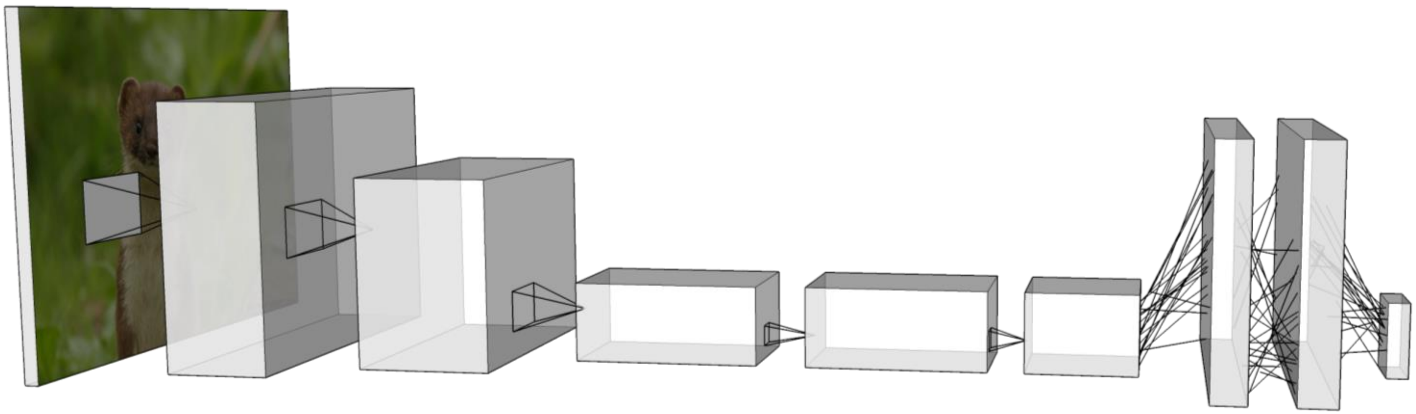
mathsml.group.cam.ac.uk

An introduction to backpropagation

Damon Wischik
Computer Laboratory

A neural network for classifying images



$x \in [0,1]^{224\times224\times3}$

input image,
3 colour
channels

$F_{w,b}(x) = \xi$

neural network, with
138 million
parameters w, b

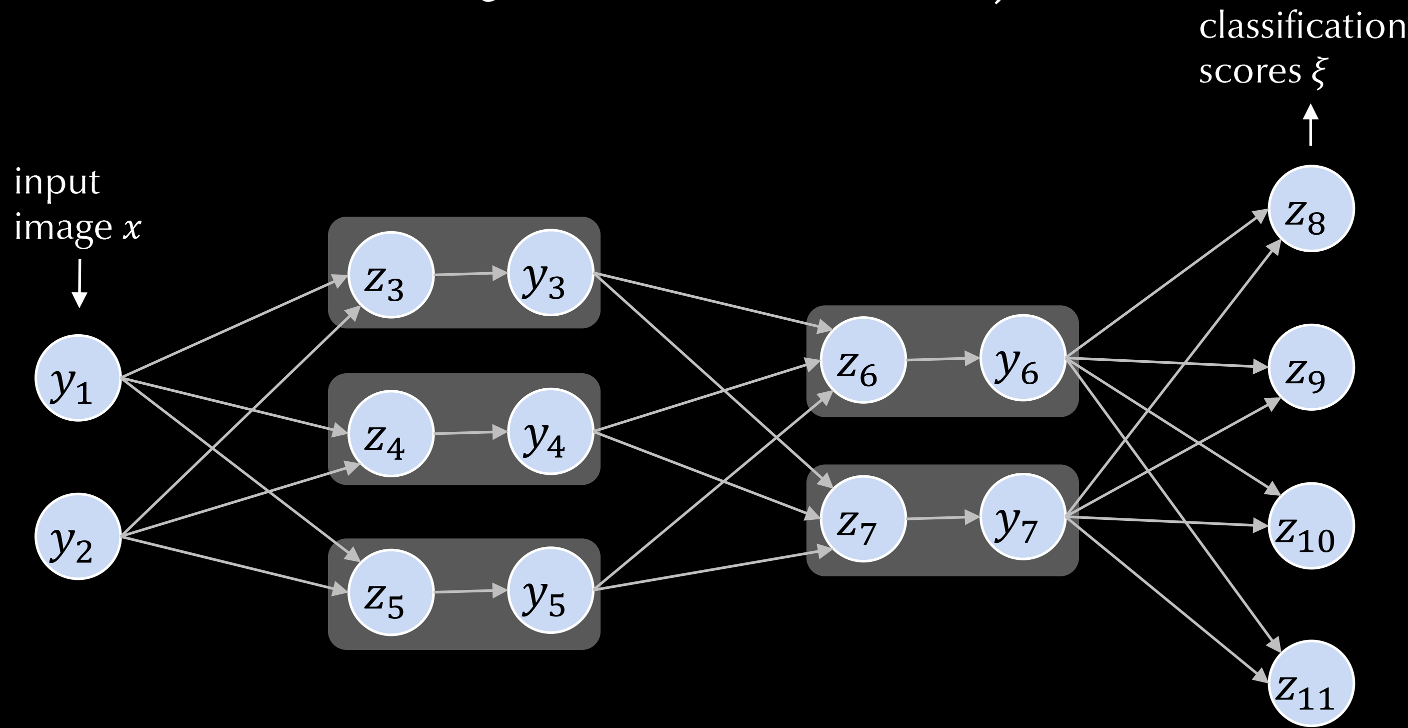
$\xi \in \mathbb{R}^{1000}$

classification scores,
interpreted as

$\mathbb{P}(\text{stoat}) = \dfrac{e^{\xi_{\text{stoat}}}}{\sum_r e^{\xi_r}}$

Network architecture and training
Very deep convolutional networks for large-scale image recognition, Simonyan and Zisserman (Oxford visual geometry group), 2015.

The neural network function $\xi = F_{w,b}(x)$



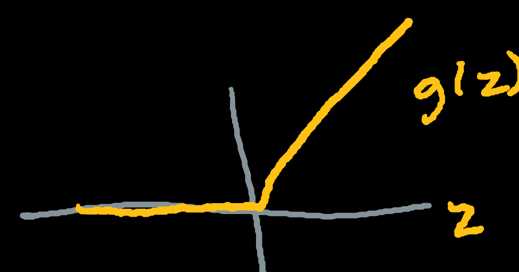
The network is a directed acyclic graph.
Input nodes and output nodes store one value;
intermediate hidden nodes store two values.

input nodes n: $\quad y_n = x_{j(n)}$

hidden nodes n: $\quad z_n = b_n + \sum_{m:m\to n} y_m w_{m\,n}$

$\quad y_n = g(z_n) = \max(z_n, 0)$

output readout r: $\quad \xi_r = z_{n(r)}$



The objective function

Given a dataset of images $x^1, x^2, \ldots,$
annotated with their actual classification $l^1, l^2, \ldots$

we want to find parameters w and b to minimize

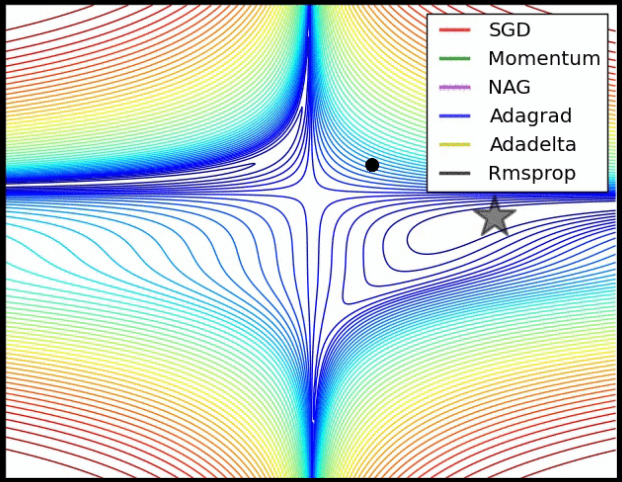
$$E(w, b) = -\log \mathrm{lik}(w, b \mid x, l) = -\sum_i \log \mathrm{lik}(w, b \mid x^i, l^i)$$

where the log likelihood is obtained from the model

$$\mathbb{P}(\text{class of image } i = l) = \frac{e^{\xi^i_l}}{\sum_r e^{\xi^i_r}}, \quad \xi^i = F_{w,b}(x^i)$$

Training dataset
ImageNet: a large-scale hierarchical image database, Deng, Dong, Socher, Li, Li, Fei-Fei, 2009.
1.3 million images, each annotated with one of 1000 labels

Training is by gradient descent and hyperparameter tuning



Objective: minimize over $\theta = (w, b)$ the loss function

$$E = \sum_{\text{images } i} E_1(w, b | x^i, l^i)$$

The basic method is iterative gradient descent,

$$\theta \leftarrow \theta - \delta \frac{\partial E}{\partial \theta}$$

with endless variations and lots of babysitting.

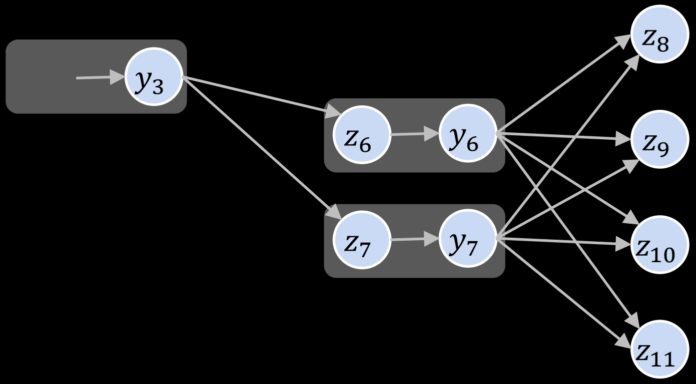
"Usually, there are lots and lots of equally good minima."
Karpathy

"The batch size was set to 256, momentum to 0.9. The learning rate was initially set to 10^{-2} and then decreased by a factor of 10 … the learning rate was decreased 3 times. … The initialisation of the network weights is important."
Simonyan+Zisserman

"During training, monitor the loss, the training/validation accuracy, and if you're feeling fancier, the magnitude of updates in relation to parameter values (it should be ~10^{-3}) … Decay your learning rate over the period of the training. … Search for good hyperparameters with random search (not grid search)."
CS231n lecture notes

Gradient descent animation: Andrej Karpathy, for CS231n at Stanford
Practical advice: CS231n, http://cs231n.github.io/neural-networks-3/
Practical recommendations for gradient-based training of deep architectures, Bengio, 2012

Backpropagation



Neural network function:

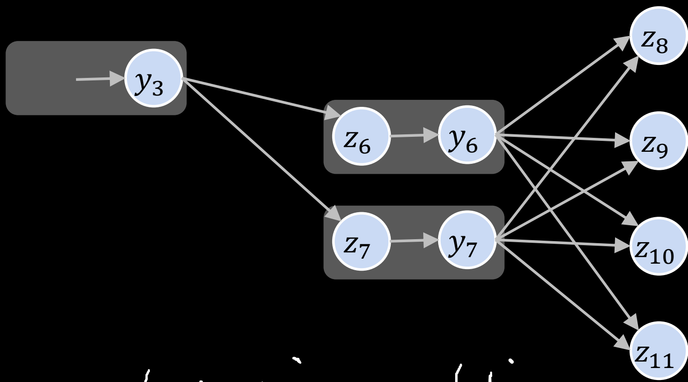
$$z_n = b_n + \sum_{m:m\to n} y_m w_{m\,n}$$

$$y_n = g(z_n) = \max(z_n, 0)$$

The objective is to minimize:

$$E = \sum_{\text{images } i} E_1(w, b \mid x^i, l^i)$$

Backpropagation



Neural network function:

$$z_n = b_n + \sum_{m:m\to n} y_m w_{m\,n}$$

$$y_n = g(z_n) = \max(z_n, 0)$$

The objective is to minimize:

$$E = \sum_{\text{images } i} E_1(w, b \mid x^i, l^i)$$

For a single training example i:

For n in the output layer:

$$\frac{\partial E_i}{\partial y_n} = \text{easy to derive}$$

For other nodes n:

$$\frac{\partial E_i}{\partial y_n} = \sum_{l:n\to l} \frac{\partial E_i}{\partial z_l} \cancel{\frac{\partial z_l}{\partial y_n}}\, w_{nl}$$

$$\frac{\partial E_i}{\partial z_n} = \frac{\partial E_i}{\partial y_n} \cancel{\frac{\partial y_n}{\partial z_n}}\, g'(z_n)$$

For the parameters:

$$\frac{\partial E_i}{\partial w_{mn}} = \frac{\partial E_i}{\partial z_n} y_m$$

$$\frac{\partial E_i}{\partial b_n} = \frac{\partial E_i}{\partial z_n}$$

Method:

forward

$$\text{image} \longrightarrow \underbrace{\cdot \rightarrow \cdot}_{\text{int. nodes}} \longrightarrow \text{output layer} \longrightarrow E_i$$

$$\underbrace{\cdot \leftarrow \cdot}_{\text{int. nodes}} \longleftarrow \frac{\partial E_i}{\partial y_n} \text{ for output layer}$$

$$\frac{\partial E}{\partial z_n} \quad \frac{\partial E_i}{\partial y_n}$$

$$\downarrow$$

$$\frac{\partial E_i}{\partial w_{mn}} \quad \frac{\partial E_i}{\partial b_n}$$

backwards.

```python
g = tf.Graph()
with g.as_default():
    # inputs
    x = tf.placeholder(tf.float32, shape=[BATCH_SIZE, 784], name='x')
    y = tf.placeholder(tf.float32, shape=[BATCH_SIZE, 10], name='y')

    # reshape a batch of inputs to be of dimension [28,28,1] 28=width, 28=height, 1=channels
    x0 = tf.reshape(x, [-1, 28,28,1])

    # convolve with a 5x5x1x32 matrix (gives 32 features for every 5x5x1 tile), then add constant
    # then apply the relu function elementwise
    # then pool over 2x2x1 blocks, giving a 14x14x32 image
    W_conv1 = tf.Variable(tf.truncated_normal(mean=0.0, stddev=0.1, shape=[5,5,1, 32]), name='w_conv1')
    b_conv1 = tf.Variable(tf.constant(0.1, shape=[32]), name='b_conv1')
    z1 = tf.nn.conv2d(x0, W_conv1, strides=[1,1,1,1], padding='SAME') + b_conv1
    y1 = tf.nn.relu(z1)
    h1 = tf.nn.max_pool(y1, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME', name='h1')

    # ...

    # another fully-connected layer, giving an output of size 10
    W_cls = tf.Variable(tf.truncated_normal(mean=0.0, stddev=0.1, shape=[1024,10]), name='w_cls')
    b_cls = tf.Variable(tf.constant(0.1, shape=[10]), name='b_cls')
    y4 = tf.matmul(z4, W_cls) + b_cls
```

Define the computation graph via code.

Use matrix/tensor syntax for repeated operations.

```python
    # define the loss function and accuracy metrics
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y, logits=y4), name='loss')
    is_correct = tf.equal(tf.argmax(y,1), tf.argmax(y4,1))
    accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
```

Include nodes that compute E, and other metrics for logging

```python
    # add necessary computation nodes for gradient descent
    train_step = tf.train.AdamOptimizer(1e-4).minimize(loss)
```

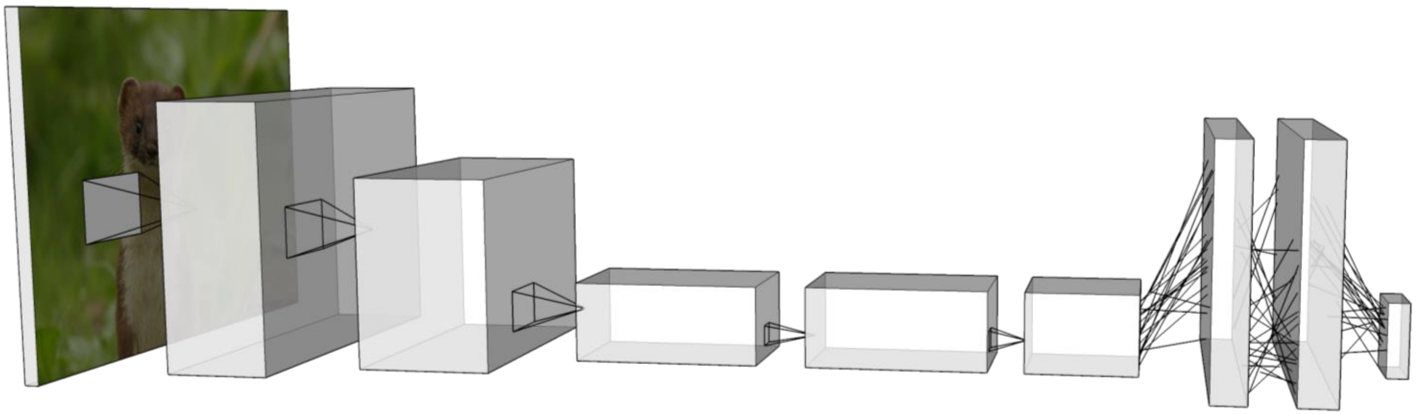
automatically generate computation nodes for $\frac{\partial E}{\partial x}$ and specify the gradient descent method

```python
with tf.Session(graph=g) as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(20000):
        batch = mnist.train.next_batch(50)
        train_data = {x: batch[0], y: batch[1], keep_prob: 0.5}
        test_data = {x: mnist.validation.images, y: mnist.validation.labels, keep_prob: 1}
        sess.run(train_step, train_data)
        if i % 100 == 0:
            print(i, "train", sess.run([loss, accuracy], train_data))
            print(i, "test", sess.run([loss, accuracy], test_data))
```

run it, on CPU / GPU / cluster

TensorFlow code. https://notebooks.azure.com/djw1005/libraries/mathsml/html/mnist.ipynb

"A structure primed for vision"



Re-use the parameters, by treating them as convolutions.

For pixel (i, j) in layer $L + 1$,

$$z_{i,j}^{L+1} = b^L + \sum_{i',j'} w_{i-i',j-j'}^L \, y_{i',j'}$$

Additionally, we can store multiple features f for each pixel,

$$z_{i,j,f}^{L+1} = b_f^L + \sum_{i',j',f'} w_{i-i',j-j',f',f}^L \, y_{i',j',f'}$$

Training works better if you "prime the structure" to suit your problem.

There are neural network architectures primed for

- vision (convolutional networks)
- time series / language (recurrent networks)
- dimension reduction (autoencoders, generative adversarial networks)
- branching processes
- relational reasoning

They can be mixed together, e.g. vision + time series = image captioning.

a dog is standing in the snow with a frisbee

`logprob: -8.44`

a red double decker bus driving down a street

`logprob: -6.18`

a bathroom with a toilet and a sink

`logprob: -6.23`

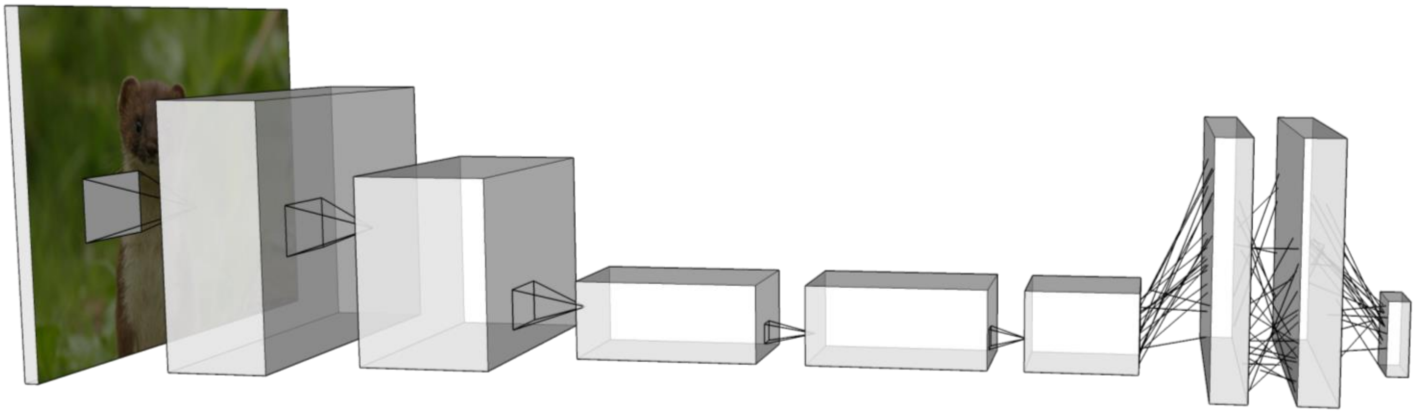
a cat sitting on a window sill looking out the window

`logprob: -7.69`

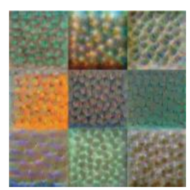
NeuralTalk, Karpathy. https://github.com/karpathy/neuraltalk2

Q. How can a neural network identify what's in a picture if it doesn't understand *the picture?*

~~A. Ha ha, silly, there is no *understand*, there is only *optimize*.~~

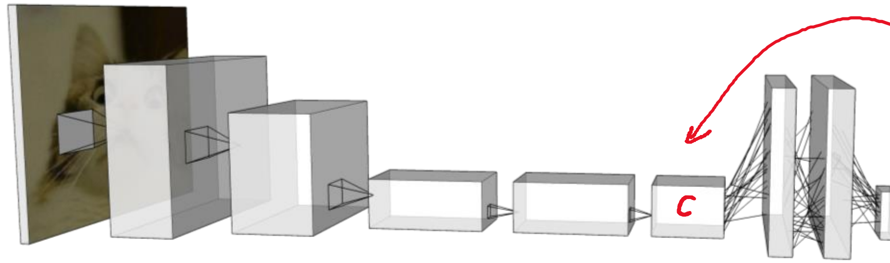


What patterns of input will maximally excite a node, at various depths in the network?

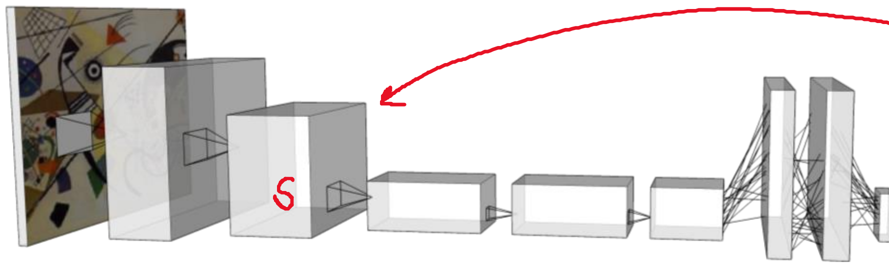


"A structure primed for vision"

Load in a pre-trained network for image classification. Run it on two images:

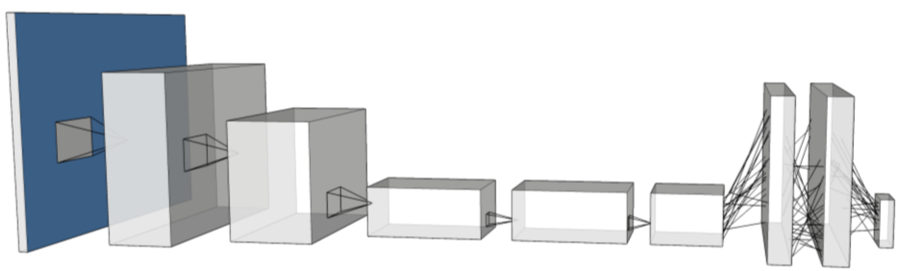


$c($$)$ = values in a deep layer



$s($$)$ = feature×feature correlation matrix at an early layer

Find the image p that minimizes $\|s(p) - s($$)\| + \|c(p) - c($$)\|$



Cat art from http://www.ostagram.ru. *Transverse Line,* 1923, Wassily Kandinsky
A neural algorithm of artistic style (2015) by Gatys, Ecker, Bethge. https://arxiv.org/abs/1508.06576
TensorFlow code. `https://notebooks.azure.com/djw1005/libraries/mathsml/html/style-transfer.ipynb`

Image produced by `https://turbo.deepart.io/`
Sample code at `https://notebooks.azure.com/djw1005/libraries/mathsml/html/style-transfer.ipynb`

The only way to classify images well,
is to build a general-purpose visual processing cortex.
That's what the neural network learnt.

Transfer learning

Use a network trained on one task to bootstrap a second task
(Useful if the second task is data-poor)

Multi-task training

Train a network with multiple simultaneous objectives
(Improves the performance on each objective)

Dropout regularization

In each training iteration, randomly knock out half the nodes
(Improves ability to generalize)

Running backpropagation for neural style transfer is slow.
To speed it up, can we fit an approximating function?

approximating
network

x

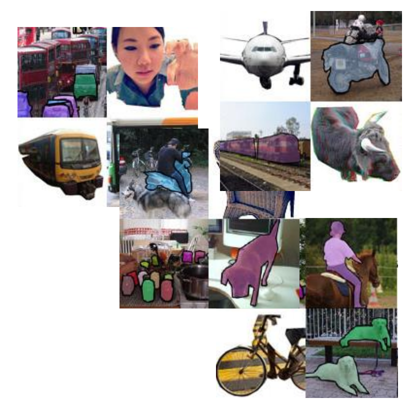
w, b

$y = F_{w,b}(x)$

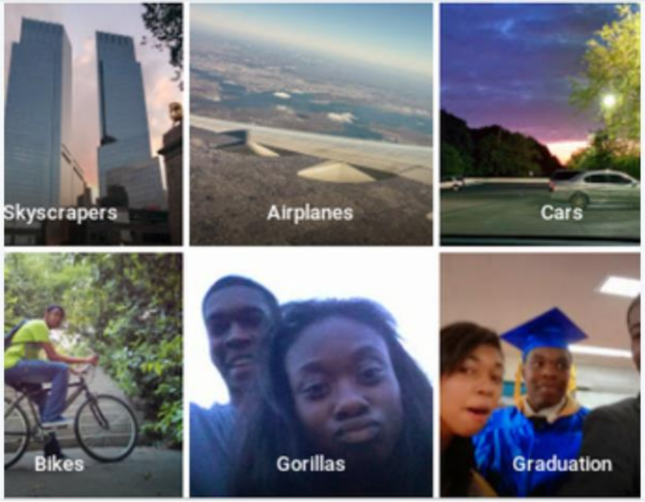
loss function

$E = \|s(y) - s($$)\| + \|c(y) - c(x)\|$

1. Pick a style image.
2. Find a large dataset of everyday content images.
3. Train a feedforward network to learn style transfer, on this dataset.
4. Now, we have a fast approximate implementation of style transfer
   (on the domain of images similar to those in the dataset).



Perceptual Losses for Real-Time Style Transfer and Super-Resolution, Johnson, Alahi, Fei-Fei (2016)
Microsoft COCO: Common Objects in Context, Lin et al. (2015)



Jacky Alciné
@jackyalcine

Follow

Google Photos, y'all fucked up. My friend's not a gorilla.

2:22 AM - Jun 29, 2015

224 3,182 2,024

Jacky Alciné @jackyalcine Jun 29, 2015

Replying to @jackyalcine

Like I understand HOW this happens; the problem is moreso on the WHY.

This is how you determine someone's target market.

(((Yonatan Zunger)))
@yonatanzunger

Follow

@jackyalcine Holy fuck. G+ CA here. No, this is not how you determine someone's target market. This is 100% Not OK.

4:07 AM - Jun 29, 2015

14 47 124

Challenges

1. It's odd to train neural networks with objective functions.

   They try to learn multi-purpose representations anyway.
   And multi-purpose representations would be more useful for us,
   to save us from having to retrain for every new question.

2. The data's the thing.

   Has the network learned the representations it needs
   to be able to extrapolate appropriately to new (counterfactual) situations?
   How can we as data scientists check this?

3. Why is training so painfully slow, and such an art?

The human brain has roughly 10^{15} connections, and a human lifetime is roughly 2.5×10^9 seconds. What are all the parameters for, and how are they trained?