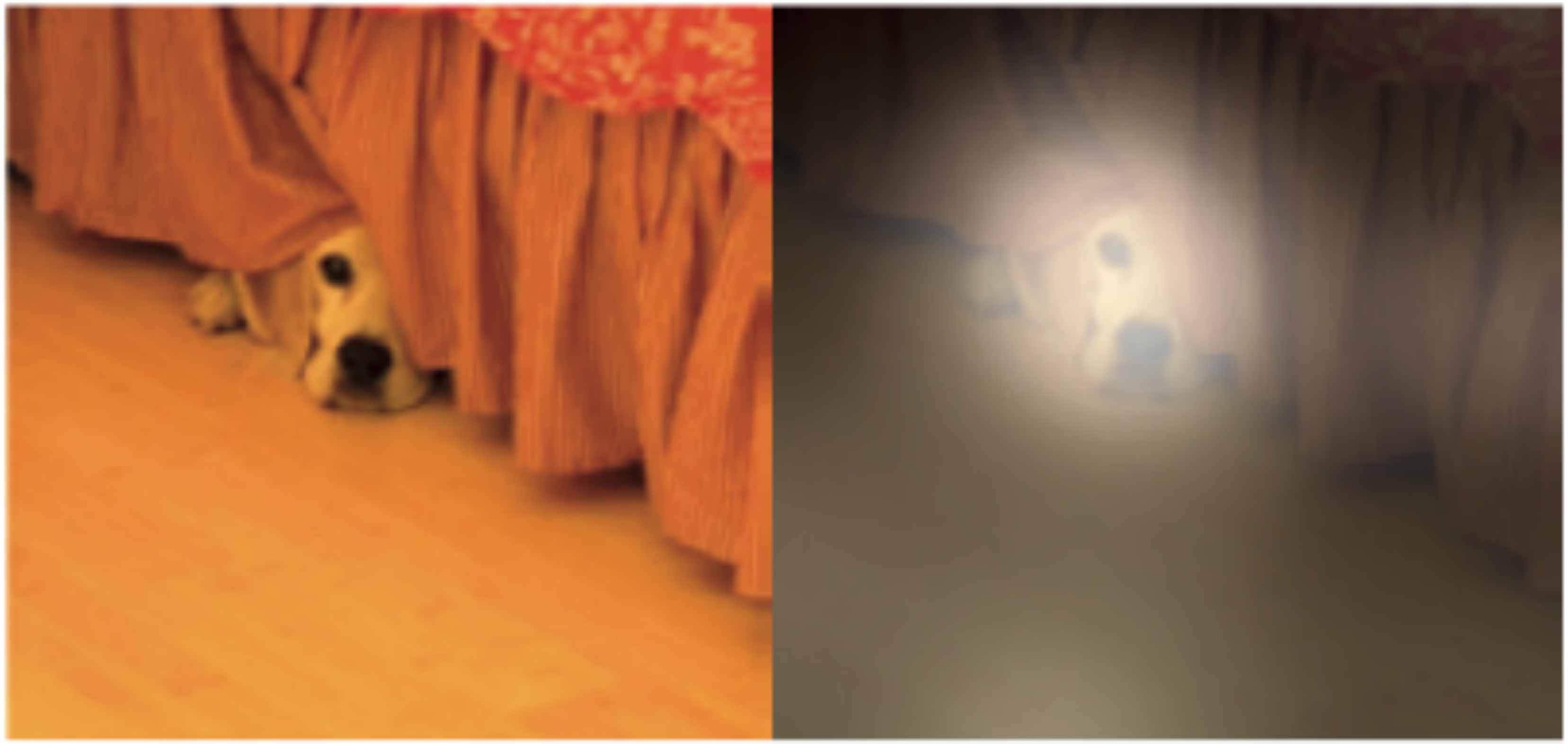
- traffic sign recognition[53], the segmentation of biological images[54] particularly for connectomics[55], and the detection of faces, text, pedestrians and human bodies in natural images[3]
- A major recent practical success of ConvNets is face recognition[59]

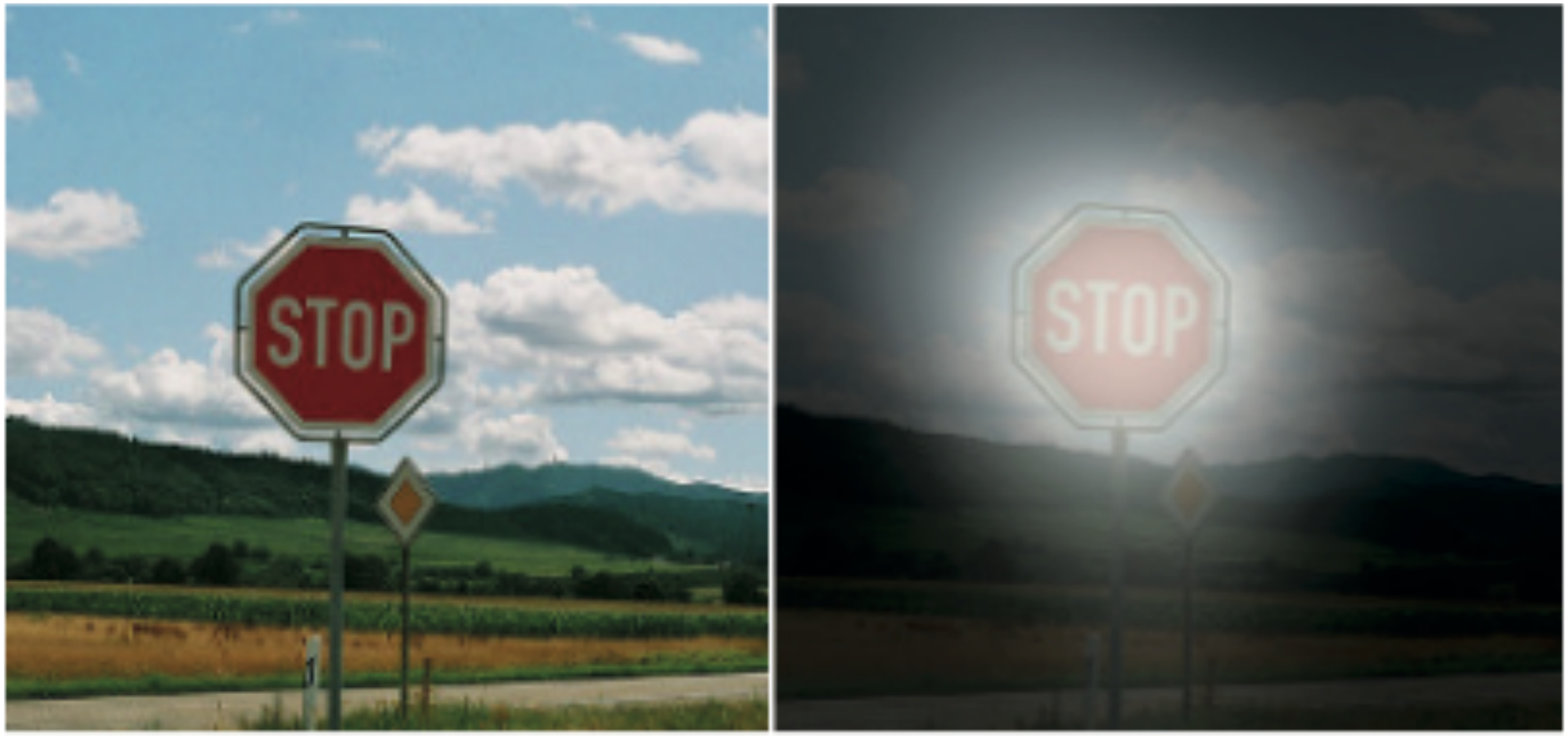


A group of people shopping at an outdoor market.

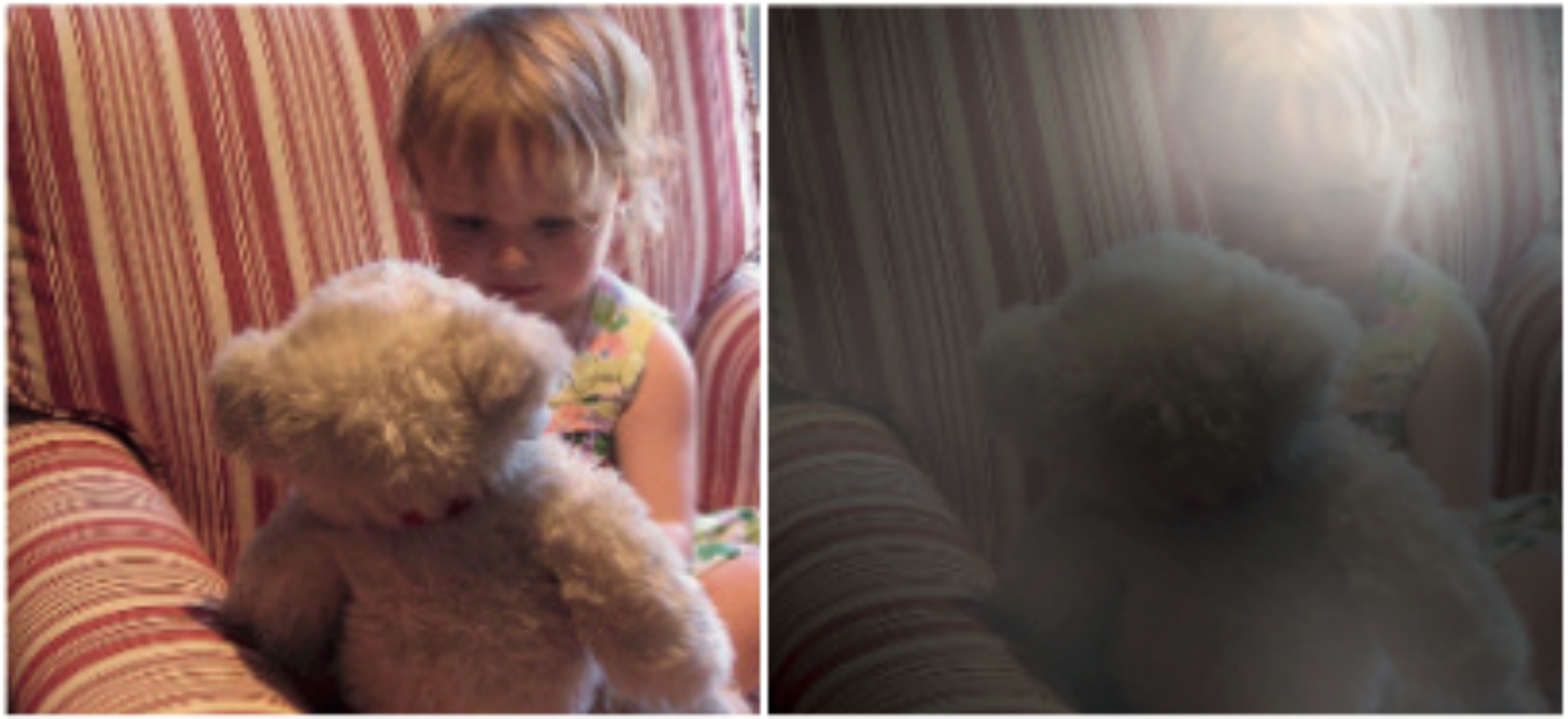
There are many vegetables at the fruit stand.


A woman is throwing a **frisbee** in a park.

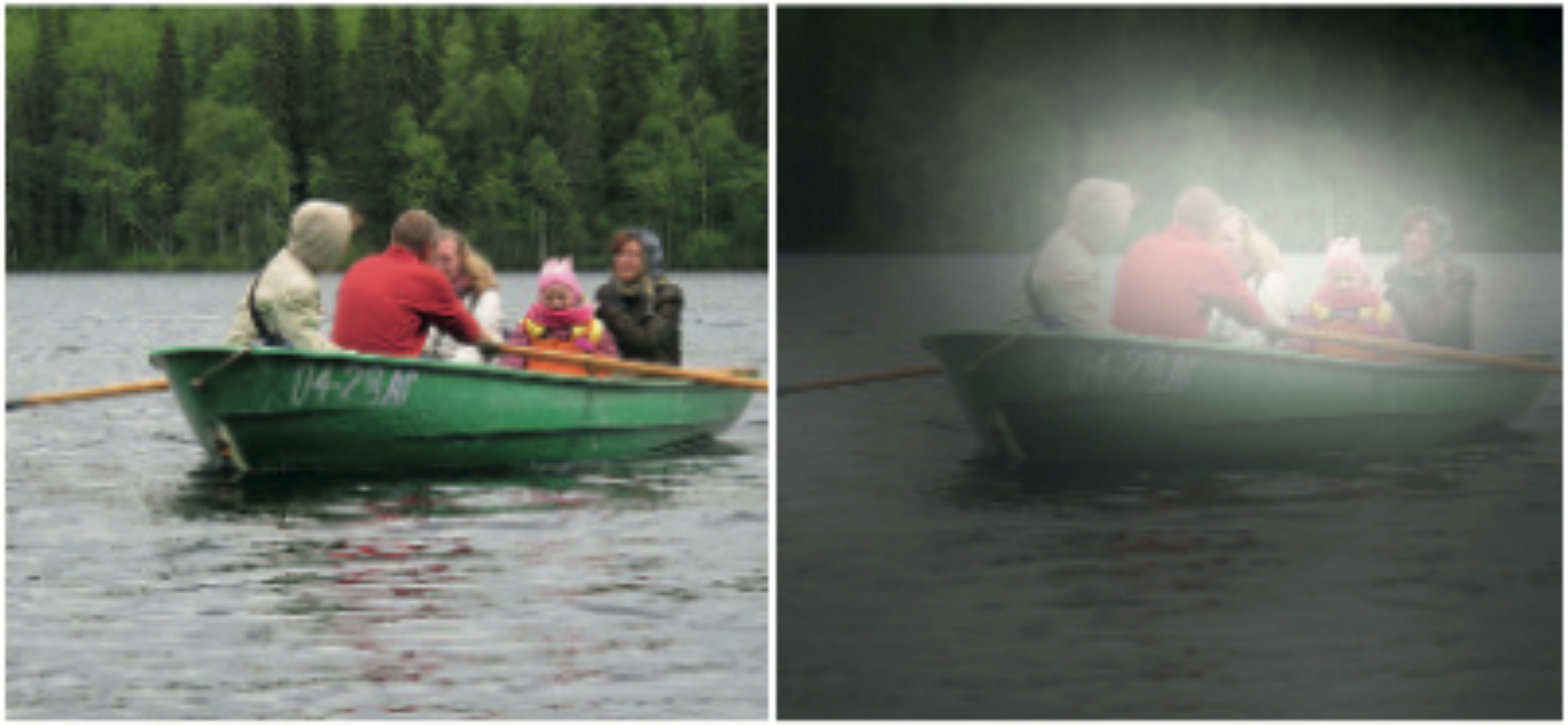

A **dog** is standing on a hardwood floor.



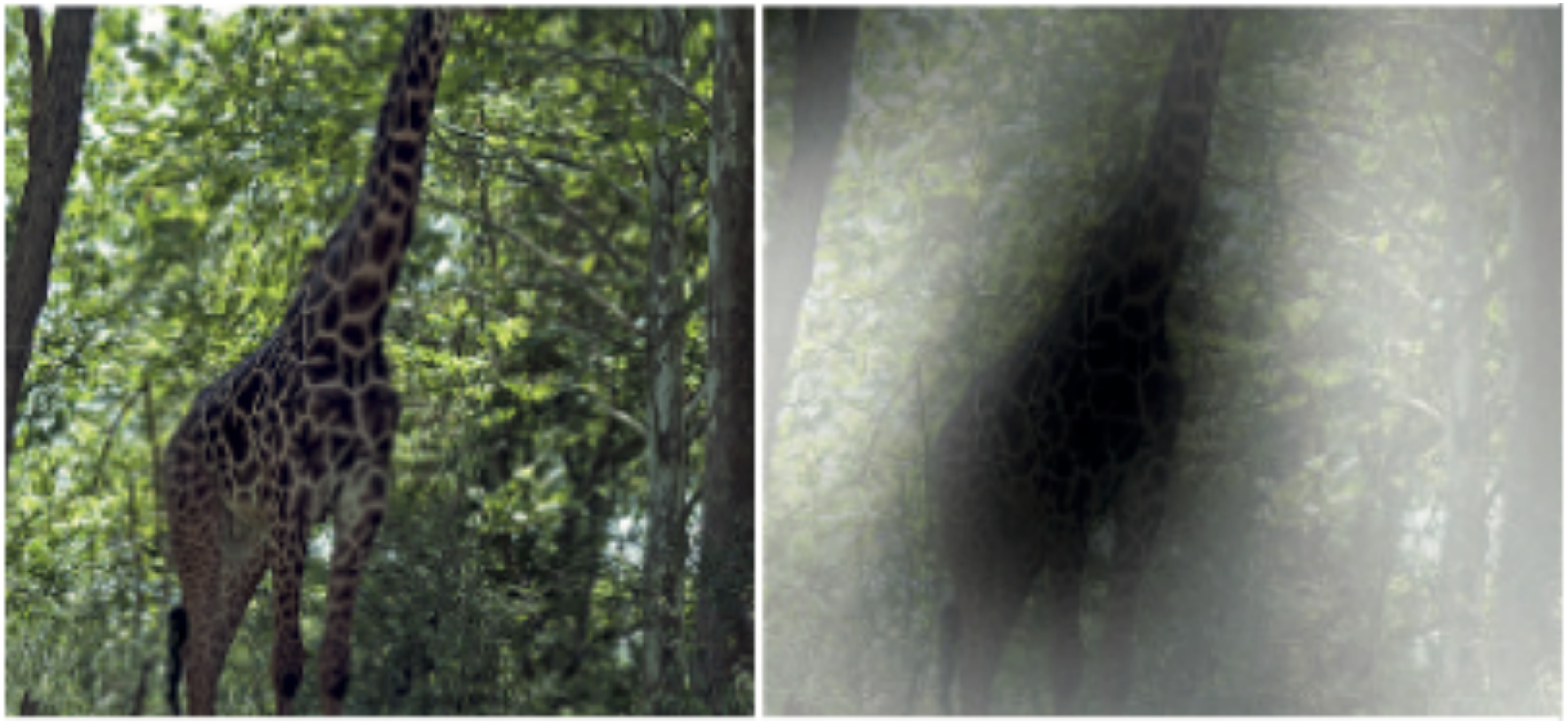
A **stop** sign is on a road with a mountain in the background



A little **girl** sitting on a bed with a teddy bear.



A group of **people** sitting on a boat in the water.



A giraffe standing in a forest with **trees** in the background.

- **Figure3lFromimagetotext.** Captions generated by a recurrentneural network (RNN) taking, as extra input, the representation extracted by a deep convolution neural network (CNN) from a test image, with the RNN trained to 'translate' high-level representations of images into captions (top). Reproduced with permission from ref. [102].
- When the RNN is given the ability to focus its attention on a different location in the input image (middle and bottom; the lighter patches were given more attention) as it generates each word (bold), we found[86] that it exploits this to achieve better 'translation' of images into captions.

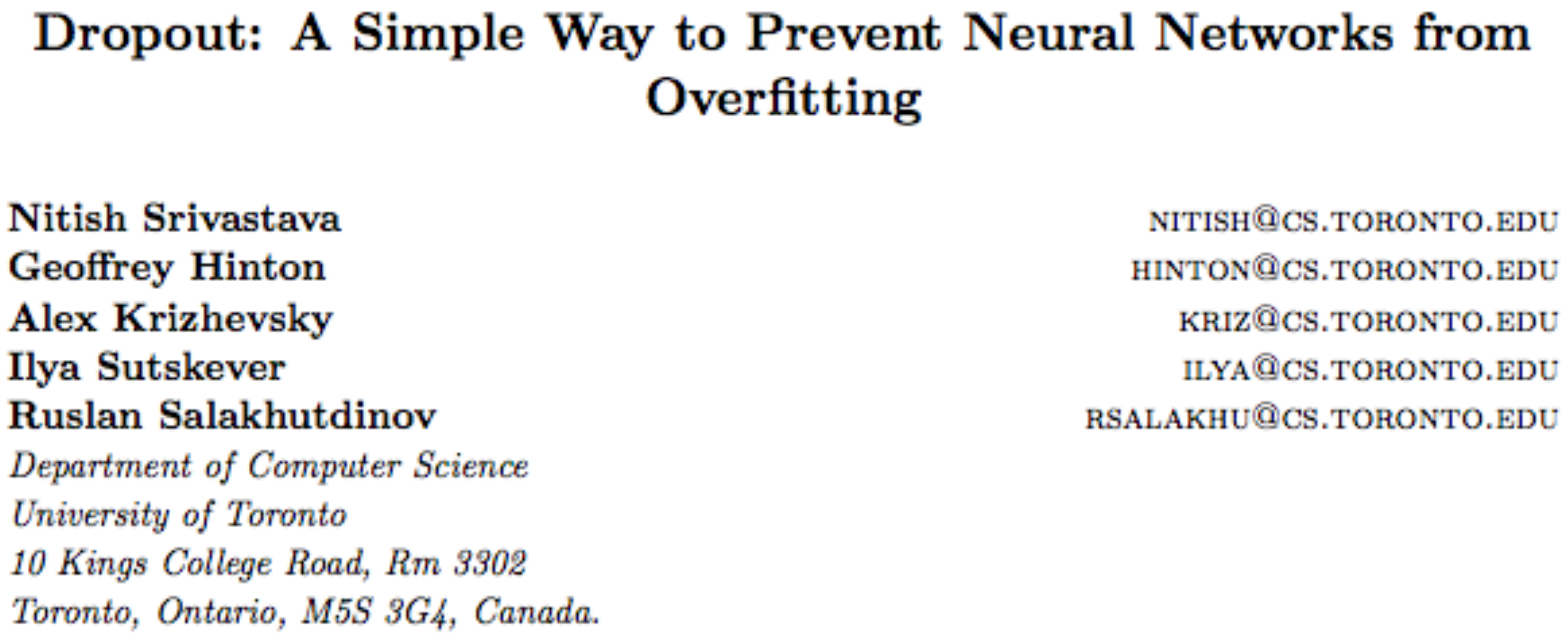
- Importantly, images can be labelled at the pixel level, which will have applications in technology, including autonomous mobile robots and
- self-driving cars

- ConvNets were largely forsaken by the mainstream computer-vision and machine-learning communities until the ImageNet competition in 2012

- When deep convolutional networks were applied to a data set of about a million images from the web that contained 1,000 different classes, they achieved spec- tacular results, almost halving the error rates of the best compet- ing approaches[1]. This success came from the efficient use of GPUs, ReLUs, a new regularization technique called dropout[62], and tech- niques to generate more training examples by deforming the existing ones. This success has brought about a revolution in computer vision; ConvNets are now the dominant approach for almost all recognition and detection tasks[4,58,59,63–65] and approach human performance on some tasks.
- A recent stunning demonstration combines ConvNets and recurrent net modules for the generation of image captions

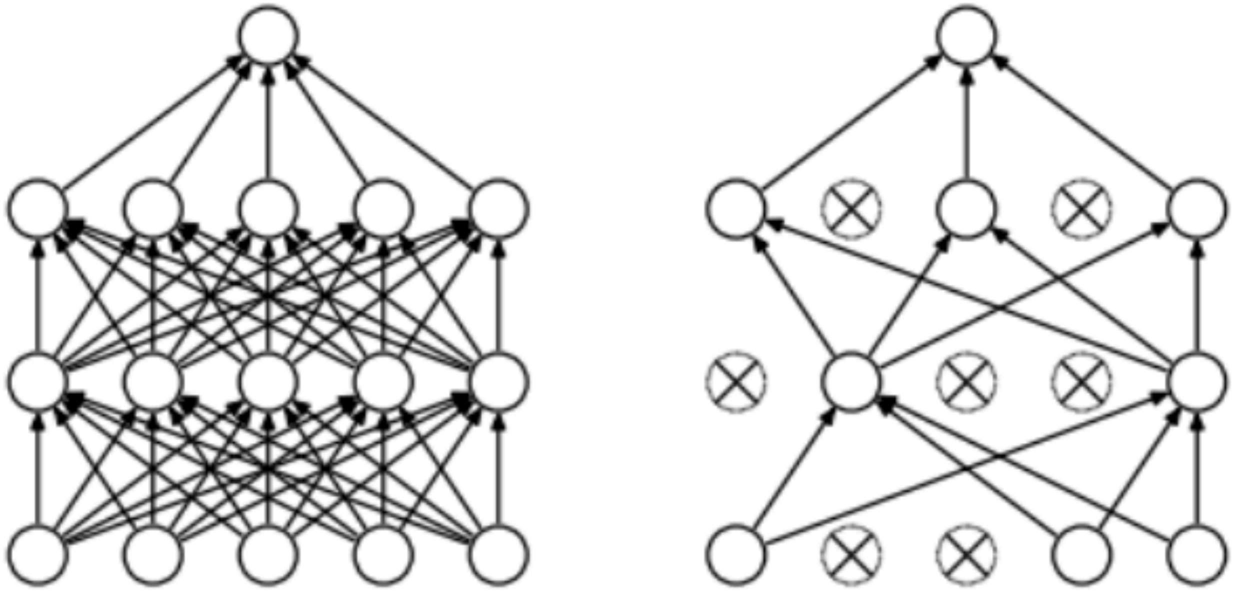
- Srivastava,N.,Hinton,G.,Krizhevsky,A.,Sutskever,I.&Salakhutdinov,R. Dropout: a simple way to prevent neural networks from overfitting. *J. Machine Learning Res.* **15,** 1929–1958 (2014).

https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf



Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Nitish Srivastava NITISH@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU
Alex Krizhevsky KRIZ@CS.TORONTO.EDU
Ilya Sutskever ILYA@CS.TORONTO.EDU
Ruslan Salakhutdinov RSALAKHU@CS.TORONTO.EDU
Department of Computer Science
University of Toronto
10 Kings College Road, Rm 3302
Toronto, Ontario, M5S 3G4, Canada.



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

igure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

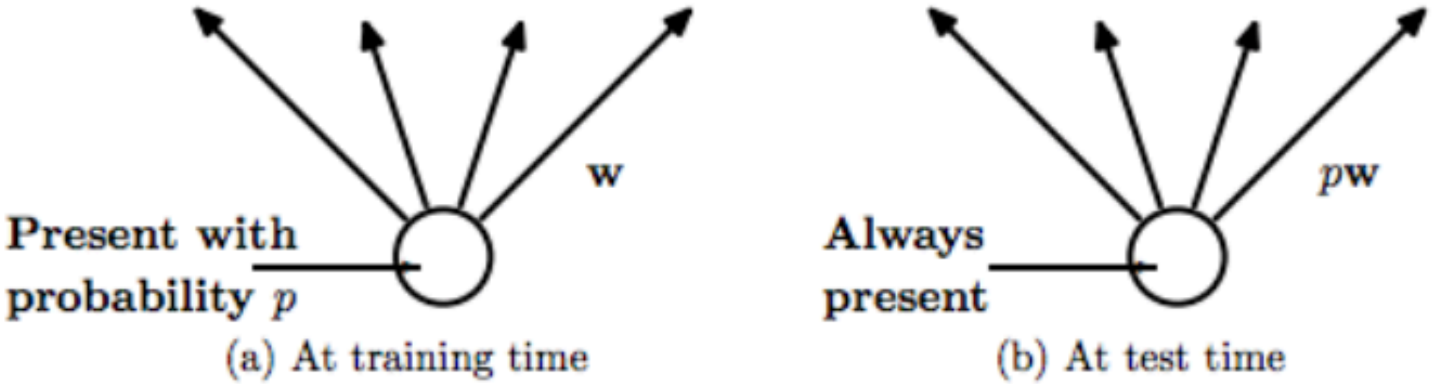


Figure 2: **Left**: A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{w}$. **Right**: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

At test time, it is not feasible to explicitly average the predictions from exponentially many thinned models. However, a very simple approximate averaging method works well in practice. The idea is to use a single neural net at test time without dropout. The weights of this network are scaled-down versions of the trained weights. If a unit is retained with probability p during training, the outgoing weights of that unit are multiplied by p at test time as shown in Figure 2. This ensures that for any hidden unit the *expected* output (under the distribution used to drop units at training time) is the same as the actual output at test time. By doing this scaling, 2^n networks with shared weights can be combined into a single neural network to be used at test time. We found that training a network with dropout and using this approximate averaging method at test time leads to significantly lower generalization error on a wide variety of classification problems compared to training with other regularization methods.

- Recent ConvNet architectures have 10 to 20 layers of ReLUs, hun- dreds of millions of weights, and billions of connections between units. Whereas training such large networks could have taken weeks only two years ago, progress in hardware, software and algorithm parallelization have reduced training times to a few hours.

- The performance of ConvNet-based vision systems has caused most major technology companies, including Google, Facebook, Microsoft, IBM, Yahoo!, Twitter and Adobe, as well as a quickly growing number of start-ups to initiate research and development projects and to deploy ConvNet-based image understanding products and services.

- ConvNets are easily amenable to efficient hardware implemen- tations in chips or field-programmable gate arrays66,67. A number of companies such as NVIDIA, Mobileye, Intel, Qualcomm and Samsung are developing ConvNet chips to enable real-time vision applications in smartphones, cameras, robots and self-driving cars.

DISTRIBUTED REPRESENTATIONS AND LANGUAGE PROCESSING

Distributed representations and language processing

Deep-learning theory shows that deep nets have two different exponential advantages over classic learning algorithms that do not use distributed representations[21]. Both of these advantages arise from the power of composition and depend on the underlying data-generating distribution having an appropriate componential structure[40]. First, learning distributed representations enable generalization to new combinations of the values of learned features beyond those seen during training (for example, 2^n combinations are possible with n binary features)[68,69]. Second, composing layers of representation in a deep net brings the potential for another exponential advantage[70] (exponential in the depth).

The hidden layers of a multilayer neural network learn to represent the network's inputs in a way that makes it easy to predict the target outputs. This is nicely demonstrated by training a multilayer neural network to predict the next word in a sequence from a local context of earlier words[71]. Each word in the context is presented to the network as a one-of-N vector, that is, one component has a value of 1 and the rest are 0. In the first layer, each word creates a different pattern of activations, or word vectors (Fig. 4). In a language model, the other layers of the network learn to convert the input word vectors into an output word vector for the predicted next word, which can be used to predict the probability for any word in the vocabulary to appear as the next word. The network learns word vectors that contain many active components each of which can be interpreted as a separate feature of the word, as was first demonstrated[27] in the context of learning distributed representations for symbols. These semantic features were not explicitly present in the input. They were discovered by the learning procedure as a good way of factorizing the structured relationships between the input and output symbols into multiple 'micro-rules'. Learning word vectors turned out to also work very well when the word sequences come from a large corpus of real text and the individual micro-rules are unreliable[71]. When trained to predict the next word in a news story, for example, the learned word vectors for Tuesday and Wednesday are very similar, as are the word vectors for Sweden and Norway. Such representations are called distributed representations because their elements (the features) are not mutually exclusive and their many configurations correspond to the variations seen in the observed data. These word vectors are composed of learned features that were not determined ahead of time by experts, but automatically discovered by the neural network. Vector representations of words learned from text are now very widely used in natural language applications[14,17,72–76].

The issue of representation lies at the heart of the debate between the logic-inspired and the neural-network-inspired paradigms for cognition. In the logic-inspired paradigm, an instance of a symbol is something for which the only property is that it is either identical or non-identical to other symbol instances. It has no internal structure that is relevant to its use; and to reason with symbols, they must be bound to the variables in judiciously chosen rules of inference. By contrast, neural networks just use big activity vectors, big weight matrices and scalar non-linearities to perform the type of fast 'intuitive' inference that underpins effortless commonsense reasoning.

Before the introduction of neural language models[71], the standard approach to statistical modelling of language did not exploit distributed representations: it was based on counting frequencies of occurrences of short symbol sequences of length up to N (called N-grams). The number of possible N-grams is on the order of V^N, where V is the vocabulary size, so taking into account a context of more than a handful of words would require very large training corpora. N-grams treat each word as an atomic unit, so they cannot generalize across semantically related sequences of words, whereas neural language models can because they associate each word with a vector of real valued features, and semantically related words end up close to each other in that vector space (Fig. 4).

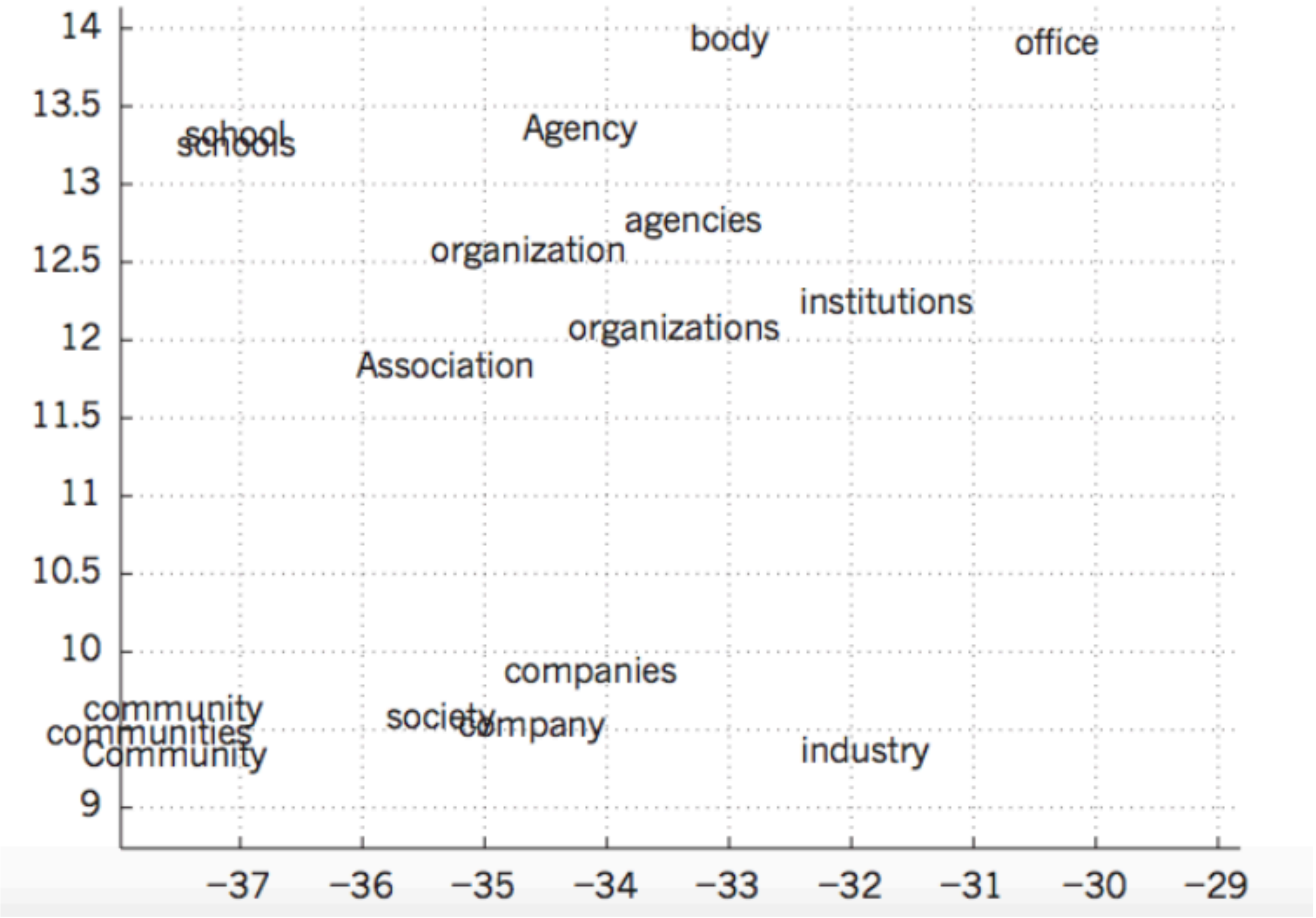
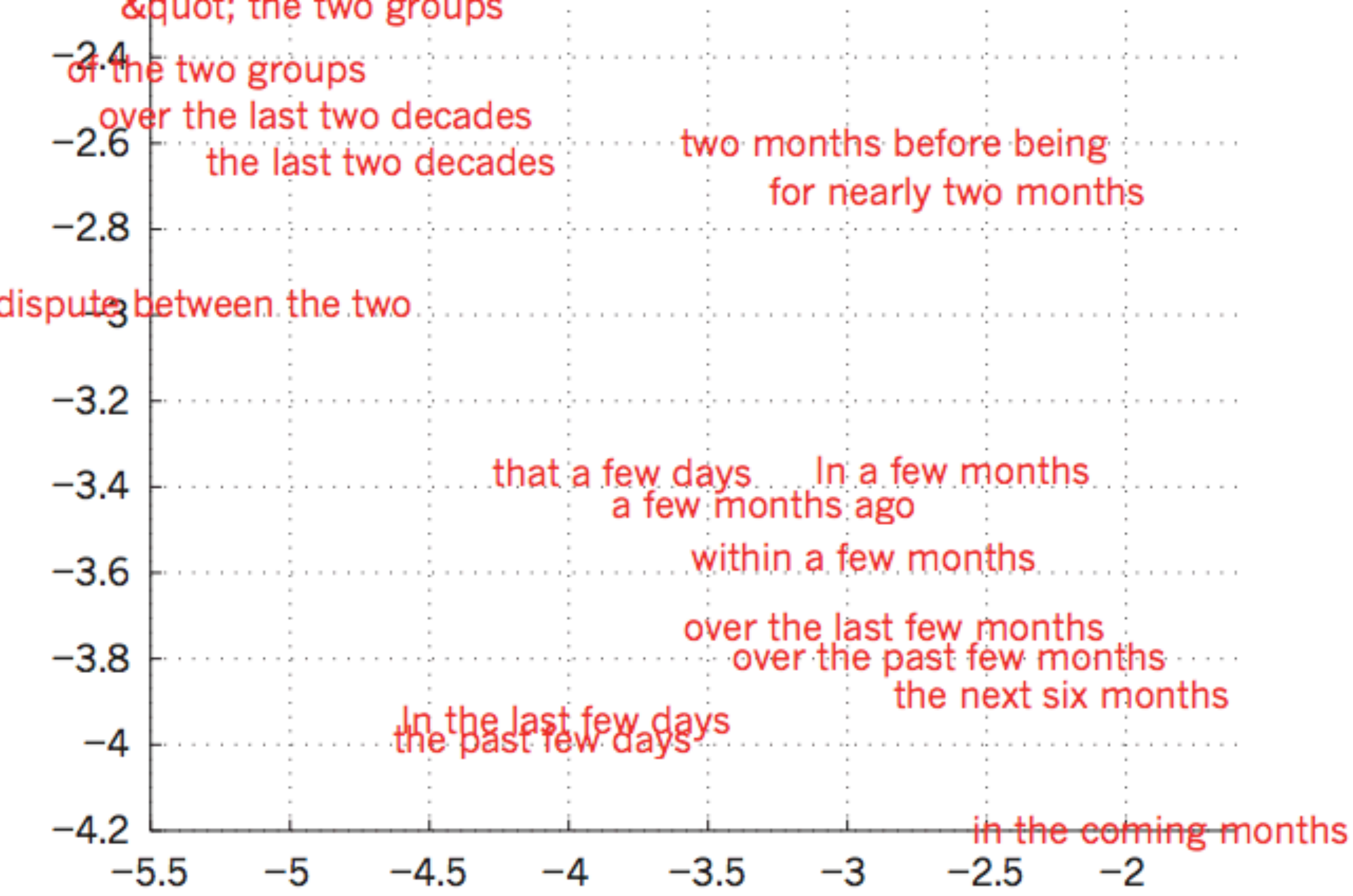


Figure 4 | Visualizing the learned word vectors. On the left is an illustration of word representations learned for modelling language, non-linearly projected to 2D for visualization using the t-SNE algorithm[103]. On the right is a 2D representation of phrases learned by an English-to-French encoder–decoder recurrent neural network[75]. One can observe that semantically similar words



or sequences of words are mapped to nearby representations. The distributed representations of words are obtained by using backpropagation to jointly learn a representation for each word and a function that predicts a target quantity such as the next word in a sequence (for language modelling) or a whole sequence of translated words (for machine translation)[18,75].

RECURRENT NEURAL NETWORKS

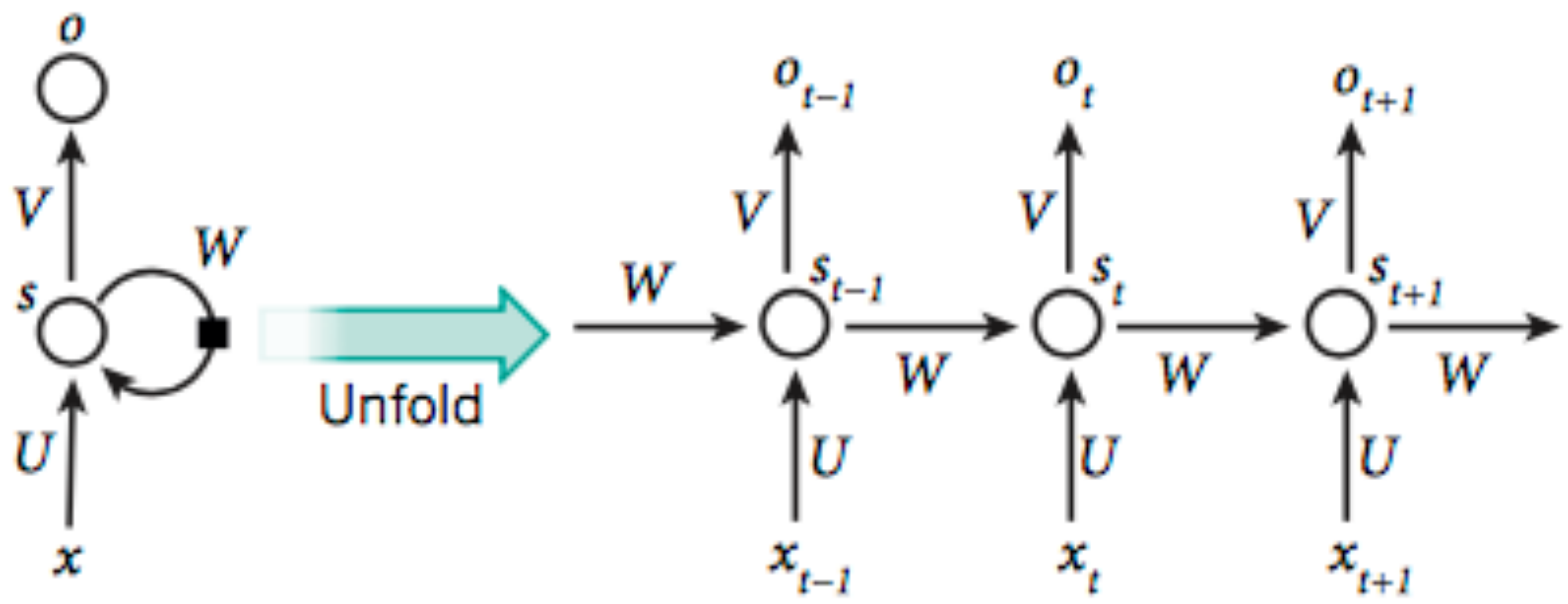

o
V
W
s
U
x
Unfold
W
o_{t-1}
V
s_{t-1}
U
x_{t-1}
W
o_t
V
s_t
U
x_t
W
o_{t+1}
V
s_{t+1}
U
x_{t+1}
W

Figure 5 | A recurrent neural network and the unfolding in time of the computation involved in its forward computation. The artificial neurons (for example, hidden units grouped under node s with values s_t at time t) get inputs from other neurons at previous time steps (this is represented with the black square, representing a delay of one time step, on the left). In this way, a recurrent neural network can map an input sequence with elements x_t into an output sequence with elements o_t, with each o_t depending on all the previous x_t' (for $t' \leq t$). The same parameters (matrices U, V, W) are used at each time step. Many other architectures are possible, including a variant in which the network can generate a sequence of outputs (for example, words), each of which is used as inputs for the next time step. The backpropagation algorithm (Fig. 1) can be directly applied to the computational graph of the unfolded network on the right, to compute the derivative of a total error (for example, the log-probability of generating the right sequence of outputs) with respect to all the states s_t and all the parameters.

Instead of translating the meaning of a French sentence into an English sentence, one can learn to 'translate' the meaning of an image into an English sentence (Fig. 3). The encoder here is a deep ConvNet that converts the pixels into an activity vector in its last hidden layer. The decoder is an RNN similar to the ones used for machine translation and neural language modelling. There has been a surge of interest in such systems recently (see examples mentioned in ref. 86).

RNNs, once unfolded in time (Fig. 5), can be seen as very deep feedforward networks in which all the layers share the same weights. Although their main purpose is to learn long-term dependencies, theoretical and empirical evidence shows that it is difficult to learn to store information for very long[78].

To correct for that, one idea is to augment the network with an explicit memory. The first proposal of this kind is the long short-term memory (LSTM) networks that use special hidden units, the natural behaviour of which is to remember inputs for a long time[79]. A special unit called the memory cell acts like an accumulator or a gated leaky neuron: it has a connection to itself at the next time step that has a weight of one, so it copies its own real-valued state and accumulates the external signal, but this self-connection is multiplicatively gated by another unit that learns to decide when to clear the content of the memory.

LSTM networks have subsequently proved to be more effective than conventional RNNs, especially when they have several layers for each time step[87], enabling an entire speech recognition system that goes all the way from acoustics to the sequence of characters in the transcription. LSTM networks or related forms of gated units are also currently used for the encoder and decoder networks that perform so well at machine translation[17,72,76].

Over the past year, several authors have made different proposals to augment RNNs with a memory module. Proposals include the Neural Turing Machine in which the network is augmented by a 'tape-like' memory that the RNN can choose to read from or write to[88], and memory networks, in which a regular network is augmented by a kind of associative memory[89]. Memory networks have yielded excellent performance on standard question-answering benchmarks. The memory is used to remember the story about which the network is later asked to answer questions.

Beyond simple memorization, neural Turing machines and memory networks are being used for tasks that would normally require reasoning and symbol manipulation. Neural Turing machines can be taught 'algorithms'. Among other things, they can learn to output a sorted list of symbols when their input consists of an unsorted sequence in which each symbol is accompanied by a real value that indicates its priority in the list[88]. Memory networks can be trained to keep track of the state of the world in a setting similar to a text adventure game and after reading a story, they can answer questions that require complex inference[90]. In one test example, the network is shown a 15-sentence version of the *The Lord of the Rings* and correctly answers questions such as "where is Frodo now?"[89].

The future of deep learning

Unsupervised learning[91–98] had a catalytic effect in reviving interest in deep learning, but has since been overshadowed by the successes of purely supervised learning. Although we have not focused on it in this Review, we expect unsupervised learning to become far more important in the longer term. Human and animal learning is largely unsupervised: we discover the structure of the world by observing it, not by being told the name of every object.

Human vision is an active process that sequentially samples the optic array in an intelligent, task-specific way using a small, high-resolution fovea with a large, low-resolution surround. We expect much of the future progress in vision to come from systems that are trained end-to-end and combine ConvNets with RNNs that use reinforcement learning to decide where to look. Systems combining deep learning and reinforcement learning are in their infancy, but they already outperform passive vision systems[99] at classification tasks and produce impressive results in learning to play many different video games[100].

Natural language understanding is another area in which deep learning is poised to make a large impact over the next few years. We expect systems that use RNNs to understand sentences or whole documents will become much better when they learn strategies for selectively attending to one part at a time[76,86].

Ultimately, major progress in artificial intelligence will come about through systems that combine representation learning with complex reasoning. Although deep learning and simple reasoning have been used for speech and handwriting recognition for a long time, new paradigms are needed to replace rule-based manipulation of symbolic expressions by operations on large vectors[101]. ■