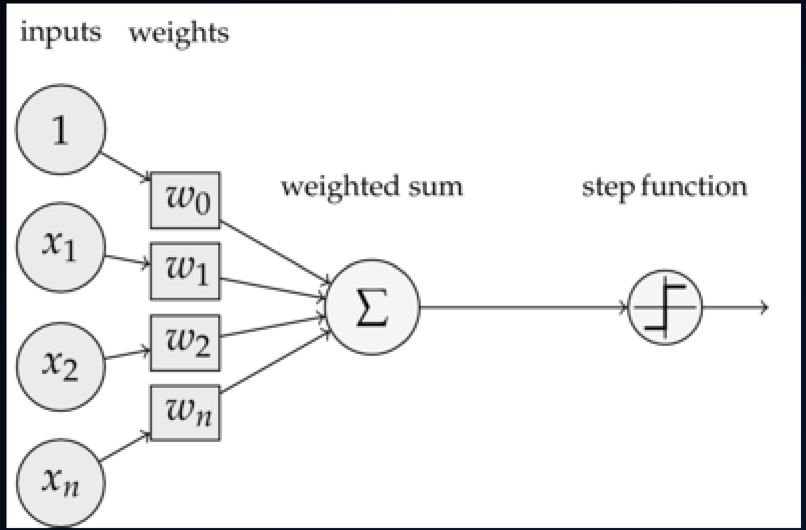
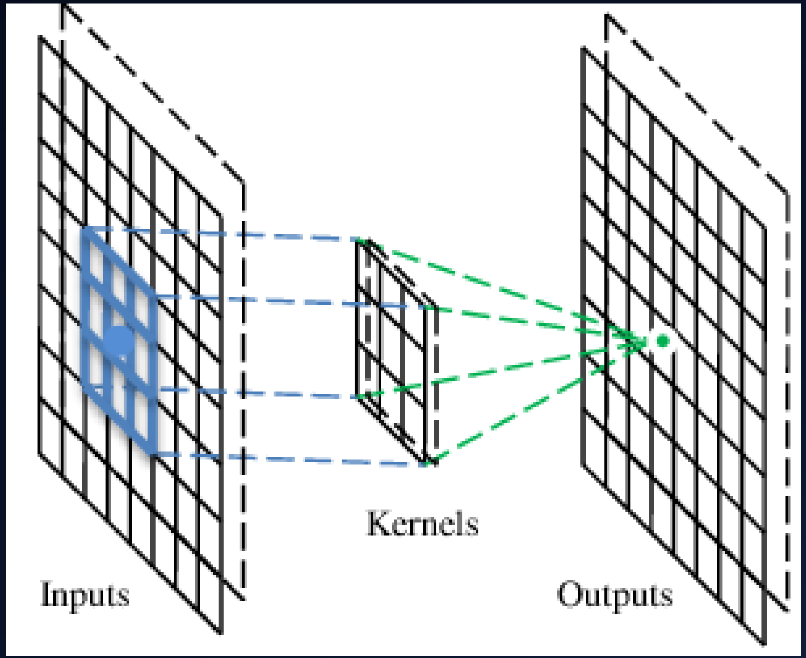
Neural Networks and Deep Learning

Joshua Achiam

Outline

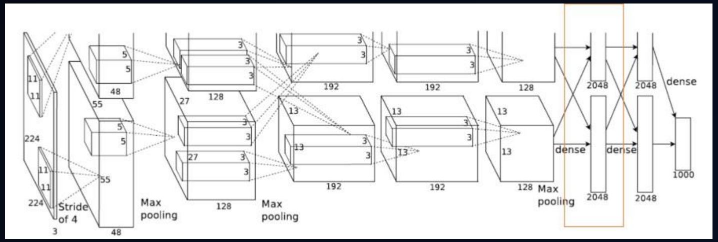
1. Fundamentals
    1. Perceptrons
    2. Early Neural Networks
    3. Backpropagation and SGD
    4. Universal Approximation Theorem
    5. Convolutional Networks
    6. Recurrent Networks





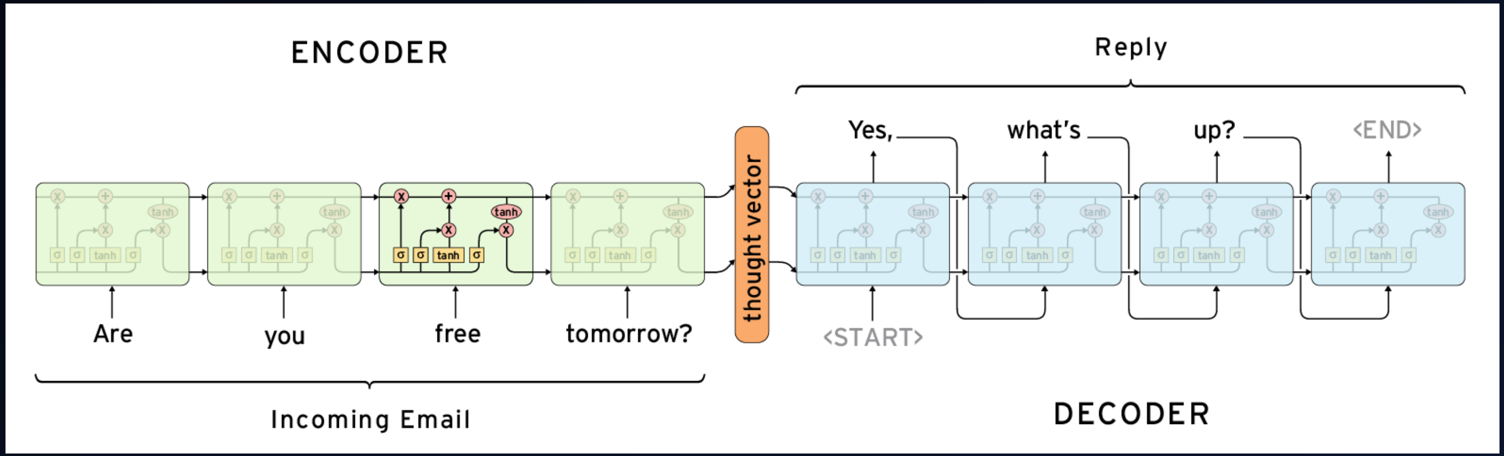
Outline

2. Recent History of Deep Learning
    1. 2012: The Year of the Deep Net
    2. 2013: The Year of Atari
    3. 2016: The Year of AlphaGo
    4. Deep Learning in Everyday Technology
    5. Frontiers in Deep Learning



Outline

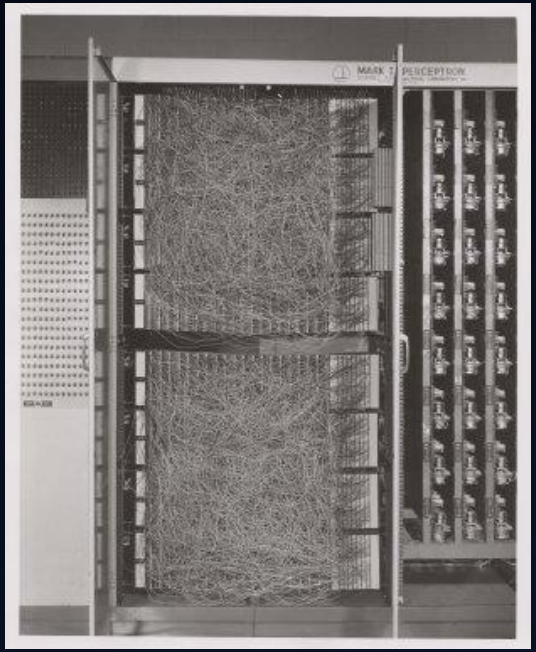
3. Selected Modern Deep Learning Approaches
    1. Residual Networks
    2. Sequence to Sequence Models
    3. Attention Mechanisms
    4. GANs

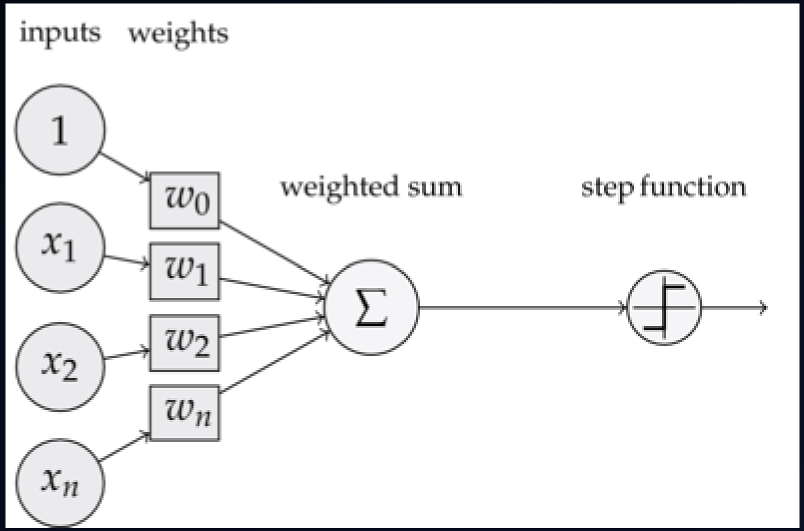


Fundamentals of Neural Nets

Early AI: the Perceptron

- Late 1950s
- Implemented on custom hardware
- Linear classification
- Awkward training scheme based on error

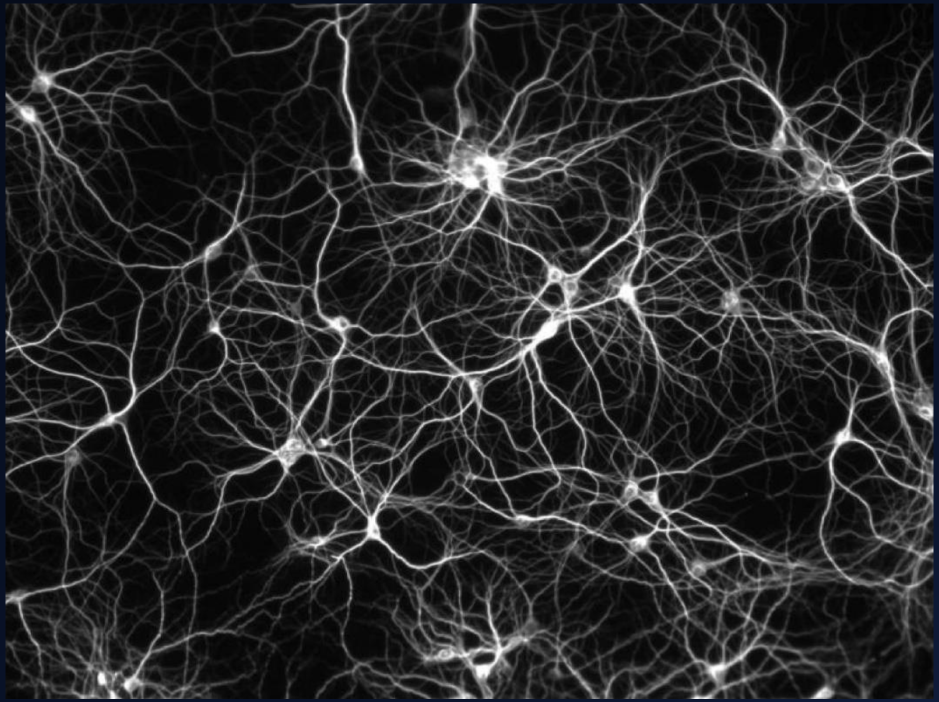




$$y = h(w^T x + b) = \begin{cases} 1 & w^T x + b > 0 \\ 0 & otherwise \end{cases}$$

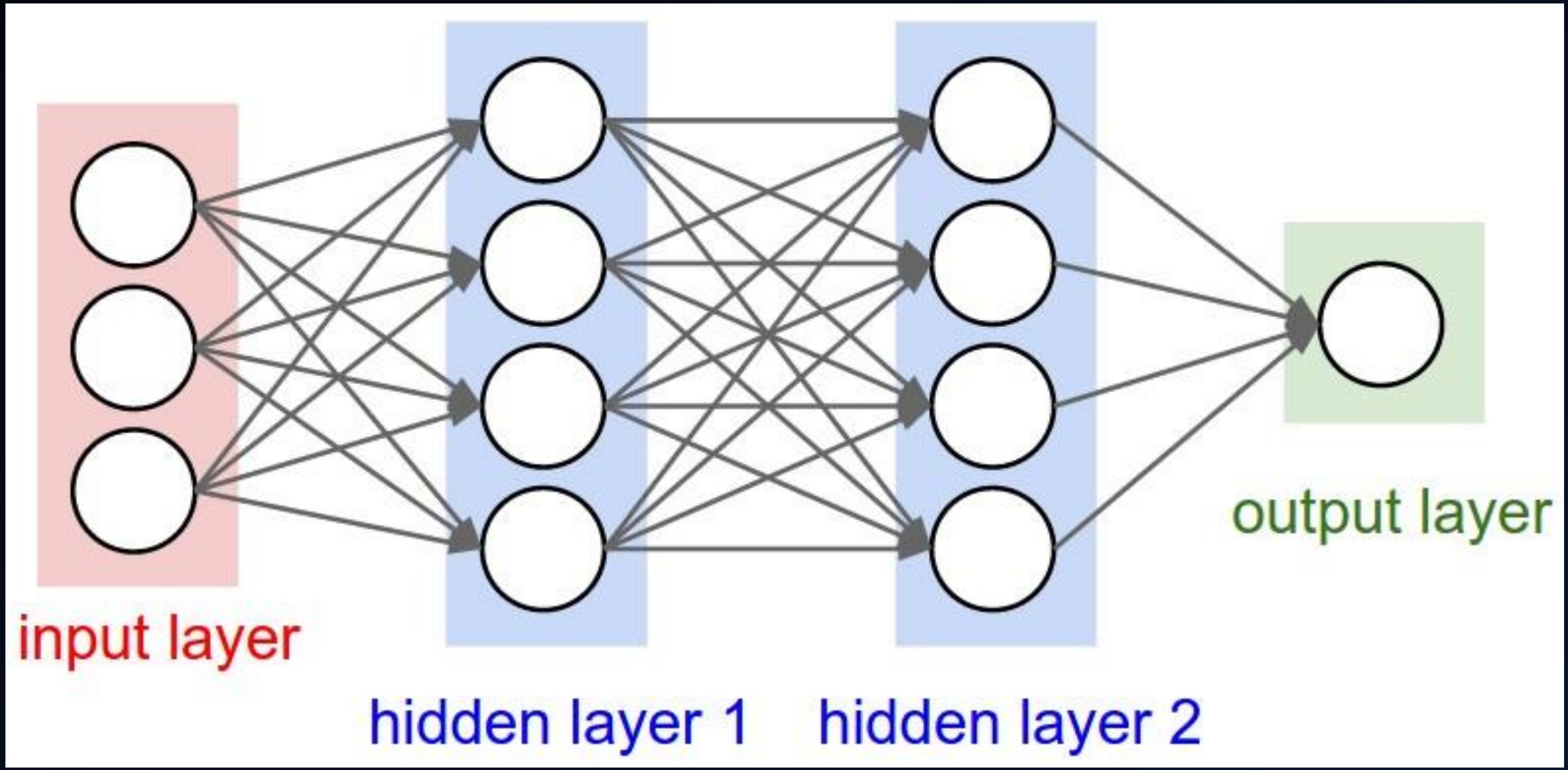
The Perceptron

- Interpret as checking for presence of a pattern and issuing binary response
- Inspired by neurons in brain
- **Can only learn very simple functions: not even expressive enough to learn XOR**



Early Neural Networks

- Neural networks emerged as differentiable, **multi-layered** perceptrons



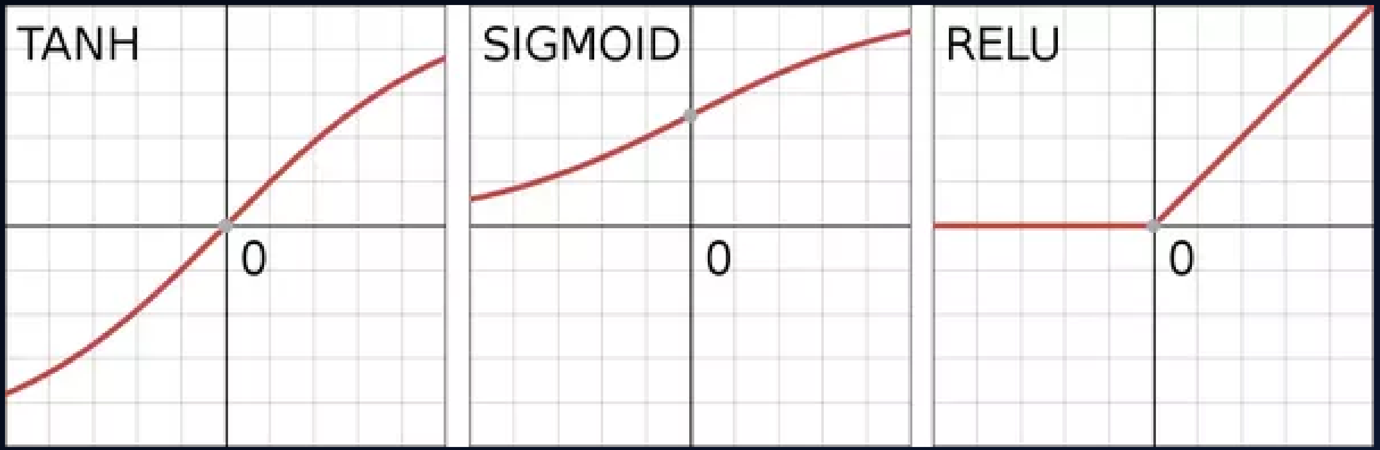
Neural Networks

- A model composed of several "layers," where the ones between input and output are called "hidden"
- Standard basic layer:
  - Linearity, $z = Wx + b$
  - Followed by elementwise nonlinearity, $y = f(z)$
- With L layers:

$$a^0 = x$$
$$a^j = f^j\left(W^j a^{j-1} + b^j\right)$$
$$y = a^L$$

Neural Networks

- Common nonlinearities (**activation functions**):
  - Sigmoid $\sigma(x) = \frac{1}{1+\mathrm{e}^{-x}}$
  - Tanh $\tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$
  - Relu $\mathrm{relu}(x) = \max(0, x)$



Neural Networks

- Common output layers:
  - No activation (good for regression)

  - Softmax activation (good for classification): with z the output of the last linear transform,

$$y_i = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)}$$

This gives a **probability distribution of dimension K** as an output

Neural Networks

- A neural network is characterized by its **architecture** and its **parameters**
- Architecture (fixed)
  - A term used to describe design choices, like
    - Number of layers
    - Size of output from each layer
    - Nonlinearities
    - Alternative connection patterns
- Parameters (learned)
  - "weights" W and "biases" b at each layer, and possibly others

Neural Networks

- What do you do with them?
  - Classification
  - Prediction / forecasting
  - Function approximation
  - Decision-making of any kind, really
- How do you train them (learn the parameters)?
  - Most powerful tool of all: **gradient descent**

Training Neural Networks: Task and Loss Function

- Suppose that f_θ is your neural net, and $\theta = (W^1, \dots, W^L, b^1, \dots, b^L)$ is your set of parameters
- Task is classification
- You have a dataset of pairs $D = \{(x^i, y^i)\}_{i=1,\dots,N}$
- Form a per-datum loss function:
$$L(\theta, x, y) = d(f_\theta(x), y)$$
- Form a **dataset average loss function:**
$$\mathcal{L}(\theta, D) = \frac{1}{N} \sum_{i=1}^{N} L(\theta, x^i, y^i)$$

Training Neural Networks: Loss Function

- Suppose $y \in \{1, \ldots, K\}$, and the last layer of f_θ is softmax
- Choose differentiable per-datum loss function, like **cross-entropy loss**:

$$L(\theta, x, y) = -\sum_{i=1}^{K} \mathbb{I}[y = 1] \log P(y = i | x, \theta)$$

where $P(y = i | x, \theta) = [f_\theta(x)]_i$

- Dataset average loss is now differentiable with respect to θ

Training Neural Networks: Gradient Descent and SGD

- Gradient descent:

$$g = \nabla_\theta \mathcal{L}(\theta, D)$$
$$\theta \leftarrow \theta - \alpha g$$

- Problem: expensive to compute gradient of loss over whole dataset
- Solution: **stochastic gradient descent (SGD)**

$$g \approx \sum_{(x,y)\in B} L(\theta, x, y)$$

where B is randomly-sampled **minibatch**

Training Neural Networks: Backpropagation

- In the 1970s people rediscovered the wheel and gave it a new name
- Algorithm for computing gradients of neural network outputs with respect to parameters is called **backprop**

Training Neural Networks: Backpropagation

Chain rule:

$$\frac{\partial y}{\partial W^j} = \frac{\partial y}{\partial a^L}\frac{\partial a^L}{\partial a^{L-1}} \cdots \frac{\partial a^{j+1}}{\partial a^j}\frac{\partial a^j}{\partial W^j}$$

Define

$$\delta^j = \frac{\partial y}{\partial a^L}\frac{\partial a^L}{\partial a^{L-1}} \cdots \frac{\partial a^{j+1}}{\partial a^j}$$

If you have δ^L, easy to perform "backprop":

For $j = L-1, L-2, \ldots, 1$:

$$\delta^j = \delta^{j+1}\frac{\partial a^{j+1}}{\partial a^j}$$

$$\frac{\partial y}{\partial W^j} = \delta^j \frac{\partial a^j}{\partial W^j}$$

Training Neural Networks: SGD Variants

- SGD stops updating when gradient is small---this should happen at loss function minima, but also happens in **valleys**
- Momentum methods break out of valleys

$$v \leftarrow \gamma v + g$$
$$\theta \leftarrow \theta - \alpha v$$

- Adaptive learning rate algorithms can also help, like RMSprop:

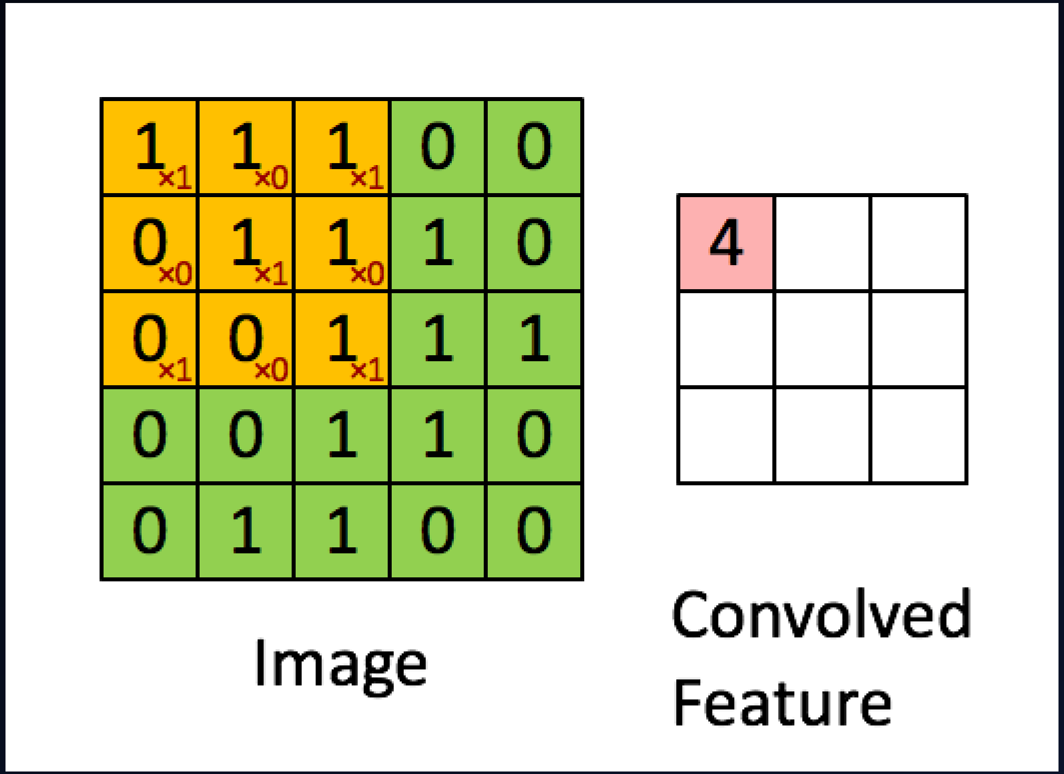
$$v \leftarrow \gamma v + (1 - \gamma) g^2$$
$$\theta \leftarrow \theta - \alpha \frac{g}{\sqrt{v} + \epsilon}$$

Special Neural Net Architectures: Convolutional Layers

- Layers we described previously are *fully-connected*
- That is bad for parameter reuse
- How can we encode **invariance** to translation?

Special Neural Net Architectures: Convolutional Layers

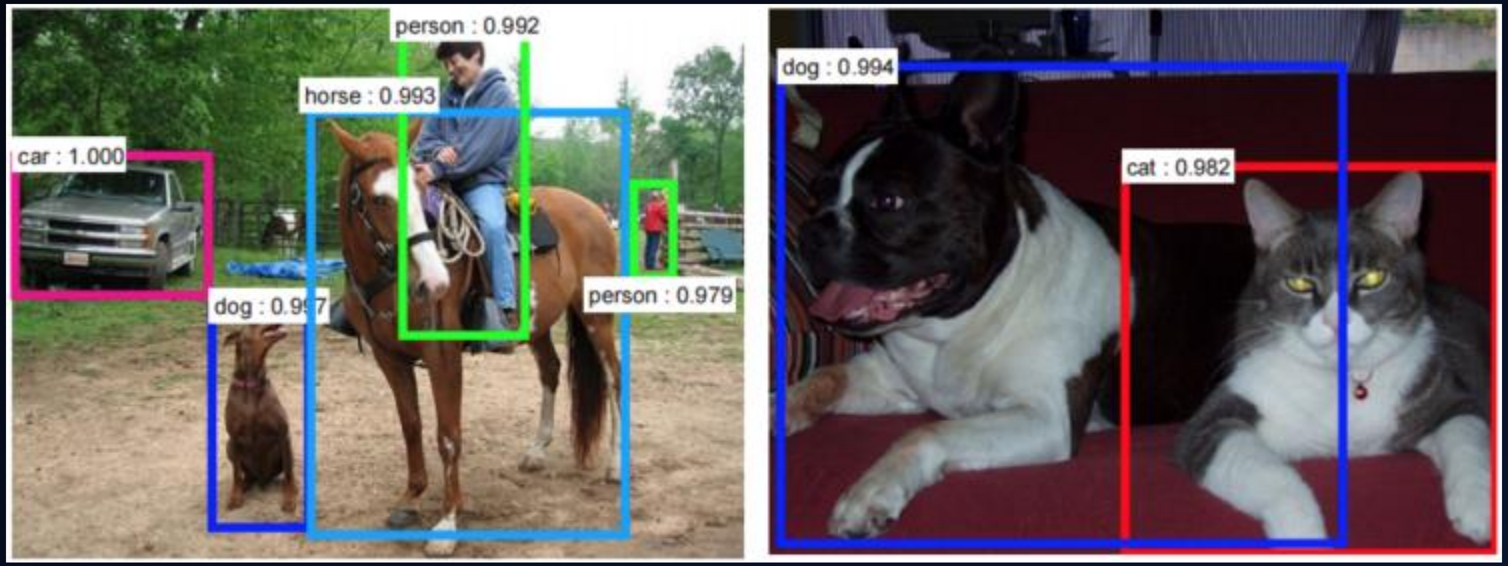
- Conv layers *convolve* an input with some feature kernel to produce a *response map*



http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

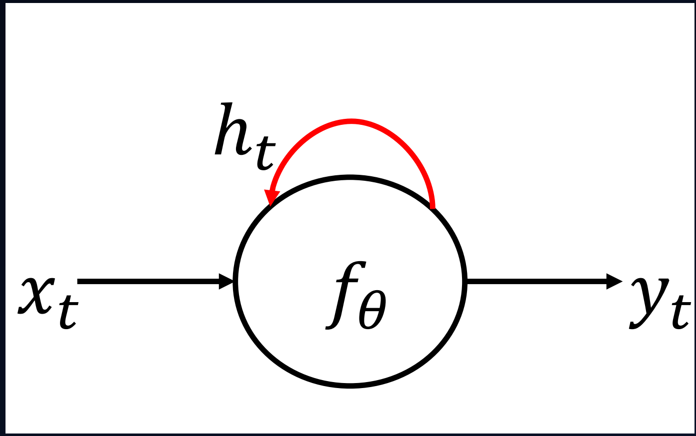
Special Neural Net Architectures: Convolutional Layers

- Conv layers are great for parameter reuse
- Far fewer parameters than fully-connected layers
- Excellent at vision---can learn to understand contents of images from raw pixels with no other feature extraction!



Special Neural Net Architectures: Recurrent Nets

- Some tasks require **memory** to solve
  - Time series prediction
  - Language modeling
- Recurrent Neural Networks (RNNs) **have memory in hidden state**



Special Neural Net Architectures: Recurrent Nets

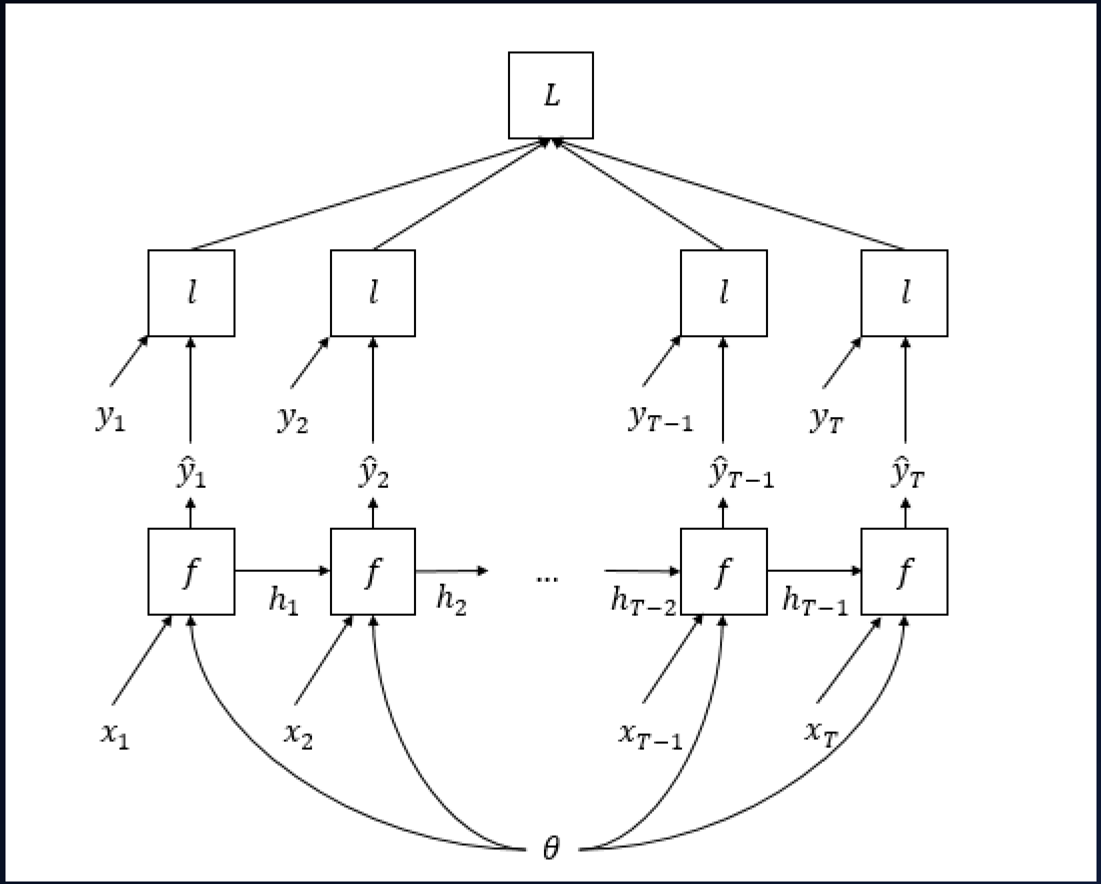
- "Vanilla" RNN layer:

$$h_t = \sigma(Wx_t + Rh_{t-1} + b)$$

- Drop in as a replacement to a fully-connected layer
- Can stack RNN layers in the same way as standard layers
- Requires special training: **backprop through time (BPTT)**

Special Neural Net Architectures: Recurrent Nets

- Backprop Through Time:
  - Loss function on **sequences:** per-sequence loss is a sum or average of per-time step losses
  - $(x_1, \ldots, x_T, y_1, \ldots, y_T)$
  - Need dataset of sequences
  - If sequences are too long, **truncate** backprop to some horizon



Vanishing Gradients in Vanilla RNNs

- Ability to learn long time dependencies requires **good gradient flow through network**
- Problem:

$$\frac{\partial h_t}{\partial h_k} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \dots \frac{\partial h_{k+1}}{\partial h_k}$$

$$\frac{\partial h_t}{\partial h_{t-1}} \propto R$$

$$\frac{\partial h_t}{\partial h_k} \propto R^{t-k}$$

- $\lambda(R)$ large → Gradient explodes!
- $\lambda(R)$ small → Gradient vanishes...

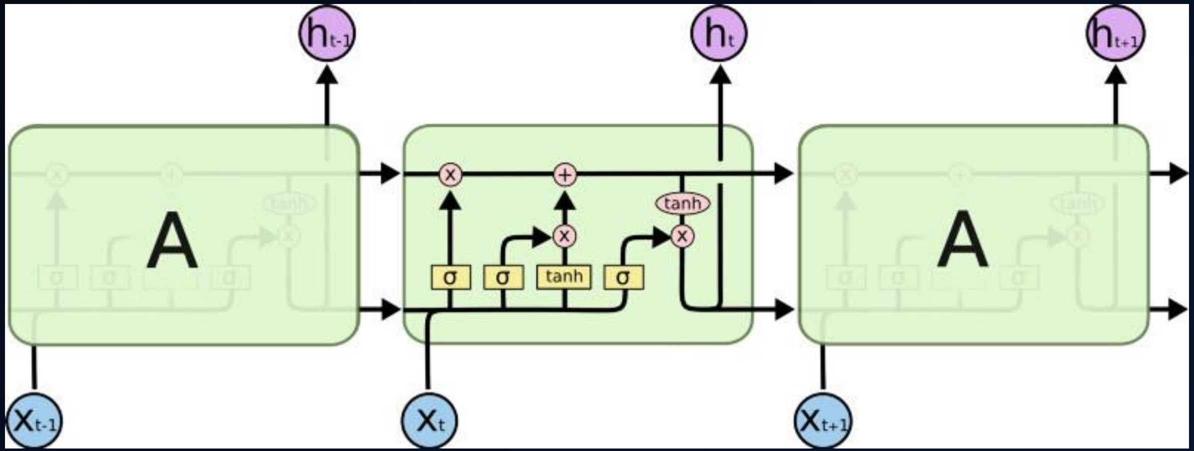
Solution to Vanishing Gradients: LSTMs

- Long Short-Term Memory (LSTM) Networks
  - Have two hidden states
  - One of which is *additive* instead of multiplicative, making gradient flow easier:

$$c_t = c_{t-1} + z_t$$

(in truth, also have forget gates and input gates – omitted here for simplicity)

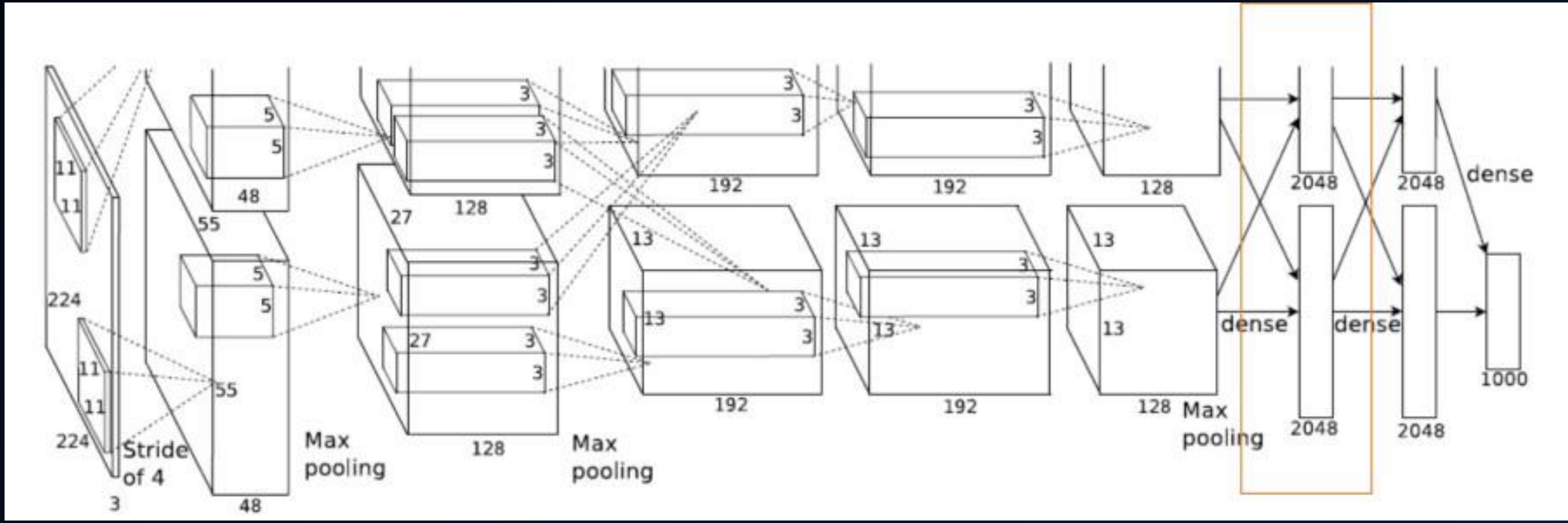
  - Allow learning of dependencies over **hundreds of time steps**!



Recent History of Deep Learning

2012 – The Year of the Deep Net

- After years of many incremental advances in neural nets (development of ReLU and advanced training techniques), a deep neural network trained end-to-end by SGD won ImageNet contest
- AlexNet (Krizhevsky et al.):



2012 – The Year of the Deep Net

- ImageNet Large Scale Visual Recognition Contest (ILSVRC)
  - ~1000 high-resolution images in each of 1000 categories
- AlexNet substantially improved SOTA using no specialized algorithms:

	Top-1 Error Rate	Top-5 Error Rate
Sparse Coding	47.1	28.2
SIFT+FVs	45.7	25.7
AlexNet	**37.5**	**17.0**

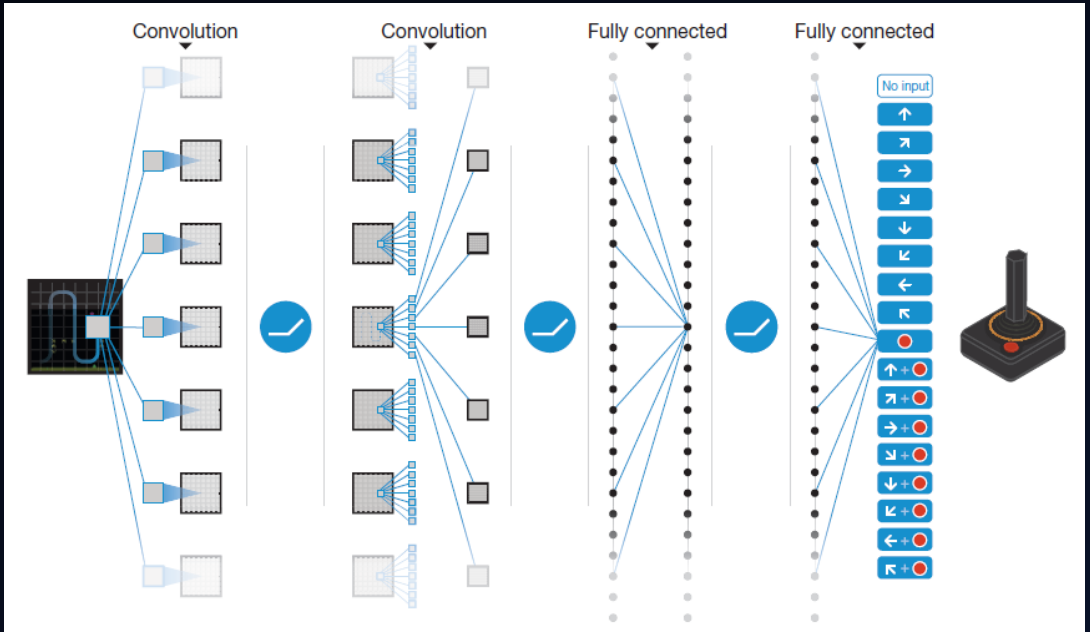
- Interest in deep convolutional networks grew substantially

2012 – The Year of the Deep Net

- Le et al. (folks at Google and Stanford) trained a deep 9-layer **autoencoder** on a huge dataset from YouTube
- Autoencoder loss: $d\left(x, f_\theta(x)\right)$
- *Unsupervised* learning (just need data, no labels)
- Learned neurons that detected human faces, bodies, cat faces, etc.

2013 – The Year of Atari

- A convolutional neural network was trained to play Atari games by **reinforcement learning (RL)**
- Mnih et al. introduced **Deep Q-Learning**, first major breakthrough in RL + deep learning



Deep Q-Learning

- Goal is to learn **optimal action-value function** $Q^*(s,a)$
- Bellman equation (dynamic programming!):

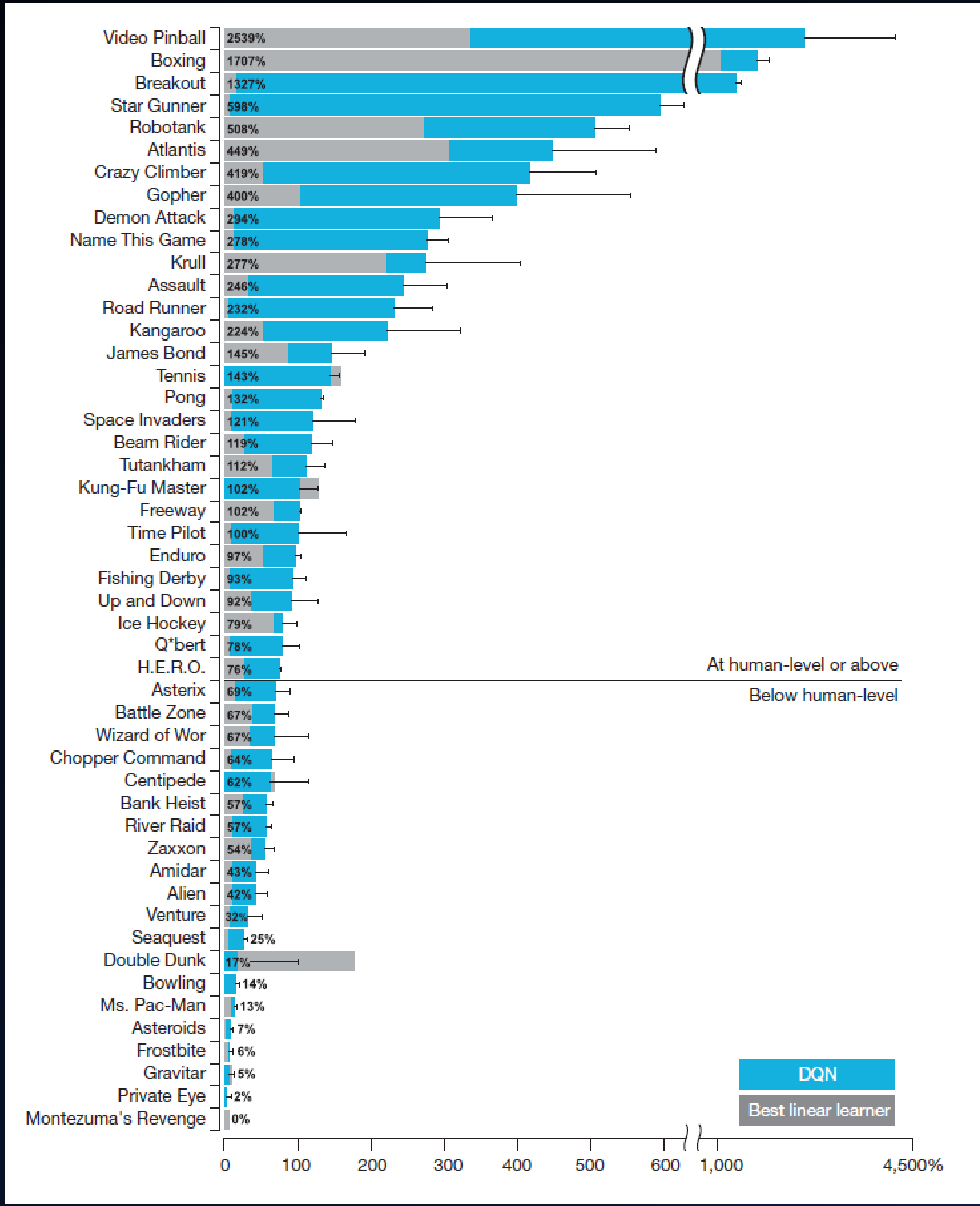
$$Q^*(s,a) = \mathbb{E}_{s'}[r(s,a) + \gamma \max_{a'} Q^*(s',a')]$$

- Learn approximator Q_θ by minimizing **squared Bellman error**

$$L(\theta, D) = \mathbb{E}_{s,a,r,s' \sim D}\left[\left(Q_\theta(s,a) - \left(r + \gamma \max_{a'} Q_\theta(s',a')\right)\right)^2\right]$$

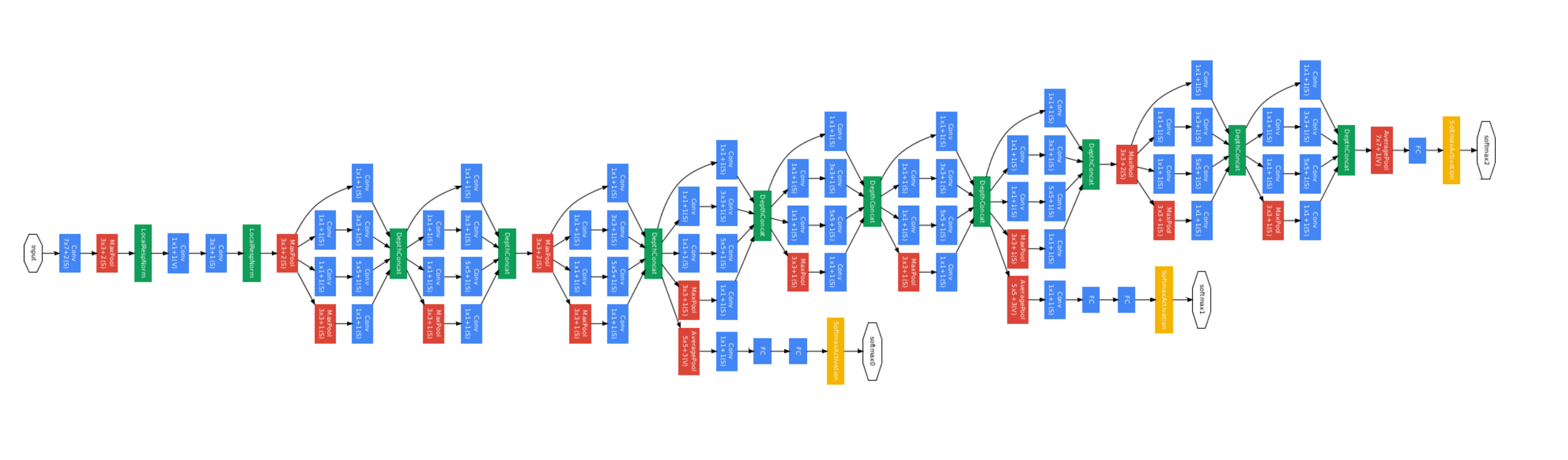
Deep Q-Learning

- Achieved **superhuman performance** on several Atari games!
- Many improvements followed
- Shown right: results from 2015 Nature paper on DQN



2014 – GoogLeNet

- GoogLeNet (Szegedy et al.): 22-layer deep convolutional network gets **human-level performance on ImageNet**



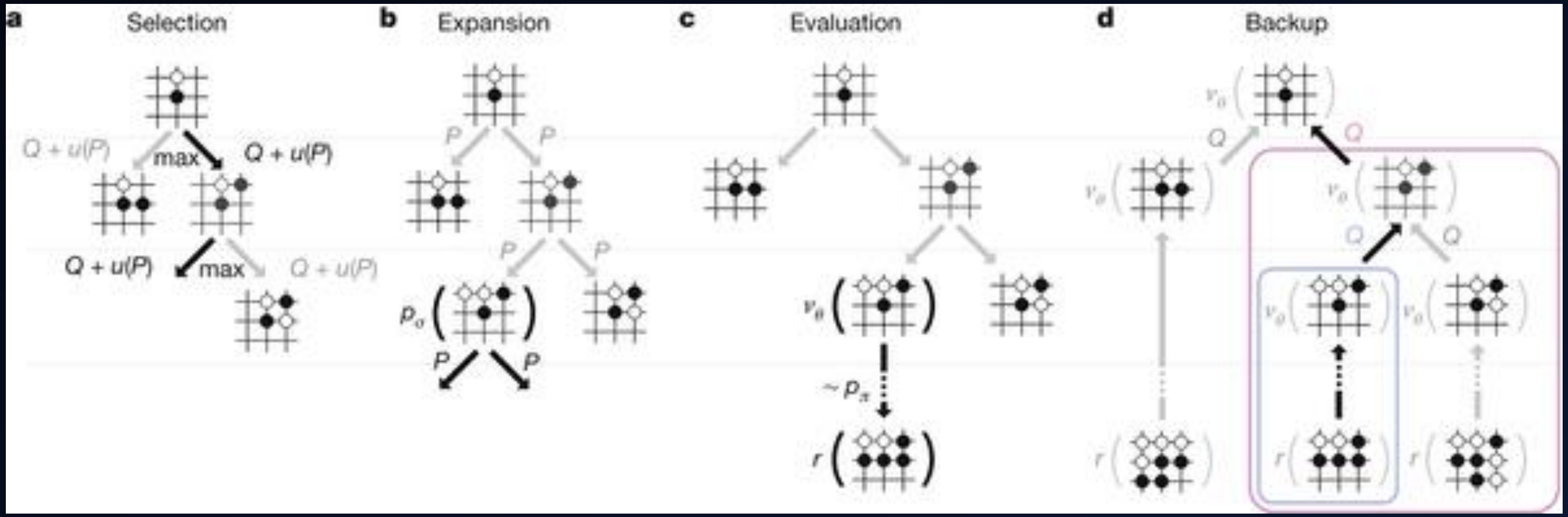
- Top-5 error rate of **6.67%**
- Every blue block is a conv layer!

2016 – The Year of AlphaGo

- Go is a game with simple rules and extremely complex play
- Impossible to solve with minimax AI / hard-coded rules

2016 – The Year of AlphaGo

- Google DeepMind developed **AlphaGo**, a deep learning AI for Go
- Used 13-layer networks to select optimal moves and evaluate positions
- **Monte Carlo Tree Search** to help AlphaGo think ahead

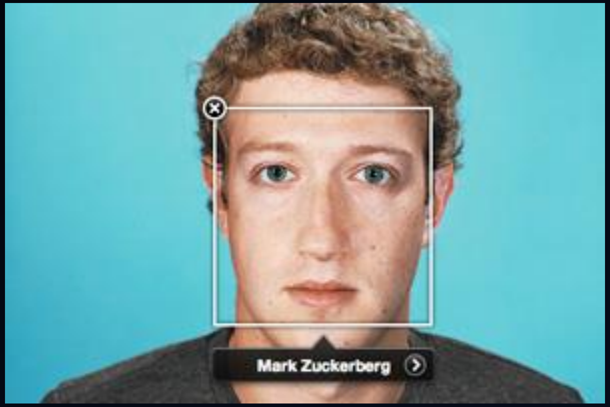


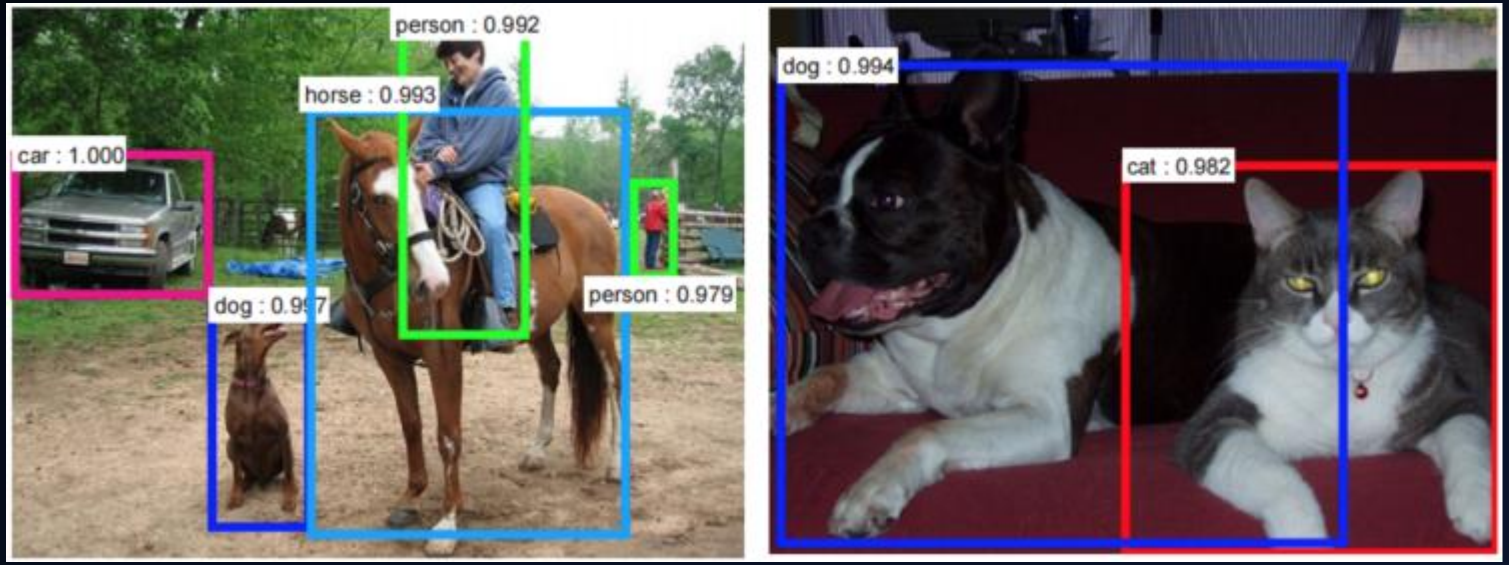
2016 – The Year of AlphaGo

- Defeated human grand master, Lee Sedol, 4-1!

Deep Learning in Everyday Technology

- Face recognition and object detection: convolutional nets

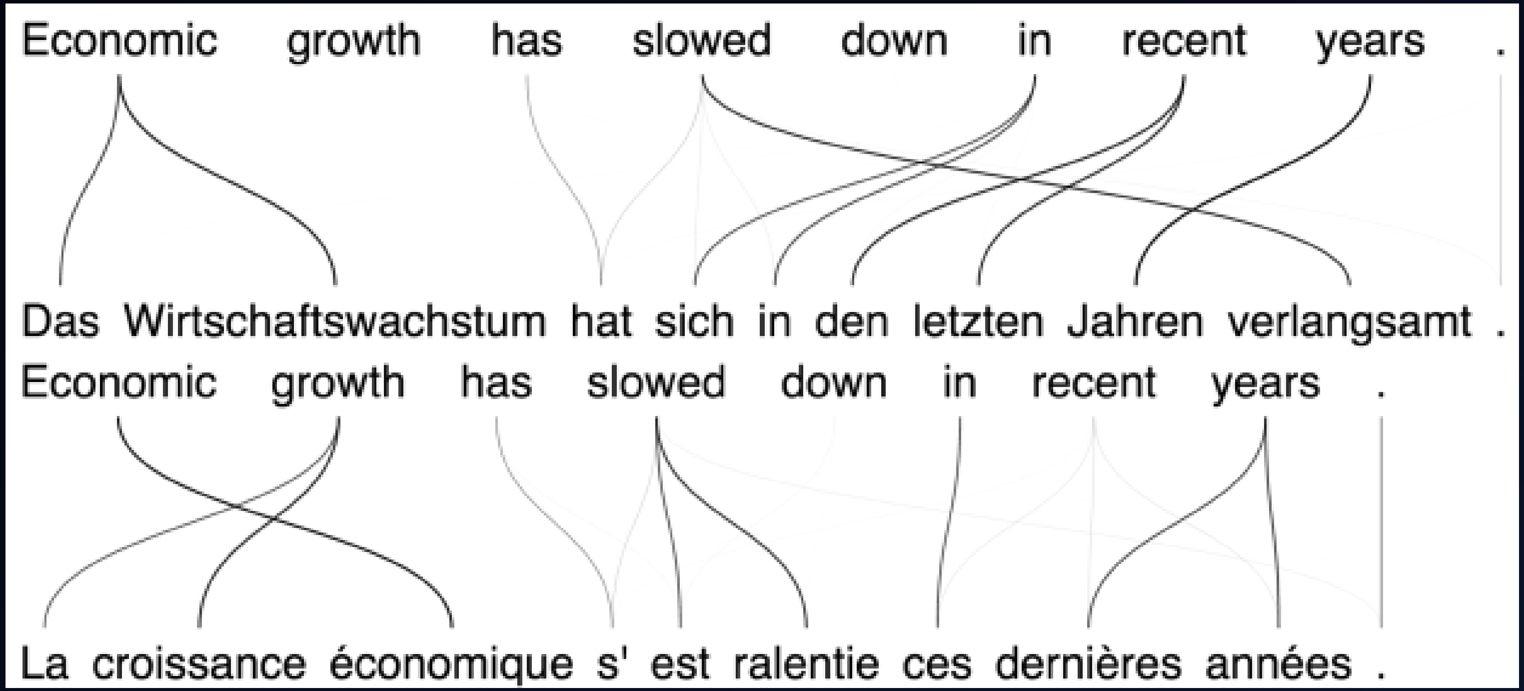




- Near future: vision for self-driving cars!

Deep Learning in Everyday Technology

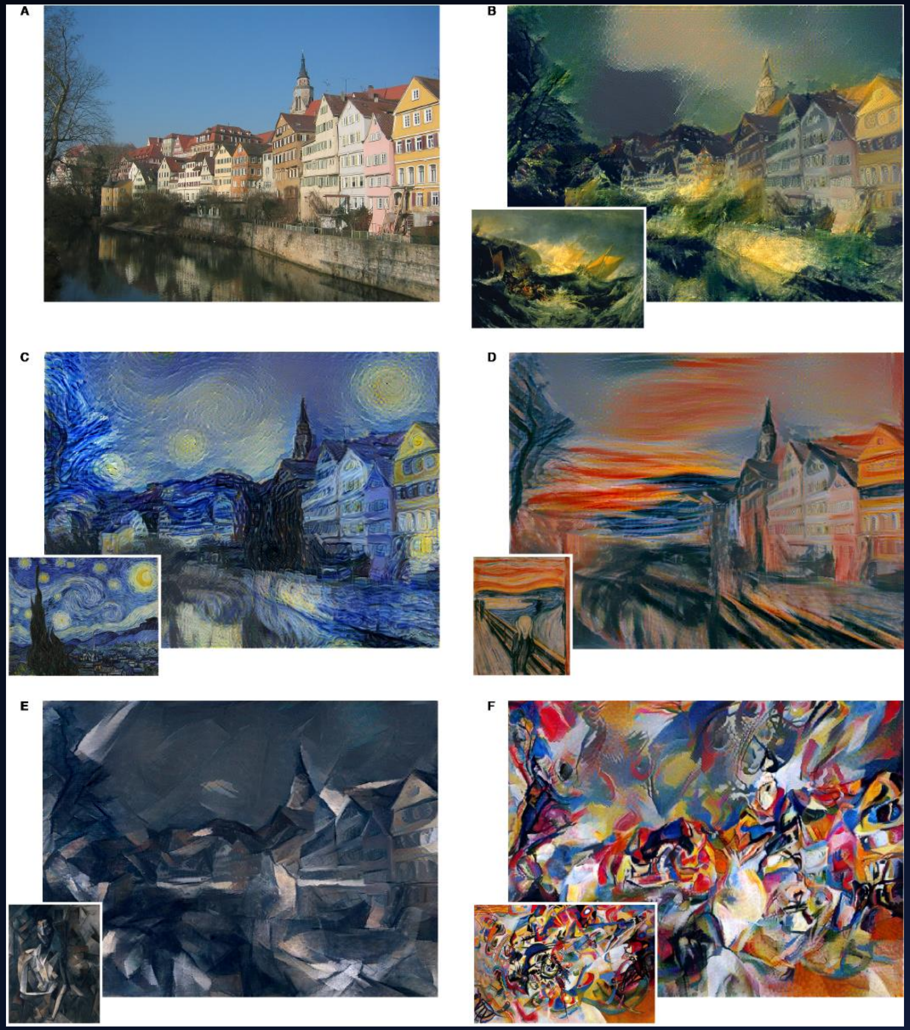
- Neural machine translation



https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-3/

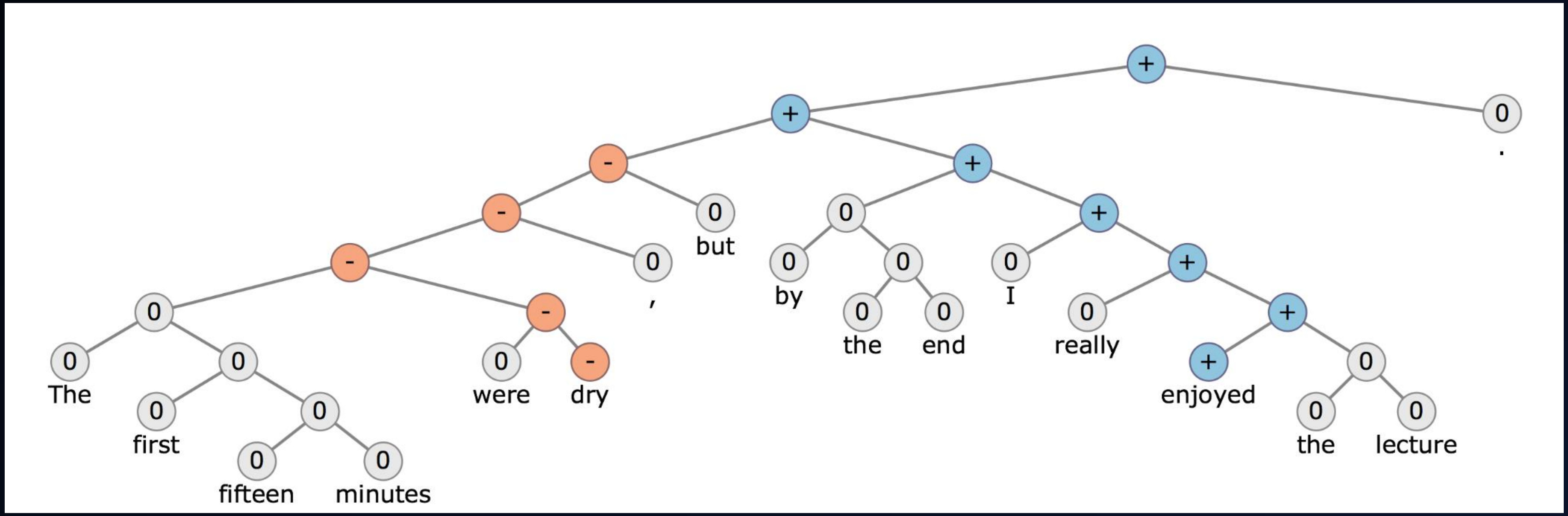
Deep Learning in Everyday Technology

- Neural style transfer (Gatys et al.)



Deep Learning in Everyday Technology

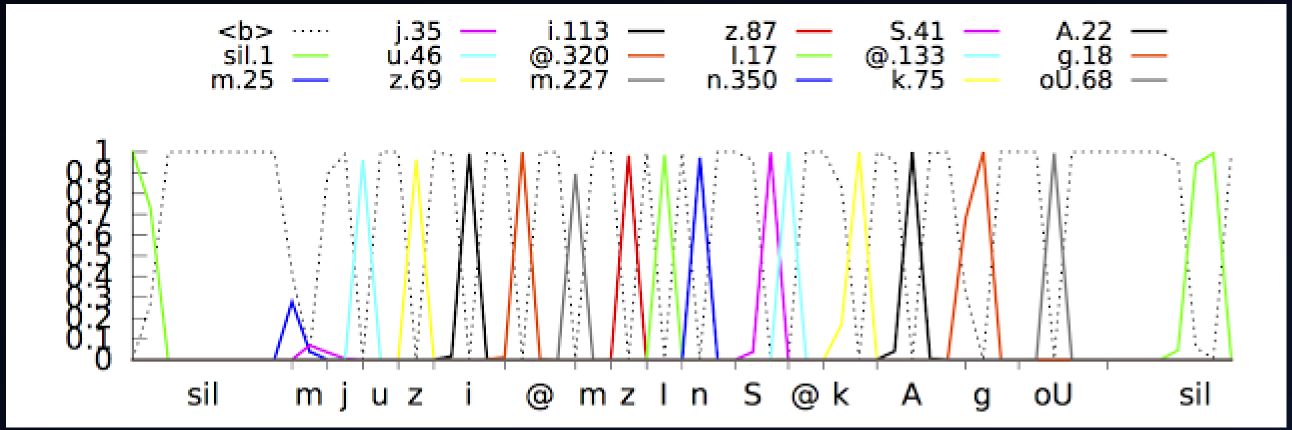
- Data analytics: sentiment analysis



(image from Stanford course on deep learning for NLP)

Deep Learning in Everyday Technology

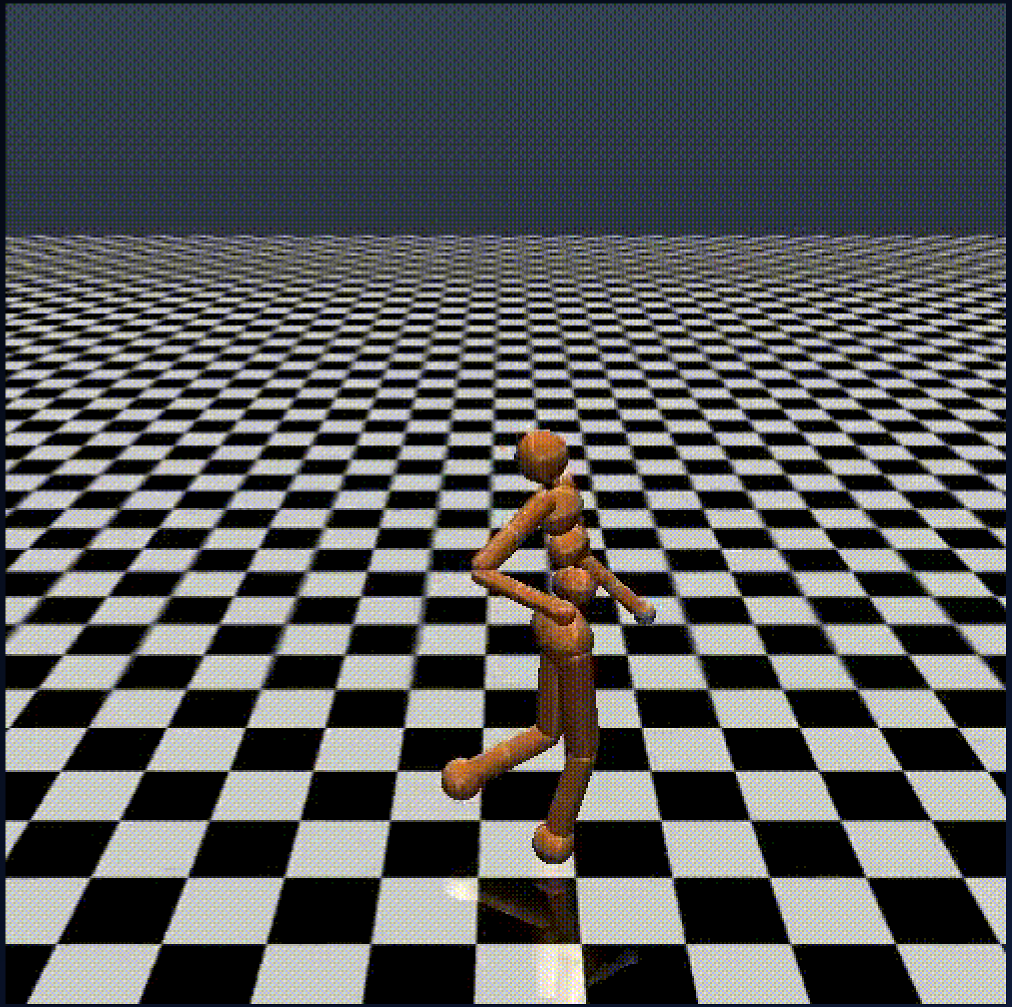
- Automatic speech recognition
  - Core technology for Google’s speech recognition is based on deep learning!



https://research.googleblog.com/2015/09/google-voice-search-faster-and-more.html

Frontiers in Deep Learning

- Speech synthesis (WaveNet)
- Robotics
- One-shot learning / Metalearning
  - Learning to Learn by Gradient Descent
  - RL^2
  - MAML
- Chatbots



Frontiers in Deep Learning: Chatbots

Human: *what is the purpose of life ?*
Machine: *to serve the greater good .*
Human: *what is the purpose of living ?*
Machine: *to live forever .*
Human: *what is the purpose of existence ?*
Machine: *to find out what happens when we get to the planet earth .*
Human: ***what is the purpose of emotions ?***
Machine: ***i don ’t know .***

(Vinyals and Le, 2015)

Selected Modern Deep Learning Approaches

Residual Networks

- He et al., 2015, used **residual networks** to achieve 3.57% top-5 error on ImageNet LSVRC

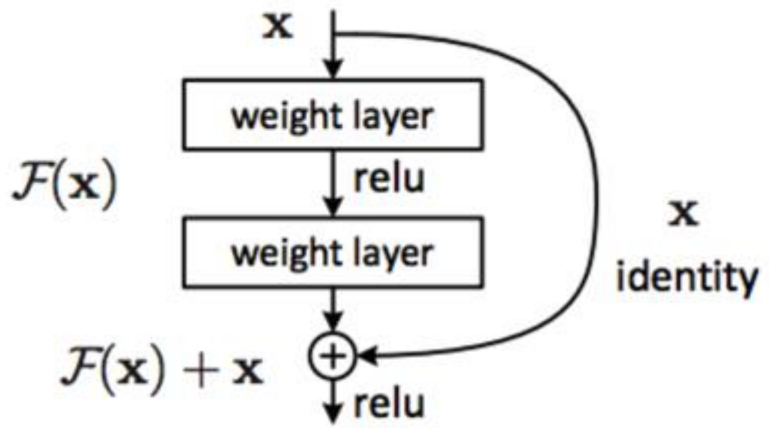
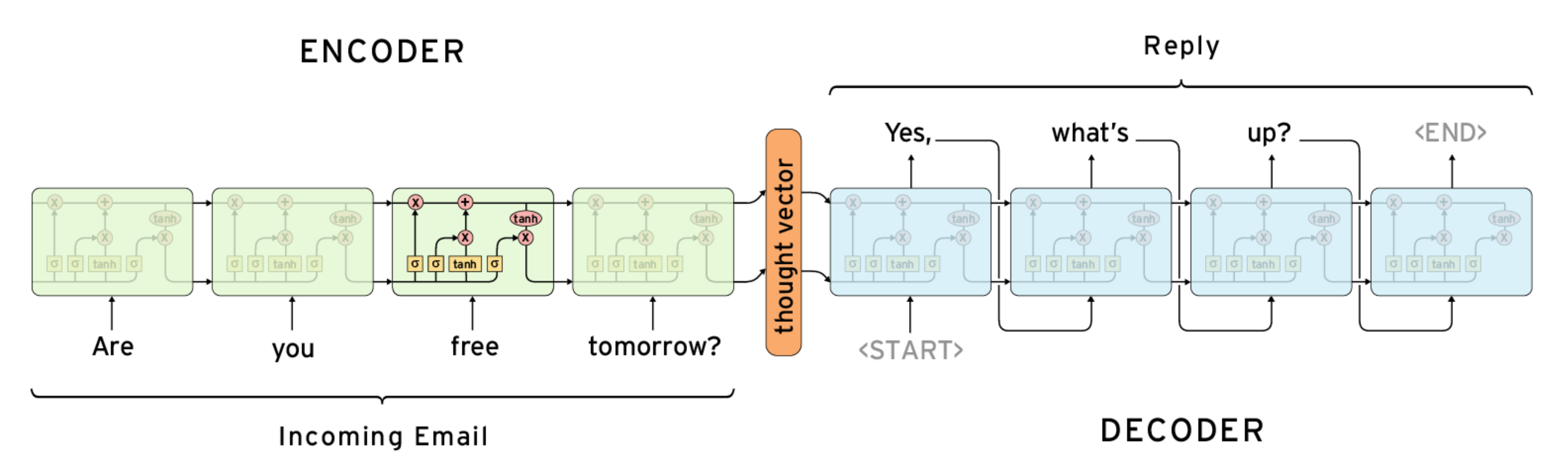


Figure 2. Residual learning: a building block.

Sequence to Sequence Models

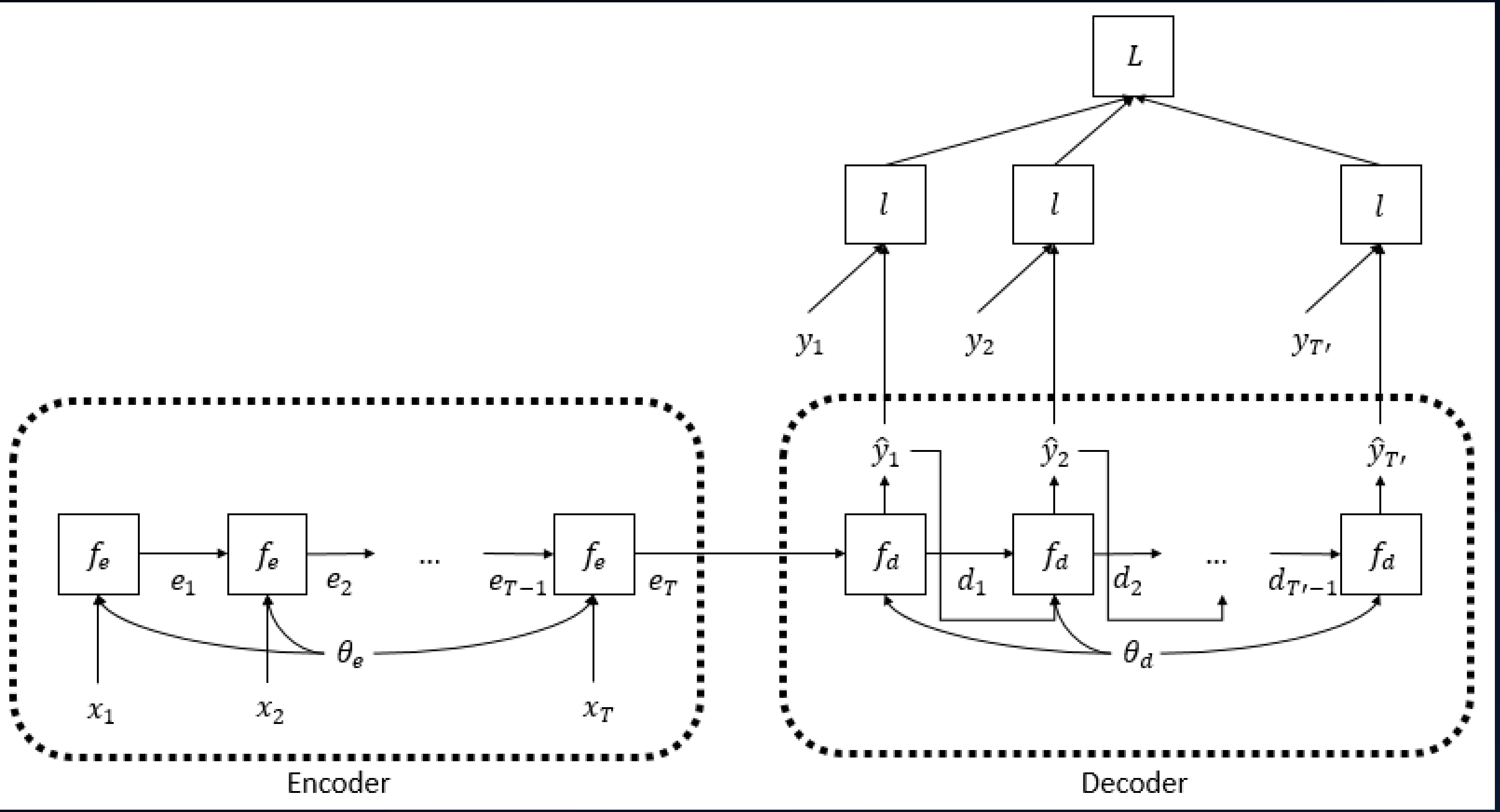
- Cho et al 2014, Sutskever et al 2014
- Key for neural machine translation



http://suriyadeepan.github.io/2016-12-31-practical-seq2seq/

Sequence to Sequence Models

- How do you train them? Per-(input sequence, output sequence) pair loss function



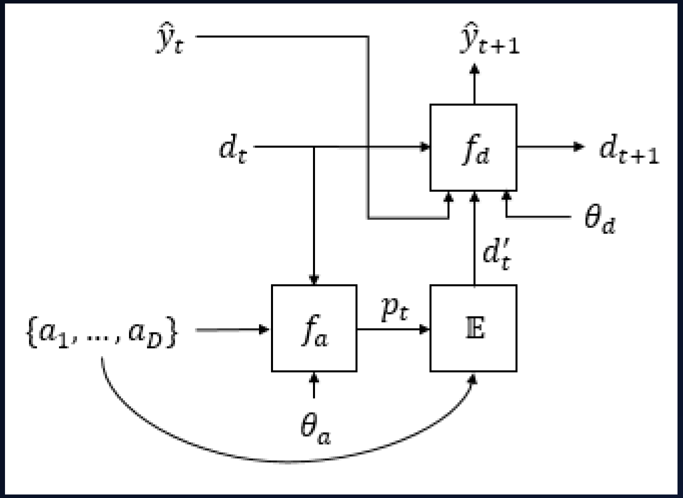
Attention Mechanisms

- Help sequence to sequence models by letting them focus on specific input (Bahdanau et al, 2015)

- For each encoder hidden state e_j, produce a weight w_j, and use

$$d' = \sum_j w_j e_j$$

as auxiliary input



Generative Adversarial Networks (GANs)

- Goodfellow et al., 2014
- Goal is to learn a **generative model**
- IE, put noise in, get sample from data distribution out
- Learn in **adversarial game** with two players
  - Generator: trying to produce outputs like data
  - Discriminator: trying to tell if input is real or from generator

$$\min_{\mathrm{G}} \max_{\mathrm{D}} \mathbb{E}_{\mathrm{x}\sim\mathrm{p_x}}[\log D(x)] + \mathbb{E}_{z\sim p_z}\left[\log\left(1 - D\left(G(z)\right)\right)\right]$$

Generative Adversarial Networks (GANs)

- Can generate almost-natural looking images, but things get weird sometimes

- Hard to train, but this is an area of active research!

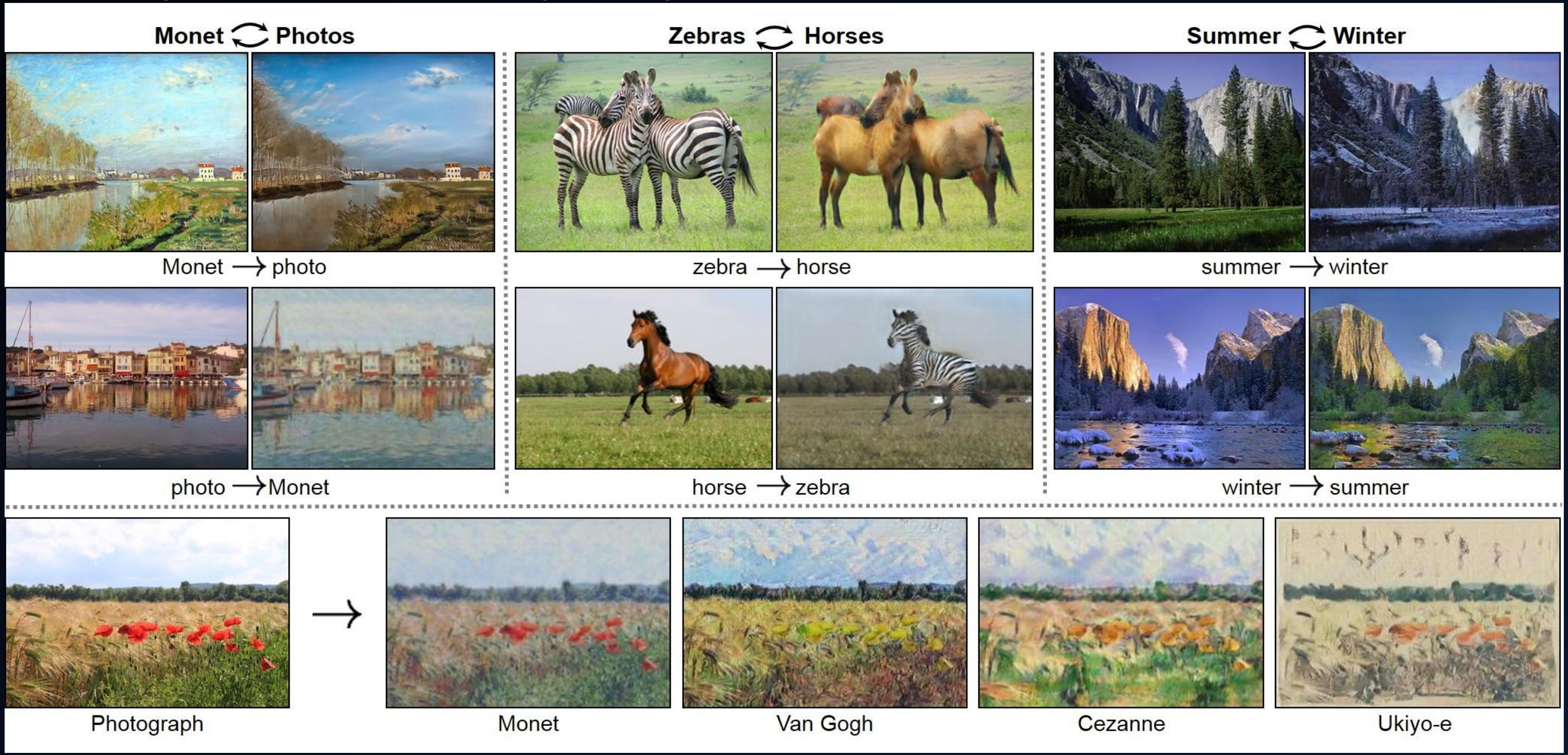
https://github.com/skaae/torch-gan

Generative Adversarial Networks (GANs)

- Many variants, including generating images from **natural language text** (Reed et al. 2016)

Generative Adversarial Networks (GANs)

- GAN + Cycle consistency = CycleGAN (Zhu and Park et al, 2017)



https://junyanz.github.io/CycleGAN/

And so much more!

- This talk only scratched the surface
- If you want, it's easy to get started learning...
  - Stanford Deep Learning Tutorial (http://deeplearning.stanford.edu/)
  - Andrej Karpathy Blog (http://karpathy.github.io/)
  - Deep Learning Textbook (http://www.deeplearningbook.org/)
  - Awesome Deep Learning Resource List (https://github.com/ChristosChristofidis/awesome-deep-learning)

And so much more!

- ...and easy to get started hacking!
  - Tensorflow (https://www.tensorflow.org/)
  - Keras (https://keras.io/)
  - PyTorch (http://pytorch.org/)
  - Caffe (https://caffe2.ai/)
  - Chainer (https://chainer.org/)