Machine learning

Part V

a biswas IIEST Shibpur

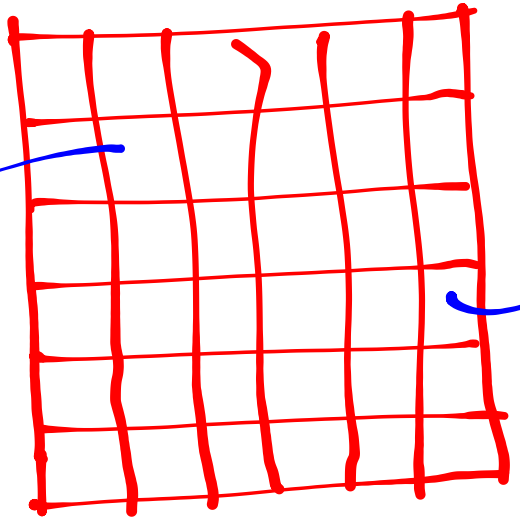
Convolutional

Networks

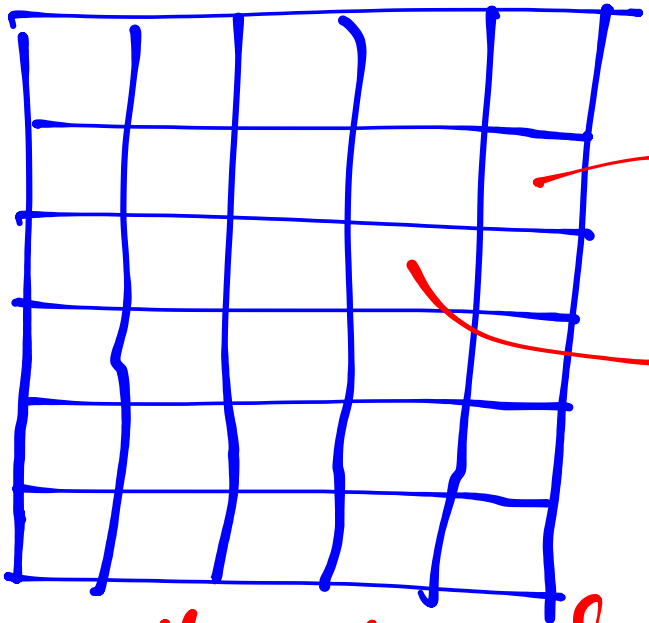
Neural Networks discussed so far are fully connected.

However, such networks may be not appropriate to classify images as it does not take into account the spatial nature of images.

For example, you have an input image



Both the cases will be treated at same pedestal by the neural network.



spatially adjacent, this information is not useful

Use something like tabula rasa which captures the spatial fracture, kind of proximity (neighborhood) information.

Convolutional Neural Networks

is a special architecture which adapts well to classify images.

. Using this, you can train the network fast.

. In turn, you can train deep, many layer networks.

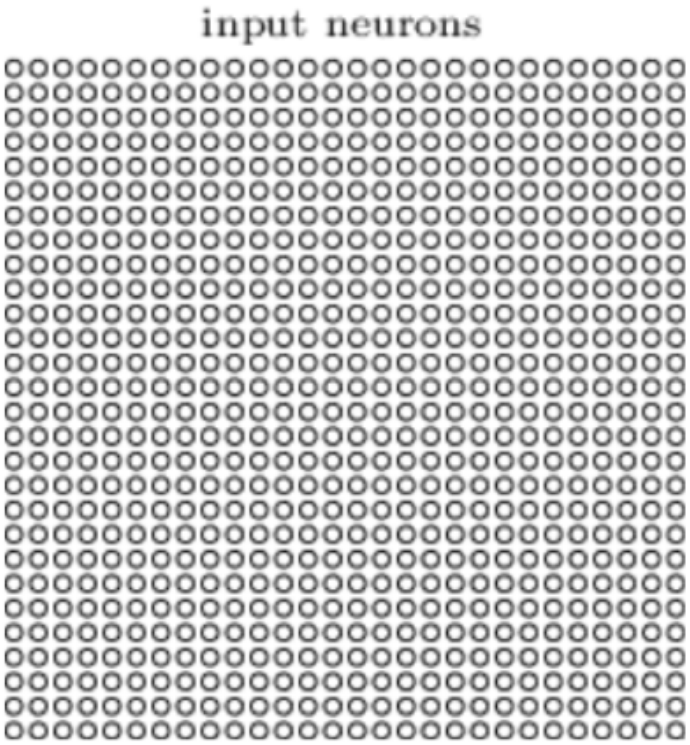
These days,
deep convolutional network or close variant
are used in most neural network
image recognition task.

Convolutional Neural Network

uses three basic ideas

1) local receptive fields
2) shared weights
3) pooling

Local receptive fields



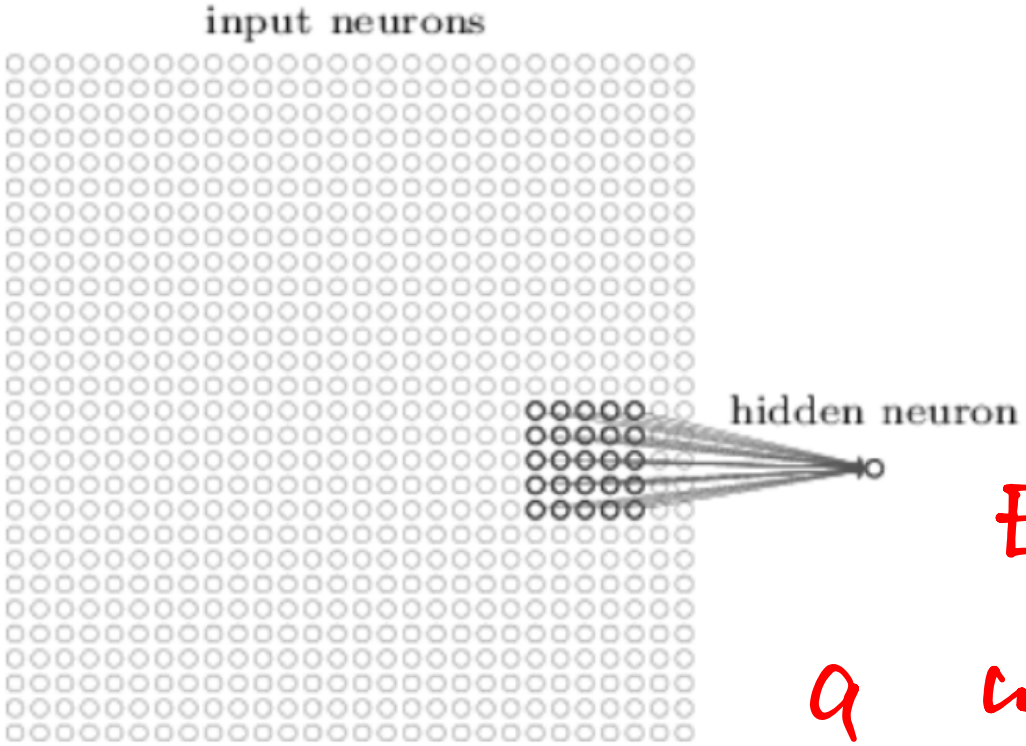
We will connect the input pixels to a layer of hidden neurons.

But, not every input pixel to every hidden neuron.

Rather, connections in small localized region.

Each neuron in the first hidden layer will be connected to a small region of the input neurons

[5 x 5 region = 25 input pixels]



The region is called local receptive field.

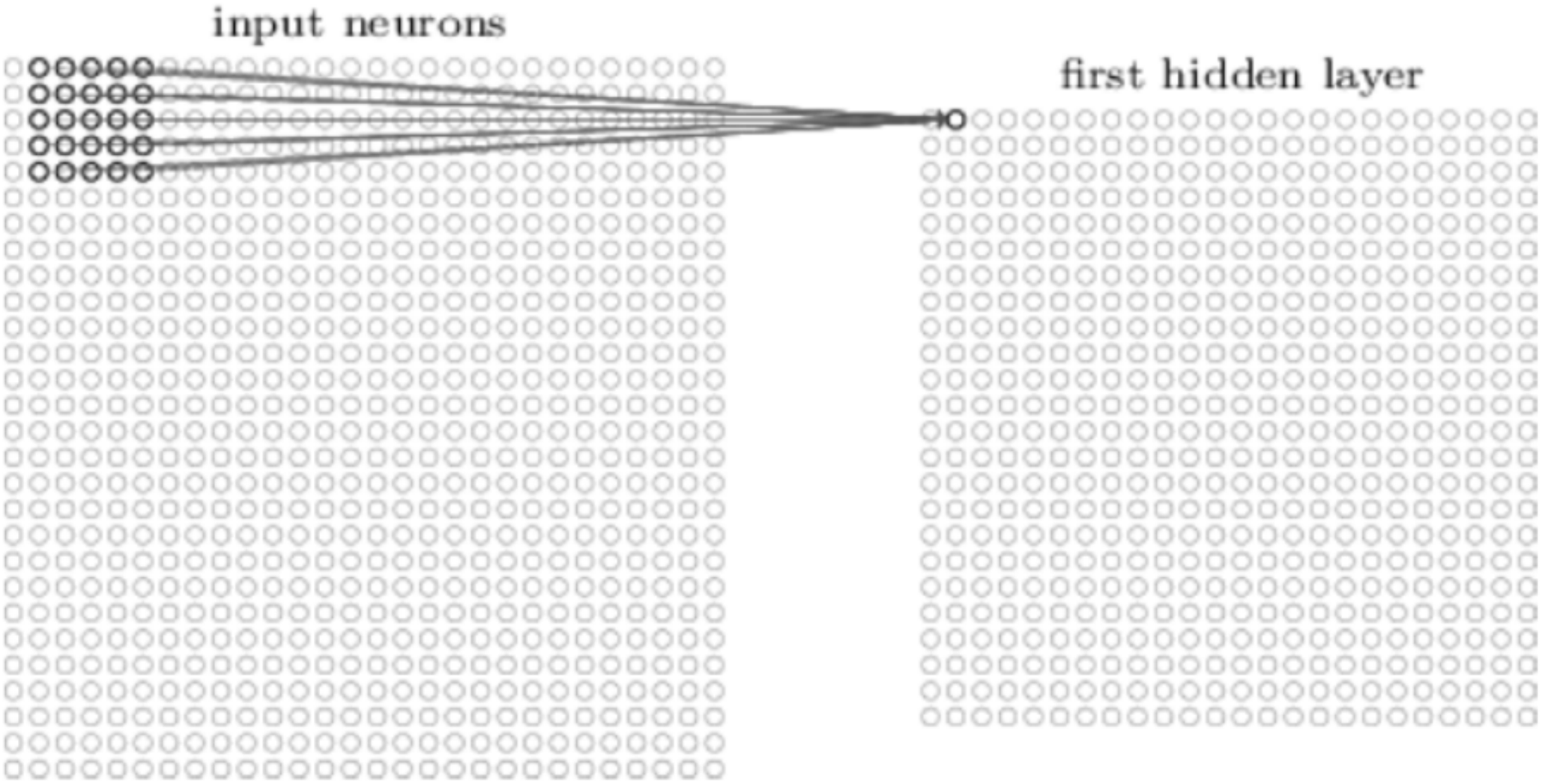
Each connection learns a weight.

Hidden neuron learns a overall bias.

Hence, to analyze the particular local receptive field, that hidden neuron learns

Now, slide the local receptive field across the image.

For each local receptive field there is a different hidden neuron.



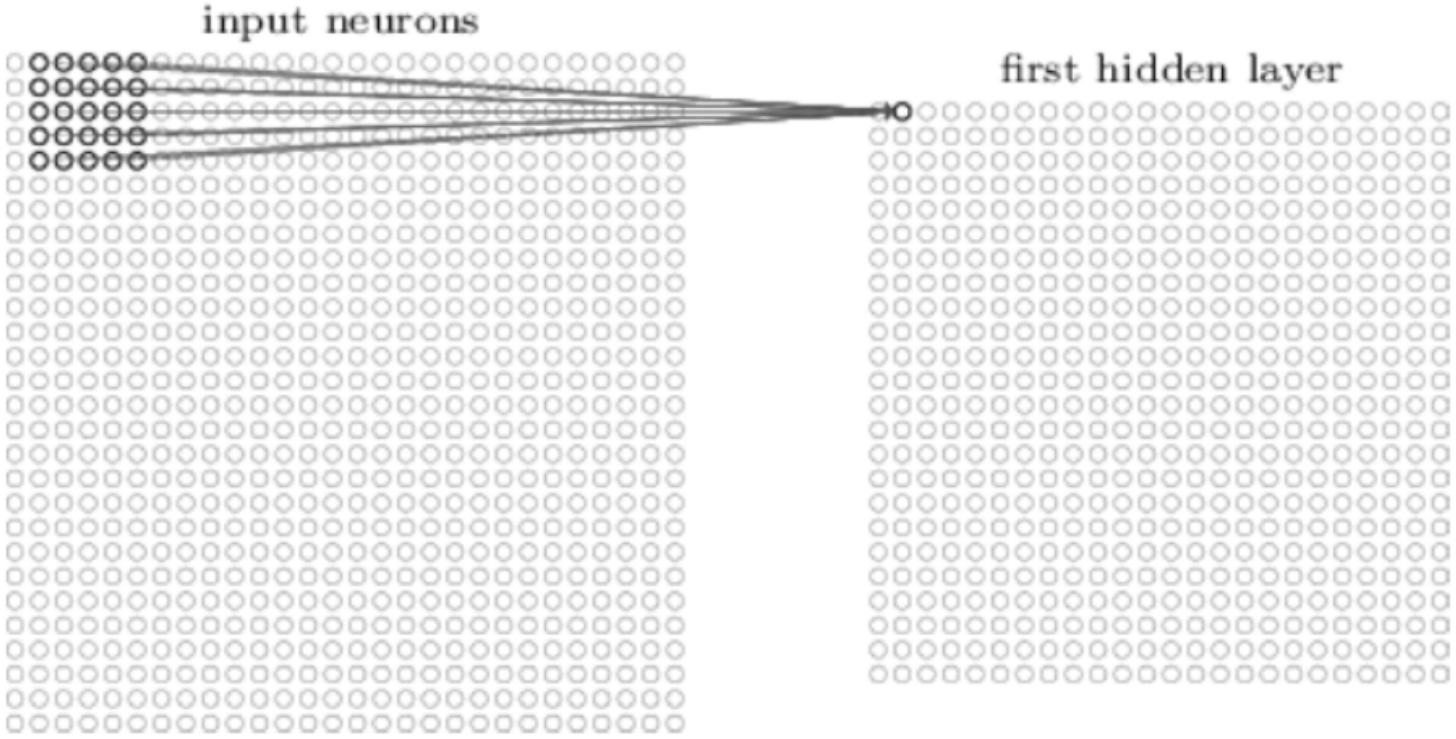
Sum at j,k th hidden neuron

$$b + \sum_{l=0}^{4} \sum_{m=0}^{4} w_{l,m}\, a_{j+l,k+m}$$

b = shared bias

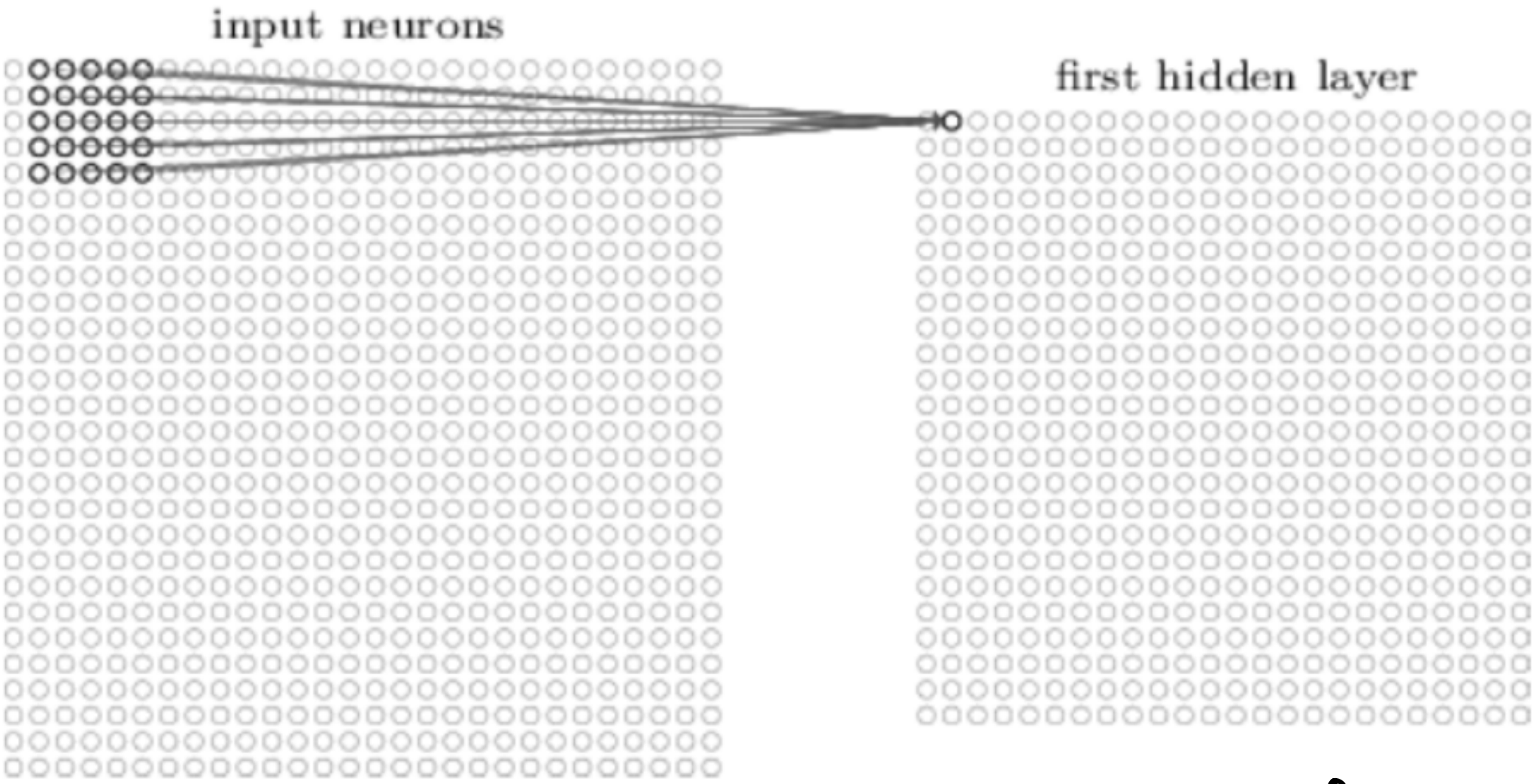
w = 5×5 weights

a = activation at (x,y), pixel value.



Output at j,k-th hidden neuron

$$\sigma\left(b + \sum_{l}\sum_{m} w_{l,m}\, a_{j+l,k+m}\right)$$



If input image is 28x28, 5x5 is local receptive field, then there will be 24x24 neurons in the hidden layer.

Stride length:

Movement of local receptive field (5×5 window) at a time.

Stride length = 1 means one pixel at a time.

Shared weights and biases

Each hidden neuron has a bias and $5 \times 5 = 25$ weights connected to it.

We use the same weights and biases for the 24×24 hidden neurons.

Output at i,k-th hidden neuron

$$\sigma\left(b + \sum_{l}\sum_{m} w_{l,m}\, a_{j+l,k+m}\right)$$

It means all the neurons in the first hidden layer detect exactly the same feature, may be of different locations.

Example: a vertical edge.

Thus, we can apply the same feature detector everywhere in the image.

Convolutional Neural Networks achieves translation invariance.

Feature map:

map from the input layer to the hidden layer.

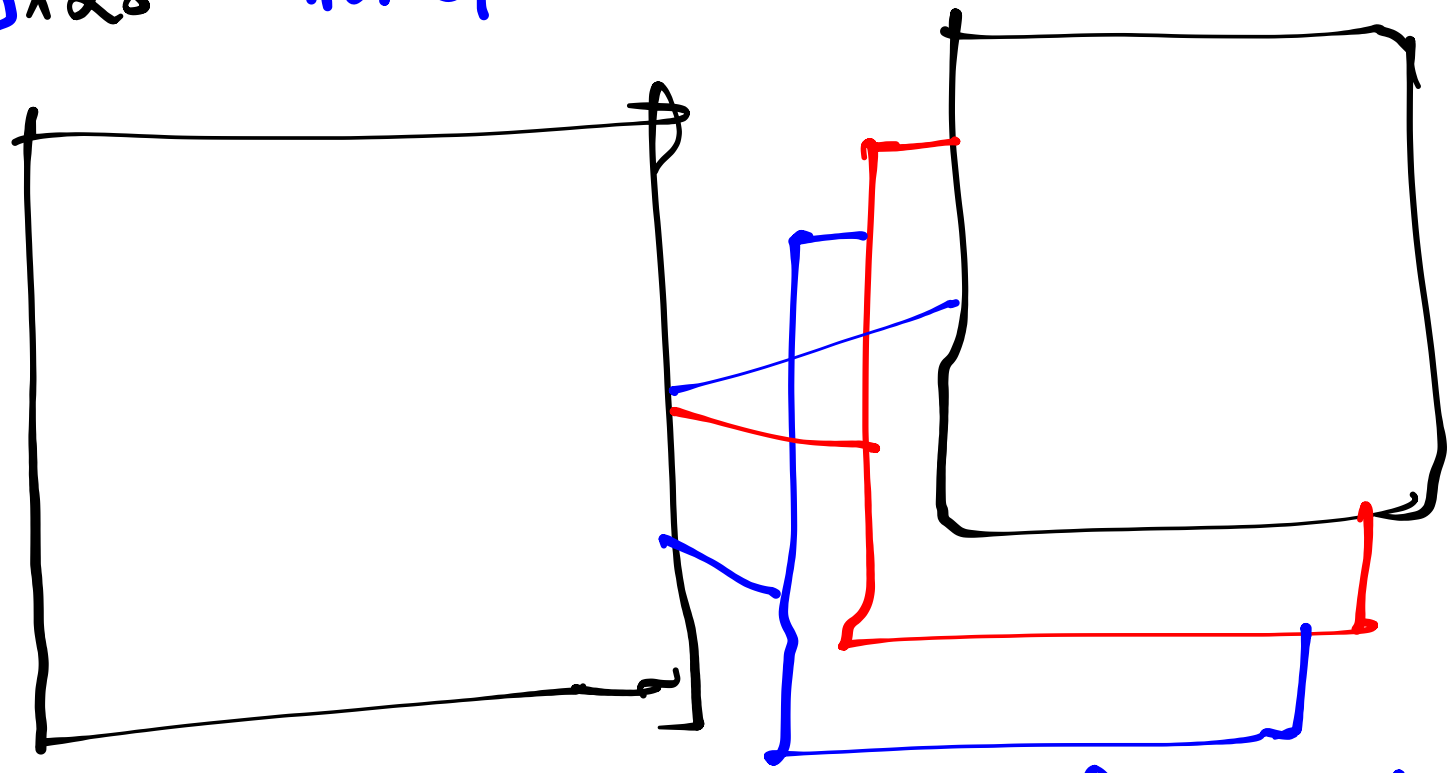
Kernel or filter

Shared weights and biases.

For image recognition, you may require more than one feature map.

A convolutional layer may consist of more than one feature map.

28x28 Kernel 5x5



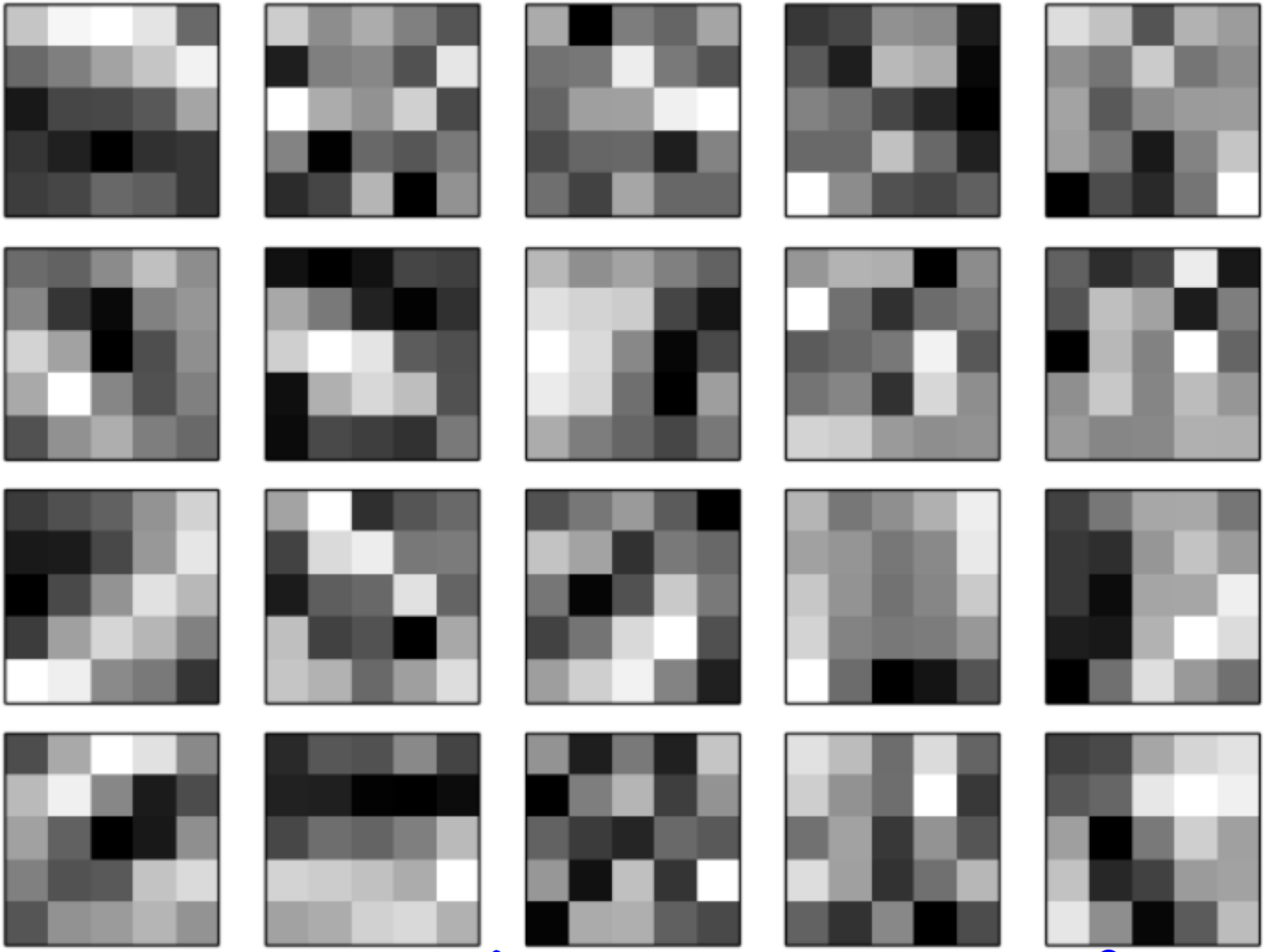
Input layer

First hidden layer
3 feature maps

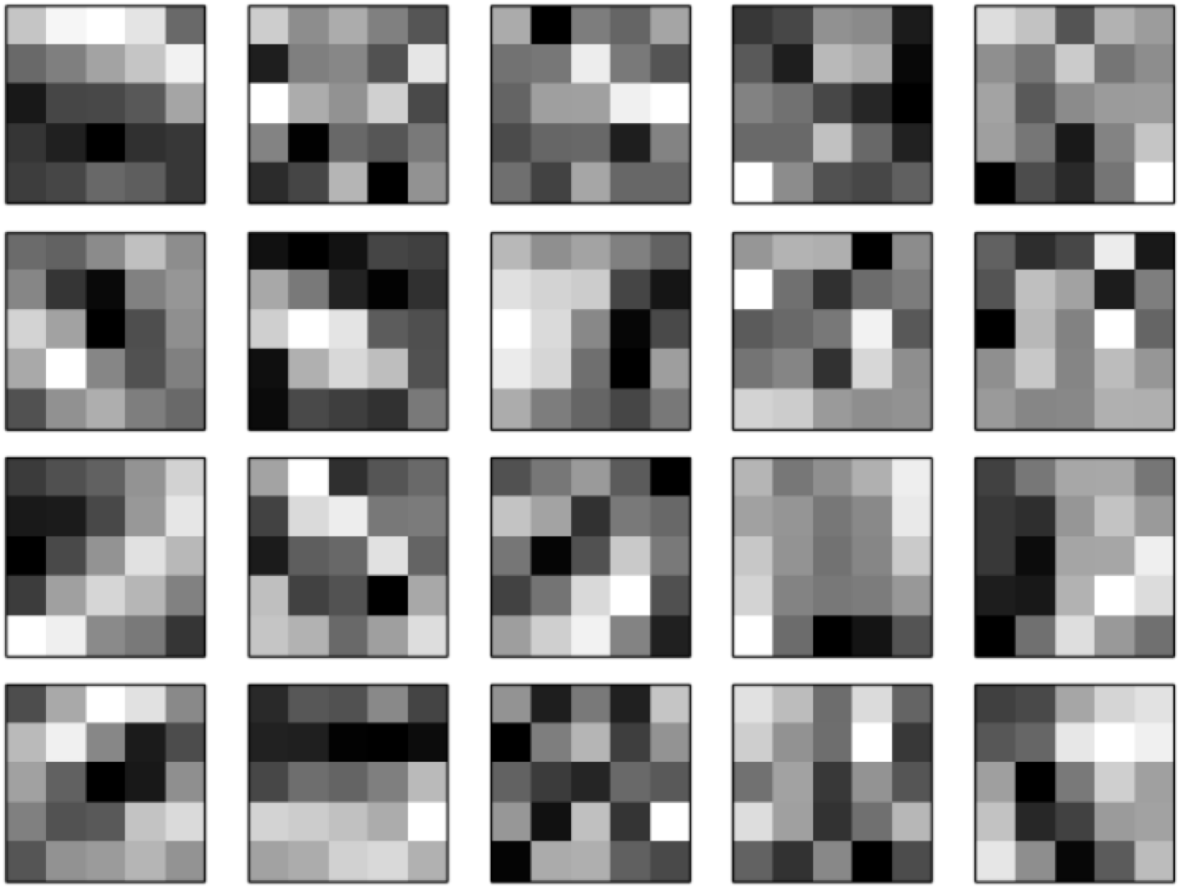
For each feature map, shared 5x5 weights and one single bias.

Thus, the network can detect three different kinds of feature with each feature being detectable across the image.

LeNet-5 used six feature maps.



20 images corresponding to 20 different feature maps.

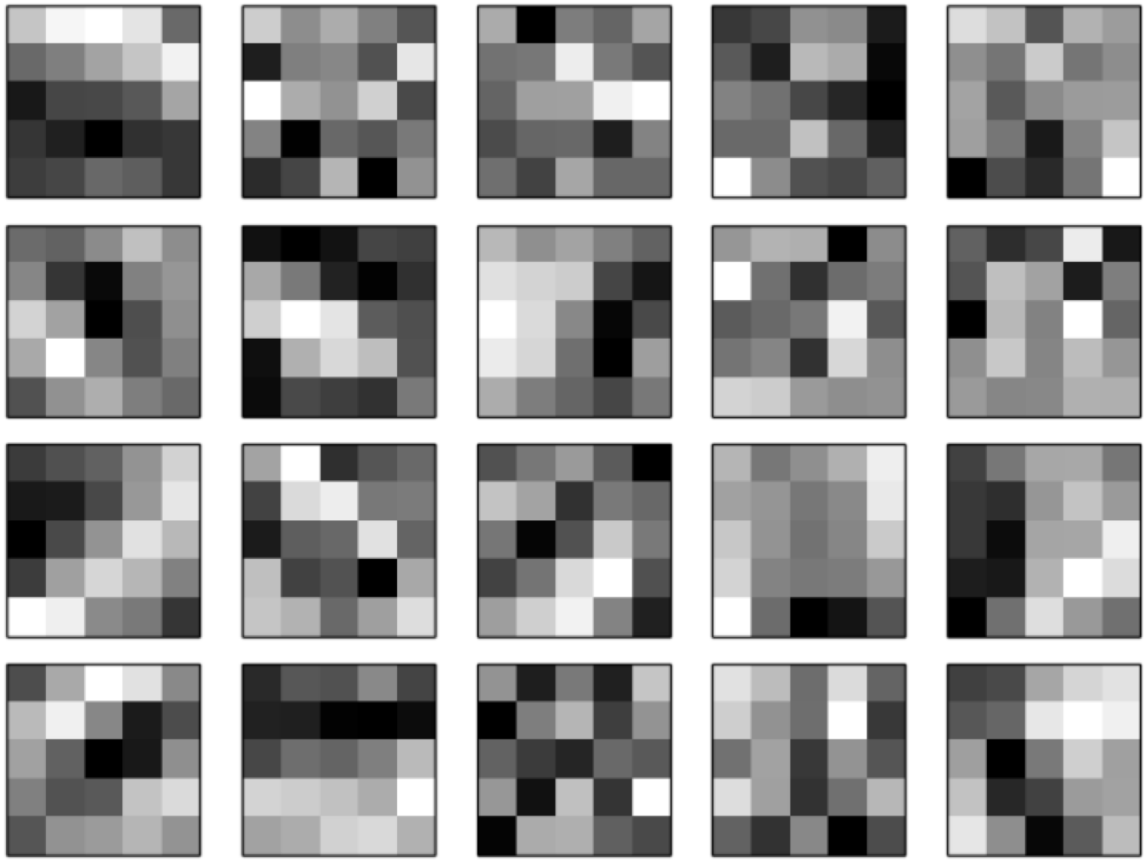


Each image is 5×5 image representing the 5×5 weights.

Block is whiter
⇒ weights are less.

So the feature map will respond less to the corresponding input pixels.

Block is darker → weights are more →
feature map will respond more to ~~those~~ the input pixels.



The images depicts the type of features the convolutional layer responds to.

Ref: Visualizing and understading convolutional networks
Mathew Zeiler and Rob Fergus (2013).

Advantage of sharing weights and biases:

Reduces the number of parameters.

How many parameters for 5×5 kernel and 20 such feature maps?

Ans: For each feature map

$$\underbrace{5\times5}_{\text{weights}} + \underbrace{1}_{\text{bias}} = 26$$

For 20 feature maps $26\times20 = 520$ parameters

How many parameters for a fully connected neural network for image size 28×28 with 30 neurons in the hidden layer?

ANS: $28 \times 28 = 784$ input neurons.

Fully Connected $\Rightarrow 784 \times 30 = 23520$ weights

Number of biases $= 30$.

A total of 23550 parameters.

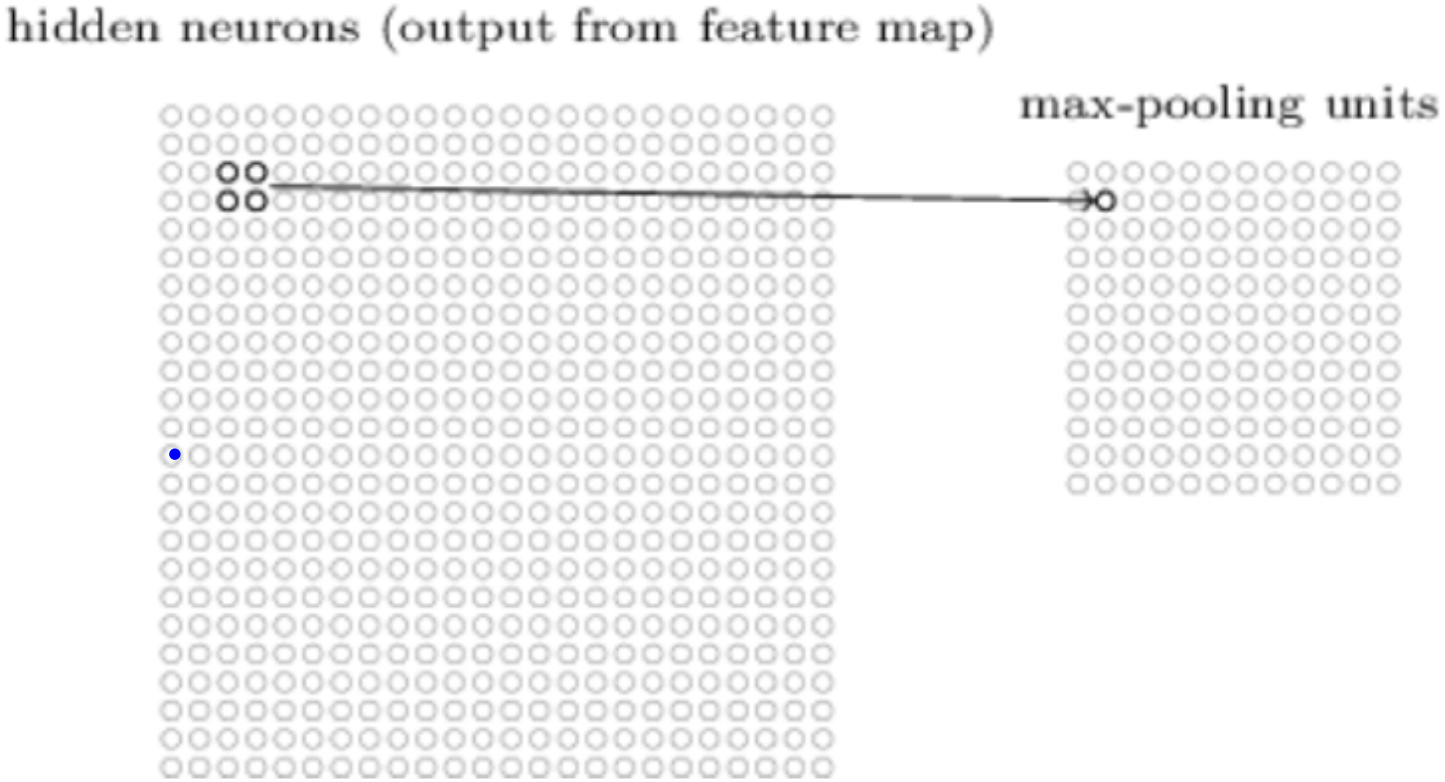
520 vs. 23550

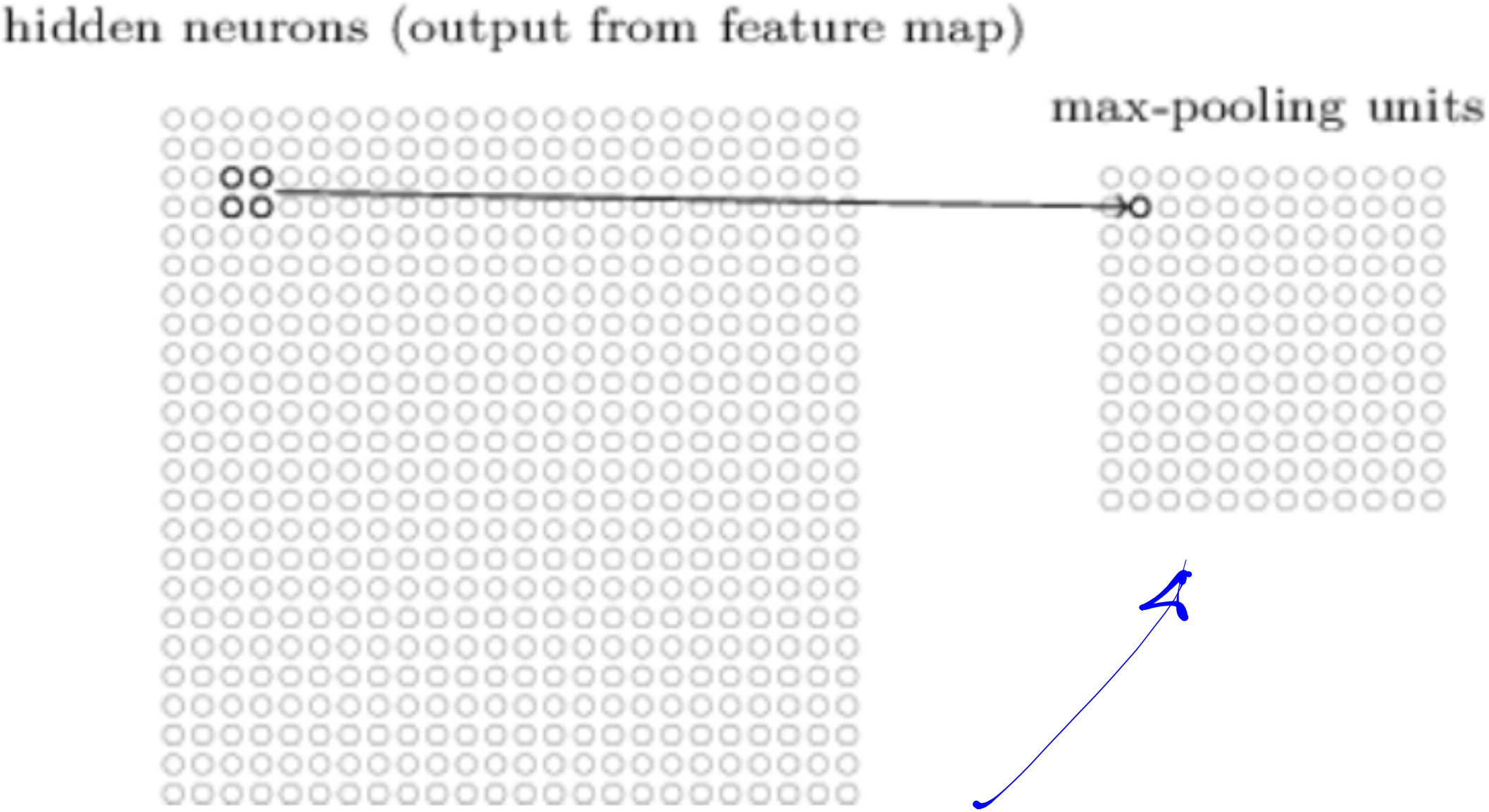
(CNN) (Fully connected neural network)

ABOUT 40 times more

CNN also contains pooling layer.

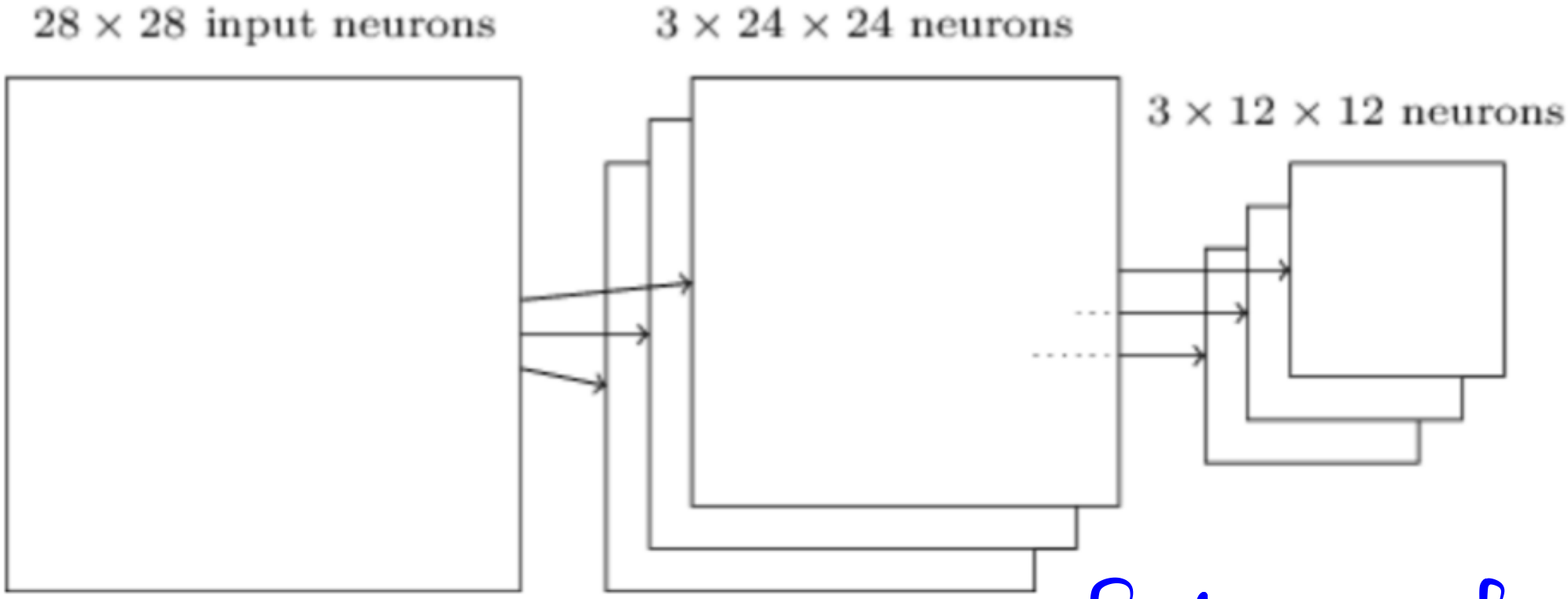
It simplifies the information in the output from the convolutional layer.





After pooling: 12 x 12 neurons

- Normally Convolutional layer involves more than → one feature map.



- Apply Max pooling to each feature map separately.

Max pooling tries to understand whether a given feature is there anywhere in the region of the image.

Throw away the exact positional information.

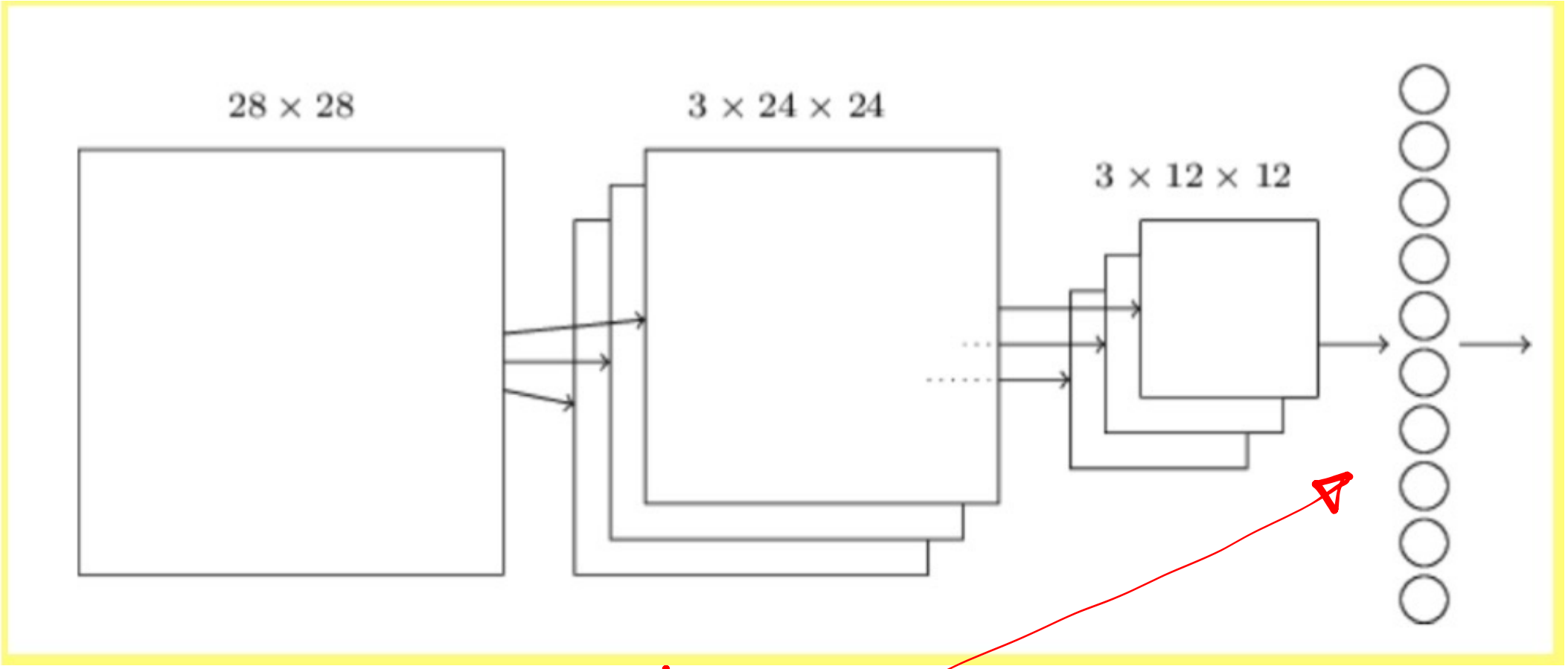
Rationale: Once a feature is found, exact location is not very important as its rough location relative to other features is.

Alternative to max-pooling

L2 pooling:

$$\sqrt{\text{Sum of the squares of activations in the } 2\times 2 \text{ region taken}}$$

Putting it all together:



Final layer is fully connected.

Connects every neeuron from the max-pooled layer to every one of 10 output neurons.

Back propagation in this context.

Earlier derivation was for fully connected NN.

It should be straightforward to adopt it for convolutional and max-pooling layers. (Task).

Characteristics of Convolutional Network:

- Patterns learnt are translation invariant
  [Once found a pattern at top left, it can identify the pattern if it is at right-bottom corner
- As opposed to, a densely connected NN would have to learn the pattern anew.

- So Convnet is efficient for processing images as visual world is fundamentally translation invariant.

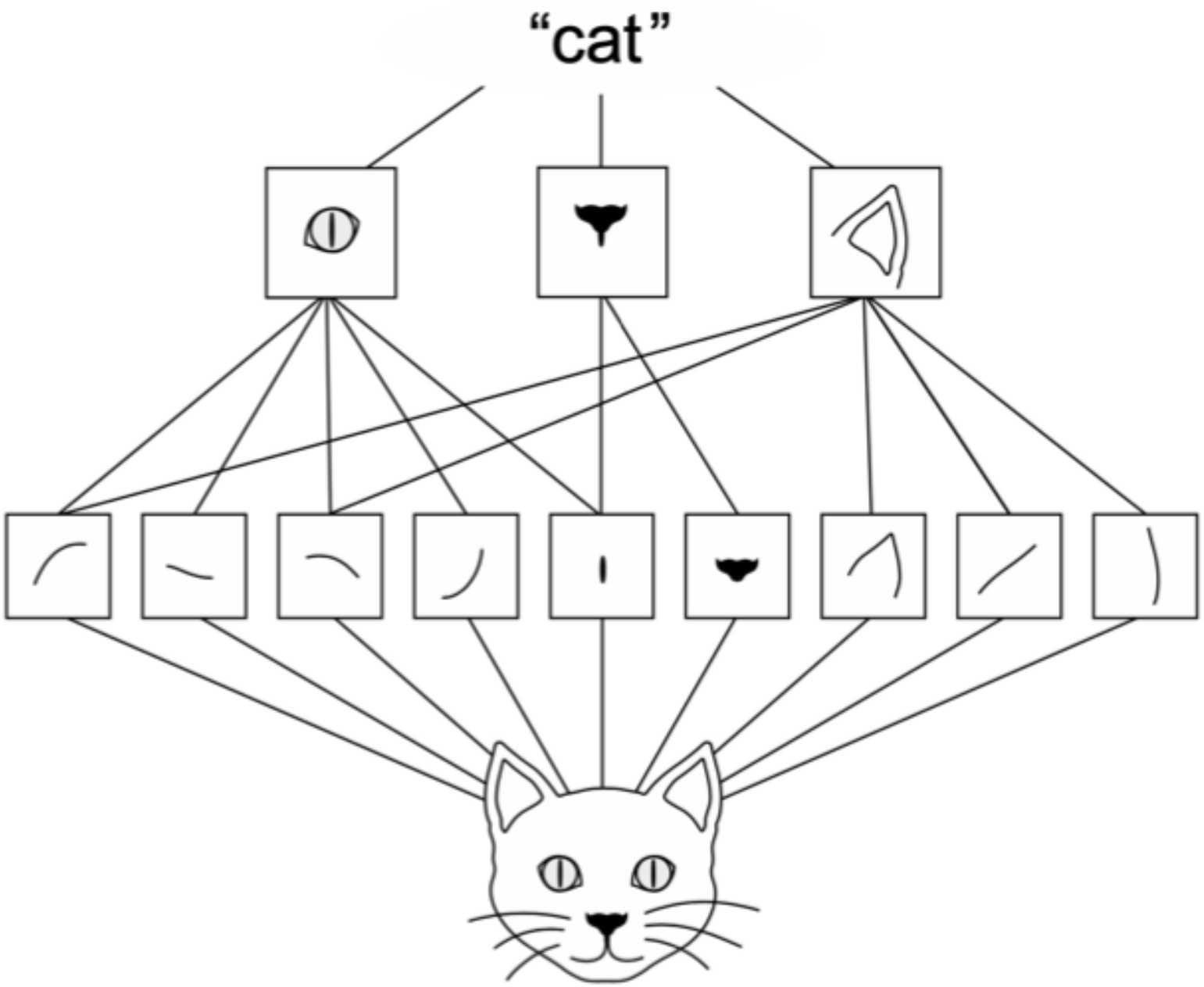
They can learn spatial hierarchy of patterns.

First convolutional layer learns local patterns such as edges

Second ... patterns made up of patterns from the first layer

and so on.

Allows to learn increasingly complex and abstract visual concepts.


“cat”

Recurrent Networks:

- Recurrent networks are artificial neural networks
- applies to time series data
- uses output of network units at time t as input to other units at time $t+1$

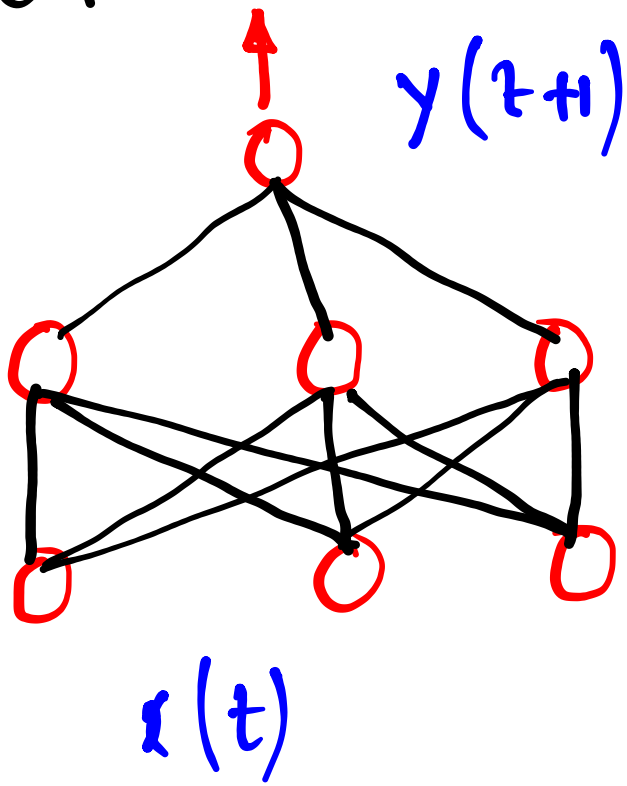
Thus, support a form of directed cycles in the network.

Illustration with an example problem

To predict next day's stock market

Average $y(t+1)$ based on current day's stock market value $x(t)$.

Suppose, such a time series data is given.
As an obvious choice, we can train a
feedforward network to predict $y(t+1)$
as its output based on the input values
$x(t)$.



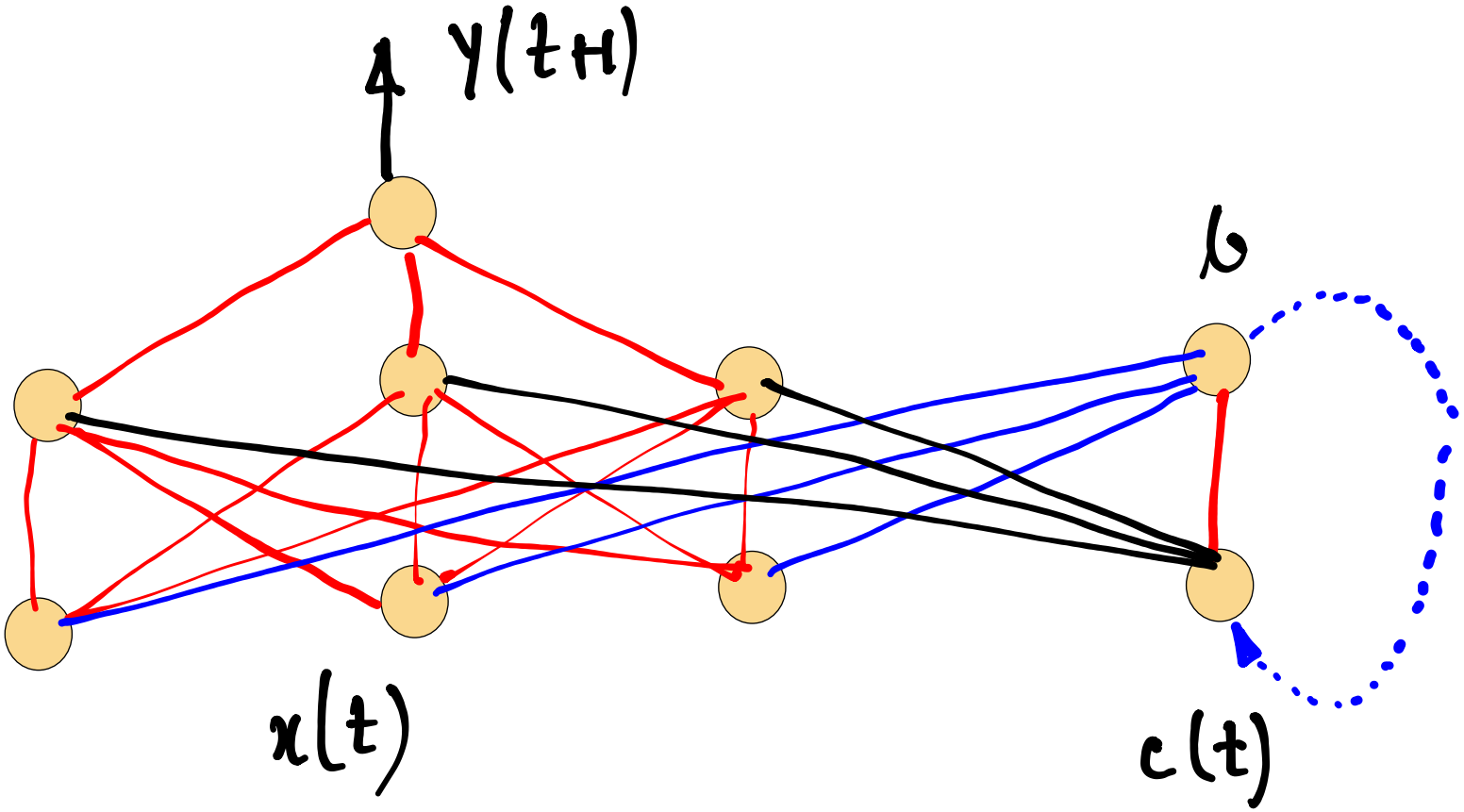
A limitation of this model will be prediction of $y(t+1)$ depends on only $x(t)$. It does not capture the possible dependencies on the earlier values of x.

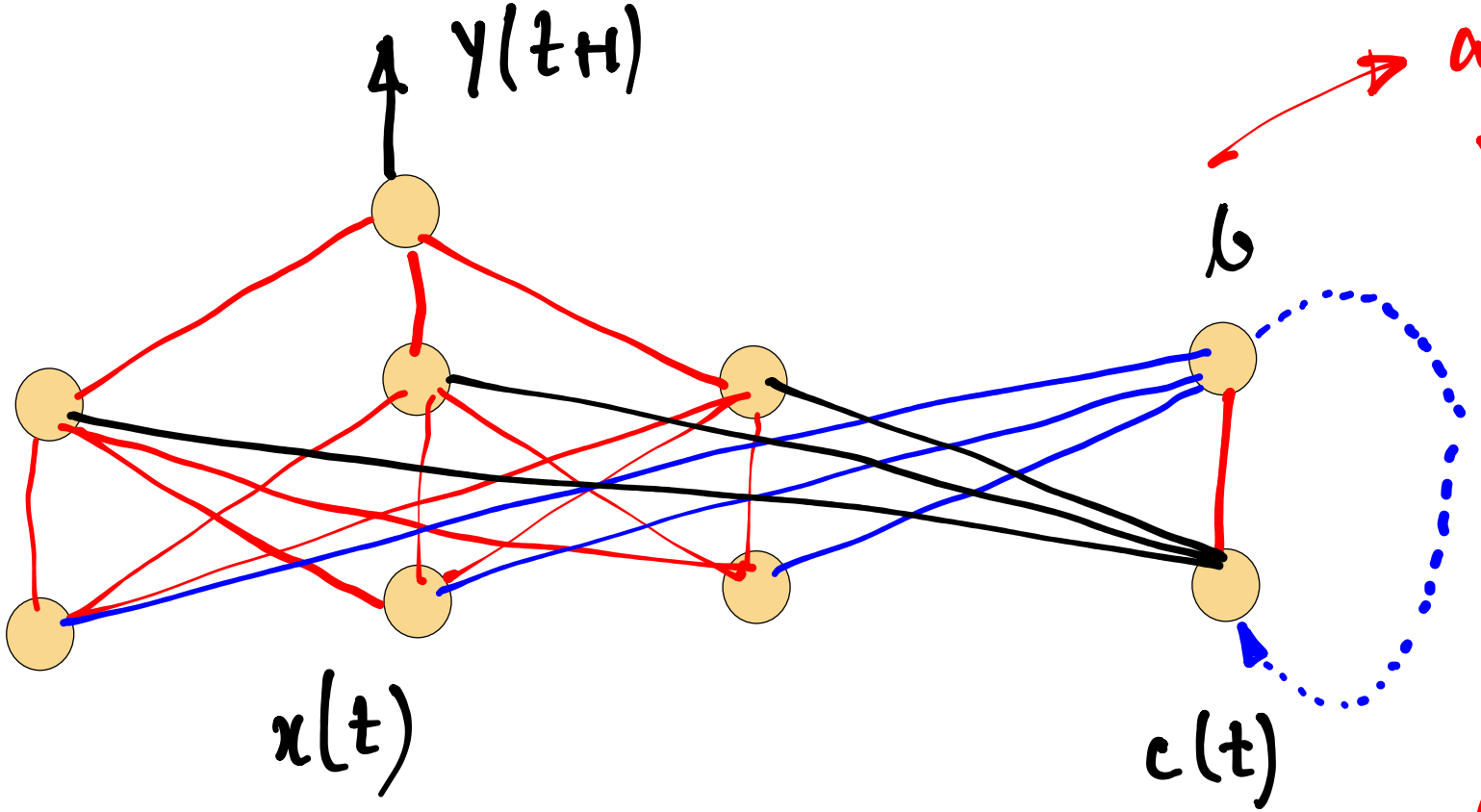
- This may be necessary in cases of stock market prediction for tomorrow, it may be depend upon $x(t)$ $x(t-1)$ $x(t-2)$ combined together in a complex way.

One solution may be

to use $x(t)$, $x(t-1)$, & $x(t-2)$ as input to the feedforward network.

However, if we want to consider an arbitrary window (say 7 days or 10 days) we will require a different solution.

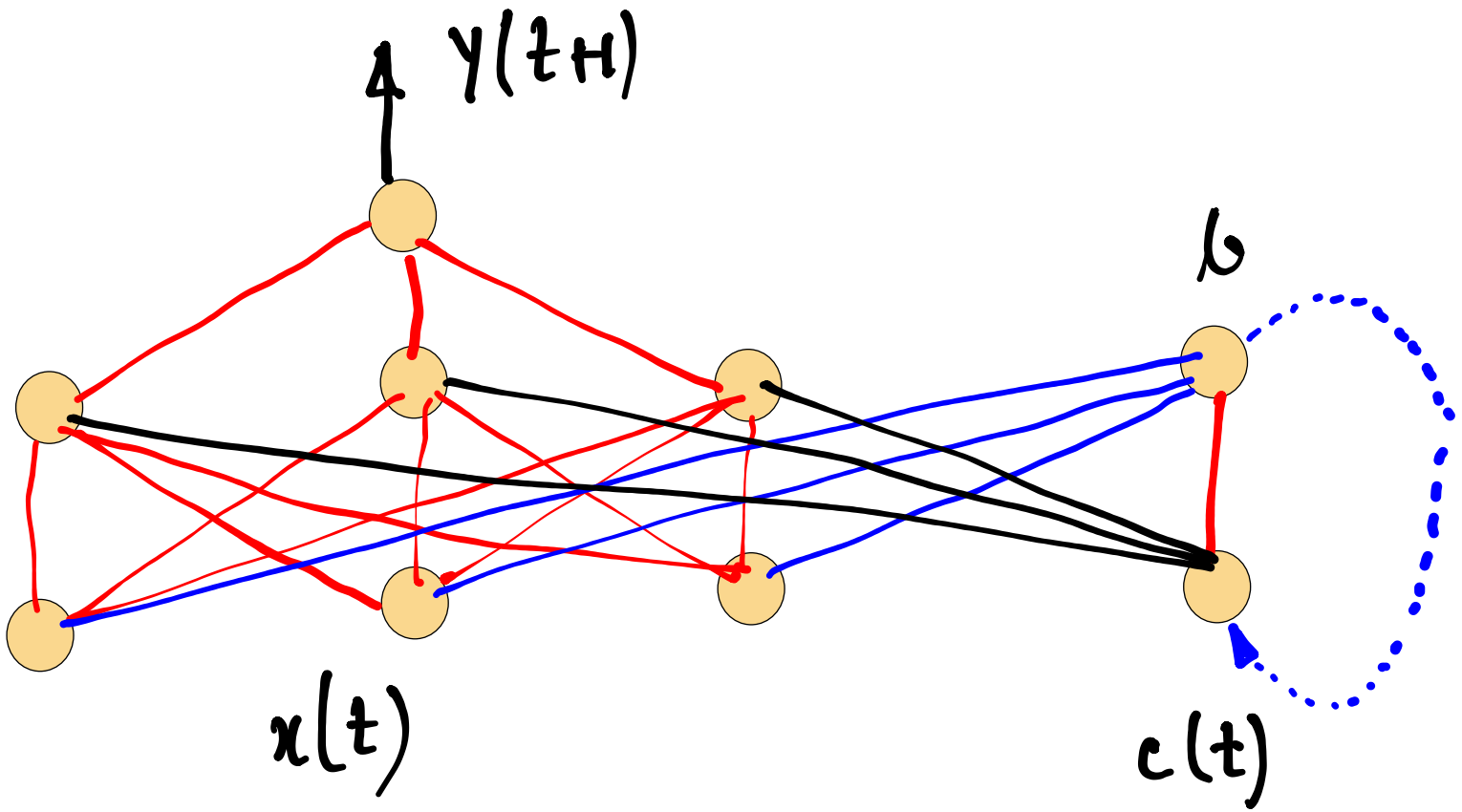

y(t+1)
b
x(t)
c(t)



add a hidden neuron b

add a new input neuron c(t).

Input value c(t) = value of b at previous time step.

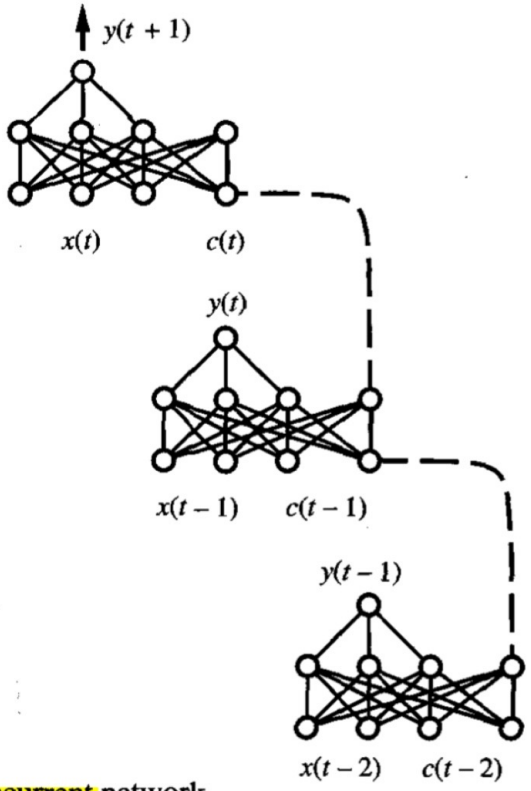


[c(t) = h at t-1]

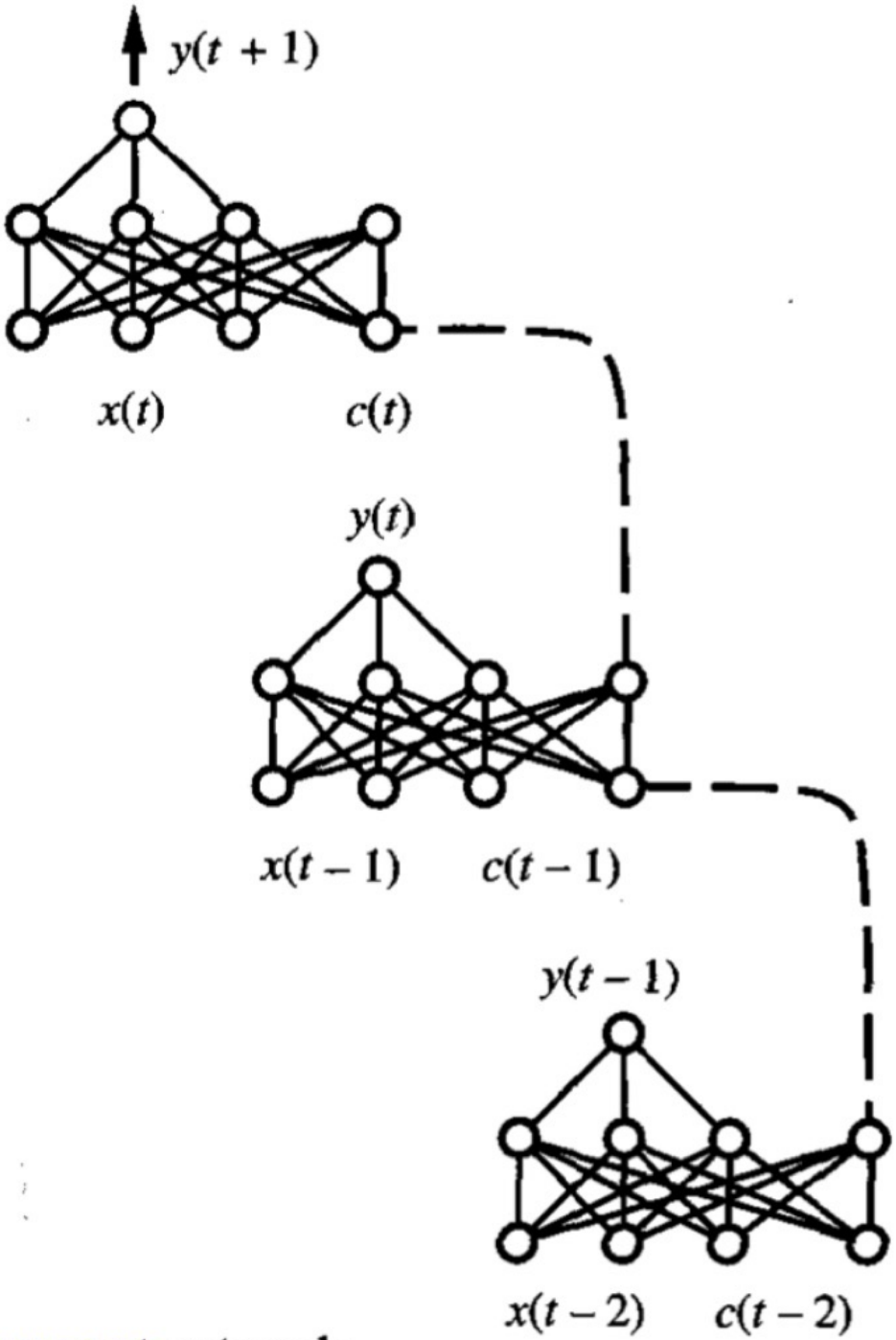
This implements a recurrence relation. h represents the history of network inputs. h summarizes information from earlier values of x (arbitrarily distant in time).

How to train a recurrent network?

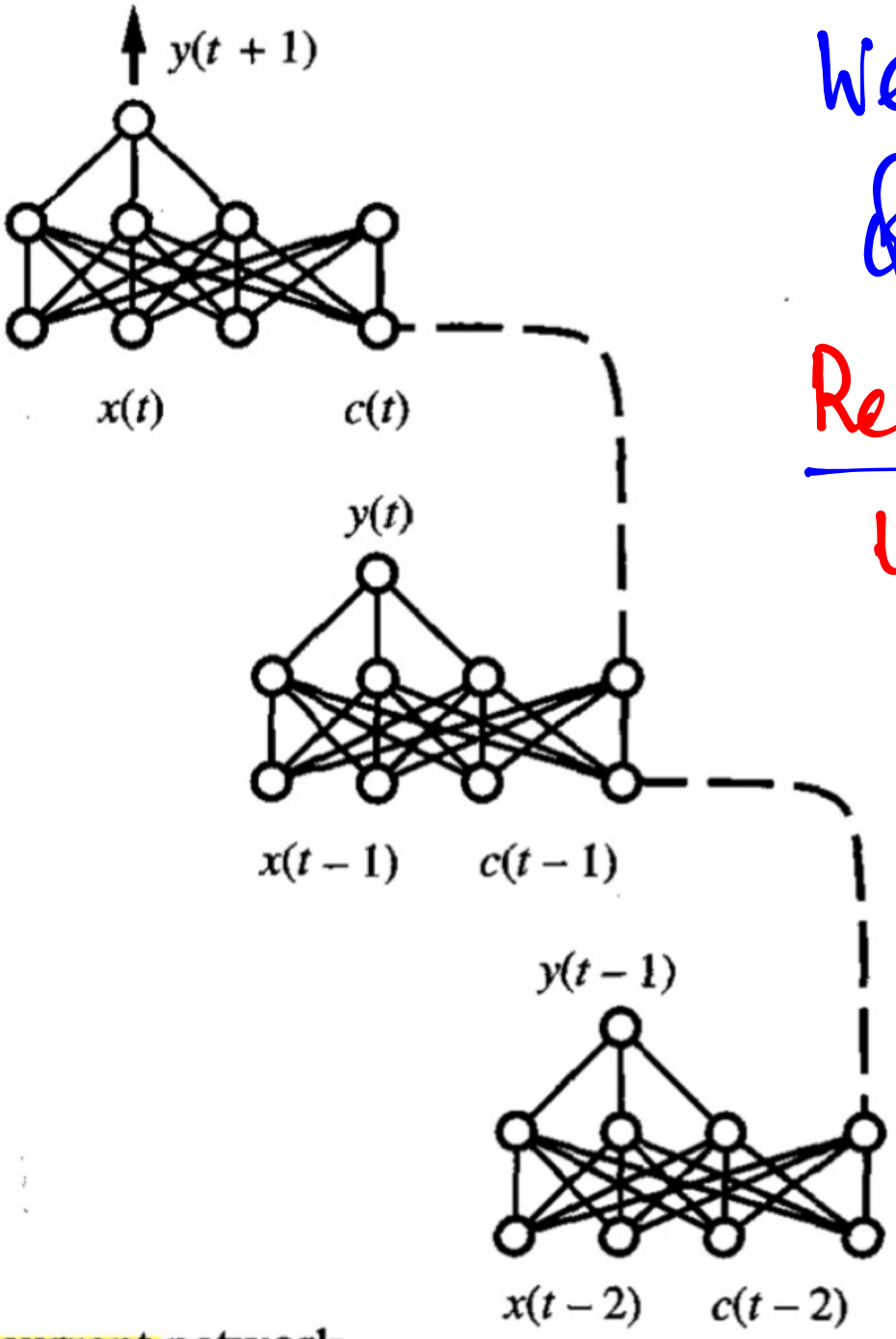
Consider the data flow of the recurrent network unfolded in time.



Recurrent network unfolded in time



Recurrent network
unfolded in time



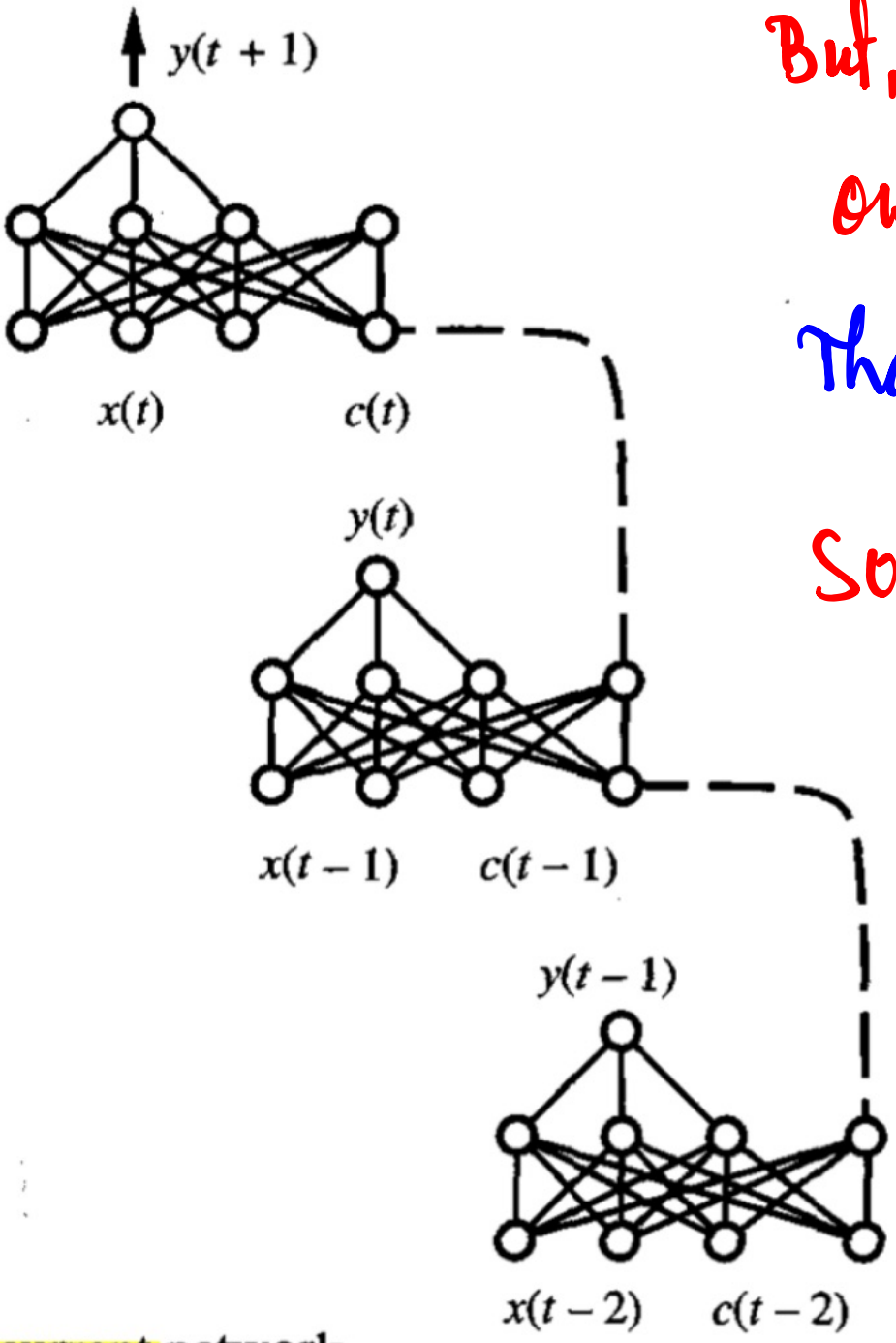
Recurrent network unfolded in time

We have made several copies of the recurrent network.

Replaced the feedback loops by connections between different copies.

There is no cycle..

So, the weights of the unfolded network can be trained by back propagation.



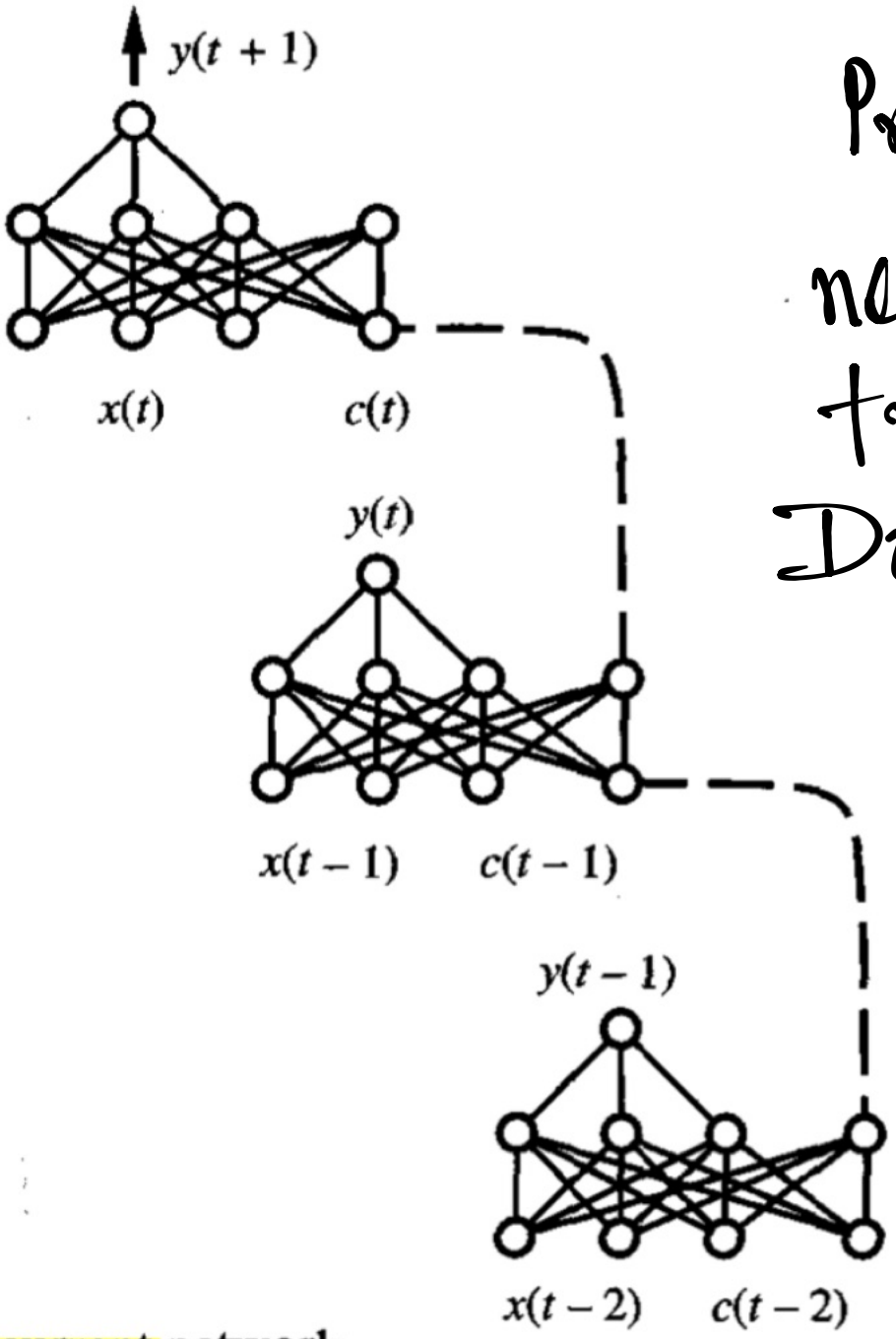
Recurrent network unfolded in time

But, in practice, we want to keep only one copy of the network.

That is, only one set of weights.

So, after training the unfolded network

Final weight w_{ji} = mean of corresponding w_{ji} in different copies.



Recurrent network unfolded in time

Practically, recurrent networks are difficult to train.
Does not generalize reliably.

However, recurrent network remains important due to their increased representational power.

Recurrent Neural Network: (RNN)

. A type of ANN which uses sequential or time series data.

Commonly used for:

language translation

NLP (Natural Language Processing)

Speech recognition etc.

Popular applications:

Siri, google translate, voice search

Ref. IBM

RNN:

Distinguished by their 'memory' as they take information from prior inputs to influence the current input and output.

RNNs use BPTT (Backpropagation Through Time).

BPTT for RNN:

Principle is same as traditional backpropagation

RNNs share parameters across layers.

BPTT sum errors at each step.

This may lead to problems:

1. exploding gradients →
2. vanishing gradients →

Vanishing gradients:

Gradient (slope of the loss function along the error curve) is too small.

Then if continues to be smaller, updating the weight parameters very insignificantly

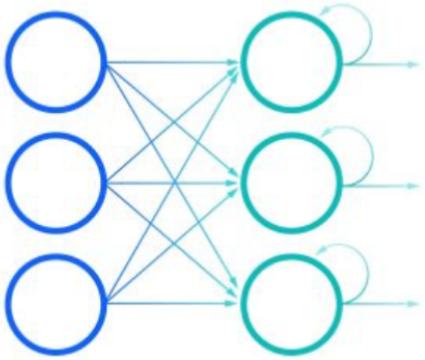
We can say that the algorithm is no longer learning.

Exploding gradients

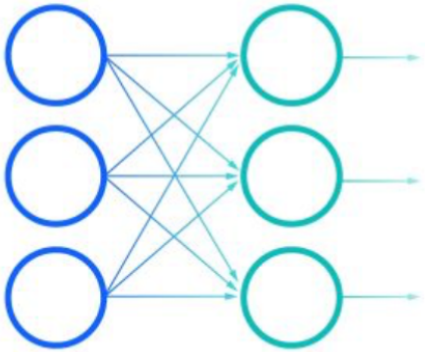
Gradient is too large → unstable model.

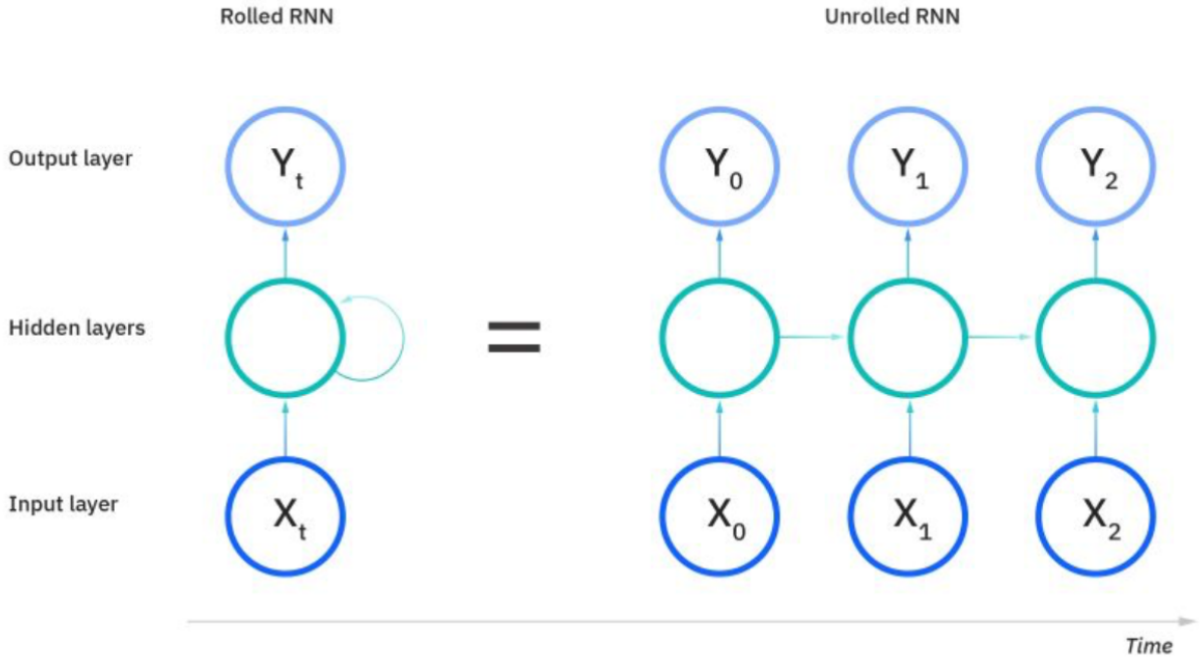
Model weights will grow too large → eventually represented by NaN.

Recurrent Neural Networks



Feedforward Neural Networks





Types of RNN:

Different types of RNNs are used for different purposes:

music generation

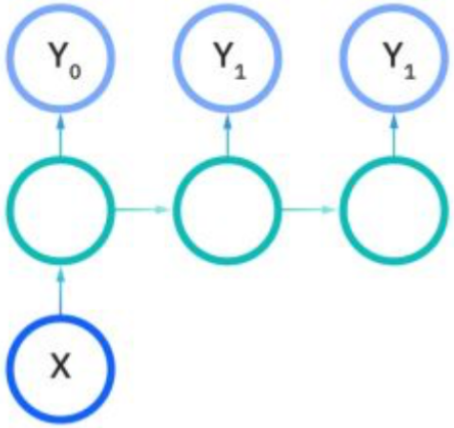
sentiment classification

machine translation.

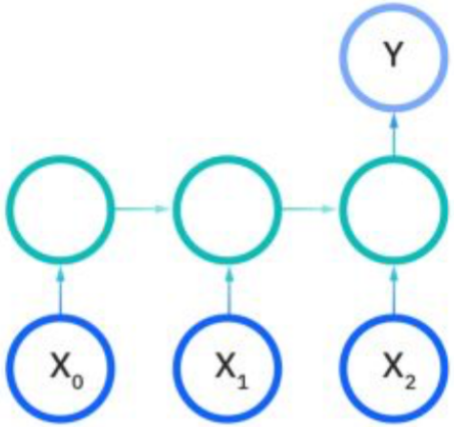
One-to-one, One-to-many, Many-to-one, Many-to-many.

One-to-one

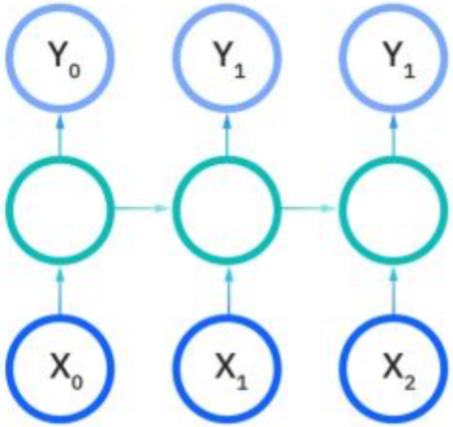
One-to-many



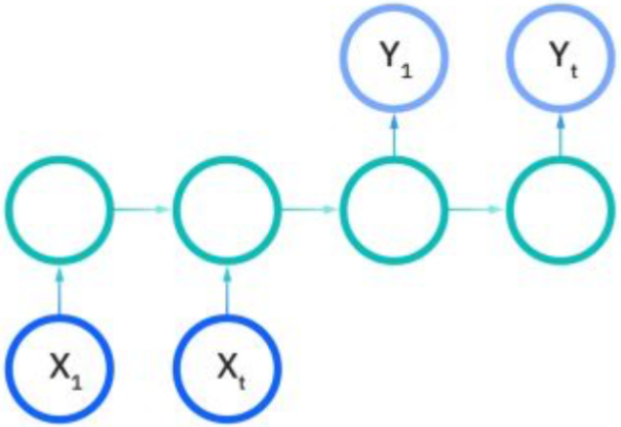
Many-to-one



Many-to-many

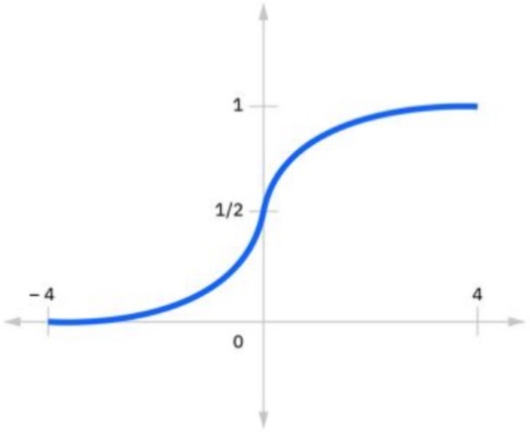


Many-to-many

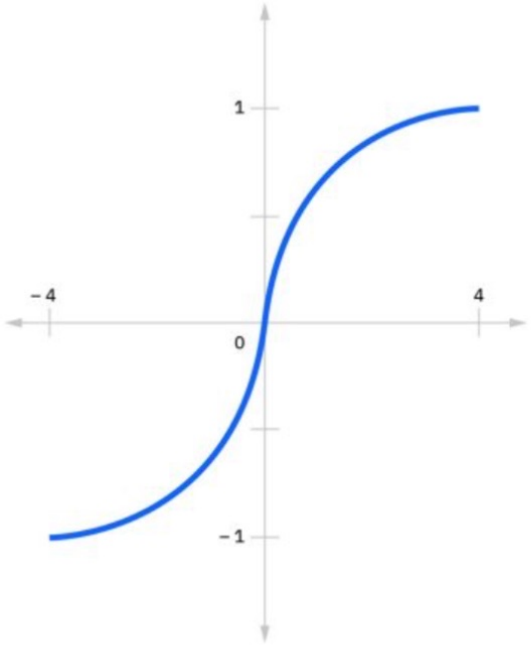


Common Activation functions used:

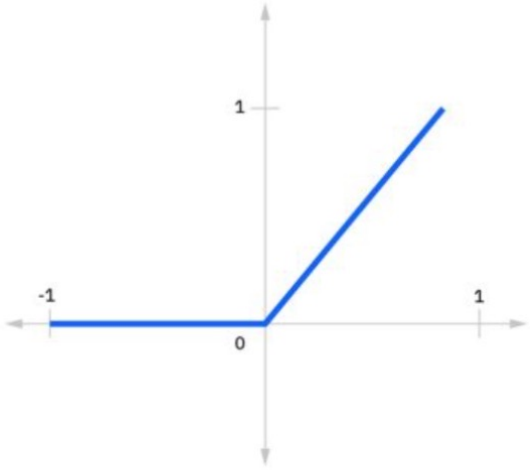
$$g(x) = \frac{1}{1 + e^{-x}}$$



$$g(x) = \frac{e^{-x} - e^{-x}}{e^{-x} + e^{-x}}$$



$$g(x) = \max(0, x)$$



Different RNN architectures:

1. Bidirectional RNN (BRNN)
2. Long short-term Memory (LSTM)
3. Gated recurrent units (GRU).