Visualization in AI

Neural Networks, NLP, & Network Analysis

A (biased) tour

Jeff Greenberg, Feb 2017

Contact:

http://jeffrey-greenberg.com
jeff@inventivity.com
@JeffGreenberg
http://www.peerwell.co

First off

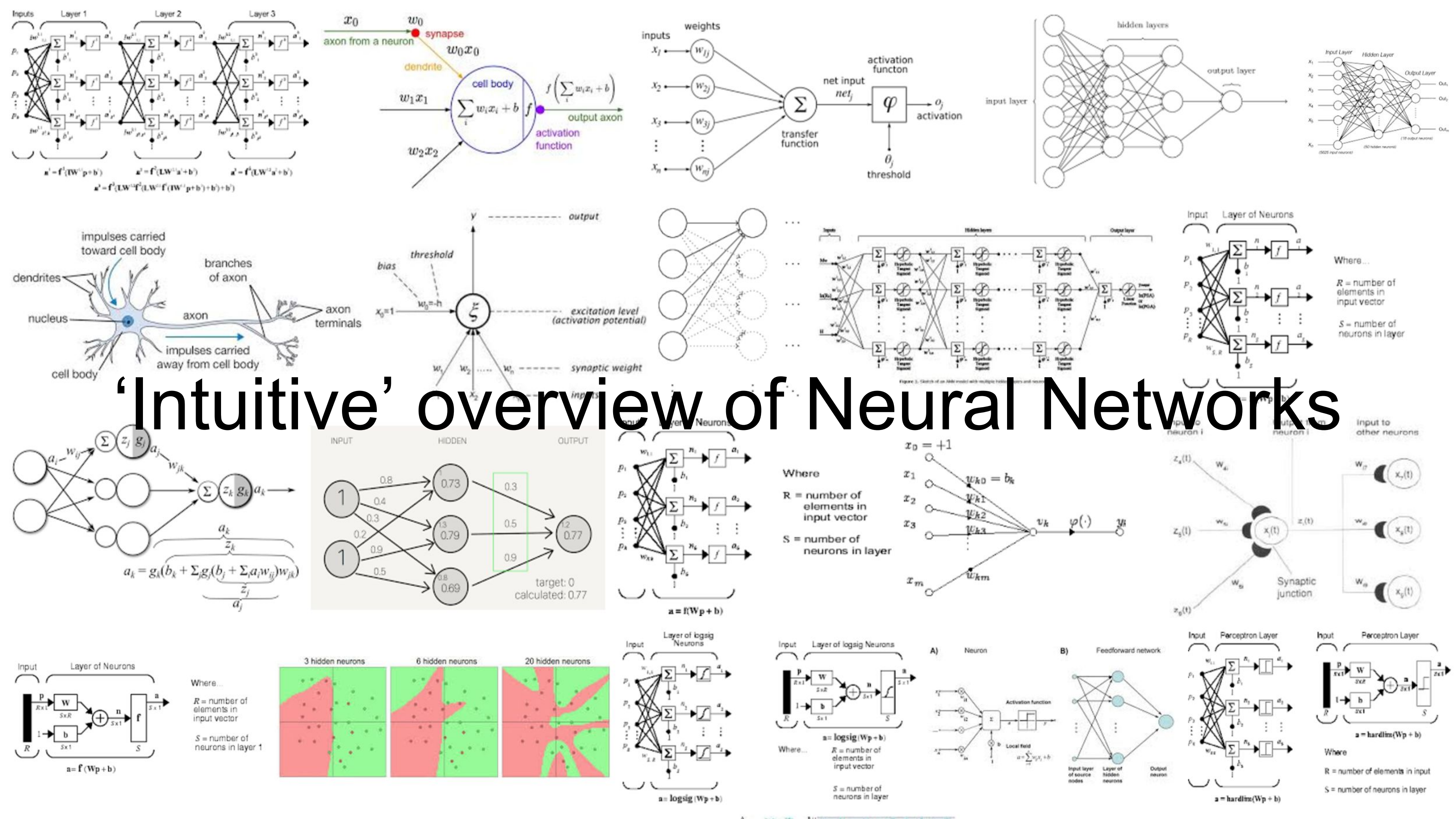
A little ai background history:

- feature detection (rules, edges)
- search space techniques (alpha/beta, mcts, GA/GP)
- Go
- Ultrasound imaging challenges then & now - detection

- http://playground.tensorflow.org/#activation=tanh&batchSize=25&dataset=spiral&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=5&networkShape=6,6,6&seed=0.79335&showTestData=false&discretize=false&percTrainData=80&x=true&y=true&xTimesY=true&xSquared=true&ySquared=true&cosX=false&sinX=true&cosY=false&sinY=true&collectStats=false&problem=classification&initZero=false&hideText=false

Let’s look at some nets… A simplified DNN in your browser...

‘Intuitive’ overview of Neural Networks

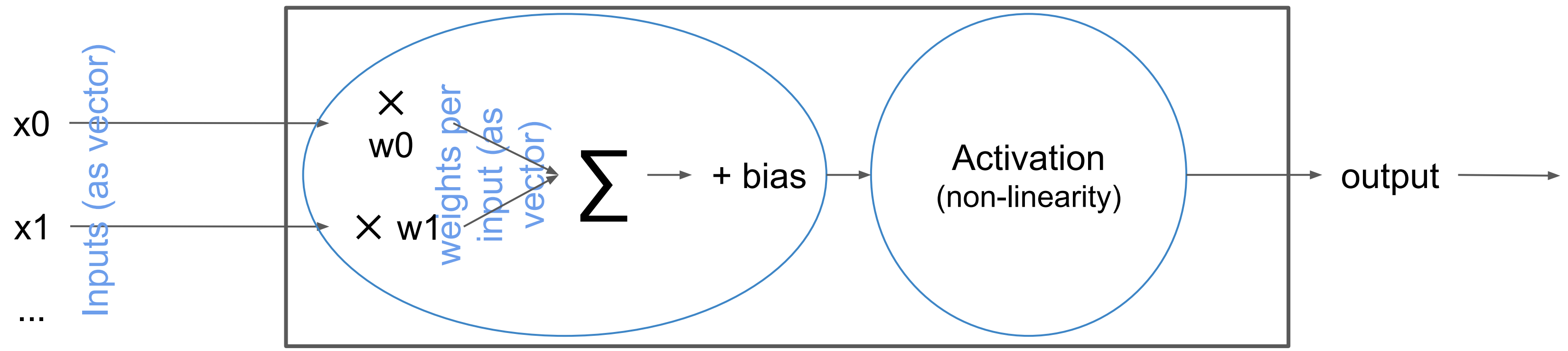


Neurons & Neuron Layers

Inputs & Outputs

Neuron - Basic processing element

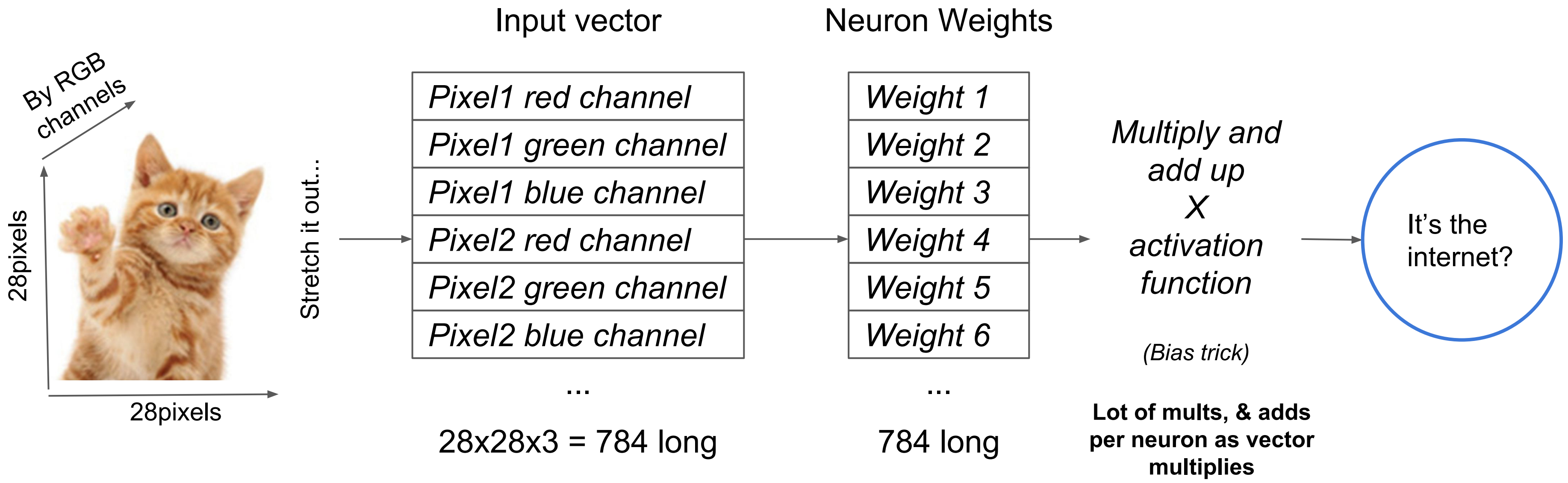
Output = ActivationFunc((Input vector * NeuronWeight) + NeuronBias)



- Each field of the input vector is multiplied by a corresponding weight, and then summed.
- That value is provided to an function, the activation function, that typically results in a non-linear output.

Set the right weights...

Output = ActivationFunc((Input vector * NeuronWeight) + NeuronBias)



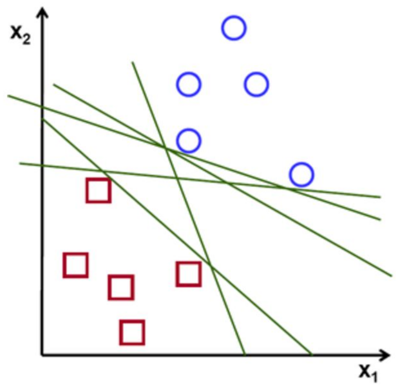
Weight picking to get a desired result...

Output = $_{\text{ActivationFunc}}$((Input vector * NeuronWeight) + NeuronBias)

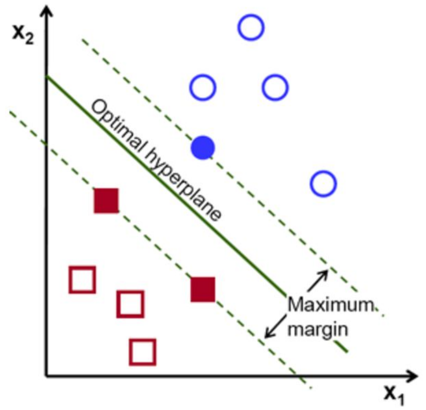
This neuron equation is analogous to the line equation for single input:

$$y = mx + b$$

Where one could pick m (slope) and b (offset) to position a line against whatever x is input and y output… like this:



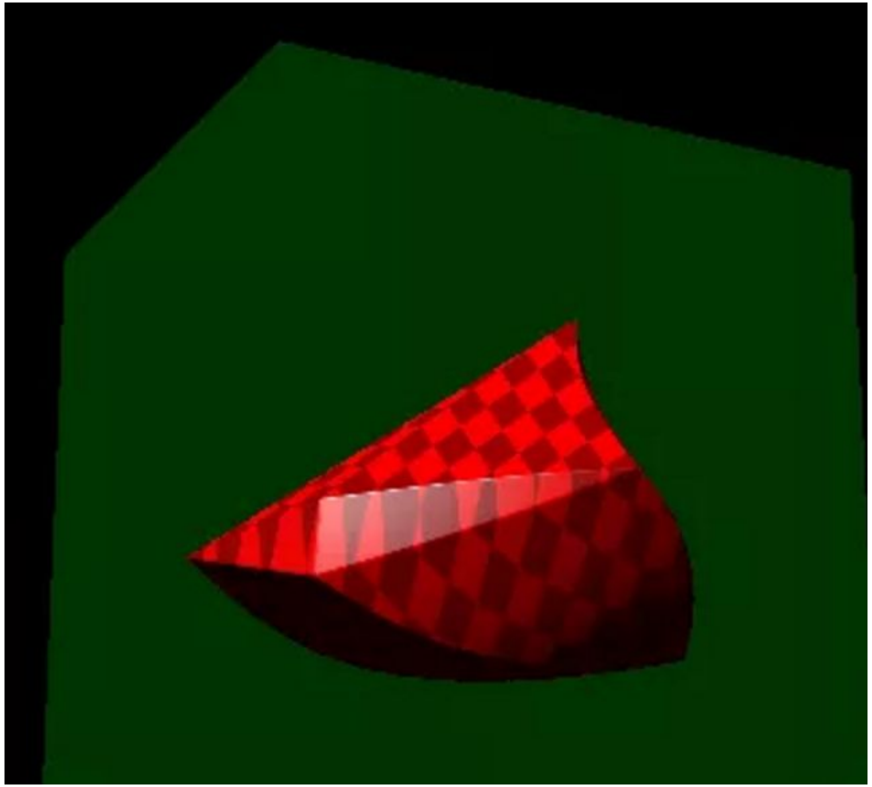
To get this =>



A layer of neurons can capture more complexity...

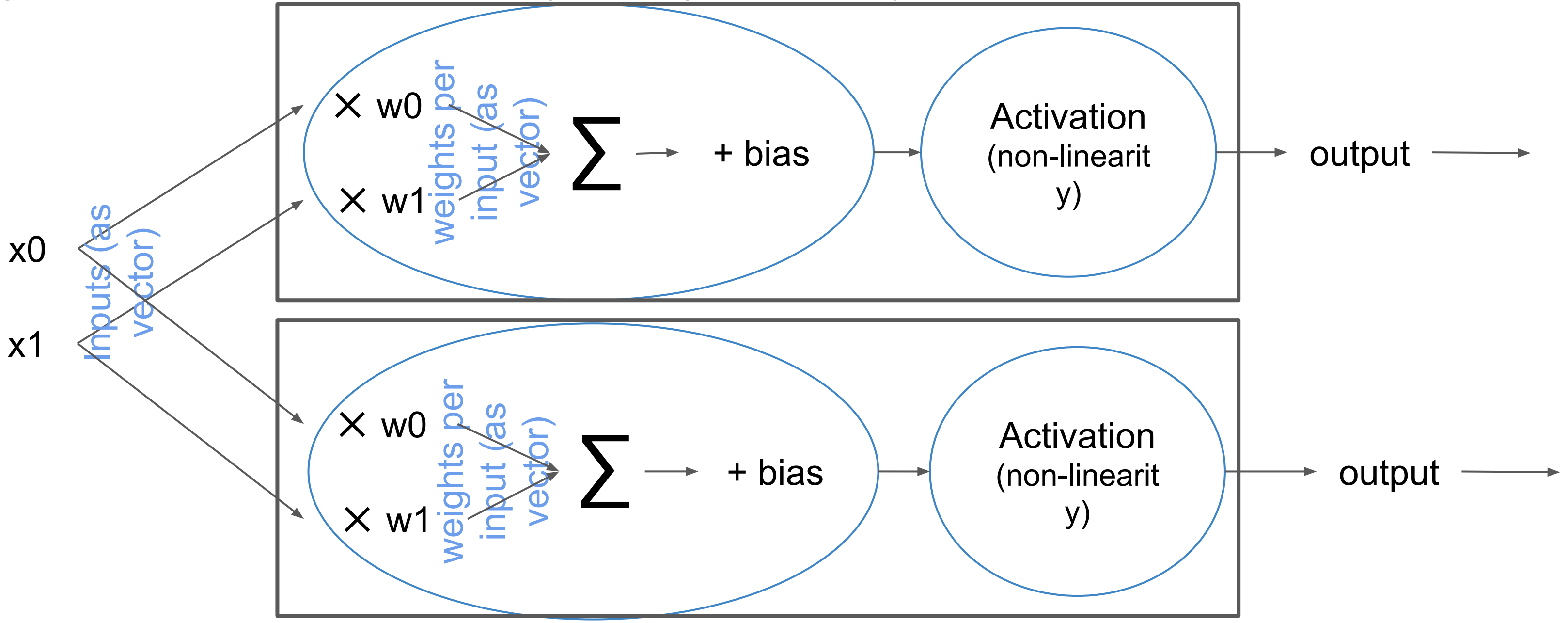
Each of the classifiers can be combined to make more subtle and complex classifications.

Whereas alone, they provide a single hyper-plane slice, in unison they can be combined into hyper-plane shapes…



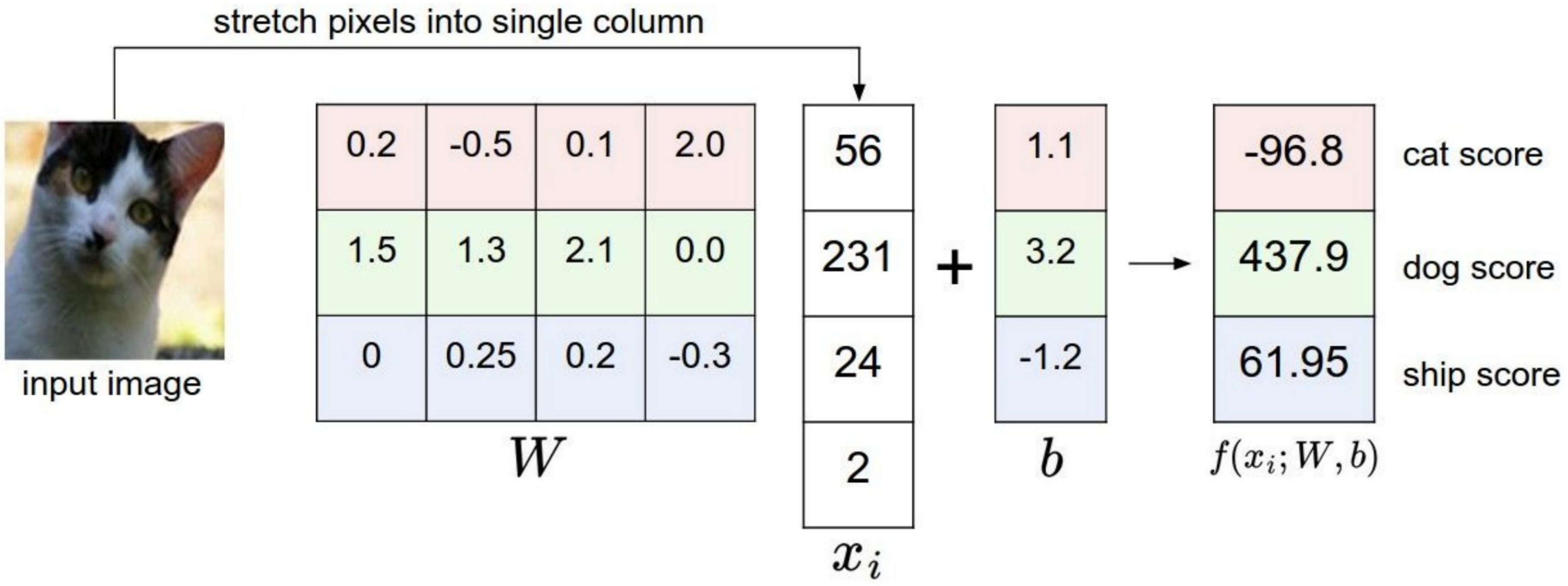
Layers: differing only by their weights

A set of individual neurons sharing the same Input vector, differing only by their weights, would each respond (output) differently to that input.



x0
x1
Inputs (as vector)
× w0
× w1
weights per input (as vector)
Σ
+ bias
Activation (non-linearity)
output
× w0
× w1
weights per input (as vector)
Σ
+ bias
Activation (non-linearity)
output

Layering up can yield computational efficiencies



From cs231n Karpathy...

Layers of layers

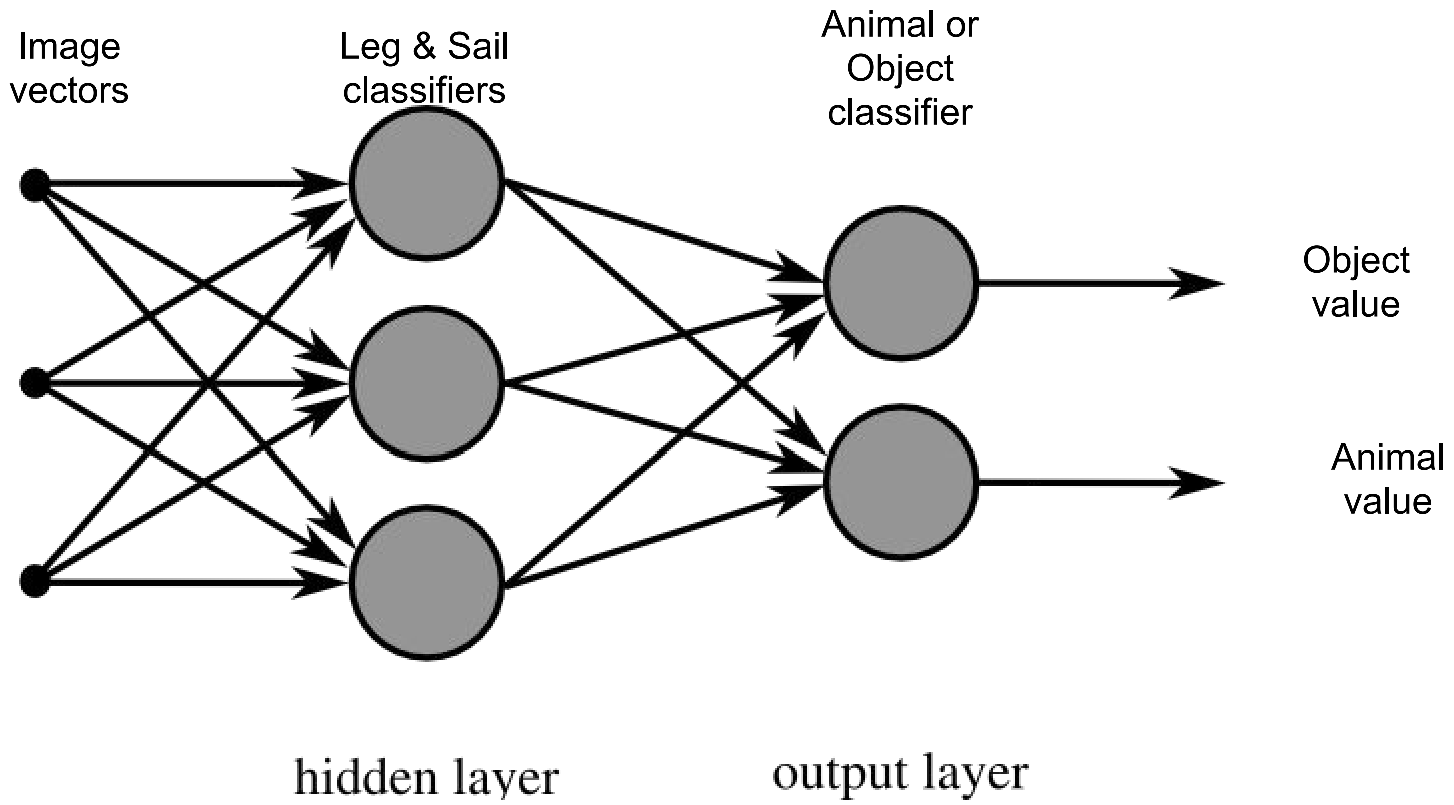
Can be efficiently computed, given the weights and initial input:

```
# Random input (for now)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
# forward-pass of a 3-layer neural network with
activationFunc_sigmoid = lambda arg: 1.0/(1.0 + np.exp(-arg)) # activation function (use sigmoid)
layer1_output = activationFunc_sigmoid( np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
layer2_output = activationFunc_sigmoid( np.dot(W2, layer1_output) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, layer2_output) + b3 # output neuron (1x1)
```

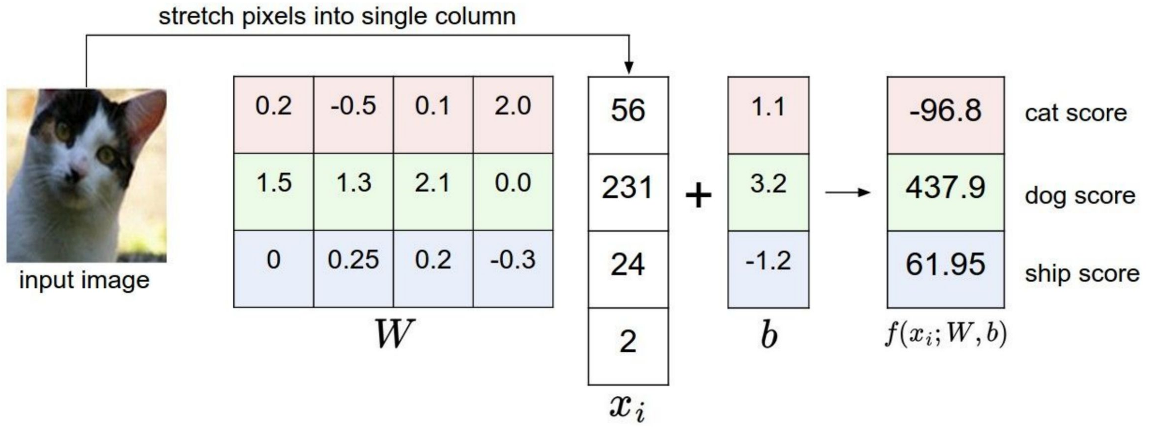
(from Karpathy CS231n)

Layers of layers...

Take layer from prior slide as our Image Classifier layer here...



Picking the best output from the final layer...



- Look for maximum value
  - SVM support vector machine
- Look for probability
  - Softmax

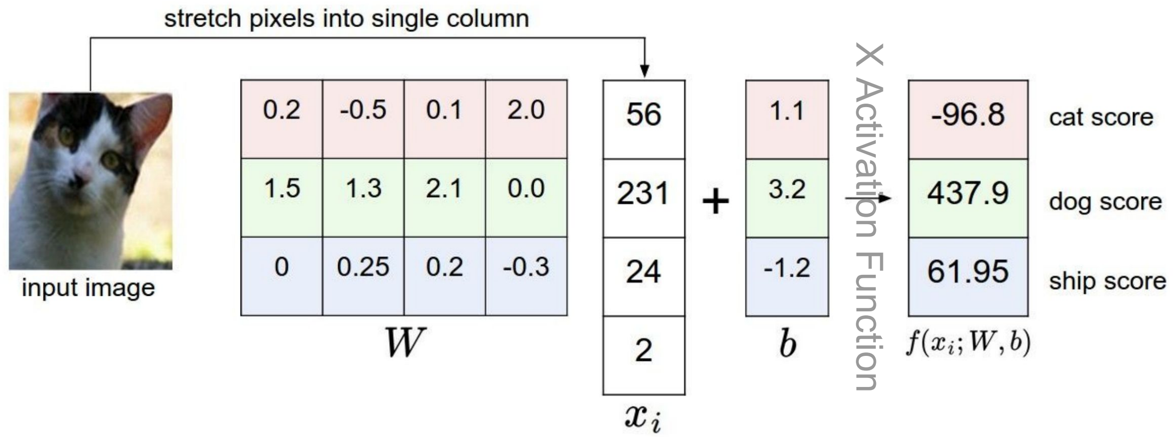
Typically the final, aka ‘output’, layer conforms to the classifications we want.

The output values per neuron per performs one of these

Setting weights automatically…

(learning from data)

If the cat score is wrong, you tweak the weights...



How to do this efficiently and well is one of the two prime advances in recent AI.

The other is insights into arrangements and variations of layers to get good results with all kinds of input.

First compute how off you are...

Given training data where each data item is paired with a known result...

Compute an error value, given that you have the expected value…

    E.g. Absolute value, squaring, distance formulas

We can use this to keep track of the error for the entire data set...

Then

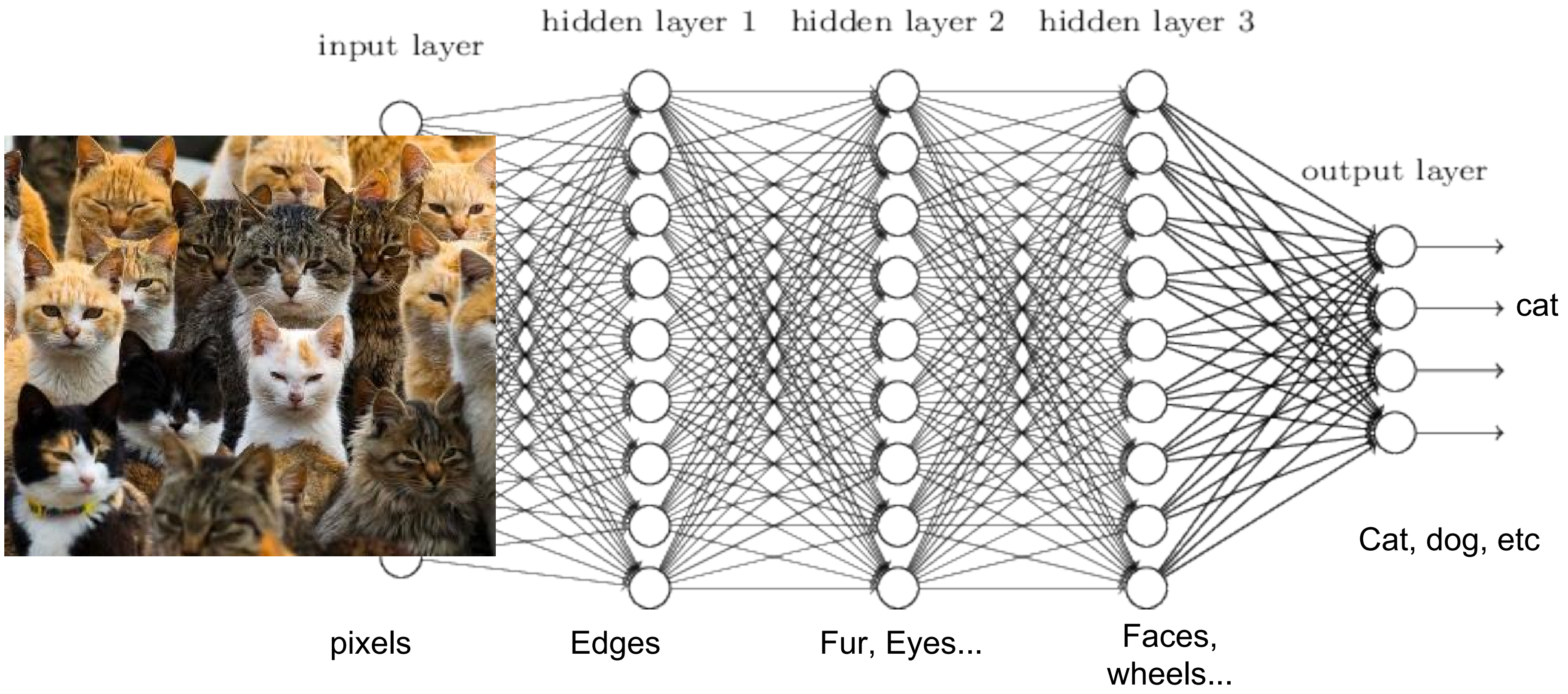
Given the error, we want to adjust any given weight by the amount it contributes to the error on any given sample, so that we reduce the error.

Backpropagate the error into network...

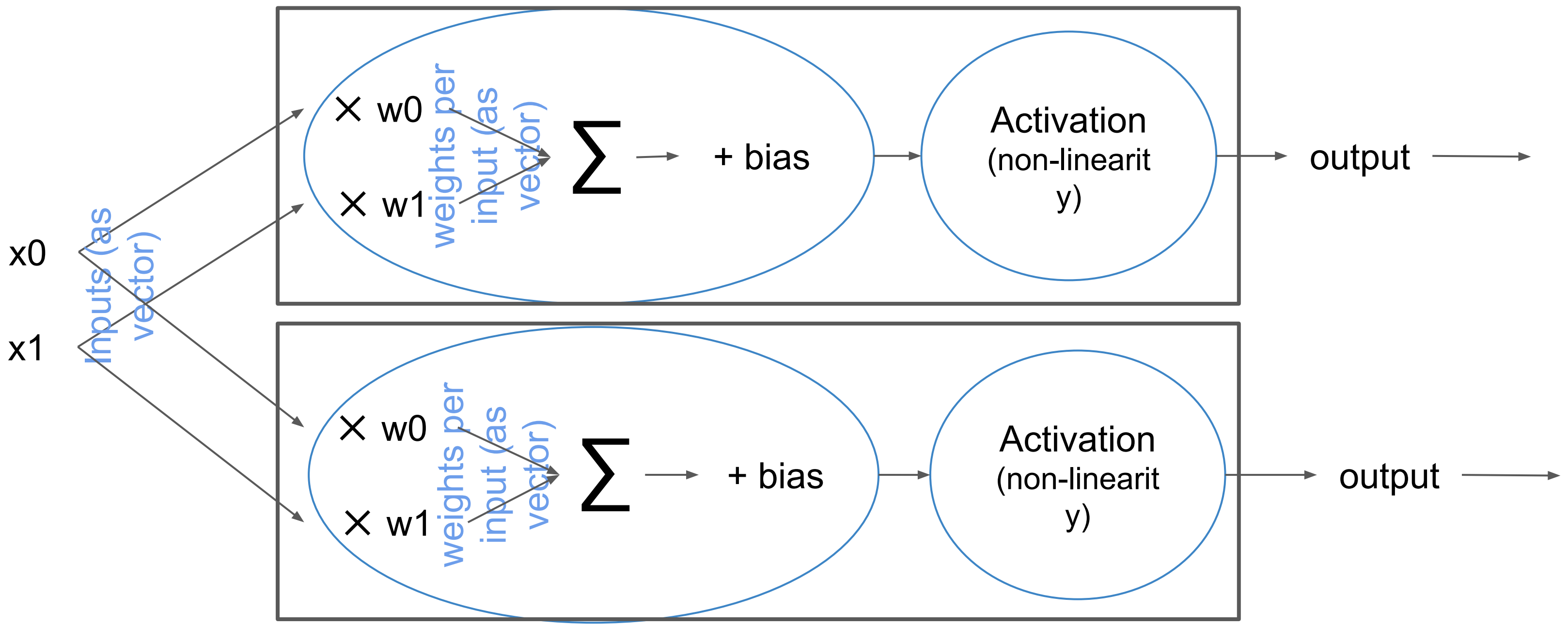
- In forward pass, weights start randomly. We save computations components wrt how they change (differentiating the multiplies & adds & activation function) so that we can know what each contributes to the overal change (the partial differentials).
- Compute the error at output layer, which we want to go to zero.
- Walk back over each connection manipulating the weights based on their partial contribution to error… aka the gradient of each weight with respect to the error.
- But take just a fraction of that change, because we want to keep adjusting over the entire data set.

That walk to lower error is known as gradient descent. The surface has hills...

Given the error, walk a bit of it back, and tweak...



Each component of neuron must be differentiable


x0
x1
Inputs (as vector)
× w0
× w1
weights per input (as vector)
Σ
+ bias
Activation (non-linearity)
output
× w0
× w1
weights per input (as vector)
Σ
+ bias
Activation (non-linearity)
output

Activation functions...

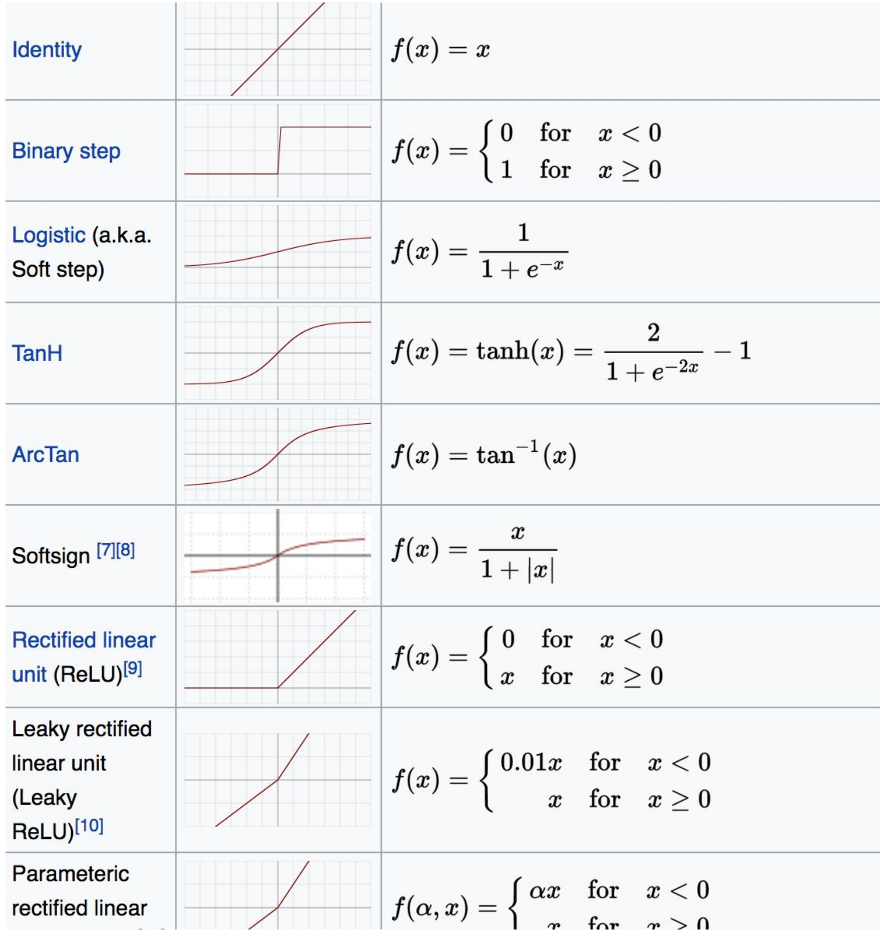
Will also normalize the output value…

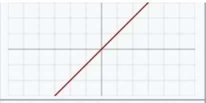
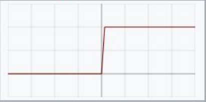
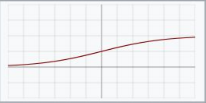
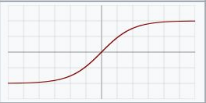
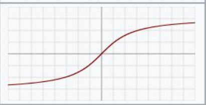
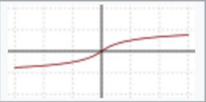
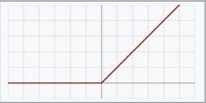
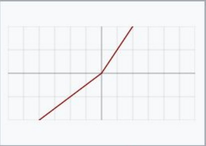
-1 to 1

0 to 1

0 to infinity

Metaphorically have one or more areas of high excitation, activation, trigger...



Identity		$f(x) = x$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$
ArcTan		$f(x) = \tan^{-1}(x)$
Softsign [7][8]		$f(x) = \frac{x}{1 + \lvert x \rvert}$
Rectified linear unit (ReLU)[9]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Leaky rectified linear unit (Leaky ReLU)[10]		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Parametric rectified linear		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$

Live examples...

Visualizing classification & errors: http://ml4a.github.io/dev/demos/mnist_confusion.html

Mnist forward with visualized layers, and code via Keras. https://transcranial.github.io/keras-js/#/mnist-cnn

In browser training of a convnet: http://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html

http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

Forward & Backprop with error display … What the learning curve looks like

- https://docs.google.com/presentation/d/1TVixw6ItiZ8igjp6U17tcgoFrLSaHWQmMOwjlgQY9co/pub?slide=id.g12eed0e537_0_67

What are CNN & Convolutions?

Filter size & strides

Single convolution

http://ml4a.github.io/demos/convolution/

A layer convolution filters
http://ml4a.github.io/demos/convolution_all/

There are a lot of details that matter for all this...

Choices & research in all this such as:

- how it’s efficiently computed alone and in groups,
- what are useful activations,
- how to reduce over-fitting by
    - preferring distributed weights with a regularization factor added to the error function,
    - Dropping-out notes randomly in the forward pass to force exploration of the solution space
- How to divide the data up between training, validation, and test folds
- Extending data to enhance learning: e.g. shifting, the image around, rotations, flips
- How to adjust the learning rate (sgd, momenta, adagrad, …)

Architectures of Layers

CNN, RNN, LSTM,

AlexNet

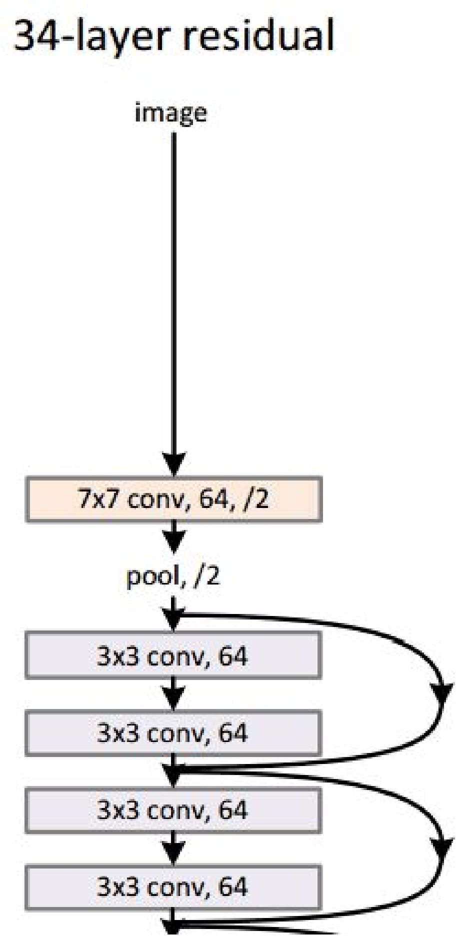
VGGNet

GoogleNet

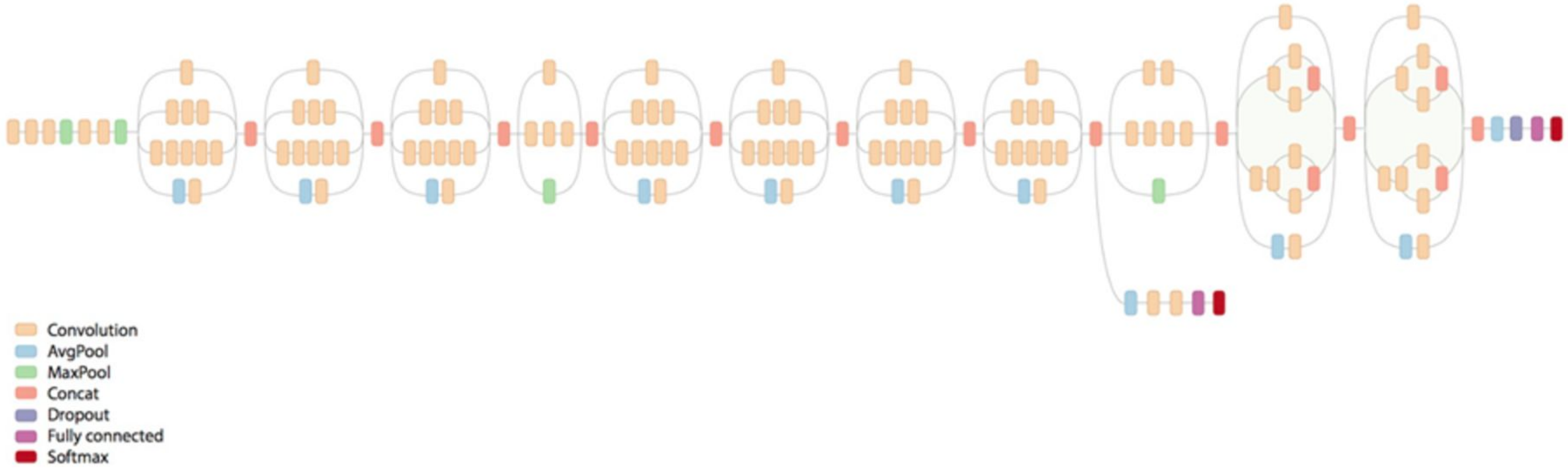
ResNets

Systems and Architectures Layers

Architectures of connection
Intuitive insight (keras demos)
Maxpooling layers as a form of convolution
Convolution as a modified network where the filter is modulated



Systems and Architectures Layers



Another view of GoogLeNet's architecture.

Transfer Learning

Using transfer learning from larger data sets to make use of smaller ones

E.g. torch & keras libraries

A layer architecture: CNN

Visualizing CNN

Visualizing Networks

What is being detected?

What is the organization of the network with respect to the input?

- Visualization of activating neurons in a layer
- Visualization of a layer weights in a layer

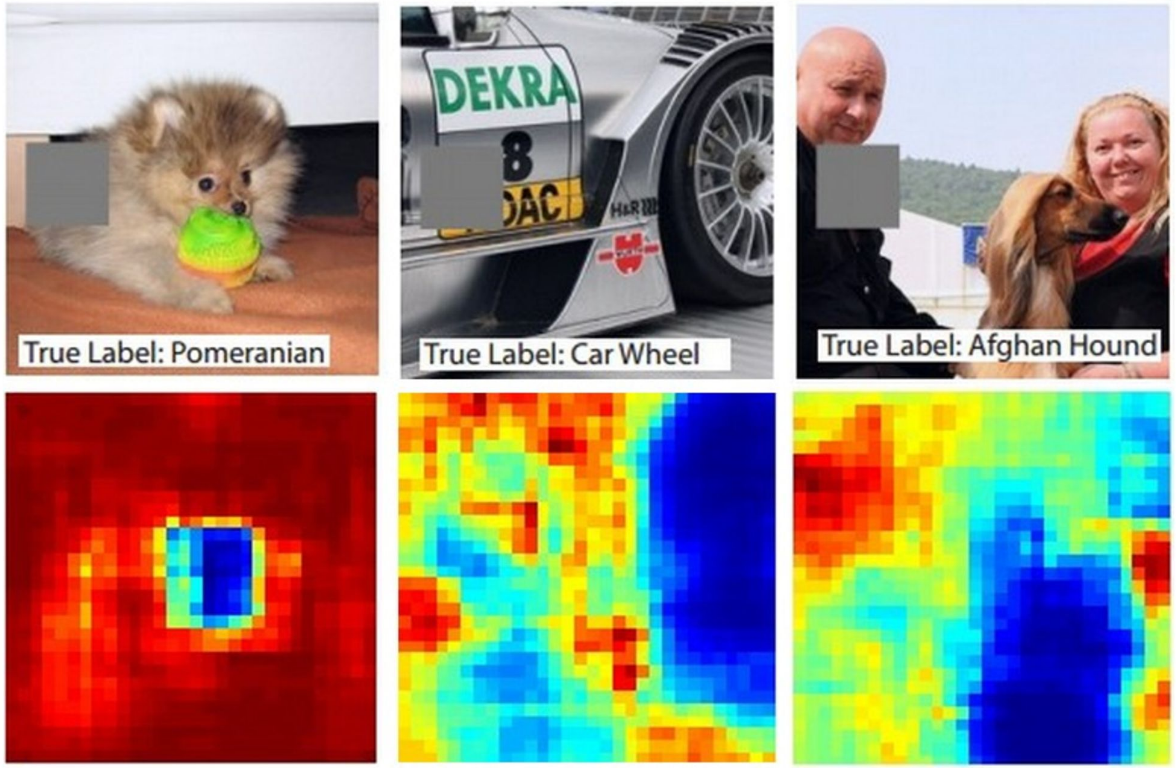
If I manipulate the network and the flow, do I get expected results?

- Occlusion heat maps
- Amplification of input based on changes in network.

Occlusion heat maps

Move a mask across known input image that the network classifies well based on the output.

As it moves record the output strength as a 'heat map' displaying the areas of strongest output.

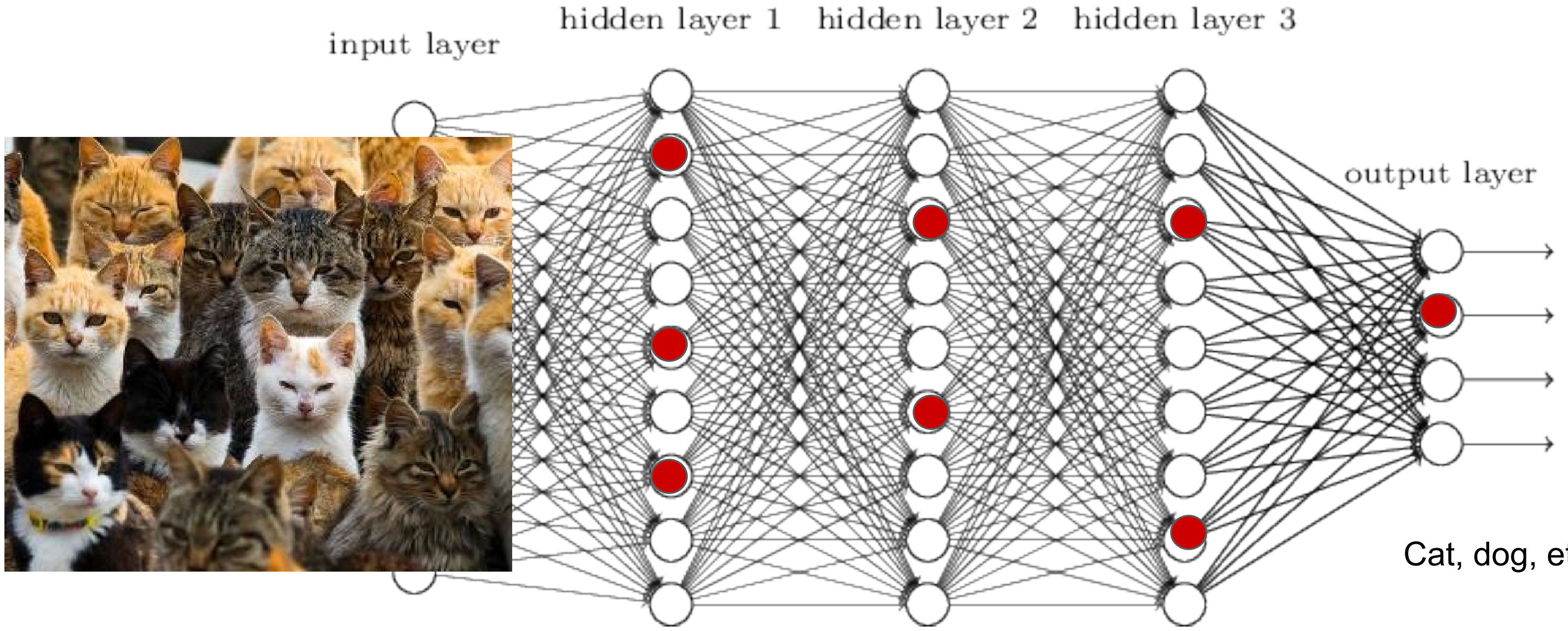


Running networks in reverse

Given a trained image classification network, repeatedly make tiny increases to those neurons in any layer that are recognizing something of interest and work the delta backwards all the way back into the image itself. The effect is to amply aspects of images that (in feedforward) effect that neuron.

DeepDream applied to video frame by frame: https://www.youtube.com/watch?v=oyxSerkkP4o&feature=youtu.be

Running classification networks in reverse



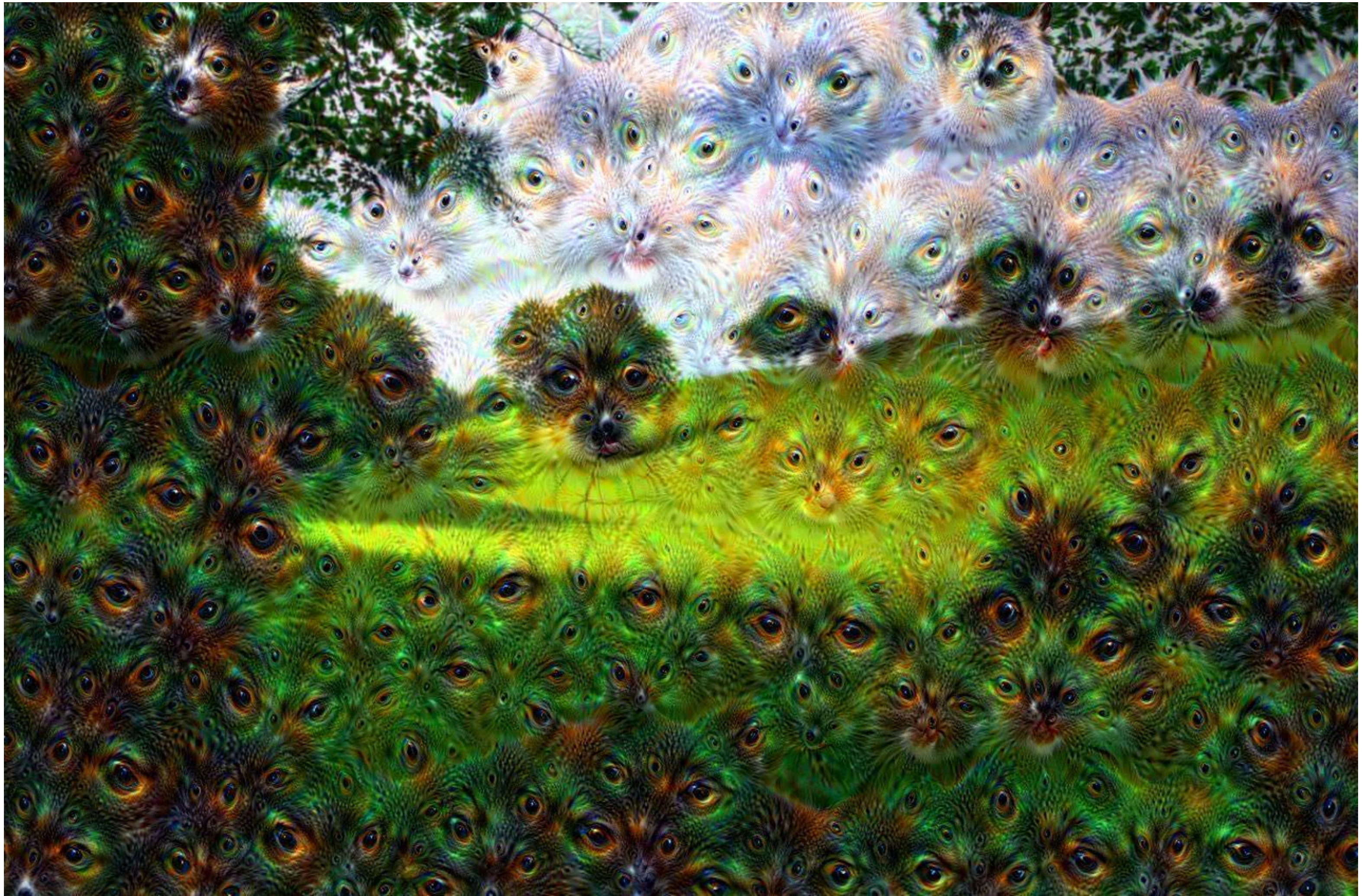
Cat, dog, etc

pixels

Edges

Fur, Eyes...

Faces, wheels...



Running Networks in Reverse

- DeepVis
    - Starting with a noise image, go down the network to find a neuron that is triggered. Amplify one of those at end by backprop, and at end add small fraction of the gradient back modifying image itself. Repeat.
    - Depending on how you focus on a given classification or neuron results in a more or less recognizable initial image. They find also that using prior images (regularization) that you can get more recognizable images by highlighting the parts that promote triggering (aka saliency maps). Or by not so extremely pushing the trigger neuron, something more understandable turns up:
    - http://yosinski.com/deepvis

Refs: Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps -Karen Simonyan, Andrea Vedaldi, Andrew Zisserman - https://arxiv.org/abs/1312.6034

tSNE - Multi-dimensional Data Visualization tool

t-Stochastic Neighbour Embedding, or t-SNE

Visualizing data similarity of high-dimensional data by maintaining the differences projected into 2 or 3-d space. More similar objects are closer and dissimilar are farther.

Clusters n-dimensional data using a similarity measure. Here CIFAR image set of 60,000 images 32x32x3 bytes is imaged into a ‘useful’ clustering based on tSNE alone.
http://cs231n.github.io/assets/pixels_embed_cifar10_big.jpg
Images are regarded as near (by pixel value alone!).
where as there is no semantic meaning as one might get by learning classification via labeling. Not useless, but a distinction.

Similarly, here is the MINST images of numbers, (standard used for NN testing) using tSne alone. http://scienceai.github.io/tsne-js/

Visualization of the FC7 layer of an Imagenet CNN after training:
http://cs.stanford.edu/people/karpathy/cnnembed/cnn_embed_full_4k.jpg

Refs: https://lvdmaaten.github.io/tsne/ Laurens van der Maaten and Geoffrey Hinton

tSNE

Word relationship (in multiple dimensions) via tSne:
http://homepage.tudelft.nl/19j49/multiplemaps/visualization/index.html?map=word_maps.json.txt

Visualization of 500 most followed accounts on Twitter, based on a sampling 200 of their tweets, computing a similarity value based on the words they use… with tSne:
http://cs.stanford.edu/people/karpathy/tsnejs/

Create your own with a google spreadsheet:
http://homepage.tudelft.nl/19j49/tsnejs/
Or CSV file
US States by long/lat: http://cs.stanford.edu/people/karpathy/tsnejs/csvdemo.html   perplexit of 6, lr 20

Parameters: Perplexity - something like a how many neighbors can be near. Data specific

Nice guide on params & meaning: http://distill.pub/2016/misread-tsne/

Neural Nets and NLP as visualization tools

NLP - Word2Vec

Uses word adjacency, aka word-embeddings, to generate word vectors.

Lower compute effort than NN.

Discovers semantic-esque relationships from word sequences.

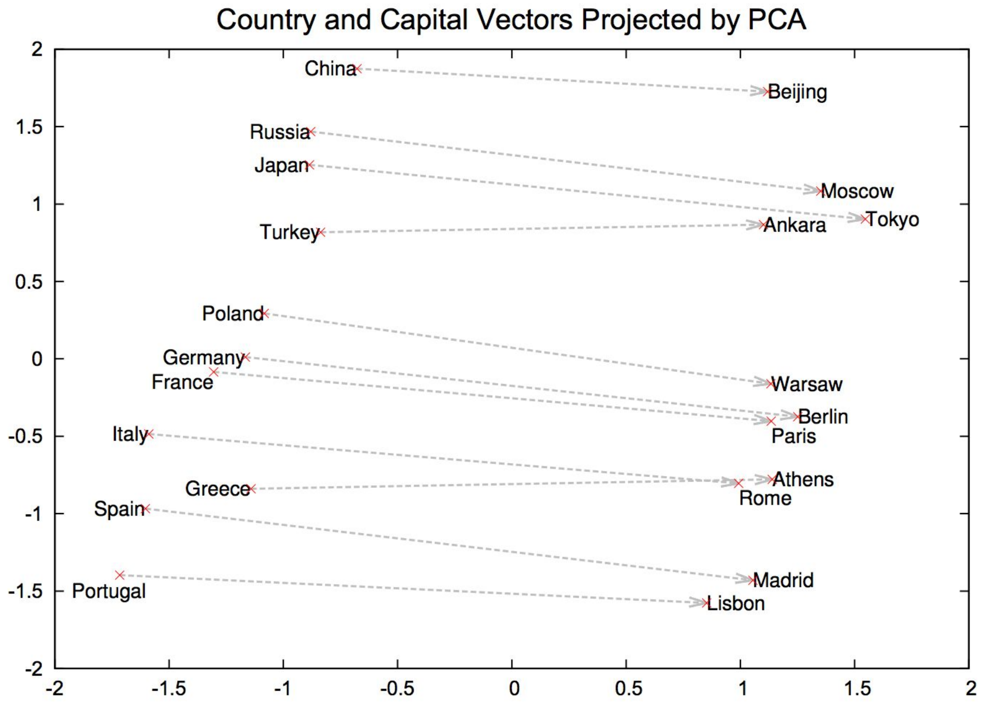


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

Vector math on words...

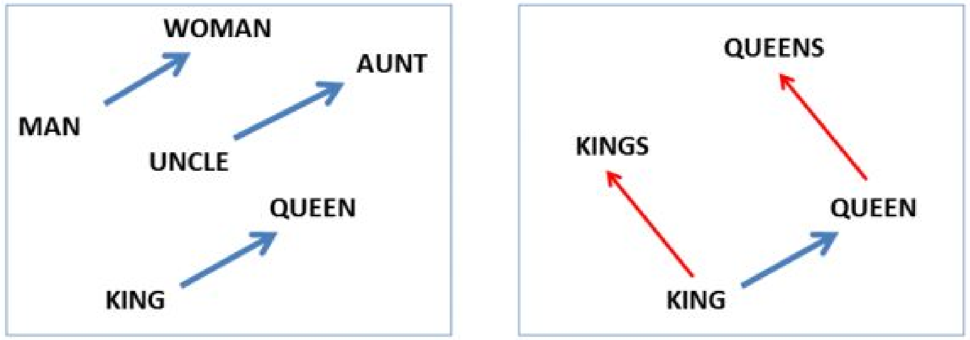
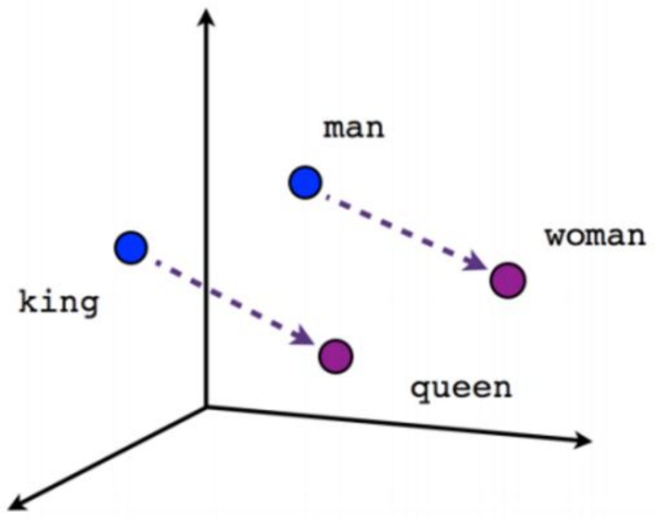


Figure 2: Left panel shows vector offsets for three word pairs illustrating the gender relation. Right panel shows a different projection, and the singular/plural relation for two words. In high-dimensional space, multiple relations can be embedded for a single word.



Using tSne on Word2Vec output...

Visualization of 50 dimensional word vectors:
http://cs.stanford.edu/people/karpathy/tsnejs/wordvecs.html

Refs...

- Efficient Estimation of Word Representations in Vector Space – Mikolov et al. 2013
- Distributed Representations of Words and Phrases and their Compositionality – Mikolov et al. 2013
- Linguistic Regularities in Continuous Space Word Representations – Mikolov et al. 2013
- word2vec Parameter Learning Explained – Rong 2014
- word2vec Explained: Deriving Mikolov et al's Negative Sampling Word-Embedding Method – Goldberg and Levy 2014

Capabilities of other NN Architectures

RNN & LSTM for text : generate texts by learning character sequence. Sequence to sequence = translation

pixrnn & pixelcnn: generate pictures as sequence of next pixel predictions

“PixelRNN and PixelCNN models, published earlier this year, showed that it was possible to generate complex natural images not only one pixel at a time, but one colour-channel at a time, requiring thousands of predictions per image”
https://arxiv.org/pdf/1601.06759.pdf

Wavenet for sound etc: ditto for sound

See audio samples of generative voice here, voicing text, just vocalizations, and for music as output.
https://deepmind.com/blog/wavenet-generative-model-raw-audio/

Controversy Measures

Controversy & Isolation measures

Word/hashtag -> Co-Occuring words/hashtags (aka topic) -> conversants who tweet these topics and who either retweet these tweets or follow people tweet or retweet these are regarded as endorsers and are combined into a conversation network (topology) for this topic -> Partition the network into two halves (find two clusters where there's a minimum of connection between parts of the network) -> Use a random walk measure between halves to determine how apart each halve is. The more unlikely one have is to reach the other, the more controvrsial and echo chambery this topic is. One can measures the isolation one one's echo chamber. That's the controversy measure.

Code here: https://github.com/gvrkiran/controversy-detection … Uses Metis library for partitioning. From paper: "Quantifying Controversy in Social media" Author: Kiran Garimella https://arxiv.org/pdf/1507.05224.pdf (kiran.garimella@aalto.fi)

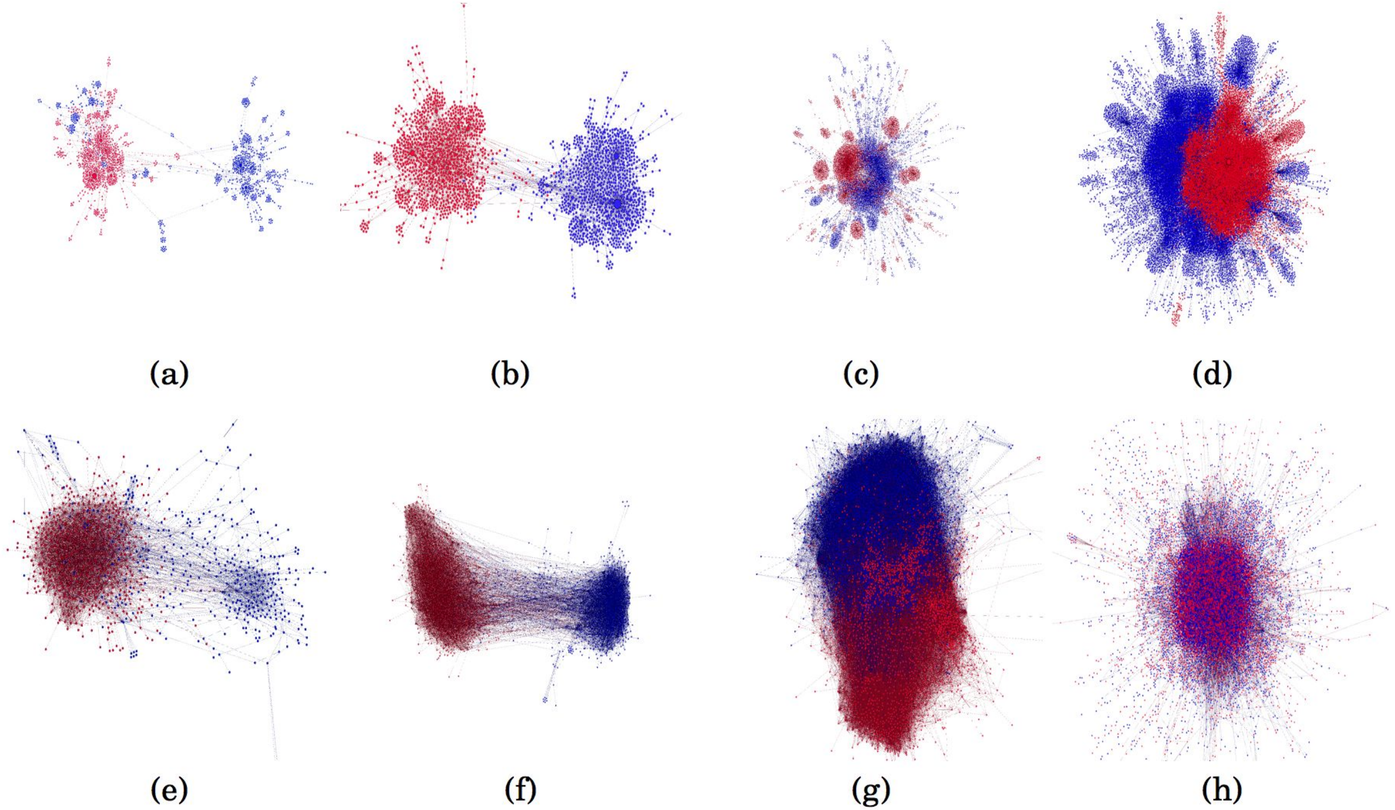


Fig. 3: Sample conversation graphs with retweet (top) and follow (bottom) aspects (visualized using the force-directed layout algorithm in Gephi). The left side is controversial, (a,e) #beefban, (b,f) #russia_march, while the right side is non-controversial, (c,g) #sxsw, (d,h) #germanwings.