Neural Networks and Deep Learning

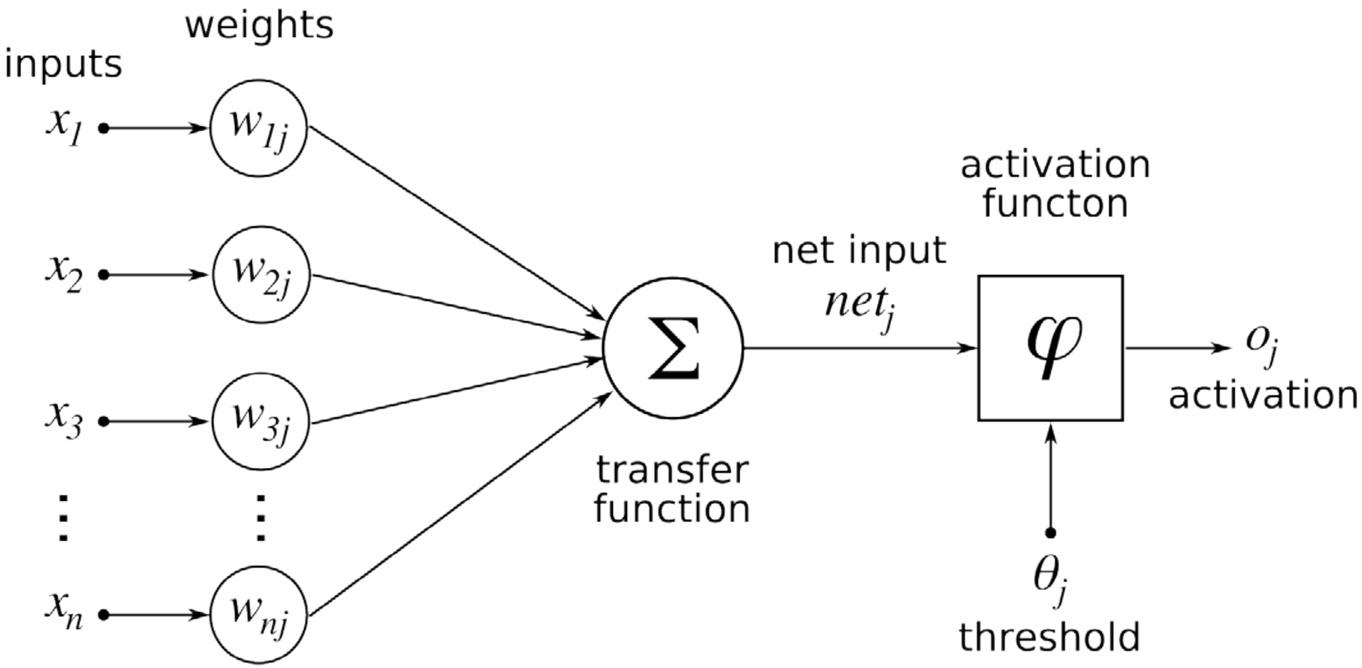
Outline

- Neural Networks
- Deep Learning: A Short Introduction

Neural Networks

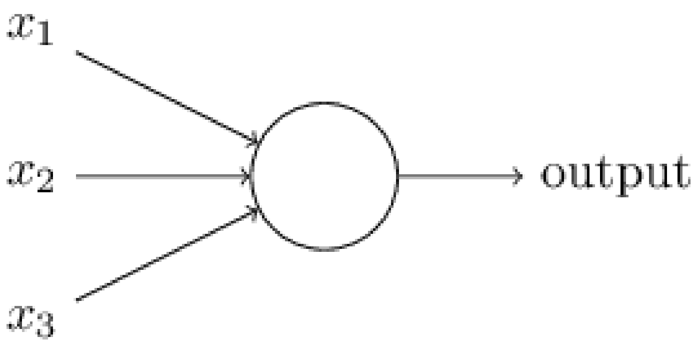
Neural Network for Classification

- Started by psychologists and neurobiologists to develop and test computational analogues of neurons
- A neural network: A set of connected input/output units where each connection has a **weight** associated with it
  - During the learning phase, the **network learns from the training examples by adjusting the weights** so as to be able to predict the correct class label
- Learning without task-specific programming



Artificial Neural Networks as an analogy of Biological Neural Networks

Perceptron: Predecessor of a Neural Network



$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

- The perceptron algorithm: Invented in 1957 by Frank Rosenblatt
- Input: An *n*-dimensional input vector **x** (with *n* variables)
- Output: 1 or 0 depending on if the weighted sum passes a threshold
- Perceptron: A device that makes decisions by weighing up evidence
- Often written in the vector form, using bias (*b*) instead of threshold, as

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

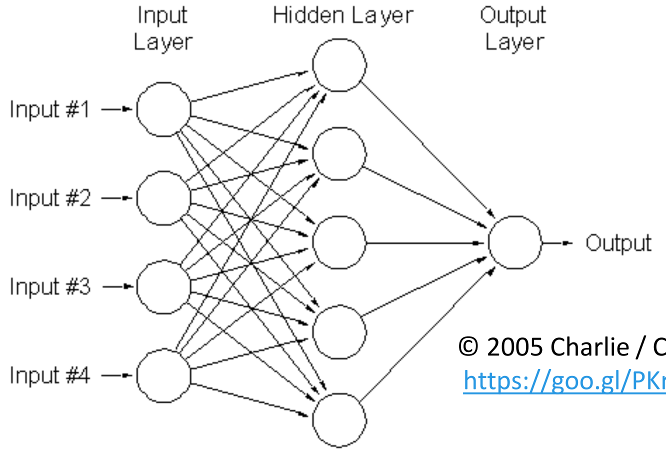
Bias: A measure of how easy it is to get the perceptron to output a 1

Sigmoid Neurons

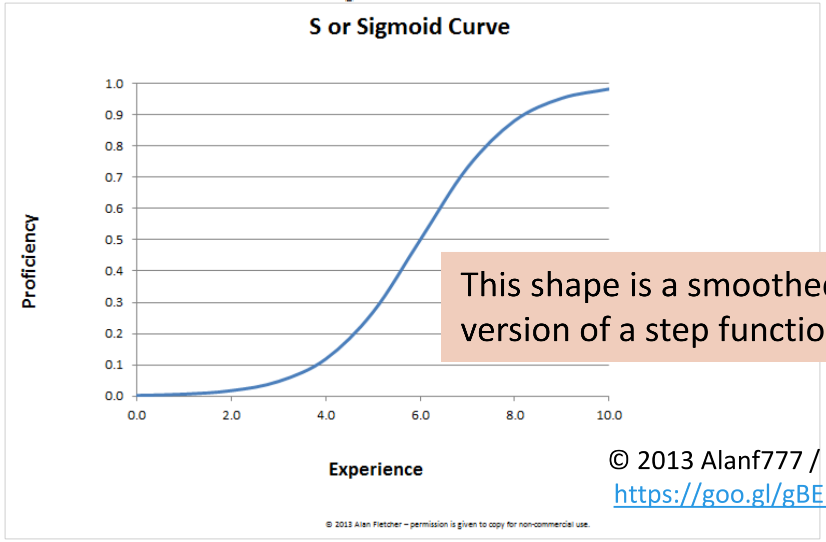
- A many-layer network of perceptrons can engage in sophisticated decision making
- Instead of assigning weights of the edges by a person, we can devise *learning algorithms* that can automatically tune the weights and biases of a network of artificial neurons
- Use sigmoid neuron instead of perceptron: Output is not 0/1 but a *sigmoid* function: σ(*w*•*x* + b) , i.e., $\frac{1}{1+exp(-\sum_j w_j x_j - b)}$
- The smoothness of σ means that small changes in the Δw_j weights and in the Δb bias will produce a small change Δ_{output} in the output from the neuron

$$\Delta \text{output} \approx \sum_j \frac{\partial\, \text{output}}{\partial w_j} \Delta w_j + \frac{\partial\, \text{output}}{\partial b} \Delta b$$

i.e., Δ_{output} is a *linear function* of the changes Δw_j and Δb



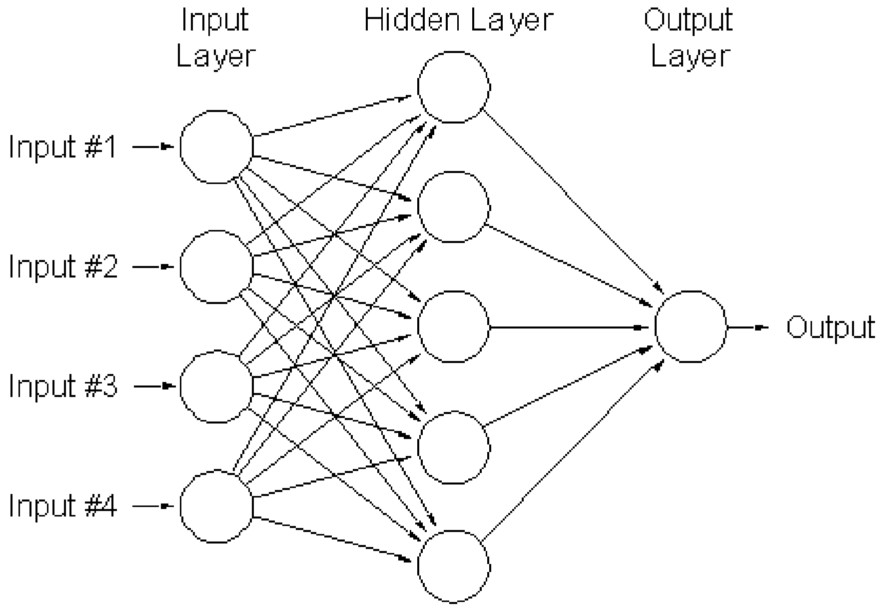
Sigmoid function: $\sigma(z) \equiv \frac{1}{1+e^{-z}}$



This shape is a smoothed out version of a step function

Architecture of (Feed-Forward) Neural Network (NN)

- **Input layer**
  - The **inputs** to a neural network correspond to the attributes measured for each training instance
  - Inputs are fed simultaneously into the units making up the **input layer**
- **Hidden layer(s)**
  - Inputs are weighted and fed simultaneously to a hidden layer
  - The number of hidden layers is arbitrary
- **Output layer**
  - The weighted outputs of the last hidden layer are input to units making up the **output layer**, which emits the network's prediction



Neural Network Architecture: Feed-Forward vs. Recurrent

- **Feed-Forward Neural Network:** Typical neural network architecture
  - The output from one layer is used as input to the next layer (no loops)
  - Information is always fed forward, never fed back
  - From a statistical point of view, networks perform **nonlinear regression**
  - Given enough hidden units and enough training samples, they can closely approximate any function
- **Recurrent neural network**: Feedback loops are possible (cascade of neurons firing)
  - Some neurons fire for some limited duration of time, before becoming quiescent
  - That firing can stimulate other neurons, which may fire a little while later, also for a limited duration, which causes still more neurons to fire, and so on
  - Loops do not cause problems since a neuron's output only affects its input at some later time, not instantaneously

Learning with Gradient Descent

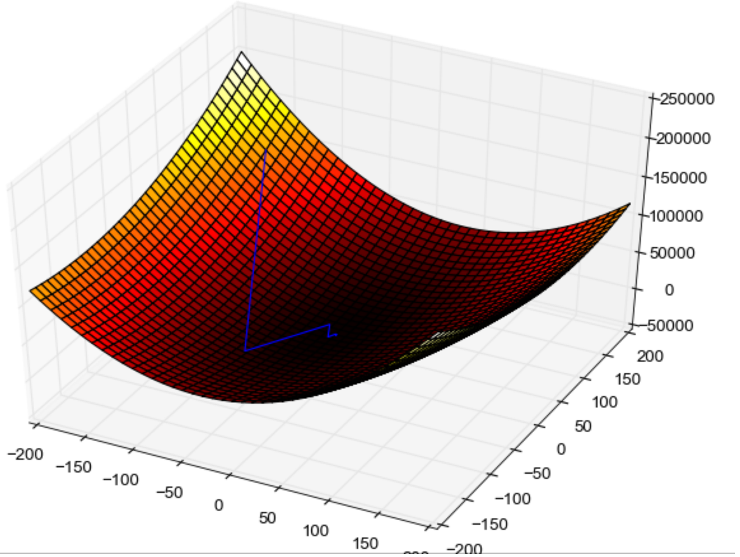
- A quadratic cost (objective) function C (or mean square error, MSE)

$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2$$

  where *w*: The collection of all weights in the network; *b*: All the biases; *n*: The total # of training inputs; *a*: The vector of outputs from the network when *x* is input

- Goal of training a network: Find weights and biases which minimize the cost *C*(*w*, *b*)
- For two variables, it means: Choose Δv_1 and Δv_2 to make negative; i.e., the ball is rolling down into the valley:

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta$$



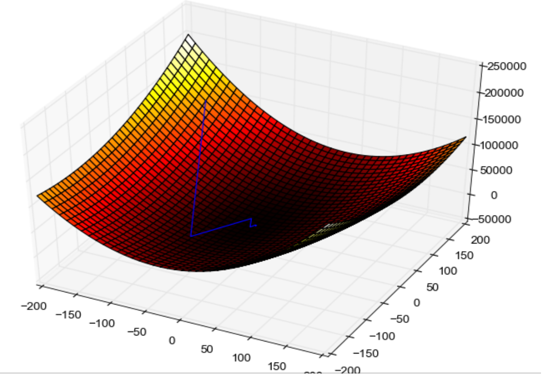
- The change ΔC *in C by a small change in v, Δv:*

$$\Delta C \approx \nabla C \cdot \Delta v$$

  where ∇C is the gradient vector:

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \ldots, \frac{\partial C}{\partial v_m}\right)$$

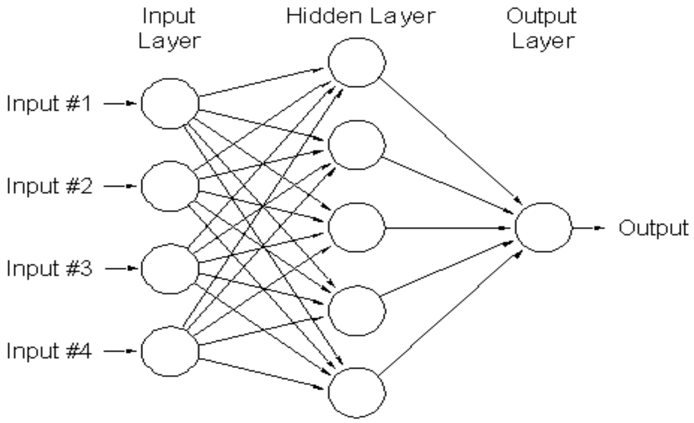
Stochastic Gradient Descent



- Gradient descent can be viewed as a way of taking small steps in the direction, which does the most to immediately decrease *C*
- To compute gradient ∇C, we need to compute the gradients ∇C_x separately for each training input, *x*, and then average them: Slow when the # of training inputs is large
- *Stochastic gradient descent* (SGD): Speed up learning
  - Computing for a small sample of randomly chosen training inputs and *averaging over them*, we can quickly get a good estimate of the true gradient
  - Method: Randomly pick out a small number (***mini-batch***) *m* of randomly chosen training inputs. Provided the sample size is large enough, we expect that the average value will be roughly equal to the average over all, that is, $\nabla C \approx \frac{1}{m}\sum_{j=1}^{m} \nabla C_{X_j}$
- Stochastic gradient descent in neural networks:
  - Pick out a randomly chosen minibatch of training inputs and train with them; then pick out another minibatch, until inputs exhausted - complete an *epoch* of training
  - Then we start over with a new training epoch

Backpropagation for Fast Gradient Computation

- **Backpropagation**: Reset weights on the "front" neural units and this is sometimes done in combination with training, where the correct result is known
- Iteratively process a set of training instances & compare the network's prediction with the actual known target value
- For each training instance, the weights are modified to **minimize the mean squared error** between the network's prediction and the actual target value
- Modifications are made in the "**backwards**" direction
  - From the output layer, through each hidden layer back to the first hidden layer, hence "**backpropagation**"
- Steps
  - Initialize weights to small random numbers, associated with b
  - Propagate the inputs forward (by applying activation function)
  - Backpropagate the error (by updating weights and biases)
    - Terminating condition (when error is very small, etc.)



More on Backpropagation

- With backpropagation, we distribute the “blame” backward through the network
  - Each hidden node sending input to the current node is somewhat “responsible” for some portion of the error in each neuron to which it has forward connection
- Local minima and backpropagation
  - Backpropagation can be stuck at local minima
  - But in practice it generally performs well
- Is backpropagation too slow?
  - Historically, backpropagation has been considered slow
  - Recent advances in computer power through parallelism and GPUs (graphics processing units) have reduced time substantially for training neural networks

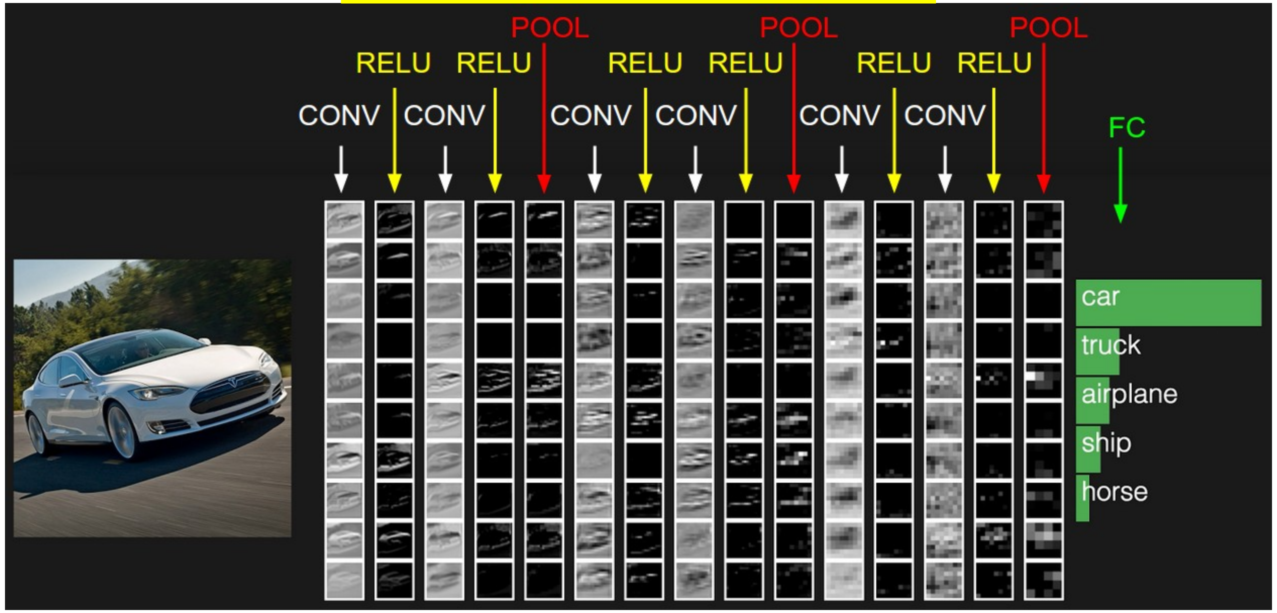
Deep Learning: A Short Introduction

From Neural Networks to Deep Learning

- Train networks with many layers (vs. shallow nets with just a couple of layers)
  - More neurons than previous networks
  - More complex ways to connect layers
  - Tremendous computing power to train networks
  - Automatic feature extraction
- Multiple layers work together to build an improved feature space
  - Analogy: Signals passing through regions of the visual cortex
    - Example: For face recognition: Edge → nose → face, layer-by-layer
- We introduce two popular deep learning frameworks for classification
  - Convolutional Neural Network (CNN)
  - Recurrent Neural Network (RNN)

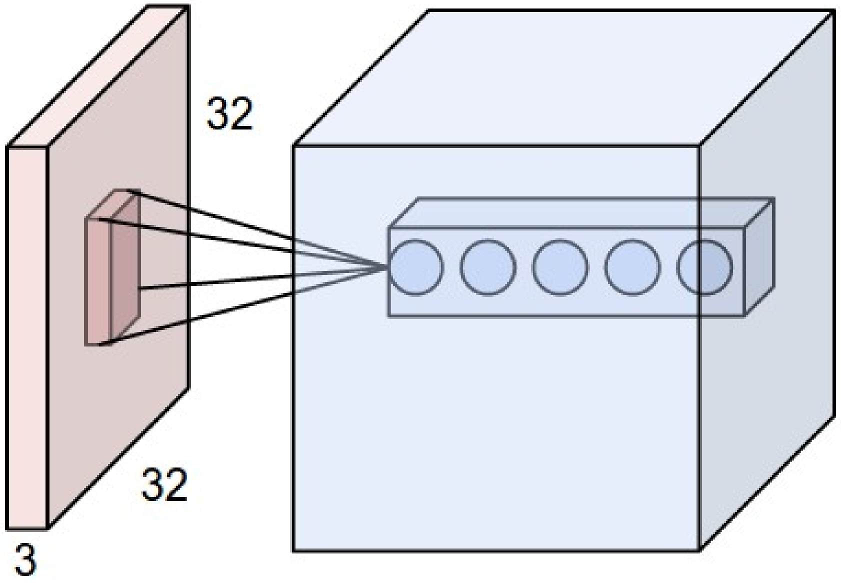
Convolutional Neural Networks: General Architecture

- Learn high-order features in the data via convolutions
  - Well suited to object recognition with images (e.g., computer vision)
  - Build position- and (somewhat) rotation-invariant features from raw image data
- CNN leverages learnable visual filters and globally shared local features
  - Specifics: High dimensional, 2D topology of pixels, invariance to translations, etc.
- High-level general CNN architecture
  - Input layer
  - Feature-extraction layers
    - (Convolution – ReLU - Pool)
  - Classification layers
- CNN properties
  - Local connectivity
  - Parameter sharing
  - Subsampling



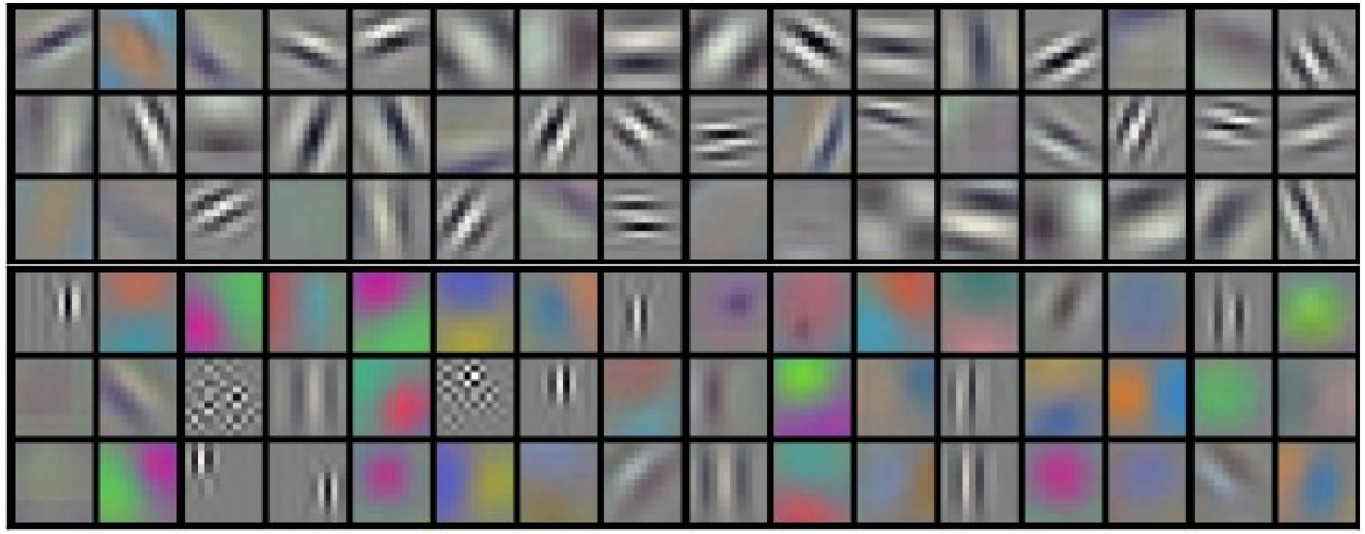
Convolutional Neural Networks: Local Connectivity

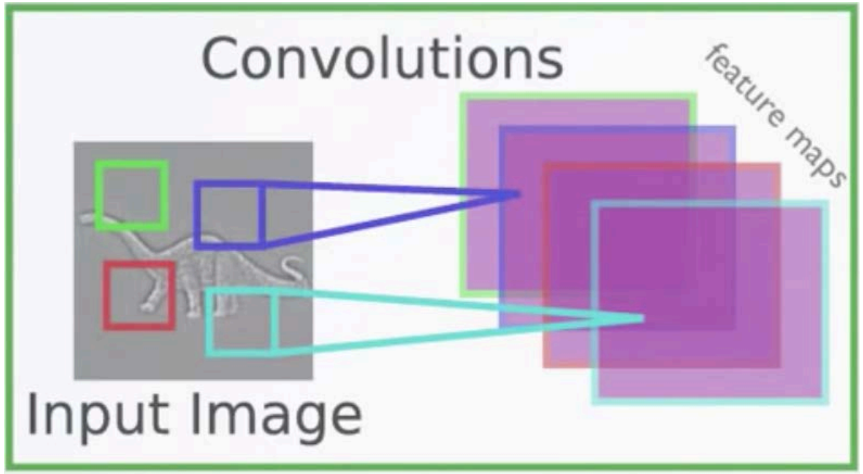
- Convolution: A math operation describing how to merge two sets of information
  - Filter (kernel): Sets of weights in a convolutional layer $(x * k)_{ij} = \sum_{pq} x_{i+p,j+q}\, k_{r-p,r-q}$
  - The filter is convolved with the input, resulting in a feature (activation) map
- Local Connectivity
  - Receptive fields: Each hidden unit is connected only to a sub-region of the image
    - Manageable number of parameters
    - Efficient computation of pre-activation
  - Spatial arrangements
    - Output Depth: Number of filters
    - Stride: How far our slide filter window will move
    - Zero-padding: Dealing with the border



Convolutional Neural Networks: Parameter Sharing

- Parameter sharing
  - Discrete convolution: Share matrix of parameters across certain units
    - Reduces even more the number of parameters
    - Extract the same feature at every position

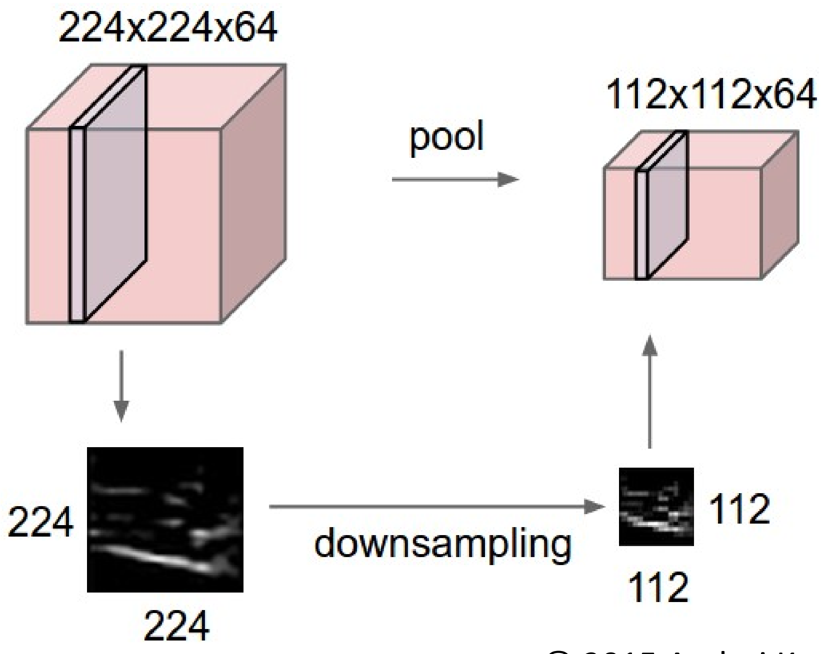




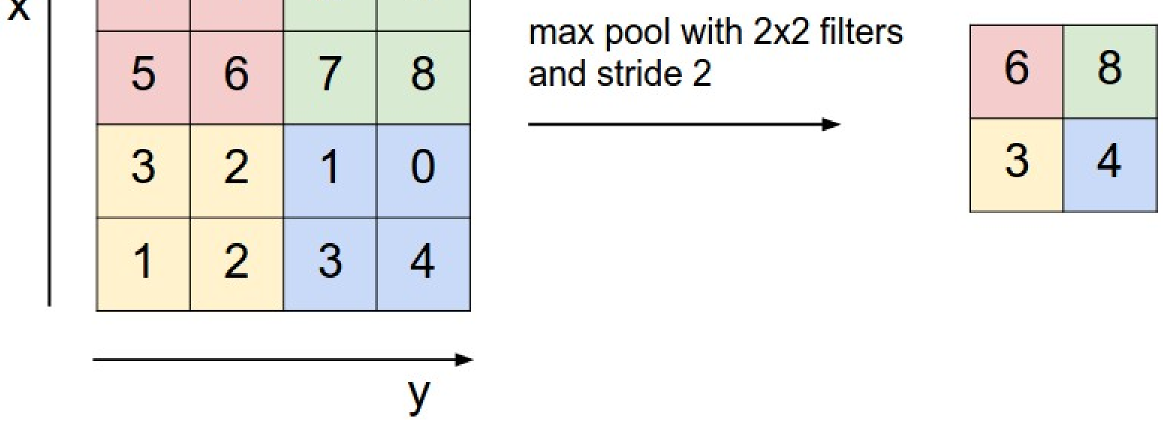
$$y_j = g_j \tanh\left(\sum_i k_{ij} * x_i\right)$$

Convolutional Neural Networks: Subsampling

- Pooling layers are commonly inserted between successive convolutional layers
- Subsampling:
  - Pooling: Pool hidden units in the same neighborhood
    - Introduces invariance to local translations
    - Reduces the number of hidden units in hidden layer



Max pooling: The most common downsampling operation



Recurrent Neural Networks: General Concepts

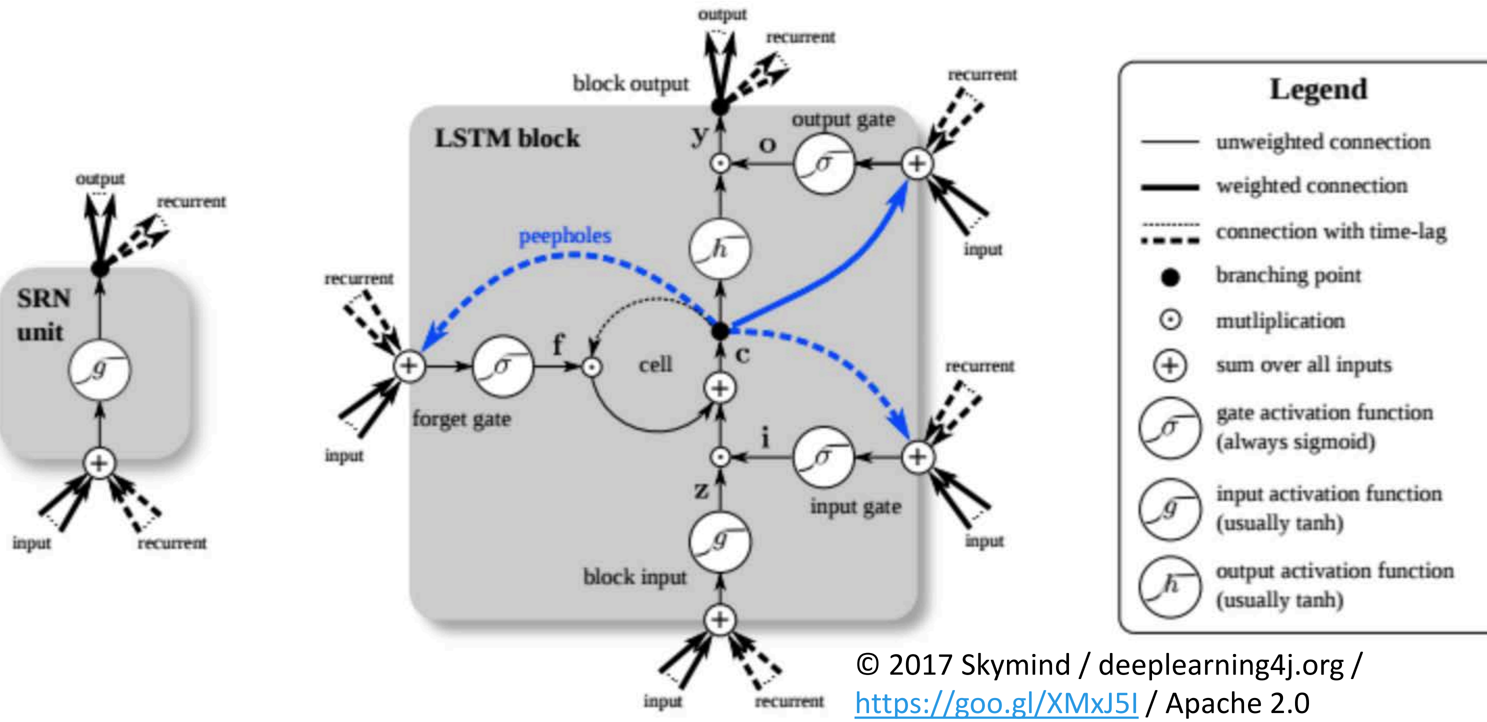
- Modeling the time dimension: By creating cycles in the network (thus “recurrent”)
  - Adding feedback loops connected to past decisions
  - Long-term dependencies: Use hidden states to preserve sequential information
  - Thus, it allows for both parallel and sequential computations
- Recurrent Neural Networks (RNNs) are trained to generate sequences: Output at each timestamp is based on both the current input and the inputs at all previous timestamps:

$$\mathbf{h}_t = \phi\left(W\mathbf{x}_t + U\mathbf{h}_{t-1}\right),$$

  - Compute a gradient with algorithm BPTT (backpropagation through time)
- Major obstacles of RNN: Vanishing and exploding gradients
  - When the gradient becomes too large or too small, it is difficult to model long-range dependencies (10 timestamps or more)
  - Solution: Use a variant of RNN: LSTM (Long Short-Term Memory) (by Hochreiter and Schmidthuber, 1997)

LSTM: A Variant of Recurrent Neural Network

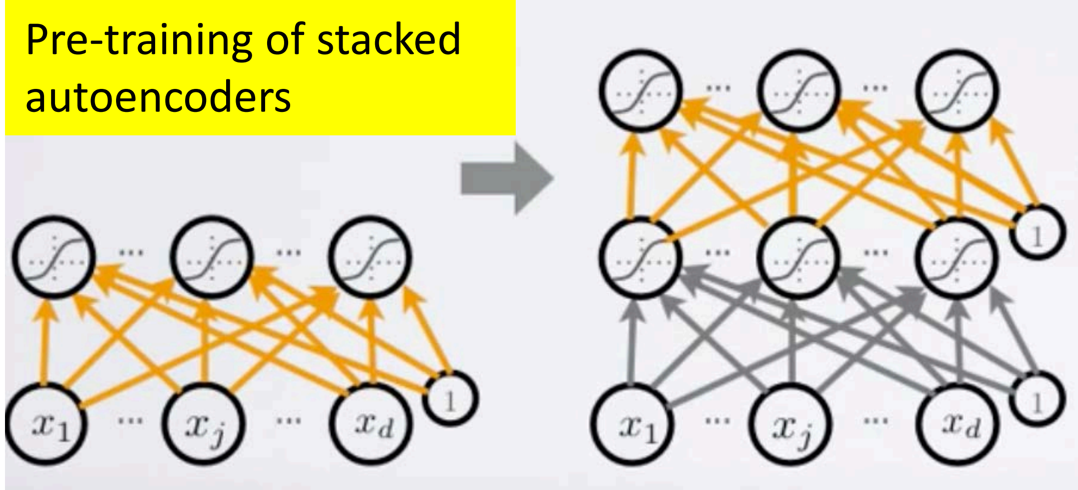
- Critical components of LSTM
  - Memory cells
  - 3 Gates (input, forget, output
- Use gated cells to
  - Write, store, forget informati
- When both gates are closed
  - The contents of the memory



- The gating structure allows information to be retained across many timestamps
  - Also allows gradient to flow across many timestamps
- By back-propagating errors and adjusting weights, one can learn what to store, and when to allow reads, writes, and erasures
- Applications: Handling sequence and time series data
  - E.g., NLP, video analysis, image captioning, robotics control

Difficulties of Training and Improvements

- Challenges
  - Vanishing gradient problem: Saturated units block gradient propagation
    - Need better optimization than SGD
  - Overfitting: High variance/low bias situation
    - Need better regularization (than L1, L2 norm)
- Many improvements proposed, such as
  - Autoencoder
    - Use unlabeled data in unsupervised learning
    - Build a compressed representation of the input data
  - Attention: Focusing on specific parts of the input
    - Taking *n* arguments (y_1, ..., y_n) and a context *c*, it returns a weighted arithmetic mean of the y_i, and the weights are chosen according to y_i's relevance to *c*



Summary

Summary

- Neural Networks

- Deep Learning: A Short Introduction

Recommended Readings

- Géron, Aurélien. (2017). *Hands-on machine learning with scikit-learn and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. Champaign, IL: O'Reilly.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. London, UK: The MIT Press.
- Patterson, J. & Gibson, A. (2017). *Deep learning: A practitioner's approach*. Champaign, IL: O'Reilly.
- Rashid, Tariq. (2016). *Make your own neural network*. Charleston, SC: CreateSpace.
- Numerous websites and online tutorials

References

- Alanf777. (2013). *130316c lc 05 learning curve S-curve or sigmoid.png* [Online image]. Retrieved Feb 19, 2018 from https://commons.wikimedia.org/wiki/File:130316c_lc_05_Learning_Curve_S-Curve_or_Sigmoid.png
- Charlie. (2005). *Neural network.gif* [Online image]. Retrieved Feb 19, 2018 from https://commons.wikimedia.org/wiki/File:Neural_Network.gif
- Chrislb. (2005). *ArtificialNeuronModel english.png* [Online image]. Retrieved Feb 19, 2018 from https://commons.wikimedia.org/wiki/File:ArtificialNeuronModel_english.png
- Karpathy, A. & Johnson, J. (2015). *CS231n convolutional neural networks for visual recognition* [Online images]. Retrieved Feb 19, 2018 from http://cs231n.github.io/convolutional-networks/
- Larochelle, H. (2013). Neural networks [7.2] : Deep learning - difficulty of training [Online video]. Retrieved Feb 19, 2018 from https://www.youtube.com/watch?v=YoiUlN_77LU&list=PL6Xpj9I5qXYEcOhn7TqghAJ6NAPrNmUBH&index=52
- Роман Сузи. (2015). *Gradient descent method.png* [Online image]. Retrieved Feb 19, 2018 from https://commons.wikimedia.org/wiki/File:Gradient_descent_method.png
- Skymind. (2017). *Long short-term memory units (LSTMs)* [Online image]. Retrieved Feb 19, 2018 from https://deeplearning4j.org/lstm.html