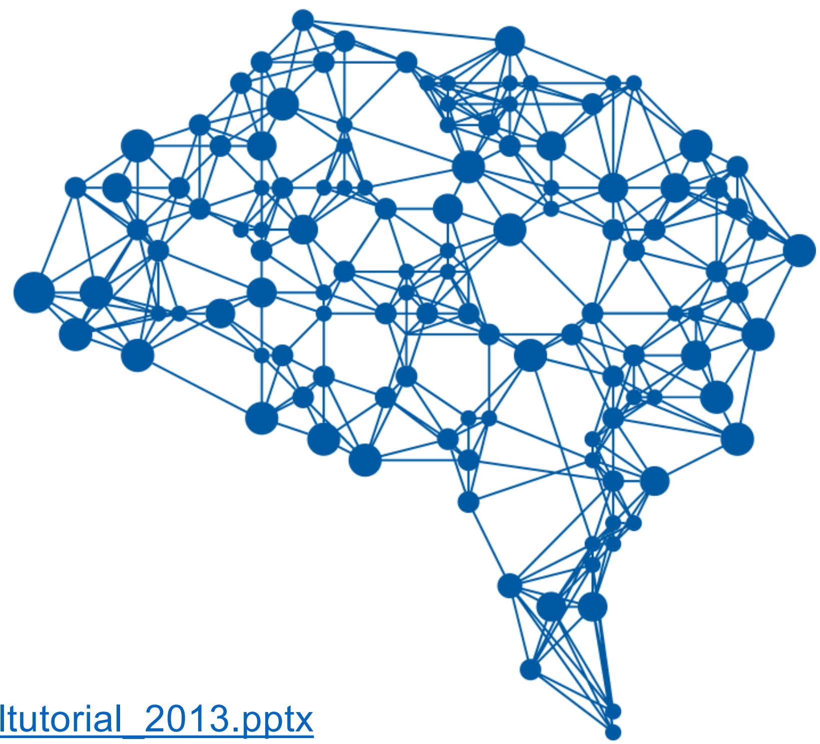
# Deep Learning Overview

**Joseph E. Gonzalez**

jegonzal@cs.berkeley.edu



Borrowed heavily from excellent talks by:

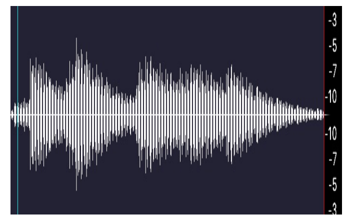
- **Adam Coates:** http://ai.stanford.edu/~acoates/coates_dltutorial_2013.pptx
- **Fei-Fei Li and Andrej Karpathy:** http://cs231n.stanford.edu/syllabus.html

# Machine Learning ➔ Function Approximation

## Object Recognition

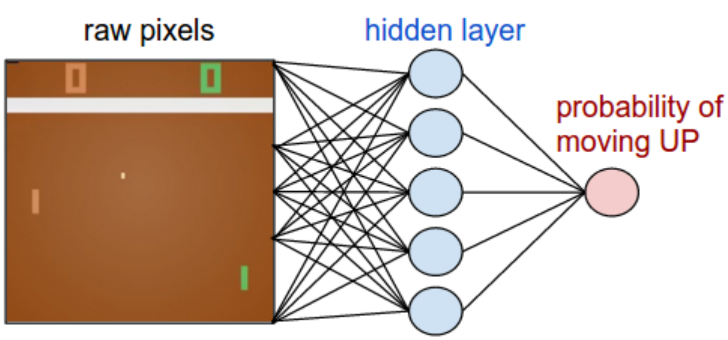
**Label**:*Cat*

## Speech Recognition

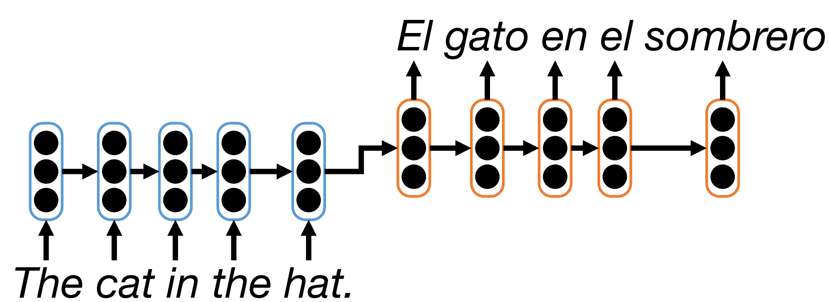


*“The cat in the hat”*

## Robotic Control




## Machine Translation

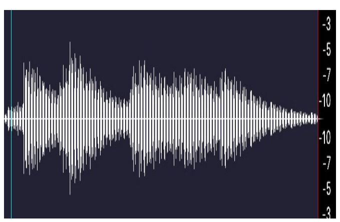


# Function Approximation Pipeline

Object Recognition

**Label**:*Cat*

Speech Recognition

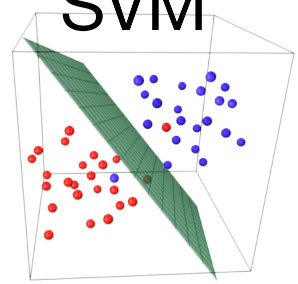


*“The cat in the hat”*

# Function Approximation Pipeline

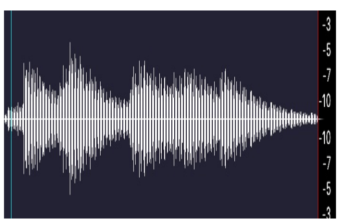
## Object Recognition

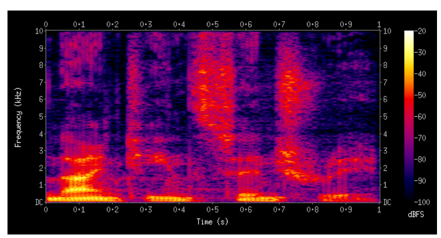
SVM



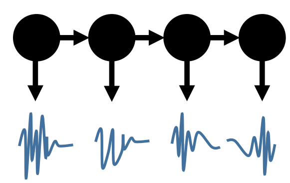
**Label**:*Cat*

## Speech Recognition



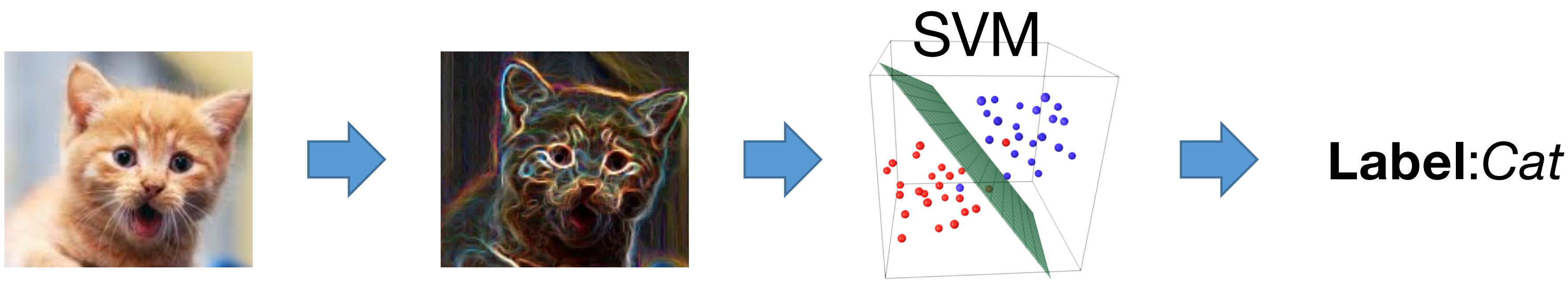


HMM

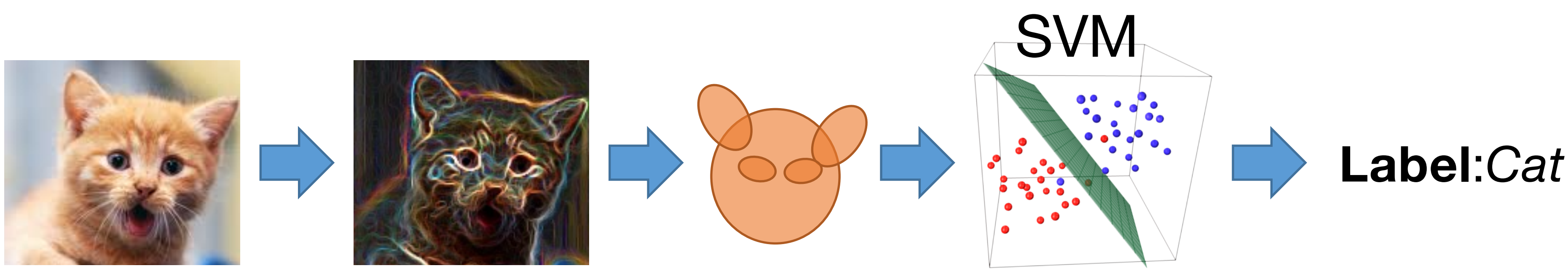


*“The cat in the hat”*

# Function Approximation Pipeline

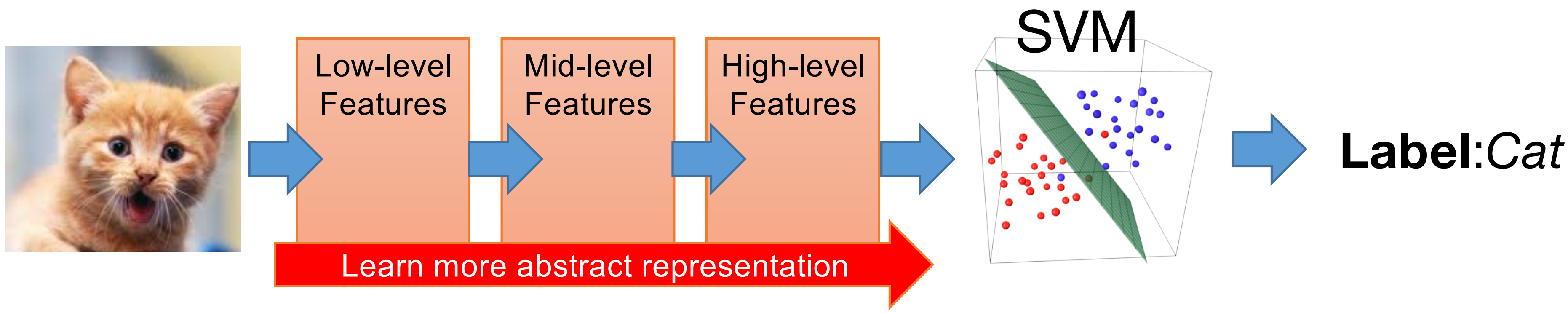



Often build multiple layers of features to abstract the input




Deep learning tries to automated this process.

# Function Approximation Pipeline




**Deep Learning:** automatically *learn a deep hierarchy of abstract features* along with the classifier.

- Typically using neural networks
  - composable general function approximators

# Why is Deep Learning so Successful?

- Feature engineering essential to many applications
  - Expensive hand-engineering of “layers” of representation.
  - Deep learning **automates** the process of **feature engineering**

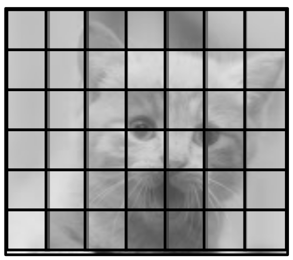
- Previous attempts were limited by data and computation
  - We now have **access to substantial** amounts of **data** and **computation**

- Deep learning techniques are inherently **compositional**
  - Easy to extend and combine → rapid development

# Crash Course in Neural Networks

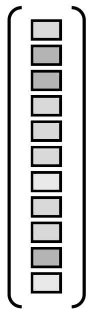
# Supervised Learning

➢ Predict is this a picture of a cat?

$$x =$$ 

# Supervised Learning

- Predict is this a picture of a cat?

$x =$  $=$  $\in \mathbb{R}^d$ $\qquad y \in \{0, 1\}$

- and training data: $\mathcal{D}_{\text{train}} = \{(x_i, y_i)\}_{i=1}^{n}$

- Learn a function: $$f_w(x) = y$$

  - By estimating the parameters *W*

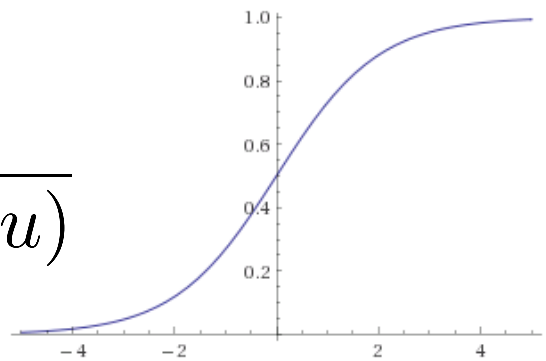
# Logistic Regression for Binary Classification

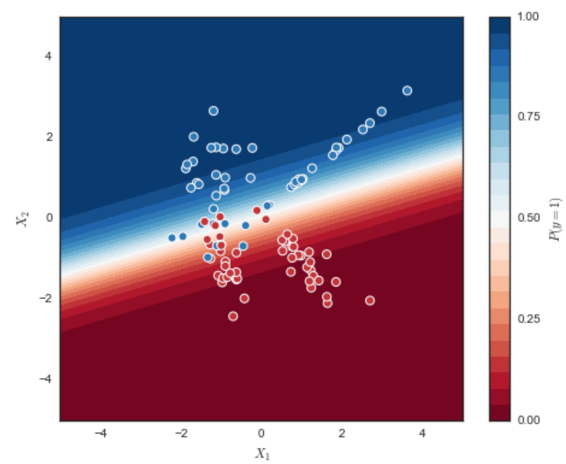
- Consider the simple function family:

$$f_w(x) = \sigma\left(w^T x\right) = \sigma\left(\sum_{j=1}^{d} w_j x_j\right) = P(y = 1 \,|\, x)$$

- With non-linearity:

$$\sigma(u) = \frac{1}{1 + \exp(-u)}$$



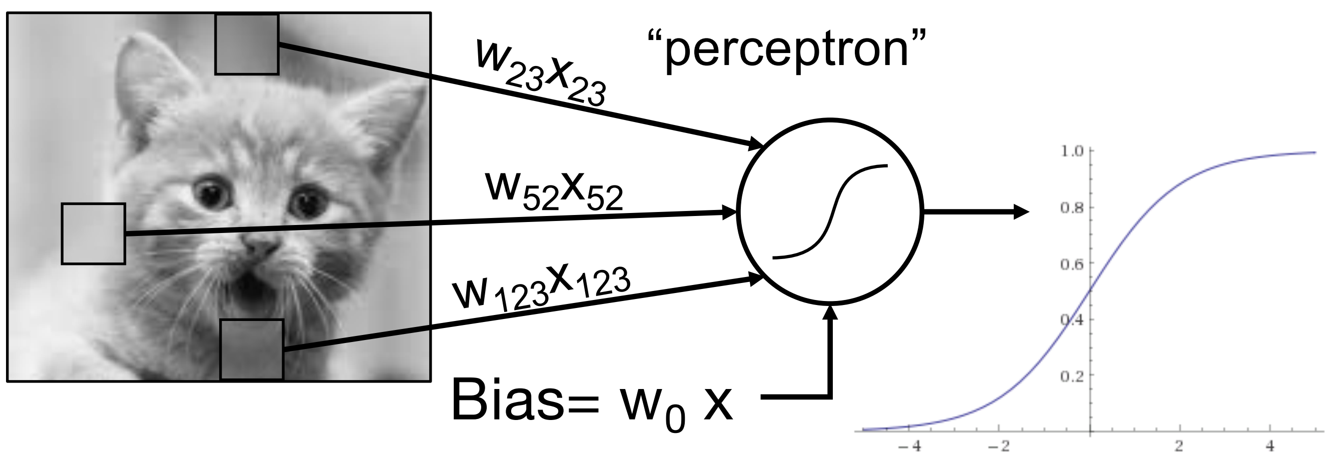



# Logistic Regression as a "Neuron"

- Consider the simple function family:

$$\sigma(u) = \frac{1}{1 + \exp(-u)}$$

$$f_w(x) = \sigma\left(w^T x\right) = \sigma\left(\sum_{j=1}^{d} w_j x_j\right) = P(y = 1 \,|\, x)$$




*Neuron "fires" if weighted sum of input is greater than zero.*

# Learning the logistic regression model

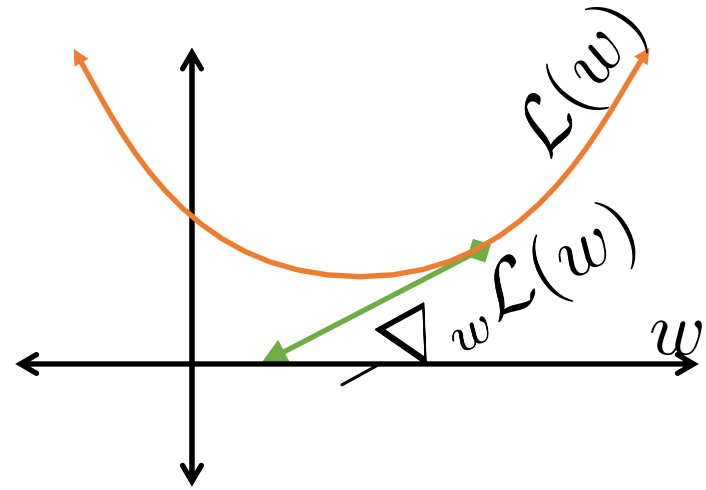
- Consider the simple function family: $$\sigma(u) = \frac{1}{1+\exp(-u)}$$

$$f_w(x) = \sigma\left(w^T x\right) = \sigma\left(\sum_{j=1}^{d} w_j x_j\right) = P(y = 1 \,|\, x)$$

- **Goal:** *find* ***w*** *that minimizes the loss on the training data:*

$$\mathcal{L}(w) = \sum_{i=1}^{n} L\left(f_w(x_i), y_i\right) = \sum_{i=1}^{n} \underbrace{f_w(x_i)^{y_i}\left(1 - f_w(x_i)\right)^{1-y_i}}_{\text{likelihood}}$$

# Numerical Optimization




- Gradient Descent:

$$w^{(t+1)} = w^{(t)} - \eta_t \nabla_w \mathcal{L}(w)$$

  - Convex → Guaranteed to find optimal *w*

- Stochastic gradient descent:

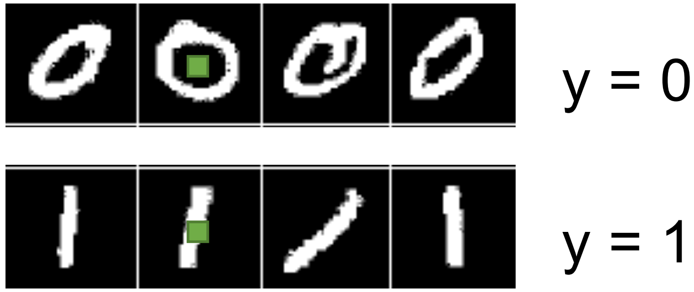
Slow

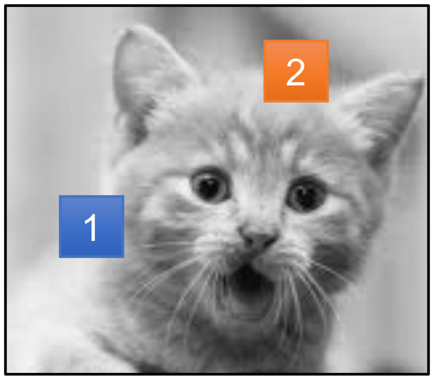
$$\nabla_w \mathcal{L}(w) = \sum_{i=1}^{n} \nabla_w L\left(f_w(x_i), y_i\right)$$

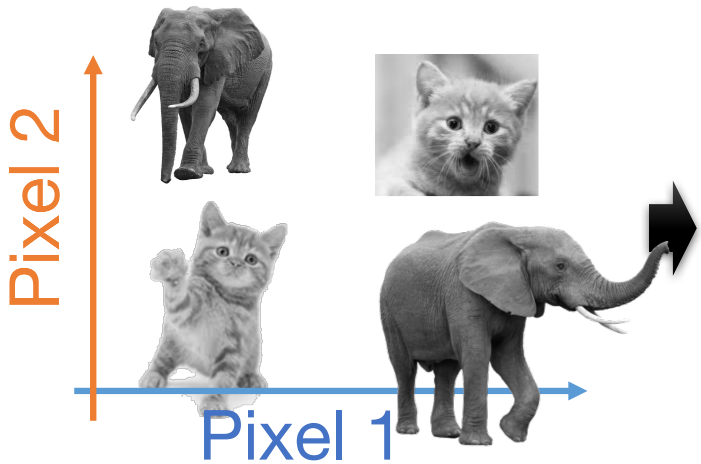
Approximate $\approx n \nabla_w L\left(f_w(x_i), y_i\right) \text{ for } (x_i, y_i) \sim \mathcal{D}$

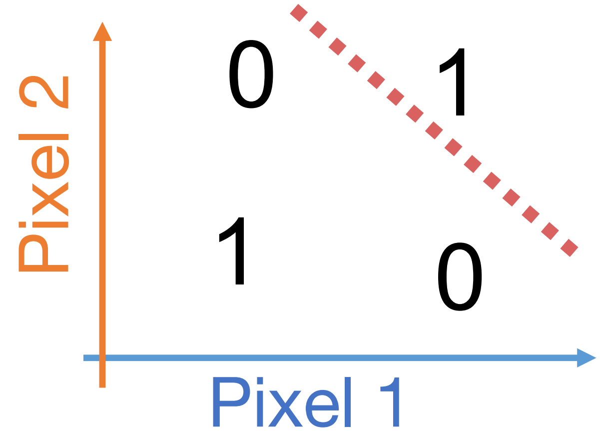
# Logistic Regression: Strengths and Limitations

- Widely used machine learning technique
  - convex → efficient to learn
  - easy to interpret model weights
  - works well given good features
- Limitations:
  - Restricted to linear relationships → sensitive to choice of features

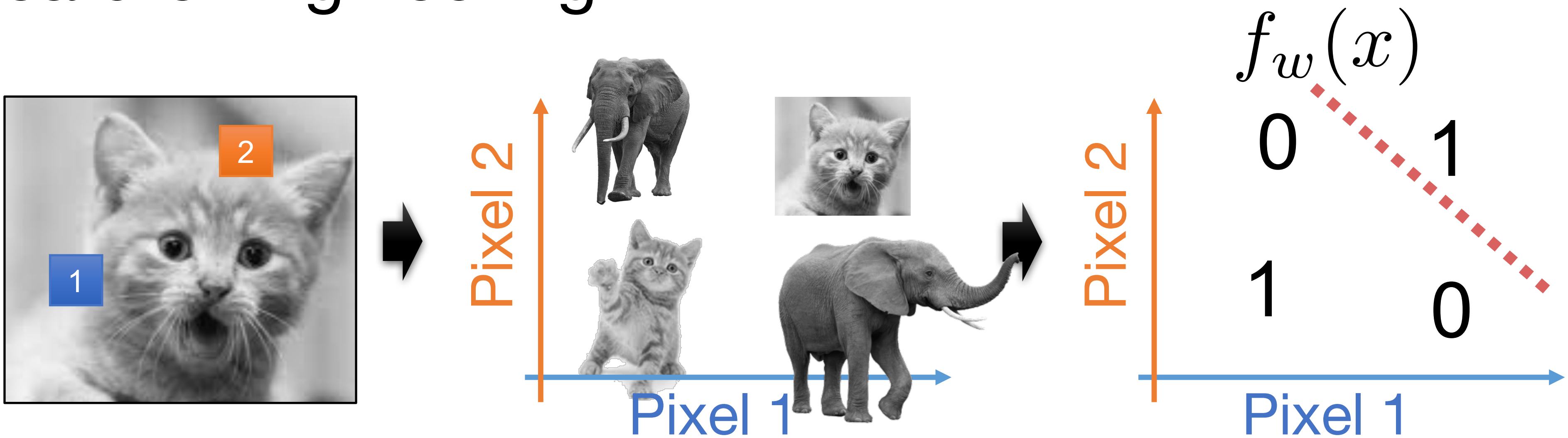


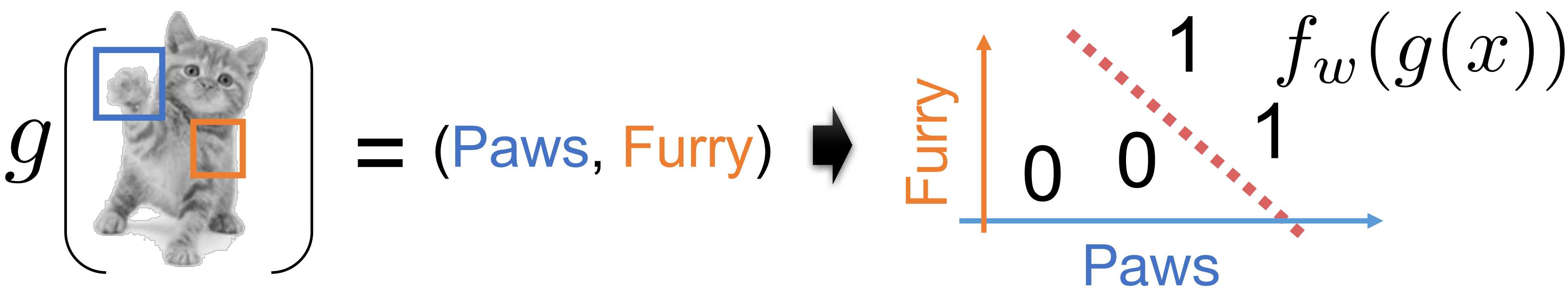



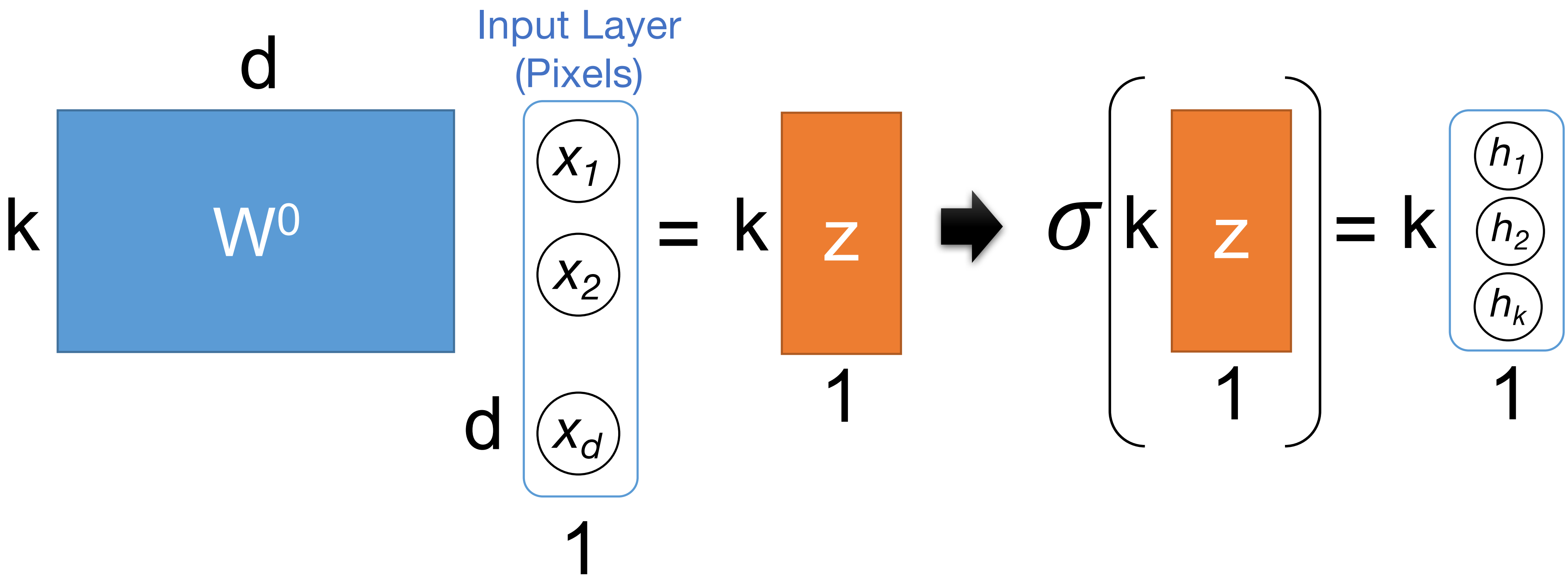
# Feature Engineering



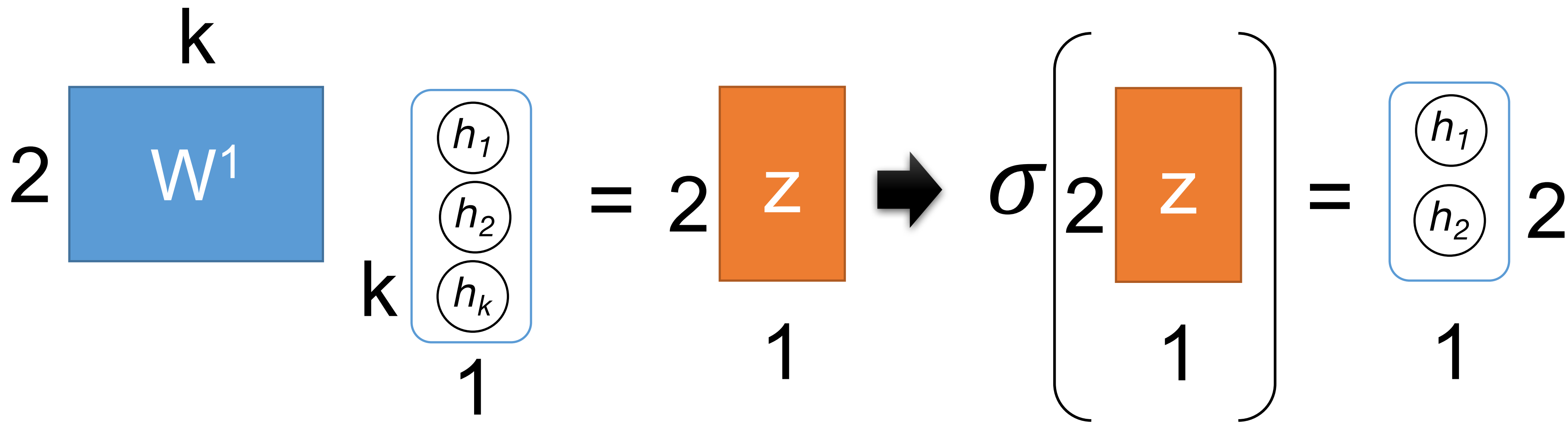

- Rather than use raw **pixels build/train feature functions**:



# Composition Linear Models and Nonlinearities



# Composition Linear Models and Nonlinearities

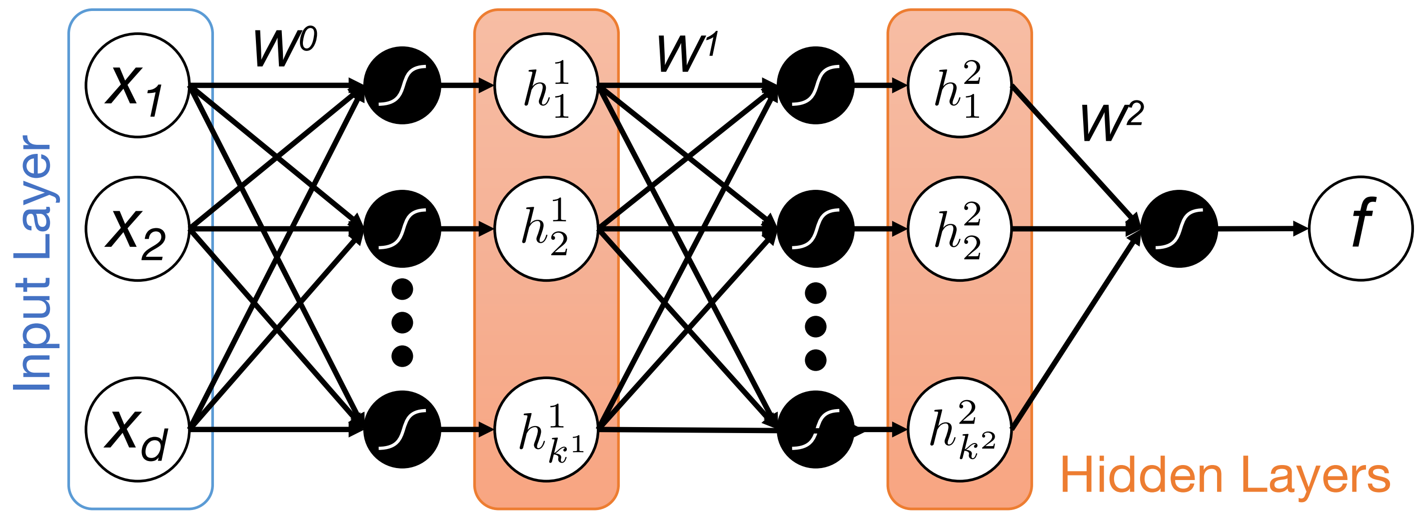


# Neural Networks

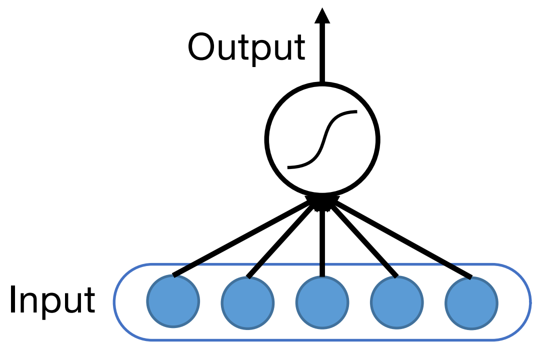
- Composing "perceptrons"

$$x \rightarrow \sigma(W^0 x) \rightarrow h^1 \rightarrow \sigma(W^1 h^1) \rightarrow h^2 \rightarrow \sigma(W^2 h^2) \rightarrow f$$

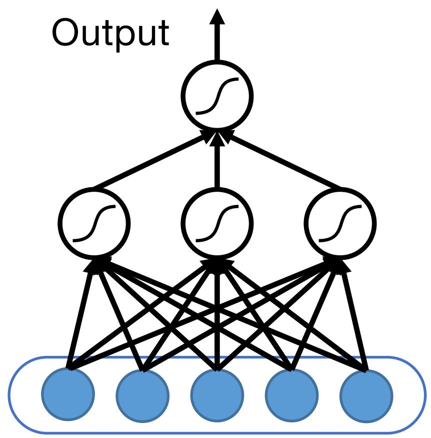
$$y = f_{W^0,W^1,W^2}(x) = \sigma\left(W^2\ \sigma\left(W^1\ \sigma\left(W^0 x\right)\right)\right)$$

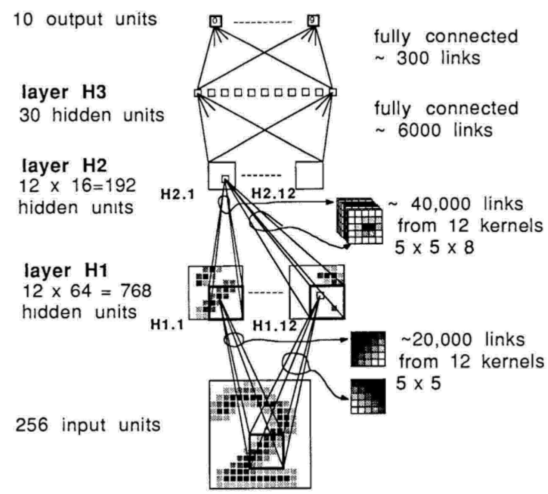


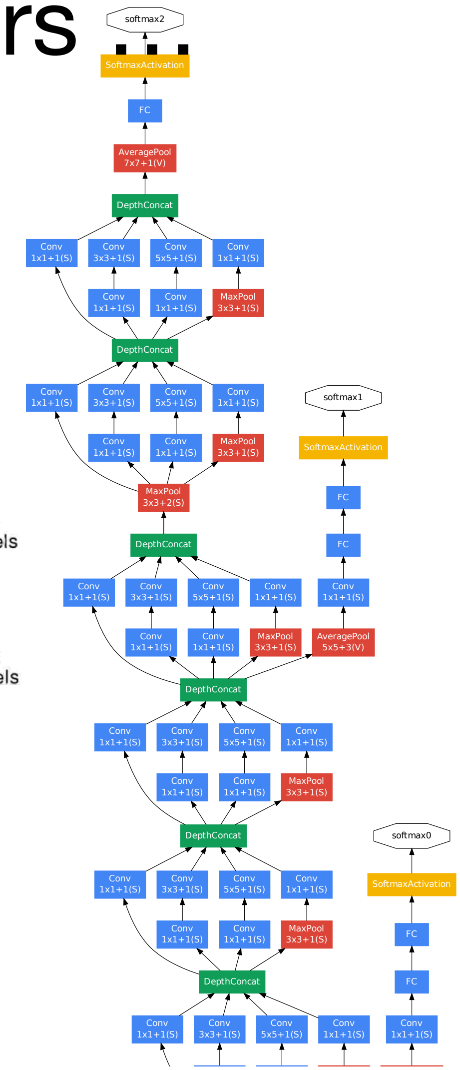
# Deep Learning →Many hidden layers




Logistic Regression



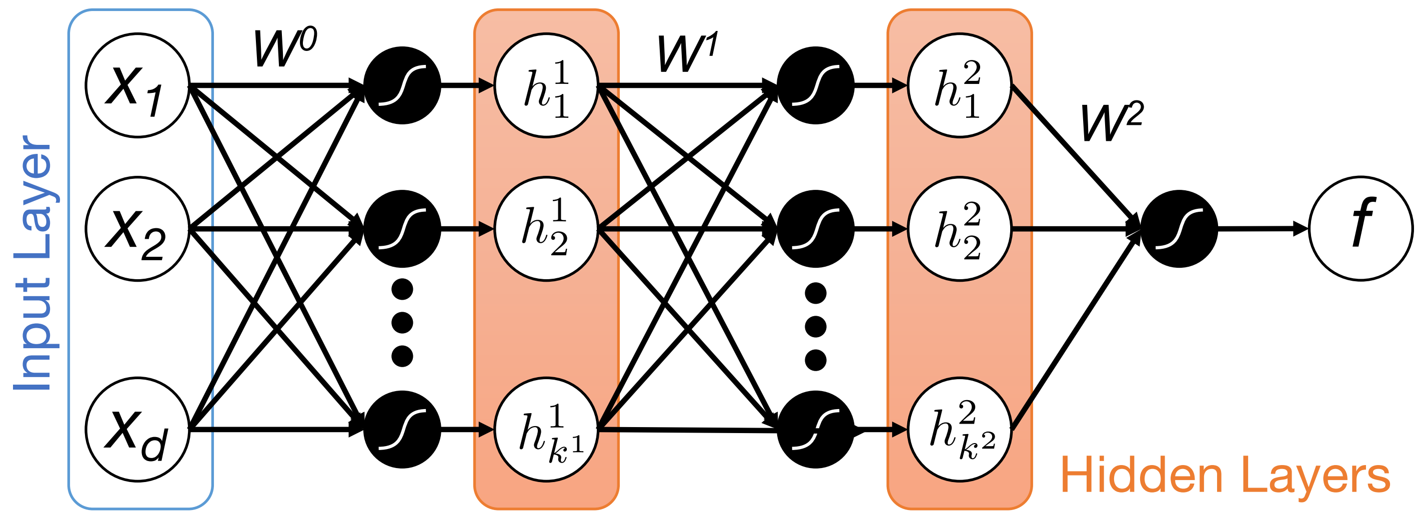



# Neural Networks

➢ Composing non-linear models (e.g., Logistic Regression):

$$y = f_{W^0,W^1,W^2}(x) = \sigma\left(W^2\ \sigma\left(W^1\ \sigma\left(W^0 x\right)\right)\right)$$




$$x \to \sigma(W^0 x) \to h^1 \to \sigma(W^1 h^1) \to h^2 \to \sigma(W^2 h^2) \to f$$

➢ Learn $W^0$, $W^1$, and $W^2$ using **Backpropagation** + SGD

# Backpropagation in Neural Networks

$$y = f_{W^0,W^1,W^2}(x) = \sigma\left(W^2\ \sigma\left(W^1\ \sigma\left(W^0 x\right)\right)\right)$$

- Need to compute the gradient of the loss wrt. $W^0$, $W^1$, and $W^2$

$$\nabla_{W^0,W^1,W^2} L(y, f_{W^0,W^1,W^2}(x))$$

- Use chain rule to push gradients back through dataflow graph:

$$\begin{tikzcd}
x \arrow[r] & \boxed{\sigma(W^0 x)} \arrow[r] & h^1 \arrow[r] & \boxed{\sigma(W^1 h^1)} \arrow[r] & h^2 \arrow[r] & \boxed{\sigma(W^2 h^2)} \arrow[r] & f \\
W^0 \arrow[ur] & & W^1 \arrow[ur] & & W^2 \arrow[ur] & &
\end{tikzcd}$$

# Backpropagation in Neural Networks

$$x \to \boxed{\sigma(W^0 x)} \to h^1 \to \boxed{\sigma(W^1 h^1)} \to h^2 \to \boxed{\sigma(W^2 h^2)} \to f$$

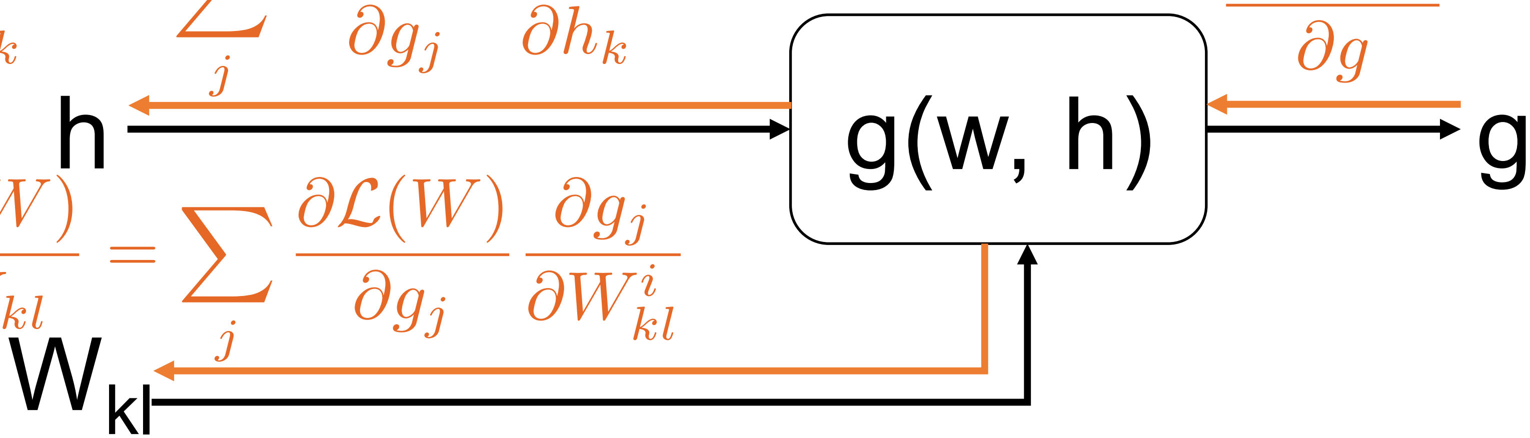
$$W^0 \qquad W^1 \qquad W^2$$

- Define a general operator:

$$\frac{\partial \mathcal{L}(W)}{\partial h_k} = \sum_j \frac{\partial \mathcal{L}(W)}{\partial g_j} \frac{\partial g_j}{\partial h_k}$$

$$\frac{\partial \mathcal{L}(W)}{\partial g}$$

$$\frac{\partial \mathcal{L}(W)}{\partial W_{kl}} = \sum_j \frac{\partial \mathcal{L}(W)}{\partial g_j} \frac{\partial g_j}{\partial W^i_{kl}}$$



# Backpropagation in Neural Networks

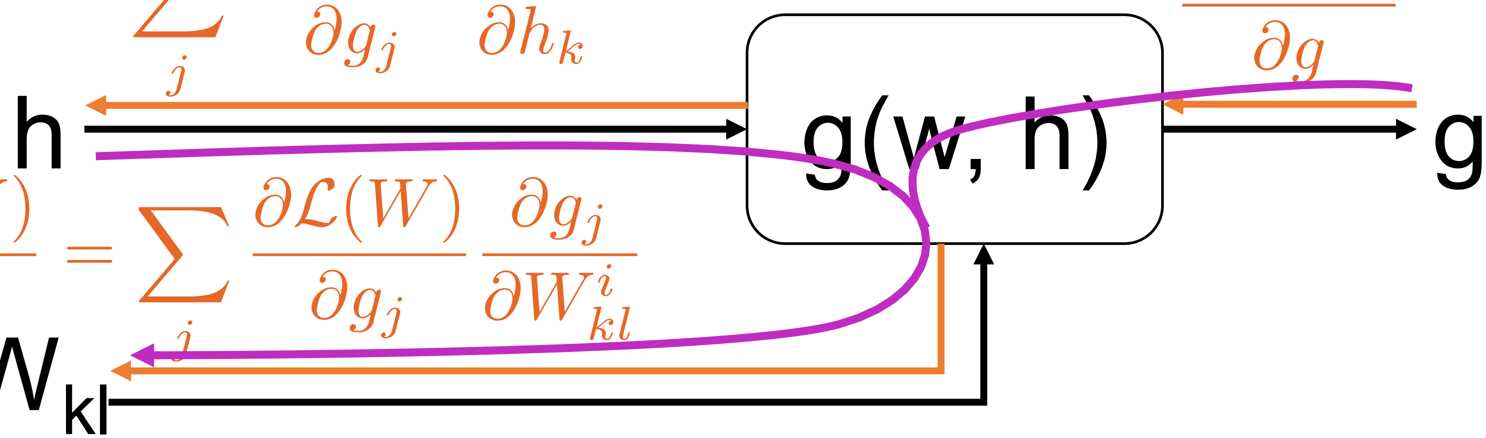
$$x \to \boxed{\sigma(W^0x)} \to h^1 \to \boxed{\sigma(W^1h^1)} \to h^2 \to \boxed{\sigma(W^2h^2)} \to f$$

$$W^0 \qquad\qquad W^1 \qquad\qquad W^2$$

- Define a general operator:

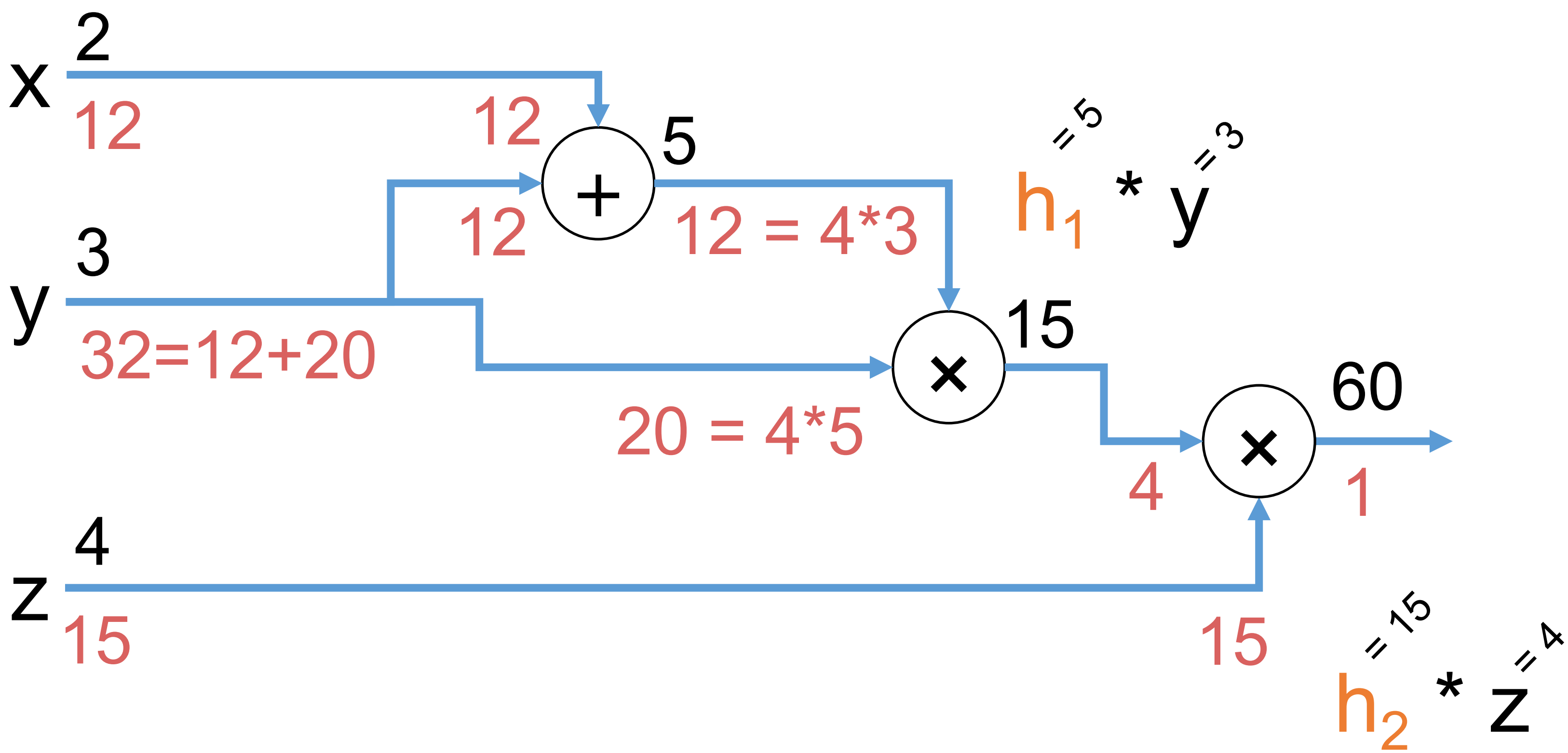
$$\frac{\partial \mathcal{L}(W)}{\partial h_k} = \sum_j \frac{\partial \mathcal{L}(W)}{\partial g_j}\frac{\partial g_j}{\partial h_k}$$

$$\frac{\partial \mathcal{L}(W)}{\partial g}$$



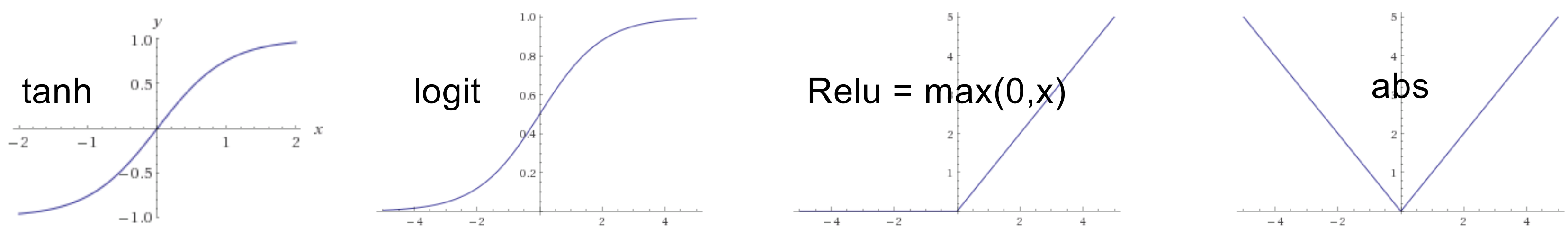

$$\frac{\partial \mathcal{L}(W)}{\partial W_{kl}} = \sum_j \frac{\partial \mathcal{L}(W)}{\partial g_j}\frac{\partial g_j}{\partial W^i_{kl}}$$

# Simple Example: $f(x, y, z) = (x + y) * y * z$



# Backpropagation

➢ Requires all operators to have well defined sub-gradients:




➢ **Enables Automatic Differentiation!**
  ➢ User defines forward flow → system derives efficient training alg.
  ➢ Easy to explore composition of new modules

➢ **Enables Efficient Gradient Computation**
  ➢ Cache forward calculation to accelerate gradients
  ➢ Compile optimized gradient computation

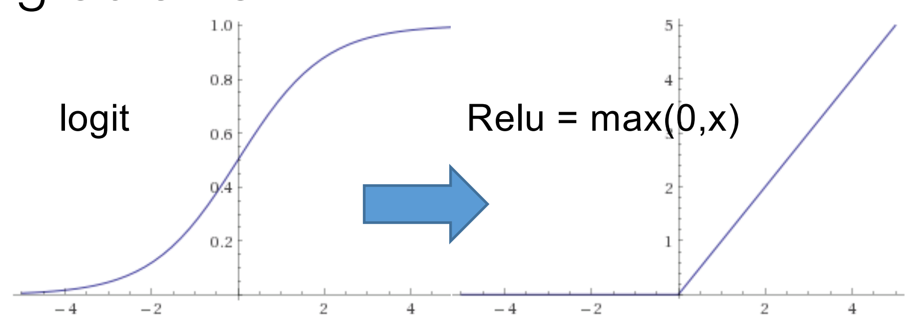
# General Purpose Systems For DNNs

- Distributed Parameter Servers
  - TensorFlow (DistBelief)
  - Microsoft Adam
- GPU Systems
  - TensorFlow
  - Caffe
  - Theano

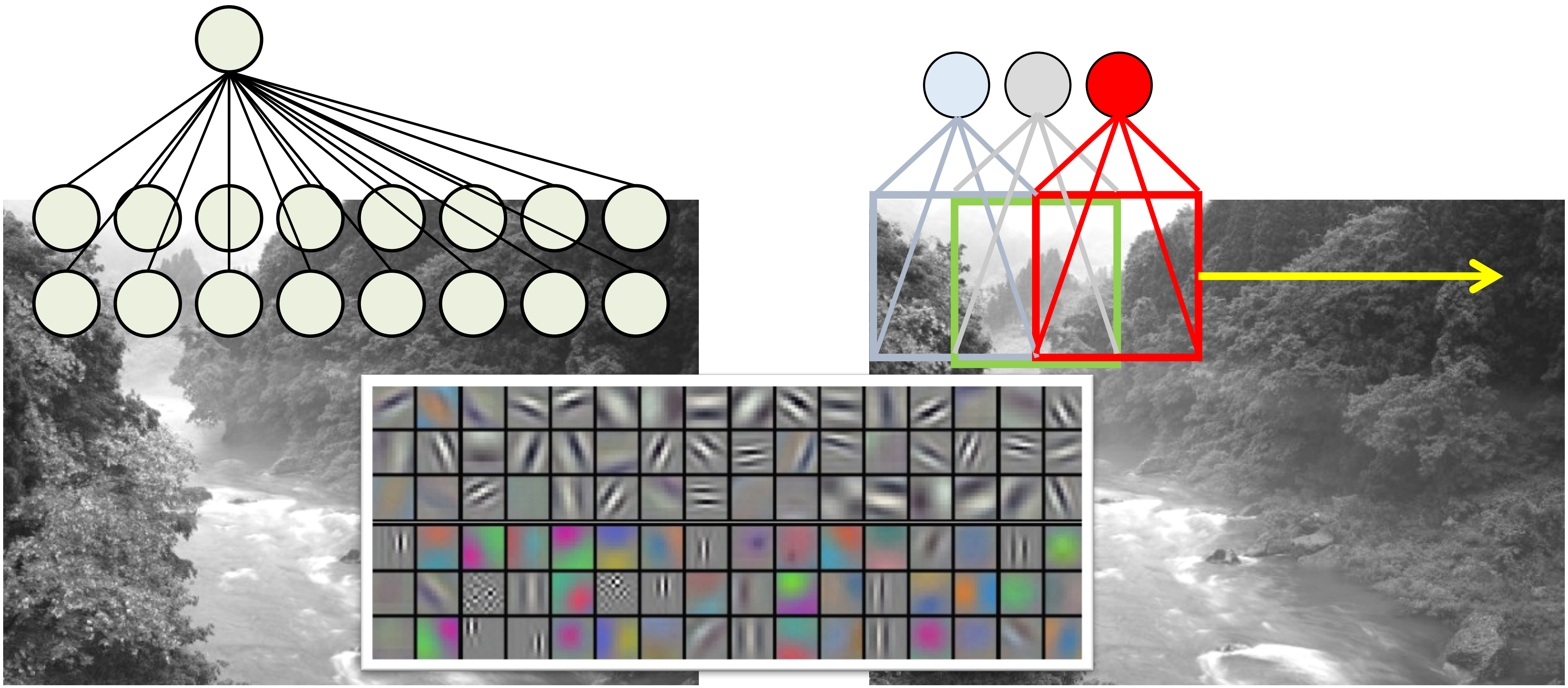
Demo of TensorFlow

# Challenges of Deep Neural Networks

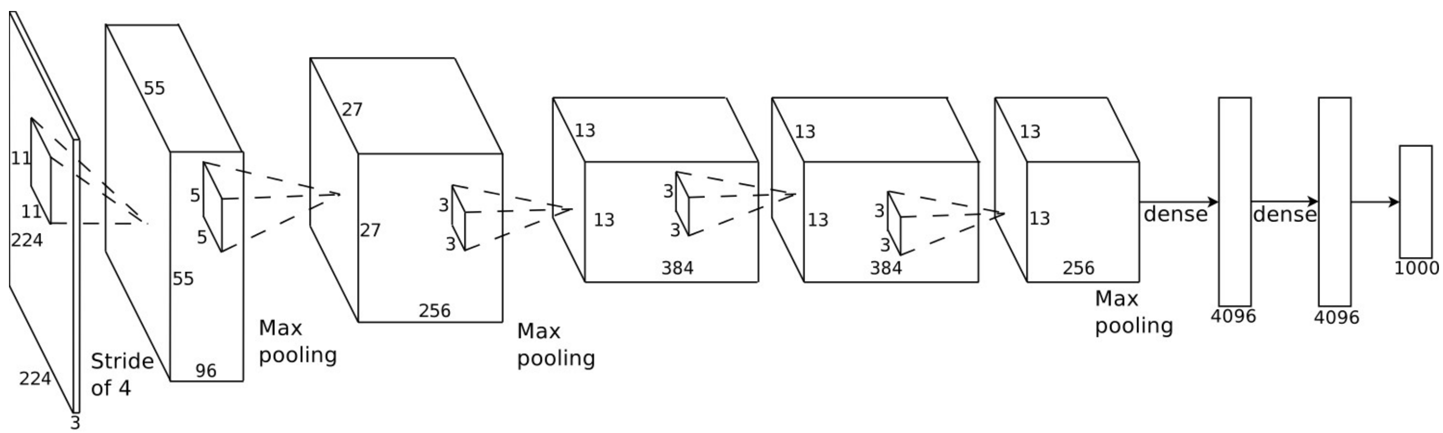
- Non-convex → (stochastic) gradient descent not guaranteed to converge to optimum
  - **Soln:** appear to be many good local optima
- High-dimensional → gradient descent converges slowly
  - **Soln:** hardware acceleration, improved algs. with momentum …
- Rich function class → overfitting
  - **Soln:** more data, early-stopping, drop-out, parameter sharing
- Saturation of sigmoid → decaying gradients
  - **Soln:** other forms of non-linearity



# Convolutional Neural Networks: Exploiting Spatial Sparsity



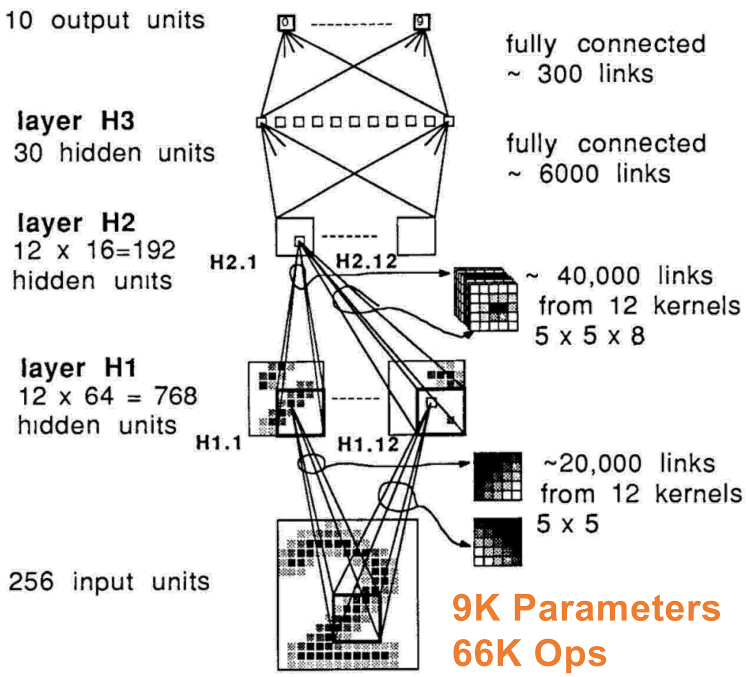
# Example: AlexNet (Krizhevsky et al., NIPS 2012)




- Introduced in 2012, significantly outperformed state-of-the-art (top 5 error of 16% compared to runner-up with 26% error)
  - Covered in reading …

# Growth in Model Complexity

LeCun et al, *"Backpropagation Applied to Handwritten Zip Code Recognition".* 1989



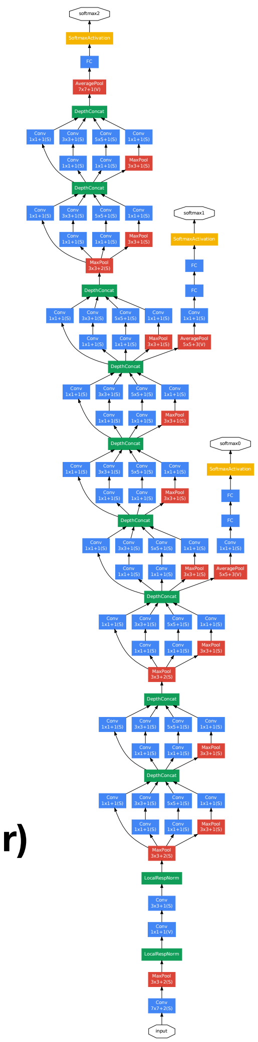

**9K Parameters**
**66K Ops**

Szegedy et al, *"Going Deeper with Convolution".* 2014

- Winner of ImageNet Large-Scale Visual Recognition Challenge 2014

- **GoogLeNet (7.89% error)**
  - **22 layers**
  - **6.8M parameters**
  - **1.5*B* flops**
  - **Ensemble of 7 models**

- **Current Best: ResNet (3.57% error)**
  - **152 layers**
  - **2.3M parameters**
  - **11.3*B* flops**
  - **Ensemble of 6 models**



# Cost of Computation (from Prediction Serving Lecture)

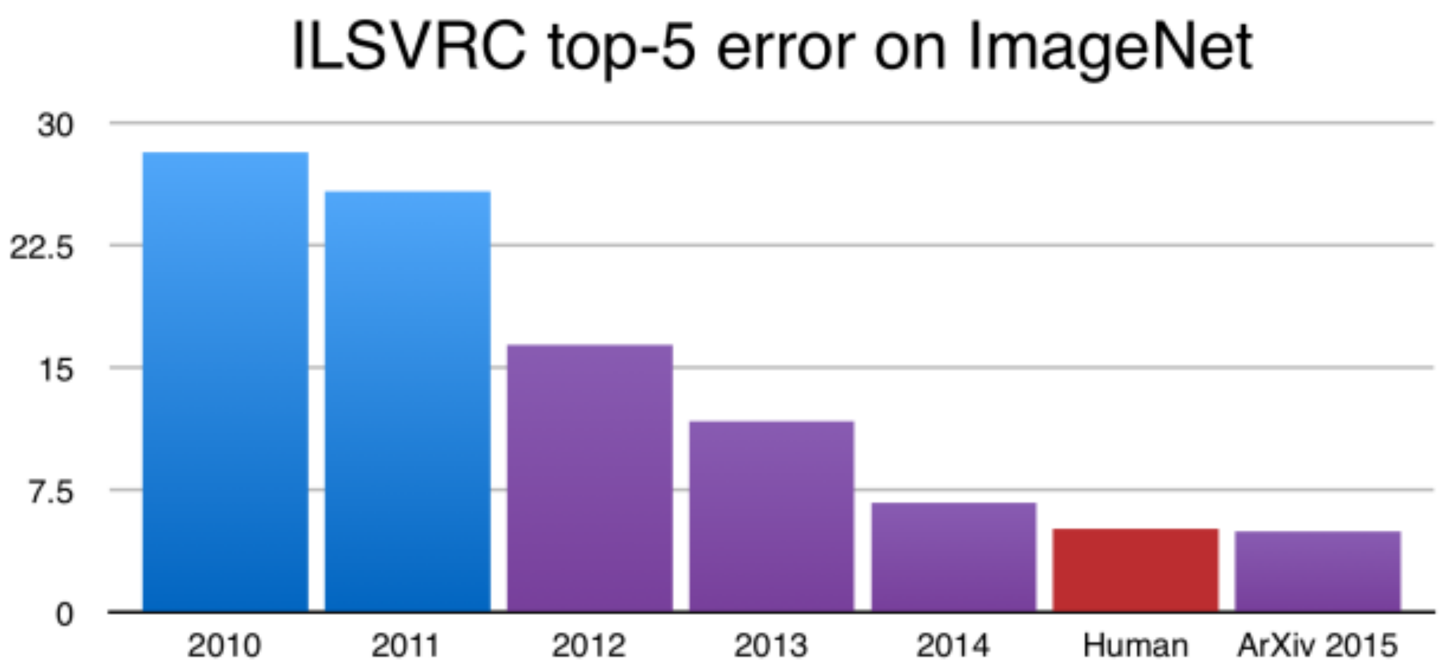
| Network: GoogLeNet | Batch Size | Titan X (FP32) | Tegra X1 (FP32) | Tegra X1 (FP16) |
|---|---|---|---|---|
| Inference Performance | 1 | 138 img/sec | 33 img/sec | 33 img/sec |
| Power | | 119.0 W | 5.0 W | 4.0 W |
| Performance/Watt | | 1.2 img/sec/W | 6.5 img/sec/W | 8.3 img/sec/W |
| Inference Performance | 128 (Titan X)<br>64 (Tegra X1) | 863 img/sec | 52 img/sec | 75 img/sec |
| Power | | 225.0 W | 5.9 W | 5.8 W |
| Performance/Watt | | 3.8 img/sec/W | 8.8 img/sec/W | 12.8 img/sec/W |

*Table 3 GoogLeNet inference results on Tegra X1 and Titan X. Tegra X1's total memory capacity is not sufficient to run batch size 128 inference.*

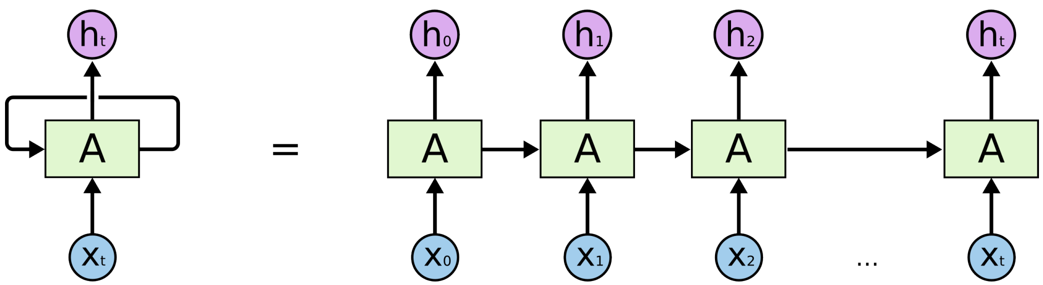
- 100's of millions of parameters + convolutions & unrolling
- Requires hardware acceleration

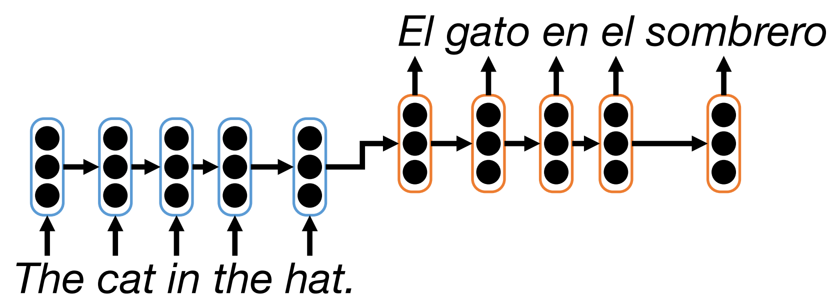
http://www.nvidia.com/content/tegra/embedded-systems/pdf/jetson_tx1_whitepaper.pdf

# Improvement on ImagNet Benchmark



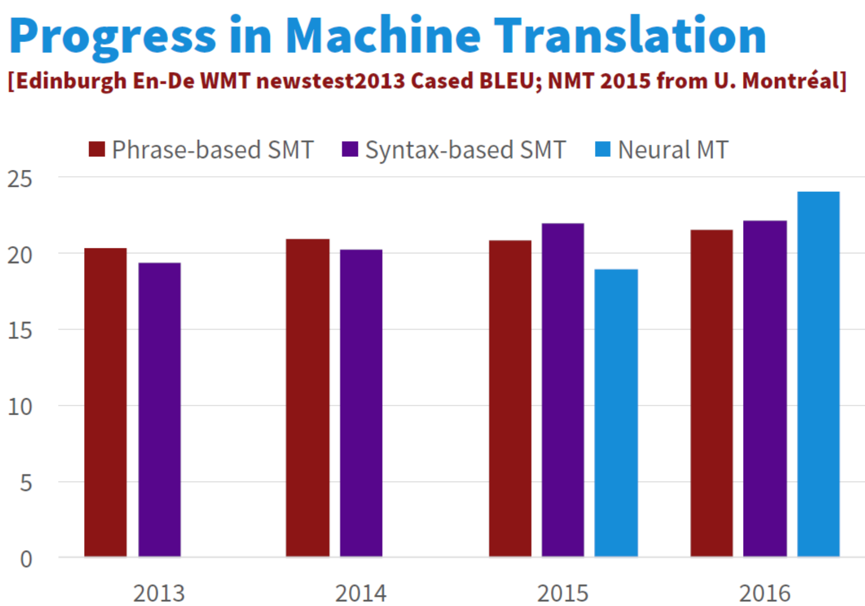
# Recurrent Neural Networks: Modeling Sequence Structure




- State of the art in speech recognition and machine translation
  - Required LSTM and GRU to address long dependencies
- Similar to the HMM from classical Bayesian ML

# Improvements in Machine Translation & Automatic Speech Recognition




From [Sennrich 2016, http://www.meta-net.eu/events/meta-forum-2016/slides/09_sennrich.pdf]