Deep Learning with Apache MXNet

Julien Simon, Principal Technical Evangelist
@julsimon

Selected customers running AI on AWS

- Zillow
- NETFLIX
- Stanford
- WOLFRAM
- The Washington Post
- American Heart Association
- Carnegie Mellon
- Pinterest
- FINRA
- slack
- NVIDIA
- MapBox
- C-SPAN
- Expedia
- UNIVERSITY of WASHINGTON
- SAMSUNG SmartThings
- realnetworks
- OhioHealth
- Capital One
- GoAnimate
- HubSpot
- RNIB
- 图森 tu Simple
- duolingo
- iTranslate
- HAPPY SNAP
- intel
- freshdesk
- NASA
- zmags

aws

Questions

- What's the business problem my IT has failed to solve?
  - That's probably where Deep Learning can help
- Should I design and train my own Deep Learning model?
  - Do I have the expertise?
  - Do I have enough time, data & compute to train it?
- Should I use a pre-trained model?
  - How well does it fit my use case?
  - On what data was it trained? How close is this to my own data?
- Should I use a SaaS solution?

Amazon AI for every developer

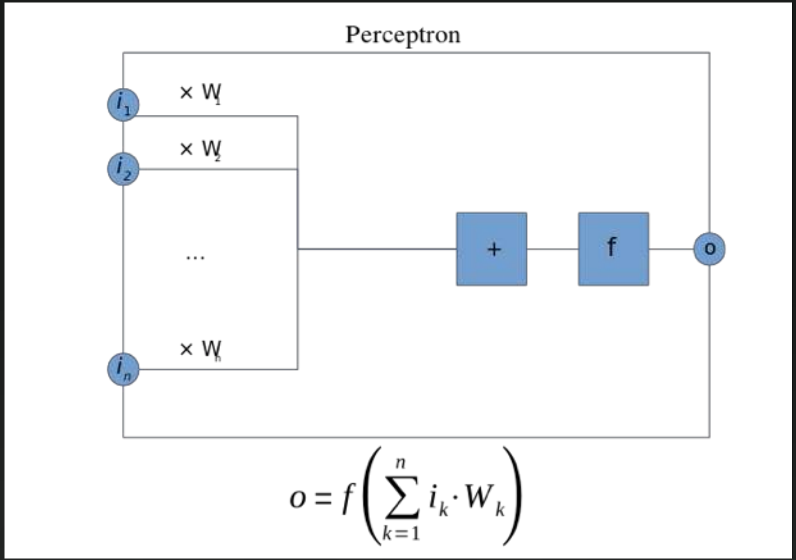
Services	**Chat** Amazon Lex	**Speech** Amazon Polly	**Vision** Amazon Rekognition			
Platforms	Amazon ML	Spark & EMR	Kinesis	Batch	ECS	
Engines	MXNet	TensorFlow	Caffe	Theano	Pytorch	CNTK
Infrastructure	GPU	CPU	IoT	Mobile		

More information at aws.amazon.com/ai

Neural Networks

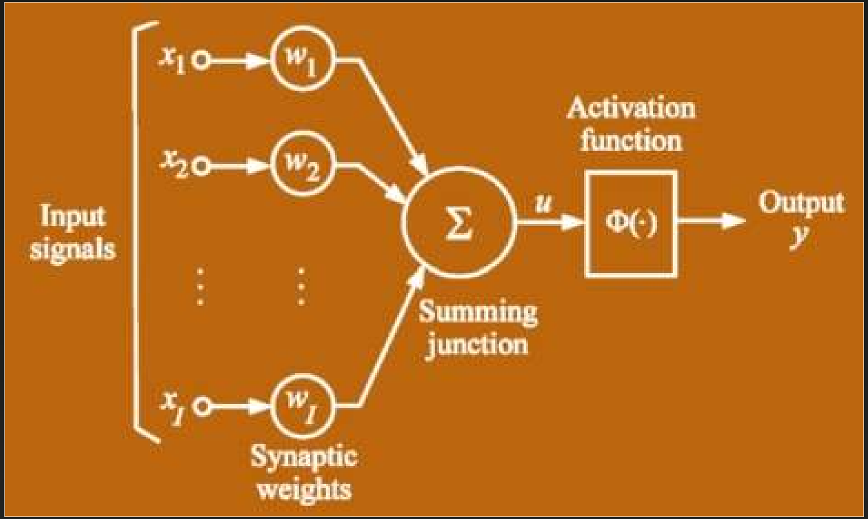
1957 – The Perceptron

Frank Rosenblatt
Artificial Intelligence pioneer



Appropriate weights are applied to the inputs, and the resulting weighted sum is passed to a function that produces the output.

The neuron



$$u = \sum_{i=1}^{n} w_i x_i$$

$x = [x_1, x_2, \dots . x_I]$

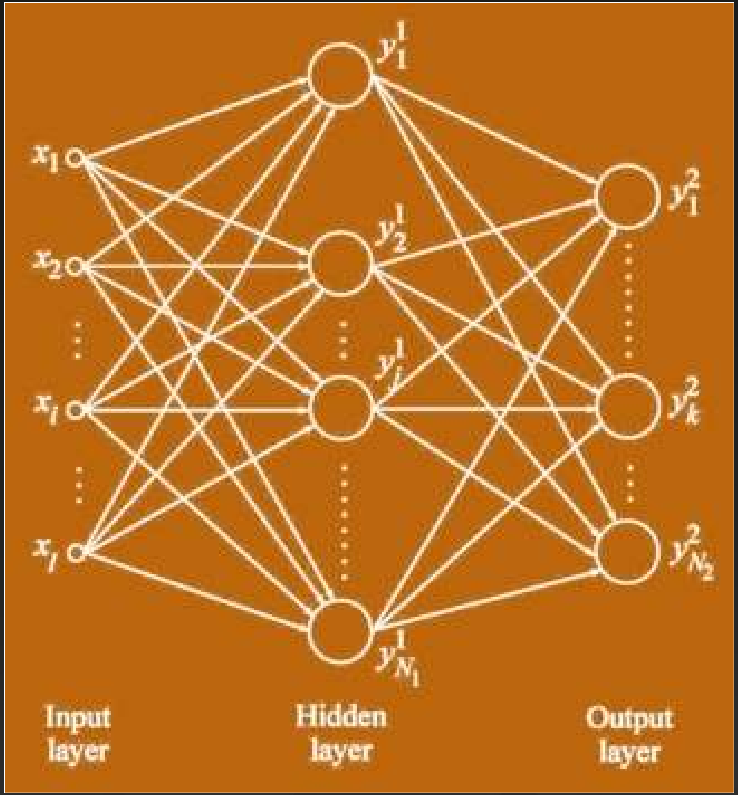
$w = [w_1, w_2, \dots . w_I]$

u =
x.w

Activation function

Name	Plot	Equation
Identity		$f(x) = x$
Binary step		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1+e^{-x}}$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$
ArcTan		$f(x) = \tan^{-1}(x)$
Softsign [7][8]		$f(x) = \frac{x}{1+\lvert x \rvert}$
Rectified linear unit (ReLU)[9]		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$

The neural network



$$x = \begin{array}{c} \overbrace{\hphantom{x_{11}, x_{12}, \ldots x_{1l}}}^{l \text{ features}} \\ \left.\begin{bmatrix} x_{11}, x_{12}, \ldots x_{1l} \\ x_{21}, x_{22}, \ldots x_{2l} \\ x_{\ldots}, x_{..}, \ldots x_{.l} \\ x_{m1}, x_{m2}, \ldots x_{ml} \end{bmatrix}\right\} \text{m samples} \end{array}$$

$$y = \left.\begin{bmatrix} y_1 \\ y_2 \\ y_{\ldots} \\ y_m \end{bmatrix}\right\} \text{m labels}$$

$$\textbf{Accuracy} = \frac{\textbf{Number of correct predictions}}{\textbf{Total number of predictions}}$$

The training process

- The difference between the predicted output and the actual output (aka *ground truth*) is called the prediction loss.

- There are different ways to compute it (loss functions).

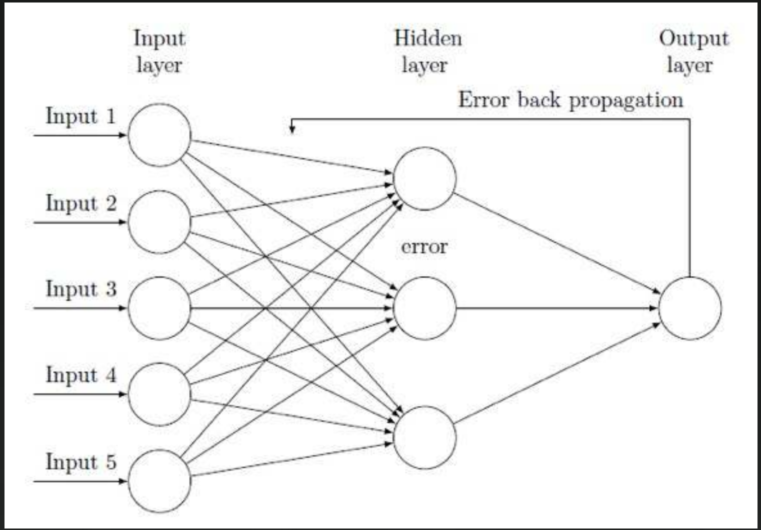
- The purpose of training is to iteratively minimize loss and maximize accuracy for a given data set.

- We need a way to adjust weights (aka parameters) in order to gradually minimize loss

  → Backpropagation + optimization algorithm

1974 – Backpropagation

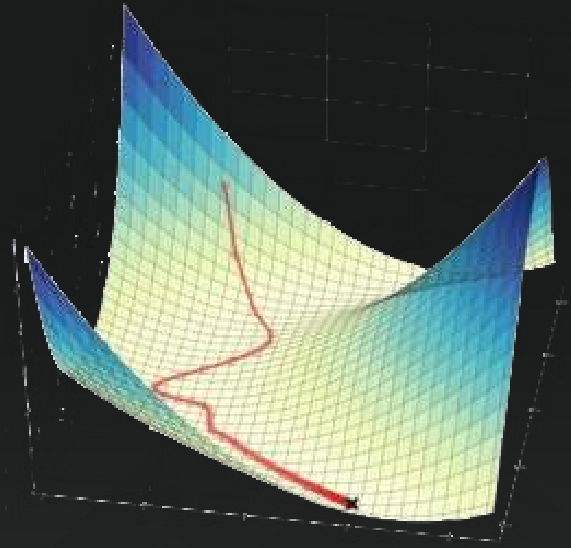
Paul Werbos
Artificial Intelligence pioneer
IEEE Neural Network Pioneer Award



The back-propagation algorithm acts as an error correcting mechanism at each neuron level, thereby helping the network to learn effectively.

Stochastic Gradient Descent (SGD)

Imagine you stand on top of a mountain with skis strapped to your feet. You want to get down to the valley as quickly as possible, but there is fog and you can only see your immediate surroundings. How can you get down the mountain as quickly as possible? You look around and identify the steepest path down, go down that path for a bit, again look around and find the new steepest path, go down that path, and repeat—this is exactly what gradient descent does.



Tim Dettmers
University of Lugano
2015

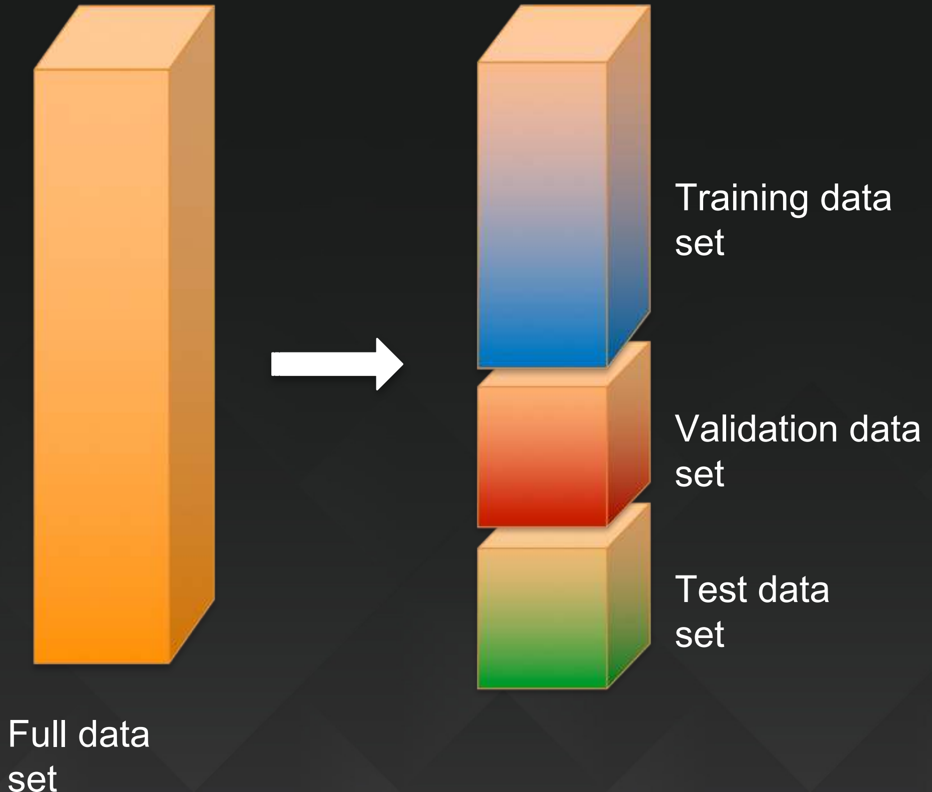
The « step size » is called the learning rate

Source: https://devblogs.nvidia.com/parallelforall/deep-learning-nutshell-history-training/

There is such a thing as « learning too well »

- If a network is large enough and given enough time, it will perfectly learn a data set (universal approximation theorem).
- But what about new samples? Can it also predict them correctly?
- Does the network generalize well or not?
- To prevent overfitting, we need to know when to stop training.
- The training data set is not enough.

Data sets

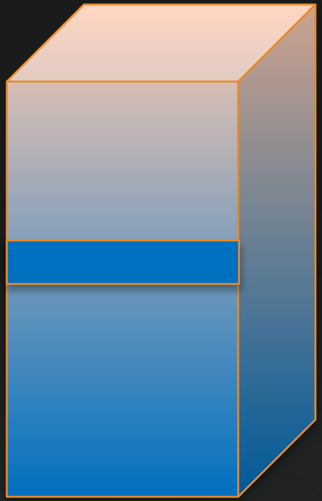


Training set: this data set is used to adjust the weights on the neural network.

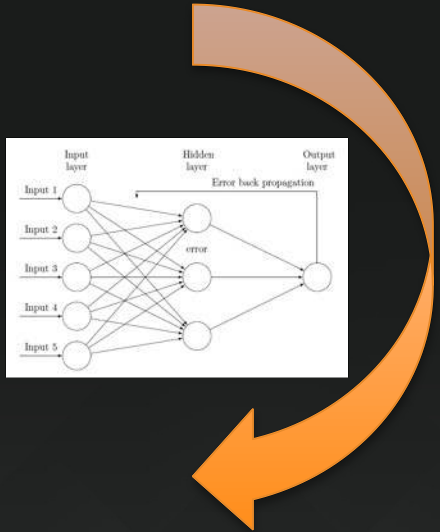
Validation set: this data set is used to minimize overfitting. You're not adjusting the weights of the network with this data set, you're just verifying that any increase in accuracy over the training data set actually yields an increase in accuracy over a data set that has not been shown to the network before.

Testing set: this data set is used only for testing the final weights in order to benchmark the actual predictive power of the network.

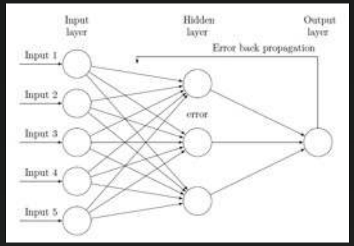
Training



Training data set



Training

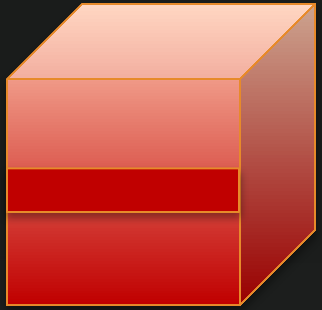


Trained neural network

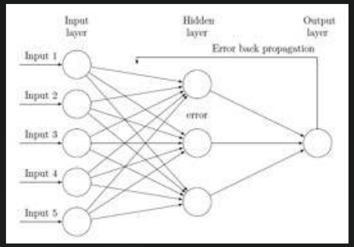
Number of epochs
Batch size
Learning rate

Hyper parameters

Validation



Validation Data set



Trained neural network

Prediction at the end of each epoch

Validation accuracy

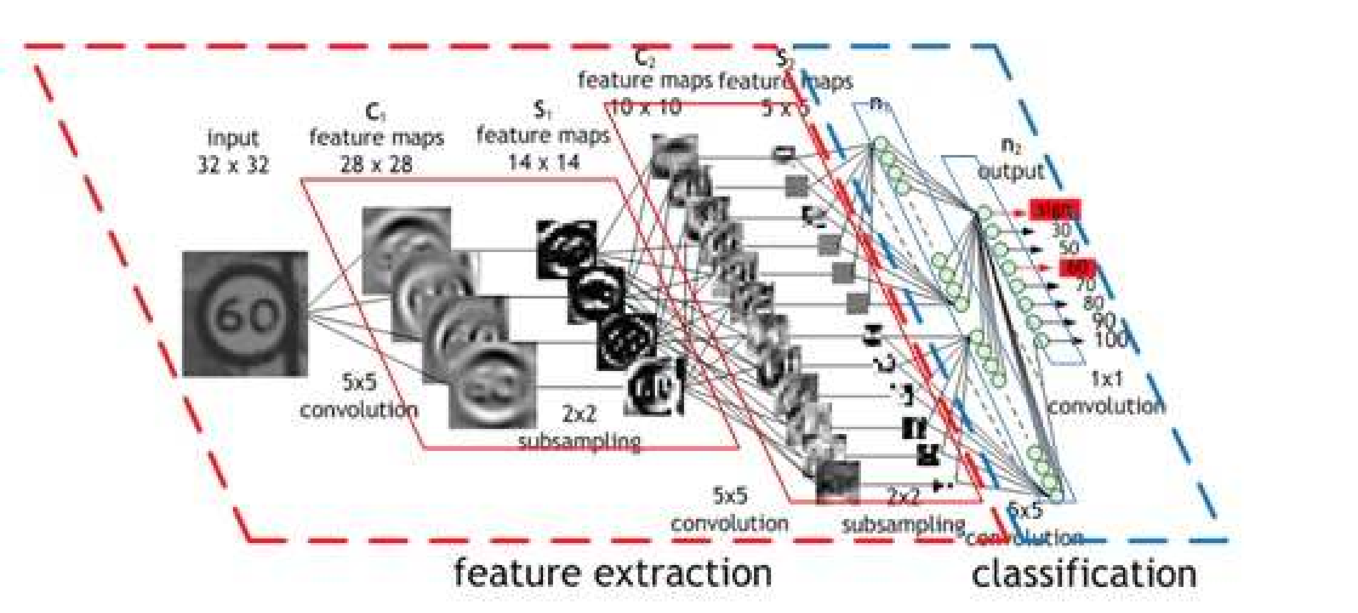
Stop training when validation accuracy stops increasing

Saving parameters at the end of each epoch is

Convolutional Neural Networks

Le Cun, 1998: handwritten digit recognition, 32x32 pixels

Feature extraction and downsampling allow smaller networks



https://devblogs.nvidia.com/parallelforall/deep-learning-nutshell-core-concepts/

Apache MXNet

Apache MXNet

Programmable

Simple syntax,
multiple languages

Portable

Highly efficient
models for mobile
and IoT

High Performance

Near linear scaling
across hundreds of GPUs

Most Open

Accepted into the
Apache Incubator

Best On AWS

Optimized for
deep learning on
AWS

Imperative Programming

```
import numpy as np
a = np.ones(10)
b = np.ones(10) * 2
c = b * a
d = c + 1
```

Easy to tweak in Python

PROS

- Straightforward and flexible.
- Take advantage of language native features (loop, condition, debugger).
- E.g. Numpy, Matlab, Torch, …

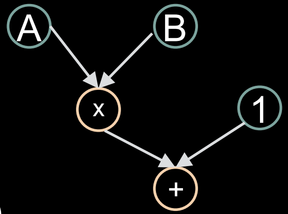
CONS

- Hard to optimize

Declarative Programming

```
A = Variable('A')
B = Variable('B')
C = B * A
D = C + 1
f = compile(D)
d = f(A=np.ones(10),

B=np.ones(10)*2)
```



C can share memory with D because C is deleted later

PROS

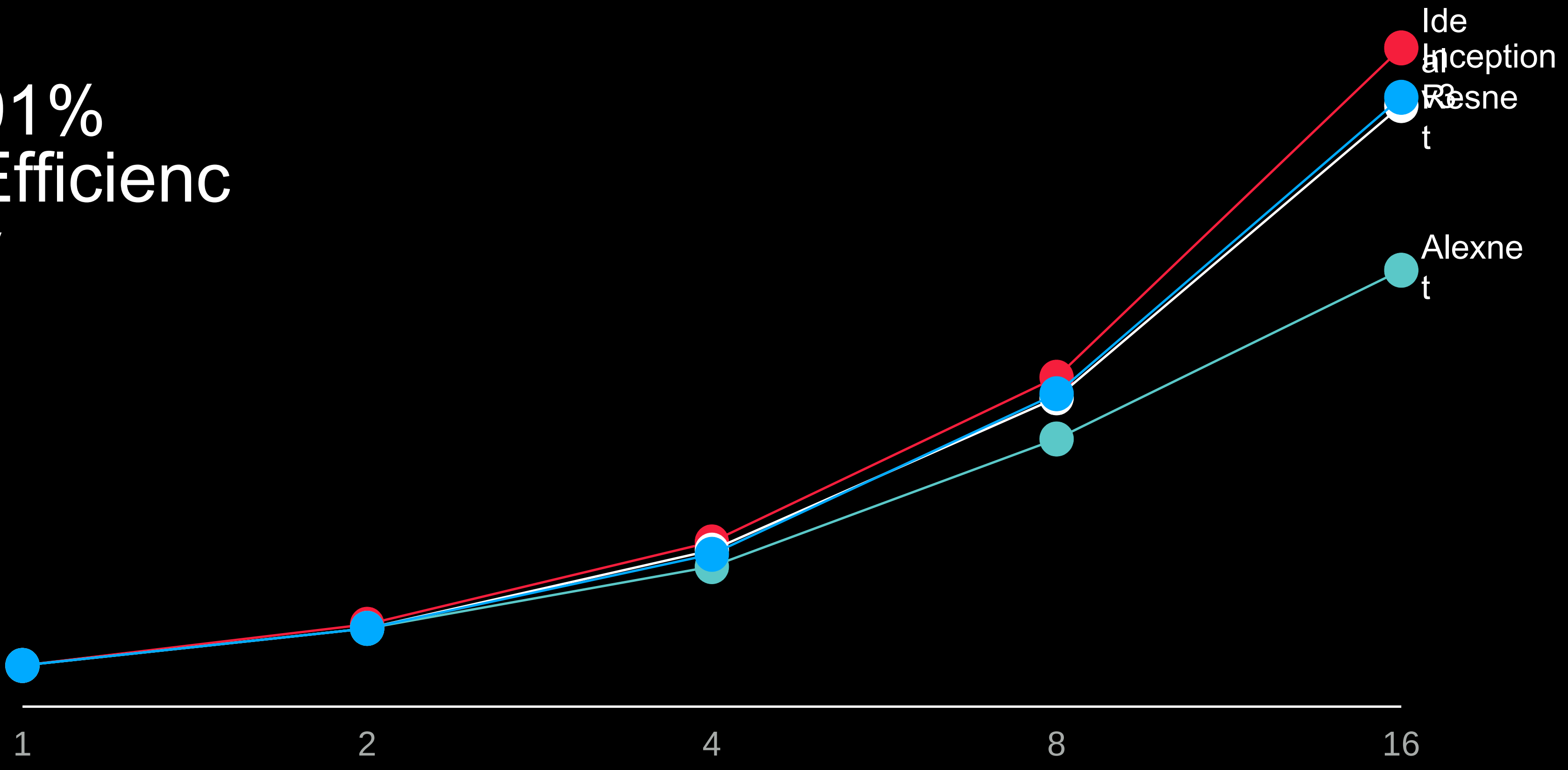
- More chances for optimization
- Cross different languages
- E.g. TensorFlow, Theano, Caffe

CONS

- Less flexible

Multi-GPU Scaling With MXNet

91%
Efficienc
y



`mx.model.FeedForward`

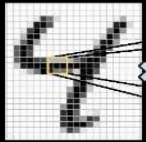
`model.fit`

Input

Output

`mx.sym.SoftmaxOutput`

`mx.sym.FullyConnected(data, num_hidden=128)`



`mx.sym.Convolution(data, kernel=(5,5), num_filter=20)`

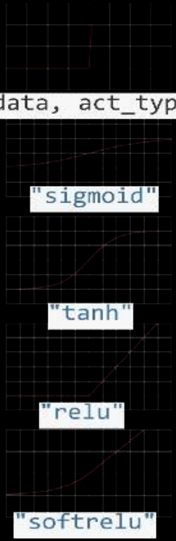
`mx.sym.Pooling(data, pool_type="max", kernel=(2,2),`
`stride=(2,2)`

`lstm.lstm_unroll(num_lstm_layer, seq_len, len, num_hidden, num_embed)`

$$\cos(w, \textbf{queen}) = \cos(w, \textbf{king}) - \cos(w, \textbf{man}) + \cos(w, \textbf{woman})$$

`mx.symbol.Embedding(data, input_dim, output_dim = k)`

`mx.sym.Activation(data, act_type="xxxx")`



Let's code

1. Our first network
2. Using pre-trained models
3. Learning from scratch (MNIST, MLP, CNN)
4. Learning and fine-tuning (CIFAR-10, Resnext-101)
5. Using Apache MXNet as a Keras backend
6. Fine-tuning with Keras/MXNet (CIFAR-10, Resnet-50)
7. A quick word about Sockeye

http://mxnet.io
http://medium.com/@julsimon
https://github.com/juliensimon/aws/tree/master/mxnet

Thank you!

http://aws.amazon.com/evangelists/julien-simon

@julsimon