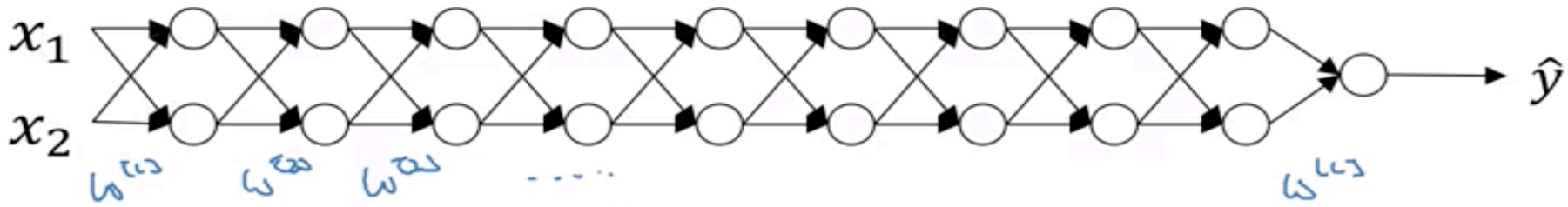
IND STUDY WEEK2

DEEP LEARNING

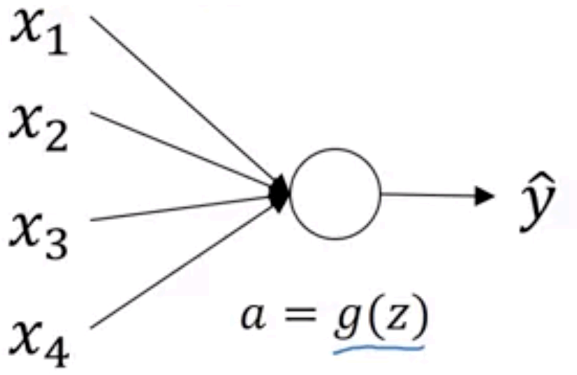
INDEX

Vanishing/Exploring Gradients



- g(z)=z, $b^{[l]}$ =0
- Y=$w^{[l]}$ $w^{[l-1]}$ $w^{[l-2]}$ $w^{[l-3]}$ …….$w^{[3]}$ $w^{[2]}$ $w^{[1]}$ x ; $z^{[l]}$= $w^{[l]}$ x, $a^{[l]}$= g($z^{[l]}$);
- $a^{[2]}$= g($z^{[2]}$) = g($w^{[2]}$ $a^{[1]}$)
- $w^{[l]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}$; y= $w^{[L]}$ $\begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{L-1}$ x
- 1.5^L for above matrix; replace 1.5 with 0.5 it becomes 0.5^L
- Activation values will decrease exponentially if $w^{[l]}$ < I : ½, ¼, 1/8, $1/2^L$
- Activation values will increase exponentially if $w^{[l]}$ > I

Weight Initialization for Deep Networks



- $Z=w_1x_1+ w_2x_2 + w_3x_3 + w_4x_4 +\ldots..+ w_nx_n +b$
- Large n -> smaller w_i ; n is number of input features
- Var(w)=2/n
- $W^{[l]}$ =np.random.randn(shape)*np.sqrt(2/$n^{[l-1]}$); with Relu
- Xavier initialization: with tanh use np.sqrt(1/$n^{[l-1]}$);
- Some use this as well: np.sqrt(2/($n^{[l-1]}$ + $n^{[l]}$));

Gradient Checking

- Don’t use in training- only is debug; $d\Theta_{approx}[i]$ and $d\Theta[i]$
- If it is of order 10^{-7} then it is fine, if it is of order 10^{-3} very bad
- If algorithm fails grad check, look at components to try to identify bug
- Remember regularization
- Does not work with dropout
- Run at random initialization, perhaps again after some training

Quiz

1. If you have 10,000,000 examples, how would you split the train/dev/test set?

0 / 1 point

- (●) 60% train . 20% dev . 20% test

  This should not be selected

- ( ) 33% train . 33% dev . 33% test
- ( ) 98% train . 1% dev . 1% test

3. If your Neural Network model seems to have high variance, what of the following would be promising things to try?

0 / 1 point

- [x] Get more training data

  Correct

- [ ] Make the Neural Network deeper

  This should not be selected

- [ ] Increase the number of units in each hidden layer

  This should not be selected

- [ ] Get more test data

  This should not be selected

- [x] Add regularization

  Correct

4. You are working on an automated check-out kiosk for a supermarket, and are building a classifier for apples, bananas and oranges. Suppose your classifier obtains a training set error of 0.5%, and a dev set error of 7%. Which of the following are promising things to try to improve your classifier? (Check all that apply.)

0 / 1 point

- [x] Increase the regularization parameter lambda

  Correct

- [ ] Decrease the regularization parameter lambda

  Un-selected is correct

- [ ] Get more training data

  This should be selected

- [x] Use a bigger neural network

  This should not be selected

7. With the inverted dropout technique, at test time:

0 / 1 point

- (●) You apply dropout (randomly eliminating units) but keep the 1/keep_prob factor in the calculations used in training.

  This should not be selected

- ( ) You do not apply dropout (do not randomly eliminate units), but keep the 1/keep_prob factor in the calculations used in training.
- ( ) You apply dropout (randomly eliminating units) and do not keep the 1/keep_prob factor in the calculations used in training
- ( ) You do not apply dropout (do not randomly eliminate units) and do not keep the 1/keep_prob factor in the calculations used in training

Weight Initialization and L2 Regularization

- The weights W[l] should be initialized randomly to break symmetry.
- It is however okay to initialize the biases b[l] to zeros. Symmetry is still broken so long as W[l] is initialized randomly.
- Initializing weights to very large random values does not work well.
- initializing with small random values does better.
- L2 regularization makes your decision boundary smoother. If λ is too large, it is also possible to "oversmooth", resulting in a model with high bias.
- the implications of L2-regularization on:
  - The cost computation: A regularization term is added to the cost
  - The backpropagation function: There are extra terms in the gradients with respect to weight matrices
  - Weights end up smaller ("weight decay"):Weights are pushed to smaller values.

Dropout

- Dropout is a regularization technique.
- You only use dropout during training.
- Don't use dropout (randomly eliminate nodes) during test time.
- Apply dropout both during forward and backward propagation.
- During training time, divide each dropout layer by keep_prob to keep the same expected value for the activations. For example, if keep_prob is 0.5, then we will on average shut down half the nodes, so the output will be scaled by 0.5 since only the remaining half are contributing to the solution. Dividing by 0.5 is equivalent to multiplying by 2. Hence, the output now has the same expected value. You can check that this works even when keep_prob is other values than 0.5.

OPTIMIZATION ALGORITHMS

Mini Batch Gradient Descent

- Vectorization allows to compute on m examples
- $X\ (n_x,m)=[x^{[1]}\ ,\ x^{[2]}\ ,\ x^{[3]}\ \ldots\ldots\ldots\ldots\ x^{[m]}\ ]$
- $Y\ (1,m)=\ [y^{[1]}\ ,\ y^{[2]}\ ,\ y^{[3]}\ \ldots\ldots\ldots\ldots\ y^{[m]}\ ]$
- What if m =5,000,000? Split into 5000 minibatches of 1000 each
- $X^{\{1\}}=[x^{[1]}\ ,\ x^{[2]}\ ,\ x^{[3]}\ \ldots\ x^{[1000]}\ ],\ X^{\{2\}}=[x^{[1001]}\ ,\ x^{[1002]}\ \ldots\ldots\ x^{[2000]}\ ]$
- $Y^{\{1\}}=[y^{[1]}\ ,\ y^{[2]}\ ,\ y^{[3]}\ \ldots\ldots\ldots\ y^{[1000]}\ ],\ Y^{\{2\}}=[y^{[1001]}\ ,\ y^{[1002]}\ \ldots\ldots\ y^{[2000]}\ ]$
- Minibatch t: $X^{\{t\}}\ (n_x\ ,1000)\ ,\ Y^{\{t\}}\ (1,1000)$

Mini Batch Gradient Descent

- Repeat {
  for t=1,…,5000{
  forward prop on $X^{\{t\}}$
  $Z^{[1]} = W^{[1]} X^{\{t\}} + b^{[1]}$
  $A^{[1]} = g^{[1]}(Z^{[1]})$
  .....
  $A^{[L]} = g^{[L]}(Z^{[L]})$

  Vectorized implementation (1000 examples)

  Compute Cost $J=\frac{1}{1000}\sum_{i=1}^{l} L(y'^{(i)} - y^{(i)}) + \frac{\lambda}{2*1000}\sum_{l}(w^{[l]})^2$
  Backpropagate to compute gradients wrt $J^{\{t\}}$ (using ($X^{\{t\}}$, $Y^{\{t\}}$)
  $w^{[l]} = w^{[l]} - \text{alpha.}dw^{[l]}$, $b^{[l]} = b^{[l]} - \text{alpha.}db^{[l]}$
  }
}

Called as “1 epoch”- 1 pass through training set

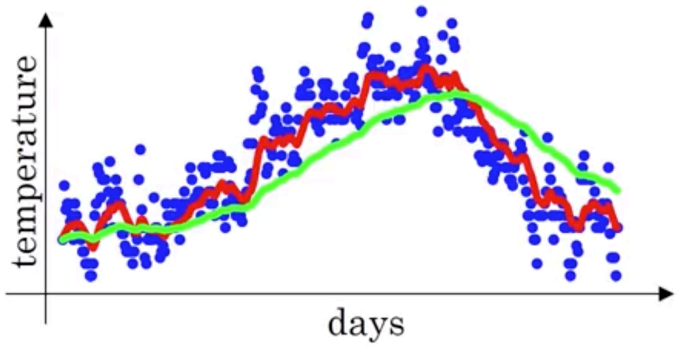
Choosing our mini-batch size

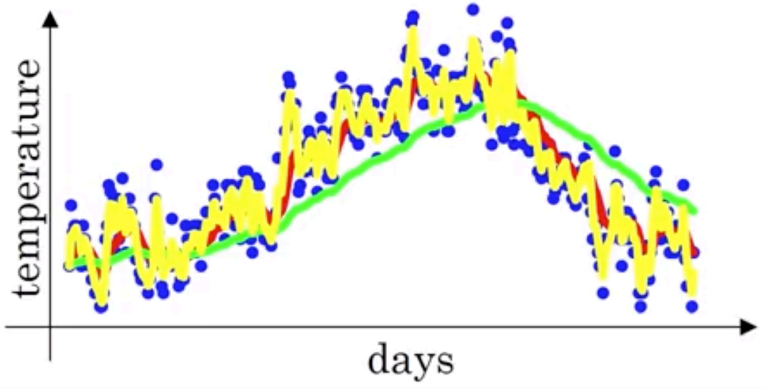
- If minibatch size=m: batch gradient descent. $(X^{\{1\}}, Y^{\{1\}})=(X,Y)$, takes too much time for large m, can be used for small training set (m<=2000)
- If minibatch size=1: stochastic gradient descent. Every example is it's own mini batch. $(X^{\{1\}}, Y^{\{1\}})=(X^{[1]}, Y^{[1]})$. Can be extremely noisy, its always oscillating, never converging. Loose all speed from vectorization
- In practice: somewhere between 1 and m (not too big/small), gives fastest learning. Make progress without processing entire training set. Take power of two: 64, 128, 256, 512, 1024

Exponential Weighted Moving Averages

- V_t =beta *V_{t-1} +(1-beta)*$theta_t$
- Beta=0.9: 10 days temperature (red line)
- Beta=0.98: 50 days temperature (Green smooth line)
- Beta=0.5: days temp (Yellow noisy line)
- V_t as approximately average over = $\frac{1}{1-beta}$ days

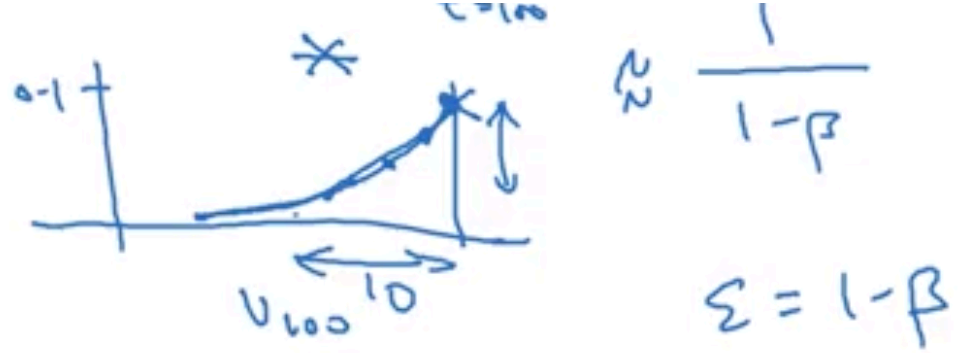
$\theta_1 = 40°F$ 4°C ←
$\theta_2 = 49°F$ 9°C
$\theta_3 = 45°F$ ⋮
⋮
$\theta_{180} = 60°F$ 15°C
$\theta_{181} = 56°F$ ⋮
⋮





Exponential Weighted Moving Averages

- V_t =beta *V_{t-1} +(1-beta)*$theta_t$
- V_{100} =0.9 *V_{99} +0.1*$theta_{100}$
- V_{99} =0.9 *V_{98} +0.1*$theta_{99}$
- V_{98} =0.9 *V_{97} +0.1*$theta_{98}$
- V_{100} =0.1*$theta_{100}$ + 0.9 *(0.1*$theta_{99}$ + 0.9 *(0.9 *V_{97} +0.1*$theta_{98}$))
- 0.9^{10} =0.35=1/e ; 0.98^{50} =1/e



Gradient Descent with momentum

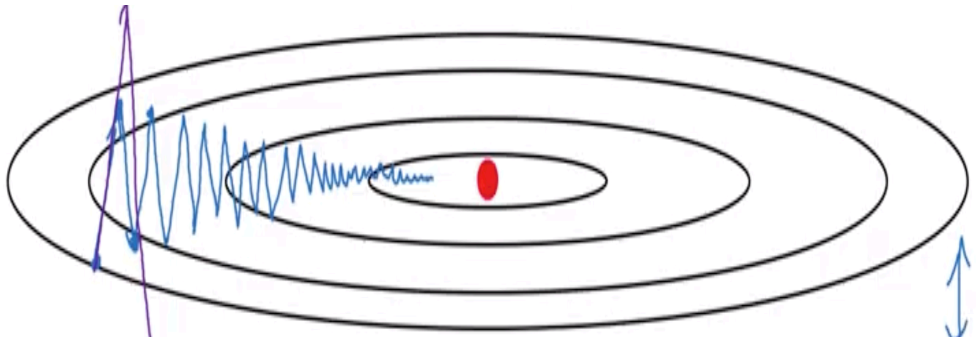
- On Vertical axis, we want slower learning rate
- On horizontal axis, we want faster learning rate
- On iteration t:

  Compute dw,db on current minibatch

  V_{dw} =beta. V_{dw} +(1-beta).dw

  V_{db} =beta. V_{db} +(1-beta).db

  w=w-alpha. V_{dw} , b=b-alpha. V_{db}



- Hyperparameters: alpha, beta; beta=0.9

Root Mean Square Prop

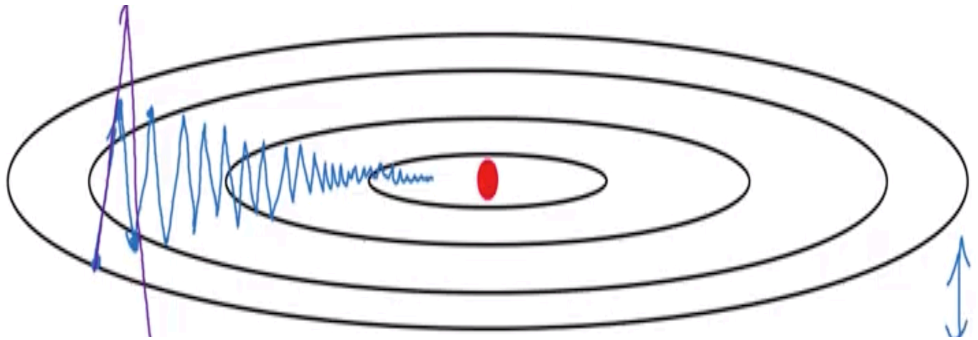
- Slow down learning on vertical(b) direction
- Fast learning on horizontal(w) direction
- On iteration t:

  Compute dw,db on current mini batch

  S_{dw} =beta. S_{dw} +(1-beta).dw^2

  S_{db} =beta. S_{db} +(1-beta).db^2

  w=w-alpha. $\frac{dw}{sqrt(S_{dw})}$, b=b-alpha. $\frac{dw}{sqrt(S_{db})}$



- dw small so S_{dw} small, db small so S_{db} large

Adam Optimization Algorithm

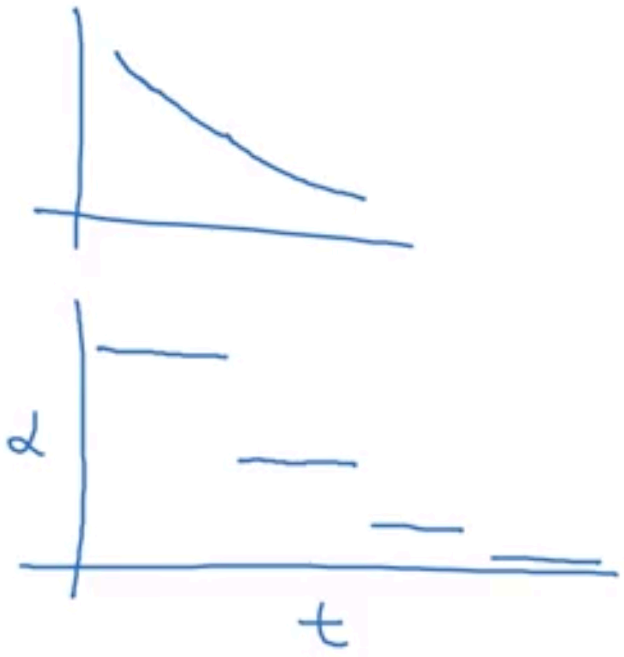
- V_{dw} =0, S_{dw} =0, V_{db} =0, V_{db} =0
- On iteration t:

  Compute dw, db using current minibatch
  V_{dw} =$beta_1$. V_{dw} +(1-$beta_1$).dw
  V_{db} =$beta_1$. V_{db} +(1-$beta_1$).db
  S_{dw} =$beta_2$. S_{dw} +(1-$beta_2$).dw^2
  S_{db} =$beta_2$. S_{db} +(1-$beta_2$).db^2
  $Vcorr_{dw}$ = V_{dw} /(1- $betat_1$), $Vcorr_{db}$ = V_{db} /(1- $betat_1$)
  $Scorr_{dw}$ = S_{dw} /(1- $betat_2$), $Scorr_{db}$ = S_{db} /(1- $betat_2$)
  w=w-alpha. $\frac{Vcorr_{dw}}{sqrt(Scorr_{dw)+epsilon}}$, b=b-alpha. $\frac{Vcorr_{db}}{sqrt(Scorr_{db)+epsilon}}$
- Alpha: needs to be tune
- Beta1: 0.9 (dw)
- Beta2: 0.999 (dw^2)
- Epsilon: 10^{-8}
- Adam: Adaptive moment estimation

Learning Rate Decay

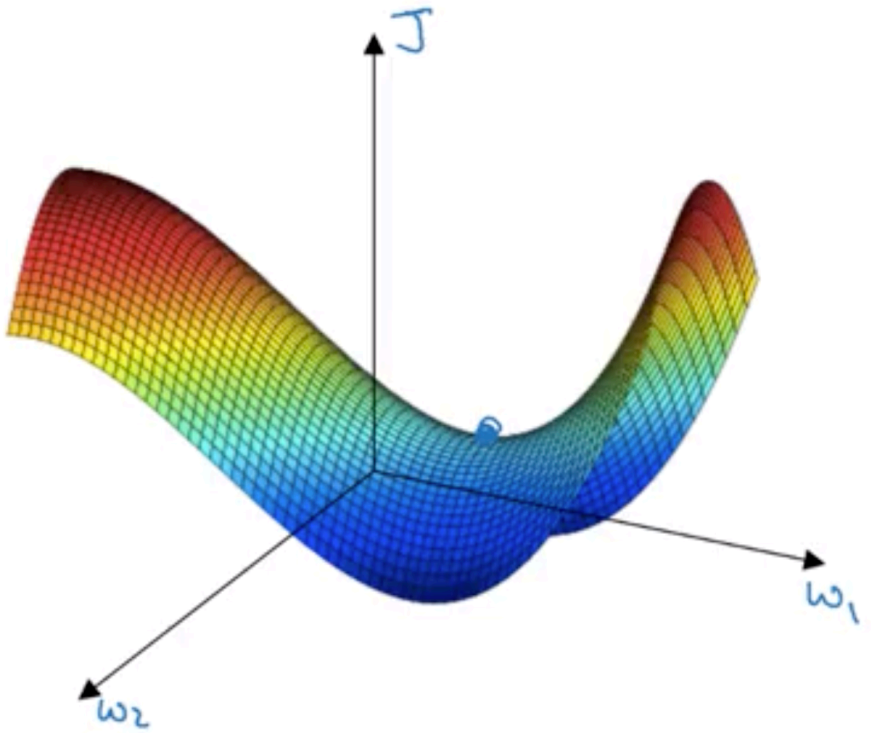
- Slowly reduce learning rate over time
- 1 epoch: 1 pass through the data
- alpha=$\frac{1}{1+decay\_rate*epoch\_num}$*alpha$_0$
- alpha=0.95^{epoch_num} . alpha$_0$
- alpha=$\frac{k}{\sqrt{epoch\_num}}$. alpha$_0$ or $\frac{k}{\sqrt{t}}$. alpha$_0$

E_{poch}	alpha
1	0.1
2	0.67
3	0.5
4	0.4



Problem of Local Optima

- Saddle point: where derivative is zero
- Problem of Plateaus:
  - Unlikely to get stuck in a bad local optima
  - Plateaus can make learning slow



Quiz-optimization

✖ 0 / 1 point

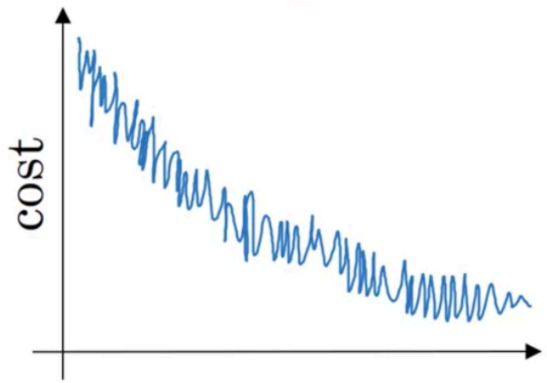
2. Which of these statements about mini-batch gradient descent do you agree with?

- ○ One iteration of mini-batch gradient descent (computing on a single mini-batch) is faster than one iteration of batch gradient descent.
- ○ You should implement mini-batch gradient descent without an explicit for-loop over different mini-batches, so that the algorithm processes all mini-batches at the same time (vectorization).
- ◉ Training one epoch (one pass through the training set) using mini-batch gradient descent is faster than training one epoch using batch gradient descent.

This should not be selected

✔ 1 / 1 point

4. Suppose your learning algorithm's cost J, plotted as a function of the number of iterations, looks like this:



Which of the following do you agree with?

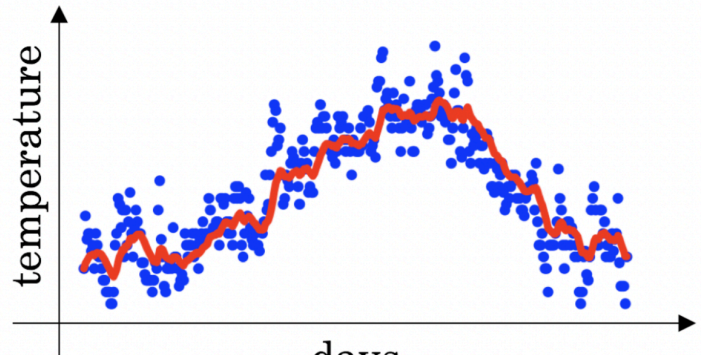
- ○ Whether you're using batch gradient descent or mini-batch gradient descent, this looks acceptable.
- ○ If you're using mini-batch gradient descent, something is wrong. But if you're using batch gradient descent, this looks acceptable.
- ◉ If you're using mini-batch gradient descent, this looks acceptable. But if you're using batch gradient descent, something is wrong.

Correct

- ○ Whether you're using batch gradient descent or mini-batch gradient descent, something is wrong.

✖ 0 / 1 point

7. You use an exponentially weighted average on the London temperature dataset. You use the following to track the temperature: $v_t = \beta v_{t-1} + (1-\beta)\theta_t$. The red line below was computed using $\beta = 0.9$. What would happen to your red curve as you vary β? (Check the two that apply)



- ☐ Decreasing β will shift the red line slightly to the right.

Un-selected is correct

- ☐ Increasing β will shift the red line slightly to the right.

This should be selected

- ☑ Decreasing β will create more oscillation within the red line.

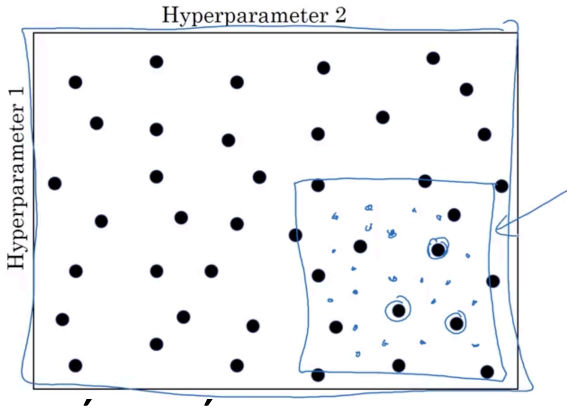
Correct
True, remember that the red line corresponds to $\beta = 0.9$. In lecture we had a yellow line $$\beta = 0.98 that had a lot of oscillations.

- ☐ Increasing β will create more oscillations within the red line.

Un-selected is correct

Hyperparameter Tuning

- Tuning alpha, beta, beta1, beta2, epsilon….
- Tuning #layers, #hidden units
- learning rate decay, mini batch size
- Choose in random, not exactly random but scale it
- Search in logarithmic scale for values: 0.0001, 0.001, 0.0
- r=np.random.rand() ; r ∈ $[-4,0]$
- alpha=10^{r} ; 10^{-4} …… 10^{0}
- Beta=0.9…….0.999, 1-beta=0.1….0.001
- Less computing power- train one at a time (Panda baby)
- More computing power-train models in parallel (Fish example)



Batch Normalization

- Makes hyperparameter search more easier, NN more robust
- Can we normalize the value of activation functions?
- Normalize the value of function z
- $\mu=\frac{1}{m}\sum_i z^i$
- $\sigma^2=\frac{1}{m}\sum_i(z_i-\mu)^2$
- $z^i_{norm}=\frac{z^i-\mu}{\sqrt{\sigma^2+\epsilon}}$
- $z^{i'}=\gamma\, z^i_{norm}+\beta$; (γ and β are learnable parameters)
- If $\beta=\mu$ and $\gamma=\sqrt{\sigma^2+\epsilon}$ then $z^{i'}=z^i$
- At test time μ, σ^2 : estimate using exponentially weighted average (across mini-batch).

Batch Normalization as Regularization

- If distribution of x changes then we need to retrain model
- Covariance shift
- Each mini-shift is scaled by the mean/variance computed on just hat mini-batch
- This adds some noise to the values $z^{[l]}$ within that minibatch. So, similar to dropout it adds some noise to each hidden layer's activations.
- This has a slight regularization effect.

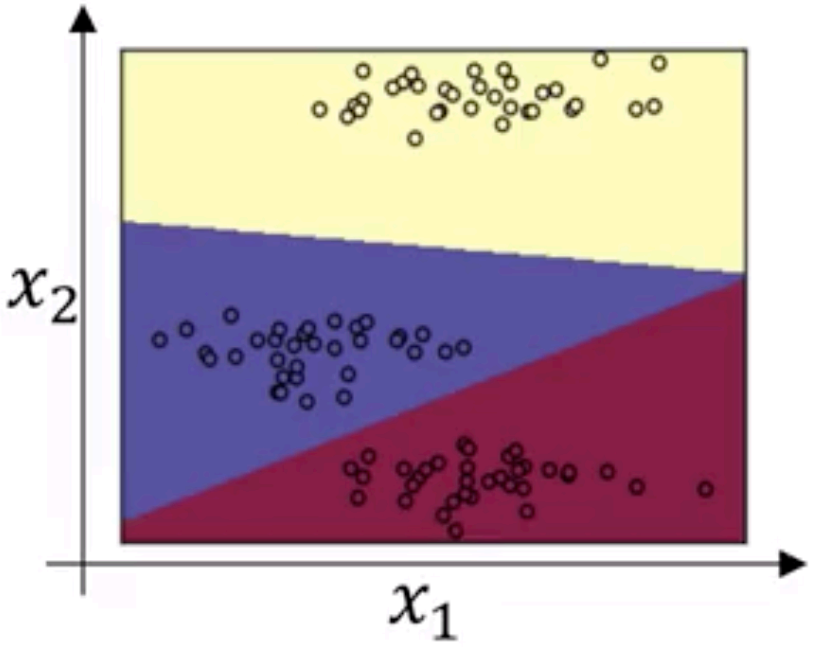
Multi-class classification

Softmax Regression

- 3 classes (yhat): P(other/x)+P(cat/x)+P(day/x)=1
- $Z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$
- $t = e^{(z^{[l]})}$
- $a^{[l]} = \frac{e^{(z^{[l]})}}{\sum_{j=1}^{3} t_i}$; $a^{[l]}$ is (3,1)
- $a_i^{[l]} = \frac{t_i}{\sum_{j=1} t_i}$

- $Z^{[l]} = \begin{matrix} 5 \\ 2 \\ -1 \end{matrix}$, t= $\begin{matrix} e^5 \\ e^2 \\ e^{-1} \end{matrix}$, t= $\begin{matrix} 148.4 \\ 7.4 \\ 0.4 \end{matrix}$, $\sum_{j=1}^{3} t_i$=156.2,

- Decision boundary between any two class is linear



Softmax classifier

- Called as softmax because it directly doesnot maps to 0 and 1 but to some probabilities
- It generalizes logistic regression to c classes

$$z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \qquad t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix}$$

$$a^{[L]} = g^{[L]}(z^{[L]}) = \begin{bmatrix} e^5/(e^5+e^2+e^{-1}+e^3) \\ e^2/(e^5+e^2+e^{-1}+e^3) \\ e^{-1}/(e^5+e^2+e^{-1}+e^3) \\ e^3/(e^5+e^2+e^{-1}+e^3) \end{bmatrix} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$$

Softmax classifier: loss function

- L(yhat,y)= $-\sum_{j=1}^{3} y_j \log y_j$ -> small

  0 0.3

- y =1 ; yhat =0.2;

  0 0.1

- L(yhat,y)=-y_2 log$(\text{yhat})_2$ = - log$(\text{yhat})_2$; make $(\text{yhat})_2$ big
- $dz^{[l]}$ =yhat-y

Quiz

✖ 0 / 1 point

4. If you think β (hyperparameter for momentum) is between on 0.9 and 0.99, which of the following is the recommended way to sample a value for beta?

```
r = np.random.rand()
beta = r*0.09 + 0.9
```

```
r = np.random.rand()
beta = 1-10**(- r - 1)
```

```
r = np.random.rand()
beta = 1-10**(- r + 1)
```

This should not be selected

```
r = np.random.rand()
beta = r*0.9 + 0.09
```

✖ 0 / 1 point

8. Which of the following statements about γ and β in Batch Norm are true?

- [ ] They can be learned using Adam, Gradient descent with momentum, or RMSprop, not just with gradient descent.

This should be selected

- [x] β and γ are hyperparameters of the algorithm, which we tune via random sampling.

This should not be selected

- [ ] The optimal values are $\gamma = \sqrt{\sigma^2 + \varepsilon}$, and $\beta = \mu$.

Un-selected is correct

- [x] They set the mean and variance of the linear variable $z^{[l]}$ of a given layer.

Correct

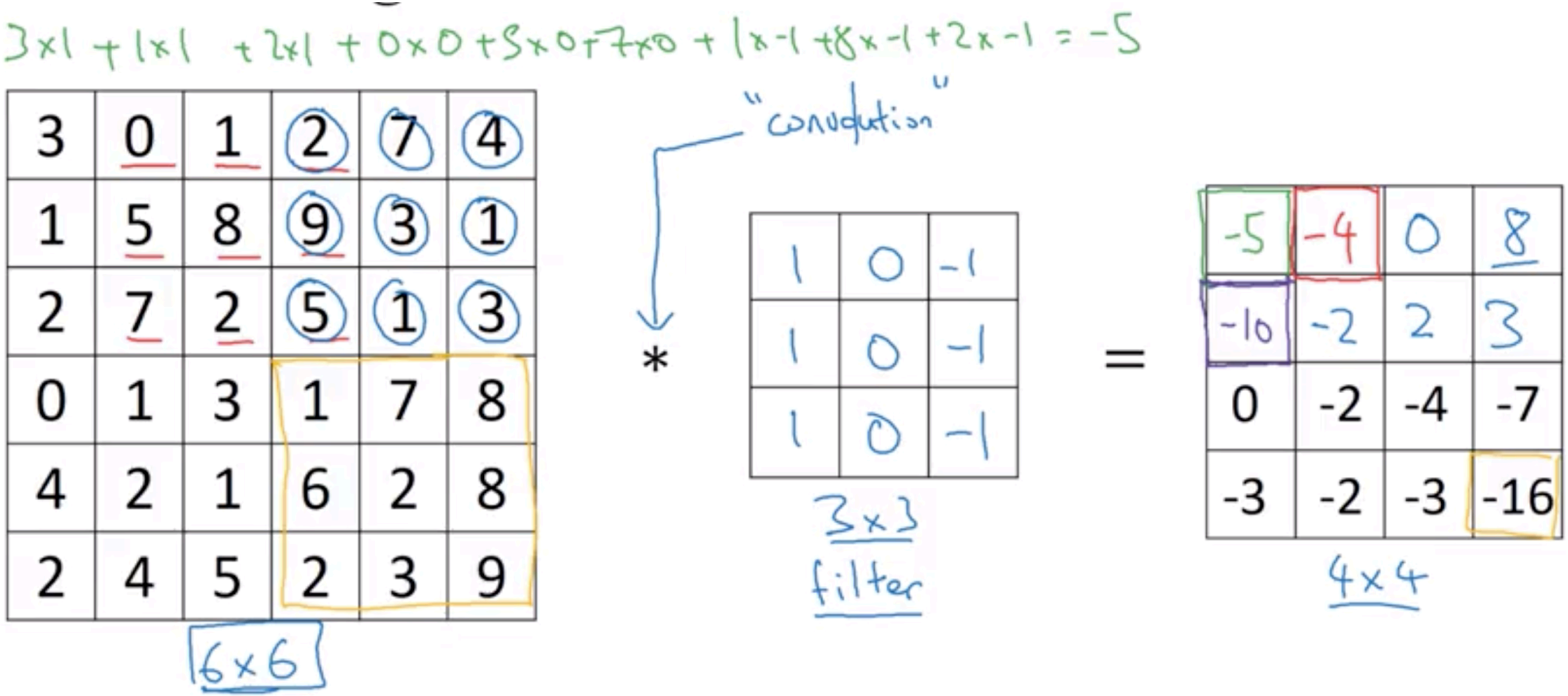
- [ ] There is one global value of $\gamma \in \mathfrak{R}$ and one global value of $\beta \in \mathfrak{R}$ for each layer, and applies to all the hidden units in that layer.

Convolutions Neural Networks

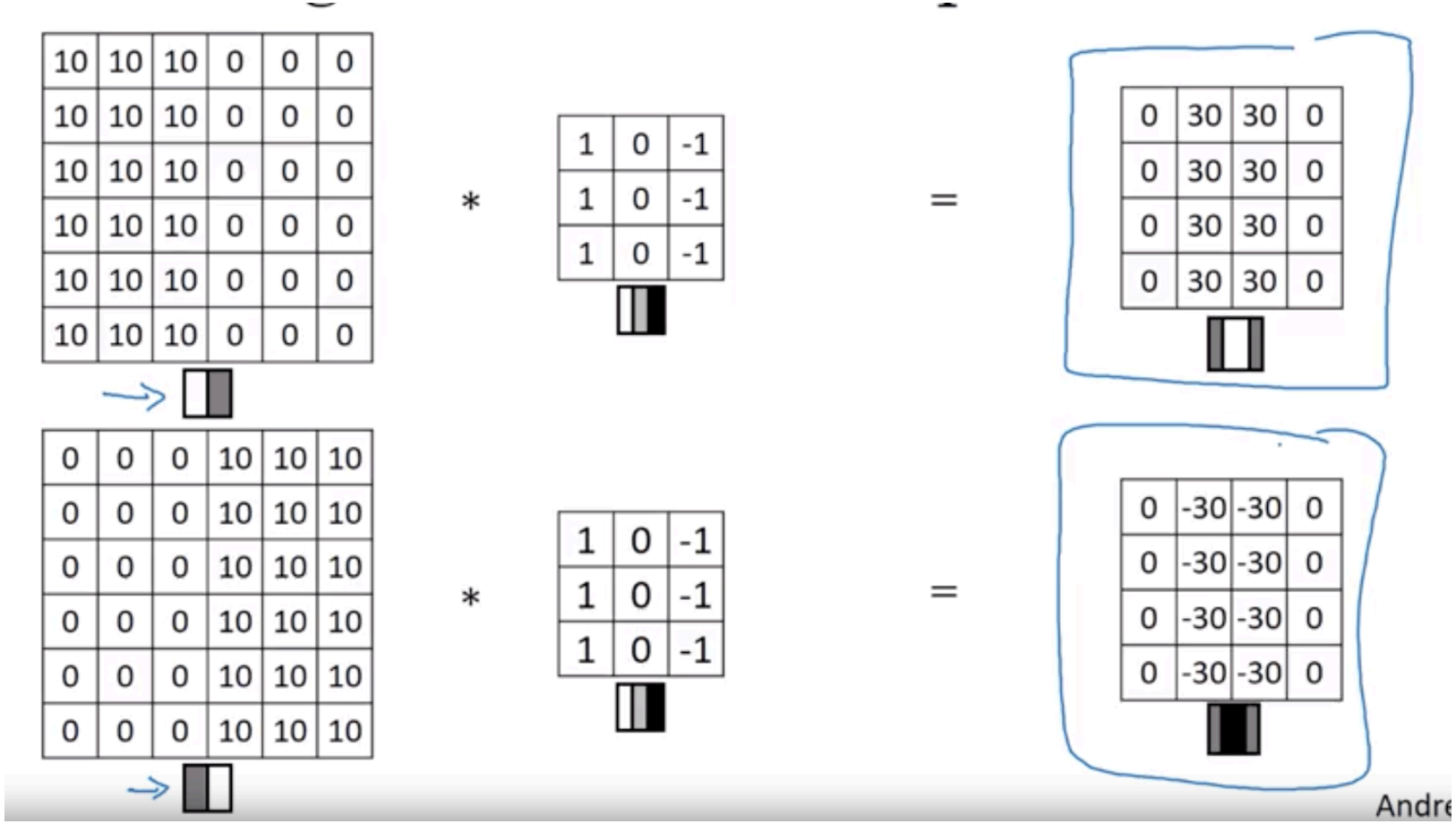
Computer Vision Problems

- Image Classifications- cat vs. non cat classification
- Object Classifications- draw boundary across objects
- Neural Style Transfer- merge two images

Vertical Edge detection



Vertical Edge Detection



Edge Detection

- Vertical- Pixels are lighter on left and darker on right
- Horizontal- Pixels are lighter on top and darker on bottom

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

Vertical

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

Horizontal

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

*

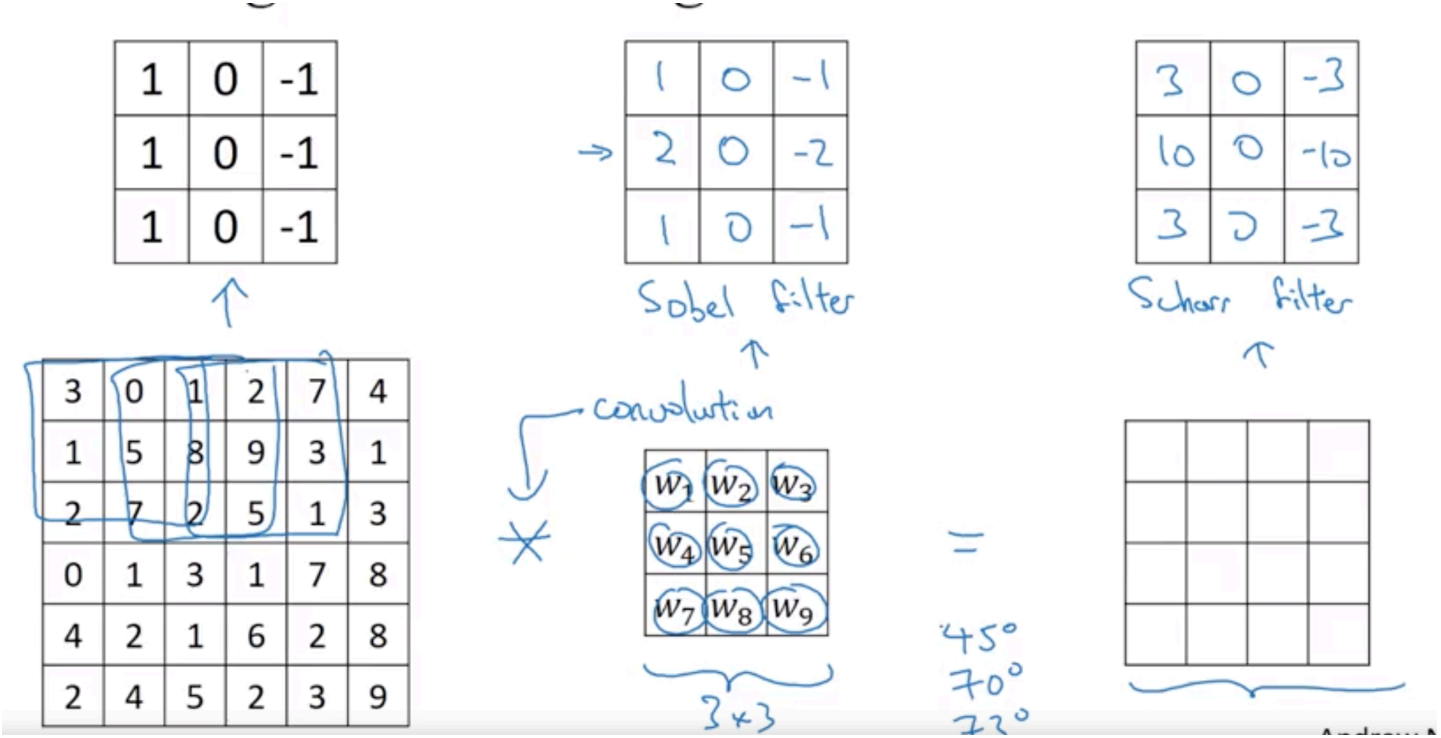
1	1	1
0	0	0
-1	-1	-1

=

0	0	0	0
30	10	-10	-30
30	10	-10	-30
0	0	0	0

Edge Detection

- If we need to detect images with some complicated numbers we don’t need to choose filter numbers, we can learn it using backpropagation

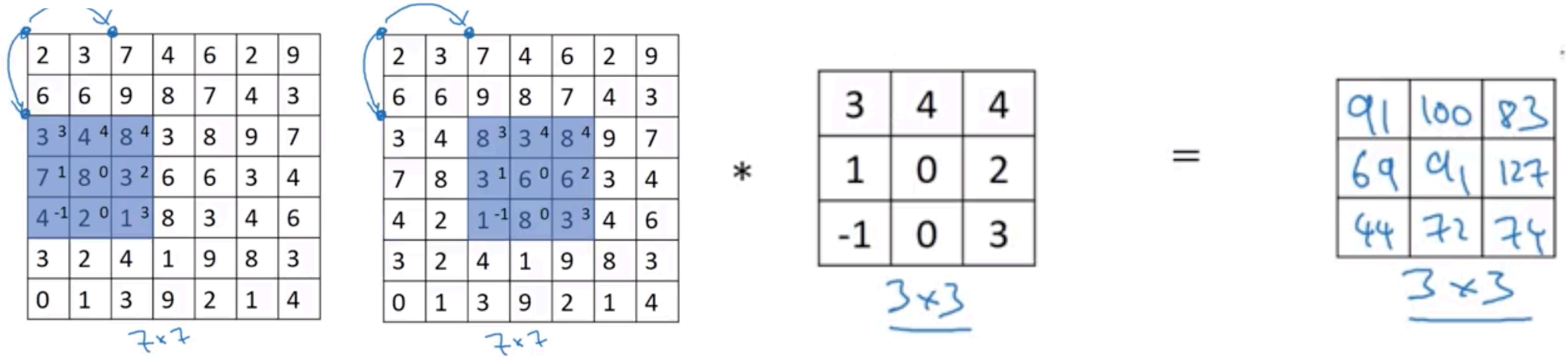


Padding

- If the image is n*n and filter is f*f, resulting image will be (n-f)+1*(n-f)+1
- Downside:
  - Image shrinks- don't want images to shrink
  - Pixels near edges are used less than pixels in middle- throws away useful information
- So we pad the border of image with p pixels (take p=1 : 6*6->8*8)
- Formula: n+2p-f+1=6+2-3+1=6
- Valid convolution: np padding n*n * f*f -> n-f+1 * n-f+1
- Same convolution: Pad so that the output size is same as input size
  - n+2p-f+1=n
  - P=(f-1)/2; filter: 3*3, p=1; filter 5*5, p=(5-1)/2=2
  - By convention f is usually odd

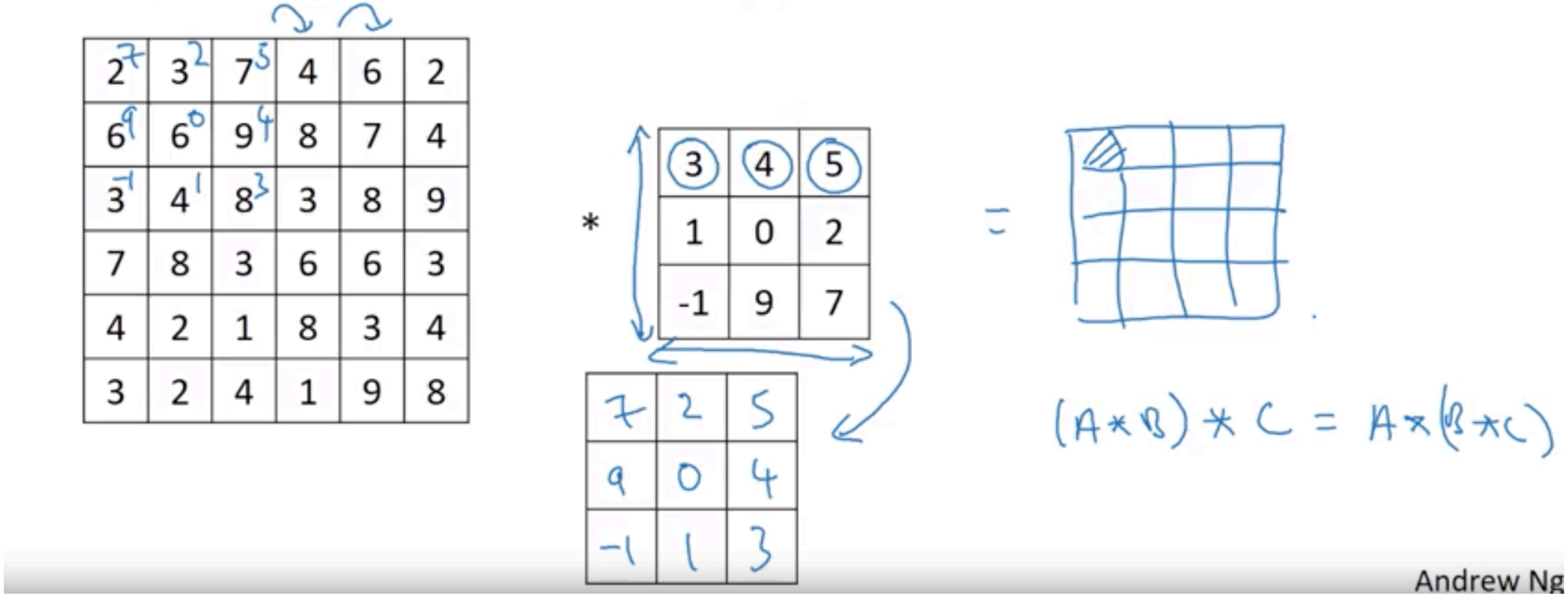
Strided Convolutions

- Stride means hoping the filter over image by n steps
- n*n (padding p) * f*f (stride s) => floor((n+2p-f)/s +1) * (n+2p-f)/s +1
- 7*7 (p=0) * 3*3 (s=2) => floor((7+0-3)/2 +1)=3

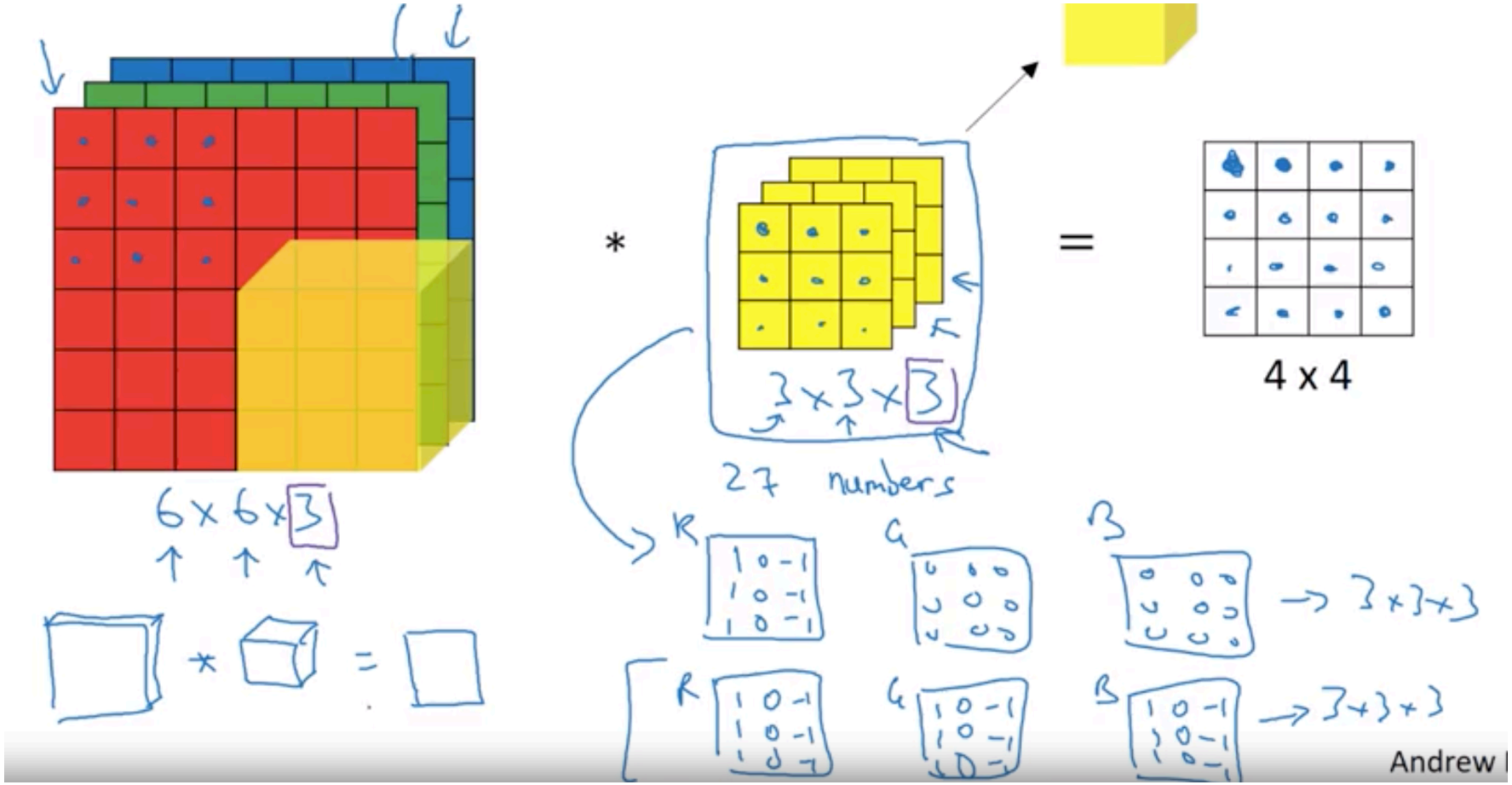


Cross correlation vs convolution

Convolution in math textbook:

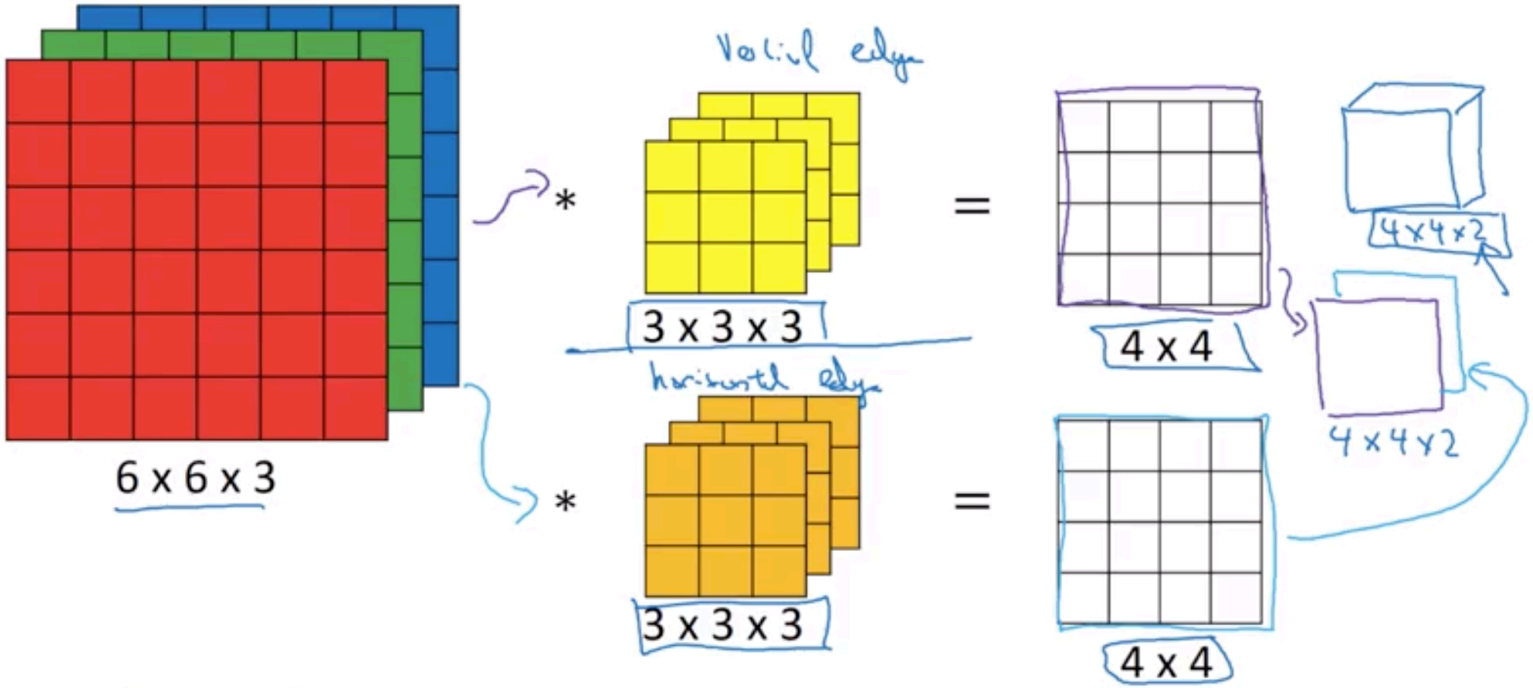


Convolutions over volumes



Convolutions over volumes

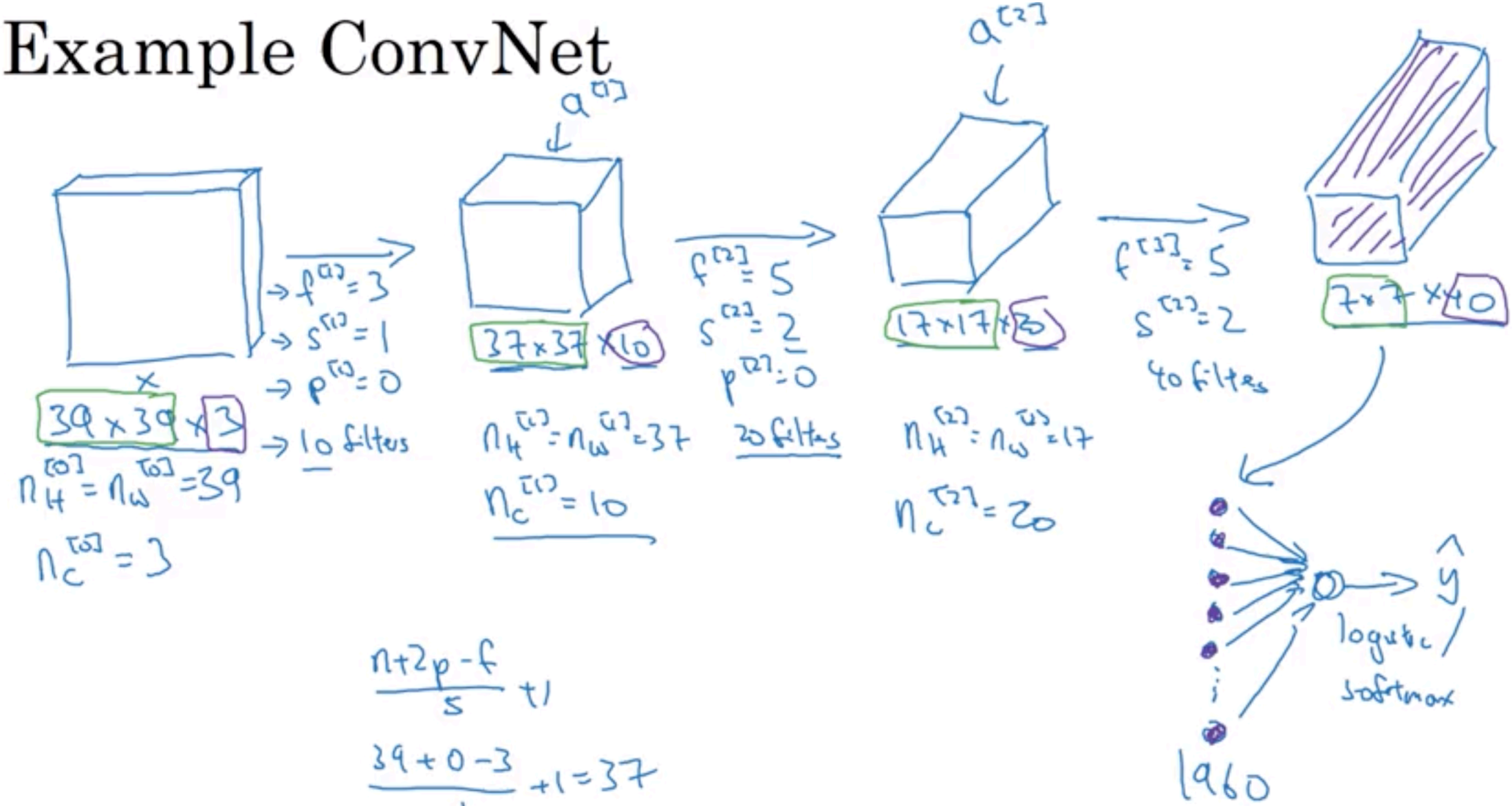
- Detecting edges in any orientation (any angle):
  - n*n*n_c * f*f*n_c -> (n-f+1)* (n-f+1) * n'_c



One layer of CNN

- If you have 10 filters that are 3*3*3 in one layer of a neural network , how many parameters does that layer have?
- 3*3*3 (dimension of each filter) +bias * 10 = (27+1)*10
- If layer l is a convolution layer:
  - $f^{[l]}$ =filter size; $p^{[l]}$ =padding; $s^{[l]}$ =stride
  - Input=$n_H^{l-1} * n_W^{l-1} * n_c^{l-1}$ (Input comes from previous layer, H-height, W-weight, c-channels)
  - Output= $n_H^l * n_W^l * n_c^l$
  - n_c^l=number of filters; each filter of size $f^{[l]} * f^{[l]} * n_c^{l-1}$
  - $n_H^l = \left\lfloor \frac{n_H^{l-1} + 2p^l - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$; same for n_w^l
  - Activations: $a^{[l]} = n_H^l * n_W^l * n_c^l$; $A^{[l]} = m * n_H^l * n_W^l * n_c^l$
  - Weights: $f^{[l]} * f^{[l]} * n_c^{l-1} * n_c^l$
  - Bias: n_c^l - (1,1,1, n_c^l)

Example ConvNet



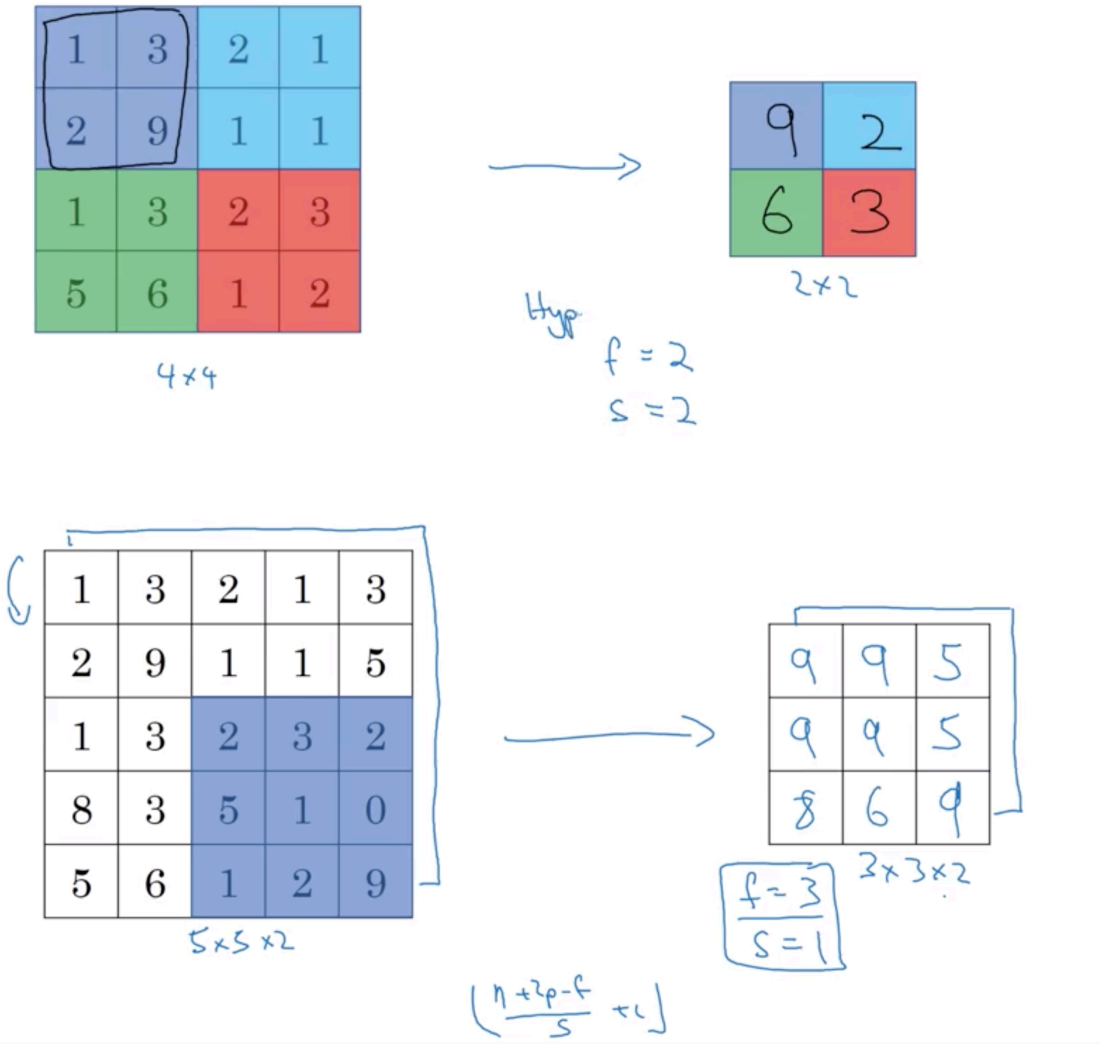
Types of layers in Convolutional network

- Convolution (CONV)
- Pooling (POOL)
- Fully Connected (FC)

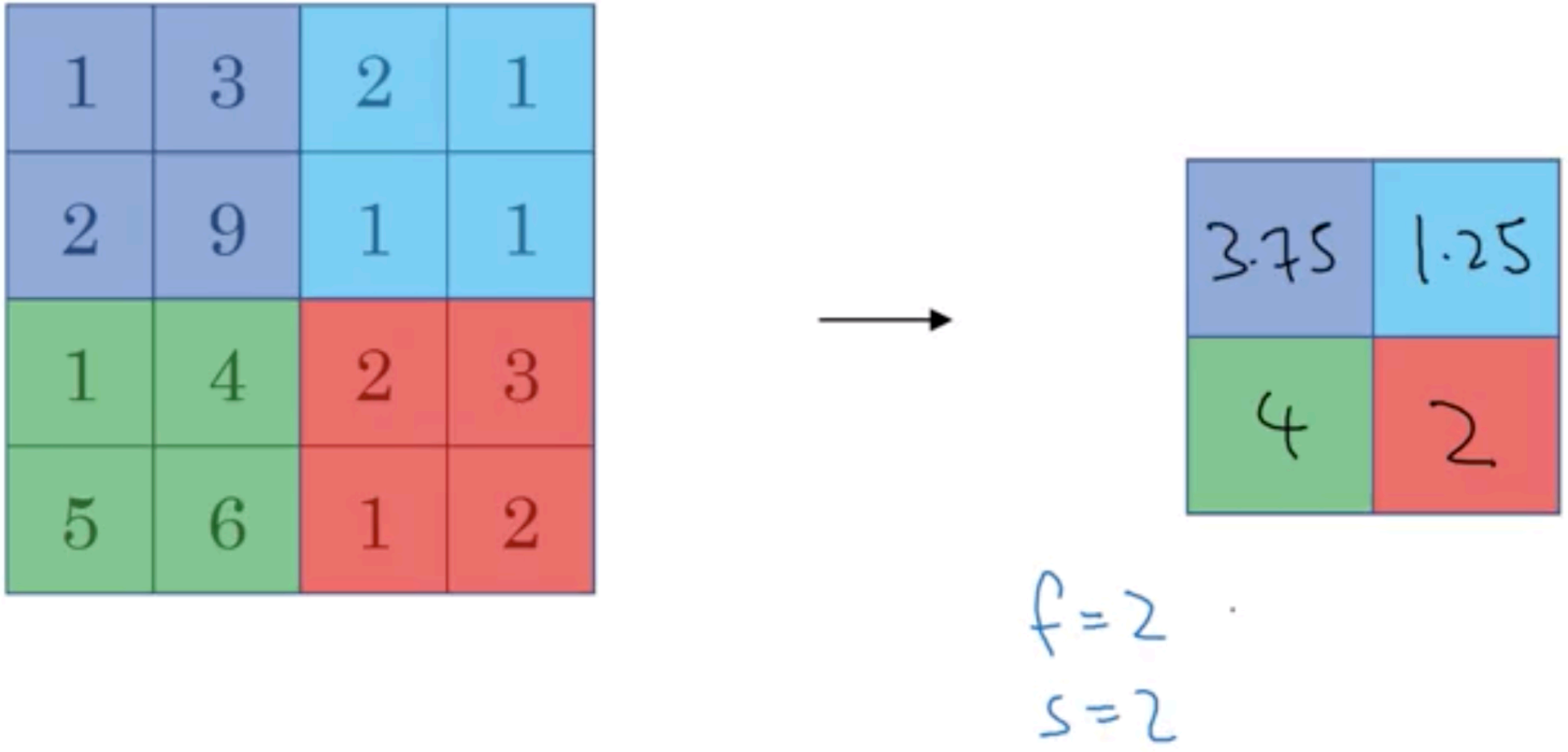
Pooling Layers: Max Pooling

- No parameters to learn like weights
- Hyperparameters
  - f: filter size
  - s: stride
  - Max or average pooling
  - P: padding (very rarely used in max pooling)

Pooling Layers: Max Pooling

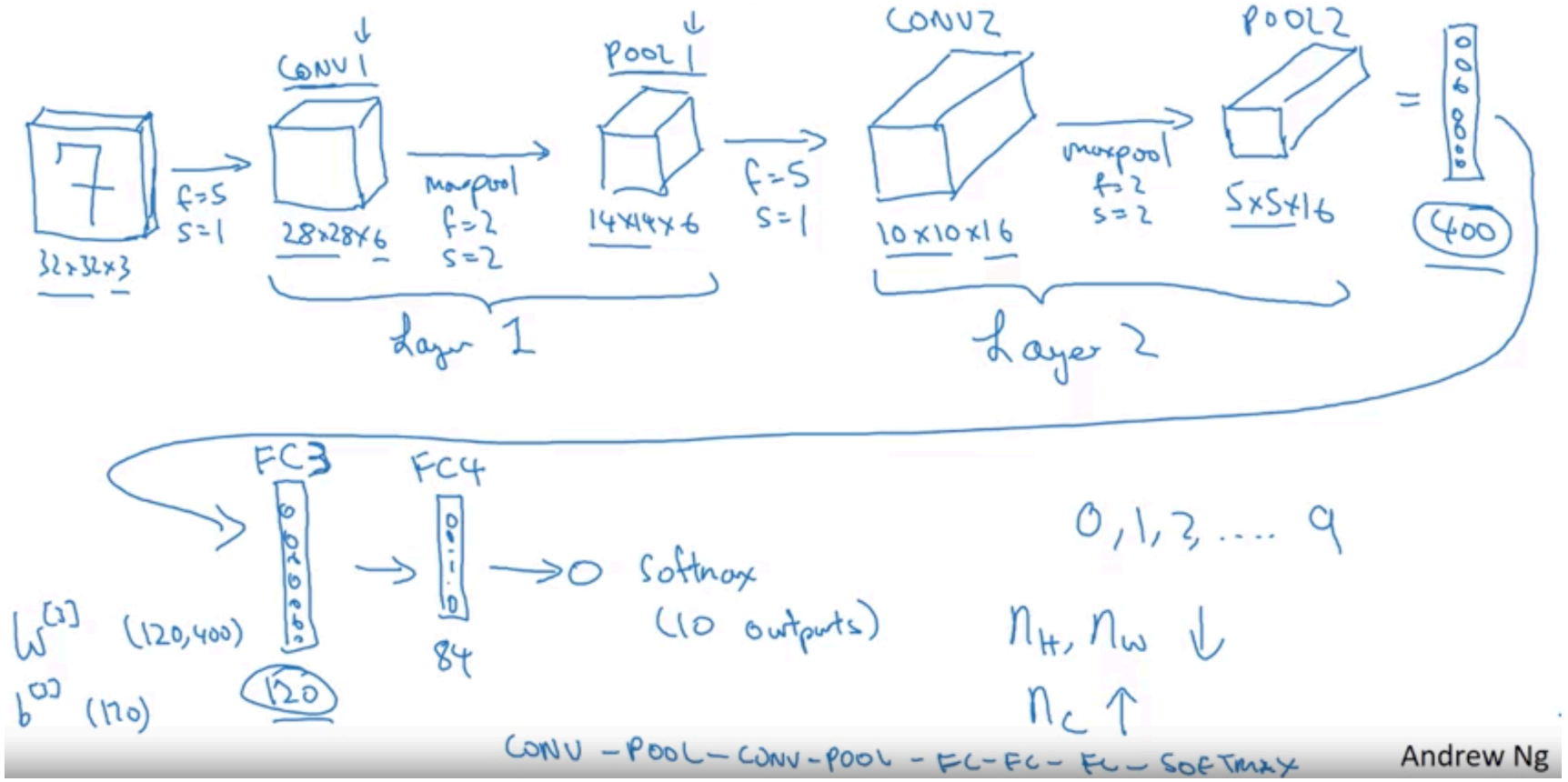


Pooling Layers: Average Pooling



Neural Network Example: LeNet-5

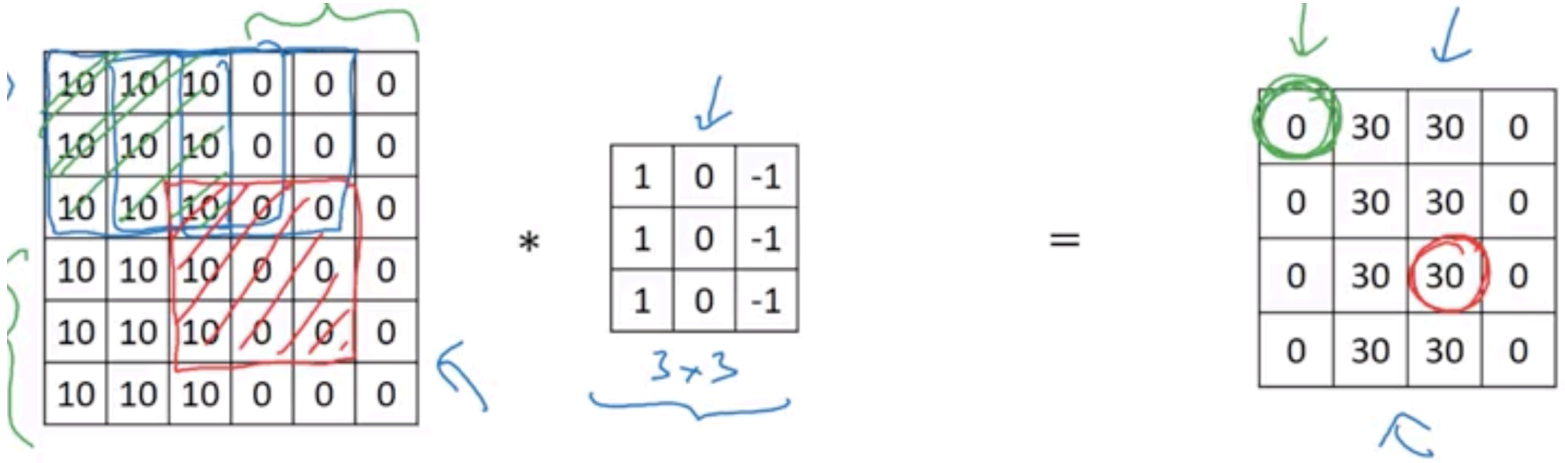
- A layer is counted when parameters can be learnt like weights. So, we call CONV+POOL as a single layer because POOL has no weights to learn



Neural Network Example: LeNet-5

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	3,072	0
CONV1 (f=5, s=1)	(28,28,8)	6,272	208
POOL1	(14,14,8)	1,568	0
CONV2 (f=5, s=1)	(10,10,16)	1,600	416
POOL2	(5,5,16)	400	0
FC3	(120,1)	120	48,001
FC4	(84,1)	84	10,081
Softmax	(10,1)	10	841

Why Convolutions



- Parameter sharing: A feature detector (such as a vertical edge detector) that's useful in one part of the image is probably useful in another part of the image
- sparsity of connections: In each layer, each output value depends only on a small number of inputs
- Good at capturing translation invariance: a image of cat shifted to right is still a cat

Quiz

✖

0 / 1 point

8. Because pooling layers do not have parameters, they do not affect the backpropagation (derivatives) calculation.

✖

0 / 1 point

9. In lecture we talked about “parameter sharing” as a benefit of using convolutional networks. Which of the following statements about parameter sharing in ConvNets are true? (Check all that apply.)

✖

0 / 1 point

1. What do you think applying this filter to a grayscale image will do?

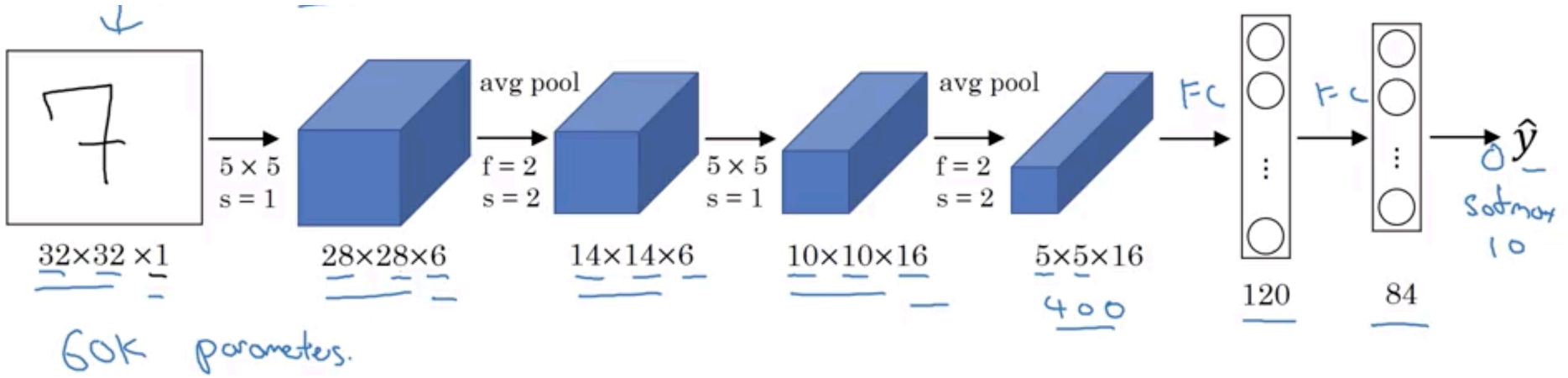
$$\begin{bmatrix} 0 & 1 & -1 & 0 \\ 1 & 3 & -3 & -1 \\ 1 & 3 & -3 & -1 \\ 0 & 1 & -1 & 0 \end{bmatrix}$$

Programming Assignment Notes

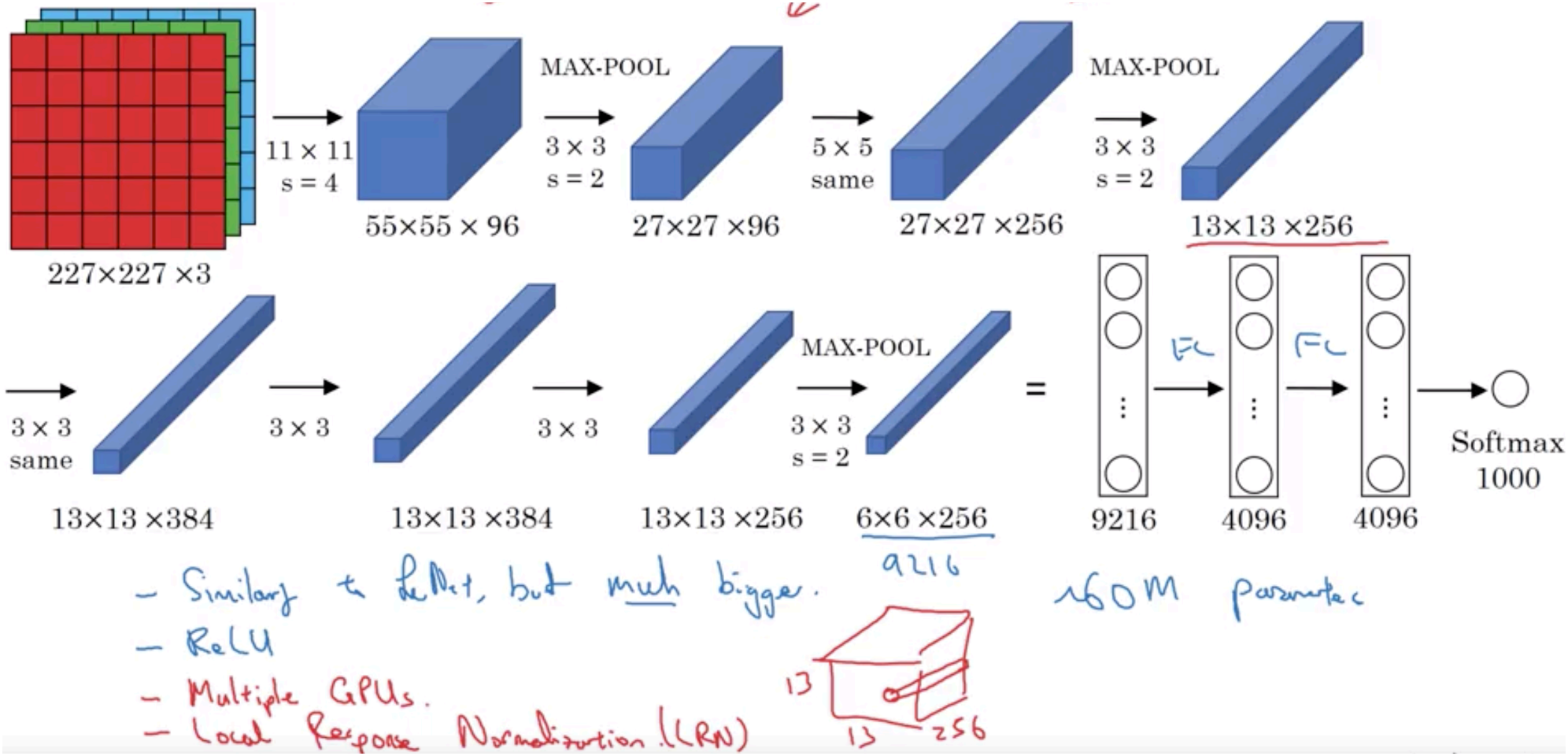
- The main benefits of padding are the following:
  - It allows you to use a CONV layer without necessarily shrinking the height and width of the volumes. This is important for building deeper networks, since otherwise the height/width would shrink as you go to deeper layers. An important special case is the "same" convolution, in which the height/width is exactly preserved after one layer.
  - It helps us keep more of the information at the border of an image. Without padding, very few values at the next layer would be affected by pixels as the edges of an image.

LeNet-5

- Graph Transformer Network for reading bank check
- Replacing hand-crafted feature with automated ML
- Pattern recognition systems combines automated learning+ hand crafted algorithms
- Feature extractor- hand crafted mostly, classifier- ML trained



AlexNet


227×227 ×3
11 × 11
s = 4
55×55 × 96
MAX-POOL
3 × 3
s = 2
27×27 ×96
5 × 5
same
27×27 ×256
MAX-POOL
3 × 3
s = 2
13×13 ×256
3 × 3
same
13×13 ×384
3 × 3
13×13 ×384
3 × 3
13×13 ×256
MAX-POOL
3 × 3
s = 2
6×6 ×256
=
9216
FC
4096
FC
4096
Softmax
1000
- Similar to LeNet, but much bigger.
- ReLU
- Multiple GPUs.
- Local Response Normalization (LRN)
9216
60M parameters
13
13
256

Next Week

- Implementation of CNNS
- Read papers for CNN architecture
- Start with data preprocessing

References:

- Adam paper: https://arxiv.org/pdf/1412.6980.pdf
- Pooling: http://deeplearning.stanford.edu/tutorial/supervised/Pooling/