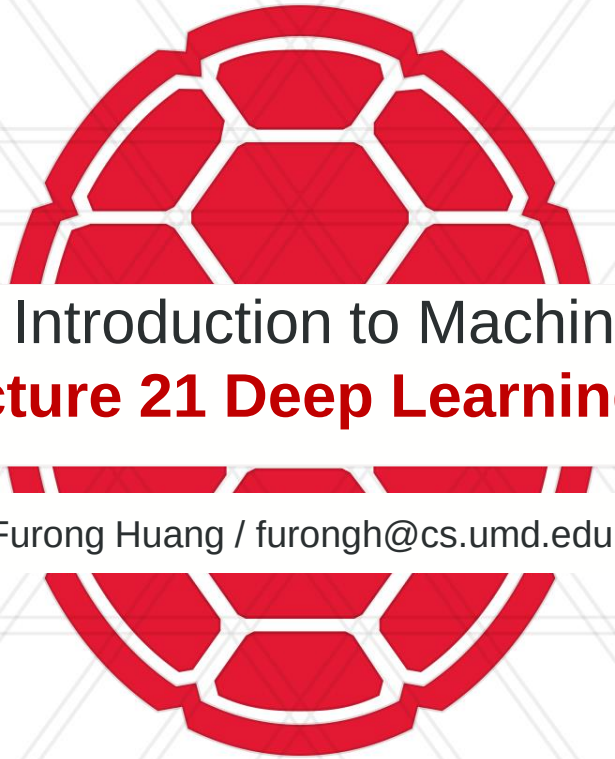


CMSC 422 Introduction to Machine Learning

Lecture 21 Deep Learning II

Furong Huang / furongh@cs.umd.edu

UNIVERSITY OF MARYLAND

Logistic Regression

- Goal: model the probability of a random variable Y being 0 or 1 given experimental data.
- Consider a generalized linear model function parameterized by θ,

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^\top x}}$$

- Attempt to model the probability that y is 0 or 1 with function

$$\Pr(y|x;\theta) = h_\theta(x)^y \left(1 - h_\theta(x)\right)^{1-y}$$

Logistic Regression

- The likelihood assuming all the samples are independent

$$L(\theta|x)$$

$$= \Pr(Y|X;\theta) = \prod_i \Pr(y_i|x_i;\theta) = \prod_i h_\theta(x_i)^{y_i}\left(1 - h_\theta(x_i)\right)^{1-y_i}$$

- Maximum likelihood

$$\max_\theta L(\theta|x) \equiv \max_\theta \prod_i h_\theta(x_i)^{y_i}\left(1 - h_\theta(x_i)\right)^{1-y_i}$$

Neural Network with Softmax Classifier

- Softmax Classifier: multinomial Logistic Regression, the number of classes more than 2.

- Score: Instead of linear function as the exponent, we use a nonlinear function (e.g., a neural network $s = f(x_i; W)$)

Softmax Classifier (Multinomial Logistic Regression)

Softmax Classifier (Multinomial Logistic Regression)

cat	**3.2**
car	5.1
frog	-1.7

Slide Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Softmax Classifier (Multinomial Logistic Regression)

Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

$$s = f(x_i; W)$$

cat	**3.2**
car	5.1
frog	-1.7

Slide Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Softmax Classifier (Multinomial Logistic Regression)

Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{where} \quad s = f(x_i; W)$$

cat	**3.2**
car	5.1
frog	-1.7

Slide Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Softmax Classifier (Multinomial Logistic Regression)

Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

where $s = f(x_i; W)$

Softmax function

cat	**3.2**
car	5.1
frog	-1.7

Slide Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Softmax Classifier (Multinomial Logistic Regression)

Softmax Classifier (Multinomial Logistic Regression)

cat **3.2**

car 5.1

frog -1.7

scores = unnormalized log probabilities of the classes.

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{where} \quad s = f(x_i; W)$$

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

$$L_i = -\log P(Y = y_i|X = x_i)$$

Slide Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Softmax Classifier (Multinomial Logistic Regression)

Softmax Classifier (Multinomial Logistic Regression)

cat	**3.2**
car	5.1
frog	-1.7

scores = unnormalized log probabilities of the classes.

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$ where $$s = f(x_i; W)$$

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

$$L_i = -\log P(Y = y_i|X = x_i)$$

---

in summary: $$L_i = -\log(\frac{e^{sy_i}}{\sum_j e^{s_j}})$$

Slide Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Softmax Classifier (Multinomial Logistic Regression)

Softmax Classifier (Multinomial Logistic Regression)

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

cat	**3.2**
car	5.1
frog	-1.7

unnormalized log probabilities

Slide Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Softmax Classifier (Multinomial Logistic Regression)

Softmax Classifier (Multinomial Logistic Regression)

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

cat	**3.2**	exp →	**24.5**
car	5.1		164.0
frog	-1.7		0.18

unnormalized log probabilities

Slide Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Softmax Classifier (Multinomial Logistic Regression)

Softmax Classifier (Multinomial Logistic Regression)

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

cat	**3.2**	exp	**24.5**	normalize	**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

unnormalized log probabilities — probabilities

Slide Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Softmax Classifier (Multinomial Logistic Regression)

Softmax Classifier (Multinomial Logistic Regression)

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

cat	**3.2**	**24.5**	**0.13**	→ L_i = -log(0.13) = **0.89**
car	5.1	164.0	0.87	
frog	-1.7	0.18	0.00	
	unnormalized log probabilities	exp →	normalize →	probabilities

Slide Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Softmax Classifier (Multinomial Logistic Regression)

Softmax Classifier (Multinomial Logistic Regression)

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

Q: What is the min/max possible loss L_i?

	unnormalized log probabilities	exp		normalize	probabilities	
cat	**3.2**	→	**24.5**	→	**0.13**	→ L_i = -log(0.13) = **0.89**
car	5.1	→	164.0	→	0.87	
frog	-1.7	→	0.18	→	0.00	

Slide Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Softmax Classifier (Multinomial Logistic Regression)

Softmax Classifier (Multinomial Logistic Regression)

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

Q2: Usually at initialization W is small so all s ≈ 0. What is the loss?

cat	**3.2**		**24.5**	**0.13** → L_i = -log(0.13) = **0.89**
car	5.1	exp →	164.0	normalize → 0.87
frog	-1.7		0.18	0.00
	unnormalized log probabilities			probabilities

Slide Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Softmax Classifier (Multinomial Logistic Regression)

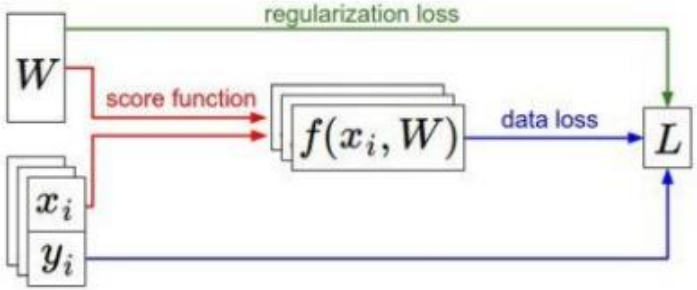
Recap

- We have some dataset of (x,y)
- We have a **score function:** $s = f(x; W) \overset{\text{e.g.}}{=} Wx$
- We have a **loss function**:

Softmax

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)$$ Full loss



Slide Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Softmax Classifier (Multinomial Logistic Regression)

Recap

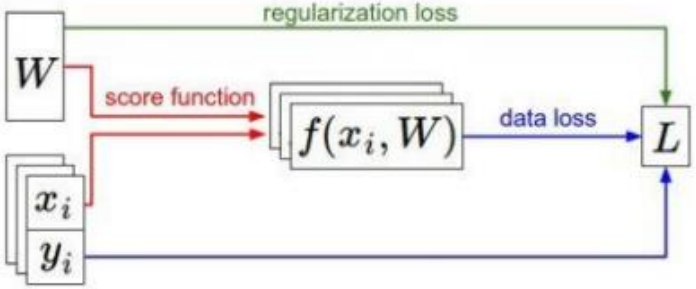
How do we find the best W?

- We have some dataset of (x,y)
- We have a **score function:** $s = f(x; W) \overset{\text{e.g.}}{=} Wx$
- We have a **loss function**:

Softmax

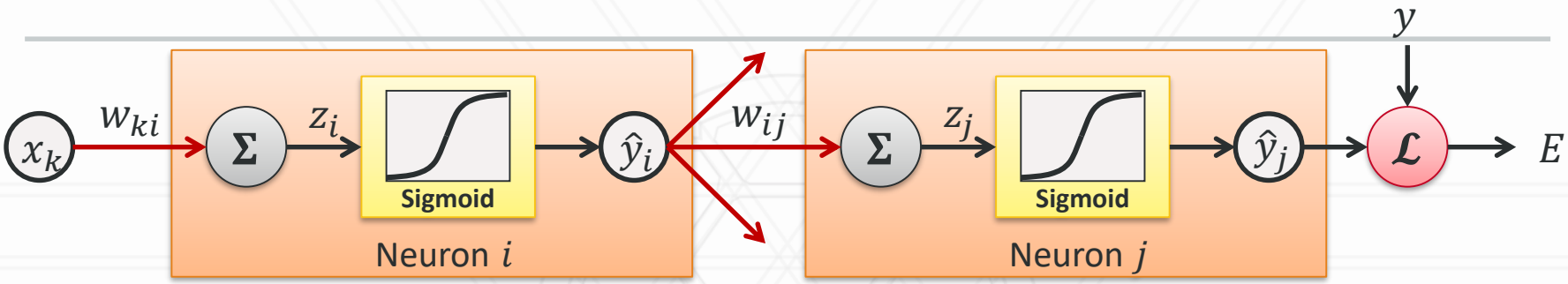
$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)$$ Full loss



Slide Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Multi-Layer: Backpropagation



$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial \hat{y}_j}\frac{d\hat{y}_j}{dz_j}$$

$$\frac{\partial E}{\partial \hat{y}_i} = \sum_j \frac{\partial E}{\partial z_j}\frac{dz_j}{d\hat{y}_i} = \sum_j w_{ij}\frac{\partial E}{\partial z_j} = \sum_j w_{ij}\frac{\partial E}{\partial \hat{y}_j}\frac{d\hat{y}_j}{dz_j}$$

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

Slide credit: Bohyung Han

Back Propagation Revisited

- On board (gradient w.r.t elements): Please take notes!
- Vector format of the backpropagation on slides

Neural Network: Definition

Neural Network Consider a neural network defined as following:

$$loss = L(Y, \bar{Y})$$
$$\bar{Y} = W_n \sigma(W_{n-1} \sigma(\cdots \sigma(W_1 X + b_1 1^T) \cdots) + b_{n-1} 1^T) + b_n 1^T$$

Neural Network: Forward Pass

Forward pass The forward pass of the neural network can be done as the following. We first initialize D_1:

$$D_1 = X$$

Then we can iteratively calculate:

$$\begin{cases} E_k = W_{k-1} D_{k-1} + b_{k-1} 1^\top \\ D_k = \sigma(E_k) = \sigma(W_{k-1} D_{k-1} + b_{k-1} 1^\top) \end{cases} \qquad \forall k = 2, \cdots, n \qquad (1)$$

Finally,

$$\bar{Y} = W_n D_n + b_n 1^\top$$

Neural Network: Back Prop I

Fact We will use the following results in deduction for backpropagation.

$$\frac{\partial f(AX)}{\partial X} = A^\top \nabla f(Y)|_{Y=AX}$$

$$\frac{\partial f(AX)}{\partial A} = \nabla f(Y)|_{Y=AX} X^\top$$

Notations We use the following notation. $.*$ means elementwise multiplication. $d\sigma(X)$ for $X \in \mathbf{R}^{a\times b}$ is also a matrix in $\mathbf{R}^{a\times b}$. The elements on the i-th row and j-th column $d\sigma(X)_{ij}$ is defined as $\frac{\partial \sigma(X_{ij})}{\partial X_{ij}}$:

Neural Network: Back Prop II

Backprop We start BP with following initialization:

$$\begin{cases} G_n = \frac{\partial L}{\partial Y} \\ \frac{\partial L}{\partial W_n} = G_n D_n^\top \\ \frac{\partial L}{\partial b_n} = G_n 1 \\ \frac{\partial L}{\partial D_n} = W_n^\top G_n \end{cases} \tag{2}$$

We know that $\frac{\partial L}{\partial D_k} = W_k^\top G_k$, so we can iteratively calculate the following:

$$\begin{cases} G_k = \frac{\partial L}{\partial E_{k+1}} = d\sigma(E_{k+1}) .* \frac{\partial L}{\partial D_{k+1}} = d\sigma(W_k D_k + b_k 1^\top) .* (W_{k+1}^\top G_{k+1}) \\ \frac{\partial L}{\partial W_k} = G_k D_k^\top \\ \frac{\partial L}{\partial b_k} = G_k 1 \end{cases} \quad \forall k = n-1, \cdots, 1 \tag{3}$$

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
- Convolution, pooling (“weight sharing”), fully connected layers

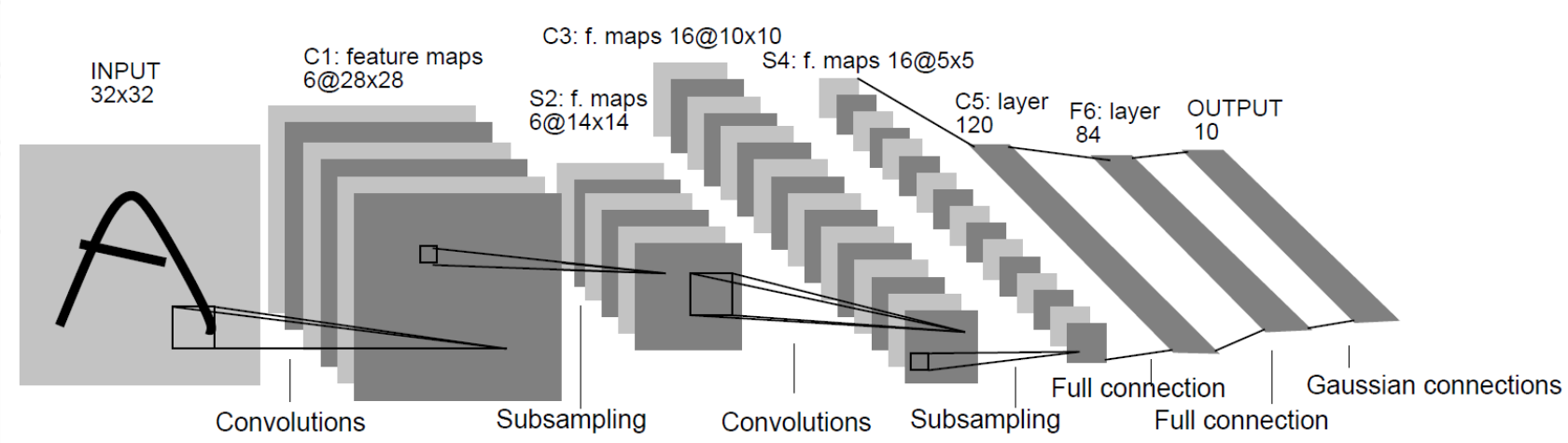
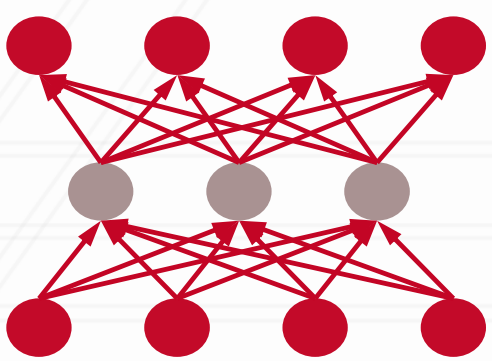


Image credit: LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. “Gradient-based learning applied to document recognition.” Proceedings of the IEEE, 1998.

Unsupervised Neural Networks

Autoencoders

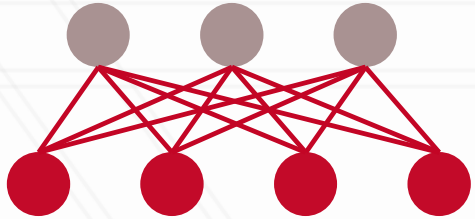
- Encode then decode the same input
- No supervision needed



H. Bourlard and Y. Kamp. 1988. Auto-association by multilayer perceptrons and singular value decomposition. Biol. Cybern. 59, 4-5 (September 1988), 291-294.

(Restricted) Boltzman Machines (RBMs)

- Stochastic networks that can learn representations
- Restricted version: neurons must form bipartite graph

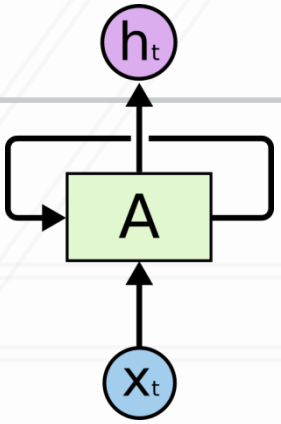


Ackley, David H; Hinton Geoffrey E; Sejnowski, Terrence J, "A learning algorithm for Boltzmann machines", Cognitive science, Elsevier, 1985.
Smolensky, Paul. "Chapter 6: Information Processing in Dynamical Systems: Foundations of Harmony Theory." In Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations, 1986.

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)

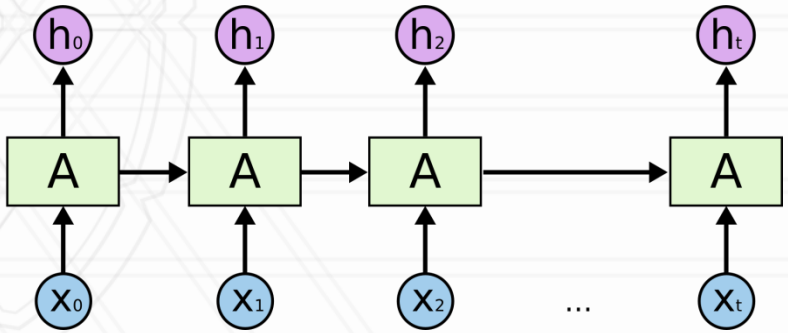


Image credit: Chritopher Olah's blog http://colah.github.io/posts/2015-08-Understanding-LSTMs/
Sepp Hochreiter (1991), Untersuchungen zu dynamischen neuronalen Netzen, Diploma thesis. Institut f. Informatik, Technische Univ. Munich. Advisor: J. Schmidhuber.
Y. Bengio, P. Simard, P. Frasconi. Learning Long-Term Dependencies with Gradient Descent is Difficult. In TNN 1994.

Long Short Term Memory (LSTM)

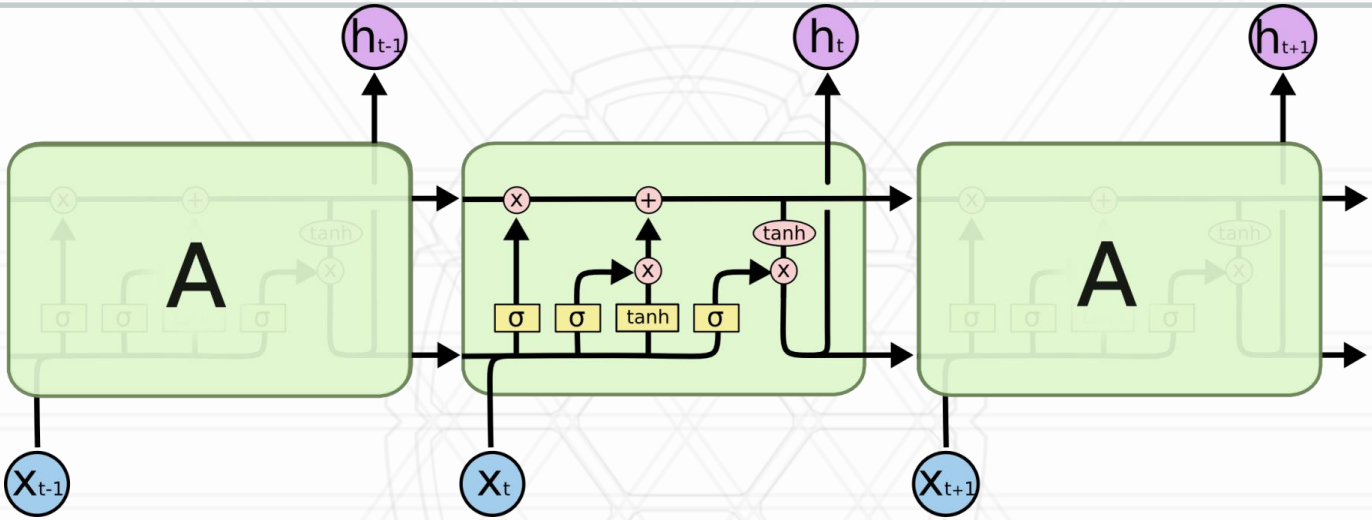


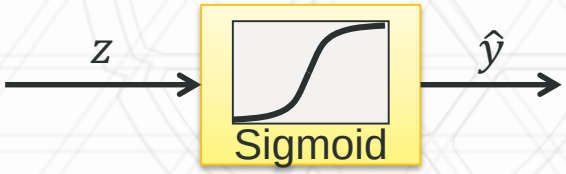
Image credit: Christopher Colah's blog, http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- 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.

Issues in Deep Neural Networks

- Large amount of 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
- Overfitting
  - Learned function fits training data well, but performs poorly on new data (high capacity model, not enough training data)
- Vanishing gradient problem



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

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

Slide credit: adapted from Bohyung Han

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
  - unsupervised layer-wise pre-training
  - ReLU, dropout, layer normalizatoin
- + Big data + GPU computing =
- Large outperformance on many datasets (Vision: ILSVRC’12)

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

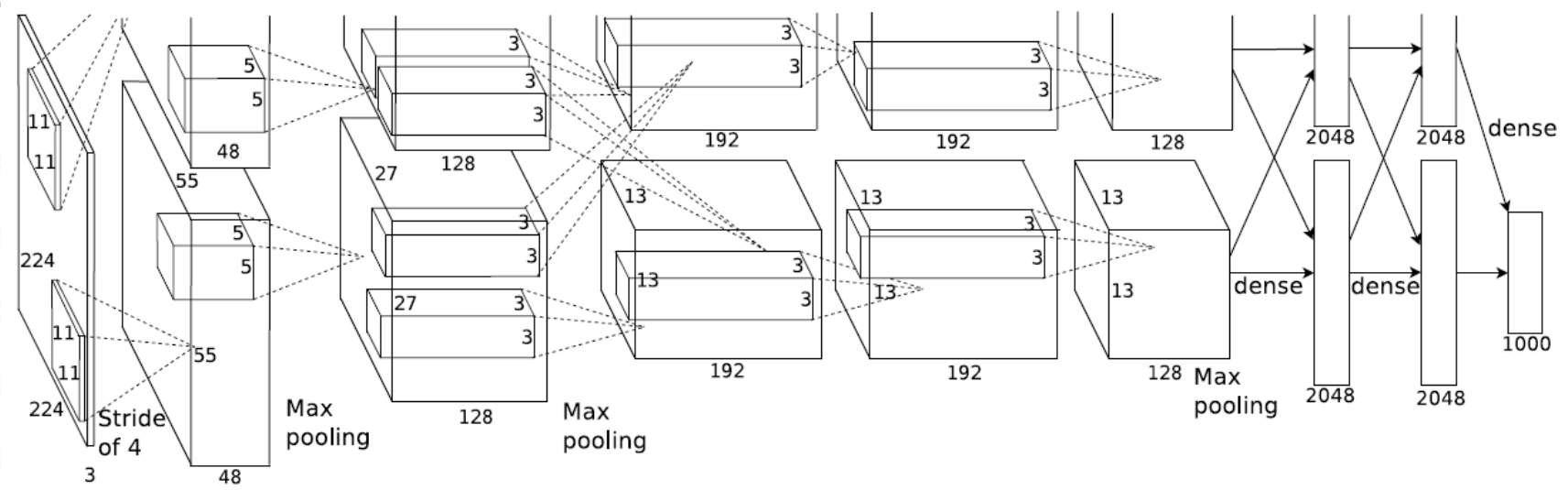
Big Data

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



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

Figure credit: Krizhevsky et al, NIPS 2012.

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

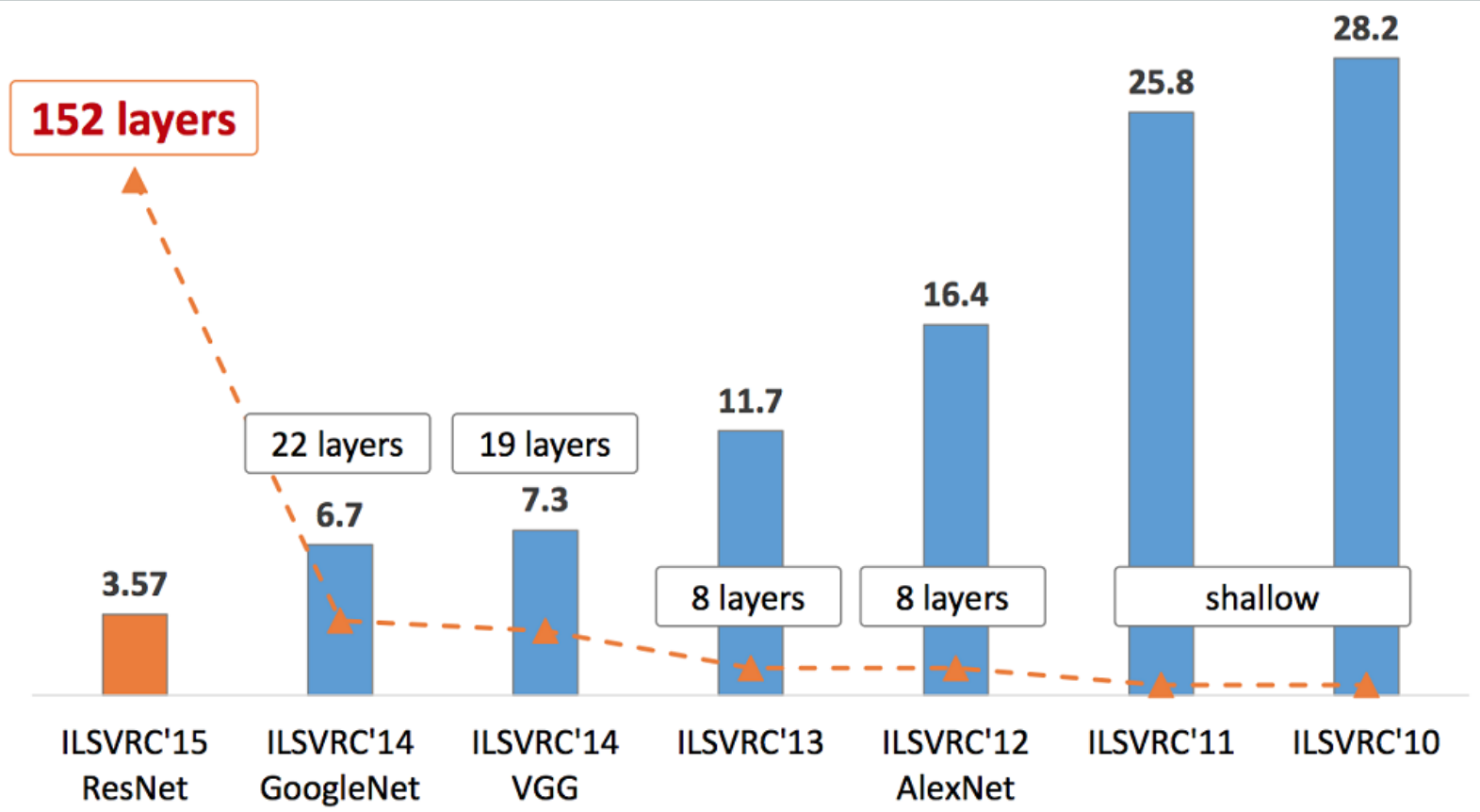


Figure from: K. He, X. Zhang, S. Ren, J. Sun. "Deep Residual Learning for Image Recognition". arXiv 2015. (slides)

Slide credit: Bohyung Han

Questions?

References (& great tutorials):

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/

UNIVERSITY OF
MARYLAND

Furong Huang
3251 A.V. Williams, College Park, MD 20740
301.405.8010 / furongh@cs.umd.edu