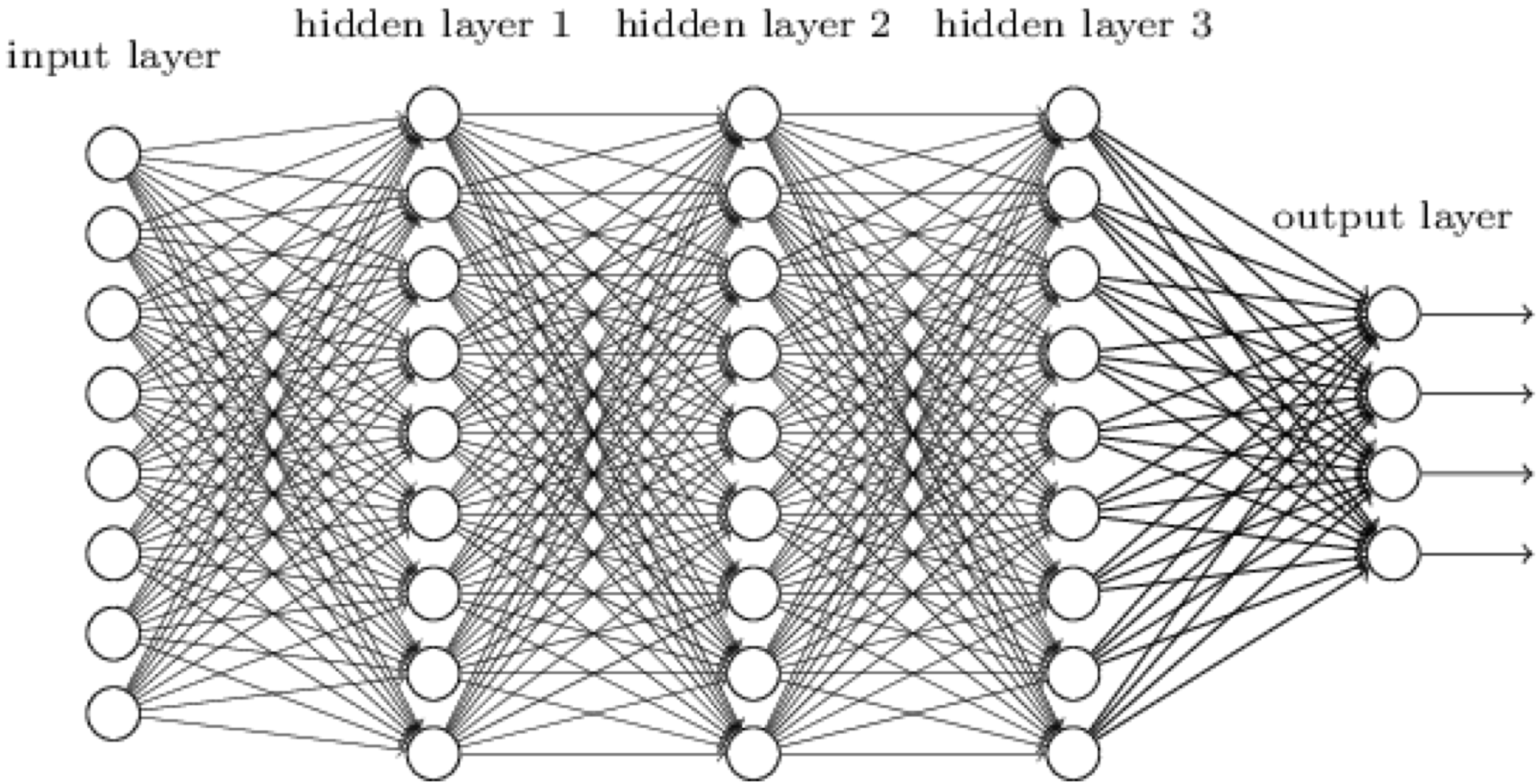
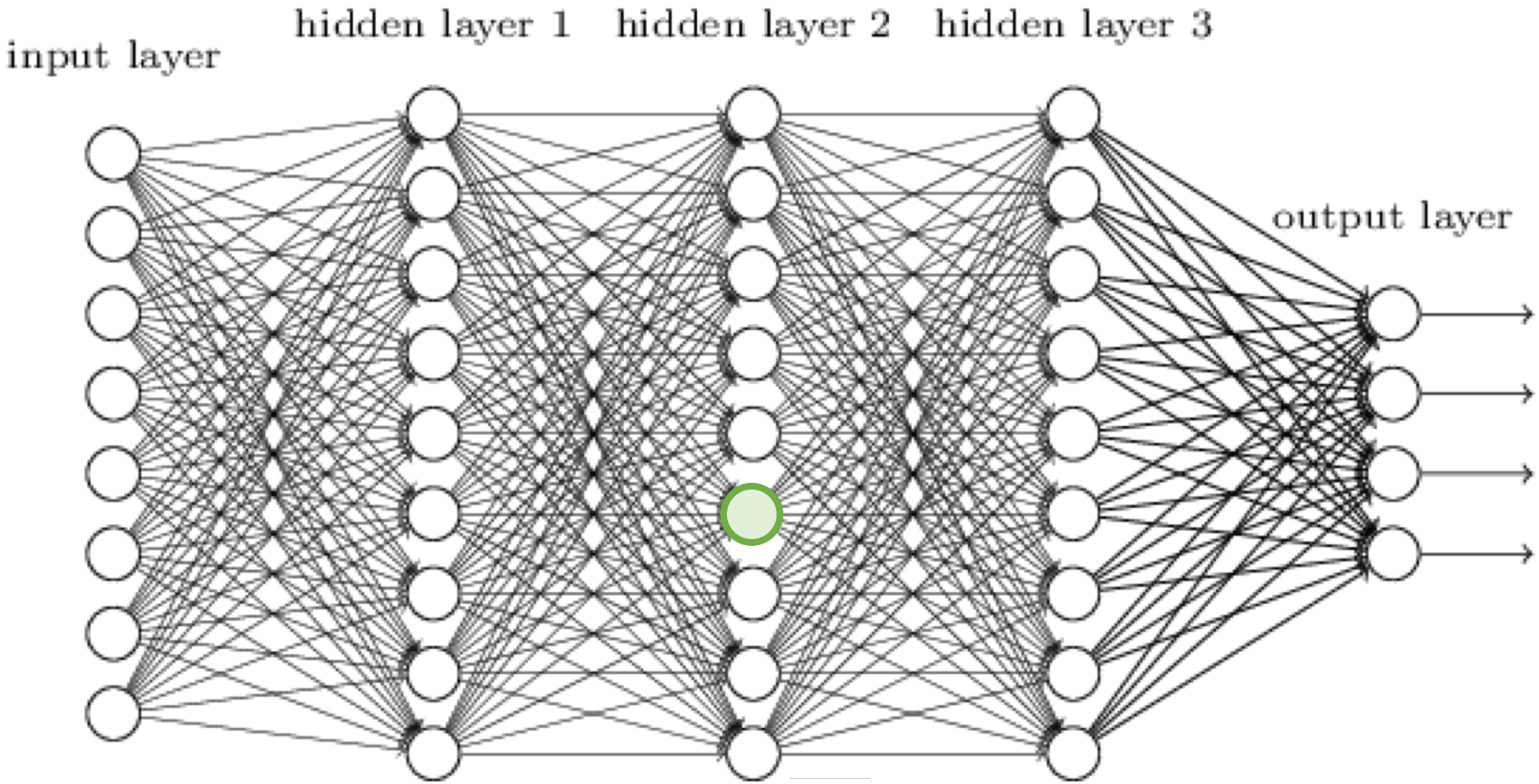
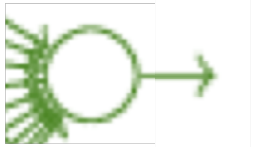


http://neuralnetworksanddeeplearning.com/chap6.html



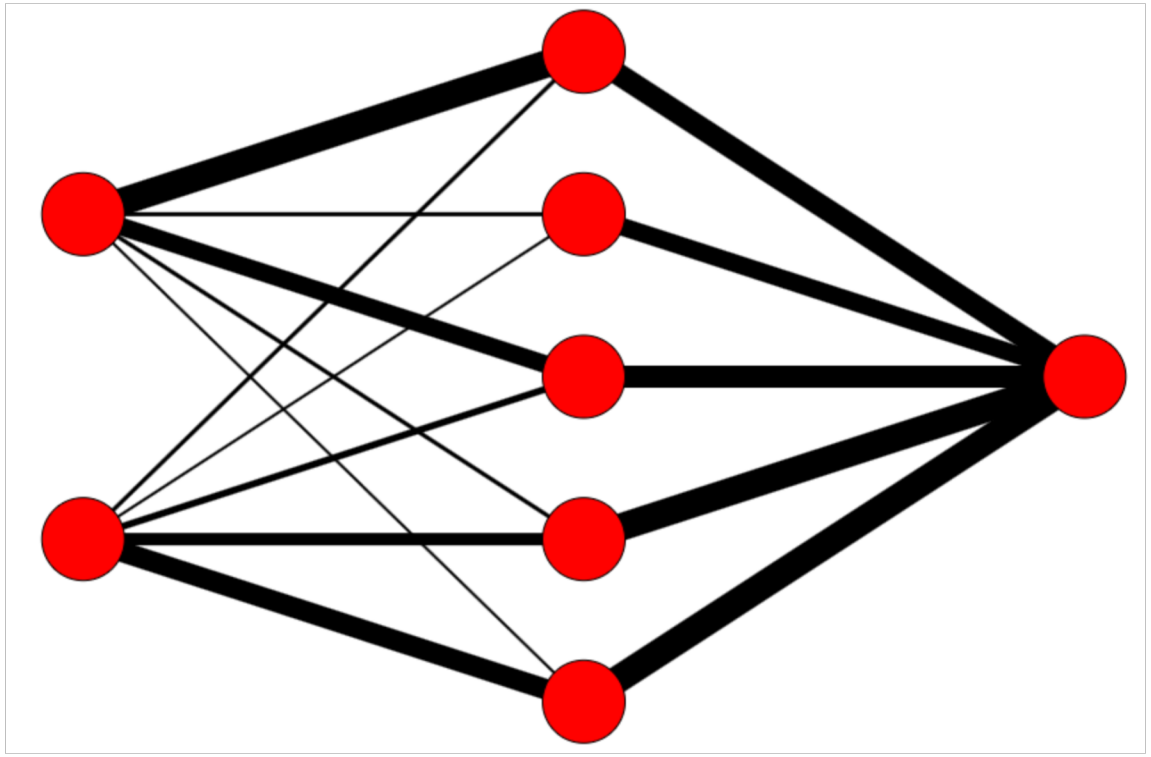
{ *inputs* }  *output*

output = **Activation** $\left(\sum(\boldsymbol{weight} * input) + \textbf{bias}\right)$

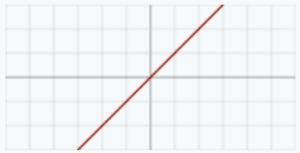
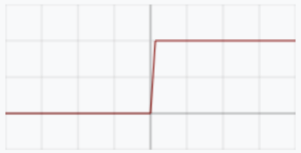
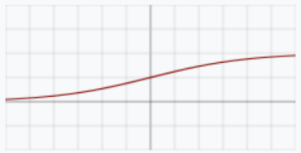
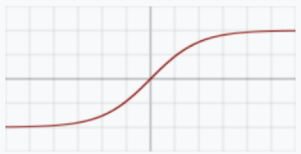
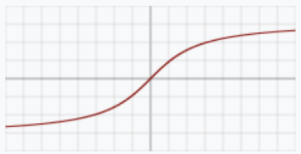
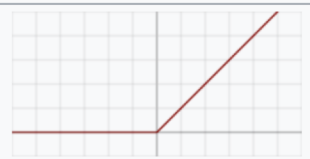
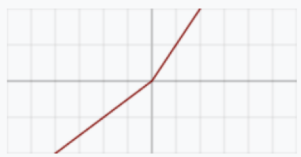
http://neuralnetworksanddeeplearning.com/chap6.html

Simple(st) network

Input 1	Input 2	Target
0	0	0
0	1	0
1	0	0
1	1	1



http://www.emergentmind.com/neural-network

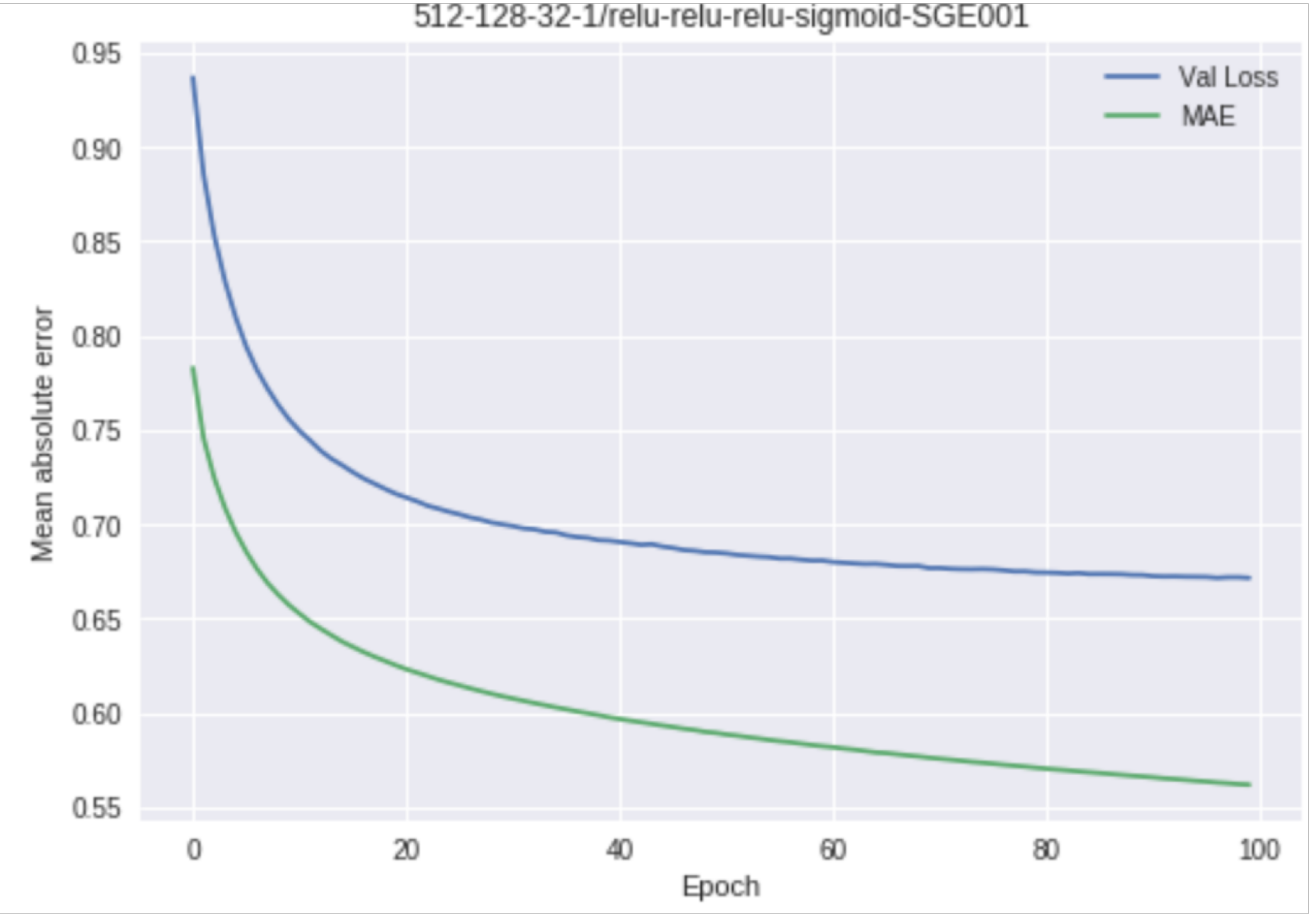
Name	Plot	Equation	Derivative (with respect to x)	Range
Identity		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$	$\{0, 1\}$
Logistic (a.k.a. Sigmoid or Soft step)		$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$ [1]	$f'(x) = f(x)(1 - f(x))$	$(0, 1)$
TanH		$f(x) = \tanh(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})}$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$	$\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$
Rectified linear unit (ReLU)[10]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$[0, \infty)$
Leaky rectified linear unit (Leaky ReLU)[11]		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$

https://en.wikipedia.org/wiki/Activation_function

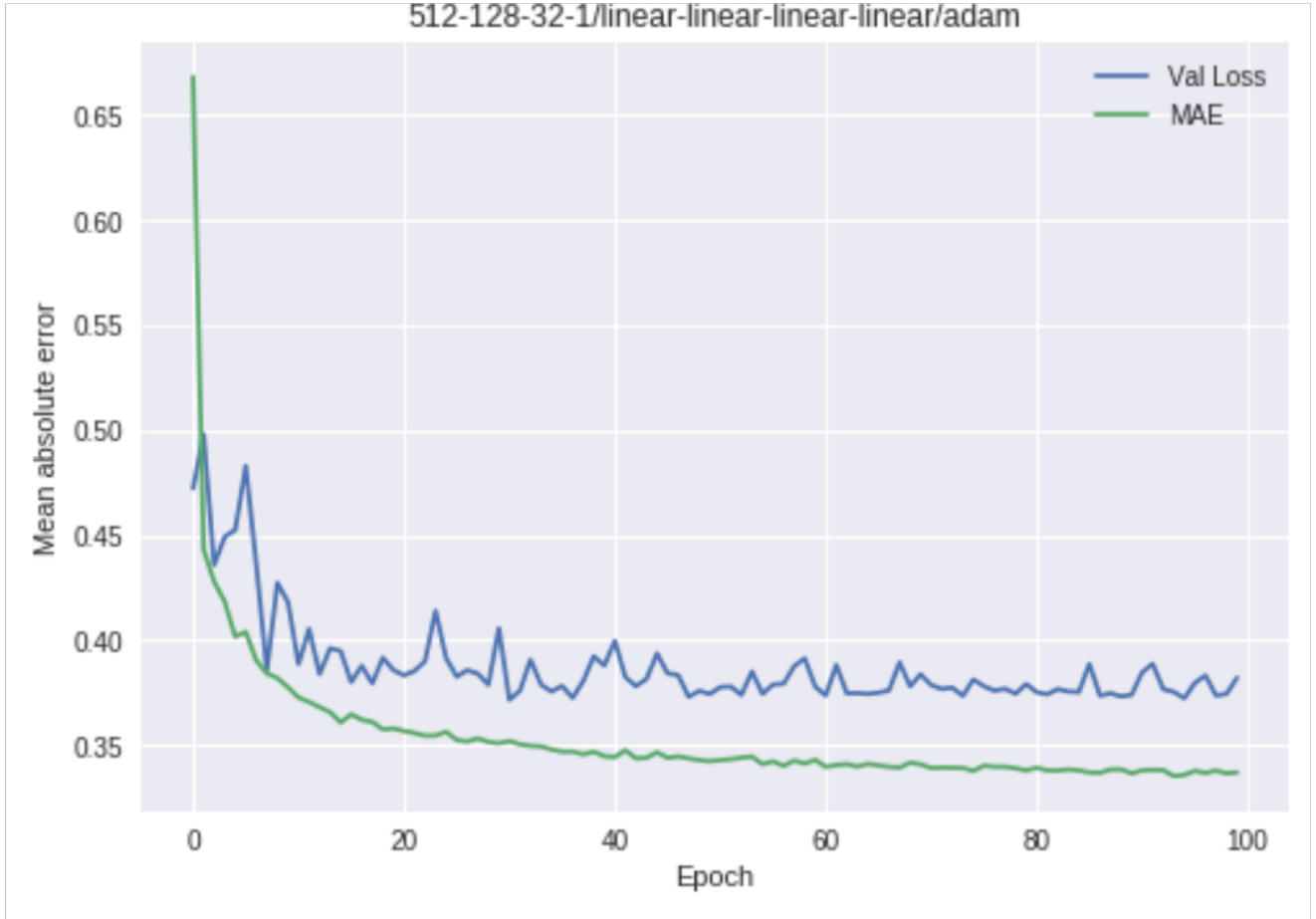
Example code and data

- https://bit.ly/2RiQesC
- https://colab.research.google.com

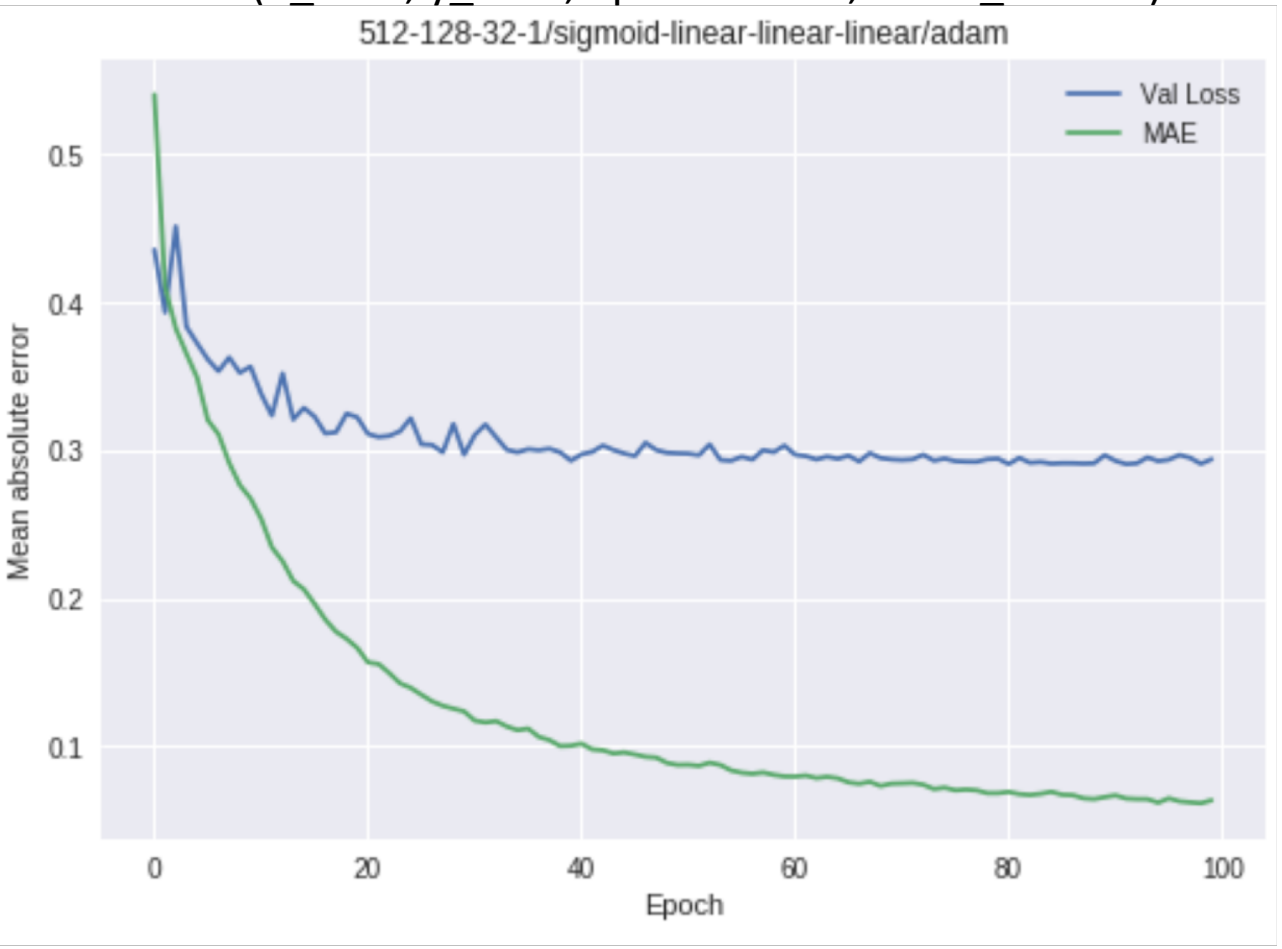
```
model = Sequential()
model.add(Dense(units=512, activation=relu', input_dim=num_columns-1))
model.add(Dense(units=128, activation='relu'))
model.add(Dense(units=32,  activation='relu'))
model.add(Dense(units=1,   activation='sigmoid'))

model.compile(loss='mae', optimizer=SGD(lr=0.001), metrics=["mae"])
model.fit(X_train, y_train, epochs=5000, batch_size=32)
```



```
model = Sequential()
model.add(Dense(units=512, activation='linear', input_dim=num_columns-1))
model.add(Dense(units=128, activation='linear'))
model.add(Dense(units=32,  activation='linear'))
model.add(Dense(units=1,   activation='linear'))

model.compile(loss='mae', optimizer='adam', metrics=["mse","mae"])
model.fit(X_train, y_train, epochs=5000, batch_size=32)
```



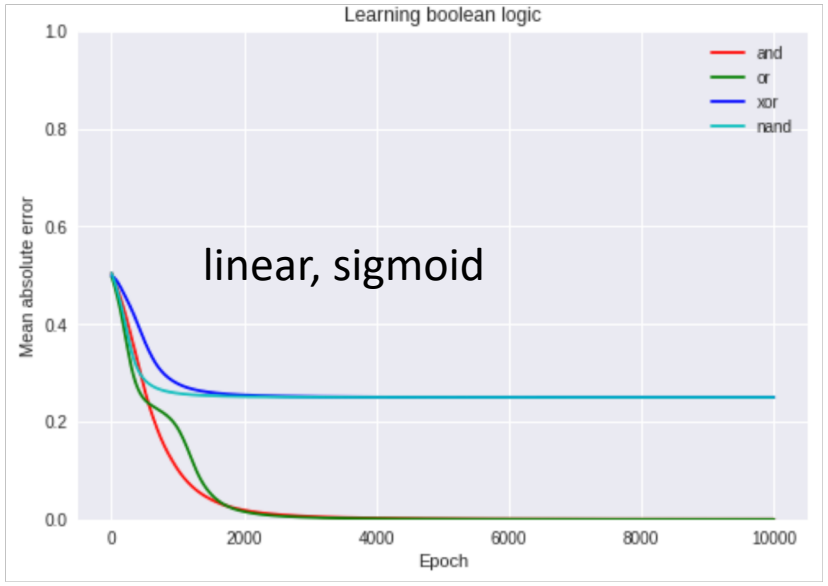
```
model = Sequential()
model.add(Dense(units=512, activation='sigmoid', input_dim=num_columns-1))
model.add(Dense(units=128, activation='linear'))
model.add(Dense(units=32,  activation='linear'))
model.add(Dense(units=1,   activation='linear'))

model.compile(loss='mae', optimizer='adam', metrics=["mse","mae"])
model.fit(X_train, y_train, epochs=5000, batch_size=32)
```

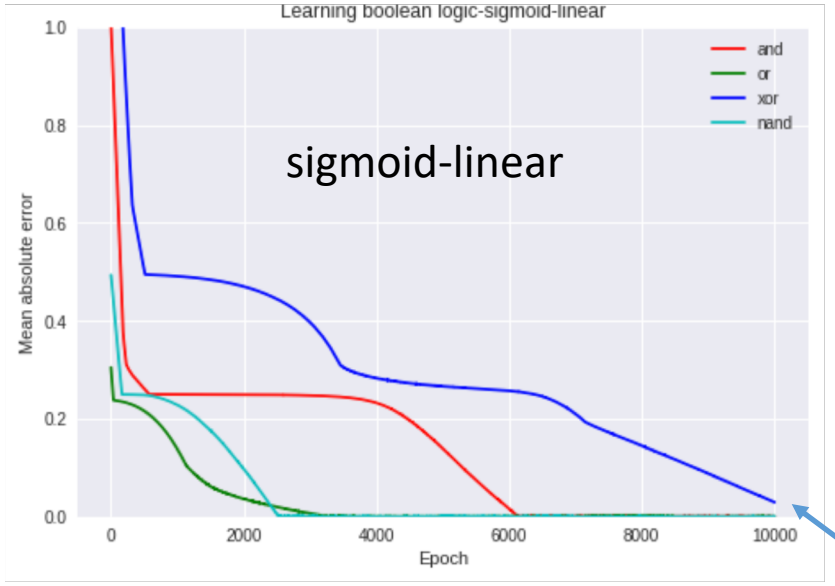


 # Learning Boolean logic

```
act1 = ‘sigmoid’
act2 = ‘linear’
optimizer = ‘adam’
epochs = 10000
batch_size = 32

boolX = numpy.array([[0,1],[1,1],[1,0],[0,0]])
andy   =  numpy.array([0,1,0,0])
ory     =  numpy.array([1,1,1,0])
xory   =  numpy.array([1,0,1,0])
nandy  =  numpy.array([1,0,1,1])

model= Sequential()
model.add(Dense(units=3, activation=act1, input_dim=2))
model.add(Dense(units=1, activation=act2))

model.compile(loss='mae', optimizer=optimizer, metrics=["mse","mae"])
h1= boolmod1.fit(boolX, andy, epochs=epochs, batch_size=batch_size, verbose=0)
h2= boolmod1.fit(boolX, ory, epochs=epochs, batch_size=batch_size, verbose=0)
h3= boolmod1.fit(boolX, xory, epochs=epochs, batch_size=batch_size, verbose=0)
h4= boolmod1.fit(boolX, nandy, epochs=epochs, batch_size=batch_size, verbose=0)
```

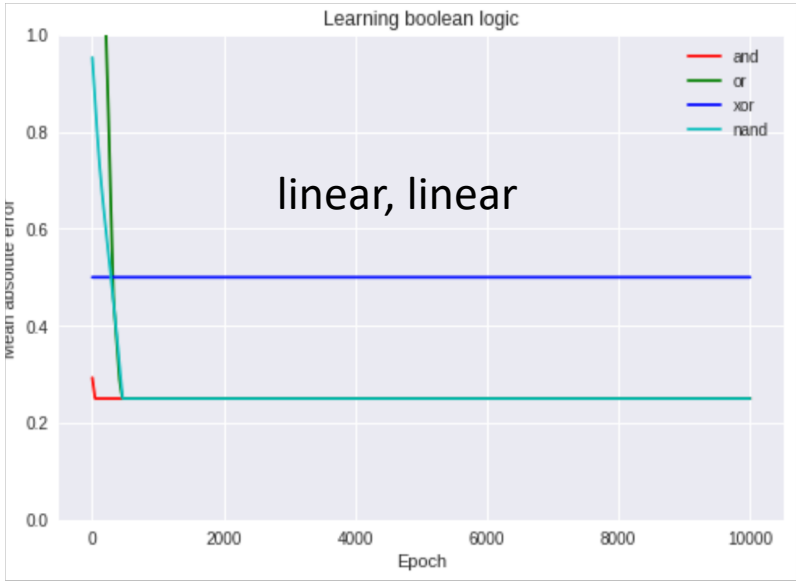
Activation functions matter

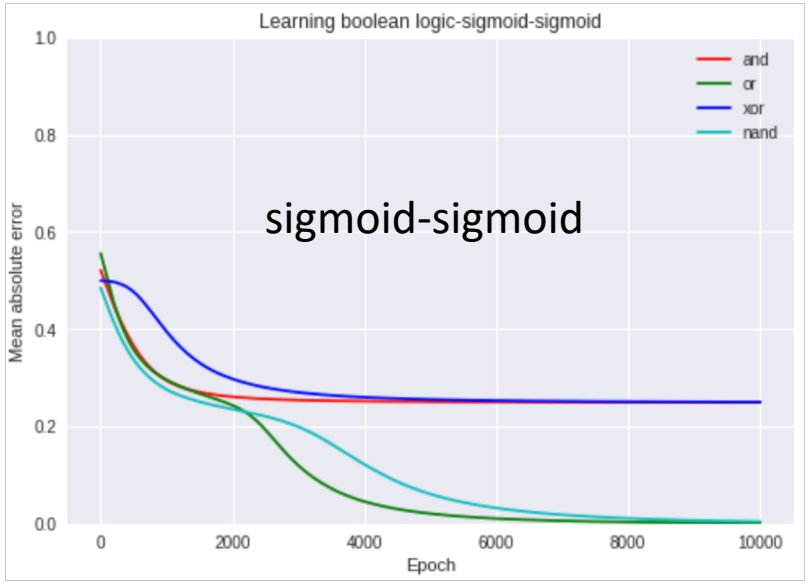




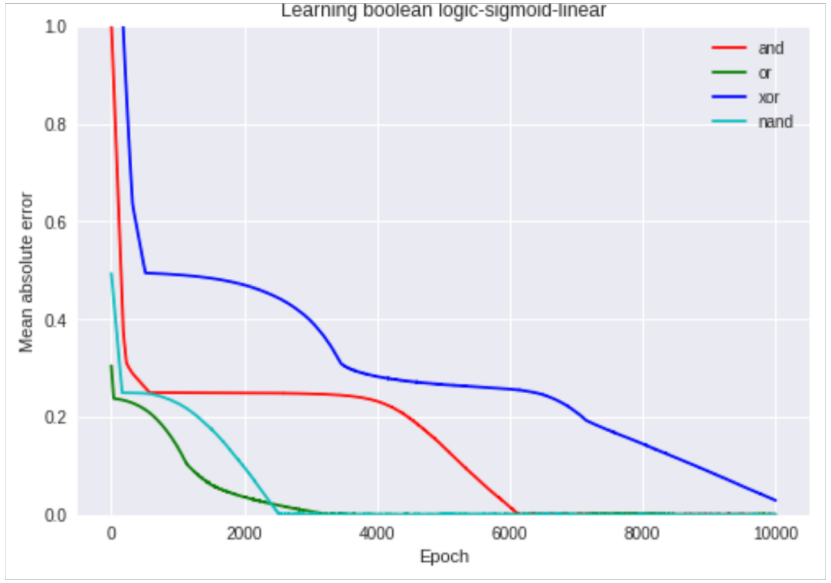
All with:
batch-size = 32
optimizer=adam

It gets there at around2,000 epochs

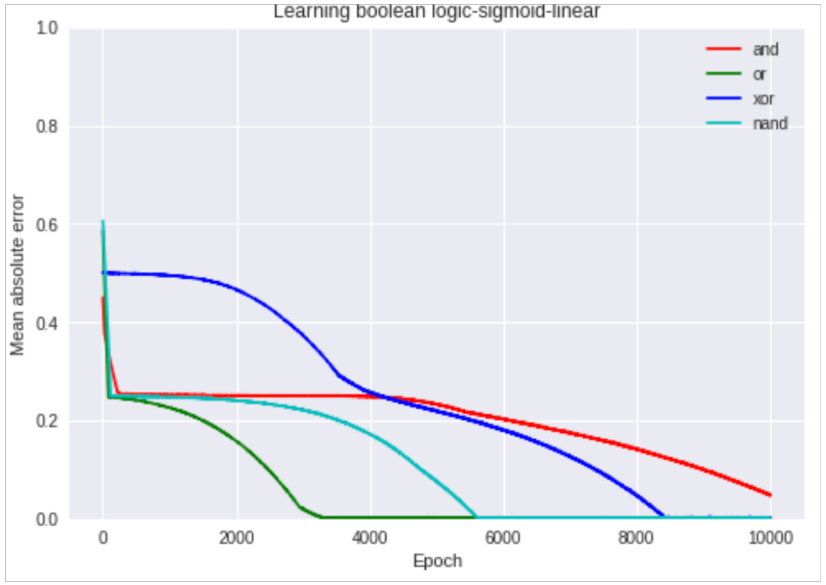




Batch size matters



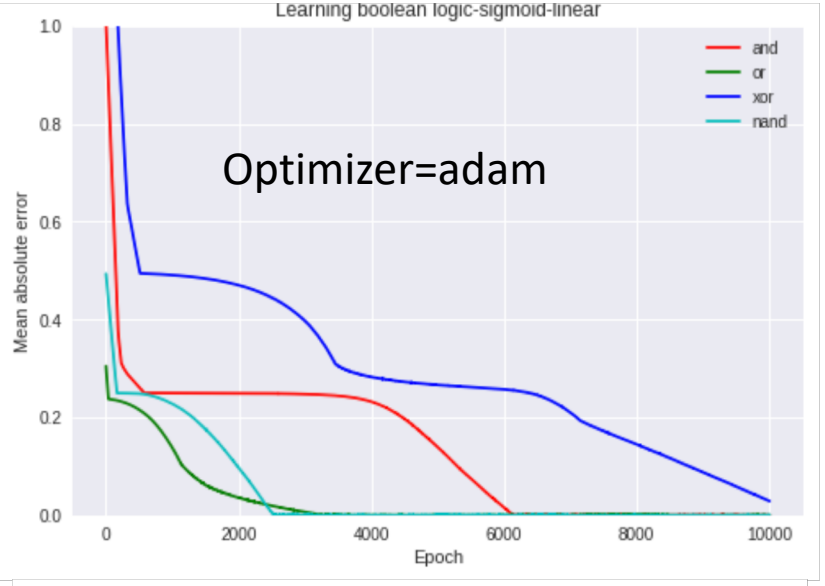
batch-size = 32



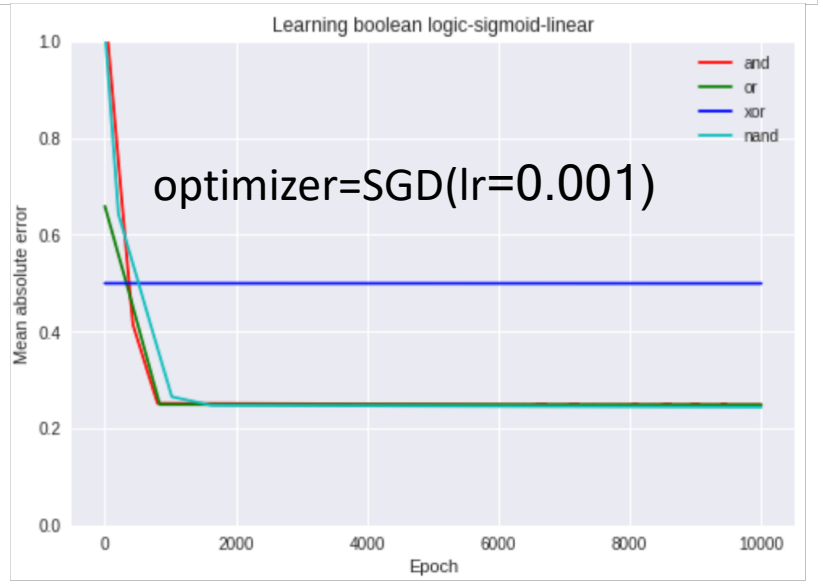
batch-size = 1

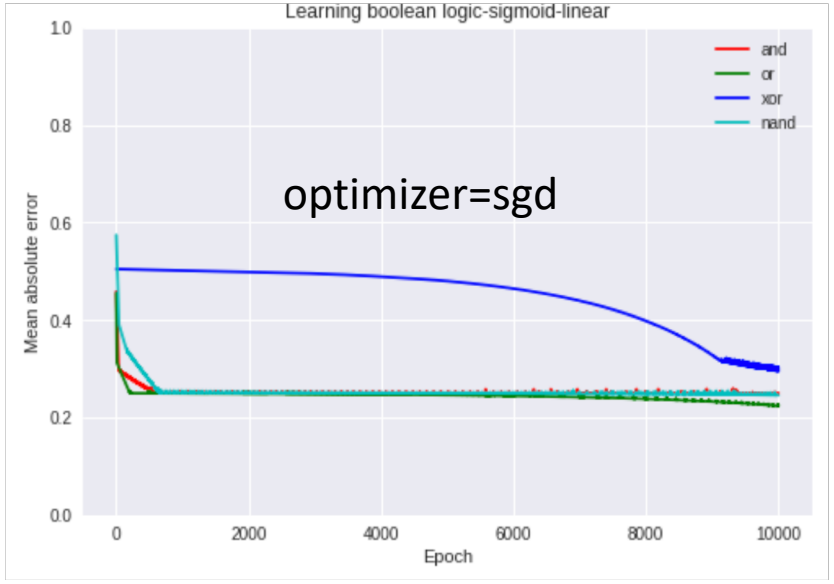
All with:
sigmoid-linear
optimizer=adam

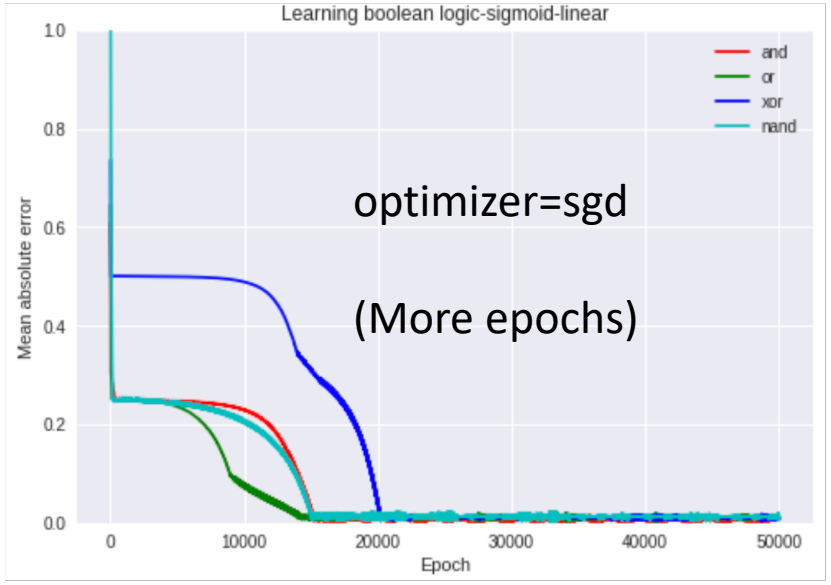
Optimizer matters



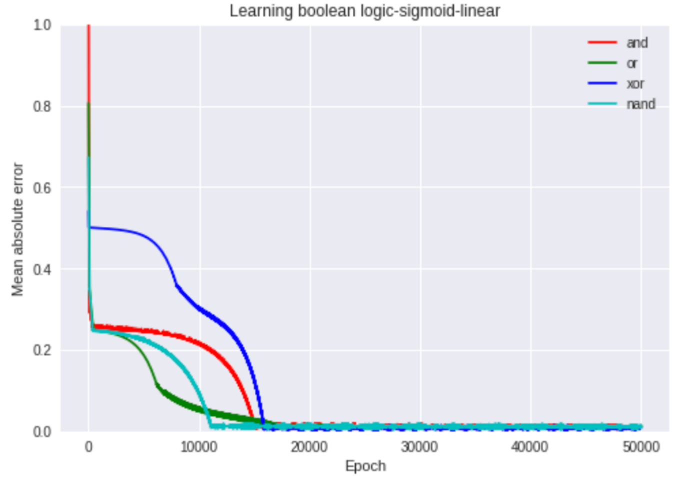
All with:
sigmoid-linear
batch-size=32

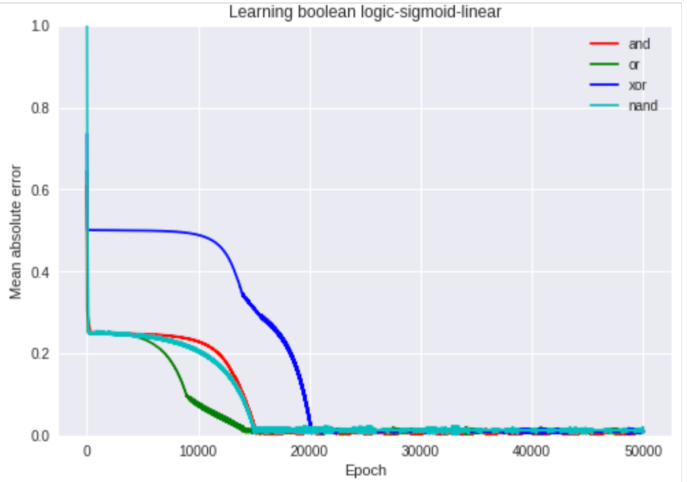






Random seed matters (at least for sgd)





Three runs, all with:
- sigmoid-linear
- batch-size=32
- Optimizer = sgd