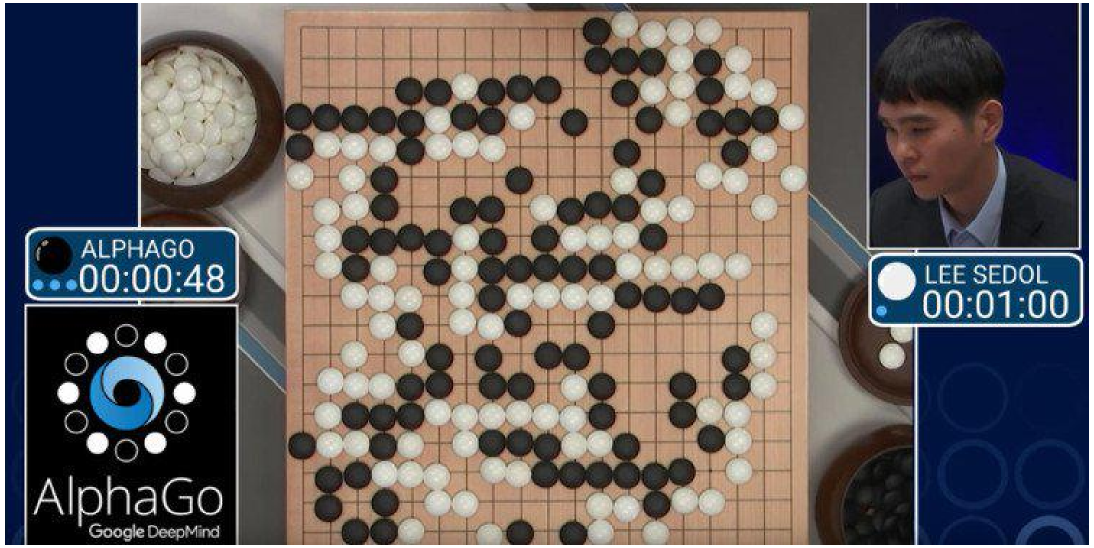
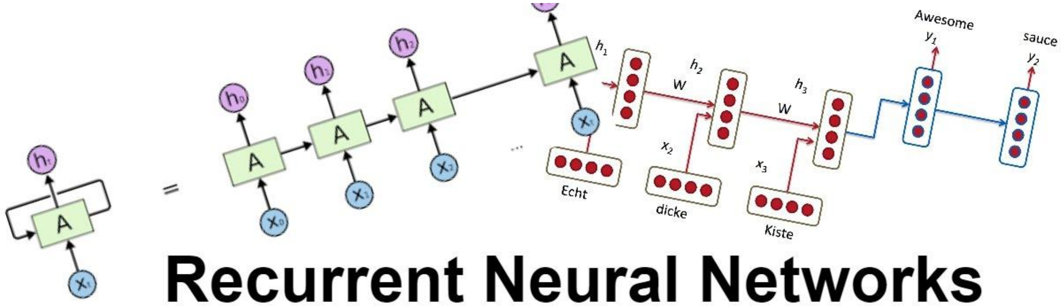
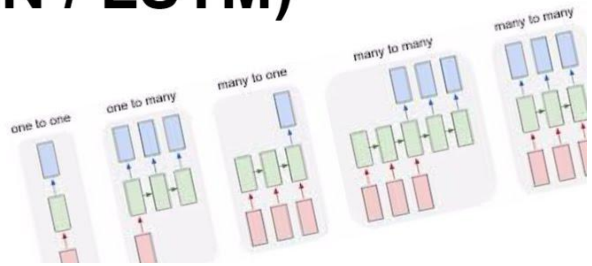
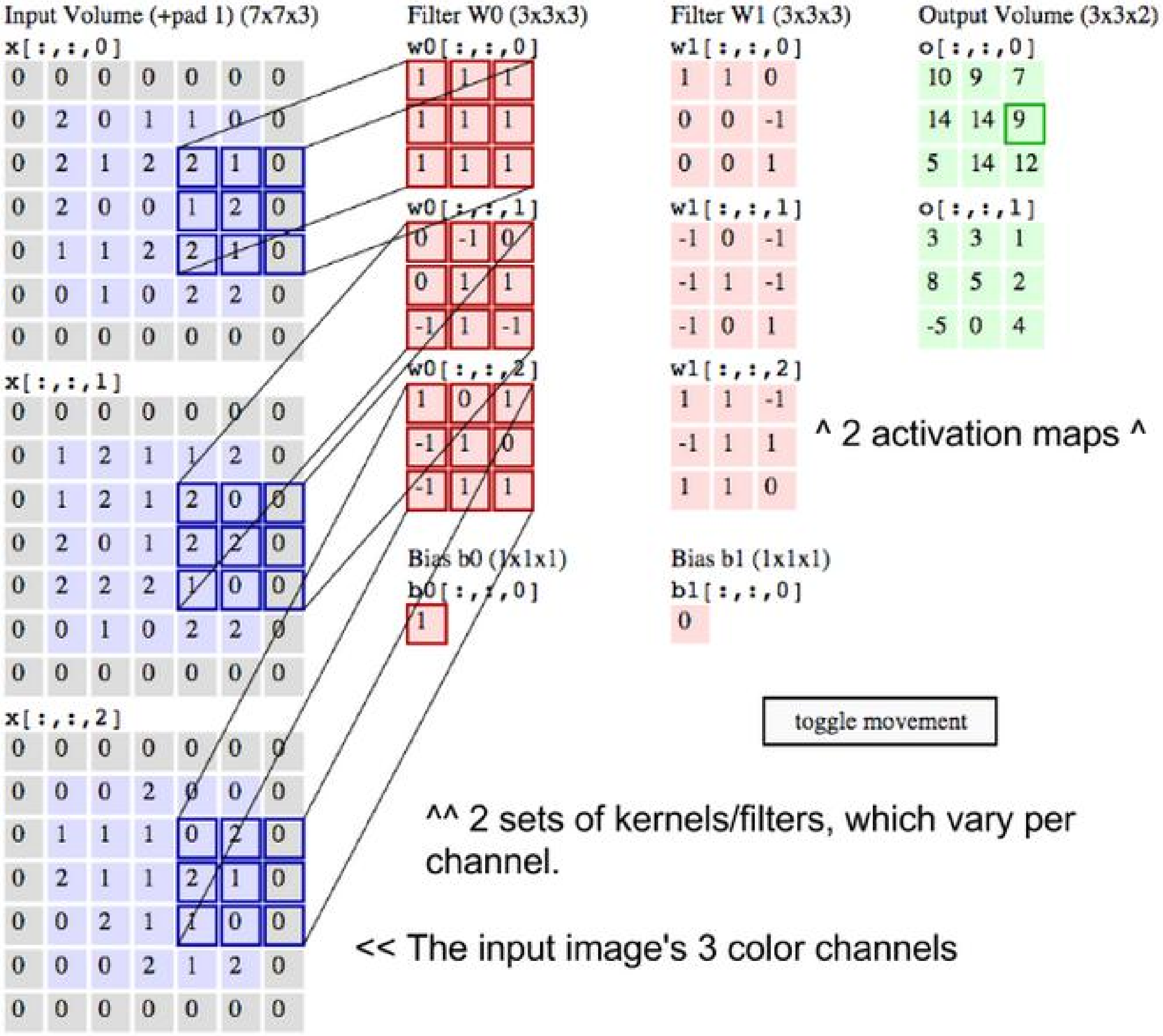
Deeplearning





Recurrent Neural Networks (RNN / LSTM)



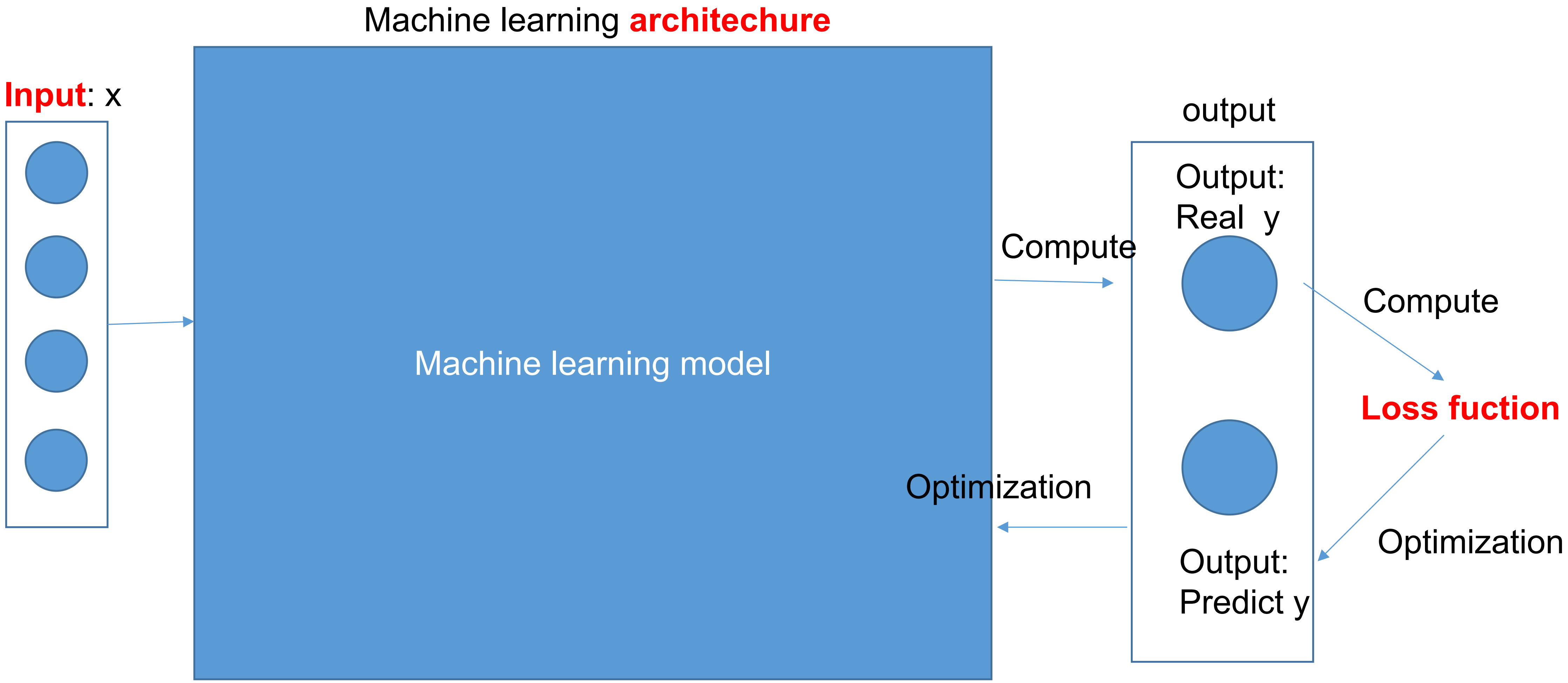


^ 2 activation maps ^

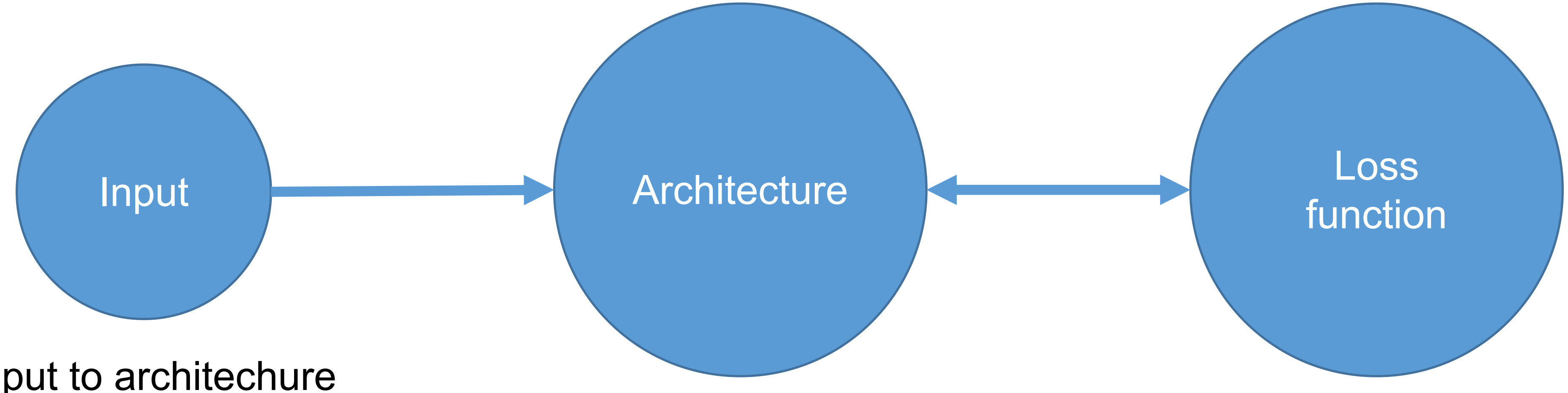
^^ 2 sets of kernels/filters, which vary per channel.

<< The input image's 3 color channels

Supervised learning



Supervised learning | Role of each part

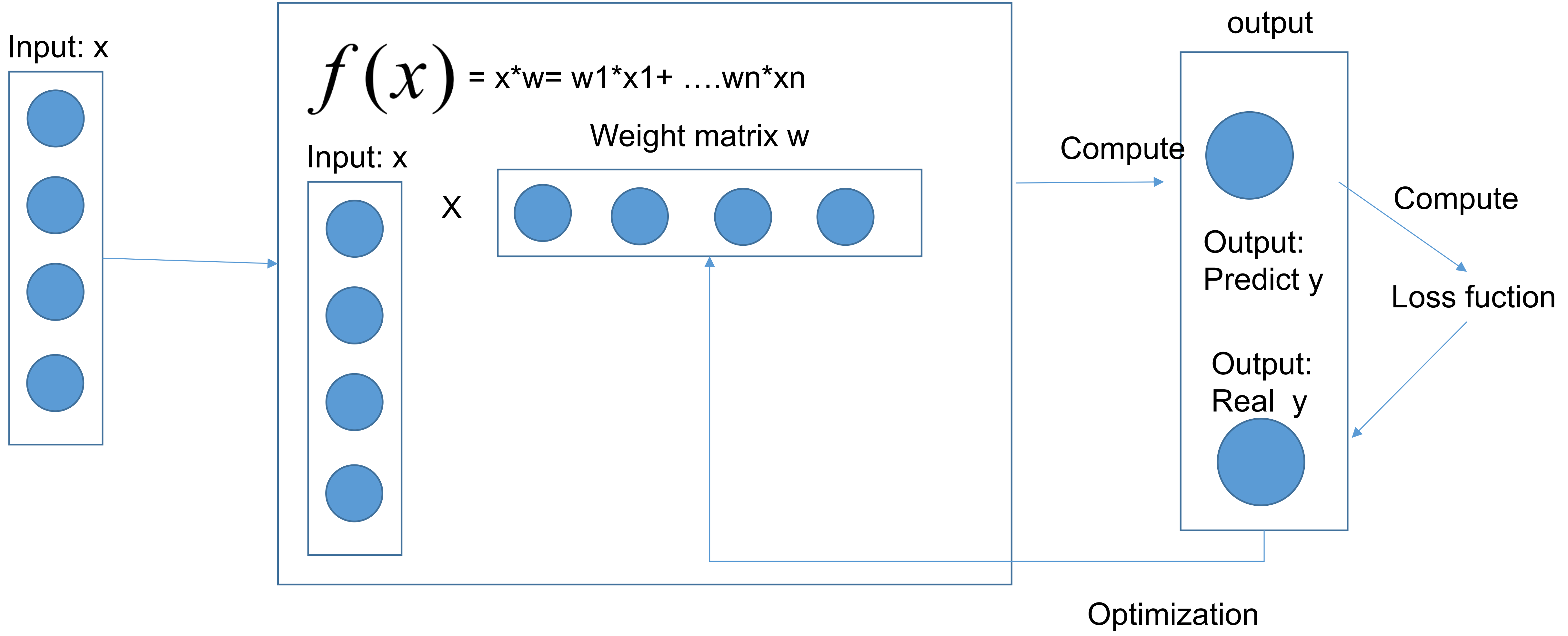


Input to architechure

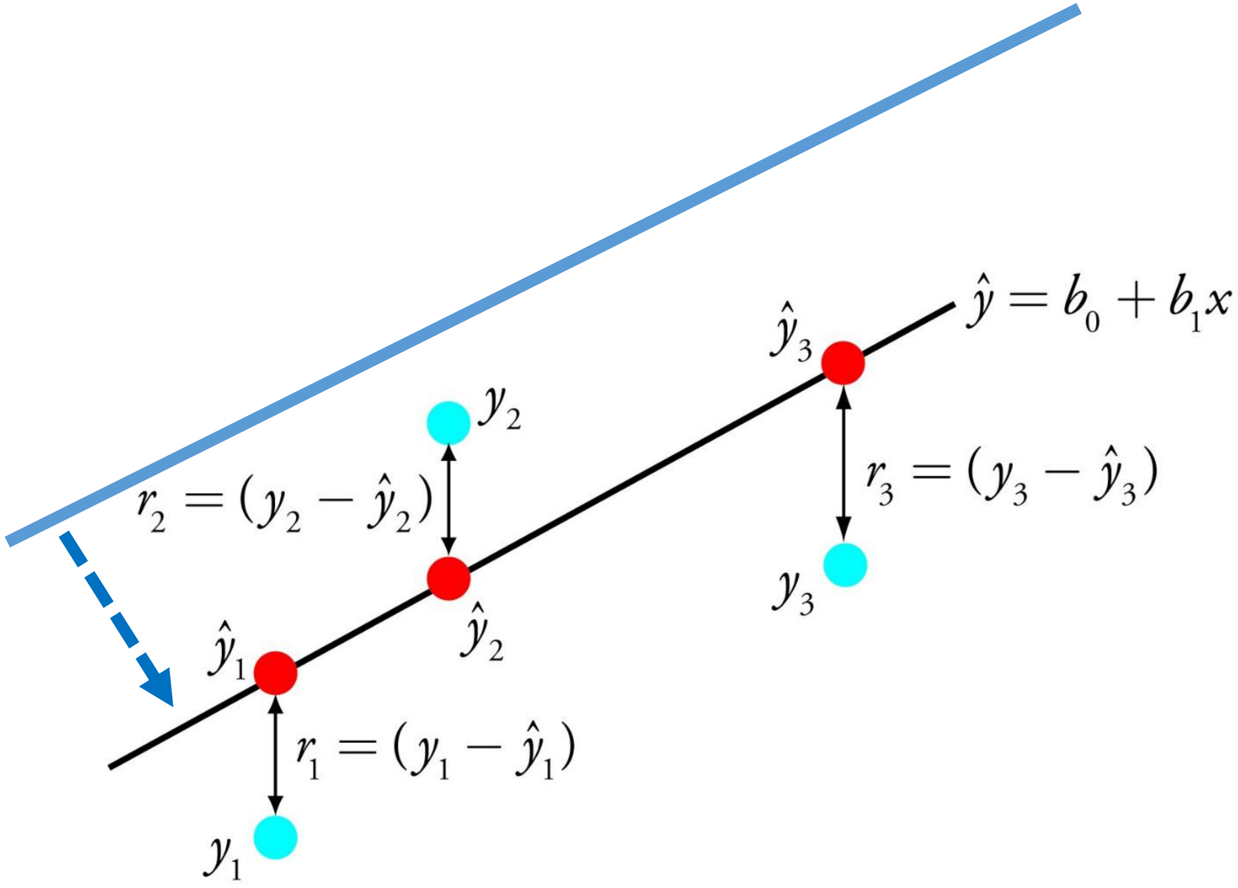
- Initialize weight matrix
- Compute output with input given
- Optimize Weights base on "signal" of loss function

- Compute loss between output and target.
- Tell Architecture how good did it work
- Tell Architechture how to optimize weights

Supervised learning | Linear regression



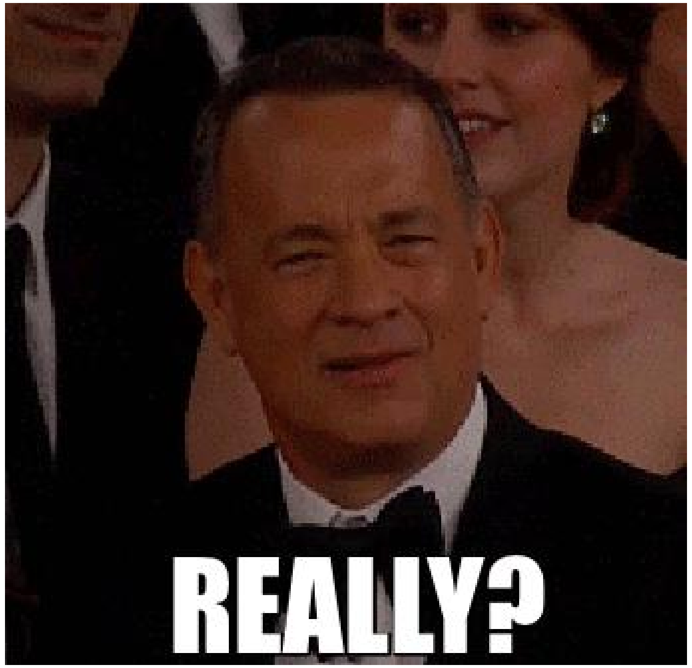
Supervised learning | Linear regression



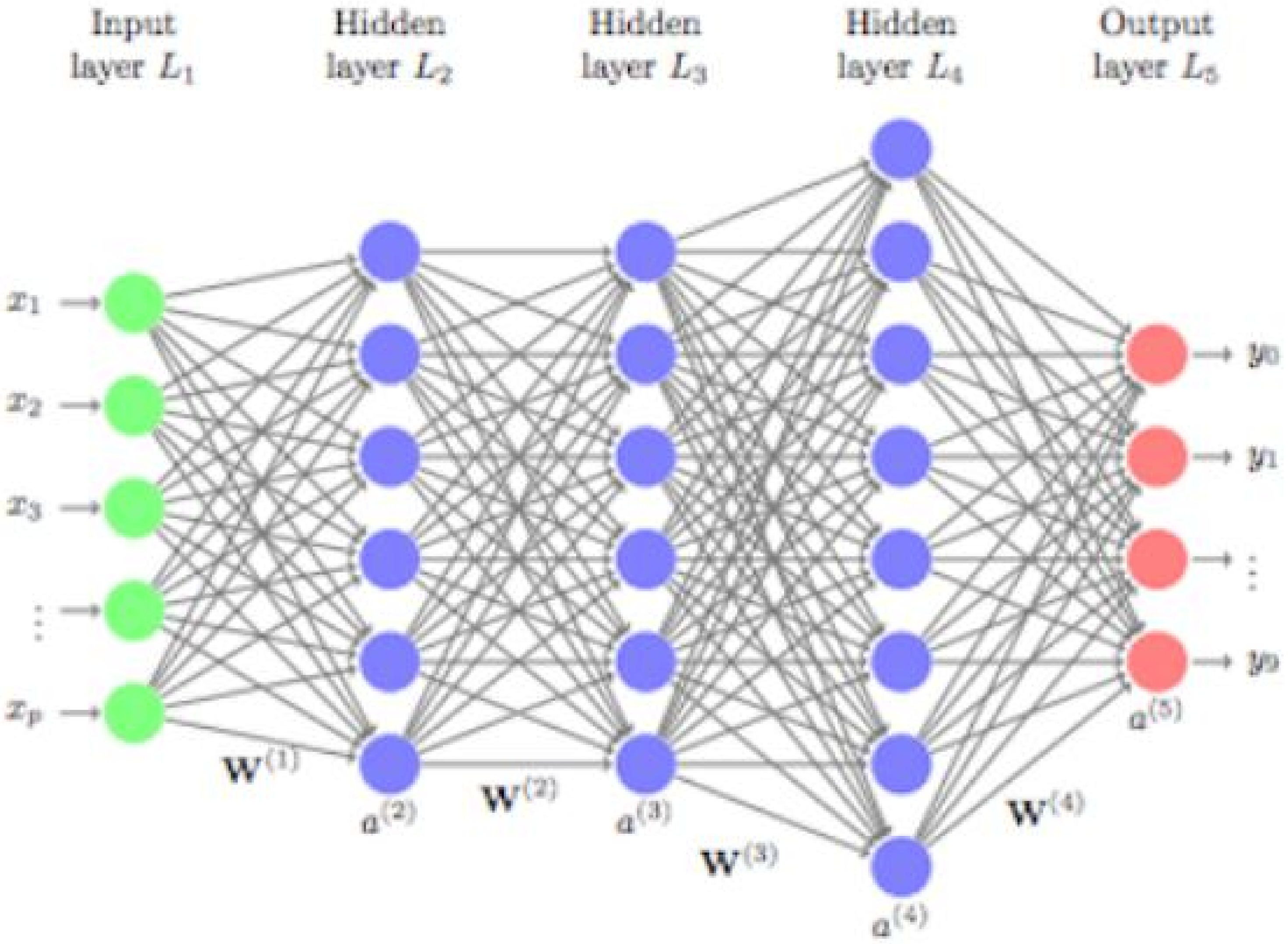
Loss fuction

$$Loss(y, \hat{y}) = (y - \hat{y})^2$$

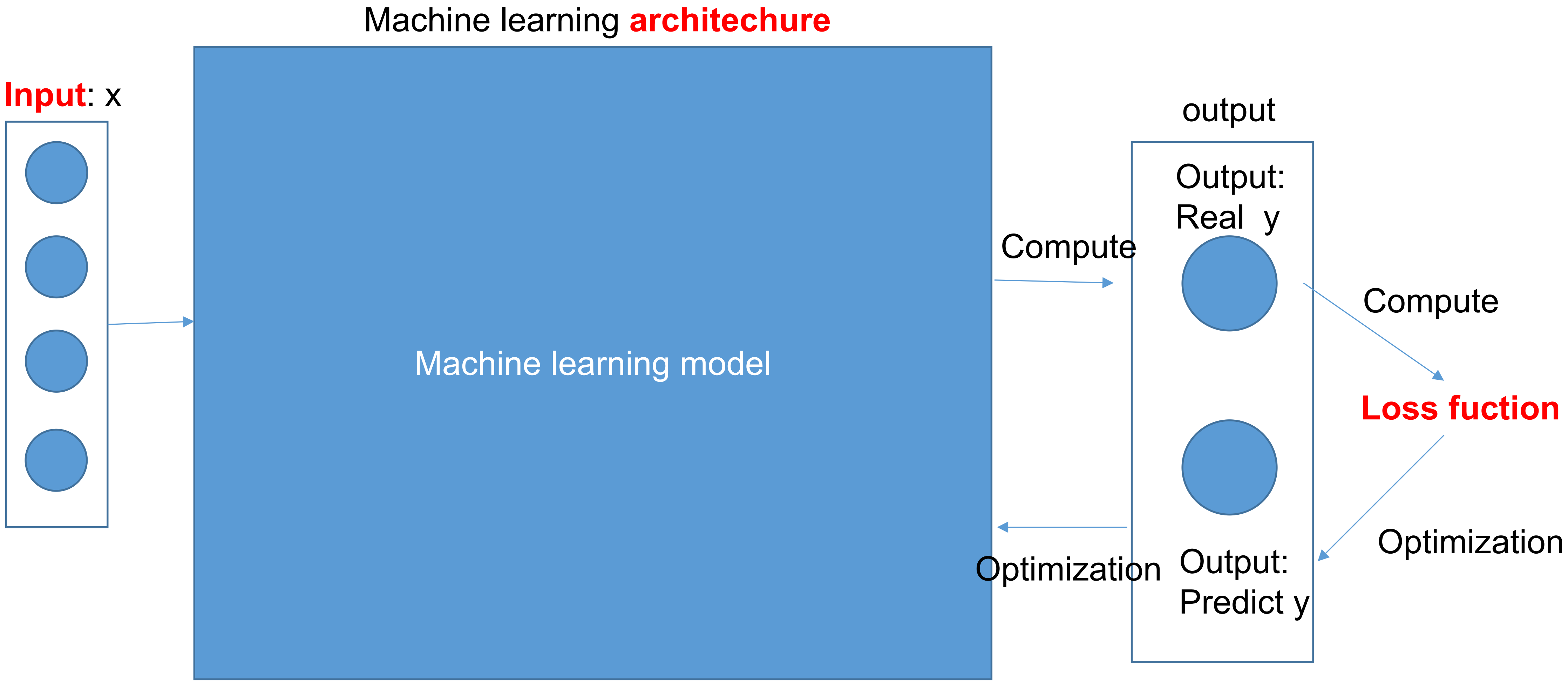
So, we already know all what we need!



But, what about thing that people call “deep learning”?



It nearly the same with what we saw before



Lets deep dive in to architechture

In linear regression, w - weight is a vector

$f(x)$ = x*w = y
= [1,n]*[n,1] = [1,1] -scalar

If w is a matrix – what happen?

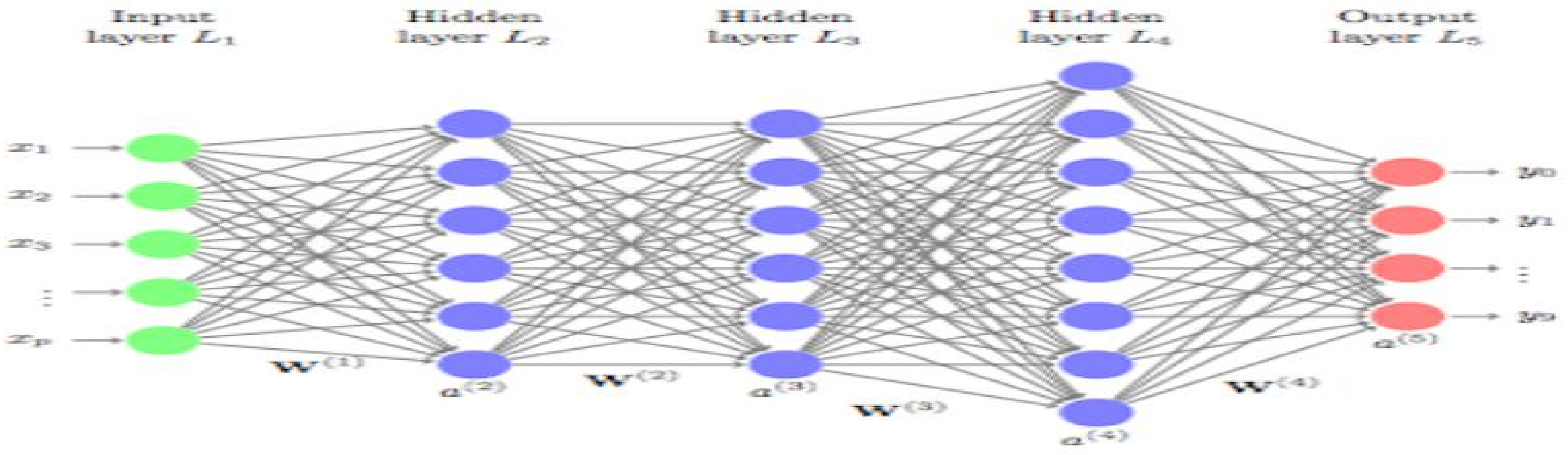
$f(x)$ = x*w = y
= [1,n]*[n,m] = [1,m] - vector

If w and even x is a matrix – what happen?

$f(x)$ = x*w = y
= [n,m]*[m,k] = [n,k] - matrix

Lets deep dive in to architechture

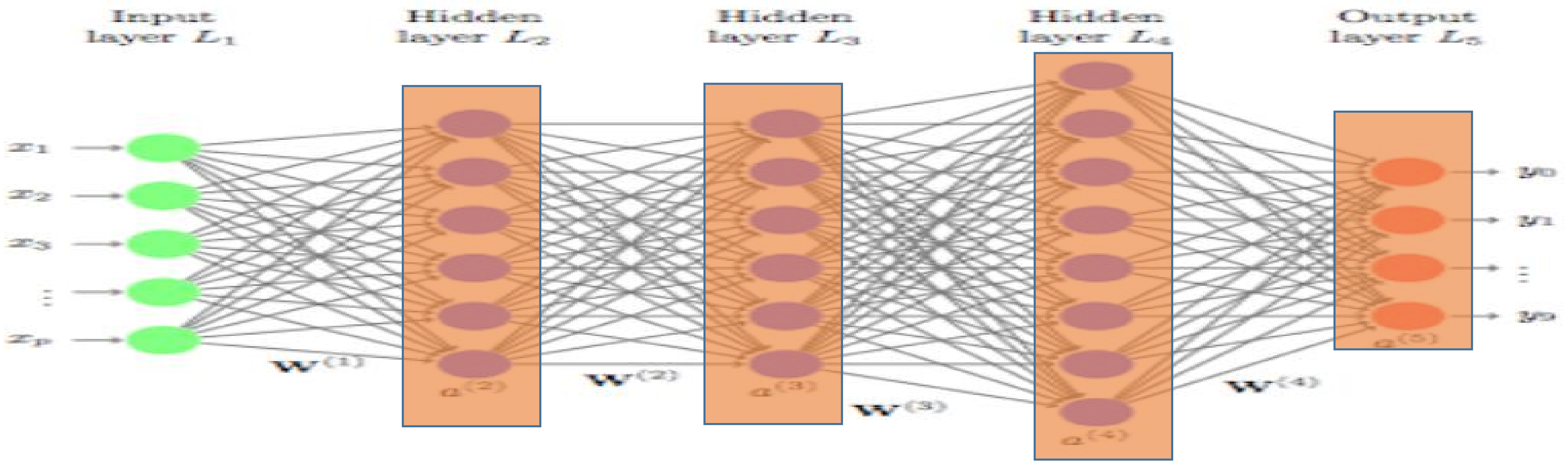
Just simple - “Deep learning” is a nested f(x)



X	**f(x)**	**g(f(x))**	**h(g(f(x)))**	**k(h(g(f(x))))**
1x5	**5x6**	**6x6**	**6x8**	**8x4**

Lets deep dive in to architechture

And all others is features



X	f(x)	g(f(x))	h(g(f(x)))	k(h(g(f(x))))
1x5	5x6	6x6	6x8	8x4

But what about

convolutional neural net
Recurrent neural net
Long shot term memory
Sequence-to-sequence
Deep pyramid convolution
Recurrent convolutional net
Very deep long shot term memory
….

It is how we combine f(x) together – like lego

Let’s begin with type of input x

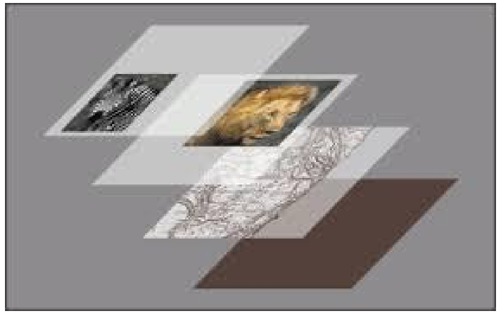
x

Vector
Fixed size, un-order → Tabular data

Place	County	Phone code	Approx. population
Basingstoke	Hampshire	01256	82913
Brighton	East Sussex	01273	155919
Carlisle	Cumbria	01228	103700
Huddersfield	Yorkshire	01484	146234
Luton	Bedfordshire	01582	203800
Nottingham	Nottinghamshire	0115	292400
Rhyll	Clwyd	01745	24889
Woking	Surrey	01483	62796

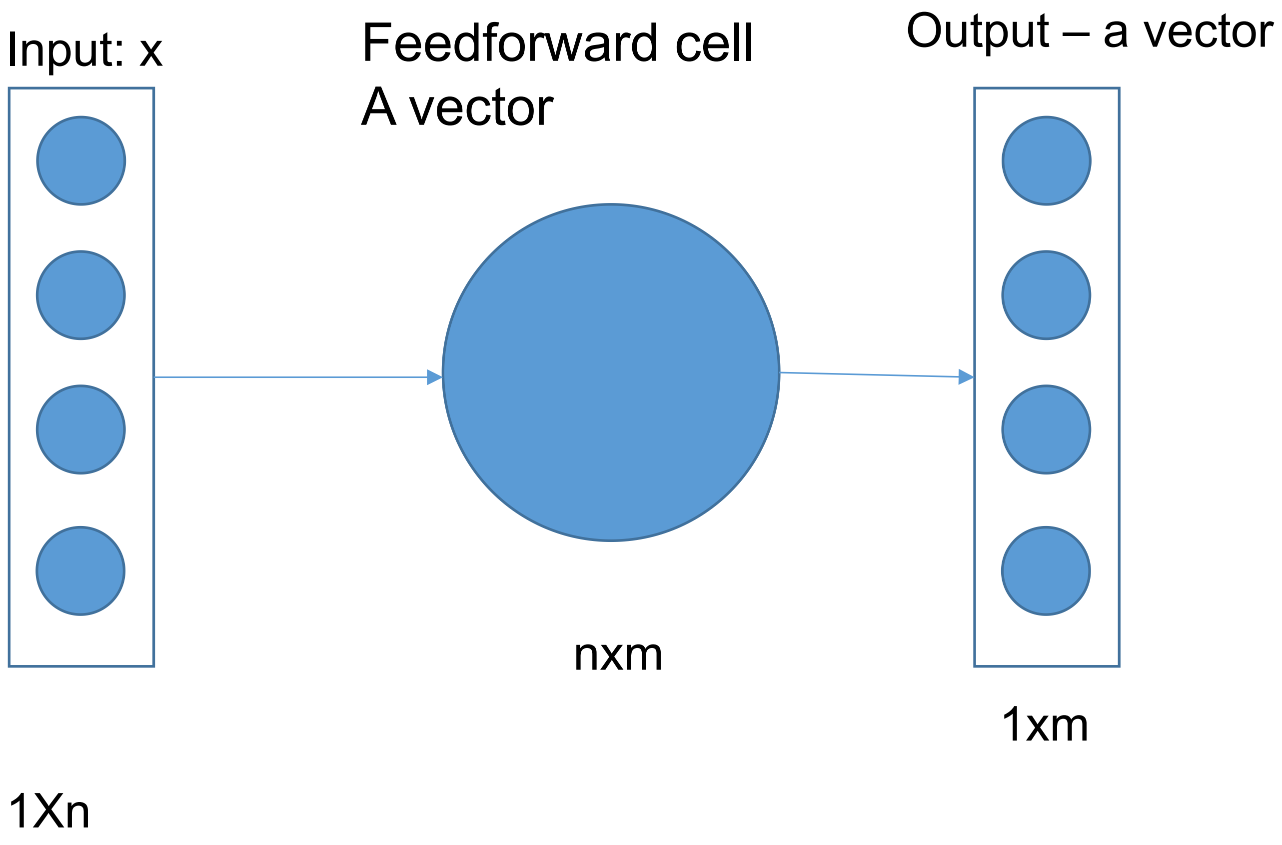
matrix

Fixed size, ordered → Image

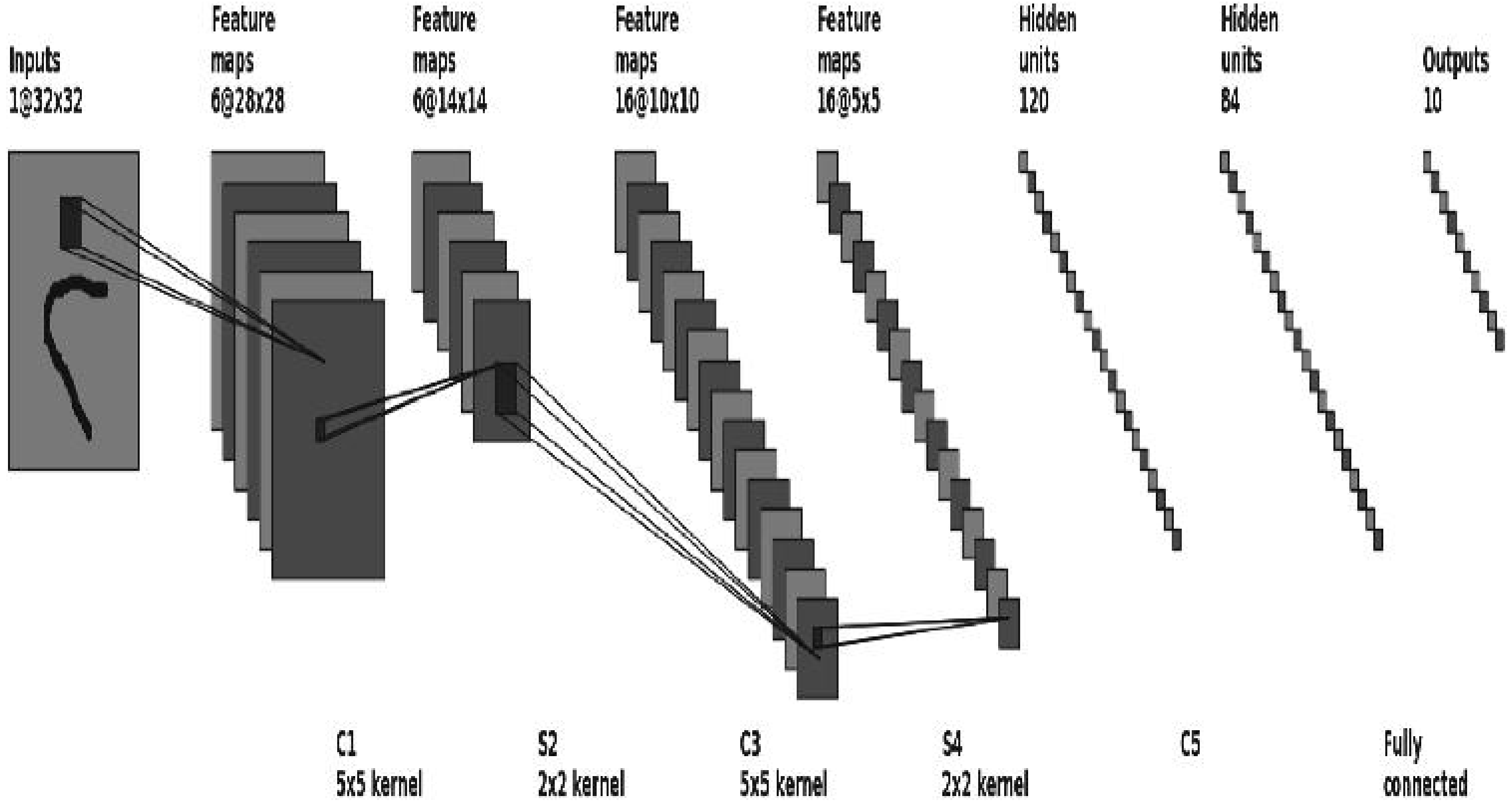


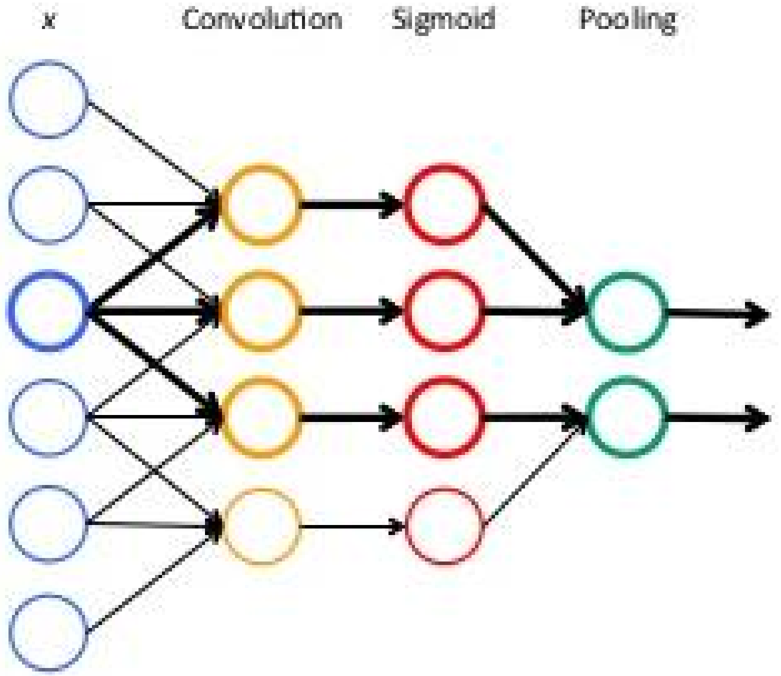
Not Fixed size, ordered → Text, voice, ..

Feed forward cell



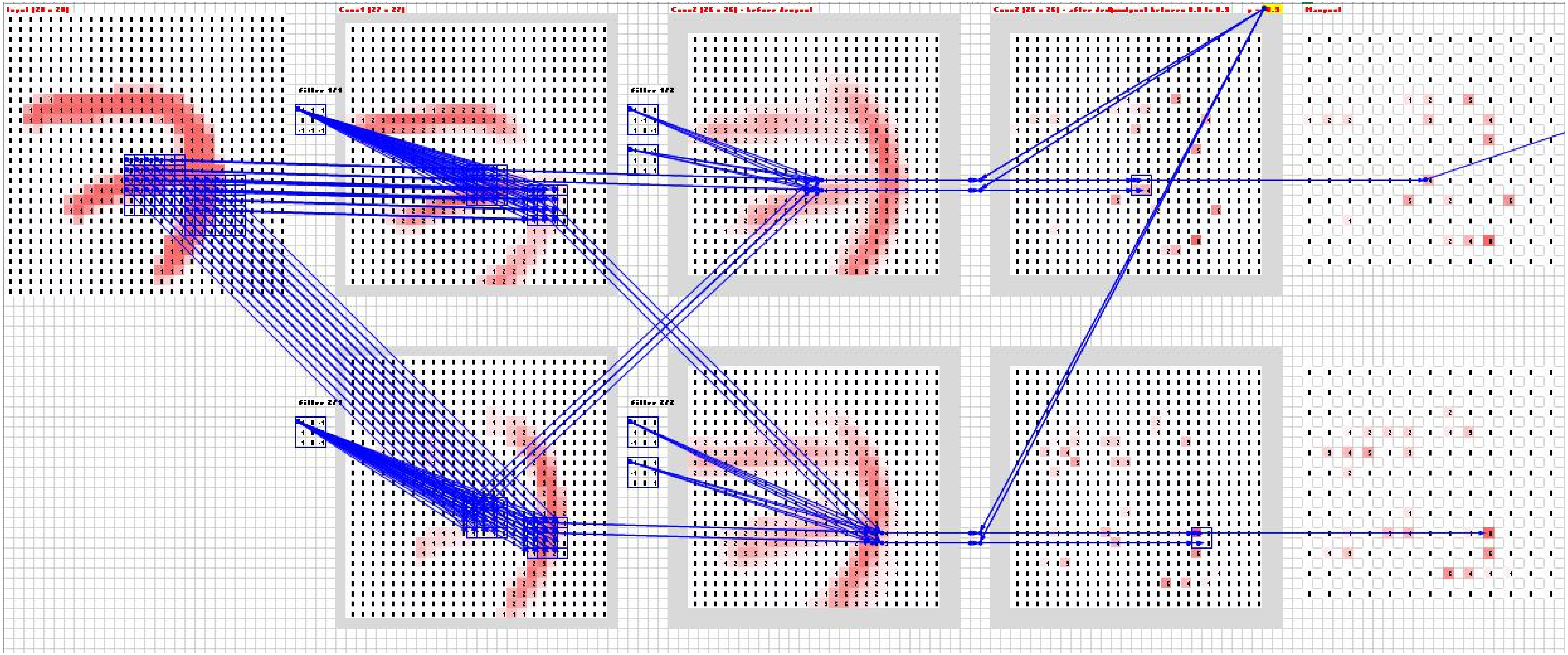
Convolutional CNN cell – output: tensor





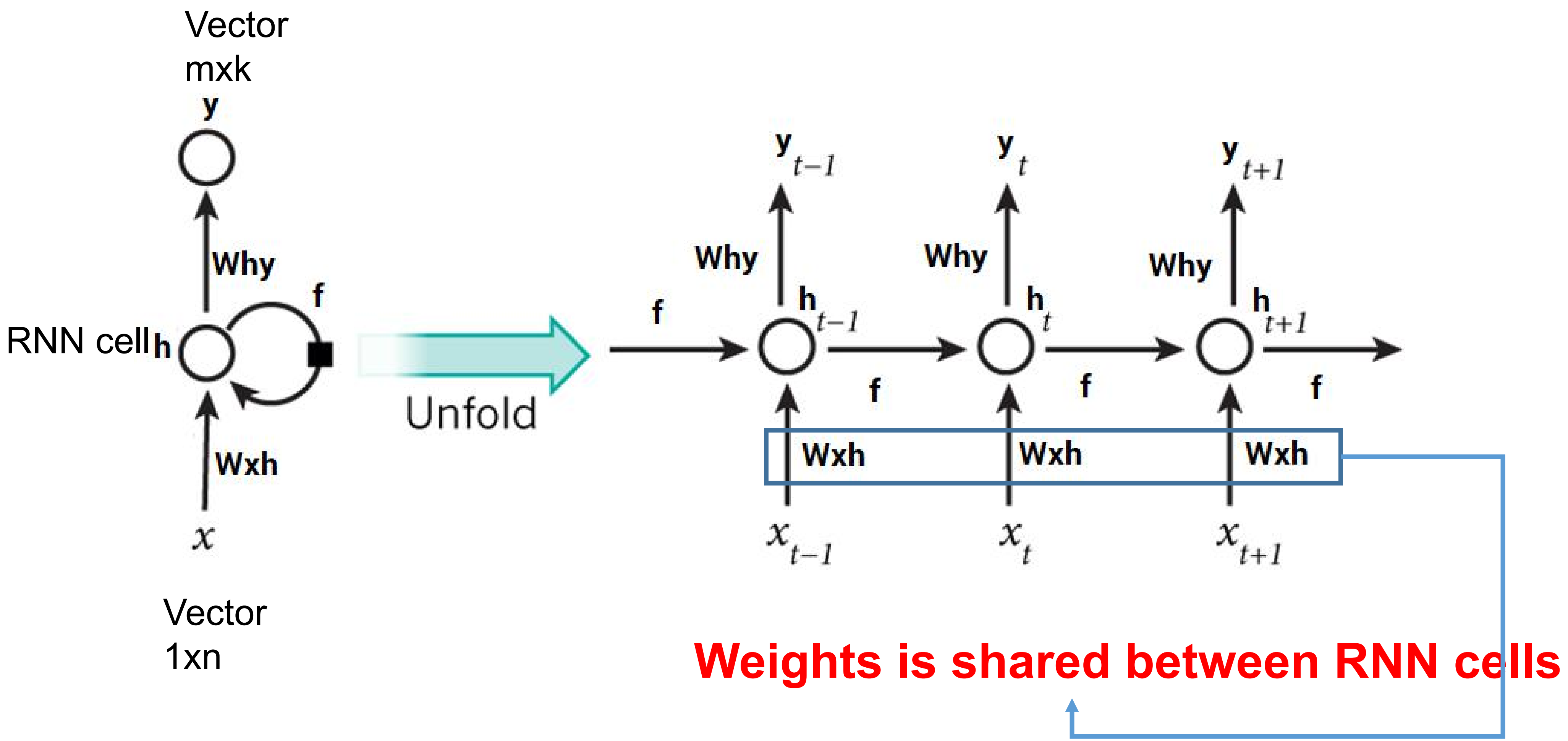
$$\left(f^{(pool)} \circ f^{(sigm)} \circ f_w^{(conv)}\right)(x)$$

Forward propagation

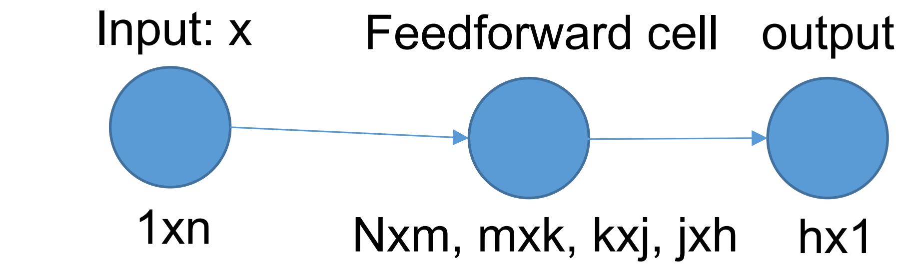


Input (28 x 28)
Conv1 (27 x 27)
Conv2 (26 x 26) - before dropout
Maxpool
Filter 1/1
Filter 1/2
Filter 2/1
Filter 2/2

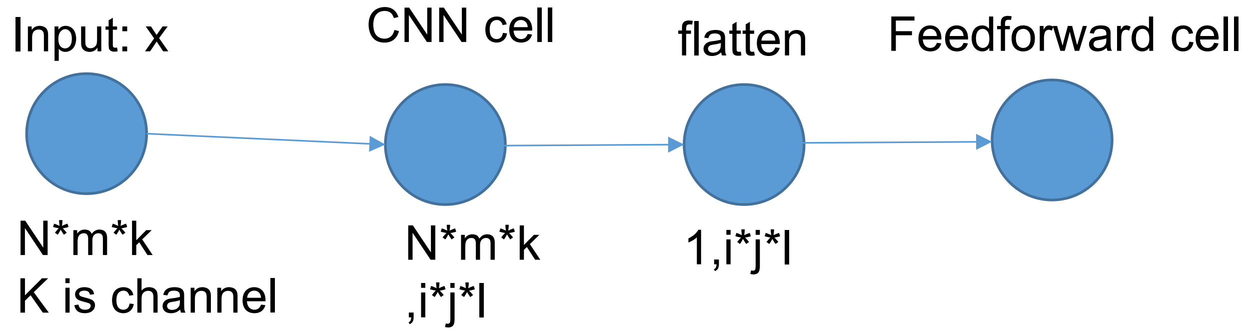
RNN cell – (RNN basic, GRU, LSTM)



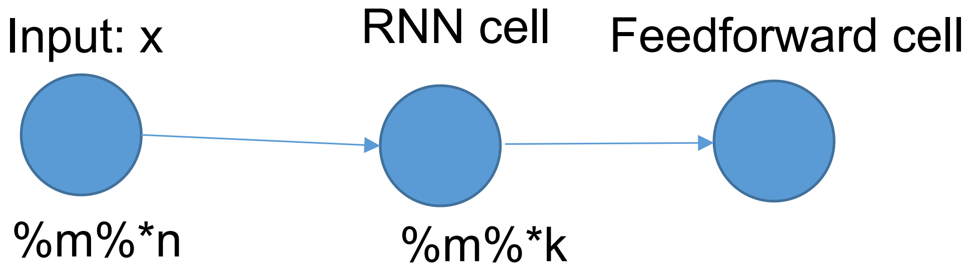
Feed forward net



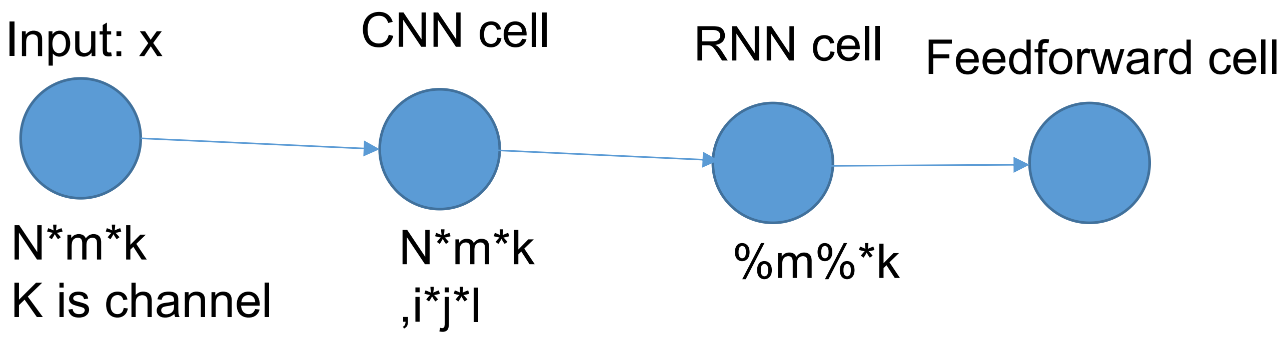
Convolutional neuralnet

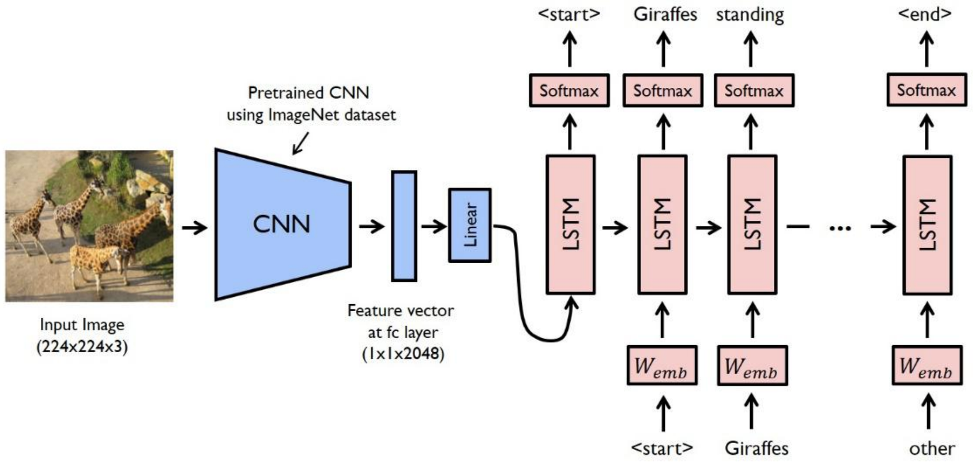


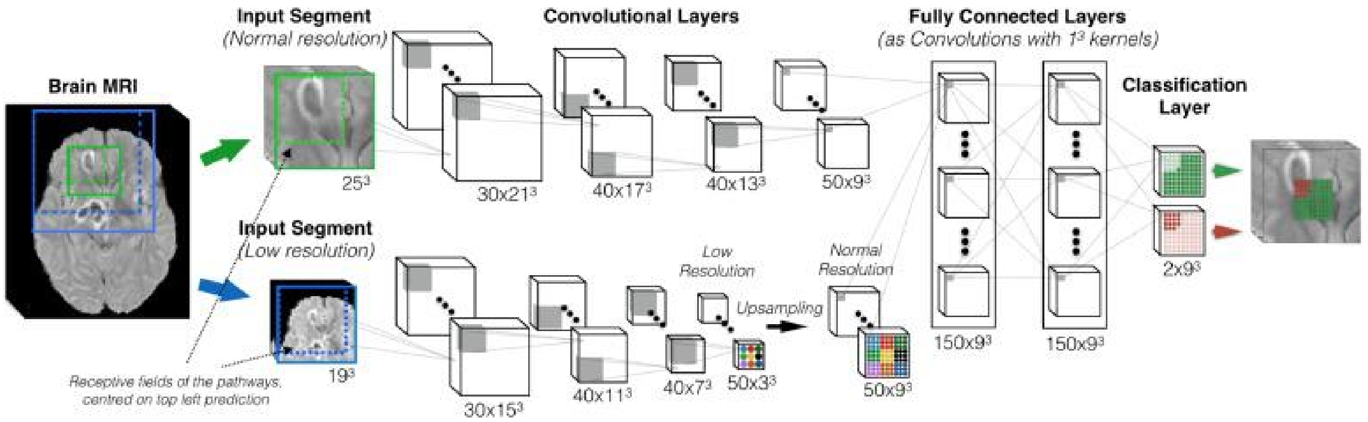
Recurrent neural net

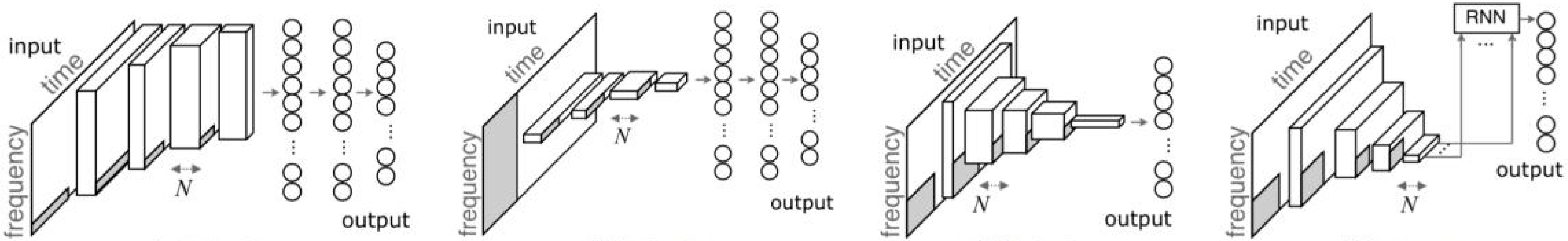


Recurrent - Convolutional neural net



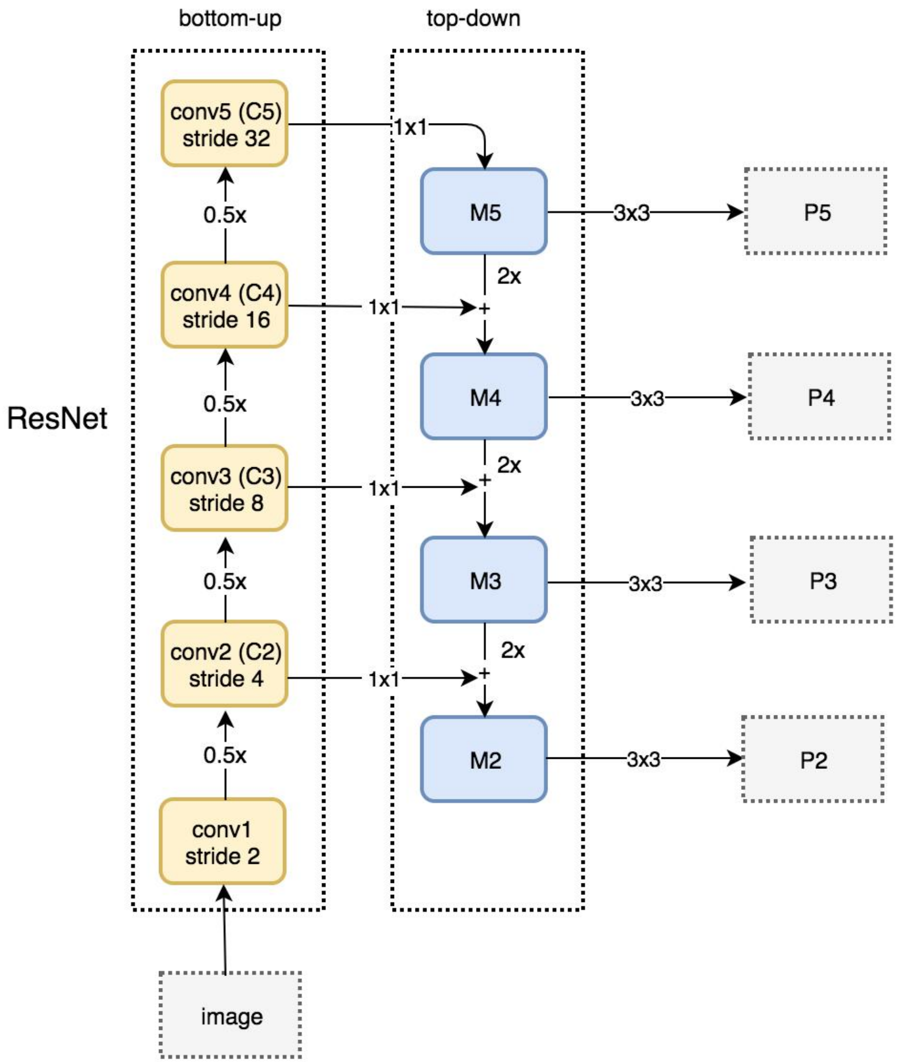


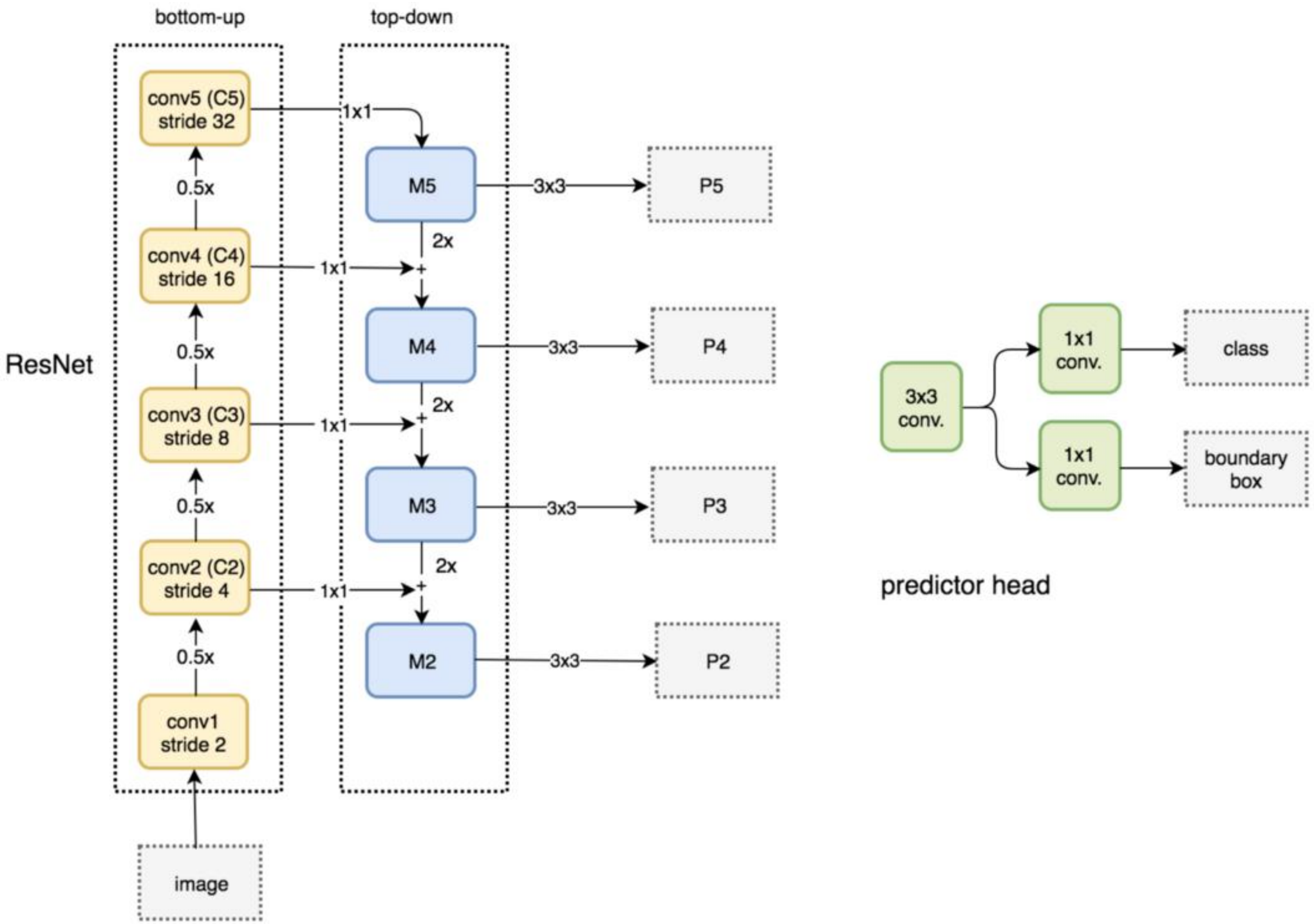




(a) k1c2 (b) k2c1 (c) k2c2 (d) CRNN

Fig. 1: Block diagrams of k1c2, k2c1, k2c2, and CRNN. The grey areas illustrate the convolution kernels. N refers to the number of feature maps of convolutional layers.


bottom-up
top-down
ResNet
conv5 (C5)
stride 32
conv4 (C4)
stride 16
conv3 (C3)
stride 8
conv2 (C2)
stride 4
conv1
stride 2
image
0.5x
1x1
2x
M5
M4
M3
M2
3x3
P5
P4
P3
P2


bottom-up
top-down
ResNet
conv5 (C5)
stride 32
conv4 (C4)
stride 16
conv3 (C3)
stride 8
conv2 (C2)
stride 4
conv1
stride 2
image
0.5x
1x1
2x
M5
M4
M3
M2
3x3
P5
P4
P3
P2
3x3
conv.
1x1
conv.
class
boundary
box
predictor head

Pytorch vs tensorflow

Define and run.

Dynamic vs static

Define -> computation graph (tf) -> run

Everything at **very** low level

Very low level vs low level

☾ Need front end for beginner

Every thing at low level

Python
Object Oriented Programing

"Pythonic' vs "tensorflowic”

Tensorflow self define nearly all method

Research orient at very low level
to better customize architechture
-> Hard for compatible at production

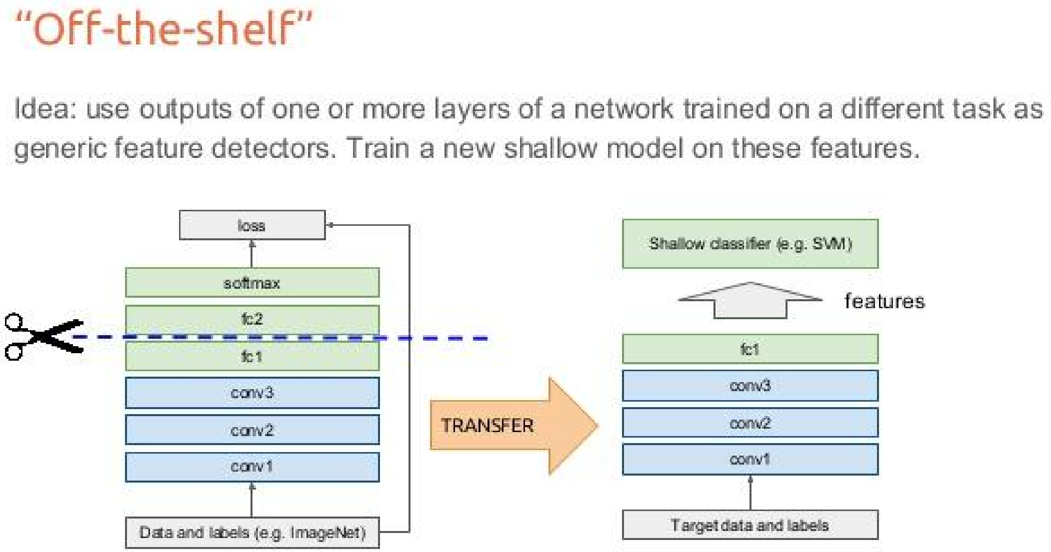
Customizeable vs
Appliable in production

Production oriented
for better performance at Google scale

Best practice for practitioner

- Transfer learning

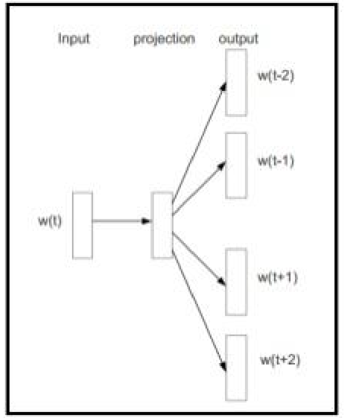
Computer vision



Trending on Deep learning Unsurpervised -> Representation -> Surpervised

Word2Vec

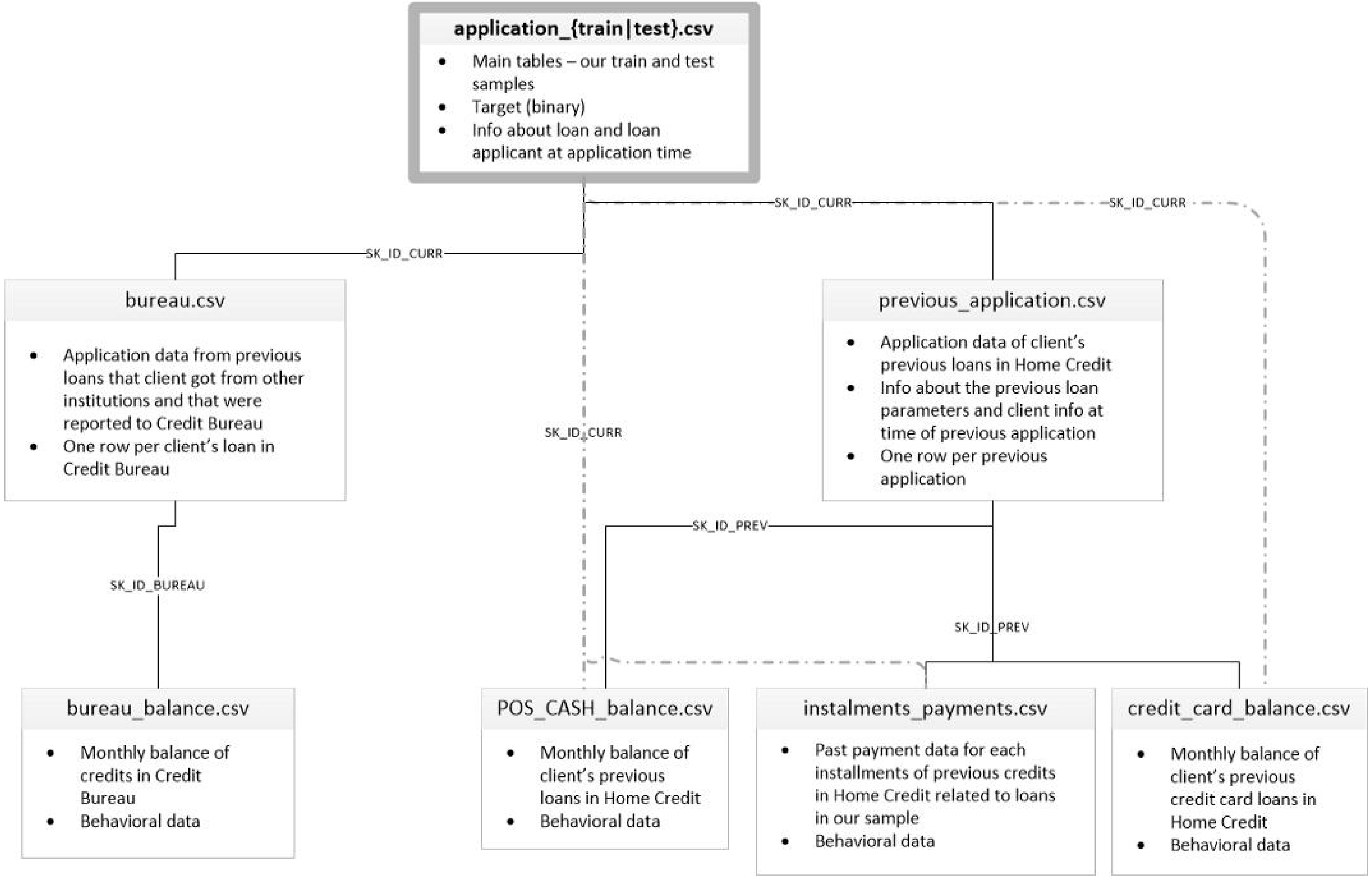
word2vec uses skip-gram neural network to predict neighbor context



Word vector – Representation of word

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	
-0.104777	-0.0893138	-0.0387239	-0.108717	0.099523	0.03715	0.0624399	-0.0641551	-0.149306	-0.126409	-0.
-0.150943	-0.18015	0.0778386	0.0872104	0.0563445	-0.24971	0.0175048	0.0280696	-0.145466	-0.135046	-0.
-0.0784339	-0.177443	0.142632	-0.0650279	0.0404134	-0.125338	0.0700381	-0.0735972	-0.107172	-0.00103557	-0.
0.0207442	-0.121042	0.205818	0.0271886	0.131417	0.00648422	0.0235201	0.0599329	-0.0792963	-0.033467	-0.
-0.124295	-0.207359	0.0547896	-0.0304305	0.154015	-0.0205962	0.0842201	-0.0706606	-0.1647	-0.108152	-(
-0.0497291	-0.172739	0.0526745	-0.142728	0.0208205	-0.101139	0.143159	-0.0821187	-0.179903	-0.135258	-(
-0.0568149	-0.0629231	0.0777229	-0.0256429	0.101506	-0.0389447	-0.0793635	-0.0593537	-0.182863	-0.0729923	-(
-0.0797212	-0.173617	0.108163	-0.107503	0.0216342	-0.140633	0.0506238	-0.0568952	-0.150429	-0.121181	-0.
-0.0732007	-0.16155	0.169509	-0.0588709	0.137003	-0.101893	0.0712918	0.0538054	-0.179305	-0.155106	-0.
-0.355958	-0.196831	-0.015794	0.0348775	0.103654	0.018093	0.156415	-0.015333	-0.0551591	-0.483309	-(

Case study | Default risk



Case study | Default risk

application_train.csv (34.48 MB) — 20 of 122 columns

	# SK_ID_CURR	# TARGET	A NAME_CONTRACT	A CODE_GENDER	A FLAG_OWN_CAR	A FLAG_OWN_REALT	# CNT_CHILDREN	# AI
	1k – 5k	0.00 – 1.00	Cash loans 90% Revolving loans 10% Other 0%	F 66% M 34% Other (1) 0%	N 66% Y 34% Other 0%	Y 69% N 31% Other 0%	0.00 – 19.0	2.6k
1	100002	1	Cash loans	M	N	Y	0	
2	100003	0	Cash loans	F	N	N	0	
3	100004	0	Revolving loans	M	Y	Y	0	
4	100006	0	Cash loans	F	N	Y	0	
5	100007	0	Cash loans	M	N	Y	0	
6	100008	0	Cash loans	M	N	Y	0	
7	100009	0	Cash loans	F	Y	Y	1	
8	100010	0	Cash loans	M	Y	Y	0	
9	100011	0	Cash loans	F	N	Y	0	
10	100012	0	Revolving loans	M	N	Y	0	
11	100014	0	Cash loans	F	N	Y	1	
12	100015	0	Cash loans	F	N	Y	0	

Case study | Default risk

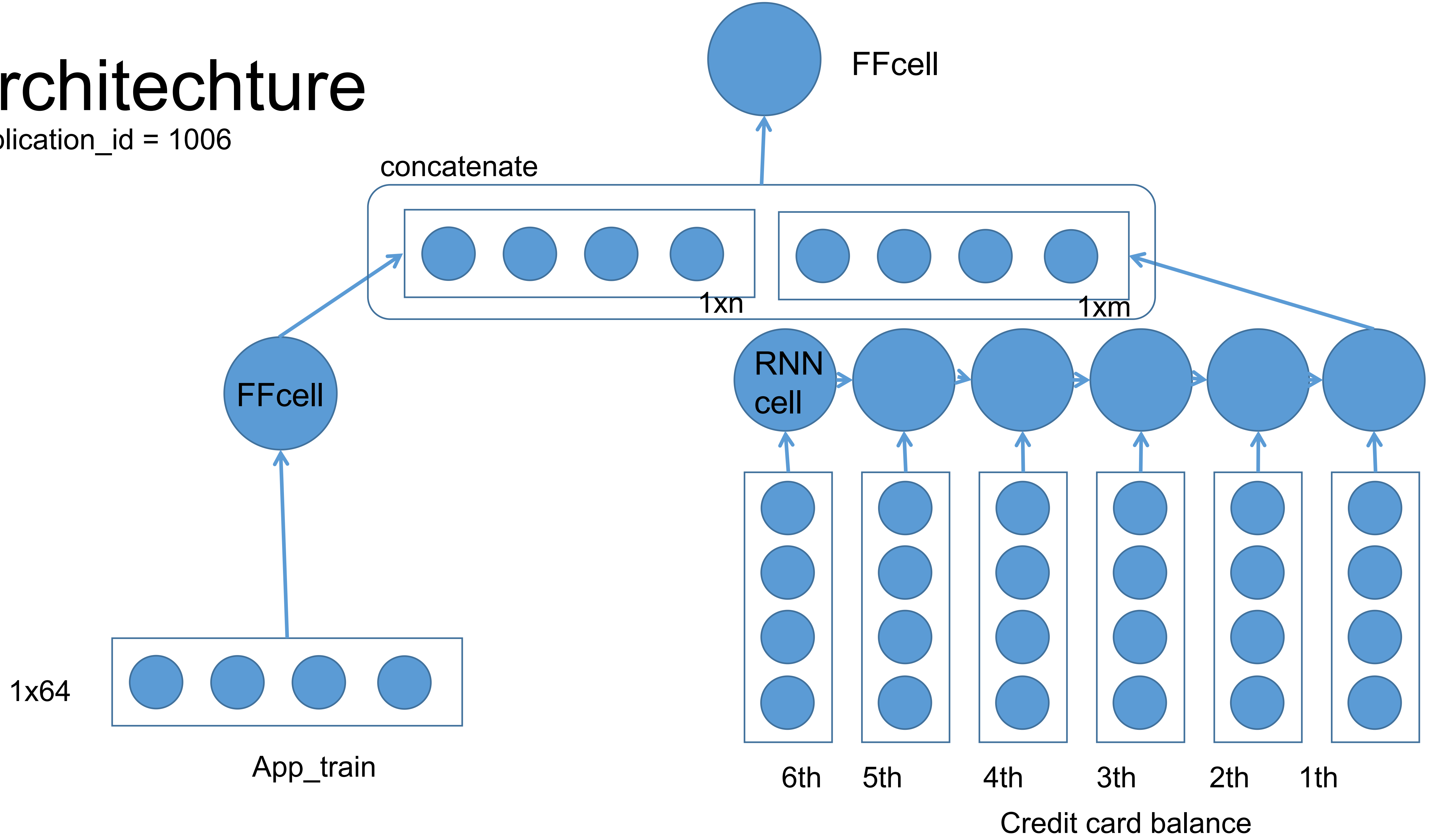
credit_card_balance.csv (94.2 MB)

20 of 23 columns

	# SK_ID_PREV	# SK_ID_CURR	# MONTHS_BALANCE	# AMT_BALANCE	# AMT_CREDIT_LIMIT_AC	A AMT_DRAWINGS_ATM_	# AMT_DRAWINGS_CURR	A AM
	1.00m – 2.84m	1k – 5k (299502.64 - 306627.52 Count: 75,282)	-96.00 – -1.00	-420250.18 – 1.51m	0.00 – 1.35m	0.0 69%; 4500.0 1%; Other (2265) 30%	-6211.62 – 2.29m	0.0; 3343.; Other
1	1489396	100006	-2	0.0	270000		0.0	
2	1489396	100006	-1	0.0	270000		0.0	
3	1489396	100006	-5	0.0	270000		0.0	
4	1489396	100006	-3	0.0	270000		0.0	
5	1489396	100006	-4	0.0	270000		0.0	
6	1489396	100006	-6	0.0	270000		0.0	
7	1843384	100011	-48	75109.95	180000	0.0	0.0	0.0
8	1843384	100011	-4	0.0	90000	0.0	0.0	0.0
9	1843384	100011	-6	0.0	90000	0.0	0.0	0.0
10	1843384	100011	-54	104130.63	180000	0.0	0.0	0.0
11	1843384	100011	-18	0.0	180000	0.0	0.0	0.0
12	1843384	100011	-43	804.195	180000	0.0	0.0	0.0
13	1843384	100011	-65	151067.115	180000	0.0	0.0	0.0
14	1843384	100011	-14	0.0	90000	0.0	0.0	0.0
15	1843384	100011	-24	0.0	180000	0.0	0.0	0.0
16	1843384	100011	-30	0.0	180000	0.0	0.0	0.0
17	1843384	100011	-58	122180.265	180000	0.0	0.0	0.0

Architechture

Application_id = 1006



Code

```
class RNNModel(nn.Module):
    def __init__(self, app_ref, cc_ref, cc_sk_id, szs, app_drop, cat_drop, rnn_drop, lin_drops, bs):
        super().__init__()
        self.bs, self.cc_sk_id = bs, cc_sk_id
        self.app_ref, self.cc_ref = app_ref, cc_ref

        szs = [309] + szs
        self.rnn = nn.GRU(input_size = 37, hidden_size = 64, num_layers = 2, dropout=rnn_drop)

        #linear layer
        self.lins = nn.ModuleList([nn.Linear(szs[i], szs[i+1]) for i in range(len(szs)-1)])
        for o in self.lins: kaiming_normal(o.weight.data)
        self.l_outp = nn.Linear(szs[-1], 1)
        kaiming_normal(self.l_outp.weight.data)
        #batchnorm layer
        self.bns_app = nn.BatchNorm1d(245)
        self.bns_lins = nn.ModuleList([nn.BatchNorm1d(sz) for sz in szs[1:]])
        #dropout
        self.app_drop = nn.Dropout(app_drop)
        self.cat_drop = nn.Dropout(cat_drop)
        self.drops_lins = nn.ModuleList([nn.Dropout(drop) for drop in lin_drops])

        self.zeros = V(torch.zeros(self.bs, 1, 37))

    def forward(self, x_in):
        x_inp = x_in.cpu().data.numpy()

        app_input = torch.stack([V(i[0]) for i in self.app_ref[x_inp]])
        app_input = self.app_drop(self.bns_app(app_input))

        cc_input = self.zeros if x_inp[0] not in self.cc_sk_id else torch.stack([V(i) for i in self.cc_ref[x_inp]])
        self.rnn.flatten_parameters()
        outp,_ = self.rnn(cc_input)

        x = self.cat_drop(torch.cat([app_input, outp[:,-1,:]], 1))

        for linear,drop_out,batch_norm in zip(self.lins, self.drops_lins, self.bns_lins):
            x = drop_out(batch_norm(F.relu(linear(x))))
        x = F.sigmoid(self.l_outp(x))
        return x
```

Question?