LECTURE 14: Neural Networks, Deep Networks, Convolutional Networks, …

- We start with SVM, which is a linear classifier
- We introduce nonlinearity, which expands the space of allowed solutions
- Next we introduce several layers
- Different types of layers: convolutions, pooling…
- Key numerical method is (stochastic, momentum…) gradient descent, so we need a loss function and a gradient: backpropagation algorithm
- Generative models: adversarial networks, autoencoders…

From SVM to NN

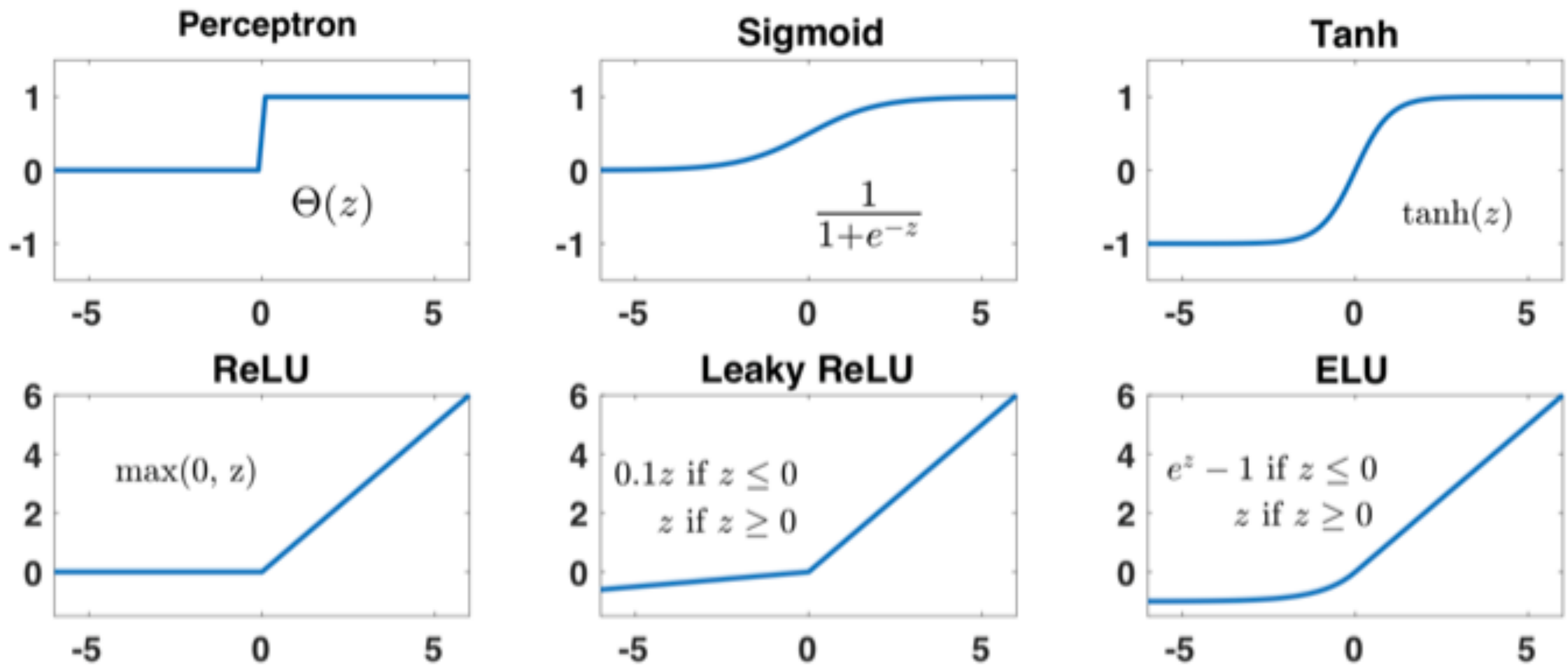
- SVM is a linear classifier (lecture 14): $\mathbf{s} = \mathbf{W}\mathbf{x}$, $\mathbf{x}$ is 3072 dim for CIFAR-10 and we classify 10 objects, so $\mathbf{W}$ = (10,3072) matrix, $\mathbf{s}$ is score in 10 dim
- Neural networks: instead of going straight to SVM classification using $max(0,\mathbf{s})$ we perform several intermediate steps of the form

  $\mathbf{s} = \mathbf{W}_2\, max(0, \mathbf{W}_1\mathbf{x})$ (2-layer network)

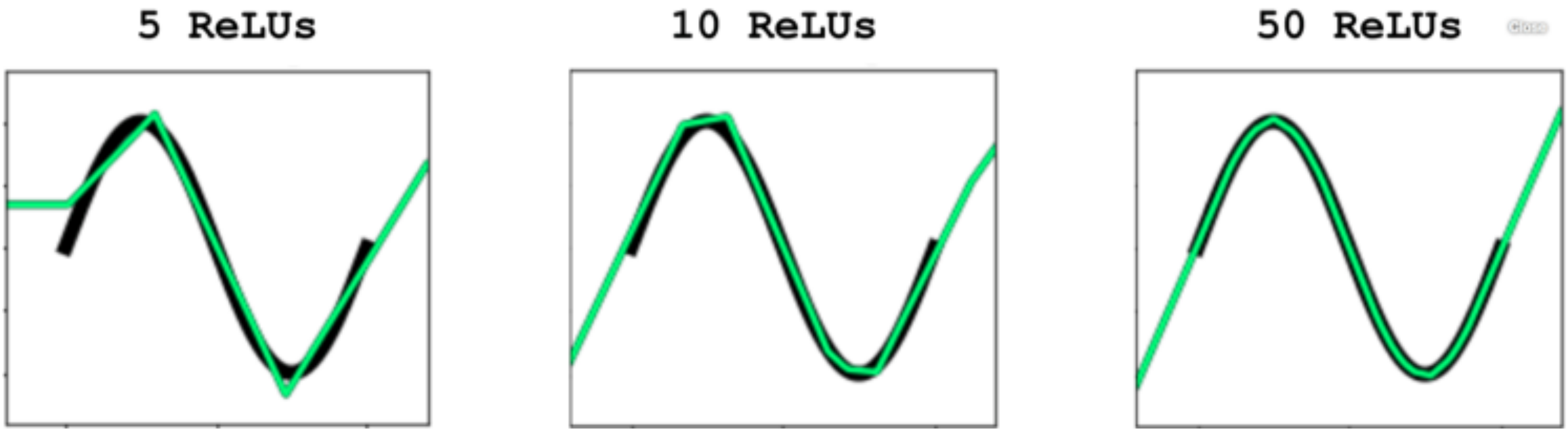
  $\mathbf{s} = \mathbf{W}_3\, max(0, \mathbf{W}_2\, max(0, \mathbf{W}_1\mathbf{x}))$ (3-layer network)
- Here $\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3 \ldots$ all need to be trained
- Alternative names: Artificial NN (ANN), multi-layer perceptron (MLP)

Activation Functions

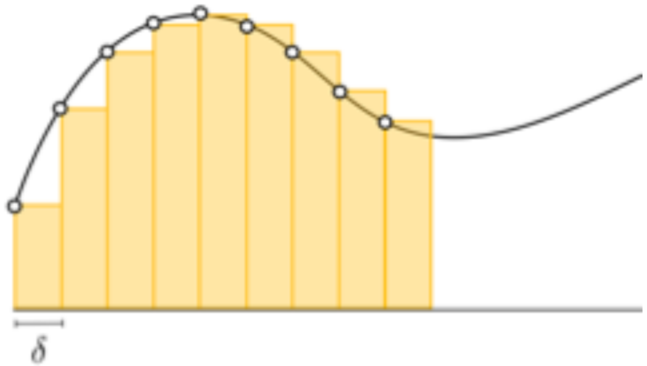
- Max(0,x) is a ReLU (rectified Linear Unit) non-linearity (activation function). Other options are sigmoid, tanh… sigmoid maps from 0 to 1, hence called activation
- ReLU is argued to accelerate convergence of stochastic gradient (Krizhevsky etal 2012)

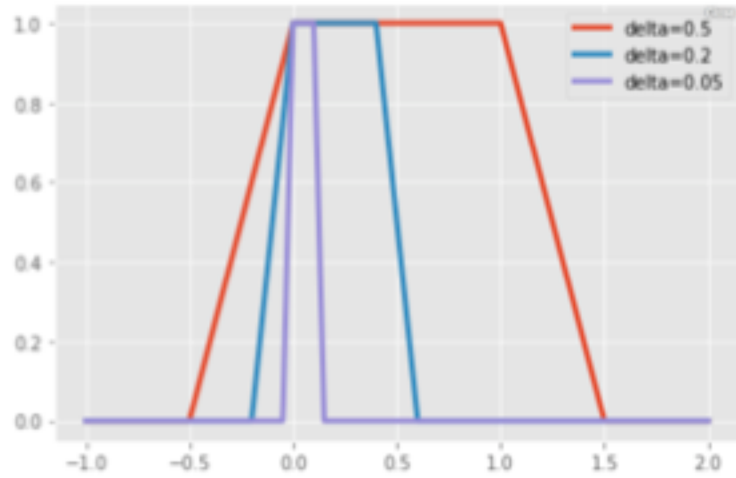


ReLU as a universal approximator



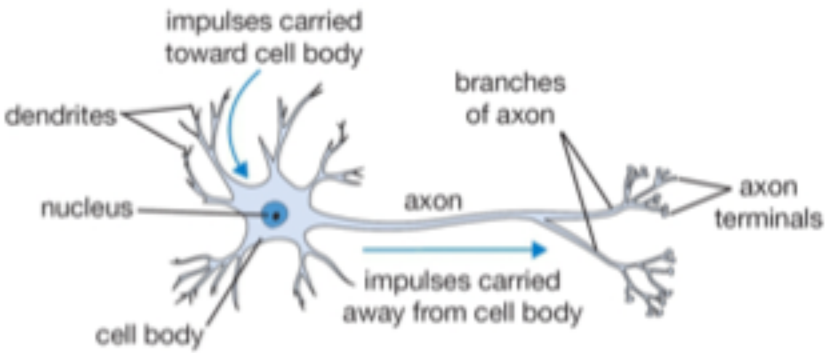
$$\frac{1}{\delta}[\max(0, x+\delta) - \max(0, x) - \max(0, x-\delta) + \max(0, x-2\delta)]$$

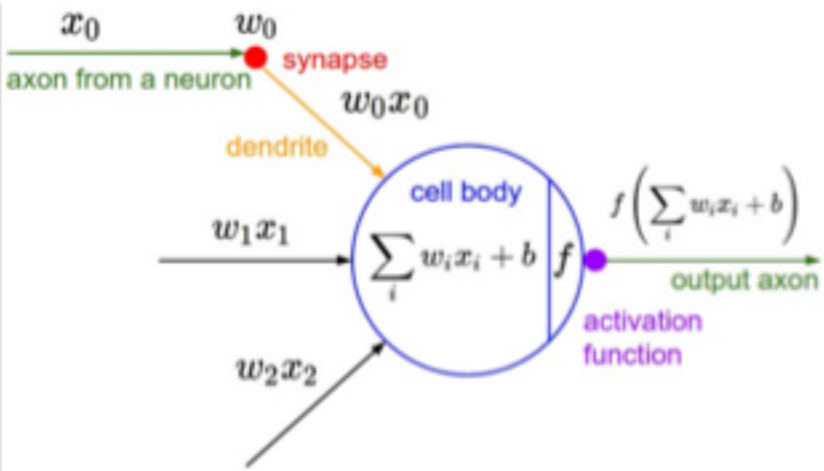




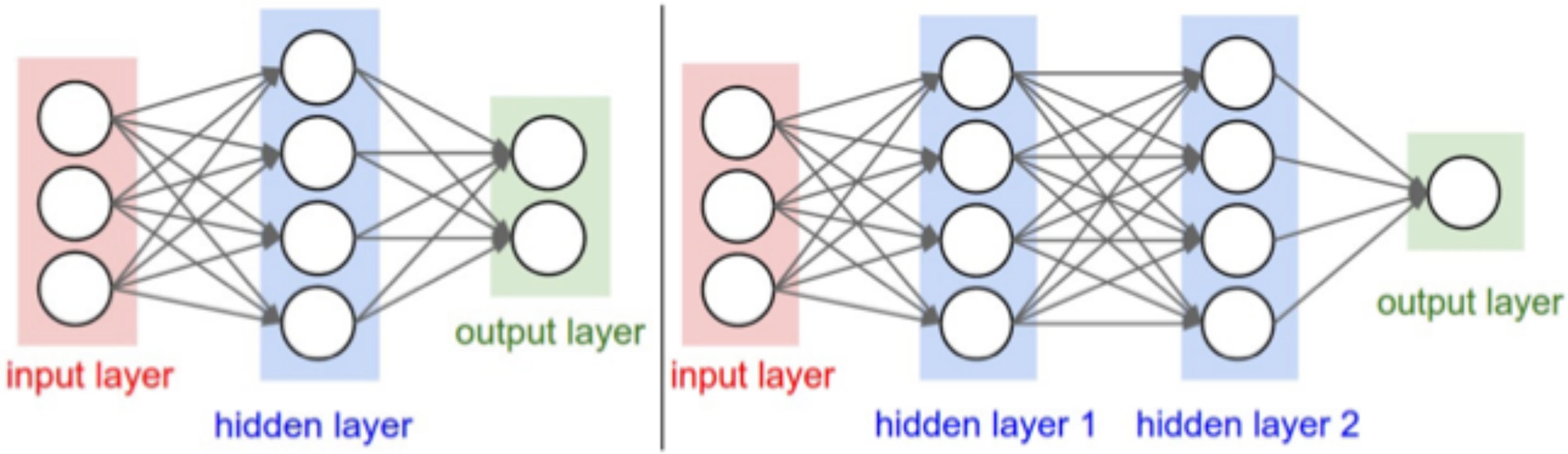
Why Neural Networks?

- There is a useful biological picture that inspired ANN
- Brain has 10^{11} neurons (cells) connected with up to 10^{15} synapses (contact points between the cells: axon terminals on one end, dendrites on the other)
- Neurons are activation cells: $f(\mathbf{Wx})$, synapses are weights w_i multiplying inputs from previous neuron layer $\mathbf{x}_i$



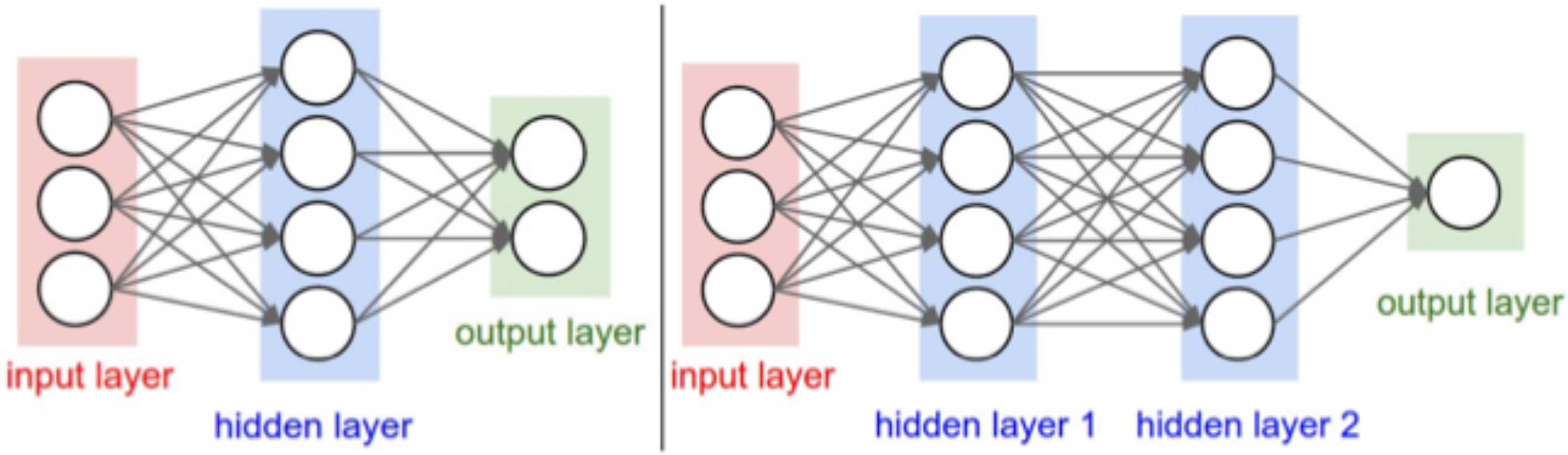


Neural Network Architectures



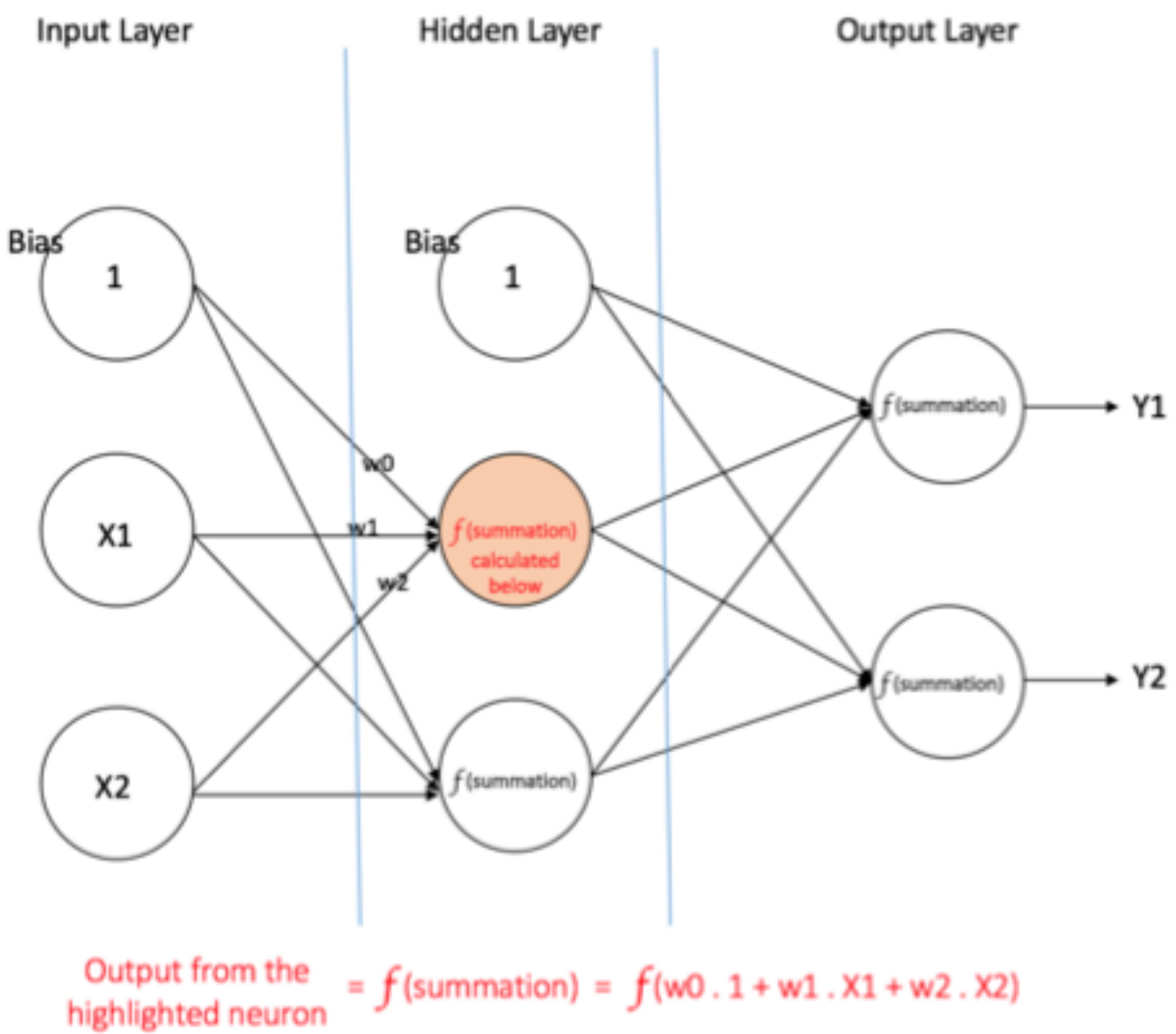
- Naming: N-layer not counting input layer. Above examples of 2-layer and 3-layer. SVM is 1-layer NN
- Fully connected layer: all neurons connected with all neurons on previous layer
- Output layer: class scores if classifying (e.g. 10 for CIFAR 10), a real number if regression (1 neuron)

Neural Network Architectures

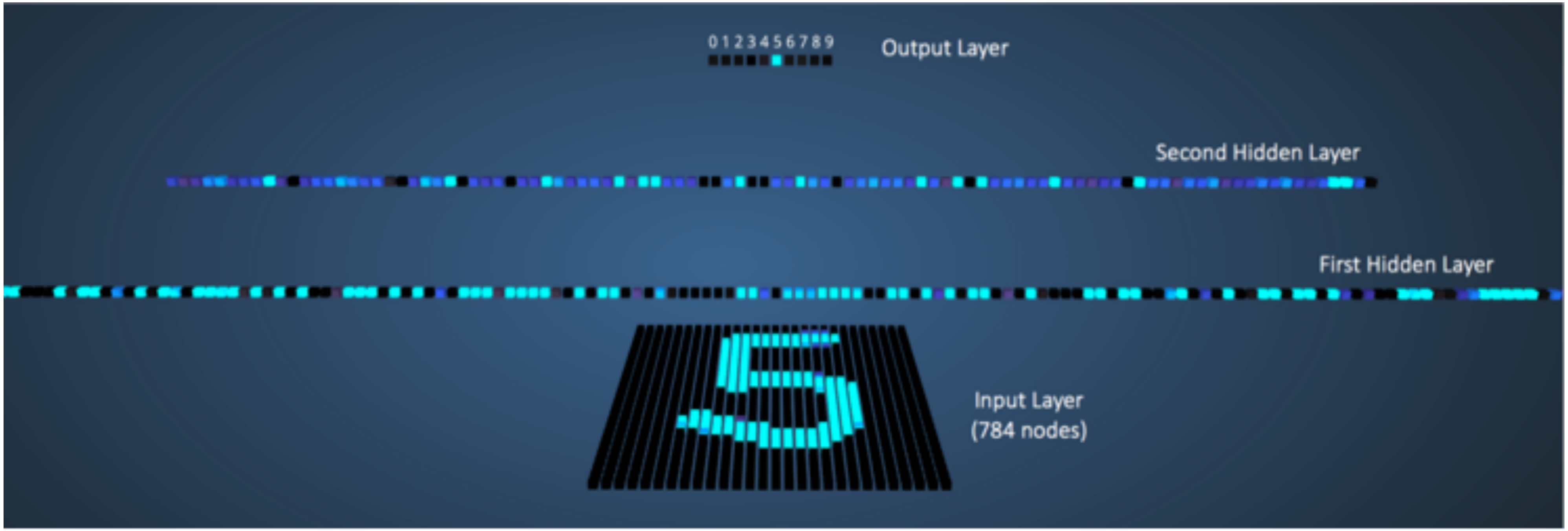


- Left layer: 4+2 = 6 neurons (not counting input), 3x4 + 4x2 = 20 weights, 4+2 biases, 26 learning parameters
- Right layer: 4+4+1 = 9 neurons, 3x4 + 4x4 + 4x1 = 32 weights, 9 biases, total 41 parameters

NN Examples


Input Layer
Hidden Layer
Output Layer
Bias
1
Bias
1
X1
X2
w0
w1
w2
f(summation)
calculated
below
f(summation)
f(summation)
f(summation)
Y1
Y2
Output from the
highlighted neuron
= f(summation) = f(w0 . 1 + w1 . X1 + w2 . X2)

NN Examples



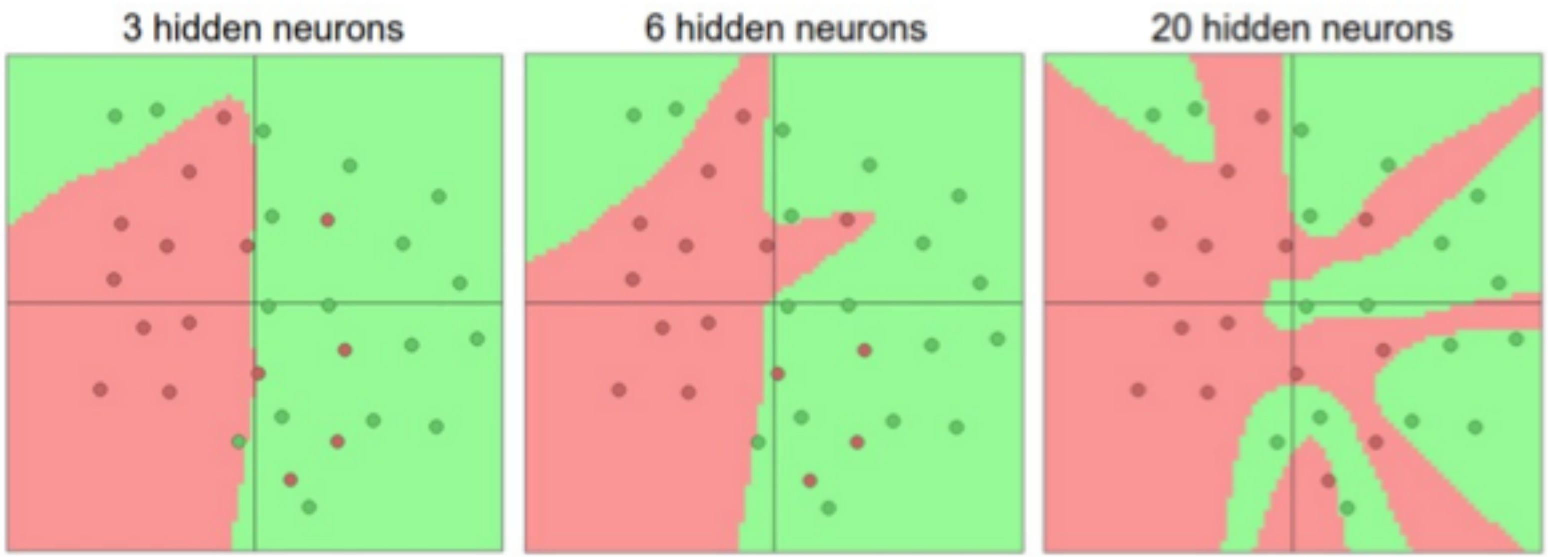
0 1 2 3 4 5 6 7 8 9
Output Layer
Second Hidden Layer
First Hidden Layer
Input Layer
(784 nodes)

Representational Power

- NN can be shown to be universal approximators: even one layer NN can approximate any continuous function
- This is however not very useful. $g(x) = \sum_i c_i \mathbb{1}(a_i < x < b_i)$ Where a,b,c are vectors is also universal approximator, but not a very useful one.
- ANN are (potentially) useful because they can represent the problems we encounter in practice

Setting Number of Layers and Their Sizes

- As we increase number of layers and their size the capacity increases: larger networks can represent more complex functions
- We encountered this before: as we increase the dimension of the basis function we can represent more complex functions
- The flip side is that we start fitting the noise: overfitting problem

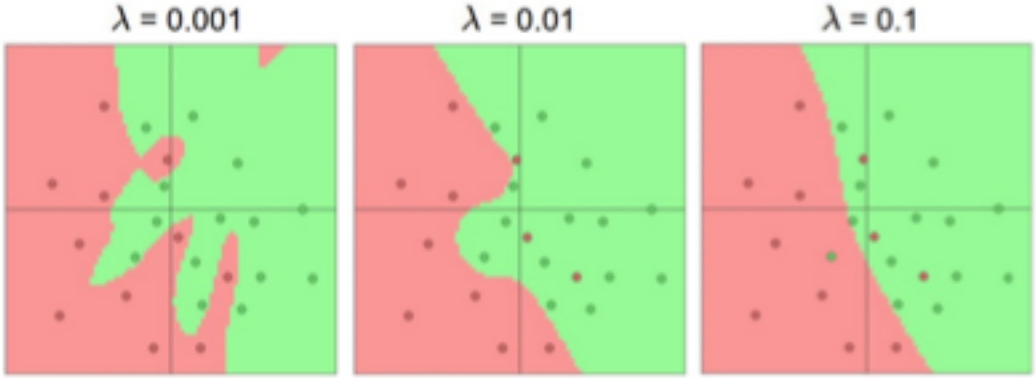


Regularization

- Instead of reducing the dimensionality (number of neurons) we can allow larger dimensionality (more neurons) but with regularization (L1, L2…): e.g. L2

$$\lambda \sum_k \sum_l W_{k,l}^2$$

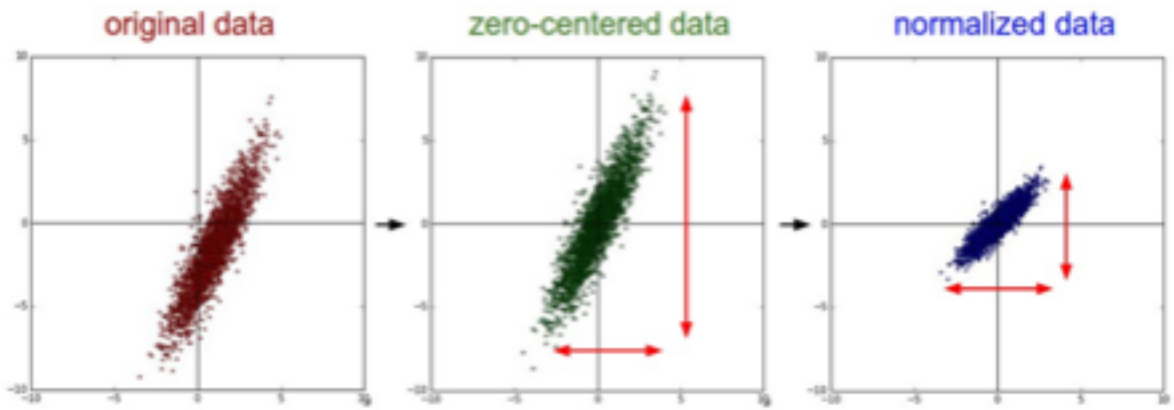
- Example:
  20 neurons



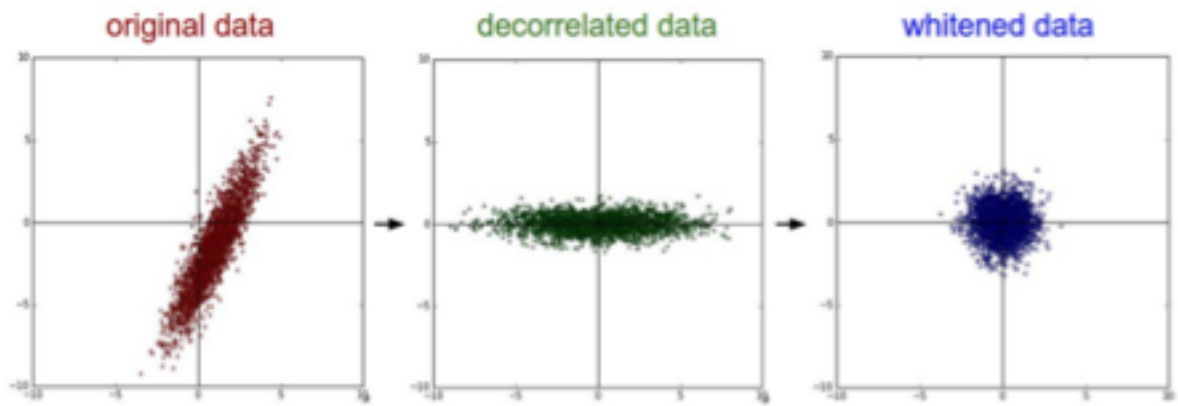
- Advantage of higher dimensions: these are non-convex optimizations (nonlinear problem). In low dimensions the solutions are local minima with high loss function (far from global minimum). In high dimensions local minima are closer to global.
- Lesson: use high number of neurons/layers and regularize

Preprocessing

- In NN one uses standard preprocessing methods we have seen before (PCA/ICA): mean subtraction, normalization:



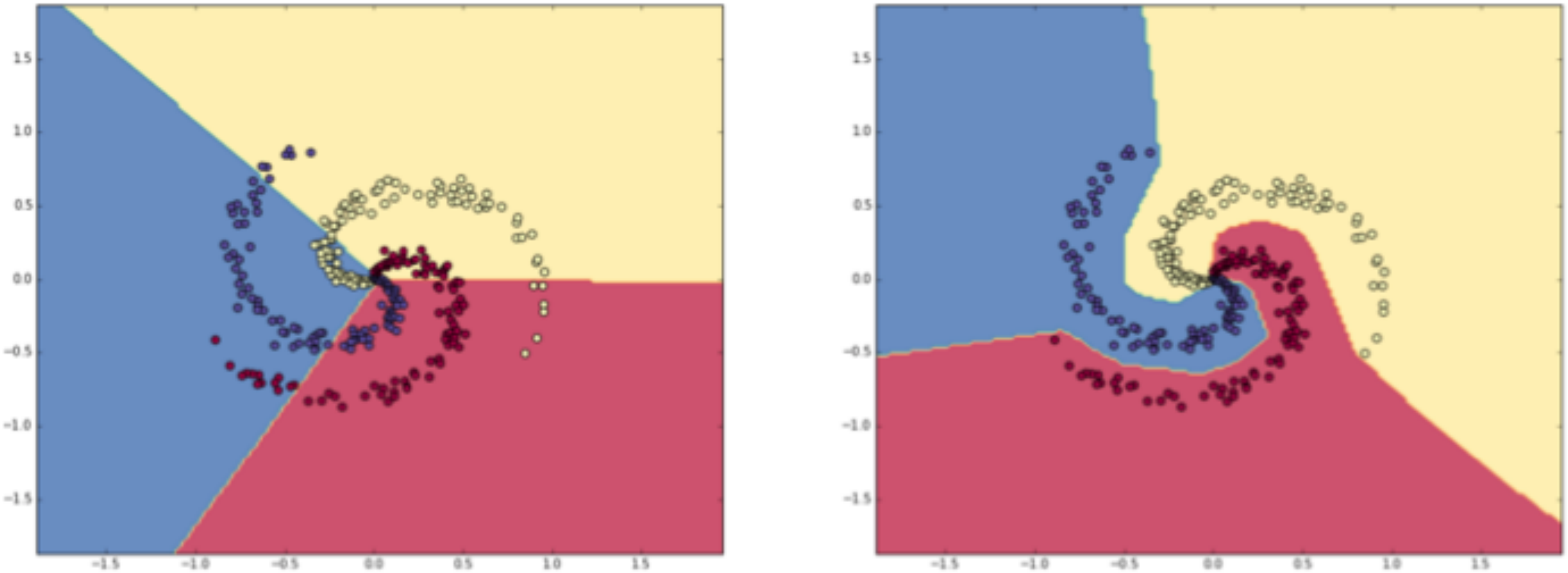
- PCA, whitening, dimensionality reduction are typically not used in NN (too expensive)



Example: NN vs. SVM

- Spiral example: cannot be classified well with SVM

- 1 layer NN (SVM) 2-layer NN with ReLU



Loss function choices

- For regression the usual L_2 or L_1 norm loss

$$E(\mathbf{w}) = \frac{1}{N}\sum_i (y_i - \hat{y}_i(\mathbf{w}))^2 \qquad E(\mathbf{w}) = \frac{1}{N}\sum_i |y_i - \hat{y}_i(\mathbf{w})|$$

- Regularization can be explicit L_2 or L_1 norm or implicit
- For classification the usual cross-entropy: 2 labels

$$E(\mathbf{w}) = -\sum_{i=1}^{n} y_i \log \hat{y}_i(\mathbf{w}) + (1 - y_i) \log [1 - \hat{y}_i(\mathbf{w})]$$

- M labels

$$y_{im} = \begin{cases} 1, & \text{if } y_i = m \\ 0, & \text{otherwise} \end{cases} \qquad E((\mathbf{w})) = -\sum_{i=1}^{n}\sum_{m=0}^{M-1} y_{im} \log \hat{y}_{im}(\mathbf{w}) + (1 - y_{im}) \log [1 - \hat{y}_{im}(\mathbf{w})]$$

Implicit regularization choices

- Stochastic gradient descent has noise, less likely to overfit
- Early stopping criterion: compare performance between training and validation data (not used for training), when the loss functions start to deviate stop
- Dropout
- Batch normalization:

mean subtraction and normalization

done on hidden layer outputs

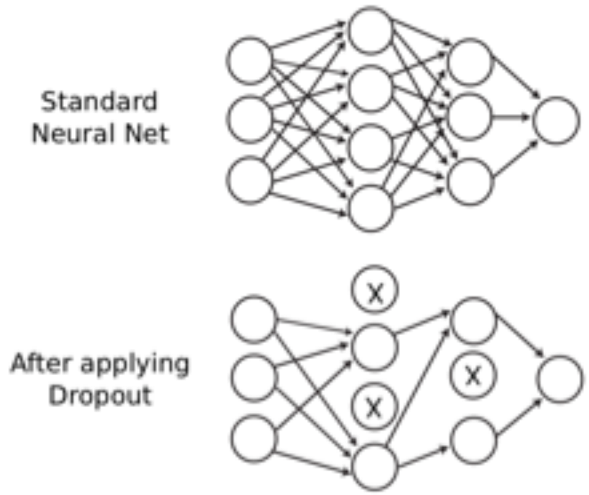
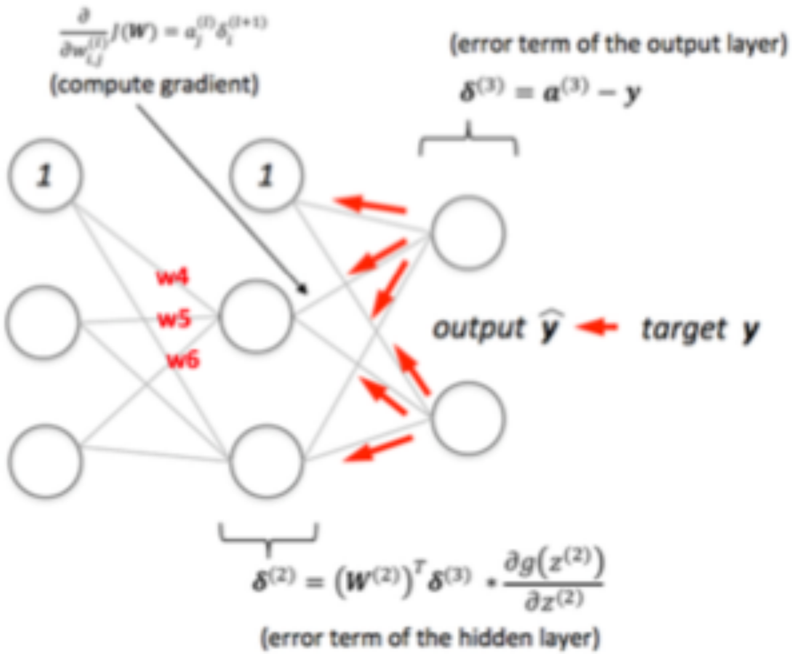


FIG. 39 **Dropout** During the training procedure neurons are randomly "dropped out" of the neural network with some probability p giving rise to a thinned network. This prevents overfitting by reducing correlations among neurons and reducing the variance in a method similar in spirit to ensemble methods.

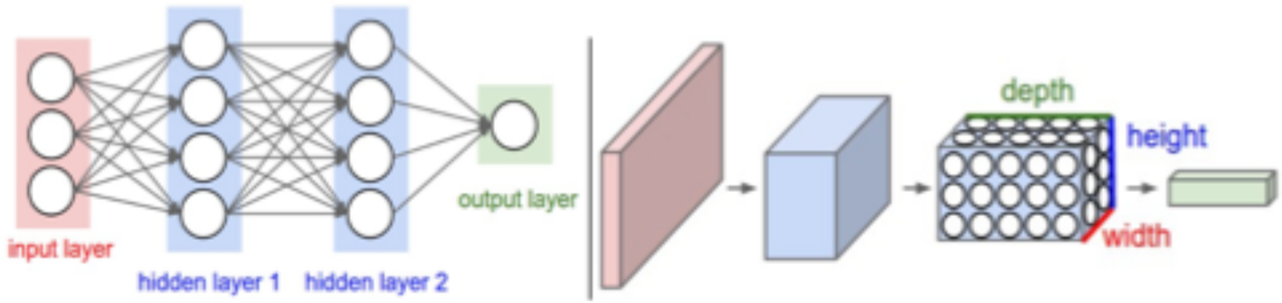
Backpropagation on NN: Example

- Propagate the gradient of the loss function $(f(W)-y)^2$ (plus regularization) with respect to all weights w
- Do this for each training data input: add them all up (full batch), or subsample (stochastic gradient)
- Initialize weights to small random (cannot all be 0)



Convolutional NN (CNN, Convnets)

- For images the number of dimensions quickly explode: e.g. pixelized 200x200x3 colors=120,000
- It is not useful to see every pixel as its own dimension: instead one wants to look at the hierarchy of scales, using translational invariance on each scale. Train the image on small scale features first, then intermediate scale features, large scale features… Spatial relation between pixels must be preserved
- To achieve this we arrange neurons in a 3-d volume of width, height, depth, and then connect nearby neurons
- CIFAR-10
  32 width,
  32 height,
  3 depth

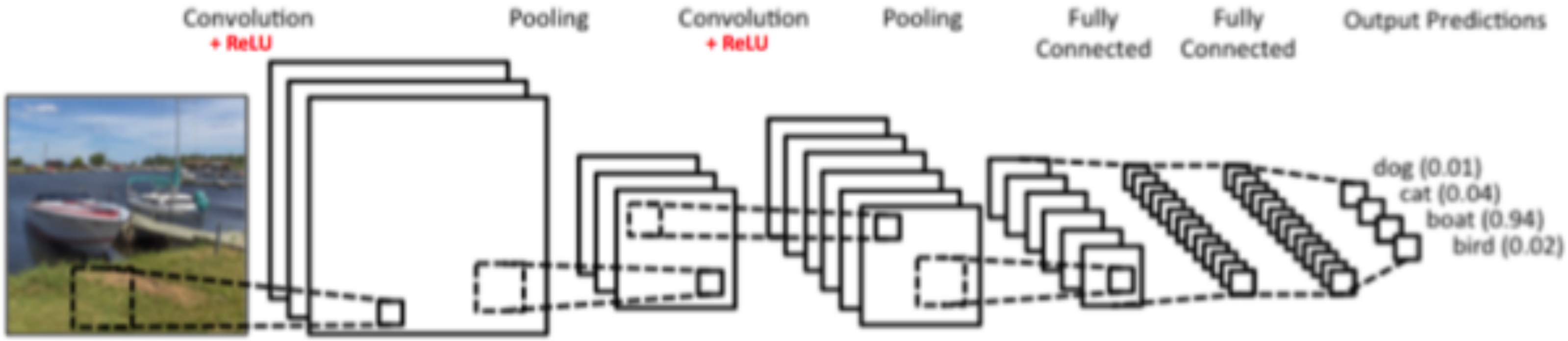


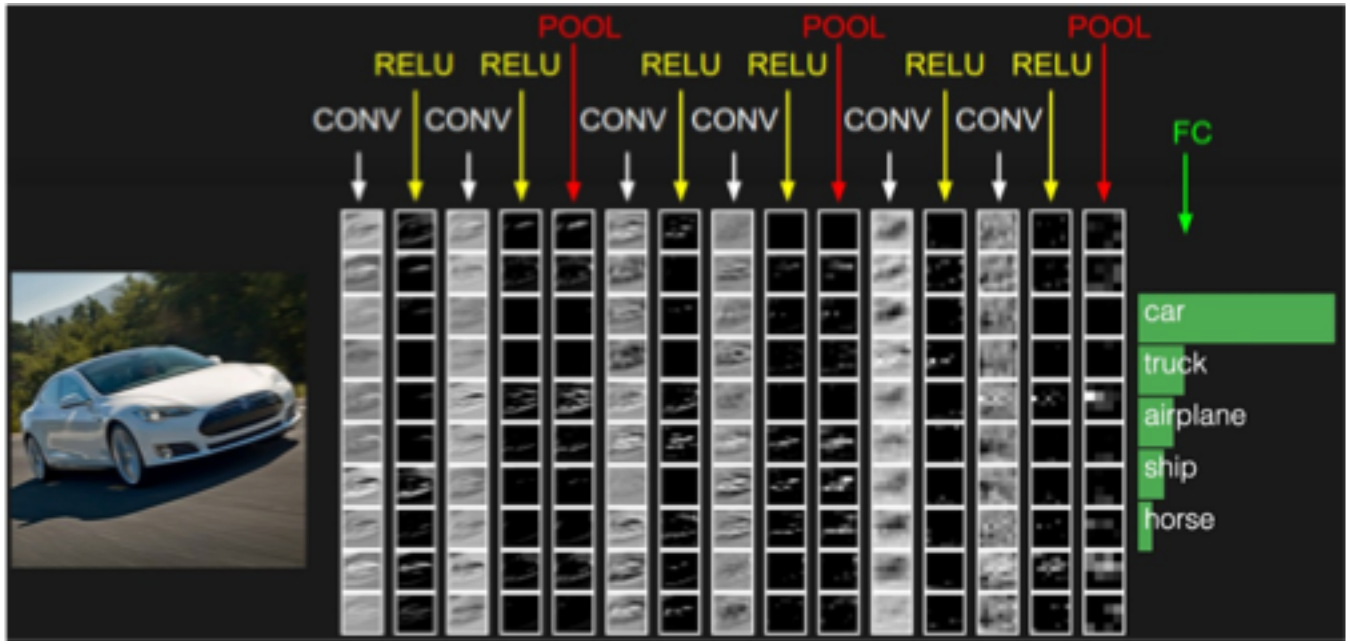
Convnet Layers

- Input layer: raw images, e.g. 32x32x3
- Convolutional layer: computes outputs of neurons that are connected to a local region in input, each evaluating a dot product between their weights and a small region they are connected to in inputs. This can give e.g. 32x32x12 if we use 12 filters (kernels)
- Depth: input layer number of colors (RGB), later number of filters
- Activation such as ReLU max(0,x): this leaves dimension unchanged 32x32x12
- POOL layer downsamples in spatial dimensions, e.g. 16x16x12 (coarse-graining)
- FC (fully-connected) layer outputs scores, e.g. 1x1x10 for CIFAR-10
- POOL/ReLU no parameters, CONV/FC weights+biases

Examples

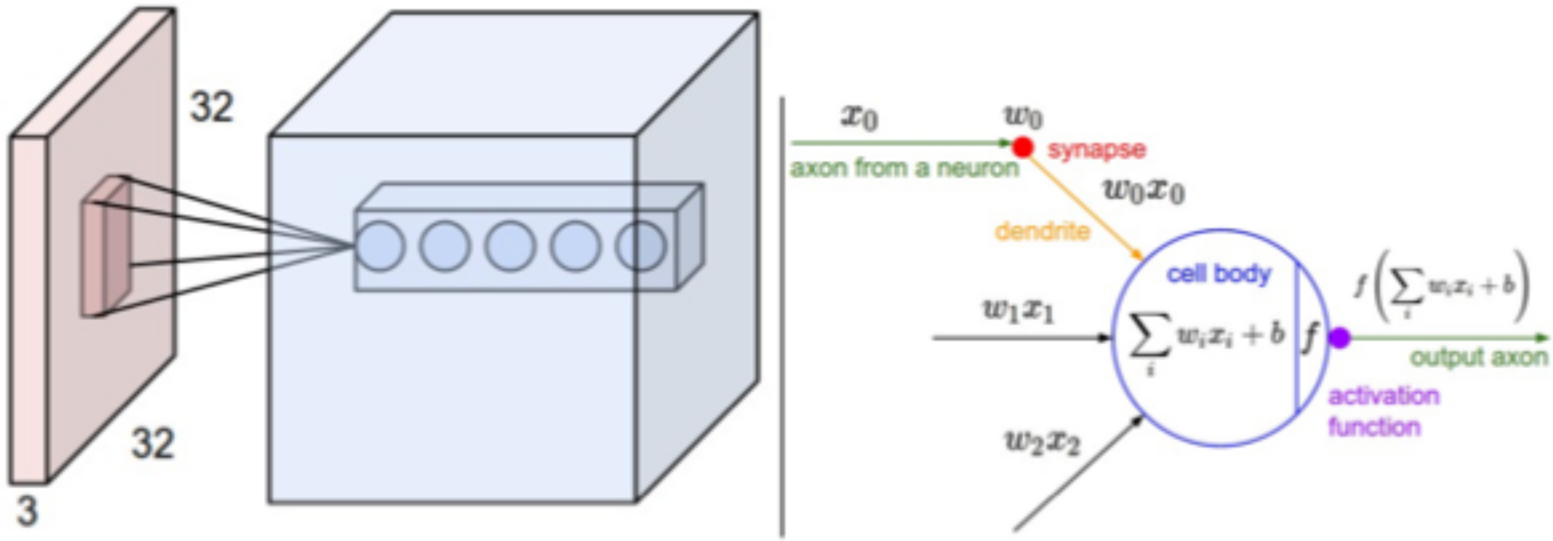
- A series of building blocks, giving a deep network





CONV Layer

- Building block of CONVNETS: it is a localized filter (kernel, feature detector) that convolves the image. Fully extends in the color(depth) dimension, but localized in width and height. Example: 5x5x3 filter for a total of 75 weights (+1)
- Note that we will typically have more filters than color on the first layer, so we are increasing the volume (32x32x5 in the example below)



Features

- One is looking for specific features
- We saw some of these in FFT lecture
- Here these are sparse matrix operations
- CNN learns these filters on its own
- Spatial size a hyperparameter (3x3 here)

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Hyperparameters of CONV Layer

- Depth, size: number of specific features, e.g. edges, blobs of color etc. and their spatial size (e.g. F=3)
- Stride: if 1 then we move the filter by 1 pixel. If 2 then we move by 2, resulting in 4 reduction of output volume.
- Zero padding: we discussed this in lecture 15

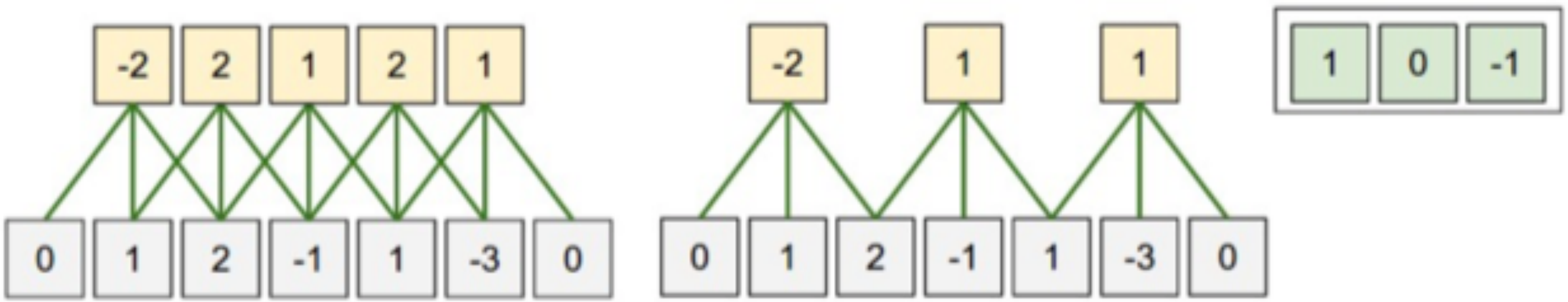
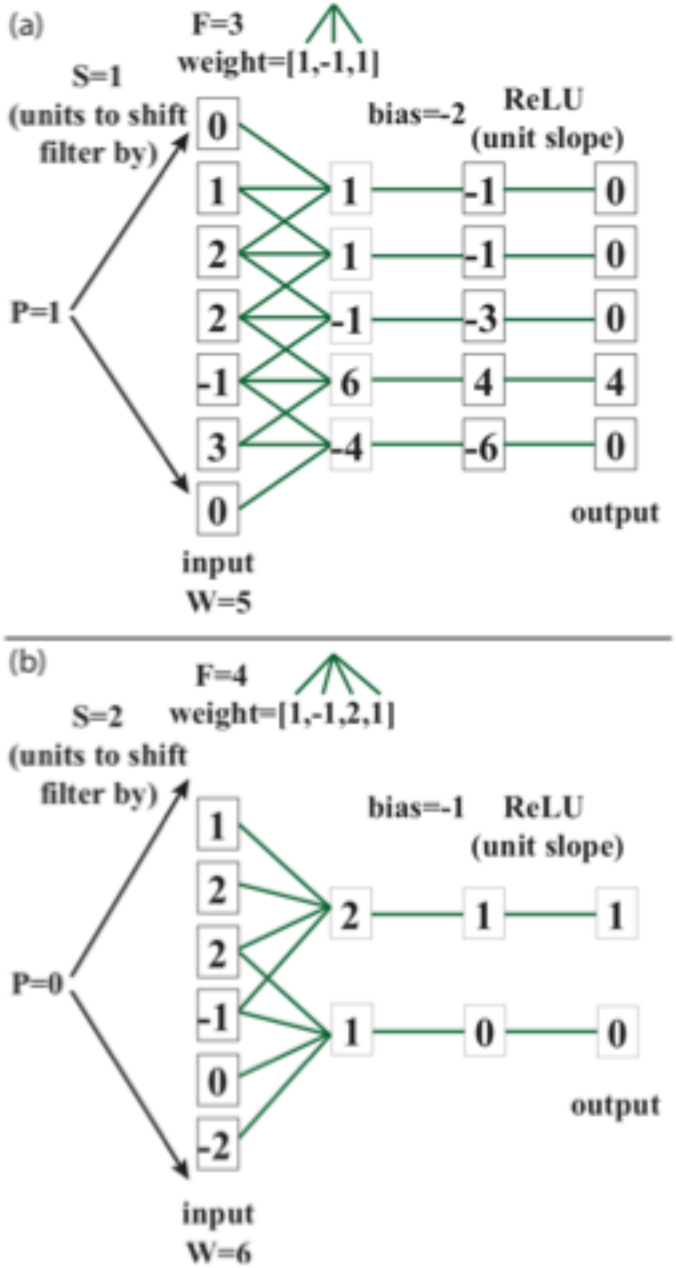
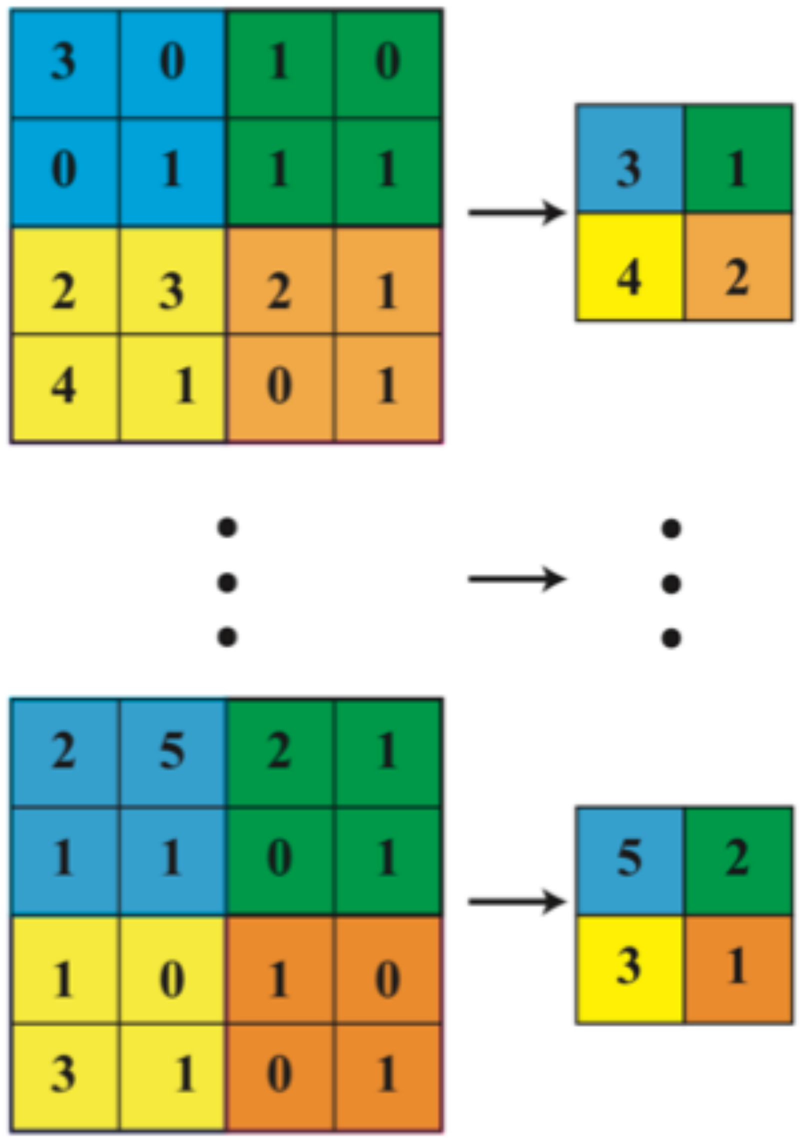


Illustration of spatial arrangement. In this example there is only one spatial dimension (x-axis), one neuron with a receptive field size of F = 3, the input size is W = 5, and there is zero padding of P = 1. **Left**: The neuron strided across the input in stride of S = 1, giving output of size (5 - 3 + 2)/1+1 = 5. **Right**: The neuron uses stride of S = 2, giving output of size (5 - 3 + 2)/2+1 = 3. Notice that stride S = 3 could not be used since it wouldn't fit neatly across the volume. In terms of the equation, this can be determined since (5 - 3 + 2) = 4 is not divisible by 3.
The neuron weights are in this example [1,0,-1] (shown on very right), and its bias is zero. These weights are shared across all yellow neurons (see parameter sharing below).

Examples: ReLU

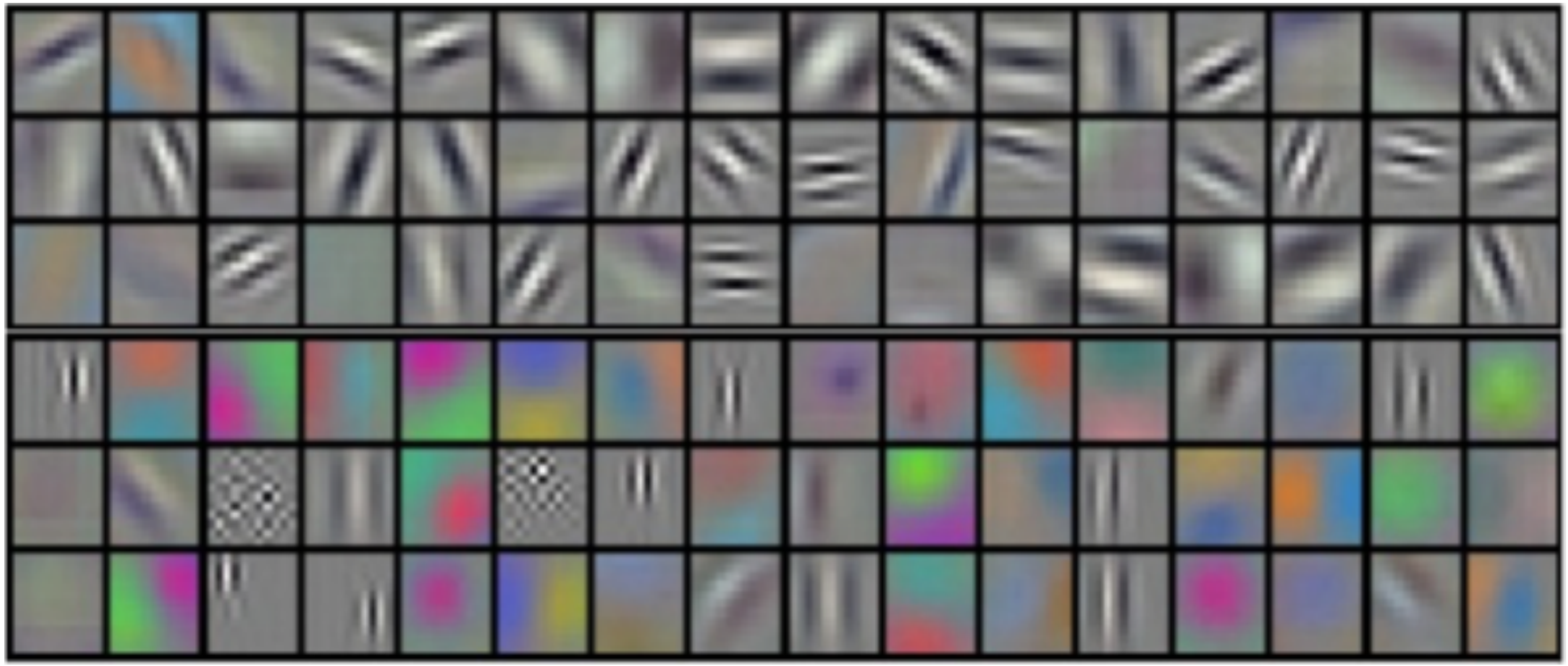


Max Pooling



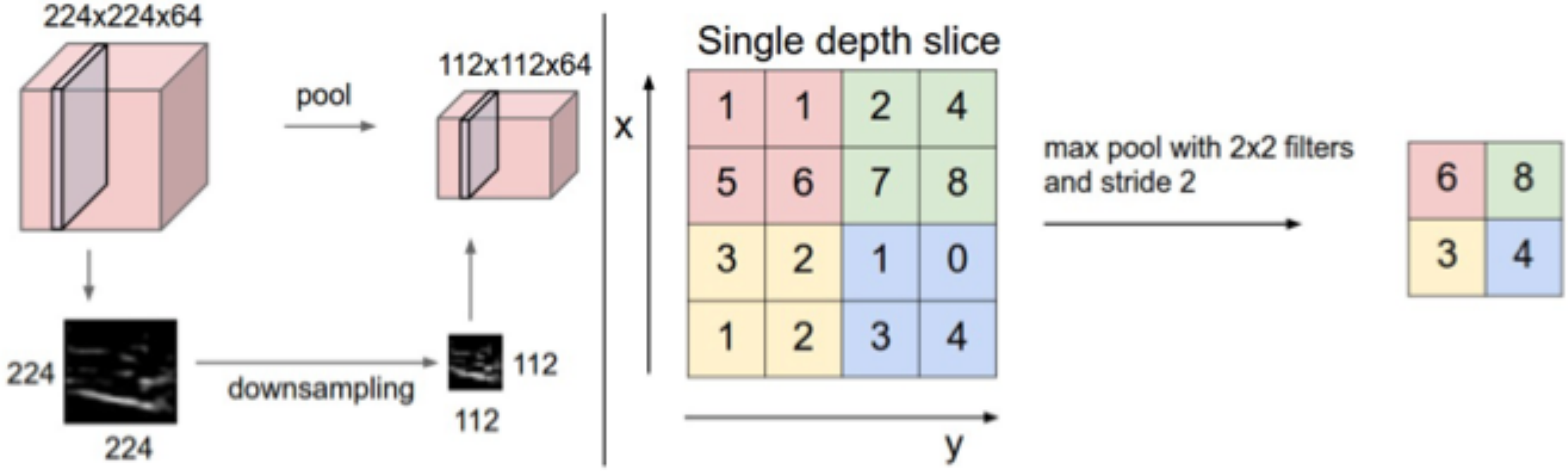
Example: Alexnet (2012)

- Input images 227x227x3, filter size 11, no zero padding, stride 4, depth 96, output layer 55x55x96
- Translational invariance: we assume the features can be anywhere in the image. Then all weights are independent of position, so one has 11x11x3=363 weights (+1) for each of 96 filters. Example filters:



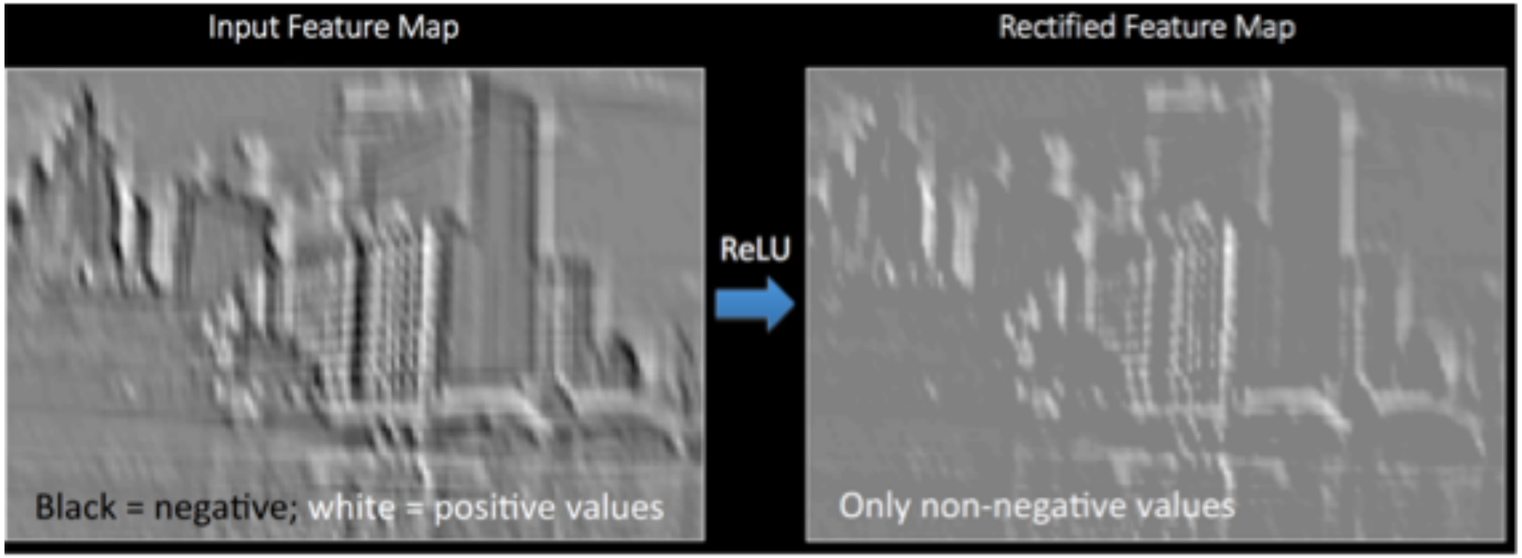
POOL Layer

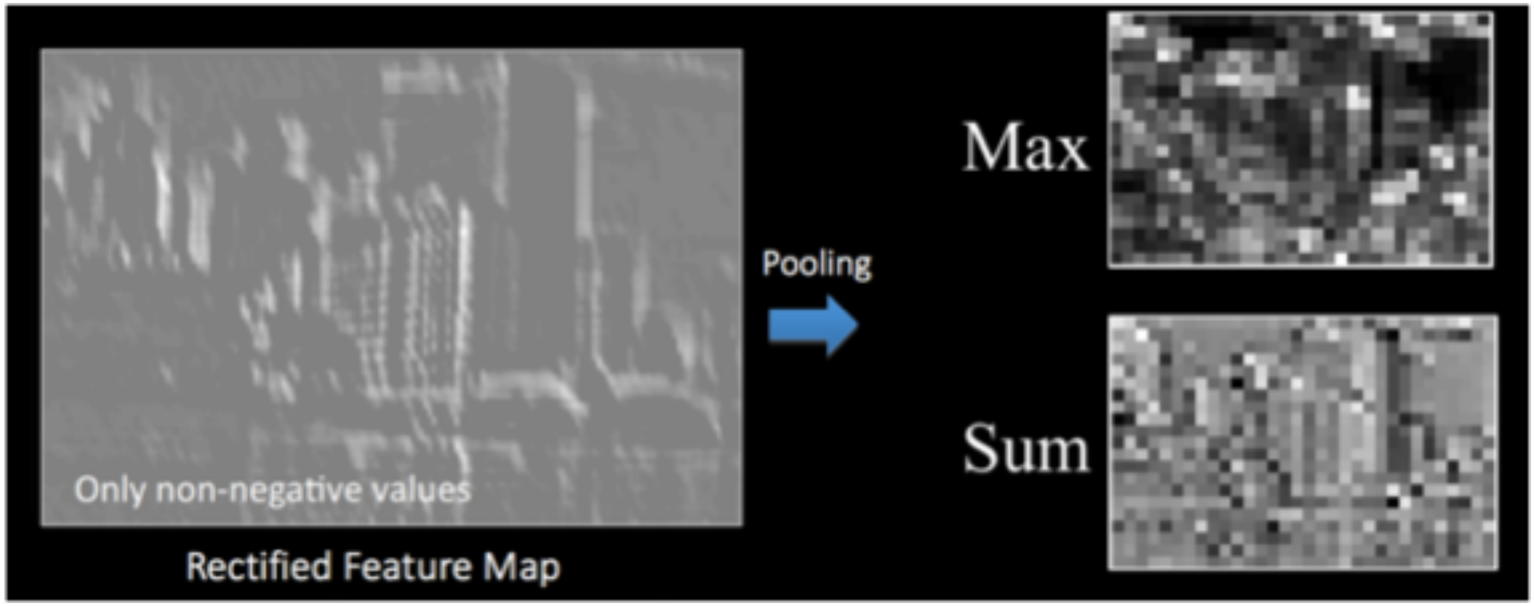
- Reduces spatial dimension, e.g. Max POOL 2x2 with stride 2 keeps the largest of 2x2 elements



- Similar to stride, so both are not necessarily needed. Opinions differ which is better.

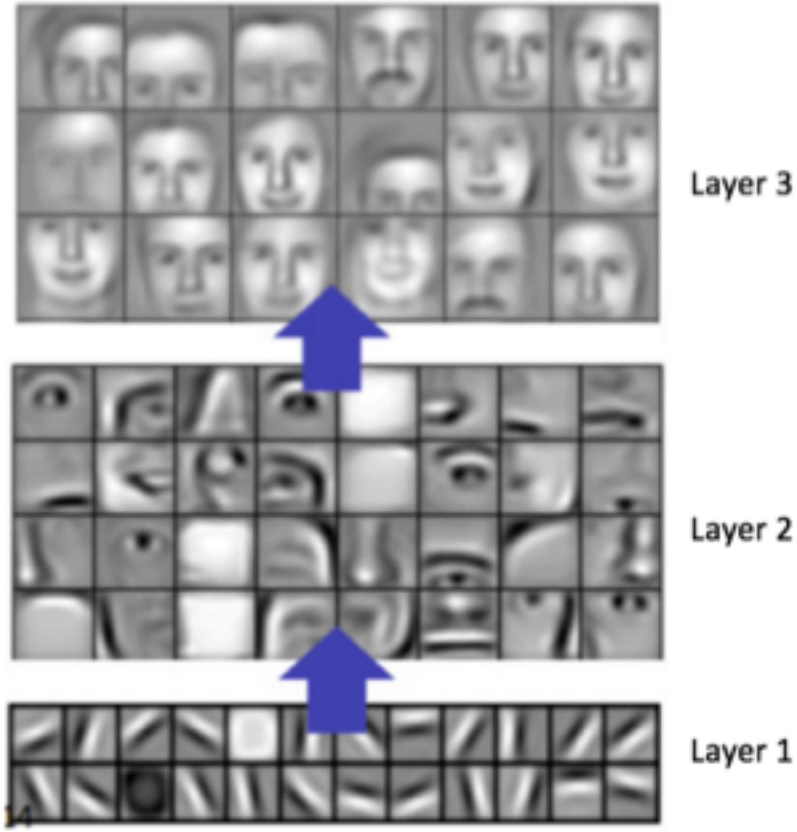
Examples of ReLu and POOL





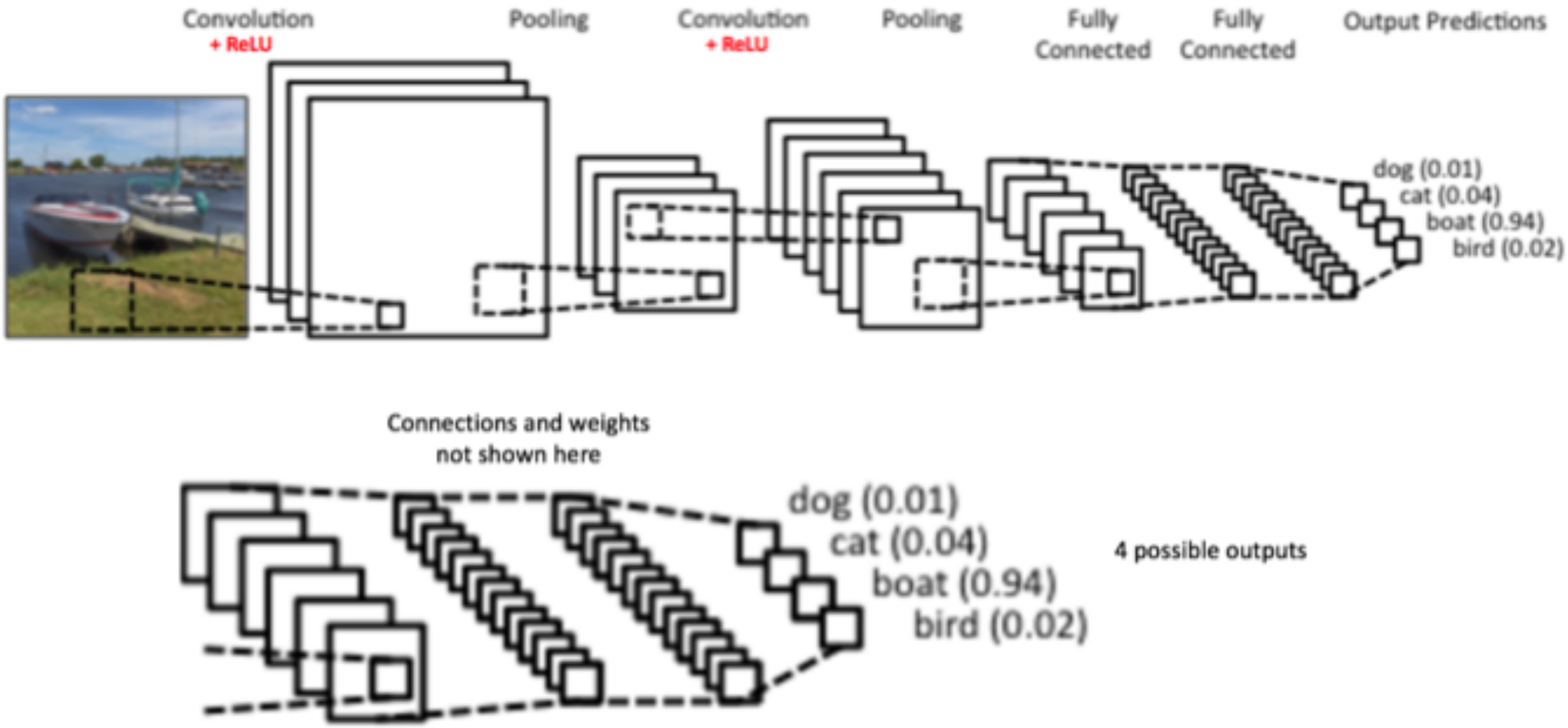
Multi-scale Feature Detection

- Through sequence of CONV/ReLU/POOL layers one uses features on smaller scale to detect features on larger scales
- This is just a cartoon picture, in practice features are not human recognizable



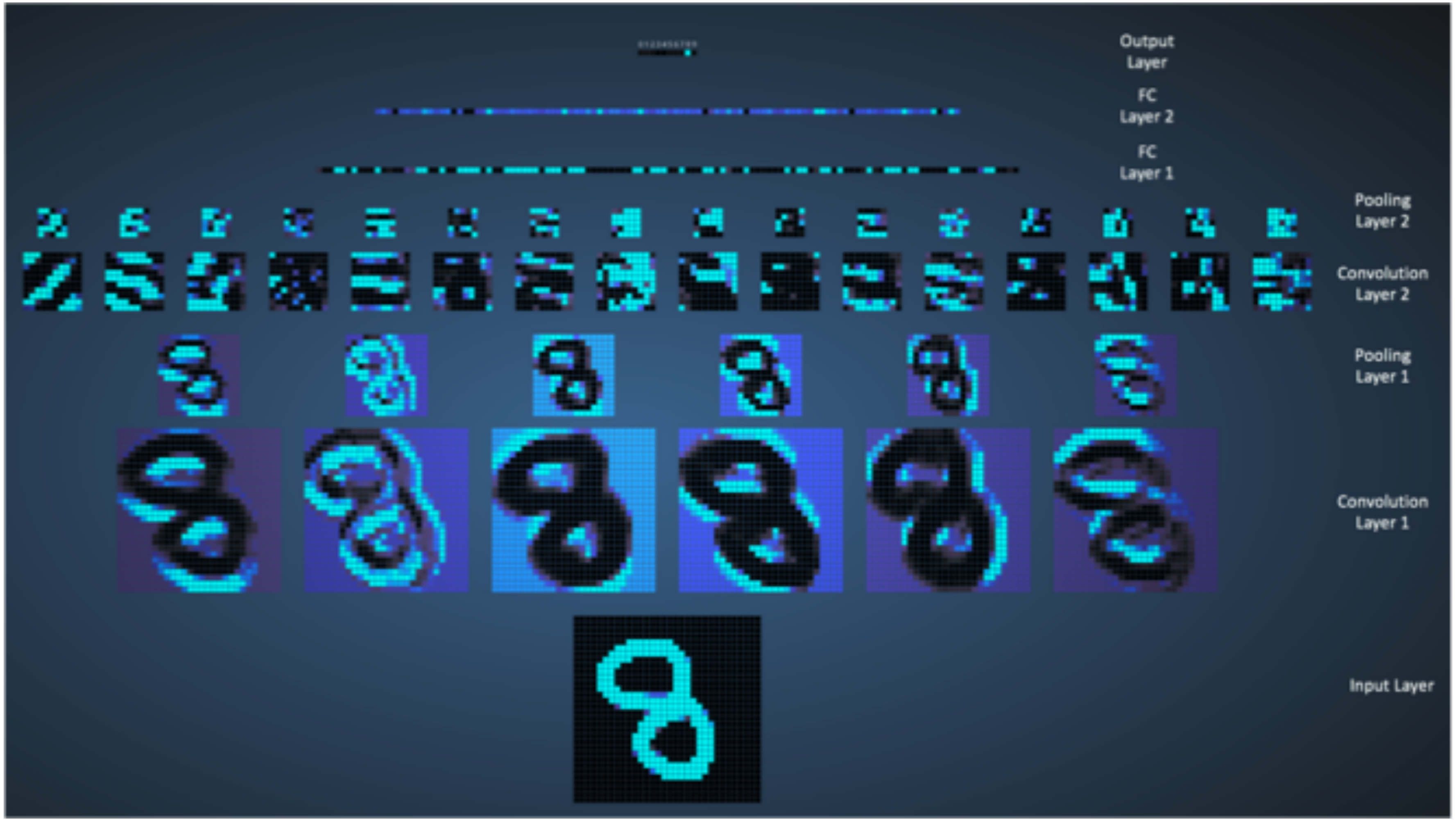
Fully Connected Layer

- So far CONV/ReLU/POOL used to define high level features
- We need to combine these to classify

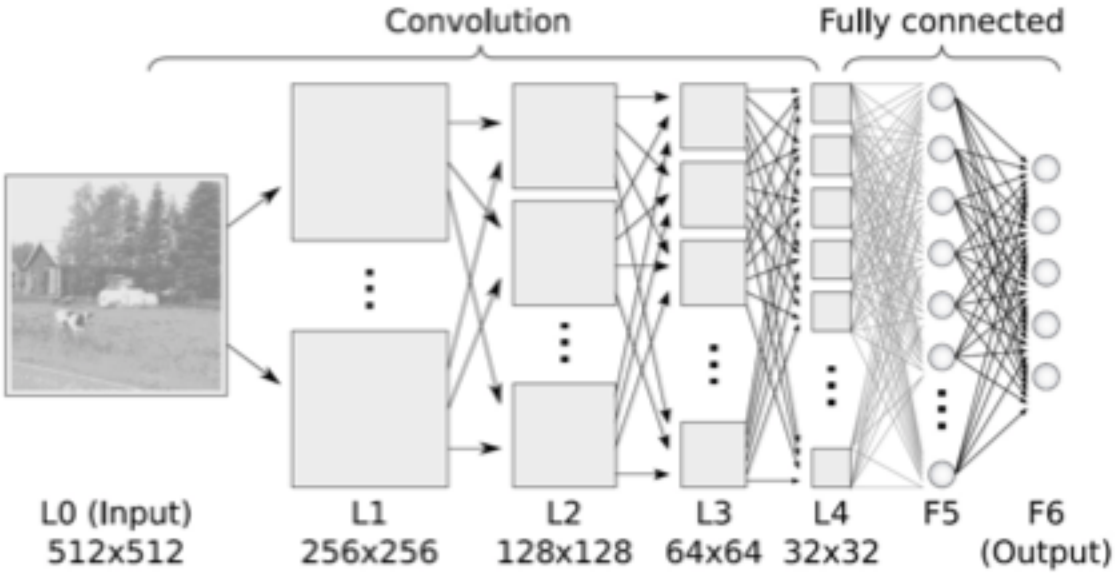


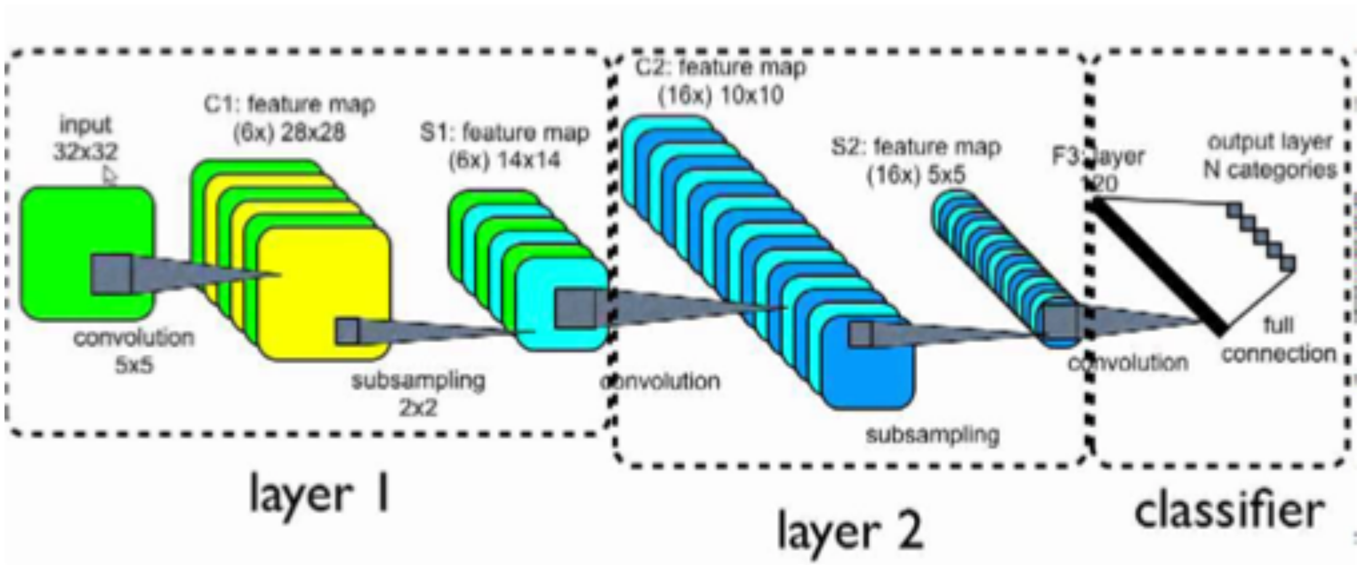
Putting it all together: Example

- 32x32x1 input, 5x5 filter, S=2, FC 120, 100



A Few More Examples



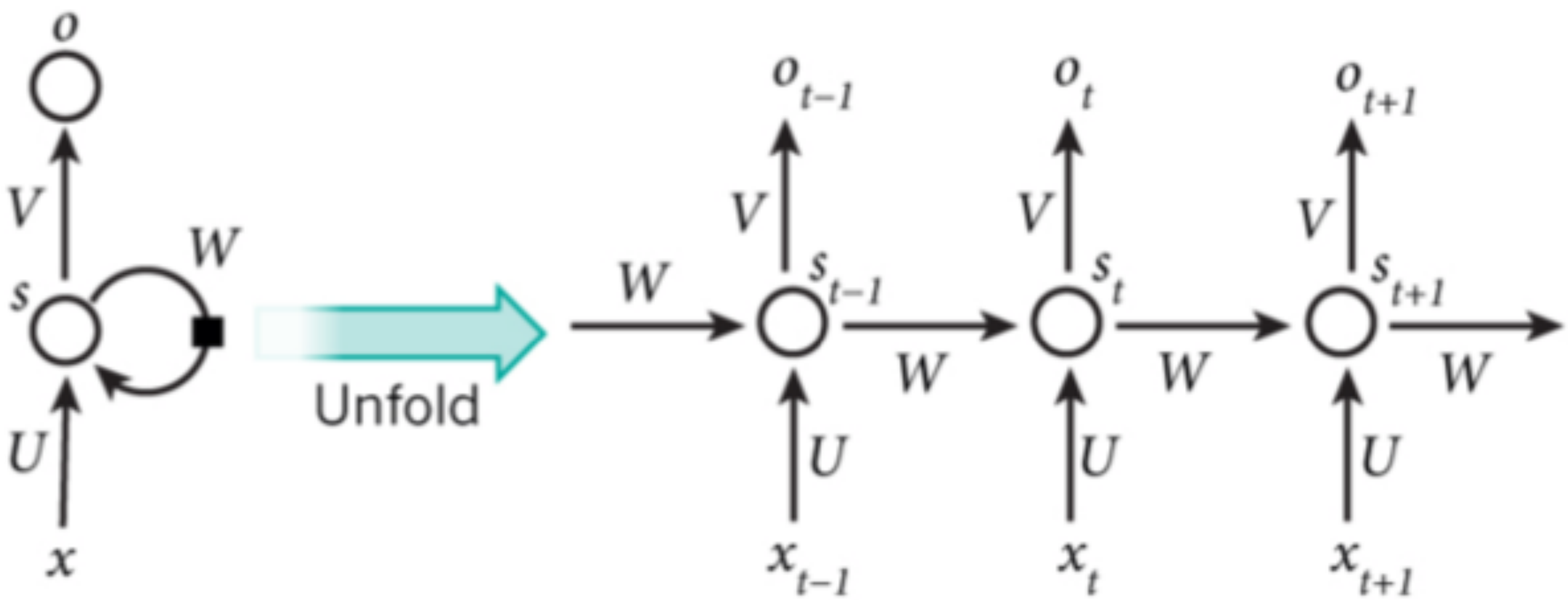


Recurrent Neural Networks

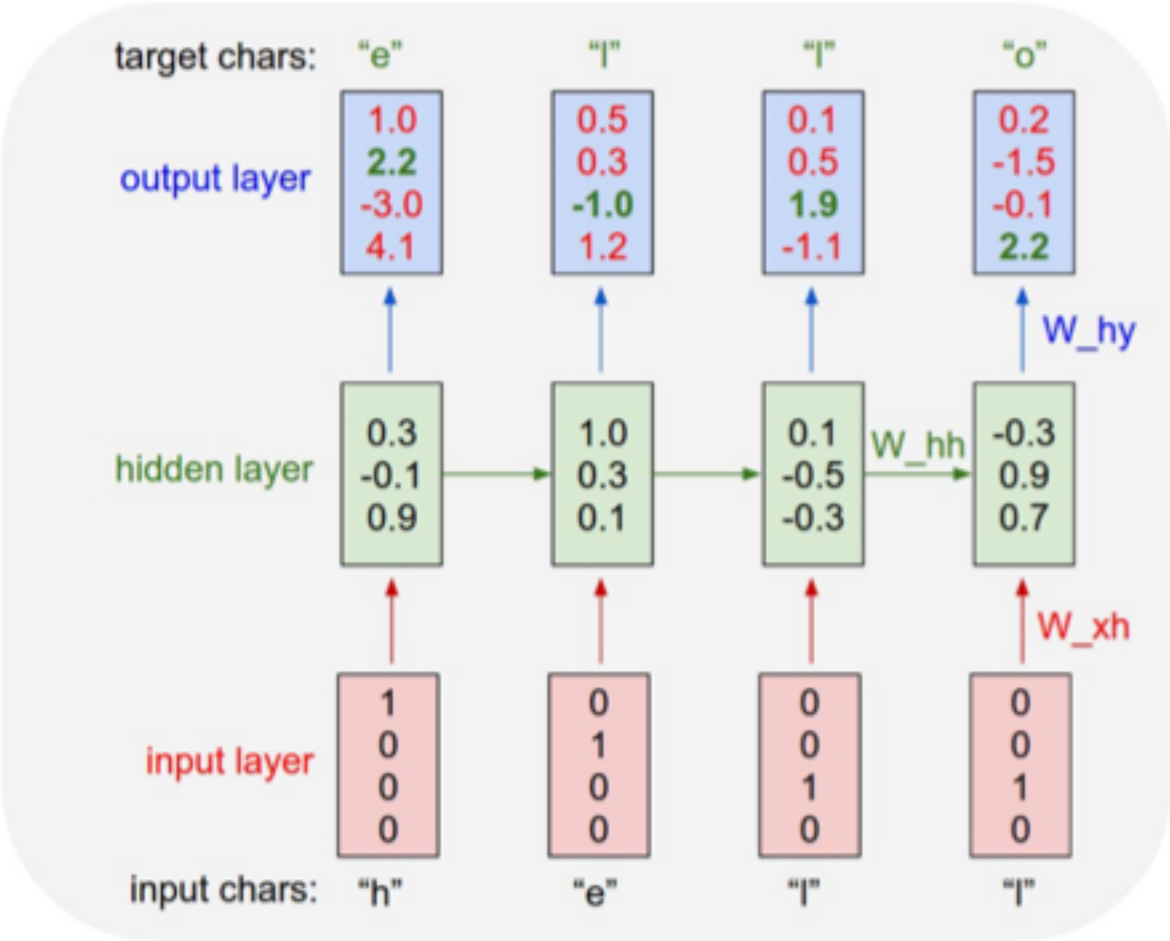
- Use sequential information: use as input new data x_t and past memory of what has been learned so far, via latent variable s_{t-1} to generate new latent variable s_t $s_t = f(Ux_t + Ws_{t-1})$, and output o_t:

$$o_t = \mathrm{softmax}(Vs_t)$$

- f is ReLu, tanh… U, V, W independent of t.



RNN Example



- Input training word is hello, 4 allowed letters helo

Example: RNN on baby names

- Note that we can use the procedure to sample: we can sample from proposed next character, feed it in and get a next proposal…
- 8000 names have been fed, here are a few that came out (and are not among 8000 inputs)

Rudi Levette Berice Lussa Hany Mareanne Chrestina Carissy Marylen Hammine Janye Marlise Jacacrie Hendred Romand Charienna Nenotto Ette Dorane Wallen Marly Darine Salina Elvyn Ersia Maralena Minoria Ellia Charmin Antley Nerille Chelon Walmor Evena Jeryly Stachon Charisa Allisa Anatha Cathanie Geetra Alexie Jerin Cassen Herbett Cossie Velen Daurenge Robester Shermond Terisa Licia Roselen Ferine Jayn Lusine Charyanne Sales Sanny Resa Wallon Martine Merus Jelen Candica Wallin Tel Rachene Tarine Ozila Ketia Shanne Arnande Karella Roselina Alessia Chasty Deland Berther Geamar Jackein Mellisand Sagdy Nenc Lessie Rasemy Guen Gavi Milea Anneda Margoris Janin Rodelin Zeanna Elyne Janah Ferzina Susta Pey Castina

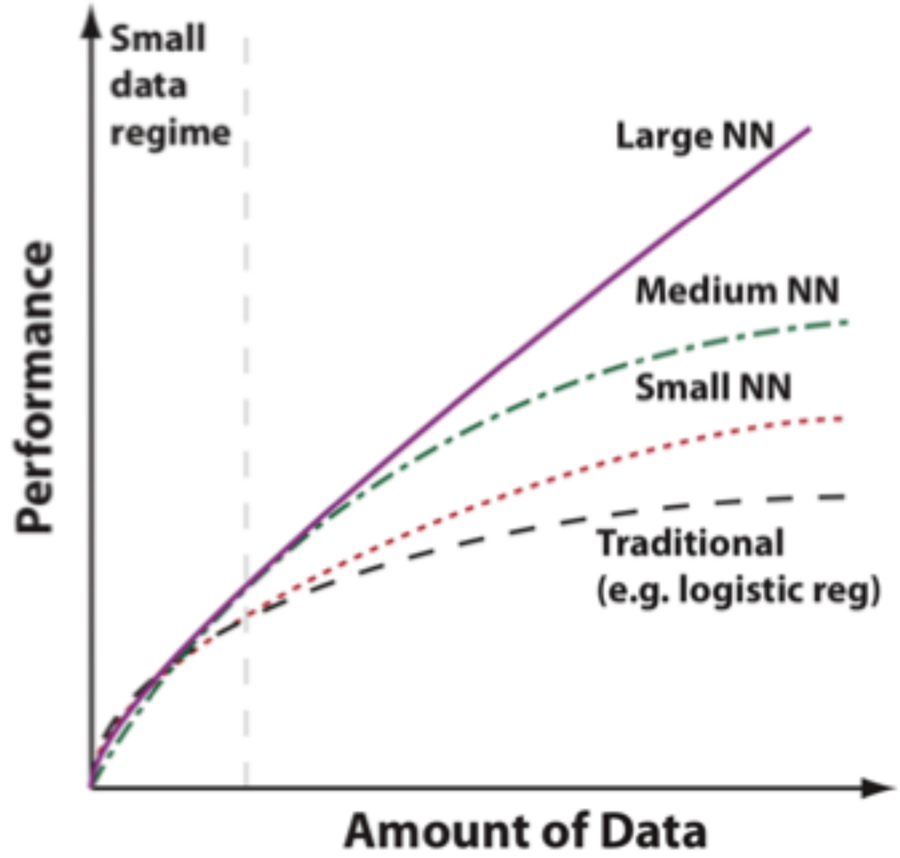
Validation and overall procedure

- We discussed this in lecture 13: typically the data are split into training, testing for hyperparameter optimization, and validation for final quantification of failure rate etc.
- Bootstrap resampling, bagging etc. can be used
- Procedure:

1. Collect and pre-process the data.
2. Define the model and its architecture.
3. Choose the cost function and the optimizer.
4. Train the model.
5. Evaluate and *study* the model performance on the validation and test data.
6. Adjust the hyperparameters (and, if necessary, network architecture) to optimize performance for the specific dataset.

Pros and cons of DNN

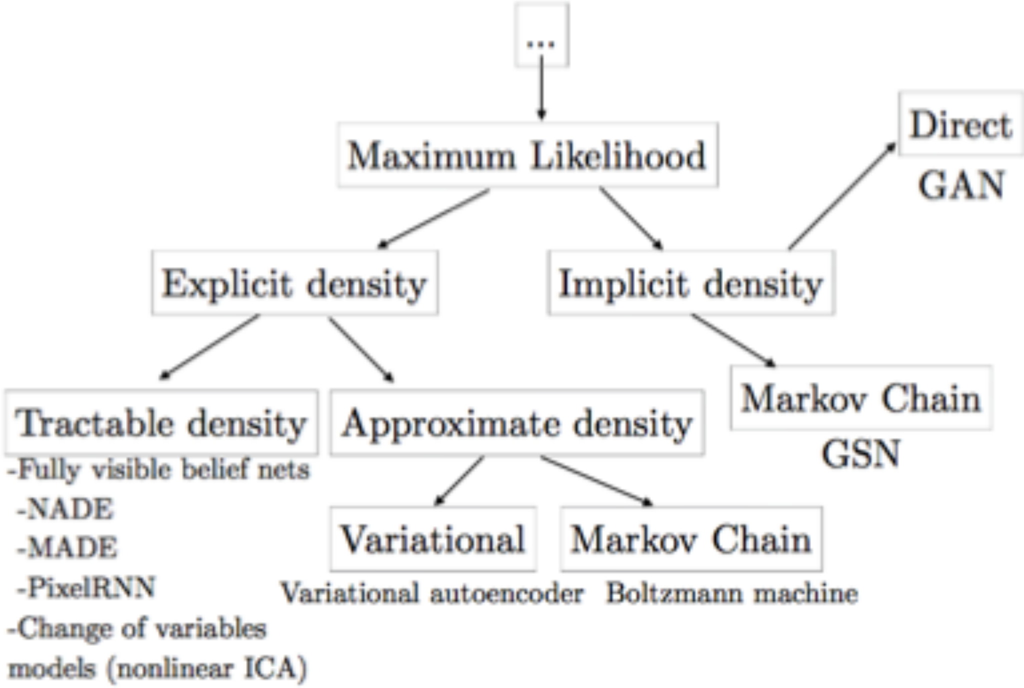
- Scale best with large data
- conversely: data hungry, performance can be worse than other methods for small data
- Need labeled data (supervised learning)
- Need homogeneous data (cannot mix pictures and time series)
- Only return classification or regression: in physics we care more about generative model and its probability distribution



Slide Andrew Ng

Generative Models can address last two problems

- Here one uses deep networks to generate new data from existing data
- One can either try to model the full PDF explicitly using tractable PDFs (NL ICA or NICE, normalizing flows), or using approximate density (Variational AutoEncoder, Boltzmann machine)
- Or one uses implicit PDF (generative adversarial networks, GAN)



Example GAN

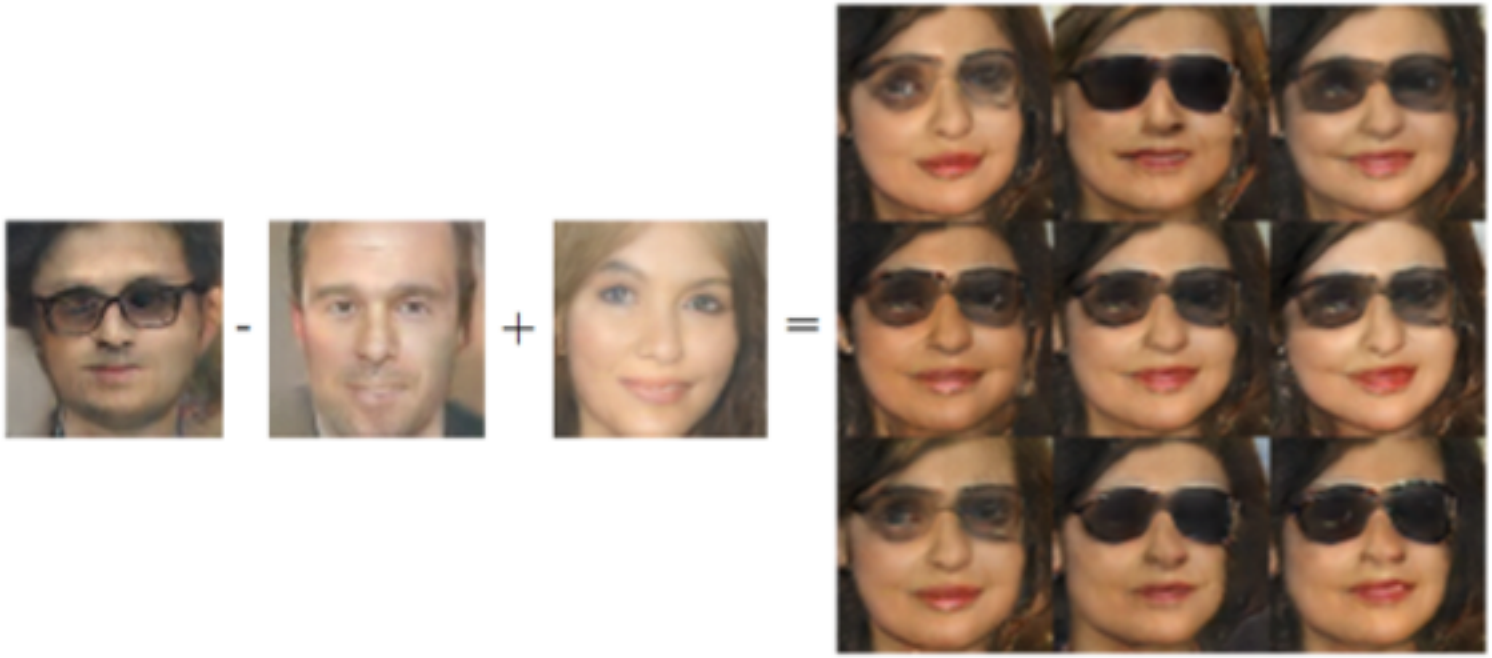


Figure 19: DCGANs demonstrated that GANs can learn a distributed representation that disentangles the concept of gender from the concept of wearing glasses. If we begin with the representation of the concept of a man with glasses, then subtract the vector representing the concept of a man without glasses, and finally add the vector representing the concept of a woman without glasses, we obtain the vector representing the concept of a woman with glasses. The generative model correctly decodes all of these representation vectors to images that may be recognized as belonging to the correct class. Images reproduced from Radford *et al.* (2015).

Summary

- Neural networks are a nonlinear extension of SVP/softmax classifiers: non-convex optimization
- They can be multi-layer (perceptrons, MLP), but fully connected MLPs rarely extend beyond 3 layers
- In contrast, convolutional NN use a hierarchy of scales, resulting in very deep networks: deep learning
- Gradient based optimization uses backpropagation algorithm and stochastic gradient descent (with various acceleration improvements).
- New advances all the time: reinforcement learning, Variational Autoencoders, generative adversarial nets…
- Some driven by new algorithms, but much also simply by better and faster computers (GPUs+Moore’s law)

Literature

- http://cs231n.github.io
- Mehta etal https://arxiv.org/pdf/1803.08823.pdf
- http://www.deeplearningbook.org