2. Call Sequences of Sudoku Robot Master:

Call Sequence 1:

Method	In code
tf.placeholder()	x = tf.placeholder("float",[None, img_size_flat],name='x') x_reshape = tf.reshape(x,shape = [-1,img_size,img_size,num_channels], name='x_reshape') y_true = tf.placeholder(dtype=tf.float32, shape=[None,10], name='y_true') y_true_cls = tf.argmax(y_true,dimension=1)
tf.reshape()	
tf.placeholder()	
tf.argmax()	

Call Sequence 2:

Method	In code

tf.nn.softmax()	y_pred = tf.nn.softmax(connectedLayer2)
	y_pred_cls = tf.argmax(y_pred, dimension = 1)
	
	#Now were are going to define the cost function to be optimize
	
	cost_function = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=connectedLayer2, labels=y_true))
	#or you can use the EQM = sum((y_pred-y_true)^2)/N where N is the number of input images
	#cost_function = tf.reduce_mean(tf.square(y_pred-y_true)) #EQM
	
	#Definition of optimization method. Here we use AdamOptimizer but you can use GradientDescendOptimizer is you use
	#for exemple the EQM
	
	#optimizer = tf.train.AdamOptimizer(learning_rate = 1e-4).minimize(cost_function)
	optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost_function)
	
	#Performance Measures
	
	predictions = tf.equal(y_pred_cls,y_true_cls) #vector of boolean that show the matched and unmatched prediction
	
	accuracy = tf.reduce_mean(tf.cast(predictions,tf.float32))
	init = tf.global_variables_initializer()
	saver = tf.train.Saver()
	session = tf.Session()

tf.argmax()	
tf.reduce_mean()	
tf.nn.softmax_cross_entropy_with_logits()	
tf.train.GradientDescentOptimizer()	
tf.equal()	
tf.reduce_mean()	
tf.cast	
tf. global_variables_initializer	
tf.train.Saver()	
tf.Session()	

Call Sequence 3:

Method	In code
tf.Variable()	return tf.Variable(tf.constant(0.05,shape=[length]))
tf.truncated_normal()	

Call Sequence 4:

Method	In code
tf.Variable()	return tf.Variable(tf.constant(0.05,shape=[length]))
tf.constant()	

Call Sequence 5:

Method	In code
tf.nn.conv2d()	layer = tf.nn.conv2d(input = inputs, filter = weights, strides = [1,1,1,1], padding ='SAME') layer += biases if(usePooling): layer = tf.nn.max_pool(value = layer, ksize = [1,2,2,1], strides = [1,2,2,1], padding = 'SAME') layer = tf.nn.relu(layer)
tf.nn.max_pool()	
tf.nn.relu()	

Call Sequence 6:

Method	In code
tf.reshape()	flattenedLayer = tf.reshape(layer,[-1,numFeatures])

Call Sequence 7:

Method	In code
tf.matmul()	layer = tf.matmul(inputs,weights)+biases if(useRelu): layer = tf.nn.relu(layer)
tf.nn.relu()	