Neural Networks III

CMSC 422

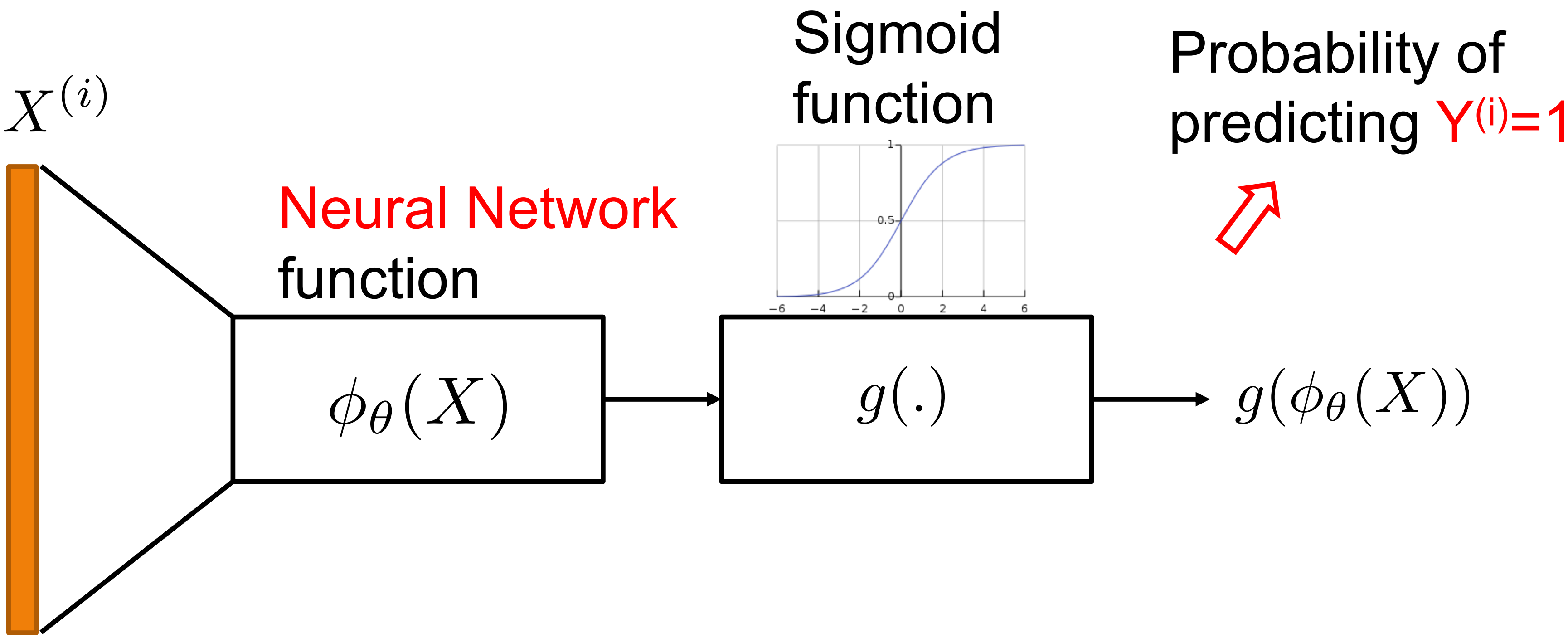
SOHEIL FEIZI

sfeizi@cs.umd.edu

Today’s topics

- Back Propagation: SGD+ Chain rule
- Multi-label classification
- SGD with momentum
- Improved NN architectures

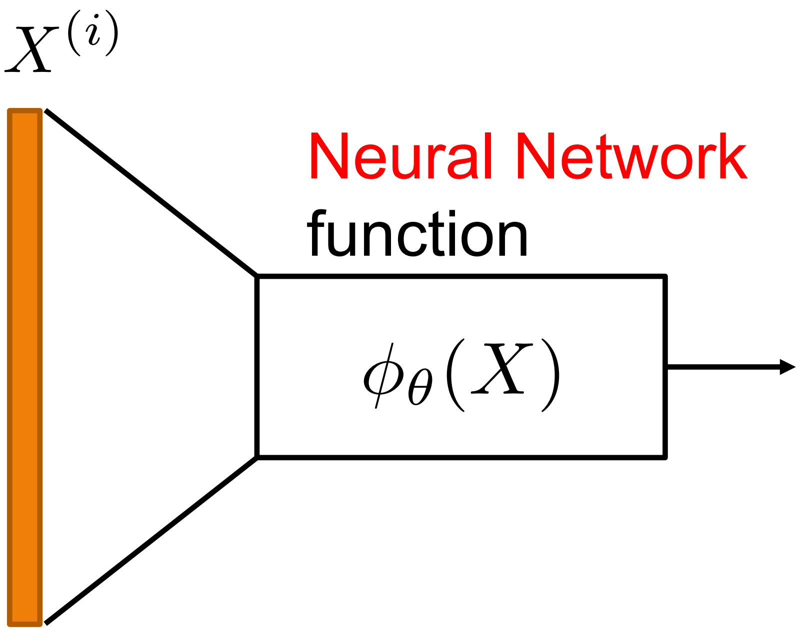
Classification using Neural Network



Compute model parameters using cross-entropy loss opt:

$$\max_{\theta} \quad \sum_{i=1}^{N} Y^{(i)} \log g(\phi_\theta(X^{(i)})) + (1 - Y^{(i)}) \log(1 - g(\phi_\theta(X^{(i)})))$$

Nonlinear Regression using Neural Network



$$\hat{y}^{(i)} = \phi_\theta(X^{(i)})$$

Compute model parameters using quadratic loss opt:

$$\min_\theta \sum_{i=1}^{N} (y^{(i)} - \phi_\theta(X^{(i)}))^2$$

Loss functions

Classification
Problem

- ❖ Hinge loss
- ❖ Cross entropy loss

Regression
Problem

- ❖ Quadratic loss

How to find optimal model parameters?

Stochastic Gradient Descent

What do we need to be able to use SGD in deep learning?

Computation of the gradient of the loss function with respect to model parameters

Training a Neural Network

The Backpropagation Algorithm

=

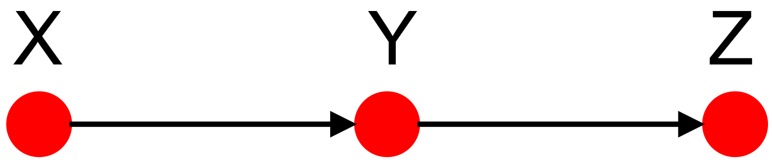
Gradient descent + Chain rule

Review of Chain Rule

X,Y,Z in R
Y=g(X)
Z=f(Y)

$$\frac{dZ}{dX} = \frac{dZ}{dY}\frac{dY}{dX}$$

Graph representation (computational graph):



This is NOT a neural network

Review of Chain Rule

$X \in \mathbb{R}^n, Y \in \mathbb{R}^m, Z \in \mathbb{R}$

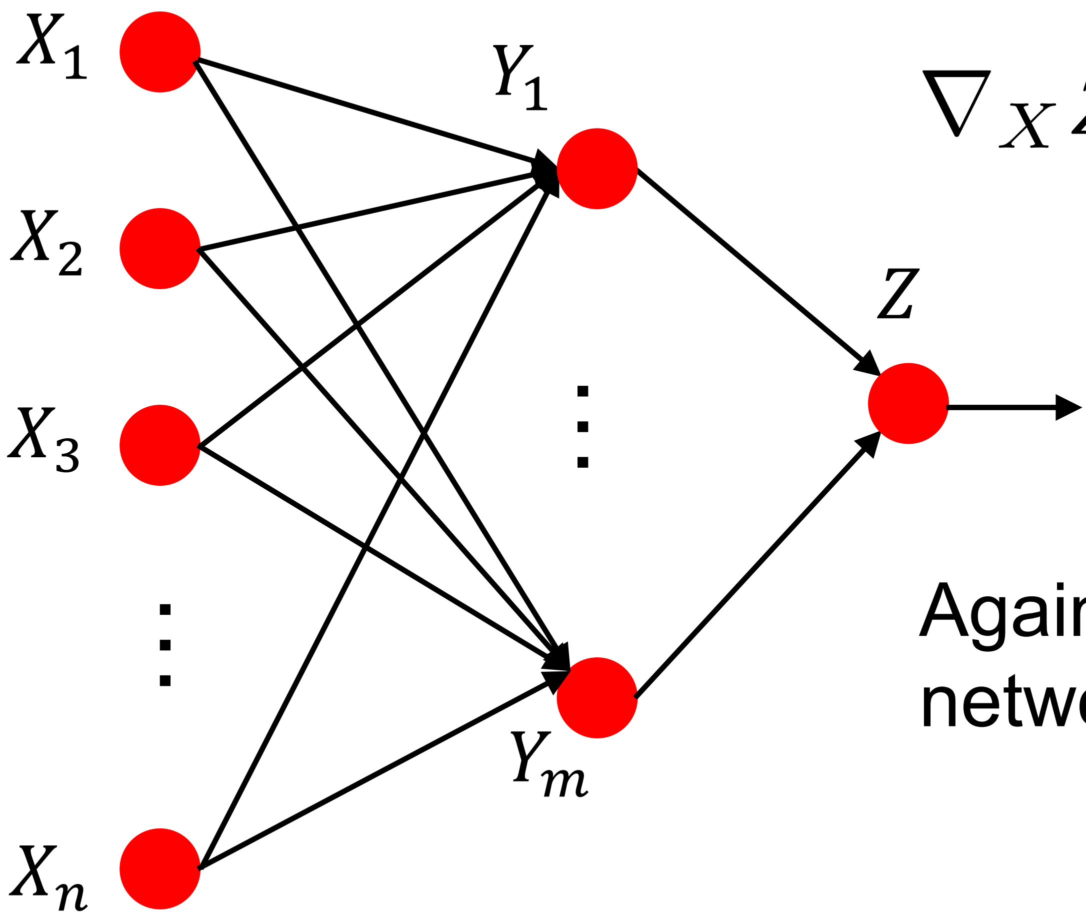
Y=g(X), Z=f(Y)

$$\nabla_X Z = \underbrace{(\frac{dY}{dX})}_{\text{Jacobian matrix}} \nabla_Y Z = \sum_{j=1}^{m} (\nabla_X Y_j) \frac{dZ}{dY_j}$$

Practice: $Y_1=X_1+X_2$^2, $Y_2=X_2+X_3$^2, $Z=Y_1^2+Y_2$^2

Compute $\nabla_X Z$

Graph Representation



$$\nabla_X Z = \sum_{j=1}^{m} (\nabla_X Y_j) \frac{dZ}{dY_j}$$

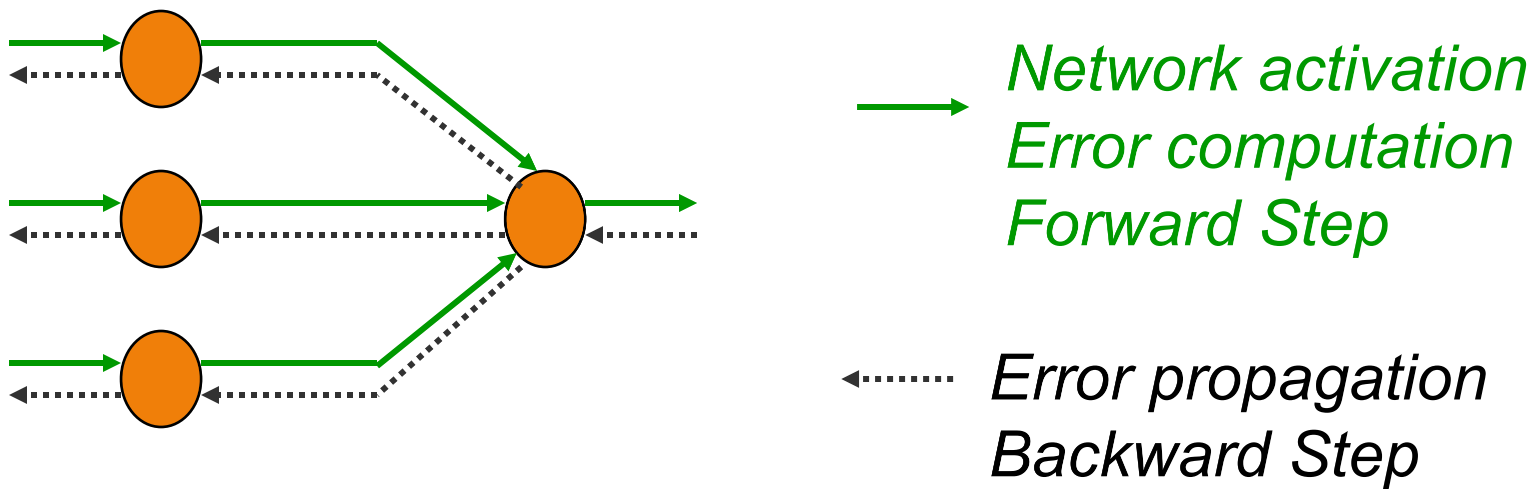
Again, this is NOT a neural network!

Training: Backpropagation Algorithm

- Searches for weight values that **minimize the total error of the network** over the set of training examples.
- **Repeated** procedures of the following two passes:
  - **Forward pass**: Compute the **outputs** of all units in the network, and the **error** of the output layers.
  - **Backward pass**: The network error is used for updating the weights.
    - Starting at the output layer, **the error is propagated backwards through the network, layer by layer**. This is done by recursively computing the local gradient of each neuron.

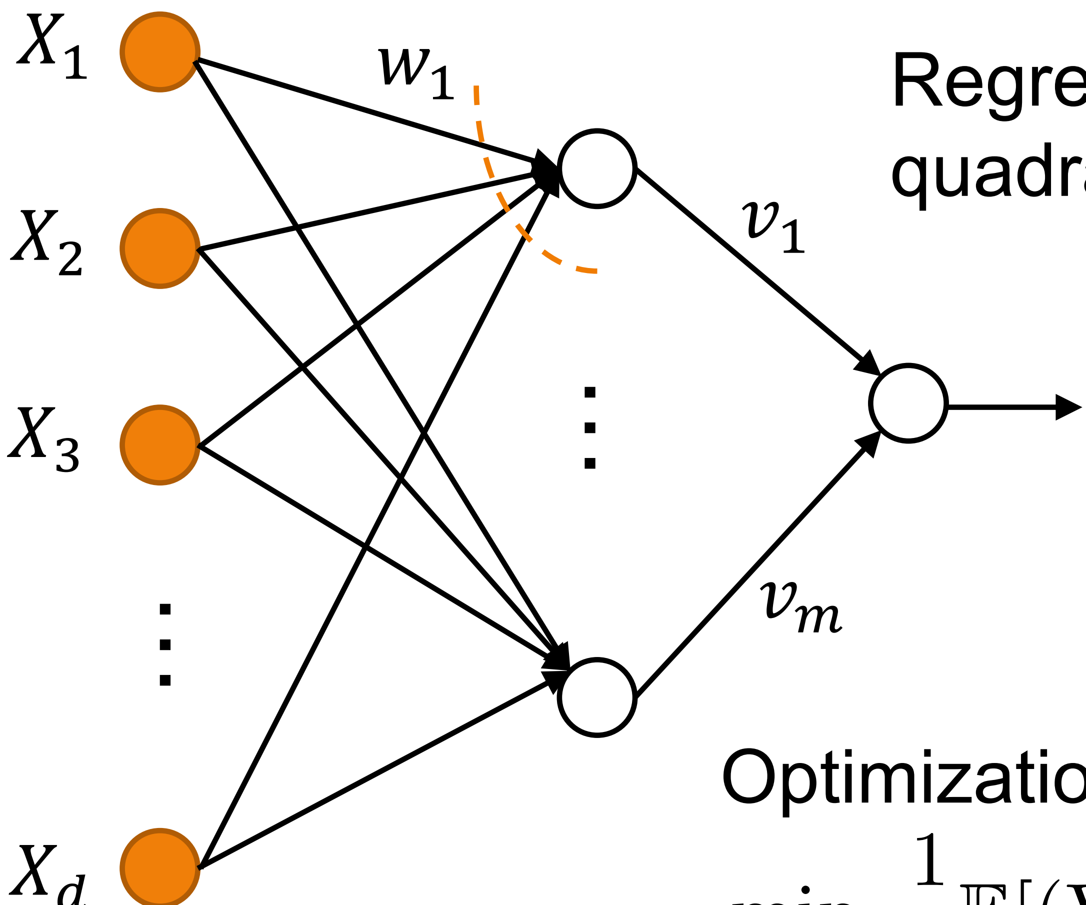
The Backpropogation Algorithm

Back-propagation training algorithm, illustrated:



Backprop adjusts the weights of the NN in order to minimize the network total mean squared error.

Example: two layer NN



Regression problem with quadratic loss function

Optimization:

$$\min_{W,v} \; \frac{1}{2}\mathbb{E}[(Y - \sum_{i=1}^{m} \overbrace{v_i\phi(w_i^t X)}^{\phi_i})^2]$$

Gradient of objective w.r.t. output layer weights v

$$\nabla_{v_i} L = \mathbb{E}\left[(Y - \sum_{i=1}^{m} v_i \phi(w_i^t X))\phi(w_i^t X)\right]$$

Error term:
Y$-\hat{Y}$

activation of hidden unit i

Gradient of objective w.r.t. hidden unit weights w_i

$$\nabla_L w_1 = \left(\frac{dL}{d\phi_1}\right)\nabla_{w_1}\phi_1$$

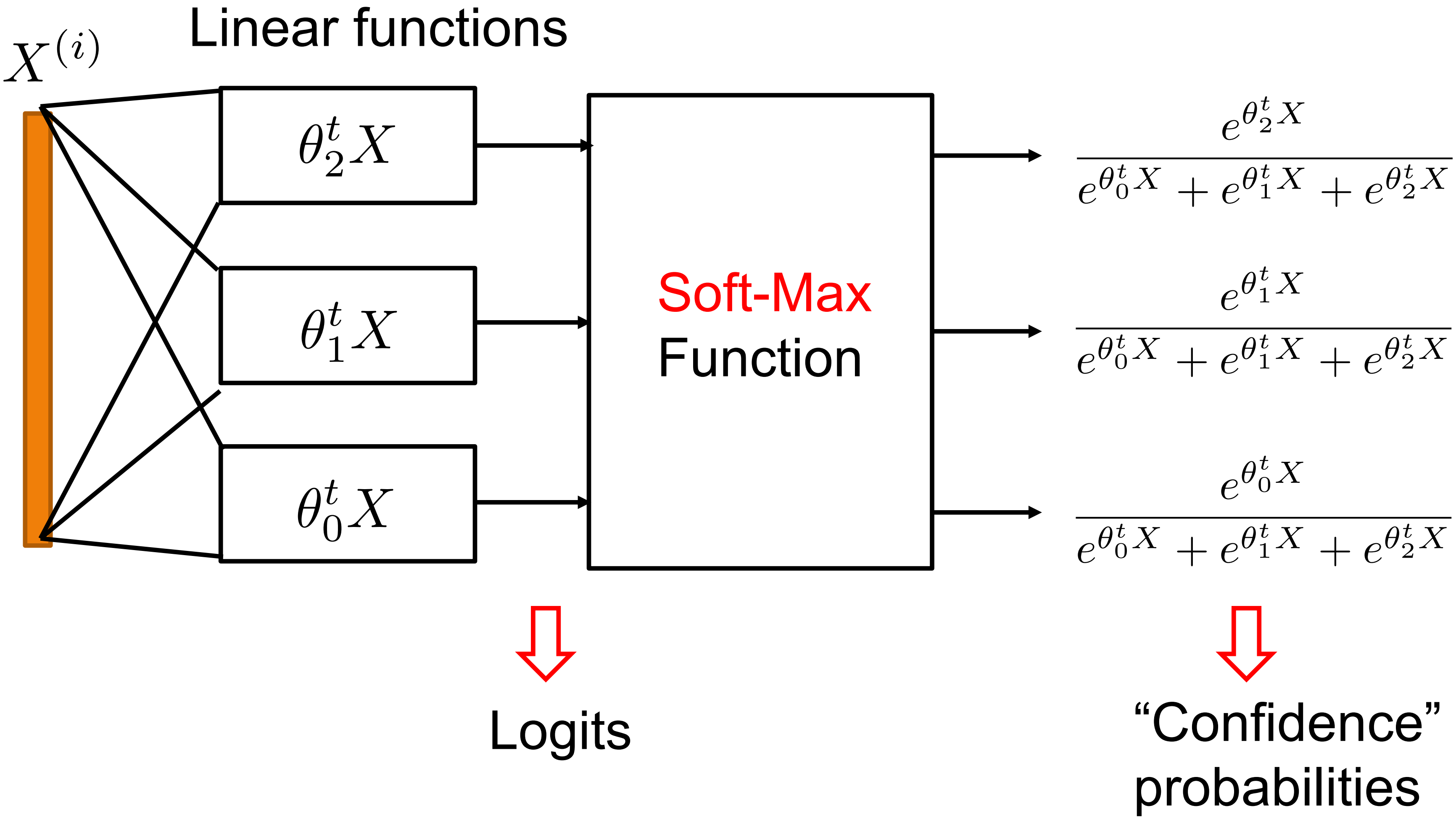
$$= -\mathbb{E}\left[(Y - \sum_{i=1}^{m} v_i \phi(w_i^t X))v_1\right]$$

$$= \mathbb{E}\left[\phi'(w_1^t X)X\right]$$

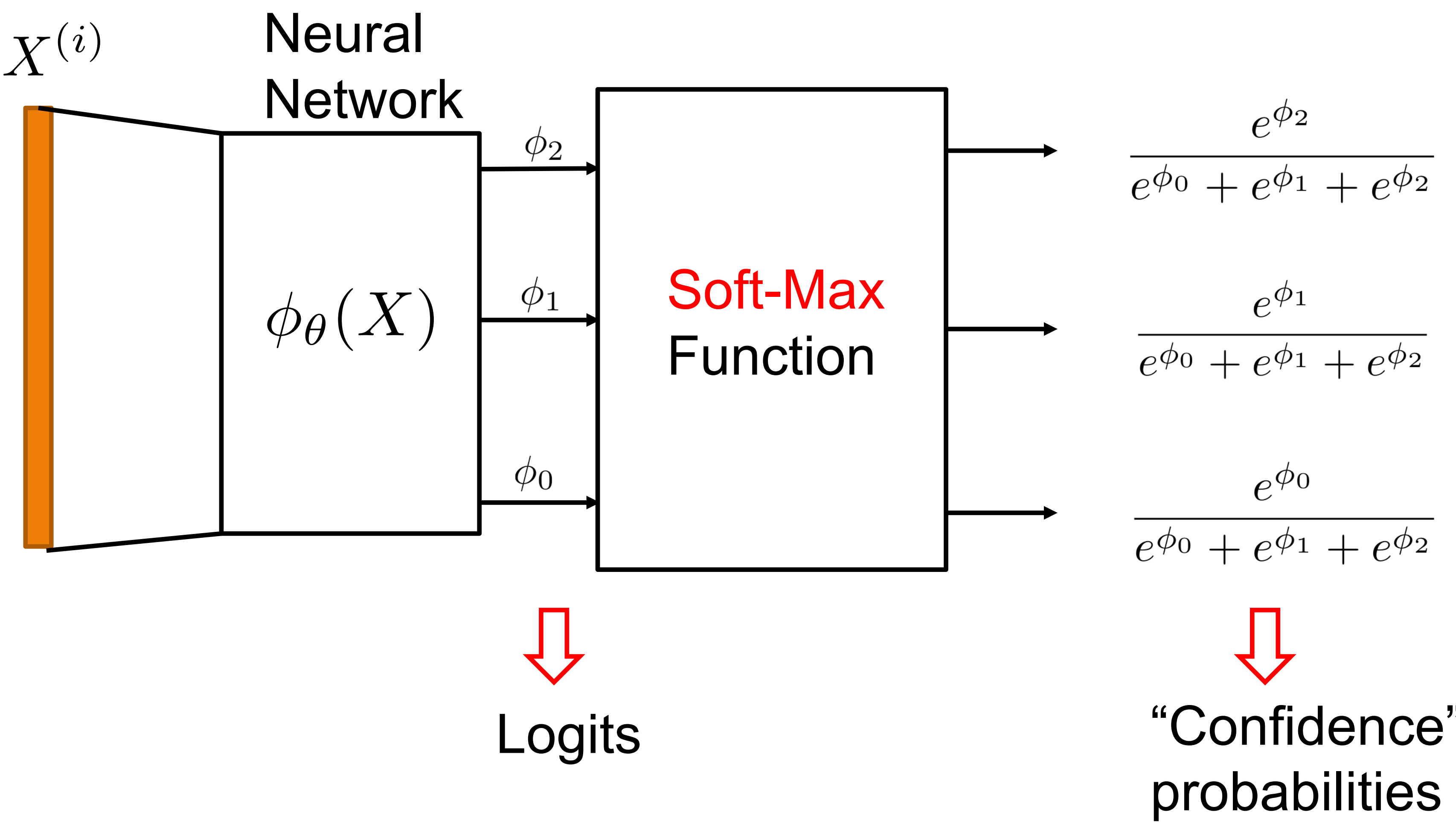
Multi-Label Classification

Q: how to extend our method for multi-label classification?

Recall: Multi-Label Classification using Logistic Regression

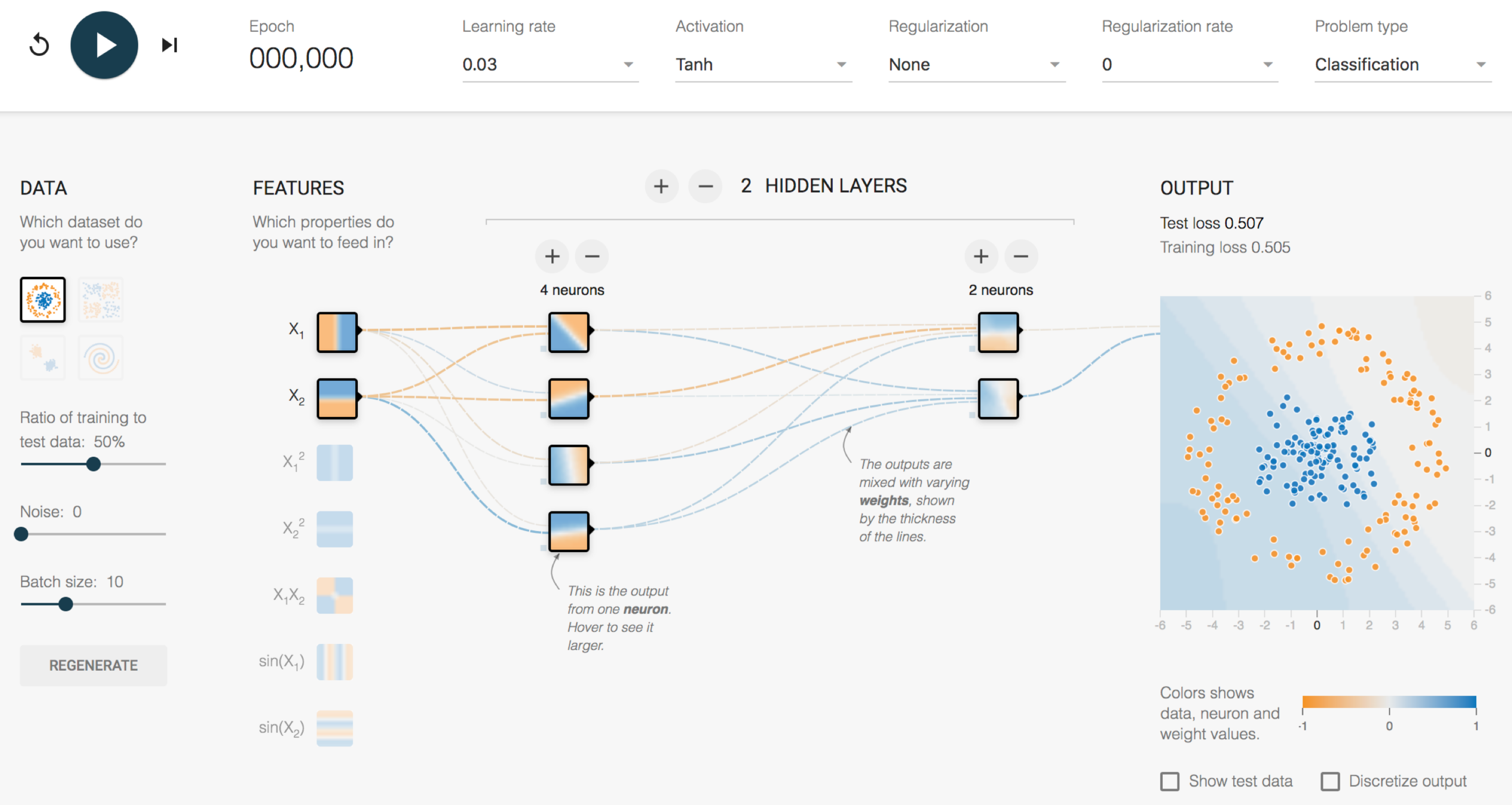


Multi-Label Classification Using NNs



Try different architectures and training parameters here:

http://playground.tensorflow.org



Epoch
000,000
Learning rate
0.03
Activation
Tanh
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
X_1
X_2
X_1^2
X_2^2
X_1X_2
$\sin(X_1)$
$\sin(X_2)$
2 HIDDEN LAYERS
4 neurons
2 neurons
This is the output from one **neuron**. Hover to see it larger.
The outputs are mixed with varying **weights**, shown by the thickness of the lines.
OUTPUT
Test loss 0.507
Training loss 0.505
Colors shows data, neuron and weight values.
-1
0
1
Show test data
Discretize output

Tricky issues with neural network training

- Sensitive to initialization
  - Objective is non-convex, many local optima
  - In practice: start with random values rather than zeros
- Many other hyper-parameters
  - Number of hidden units (and potentially hidden layers)
  - Gradient descent learning rate
  - Stopping criterion

Neural networks vs. linear classifiers

Advantages of Neural Networks:

– More expressive

– Less feature engineering

Challenges using Neural Networks:

– Harder to train

– Harder to interpret

Neural Network Architectures

- We focused on a **multi-layer feedforward** network
- Many other deeper architectures
  - Convolutional networks
  - Recurrent networks (LSTMs)
  - Dense Nets, ResNets, etc

Issues in Deep Neural Networks

- Long training time
  - There are sometimes a lot of training data
  - Many iterations (epochs) are typically required for optimization
  - Computing gradients in each iteration takes too much time

Slide credit: adapted from Bohyung Han

Improving on Gradient Descent: Stochastic Gradient Descent (SGD)

- Update weights for each example

$$E = \frac{1}{2}(y^n - \hat{y}^n)^2 \qquad \boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) - \epsilon \frac{\partial E^n}{\partial \boldsymbol{w}_i}$$

+ **Fast, online**
− **Sensitive to noise**

- Mini-batch SGD: Update weights for a small set of examples

$$E = \frac{1}{2}\sum_{n \in B}(y^n - \hat{y}^n)^2 \qquad \boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) - \epsilon \frac{\partial E^B}{\partial \boldsymbol{w}_i}$$

+ **Fast, online**
+ **Robust to noise**

Slide credit: Bohyung Han

Improving on Gradient Descent: SGD with Momentum

- Update based on gradients + previous direction

$$v_i(t) = \alpha v_i(t-1) - (1-\alpha)\frac{\partial E}{\partial w_i}(t)$$

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \boldsymbol{v}(t)$$

+ Converge faster
+ Avoid oscillation

Slide credit: Bohyung Han

Improving on Gradient Descent: SGD with Momentum

SGD w/o momentum

SGD with momentum helps dampen oscillations

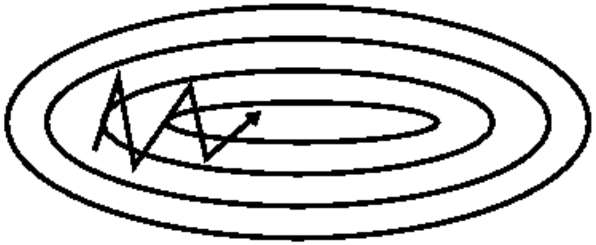


Image: http://ruder.io/optimizing-gradient-descent/index.html#momentum

Improving the Training Objective: Regularization/Weight Decay

- Penalize the size of the weights

$$C = E + \frac{\lambda}{2}\sum_i w_i^2$$

$$w_i(t+1) = w_i(t) - \epsilon\frac{\partial C}{\partial w_i} = w_i(t) - \epsilon\frac{\partial E}{\partial w_i} - \lambda w_i$$

→ **Improves generalization**

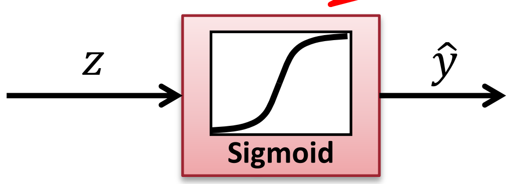
Slide credit: Bohyung Han

Vanishing Gradient Problem

In deep networks

- Gradients in the lower layers are typically extremely small
- Optimizing multi-layer neural networks takes huge amount of time

Derivative of sigmoid in [0,1]



$$\frac{\partial E}{\partial w_{ki}} = \sum_n \frac{\partial z_i^n}{\partial w_{ki}} \frac{d\hat{y}_i^n}{dz_i^n} \frac{\partial E}{\partial \hat{y}_i^n} = \sum_n \frac{\partial z_i^n}{\partial w_{ki}} \boxed{\frac{d\hat{y}_i^n}{dz_i^n}} \sum_j w_{ij} \boxed{\frac{d\hat{y}_j^n}{dz_j^n}} \frac{\partial E}{\partial \hat{y}_j^n}$$

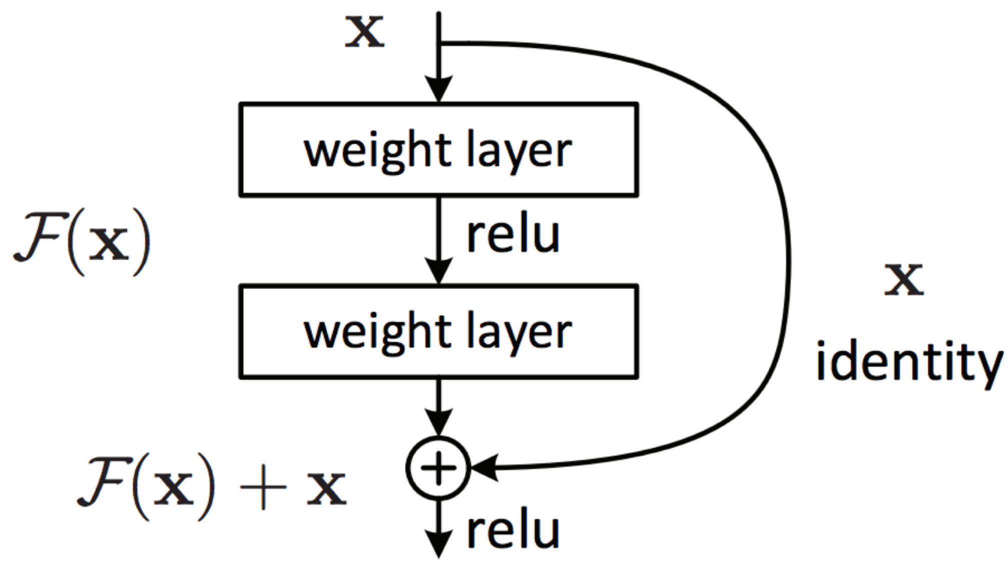
Slide credit: adapted from Bohyung Han

Vanishing Gradient Problem

- Vanishing gradient problem can be mitigated
  - Using custom neural network architectures

  - Using other non-linearities
    - E.g., Rectifier: $f(x) = \max(0,x)$

ResNet



Deep Residual Learning for Image Recognition

https://arxiv.org › cs

by K He - 2015 - Cited by 19999 - Related articles

Dec 10, 2015 - Abstract: Deeper neural networks are more difficult to train. We present a residual learning framework to ease the training of networks that are ...

Why Neural Networks?

Perceptron

- Proposed by Frank Rosenblatt in 1957
- Real inputs/outputs, threshold activation function

Revival in the 1980's

Backpropagation discovered in 1970's but popularized in 1986

- David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams. "Learning representations by back-propagating errors." In Nature, 1986.

MLP is a universal approximator

- Can approximate any non-linear function in theory, given enough neurons, data
- Kurt Hornik, Maxwell Stinchcombe, Halbert White. "Multilayer feedforward networks are universal approximators." Neural Networks, 1989

Generated lots of excitement and applications

http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

Neural Networks Applied to Vision

LeNet – vision application

- LeCun, Y; Boser, B; Denker, J; Henderson, D; Howard, R; Hubbard, W; Jackel, L, “Backpropagation Applied to Handwritten Zip Code Recognition,” in Neural Computation, 1989
- USPS digit recognition, later check reading

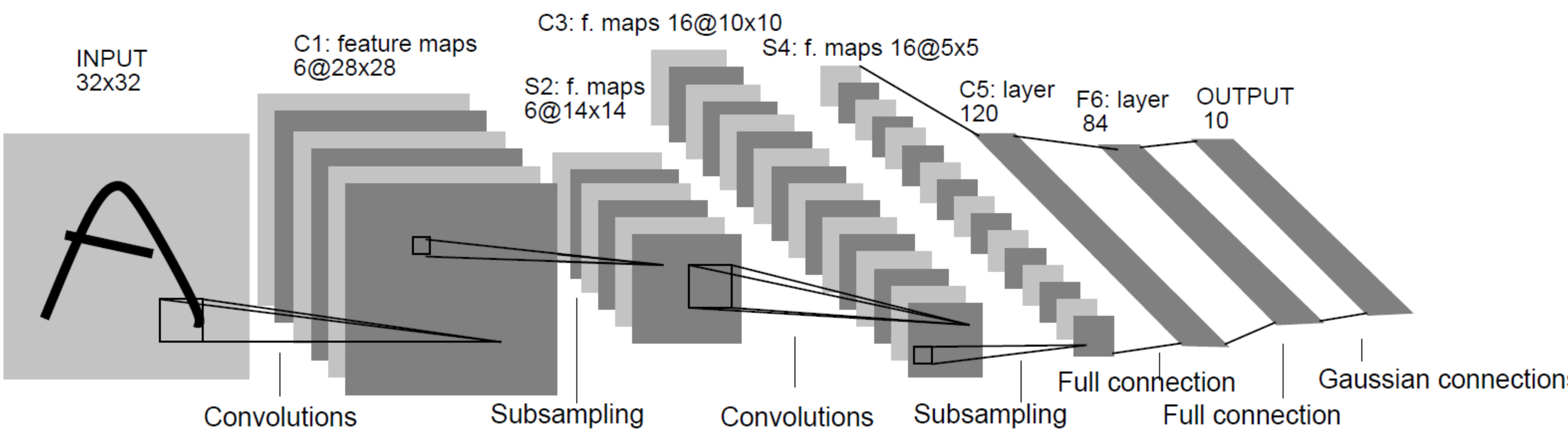


Image credit: LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. “Gradient-based learning applied to document recognition.” Proceedings of the IEEE, 1998.

New “winter” and revival in early 2000’s

New “winter” in the early 2000’s due to

- problems with training NNs
- Support Vector Machines (SVMs), Random Forests (RF) – easy to train, nice theory

Revival again by 2011-2012

- Name change (“neural networks” -> “deep learning”)
- + Algorithmic developments that made training somewhat easier
- + Big data + GPU computing
- = performance gains on many tasks (esp Computer Vision)

http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning-part-4/

Big Data

- ImageNet Large Scale Visual Recognition Challenge
  - 1000 categories w/ 1000 images per category
  - 1.2 million training images, 50,000 validation, 150,000 testing

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg and L. Fei-Fei. **ImageNet Large Scale Visual Recognition Challenge**. *IJCV,* 2015.

AlexNet Architecture

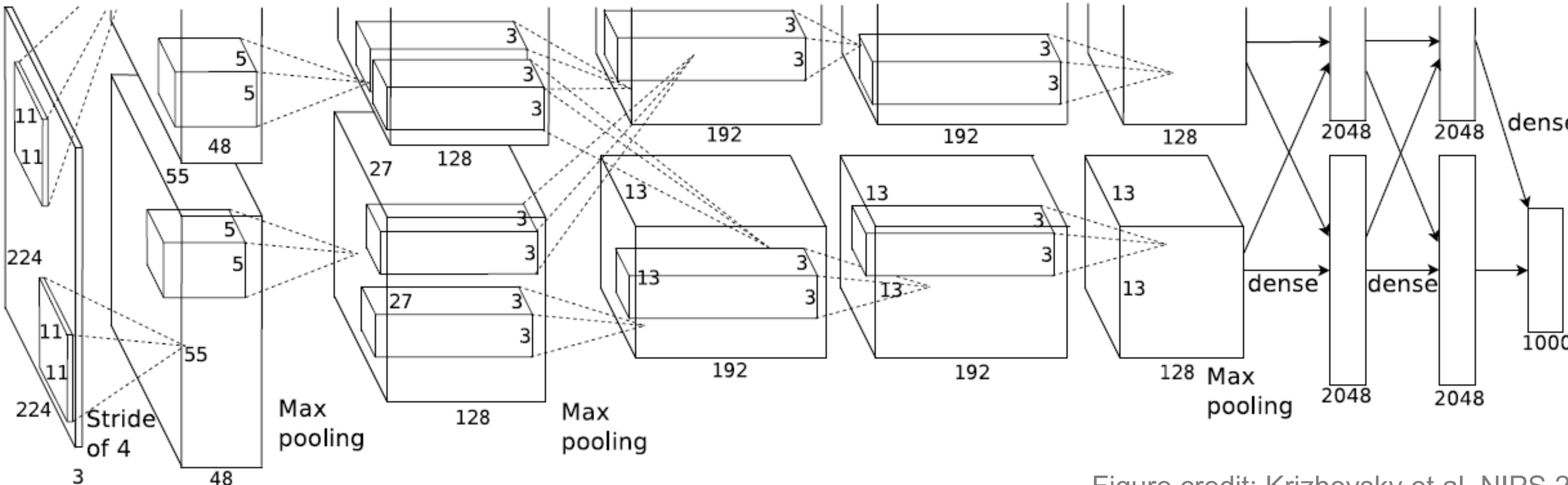


Figure credit: Krizhevsky et al, NIPS 2012.

60 million parameters!

Various tricks

- ReLU nonlinearity
- Overlapping pooling
- Local response normalization
- Dropout – set hidden neuron output to 0 with probability .5
- Data augmentation
- Training on GPUs

Alex Krizhevsky, Ilya Sutskeyer, Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. NIPS, 2012.

GPU Computing

- Big data and big models require lots of computational power
- GPUs
  - thousands of cores for parallel operations
  - multiple GPUs
  - still took about 5-6 days to train AlexNet on two NVIDIA GTX 580 3GB GPUs (much faster today)

Image Classification Performance

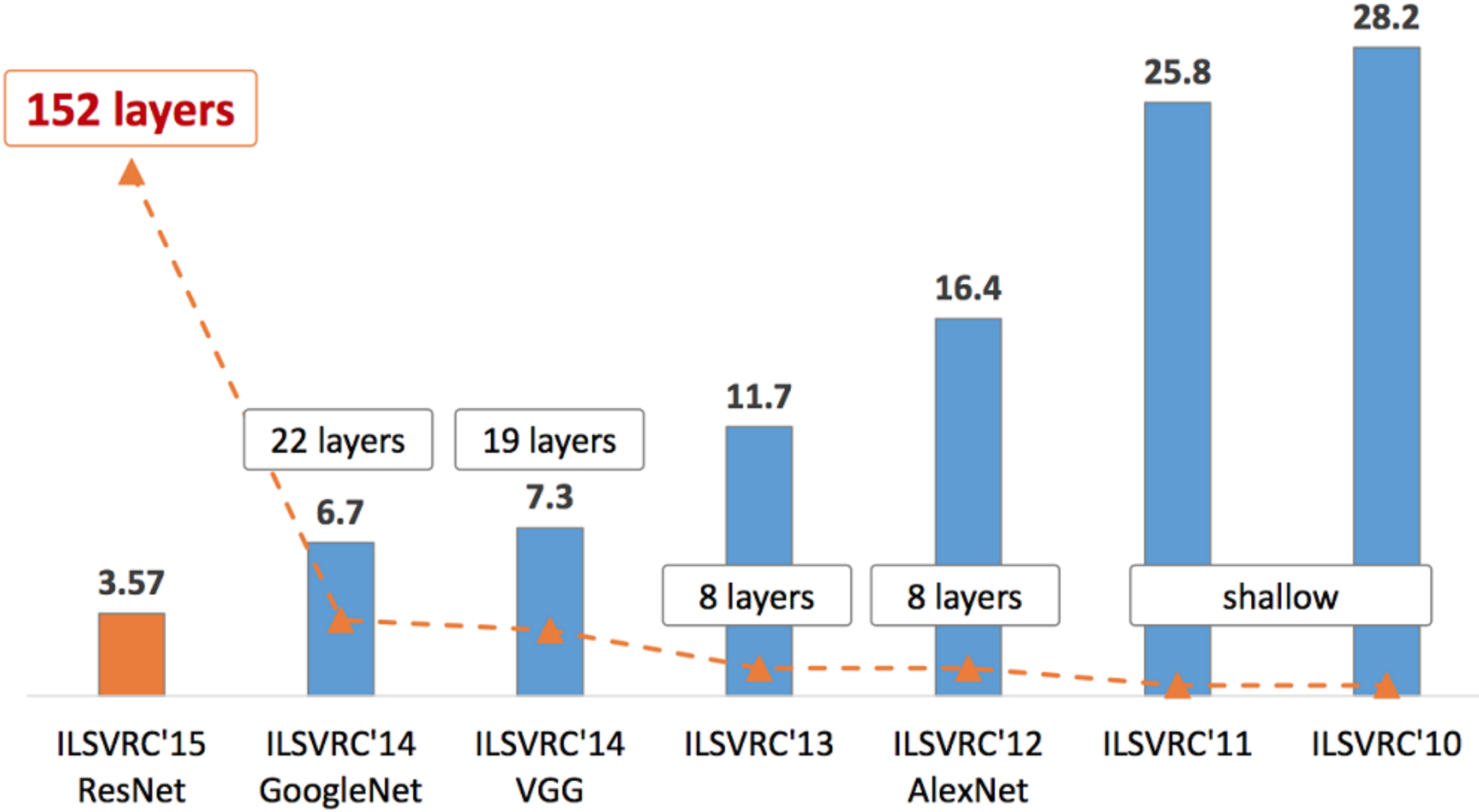
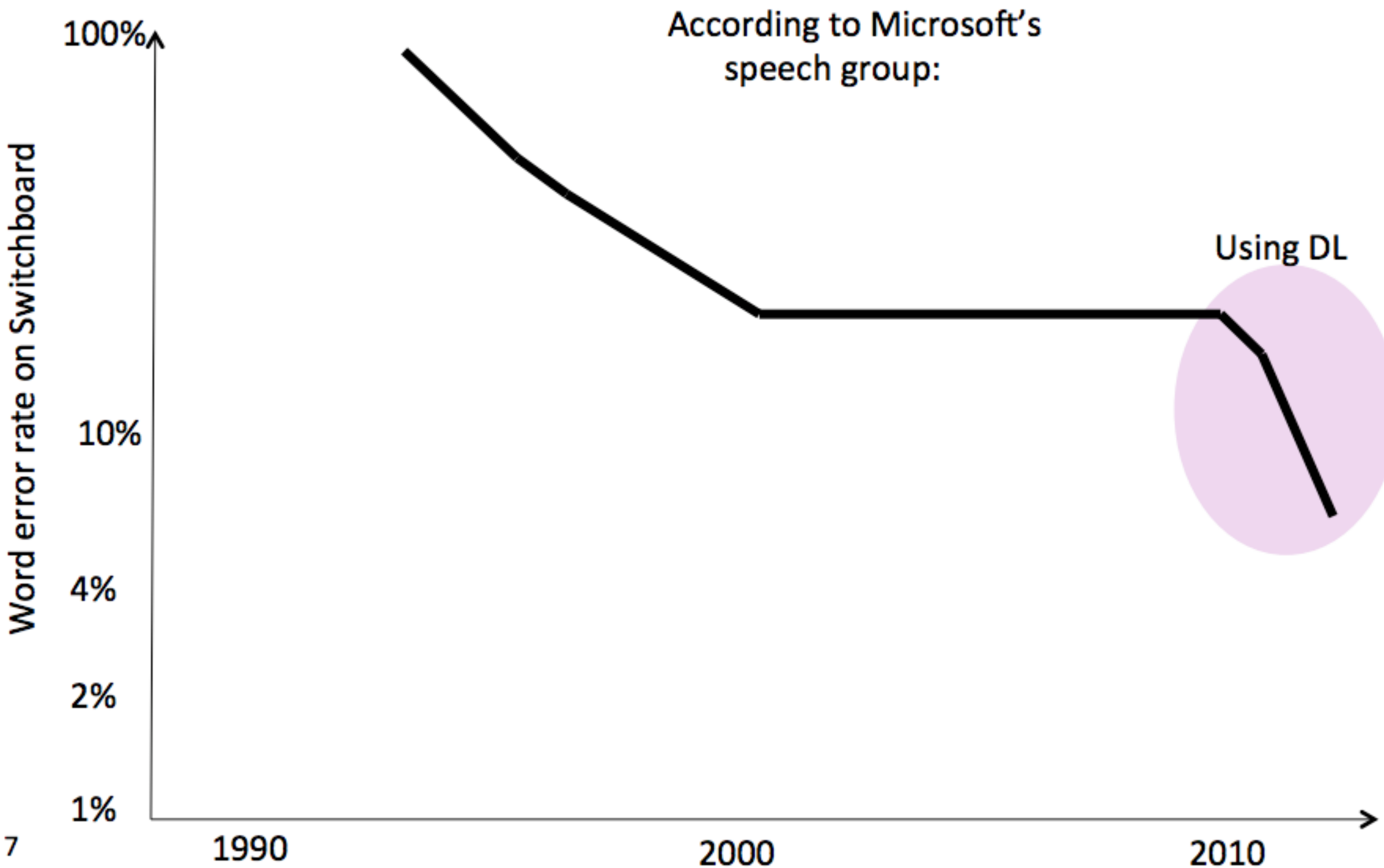


Image Classification Top-5 Errors (%)

Figure from: K. He, X. Zhang, S. Ren, J. Sun. “Deep Residual Learning for Image Recognition”. arXiv 2015. (slides)

Speech Recognition



Slide credit: Bohyung Han

Recurrent Neural Networks for Language Modeling

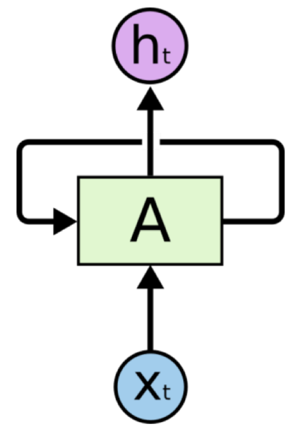
- Speech recognition is difficult due to ambiguity
  - “how to recognize speech”
  - or “how to wreck a nice beach“?

- Language model gives probability of next word given history
  - P(“speech”|”how to recognize”)?

Recurrent Neural Networks

Networks with loops

- The output of a layer is used as input for the same (or lower) layer
- Can model dynamics (e.g. in space or time)



Loops are unrolled

- Now a standard feed-forward network with many layers
- Suffers from vanishing gradient problem
- In theory, can learn long term memory, in practice not (Bengio et al, 1994)

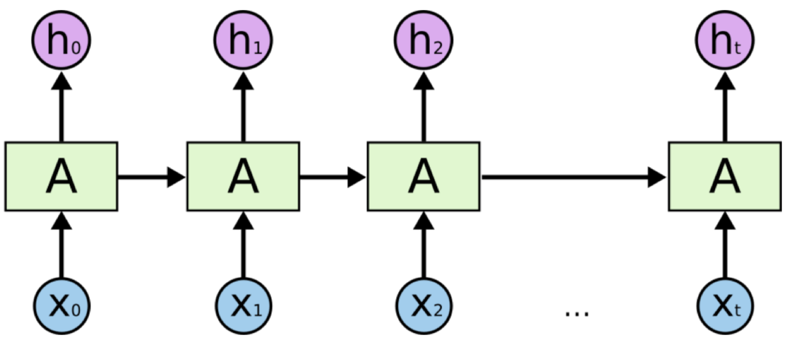


Image credit: Chritopher Olah's blog http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Sepp Hochreiter (1991), Untersuchungen zu dynamischen neuronalen Netzen, Diploma thesis. Institut f. Informatik, Technische Univ. Munich. Advisor: J. Schmidhuber.

Y. Bengio, P. Simard, P. Frasconi. Learning Long-Term Dependencies with Gradient Descent is Difficult. In TNN 1994.

Long Short Term Memory (LSTM)

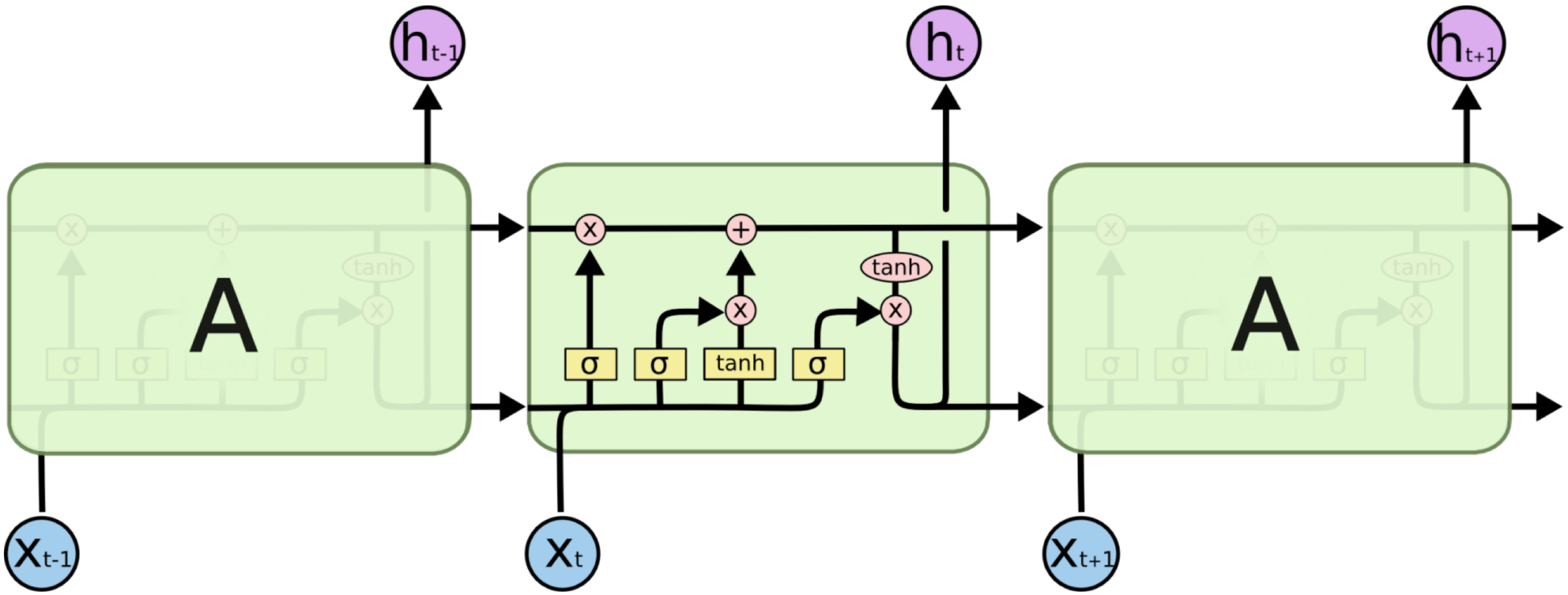


Image credit: Christopher Colah's blog, http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- A type of RNN explicitly designed not to have the vanishing or exploding gradient problem
- Models long-term dependencies
- Memory is propagated and accessed by gates
- Used for speech recognition, language modeling …

Hochreiter, Sepp; and Schmidhuber, Jürgen. "Long Short-Term Memory." Neural Computation, 1997.

Long Short Term Memory (LSTM)

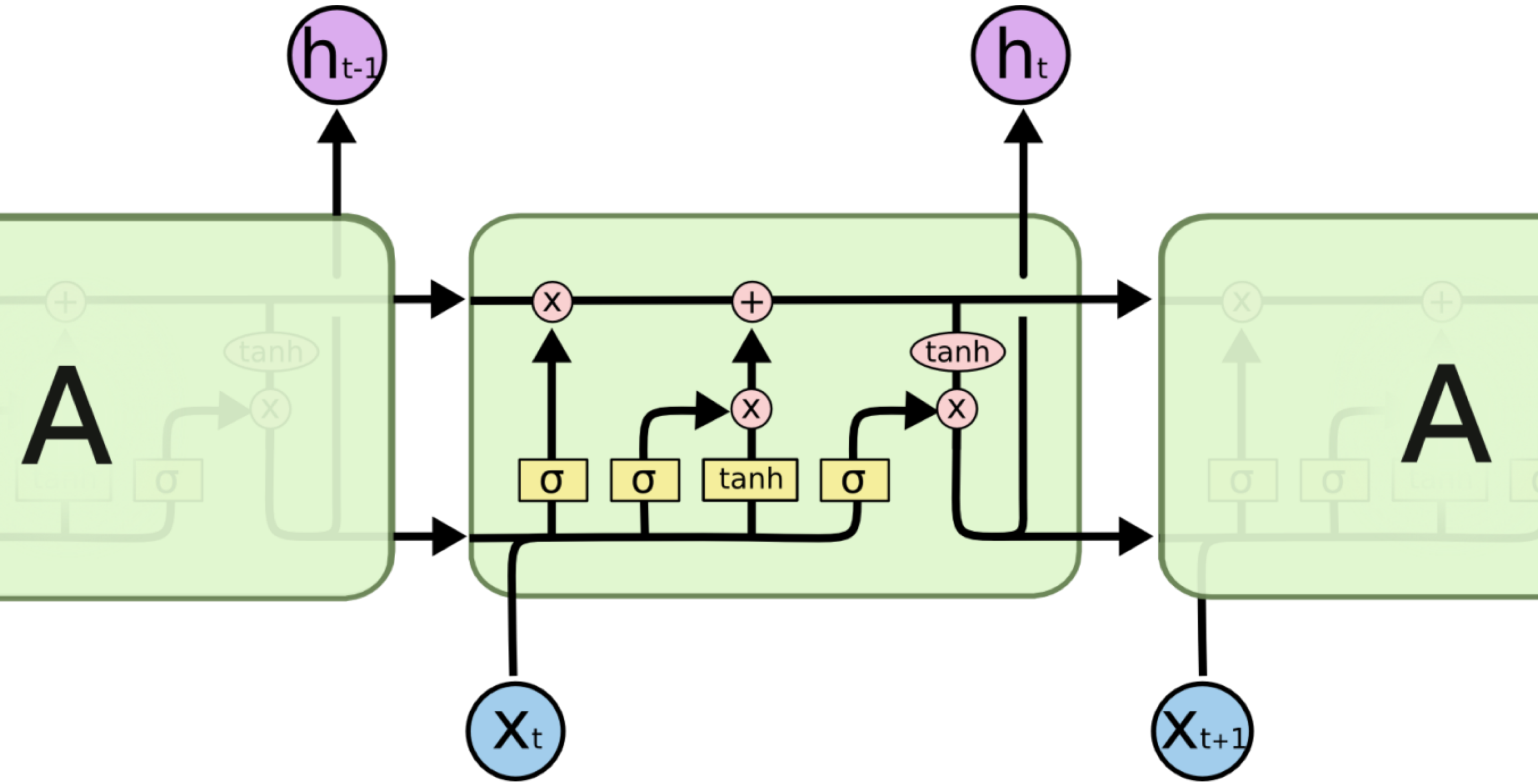


Image credit: Christopher Colah's blog, http://colah.github.io/posts/2015-08-Understanding-LSTMs/

What you should know about deep neural networks

- Why they are difficult to train
  - Initialization
  - Overfitting
  - Vanishing gradient
  - Require large number of training examples

- What can be done about it
  - Improvements to gradient descent
  - Stochastic gradient descent
  - Momentum
  - Weight decay
  - Alternate non-linearities and new architectures

References (& great tutorials) if you want to explore further:
http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning-part-1/
http://cs231n.github.io/neural-networks-1/
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Keeping things in perspective…

In 1958, the New York Times reported the perceptron to be "the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."