Deep Learning

Basics and Intuition

Constantin Gonzalez

Principal Solutions Architect, Amazon Web Services

glez@amazon.de

November 2017

Deductive Logic

How often have I said to you that when you have eliminated the impossible, whatever remains, however improbable, must be the truth?

https://en.wikiquote.org/wiki/Sherlock_Holmes

Deductive Logic

P	Q	$P \wedge Q$	$P \vee Q$	$P \therefore Q$
T	T	T	T	T
T	F	F	T	F
F	T	F	T	T
F	F	F	F	T

Deductive Logic

P	Q	P ∧ Q	P ∨ Q	P ∴ Q
T	T	T	T	T
T	F	F	T	F
F	T	F	T	T
F	F	F	F	T

- $P = T \wedge Q = T \therefore P \wedge Q = T$
- $P \wedge Q \therefore P \rightarrow Q; \sim P \therefore P \rightarrow Q$

- $$\begin{array}{l} P \rightarrow Q \\ P \\ \hline \therefore Q \end{array}$$

Complexities of Deductive Logic

Predicates (n)	# of rows in truth table (2^n)
1	1
2	4
3	8
…	…
10	1024
…	…
20	1,048,576
…	…
30	1,073,741,824

Complexities of Deductive Logic

Predicates (n)	# of rows in truth table (2^n)
1	1
2	4
3	8
…	…
10	1024
…	…
20	1,048,576
…	…
30	1,073,741,824

$$[[\Omega \rightarrow (\mathrm{E} \wedge \mathrm{H})] \vee (\Gamma \vee \sim\Gamma)]$$

$$(\Phi \wedge \sim\Phi)$$

$$\therefore$$

F

Complexities of Deductive Logic

Predicates (n)	**# of rows in truth table (2^n)**
1	1
2	4
3	8
…	…
10	1024
…	…
20	1,048,576
…	…
30	1,073,741,824

- Useless for dealing with partial information.

$$[[\Omega \rightarrow (\mathrm{E} \land \mathrm{H})] \lor (\Gamma \lor \sim\Gamma)] \land (\Phi \land \sim\Phi) \therefore F$$

Complexities of Deductive Logic

Predicates (n)	# of rows in truth table (2^n)
1	1
2	4
3	8
…	…
10	1024
…	…
20	1,048,576
…	…
30	1,073,741,824

- Useless for dealing with partial information.
- Useless for dealing with non-deterministic inference.

$$[[\Omega \rightarrow (\mathrm{E} \wedge \mathrm{H})] \vee (\Gamma \vee \sim\Gamma)] \wedge (\Phi \wedge \sim\Phi) \therefore F$$

Complexities of Deductive Logic

Predicates (n)	# of rows in truth table (2^n)
1	1
2	4
3	8
…	…
10	1024
…	…
20	1,048,576
…	…
30	1,073,741,824

- Useless for dealing with partial information.
- Useless for dealing with non-deterministic inference.
- We perform very poorly in computing complex logical statements intuitively.

$$[[\Omega \rightarrow (\mathrm{E} \land \mathrm{H})] \lor (\Gamma \lor \sim\Gamma)] \land (\Phi \land \sim\Phi) \therefore F$$

Complexities of Deductive Logic

Predicates (n)	# of rows in truth table (2^n)
1	1
2	4
3	8
…	…
10	10[illegible]4
…	…
20	1,048,57[illegible]
…	…
30	1,073,741,824

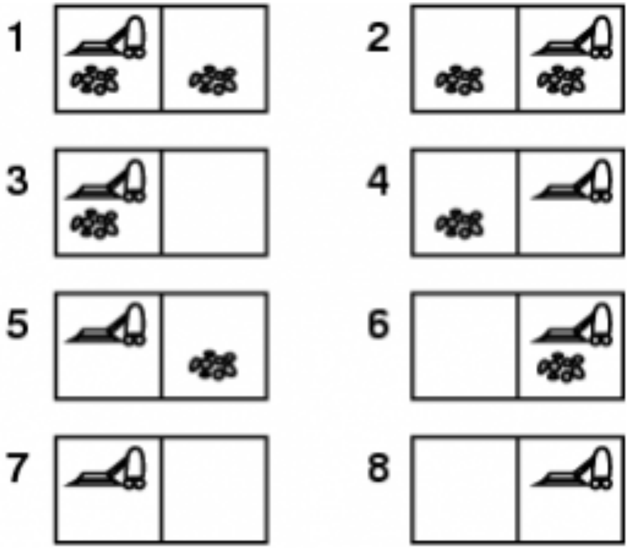
$$[[\Omega \rightarrow (\mathrm{E} \wedge \mathrm{H})] \vee (\Gamma \vee \sim\Gamma)] \wedge (\Phi \wedge \sim\Phi)$$

http://bit.ly/2rwtb0o

- Useless for dealing with partial information.
- Useless for dealing with non-deterministic inferrence.
- We perfoerm very poorly in computing the complex logical statemetns intuitively.

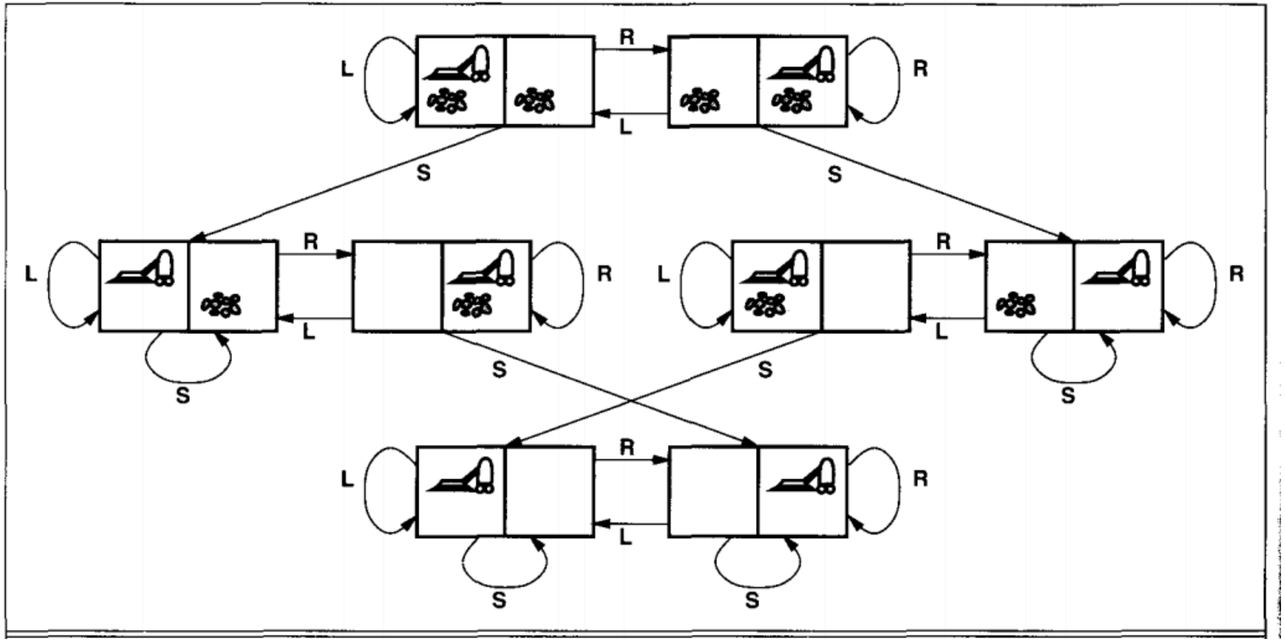
aws

Search Trees and Graphs



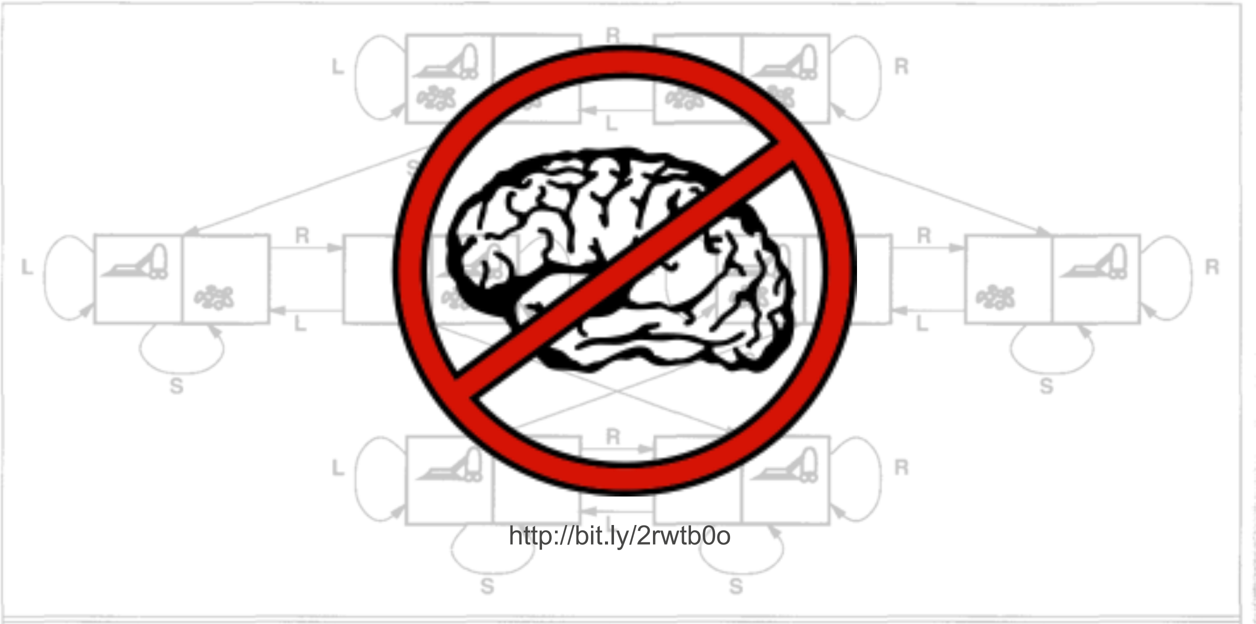
http://aima.cs.berkeley.edu/

Search Trees and Graphs



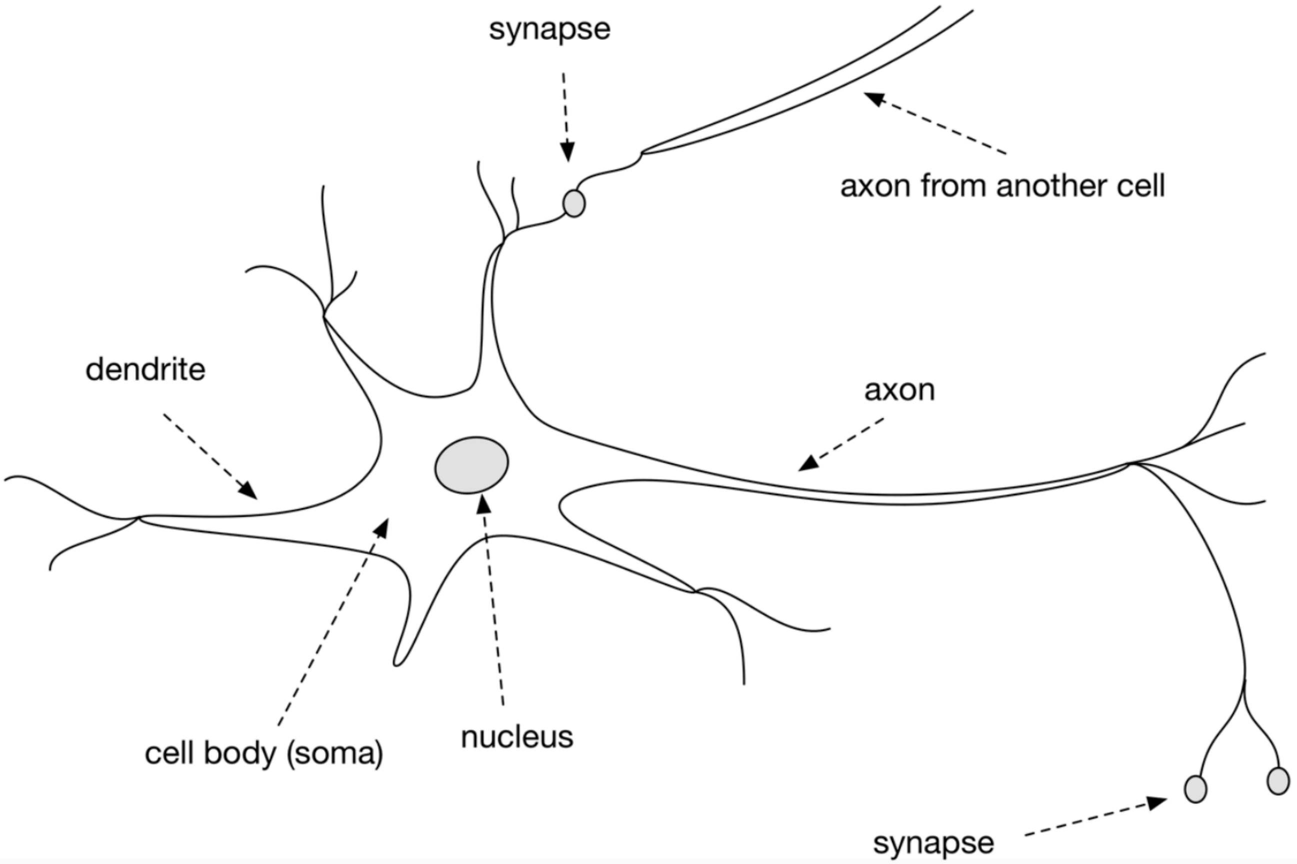
http://aima.cs.berkeley.edu/

Search Trees and Graphs

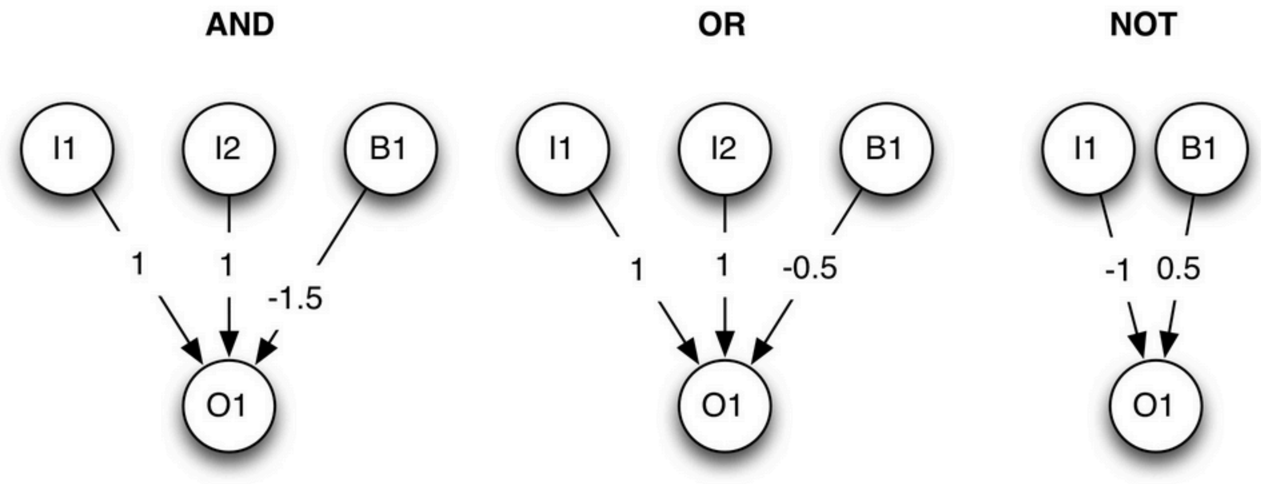


http://aima.cs.berkeley.edu/

Neurobiology



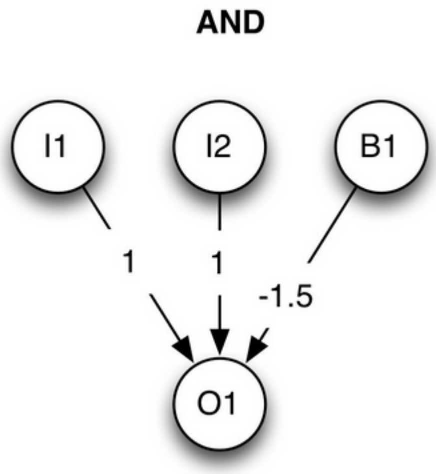
Perceptron



$$f(x_i, w_i) = \Phi(b + \Sigma_i(w_i . x_i))$$

$$\Phi(x) = \begin{cases} 1, if\ x \geq 0.5 \\ 0, if\ x < 0.5 \end{cases}$$

Example of Perceptron

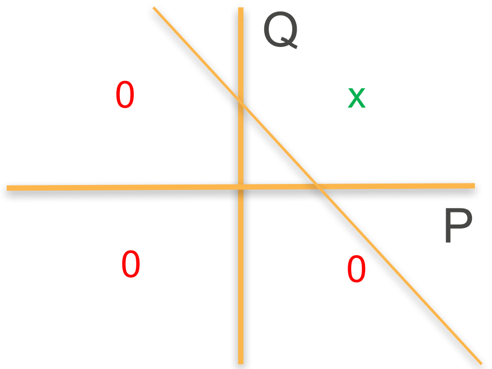


$$I1 = I2 = B1 = 1$$
$$O1 = (1x1) + (1x1) + (-1.5) = 0.5 \therefore \Phi(O1) = 1$$

$$I2 = 0\ ;\ I1 = B1 = 1$$
$$O1 = (1x1) + (0x1) + (-1.5) = -0.5 \therefore \Phi(O1) = 0$$

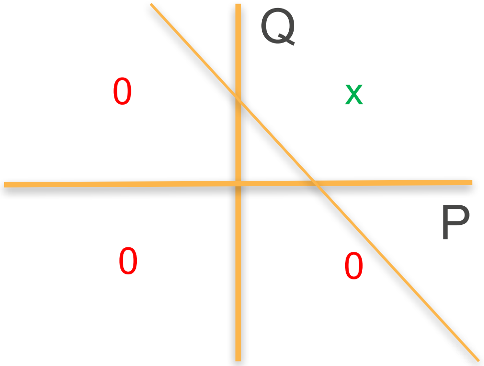
Linearity and Non-Linear Solutions

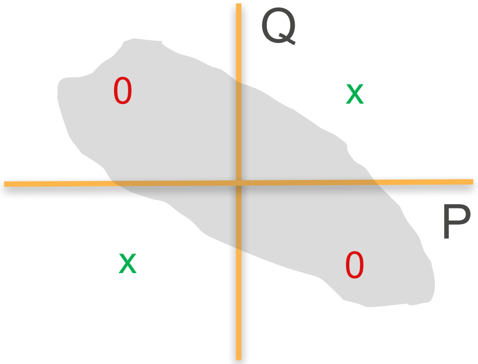
P	Q	P ∧ Q
T	T	T
T	F	F
F	T	F
F	F	F



Linearity and Non-Linear Solutions

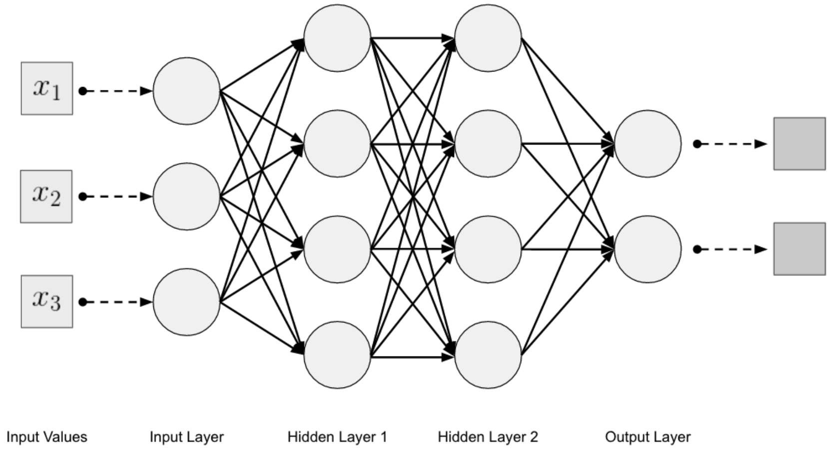
P	Q	P ∧ Q	P ⊕ Q
T	T	T	T
T	F	F	F
F	T	F	F
F	F	F	T





Multi-Layer Perceptron

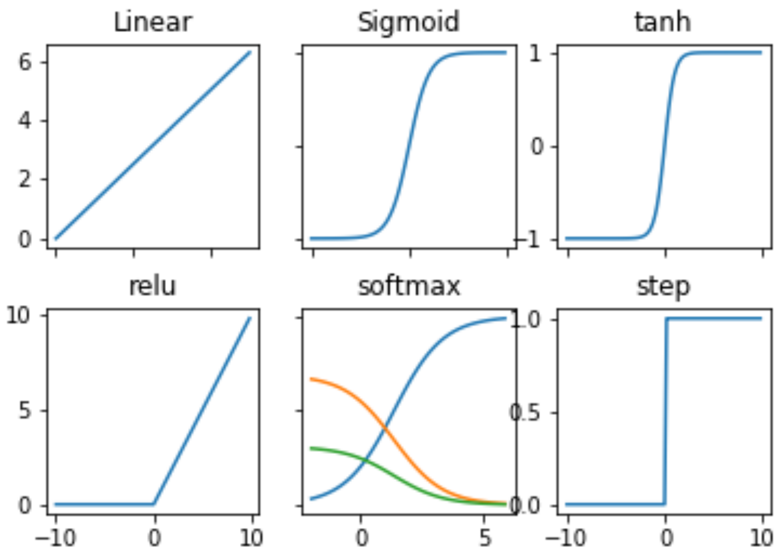
- A feedforward neural network is a biologically inspired **classification** algorithm.
- It consist of a (possibly large) number of simple **neuron-like processing *units***, organized in *layers*.
- Every unit in a layer is **connected with all the units in the previous layer.**
- Each **connection** has a weight that **encodes the knowledge** of the network.
- Data enters from the **input** layer and arrives at the **output** through **hidden** layers.
- There **is no feed back** between the layers



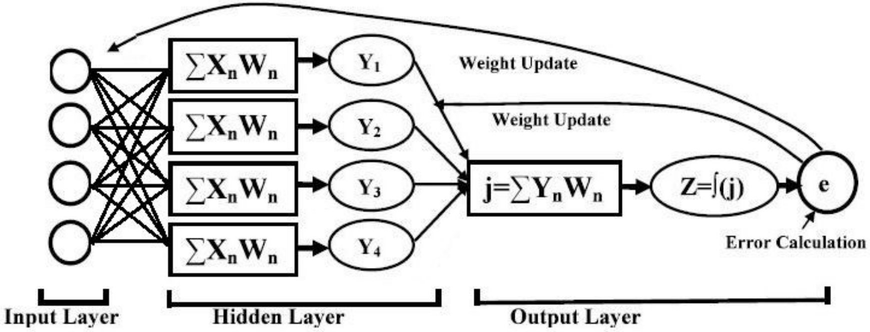
Fully-Connected *Multi-Layer Feed-Forward Neural Network Topology*

Activation Function (Φ)

- Linear: $f(x) = x$
- Sigmoid: $f(x) = \frac{1}{1+e^{-x}}$
- tanh: $f(x) = \frac{2}{1+e^{-2x}} - 1$
- relu: $f(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$
- softmax: $f(\vec{x})_i = \frac{e^{x_i}}{\Sigma_{k=1}^{K} e^{x_k}}$
- Step: $f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$

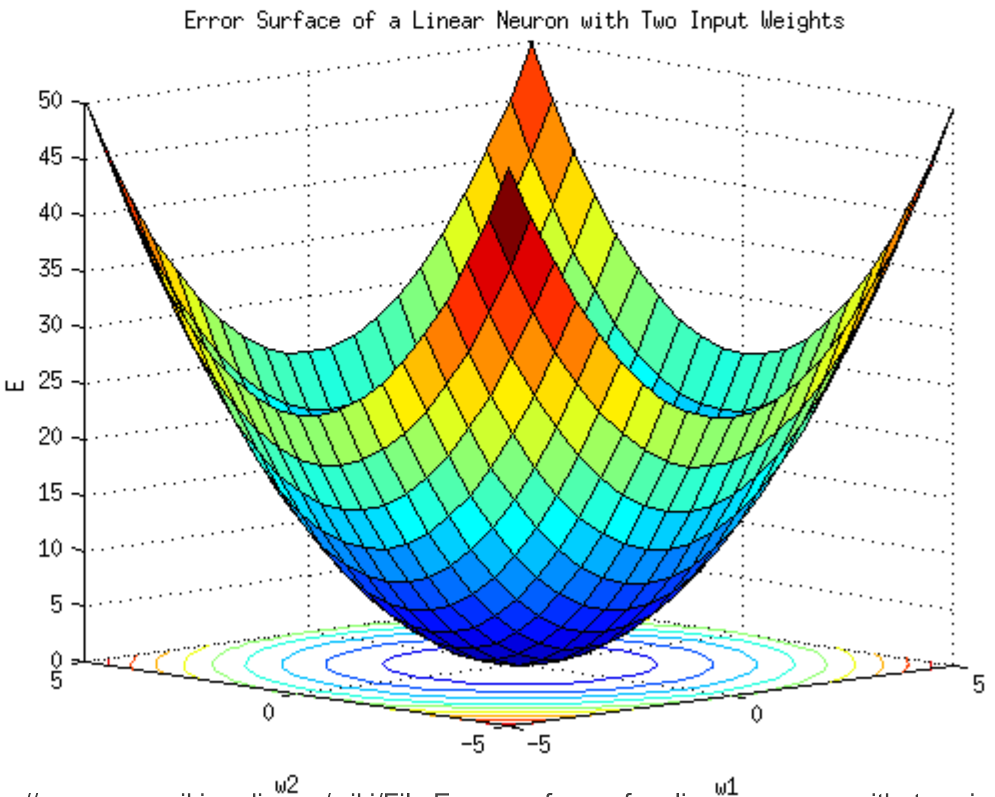


Training an ANN



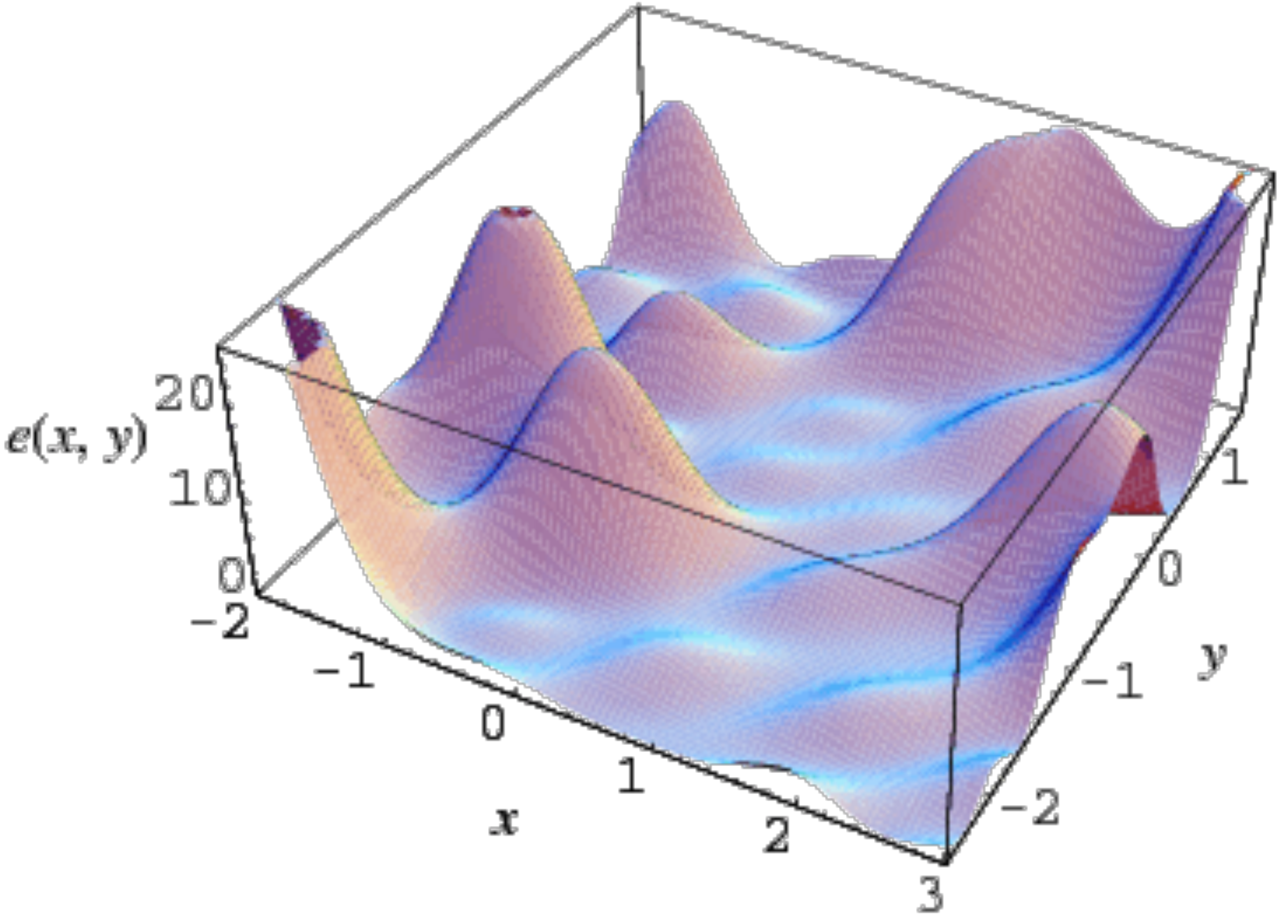
- A well-trained ANN has **weights** that amplify the signal and **dampen the noise**.
- A **bigger weig**ht signifies a **tighter correlation** between a signal and the network's outcome.
- The process of learning for any learning algorithm using weights is the process of **re-adjusting the weights and biases**
- **Back Propagation** is a popular training algorithm and is based on **distributing the blame** for the error and divide it between the contributing weights.

Error Surface



https://commons.wikimedia.org/wiki/File:Error_surface_of_a_linear_neuron_with_two_input_weights.png

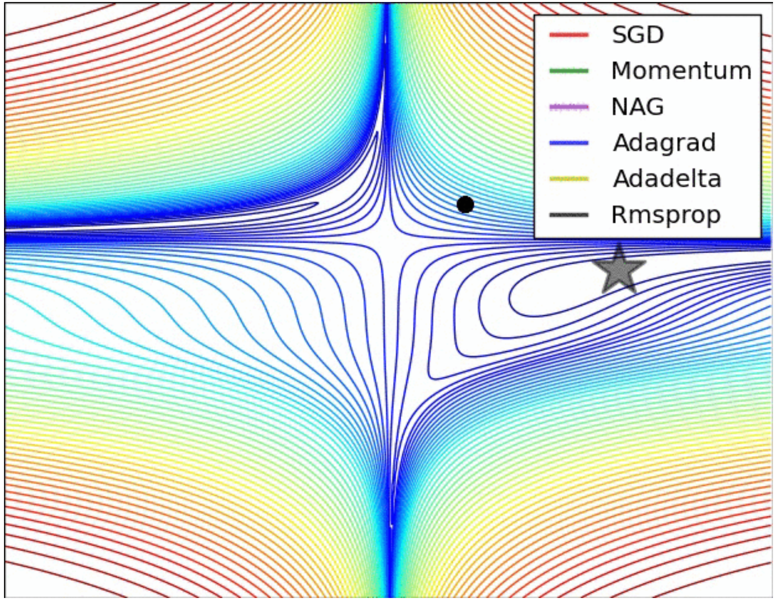
Gradient Descent

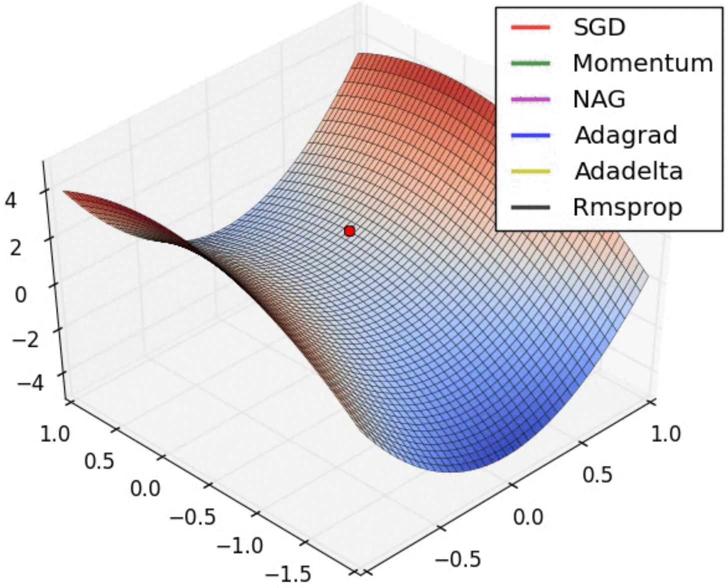


Learning Parameters

- **Learning Rate:** How far to move down in the direction of steepest gradient.
- **Online Learning:** Weights are updated at each step. Often slow to learn.
- **Batch Learning:** Weights are updated after the whole of training data. Often makes it hard to optimize.
- **Mini-Batch:** Combination of both when we break up the training set into smaller batches and update the weights after each mini-batch.

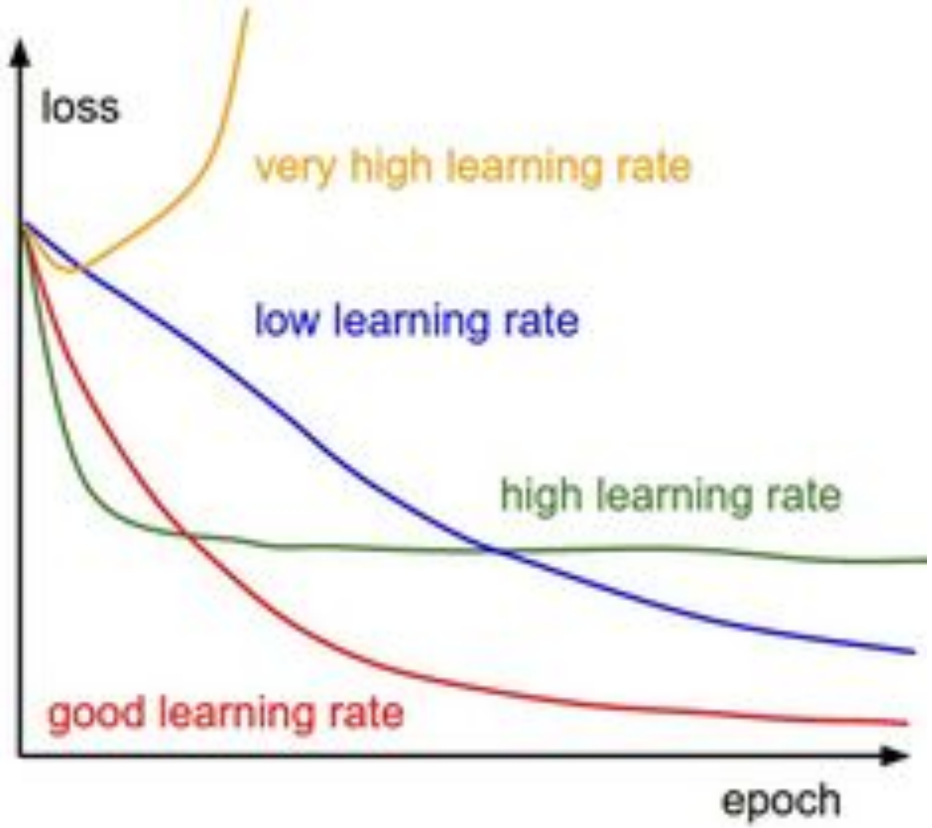
SDG Visualization



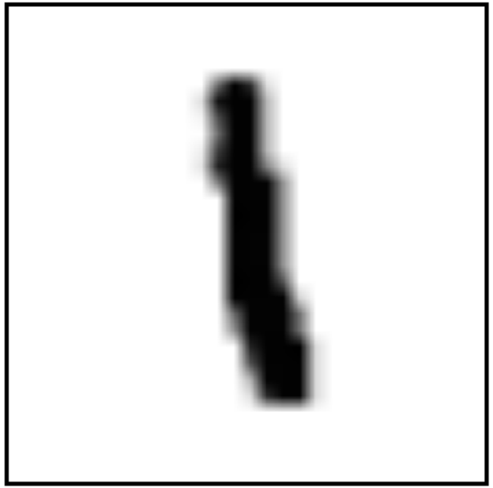


http://sebastianruder.com/optimizing-gradient-descent/

Overview of Learning Rate



Data Encoding



$$
\approx
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .6 & .8 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .5 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .7 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .9 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .3 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

https://www.tensorflow.org/get_started/mnist/beginners

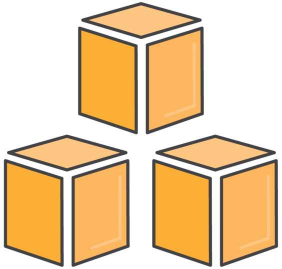
Source: Alec Radford

Apache MXNet

Why Apache MXNet?

Most Open

Accepted into the
Apache Incubator

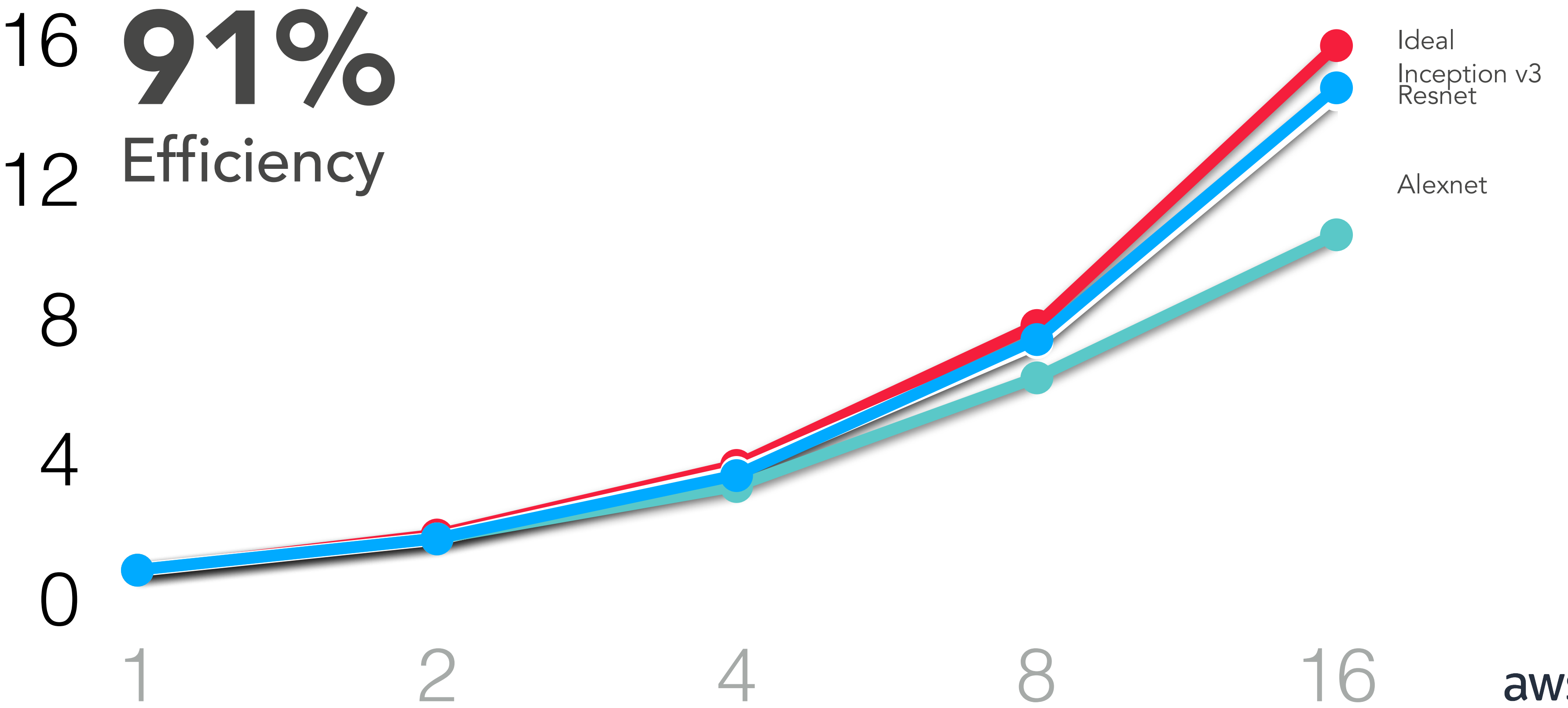


Best On AWS

Optimized for
deep learning on AWS

(Integration with AWS)

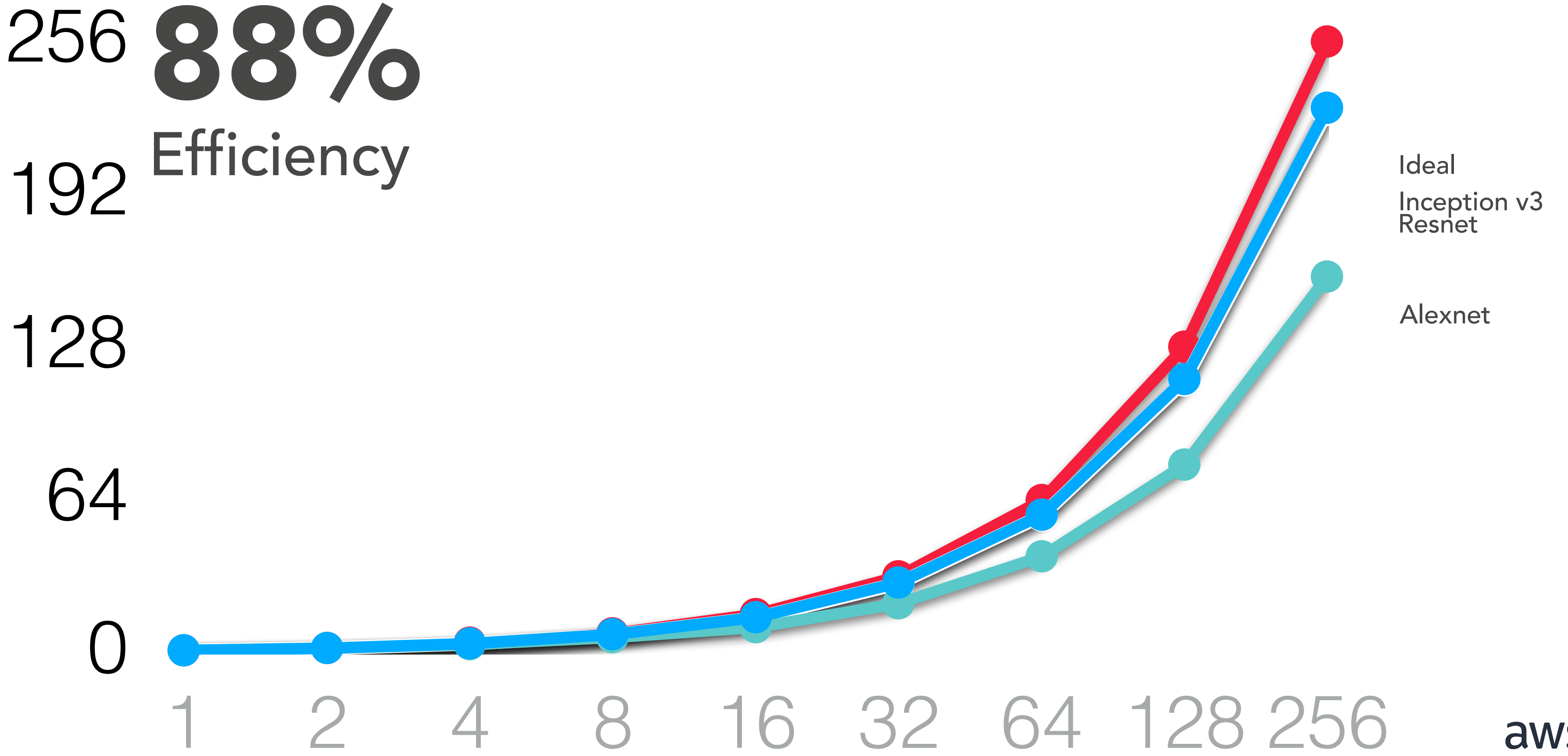
Amazon AI: Scaling With MXNet



Amazon AI: Scaling With MXNet

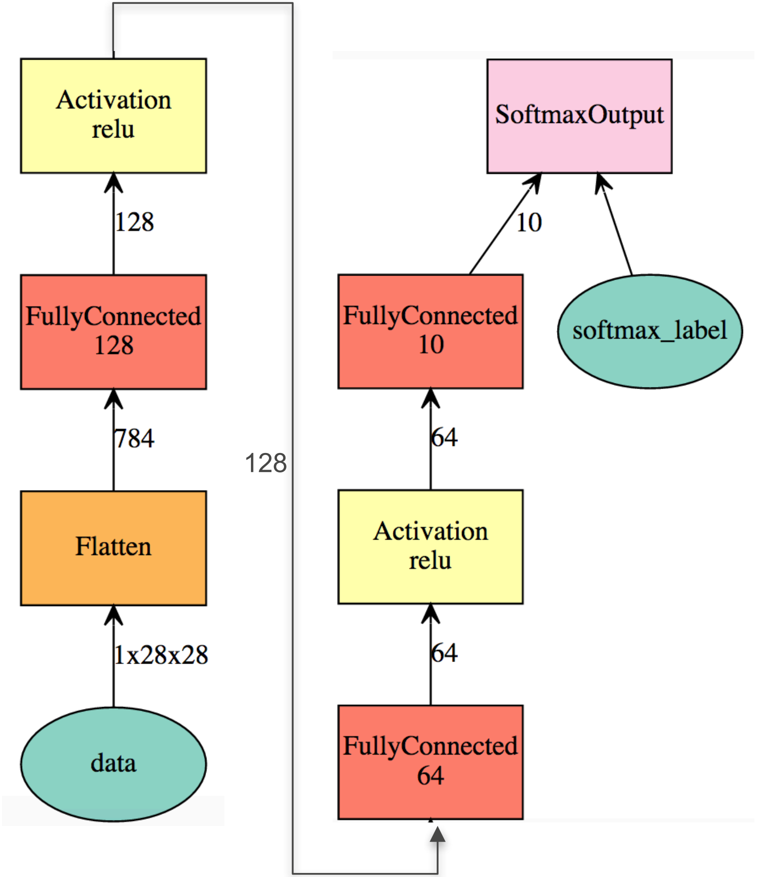
88%

Efficiency



aws

Building a Fully Connected Network in Apache MXNet



Building a Fully Connected Network in Apache MXNet

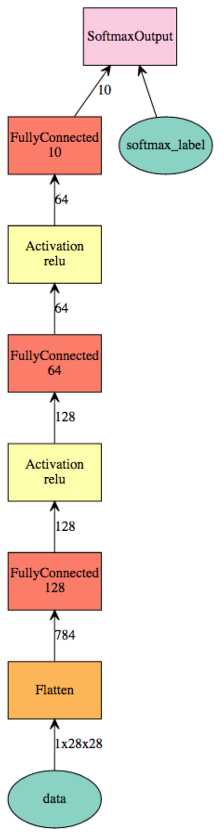
```python
# Create a place holder variable for the input data
data = mx.sym.Variable('data')
# Flatten the data from 4-D shape (batch_size, num_channel, width, height)
# into 2-D (batch_size, num_channel*width*height)
data = mx.sym.Flatten(data=data)

# The first fully-connected layer
fc1  = mx.sym.FullyConnected(data=data, name='fc1', num_hidden=128)
# Apply relu to the output of the first fully-connnected layer
act1 = mx.sym.Activation(data=fc1, name='relu1', act_type="relu")

# The second fully-connected layer and the according activation function
fc2  = mx.sym.FullyConnected(data=act1, name='fc2', num_hidden = 64)
act2 = mx.sym.Activation(data=fc2, name='relu2', act_type="relu")

# The thrid fully-connected layer, note that the hidden size should be 10, which is
fc3  = mx.sym.FullyConnected(data=act2, name='fc3', num_hidden=10)
# The softmax and loss layer
mlp  = mx.sym.SoftmaxOutput(data=fc3, name='softmax')

# We visualize the network structure with output size (the batch_size is ignored.)
shape = {"data" : (batch_size, 1, 28, 28)}
mx.viz.plot_network(symbol=mlp, shape=shape)
```



Training a Fully Connected Network in Apache MXNet

```
# @@@ AUTOTEST_OUTPUT_IGNORED_CELL
import logging
logging.getLogger().setLevel(logging.DEBUG)

model = mx.model.FeedForward(
    symbol = mlp,       # network structure
    num_epoch = 10,     # number of data passes for training
    learning_rate = 0.1 # learning rate of SGD
)
model.fit(
    X=train_iter,       # training data
    eval_data=val_iter, # validation data
    batch_end_callback = mx.callback.Speedometer(batch_size, 200) # output
)
```

Demo Time

http://localhost:9999/notebooks/mxnet-notebooks/python/tutorials/mnist.ipynb

References

- How ANN predicts Stock Market Indices? by Vidya Sagar Reddy Gopala, Deepak Ramu
- Deep Learning, O'Reilly Media Inc. ISBN: 9781491914250; By: Josh Patterson and Adam Gibson
- Artificial Intelligence for Humans, Volume 3: Deep Learning and Neural Networks, CreateSpace Independent Publishing Platform; ISBN: 1505714346; By: Jeff Heaton
- Coursera; Neural Networks for Machine Learning by Jeoff Hinton at University of Toronto
- https://www.researchgate.net/figure/267214531_fig1_Figure-1-RNN-unfolded-in-time-Adapted-from-Sutskever-2013-with-permission
- www.yann-lecun.com
- http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/
- https://github.com/cyrusmvahid/mxnet-notebooks/tree/master/python

Thank you!

Constantin Gonzalez
glez@amazon.de