TEXAS
The University of Texas at Austin

TOPIC 1 NEURAL NETWORKS

Neural Networks

- Multi-Level Perceptrons
- Convolutional Neural Networks

- In 2015 Google released an opensource tool for training neural nets called TensorFlow for python
- TensorFlow does all the backend work
- Keras is a modeling package that makes it easy to formulate neural nets and then passes them to TensorFlow for training
- conda install keras
- Or just do everything in colab

External Resources

- I don't know a great (free) text resource that explains this content very well
- I did find some pretty good youtube videos though
- https://www.3blue1brown.com/videos
  - Scroll down to the 'Neural networks' section
  - There is a 4 part video series that explains multi-level perceptrons well
- https://www.youtube.com/watch?v=FmpDIaiMIeA
  - This does an OK job at explaining convolutional neural networks

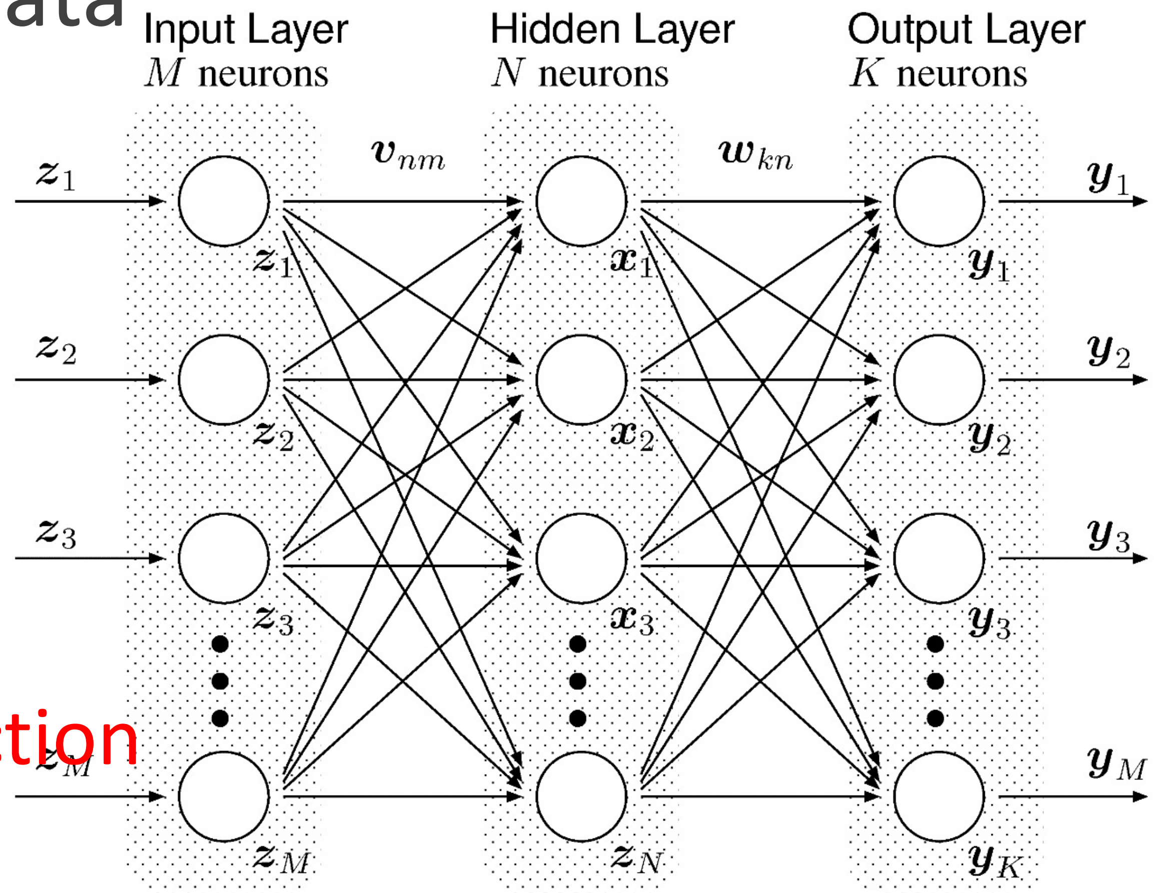
Neural Networks

- Neural Networks have become increasingly popular in the past couple decades or so
- The application area that has had the most success is image classification
  - What address is written on this envelope?
  - Hotdog or Not Hotdog?
- We will work with a data set of 60,000 images (28x28 pixels each) of handwritten numbers 0-9
  - Our goal is to take the image as input and output which number we think it is
  - The images come from the post office
  - MNIST – The “Hello World!” of image classification

Neural Networks

- A neural net is just a fancy tool for non-linear regression and classification
- Much like trees, there's no closed form solution to the parameters, but we can still get close to them
- The tools we use to fit neural networks are
  - Back Propagation
  - Stochastic Gradient Descent
- These are simple tools that are very powerful
- The first type of neural network we'll study is called
  - Multilevel Perceptron Network
  - Fully Connected (dense) Layers

- A neural network is simply a composition of several functions with parameters we estimate from data
  - Input Layer
  - Neurons
  - Hidden Layers
  - Weights
  - Biases
  - Activation Function
  - Output Layer



Neural Networks

- Our job is to find all the weights and biases
  - Assume network structure and activation functions are given
- The number of neurons in the first layer is the dimension of the input data (number of regressors: m)
- No limit to the number of hidden layers or neurons
  - More layers and neurons means more parameters
- The number of neurons in the last layer is the dimension of the output data
  - Usually 1 for quantitative variables, like regression
  - Number of categories for classification
    - Output is "probability" of each category

Neural Networks

- There are M input neurons
- Each input neuron has a weight associated with each of the N neurons on the second layer, $v_{n,m}$
- Each of the N neurons on the second layer also have a bias, b_n
- Every neuron on the second layer shares one activation function, $a_2(\cdot)$
- The neurons on layer 2 are equal to
  - $x_n = a_2\left(b_n + \sum_{m=1}^{M} v_{n,m} z_m\right)$

Neural Networks

- There are N neurons on layer 2
- Each layer 2 neuron has a weight associated with each of the K neurons on the third layer, $w_{k,n}$
- Each of the K neurons on the third layer also have a bias, c_k
- Every neuron on the third layer shares one activation function, $a_3(\cdot)$
- The neurons on layer 3 are equal to
  - $y_k = a_3\left(c_k + \sum_{n=1}^{N} w_{k,n} x_n\right)$
- And so on…

Neural Networks

- This is highly non-linear and non-convex because the activation functions are non-linear
- Some popular activation functions are
  - ReLU: $a(x) = \max(x, 0)$
  - Sigmoid: $a(x) = \frac{1}{1+e^{-x}}$
  - SoftMax: $a(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}$
  - SoftPlus: $a(x) = \log(1 + e^{x})$
  - …

Neural Networks

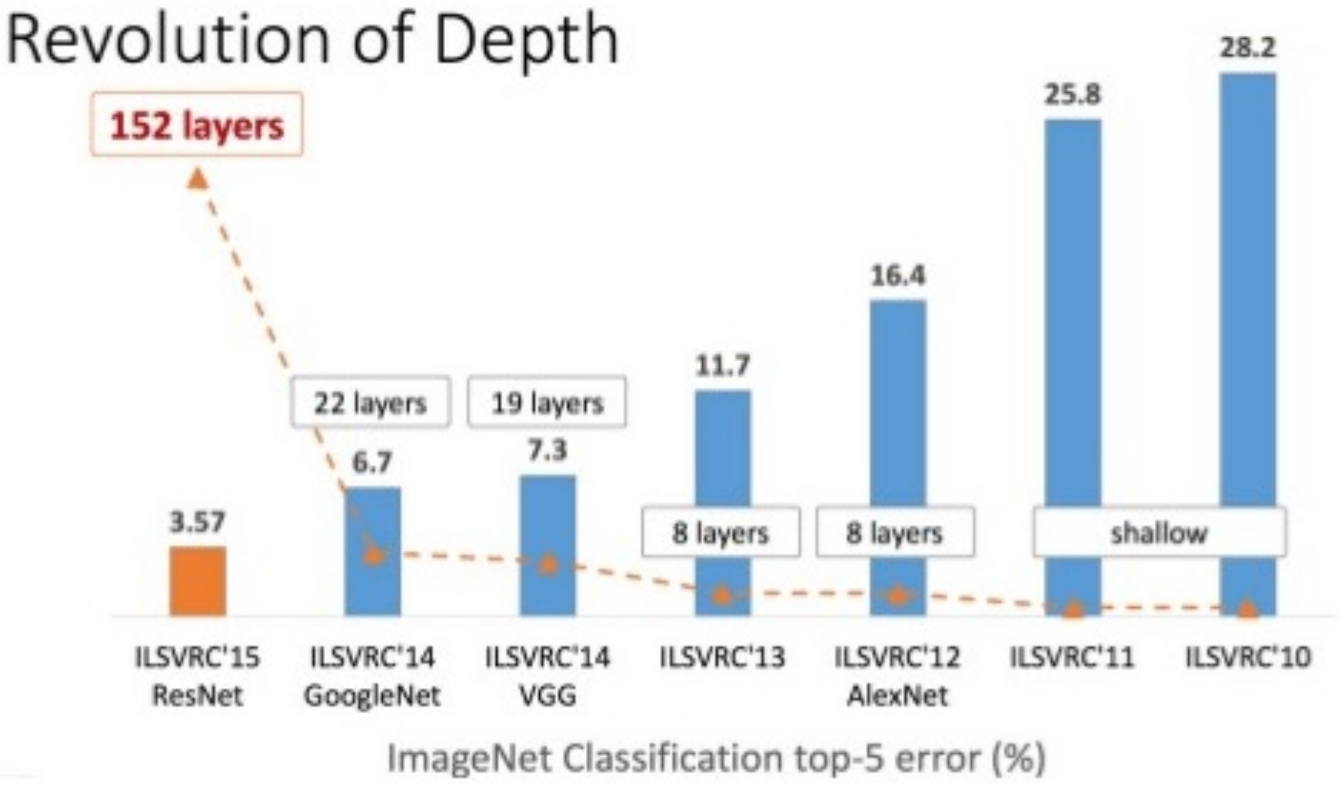
- To find the weights and biases we must define an objective function (loss function)
- The most popular loss function is mean-squared-error
  - For classification problems: one hot encoding
- Suppose the output layer has J nodes $\hat{y}_j$
- The data output is y_{ji}
- $Loss = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{J}\left(y_{ji} - \hat{y}_{ji}\right)^2$
- The objective is to minimize the MSE by finding the optimal weights and biases
- There are many other popular loss functions
  - Categorical Cross Entropy: $\frac{-1}{n}\sum_{i=1}^{n}\sum_{j=1}^{J} y_{ji}\log\left(p_{ji}\right)$

Back Propagation

- The most common way to do this is back propagation
  - This is the NN word for stochastic gradient descent
- Start with a random guess of weights and biases
- Calculate the derivative of the loss function with respect to every weight and bias
  - The vector of these derivatives is the gradient
- Take a small step in the direction of the -gradient
  - We talked about variations of this last semester
- This relies on being able to calculate the derivatives
  - It turns out this is just the chain rule

Depth Revolution

- In the early 2010’s teams participating in the ImageNet competition started using deep neural networks
  - They drastically changed the study of NN’s



Depth Revolution

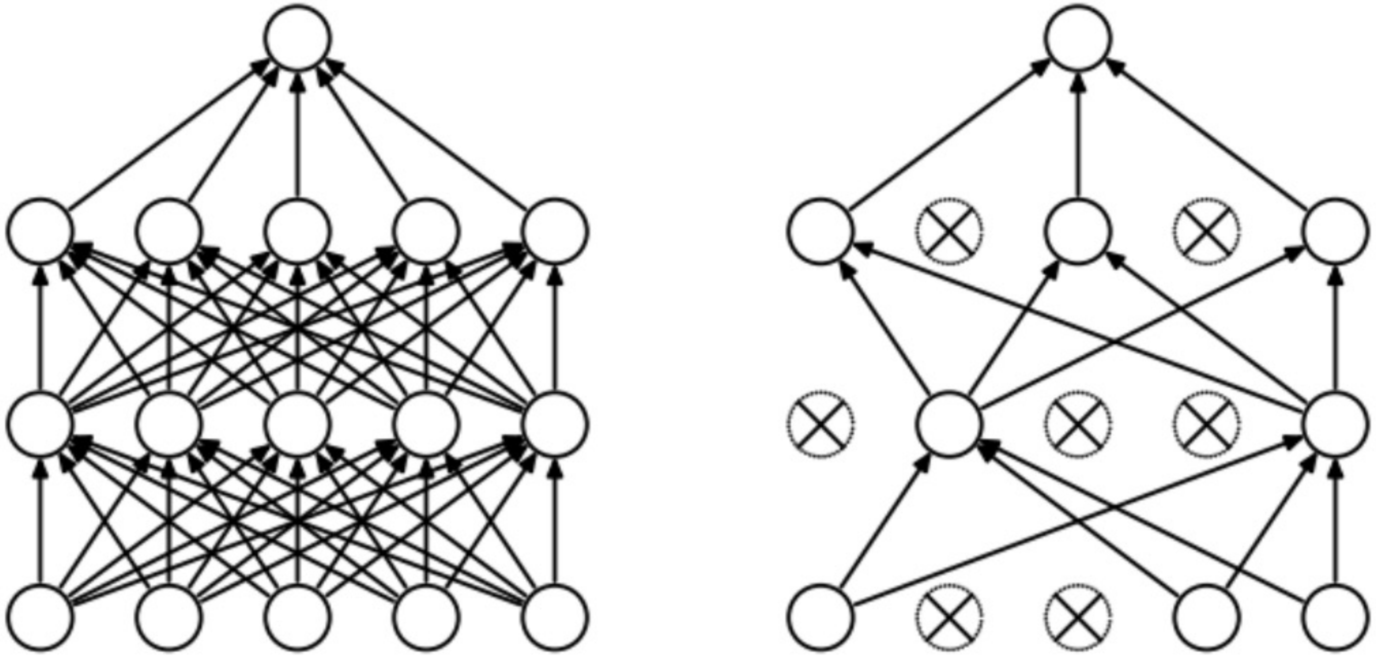
- Neural Nets get their name from the brain
- People used to use indicator functions as activation
  - $I(x) = \begin{cases} 1, x \geq 0 \\ 0, \quad else \end{cases}$
- Training this is hard: the derivative is always zero
- The sigmoid was used as a smooth approximation
  - The derivative is always in (0,1)
- If you have a deep network you have to multiply lots of numbers less than 1: vanishing derivative
- This is why ReLU is popular – derivative is 1 or 0
- We can now train deep networks!

Neural Networks

- With deep networks the number of parameters to estimate grows very quickly
  - This easily leads to over fitting
- There are a couple ways to avoid this
- We can use a Lasso or Ridge term
  - Penalize our objective with the sum of squared or absolute valued parameters
  - In the NN space these are called regularizers
- Each layer can have it's own penalty parameter, λ

Dropout

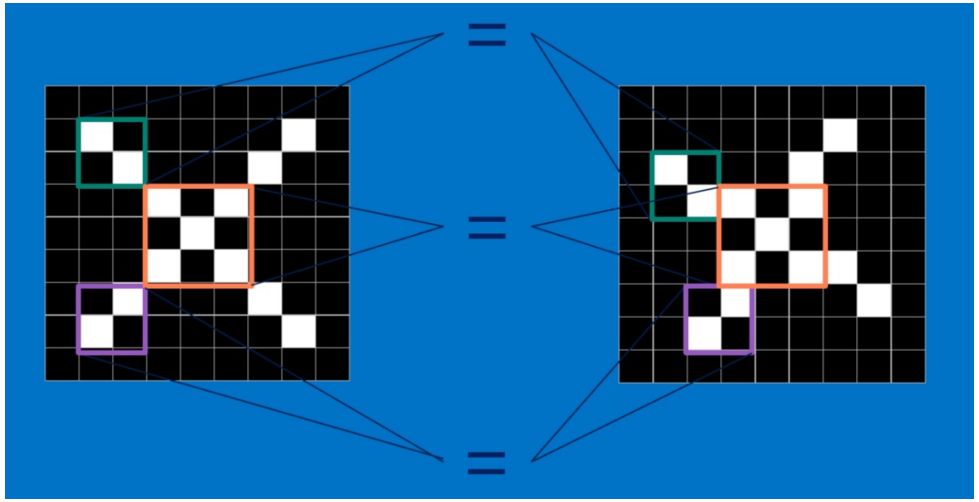
- A popular alternative in NN's is called dropout
  - On a step of SGD just randomly set some neurons equal to zero
  - On another step set a different group of neurons equal to zero

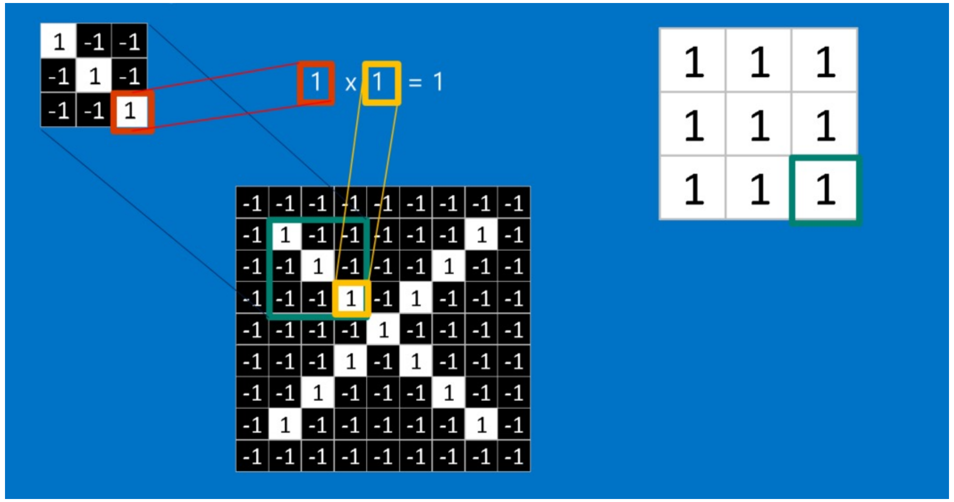


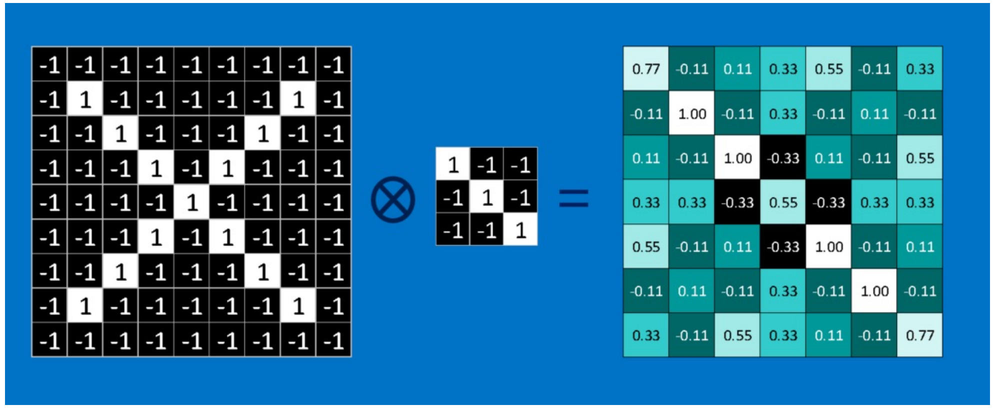
(a) Standard Neural Net (b) After applying dropout.

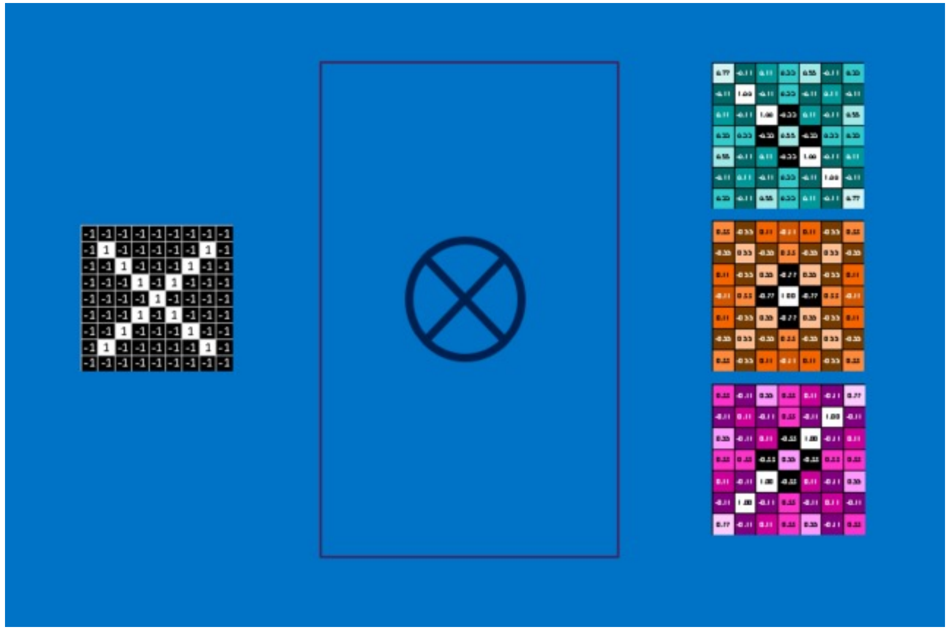
Convolutional Neural Networks

- There are other types of layers besides dense
- A layer used for images is called a convolutional layer
  - Apply a set of filters to an image
  - Take the output of the filters as the input of a dense neural network
- A filter is just a really small image, 3x3, 4x4, … pixels
- We apply the filter by going over every 3x3 set of pixels in the original image and seeing how close the original image is to the filter in that region
  - Record the closeness score everywhere
  - This is then the output of the layer



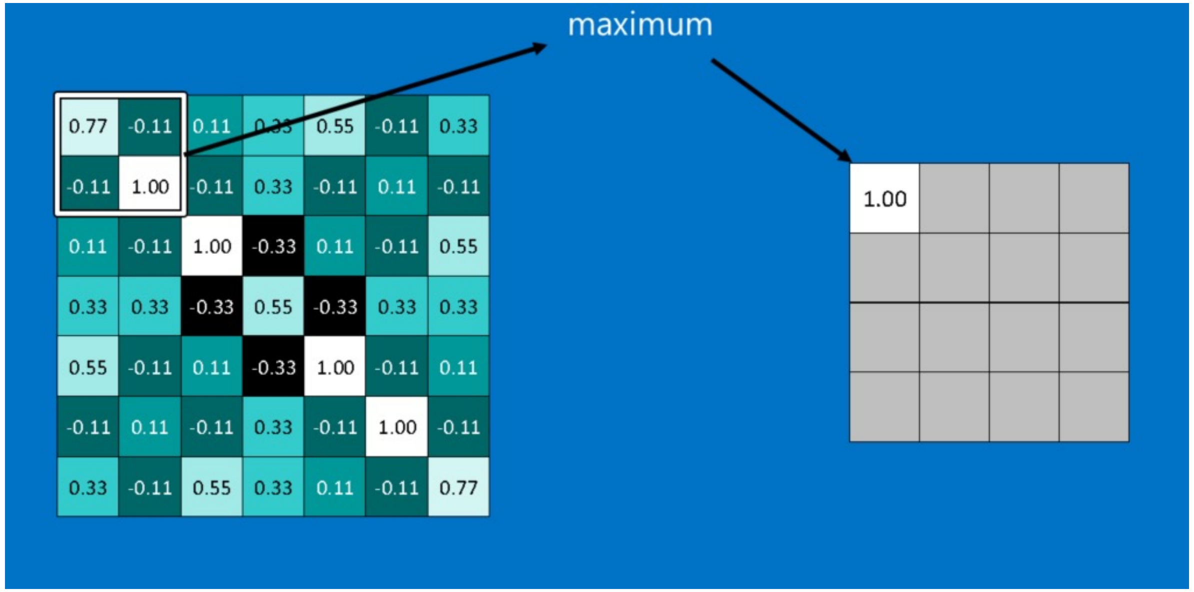

1 -1 -1
-1 1 -1
-1 -1 1
1 x 1 = 1
1 1 1
1 1 1
1 1 1

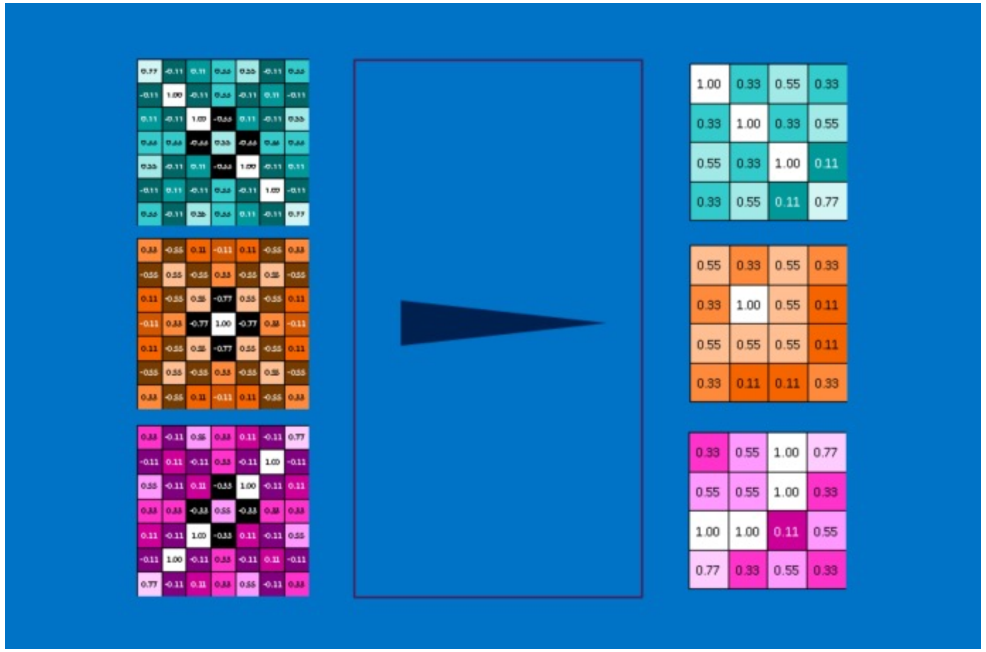

-1 -1 -1 -1 -1 -1 -1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 -1 -1 1 -1 -1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1
⊗
1 -1 -1
-1 1 -1
-1 -1 1
=
0.77 -0.11 0.11 0.33 0.55 -0.11 0.33
-0.11 1.00 -0.11 0.33 -0.11 0.11 -0.11
0.11 -0.11 1.00 -0.33 0.11 -0.11 0.55
0.33 0.33 -0.33 0.55 -0.33 0.33 0.33
0.55 -0.11 0.11 -0.33 1.00 -0.11 0.11
-0.11 0.11 -0.11 0.33 -0.11 1.00 -0.11
0.33 -0.11 0.55 0.33 0.11 -0.11 0.77



Convolutional Neural Networks

- By applying 3 filters we have tripled the amount of data we have!
- This means in our full network we'll need a lot of neurons on our dense layer
- A common way to fix this is max pooling
- We take a small box and cycle it through the output of the filters
- Everywhere the box goes, we just remember the largest number in the box
  - Throw everything else away
  - Don't let the boxes overlap


maximum
0.77 -0.11 0.11 0.33 0.55 -0.11 0.33
-0.11 1.00 -0.11 0.33 -0.11 0.11 -0.11
0.11 -0.11 1.00 -0.33 0.11 -0.11 0.55
0.33 0.33 -0.33 0.55 -0.33 0.33 0.33
0.55 -0.11 0.11 -0.33 1.00 -0.11 0.11
-0.11 0.11 -0.11 0.33 -0.11 1.00 -0.11
0.33 -0.11 0.55 0.33 0.11 -0.11 0.77
1.00


1.00 0.33 0.55 0.33
0.33 1.00 0.33 0.55
0.55 0.33 1.00 0.11
0.33 0.55 0.11 0.77
0.55 0.33 0.55 0.33
0.33 1.00 0.55 0.11
0.55 0.55 0.55 0.11
0.33 0.11 0.11 0.33
0.33 0.55 1.00 0.77
0.55 0.55 1.00 0.33
1.00 1.00 0.11 0.55
0.77 0.33 0.55 0.33

Convolutional Neural Networks

- For the convolutional step, where did we find the filters?????
- The neural network treats the entries in each pixel of the filters as parameters to be learned
- Back propagation then learns what the filters are
- We still have to pick the size and number of filters