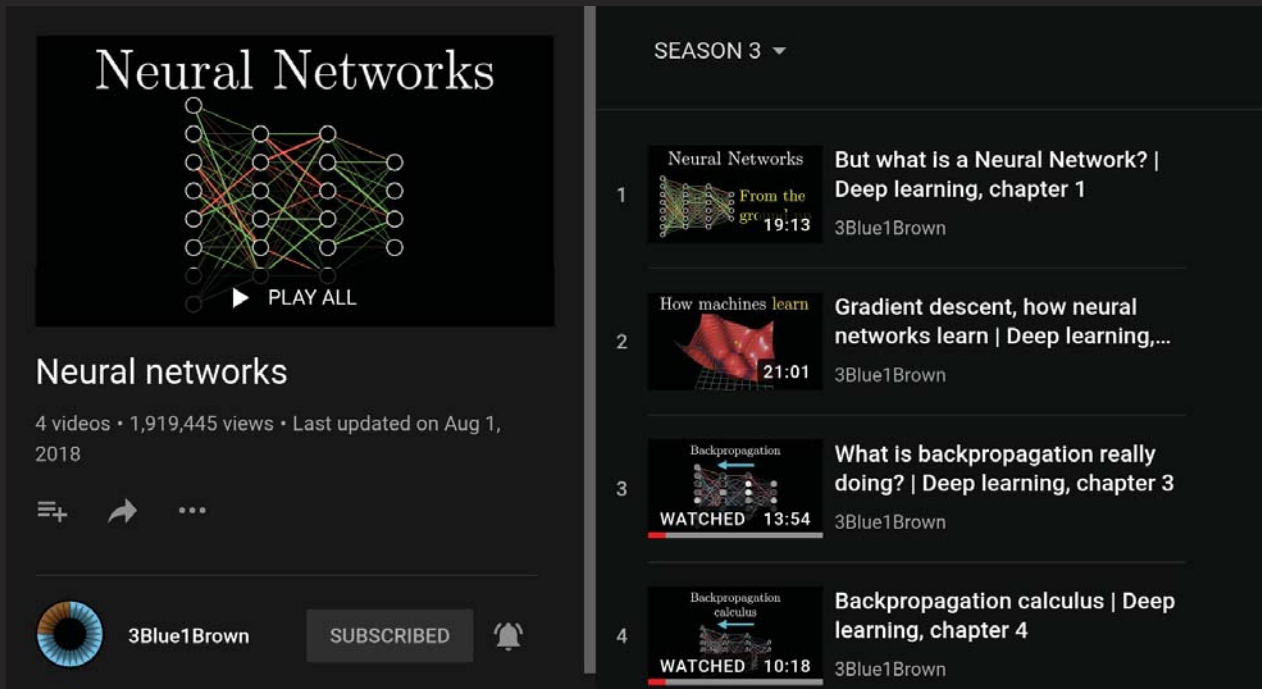
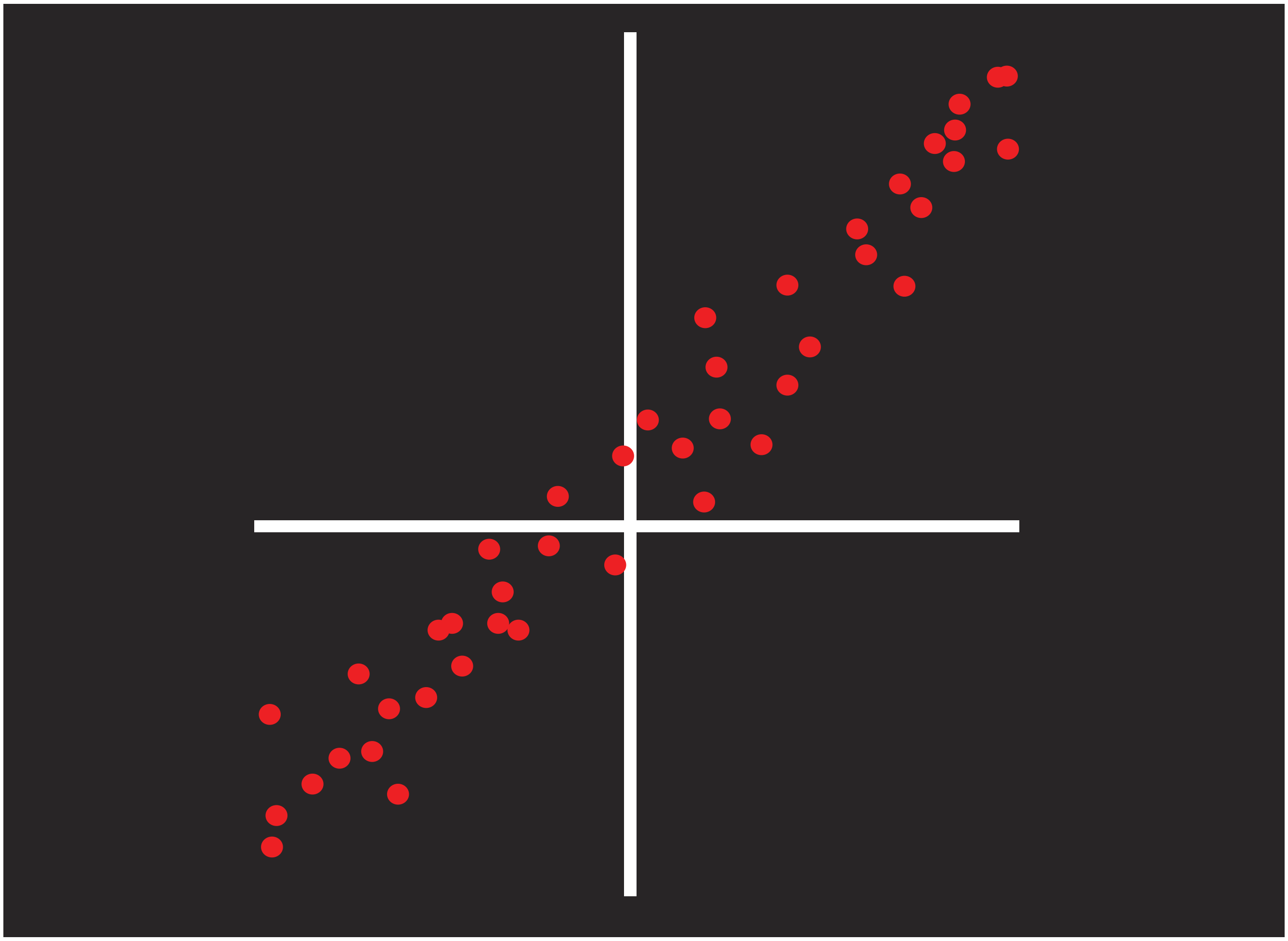
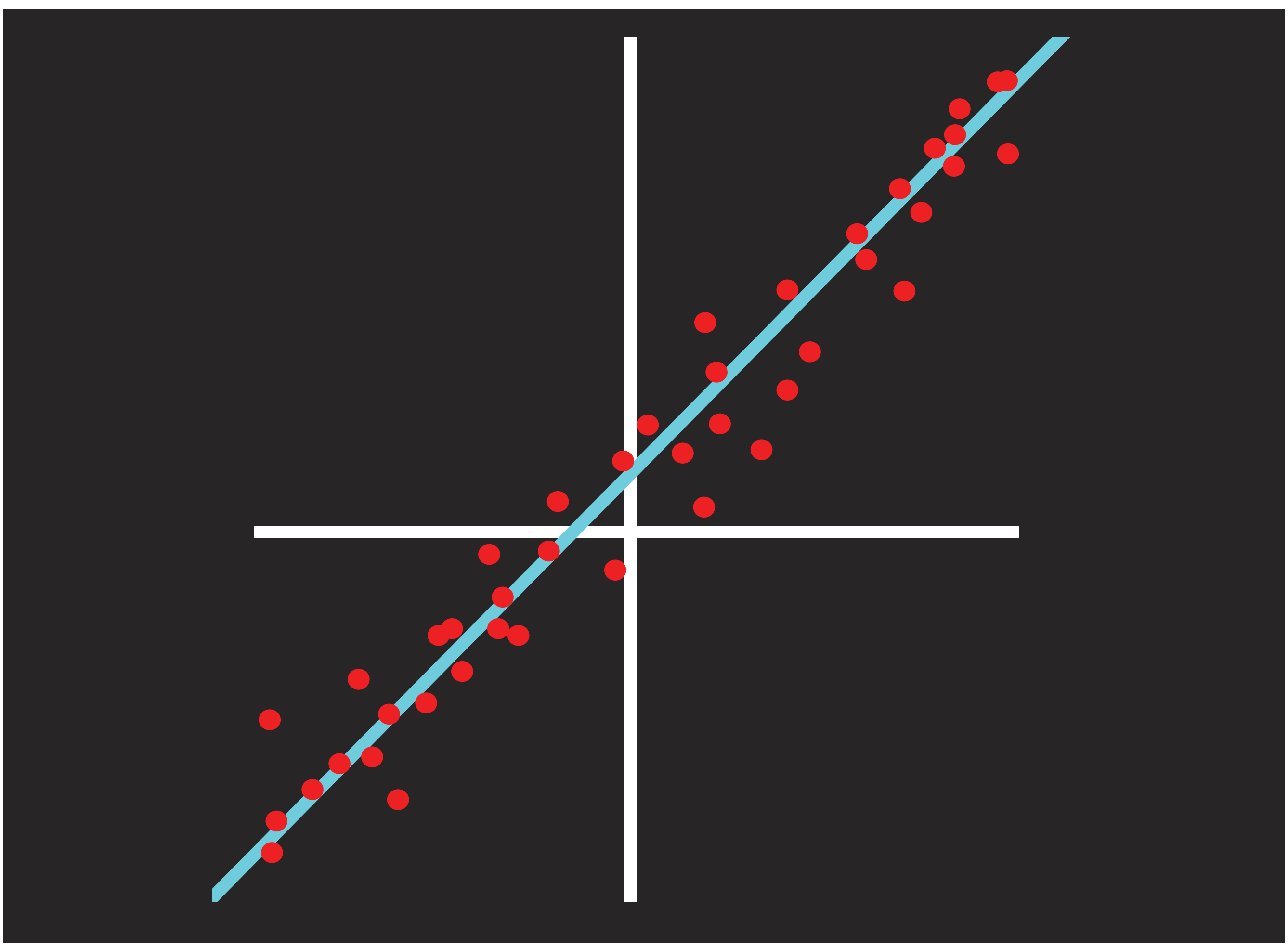
the deep learning lecture

jack garbus

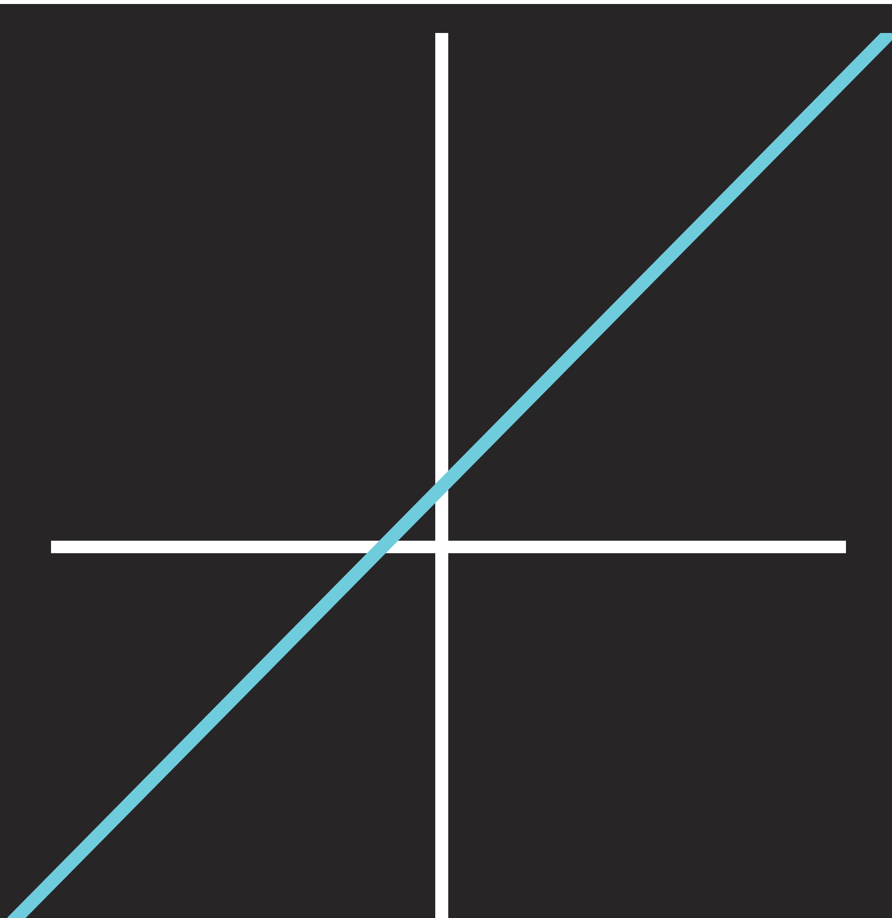


deep learning = calculus + matrix multiplication

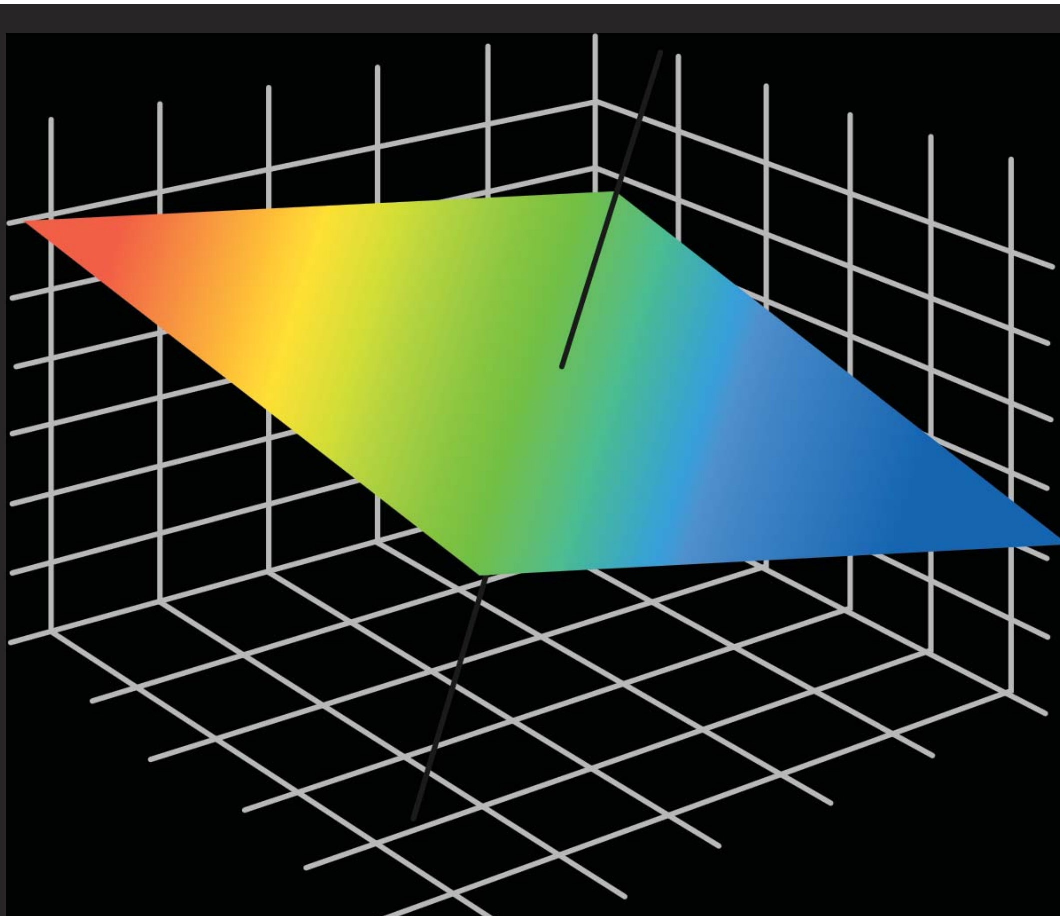




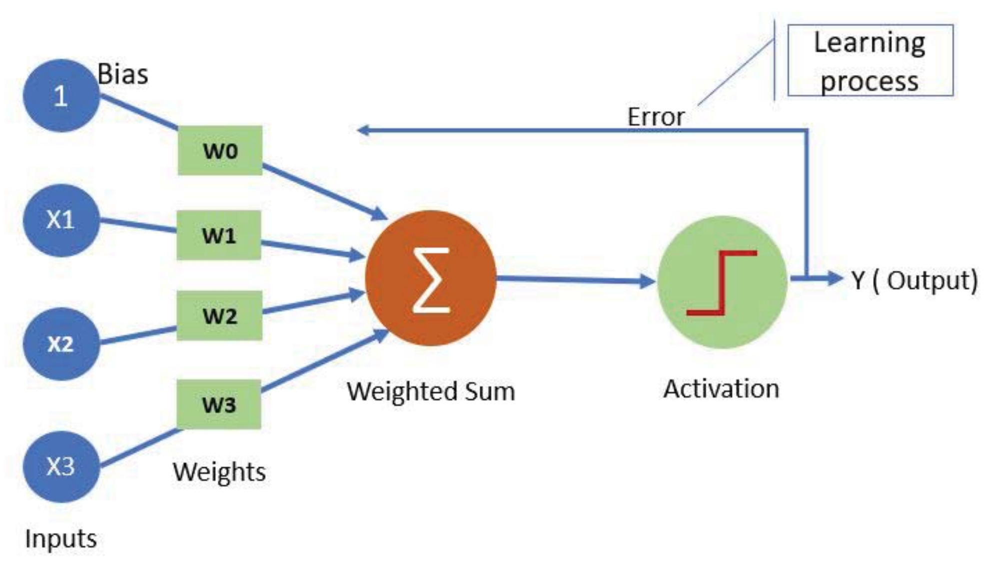
$$y = mx + b$$



$$z = ax + by + c$$



the perceptron



Single Layer Perceptron

AND

AND, OR

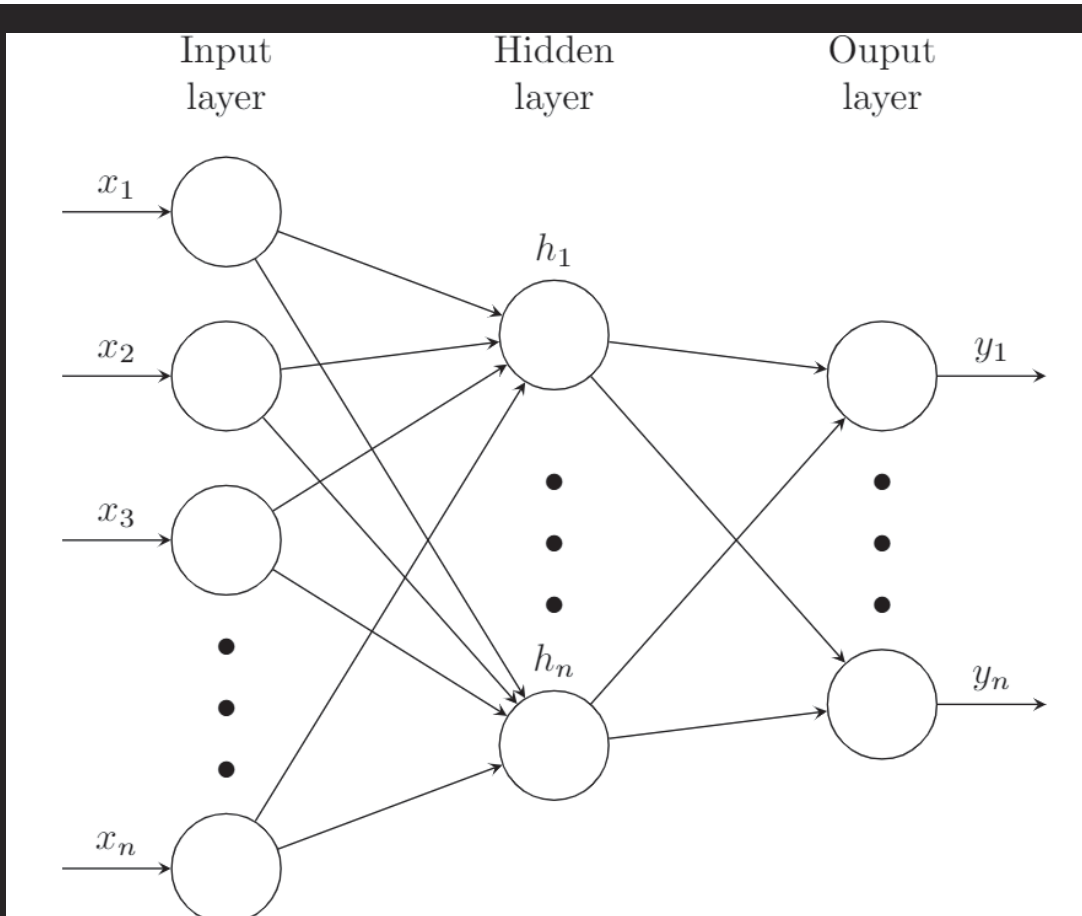
AND, OR, and NOT

BUT

can't do XOR

multi-layer perceptron

feed-foward neural network



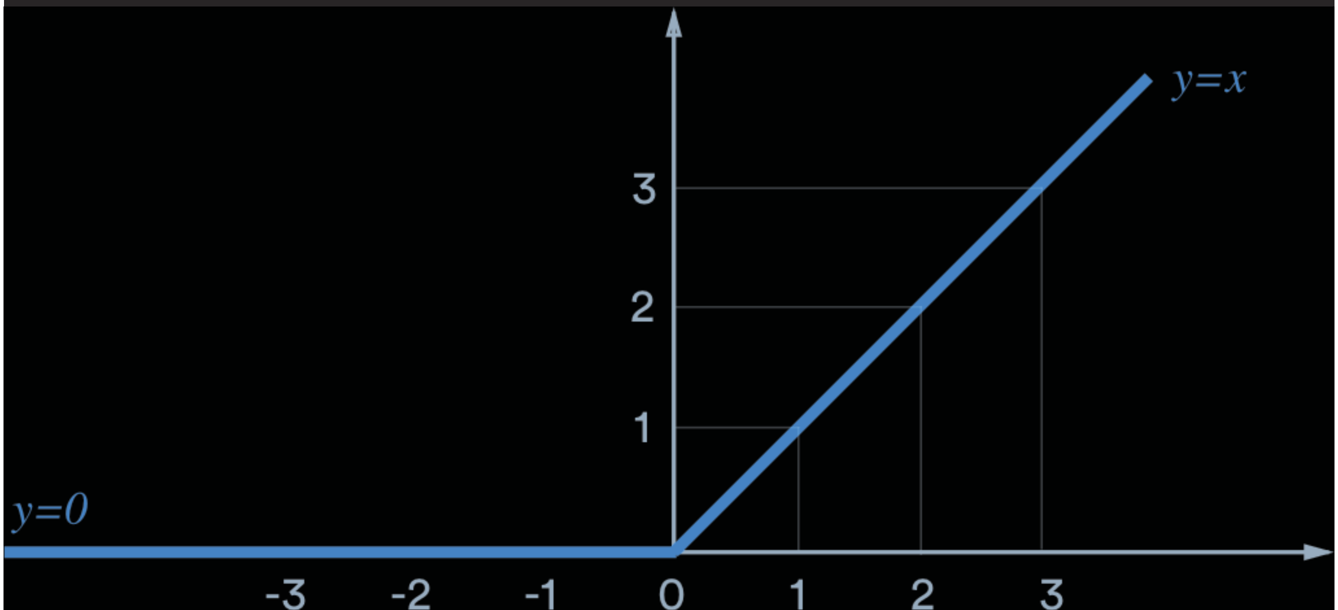
deep neural network has ≥ 1 hidden layer

```
def make_dense_net():
    inputs = layers.Input(shape=(50, 50, 4))
    layer0 = layers.Flatten()(inputs)
    layer1 = layers.Dense(64, activation='relu')(layer0)
    layer2 = layers.Dense(64, activation='relu')(layer1)
    layer3 = layers.Dense(512, activation='relu')(layer2)
    out = layers.Dense(2, activation='linear')(layer3)
    model = keras.Model(inputs=inputs, outputs=out)
    optimizer = keras.optimizers.Adam(learning_rate=0.00001)
    model.compile(loss='mse', optimizer=optimizer, metrics=['mae'
    return model
```

Linear

$$f(x) = x$$

Rectified Linear Unit (ReLU)



$$y = max(0, x)$$

without activation functions, a multilayer perceptron would be a linear combination

$$z = w_1 x + b_1$$

$$y = w_2(z) + b_2$$

$$y = w_2(w_1 x + b_1) + b_2$$

$$y = (w_2 w_1)x + (w_2 b_1) + b_2$$

$$y = (w_3)x + b_3$$

$$z = w_1 x + b_1$$

$$h = max(0, z)$$

$$y = w_2(h) + b_2$$

$$y = w_2(max(0, z)) + b_2$$

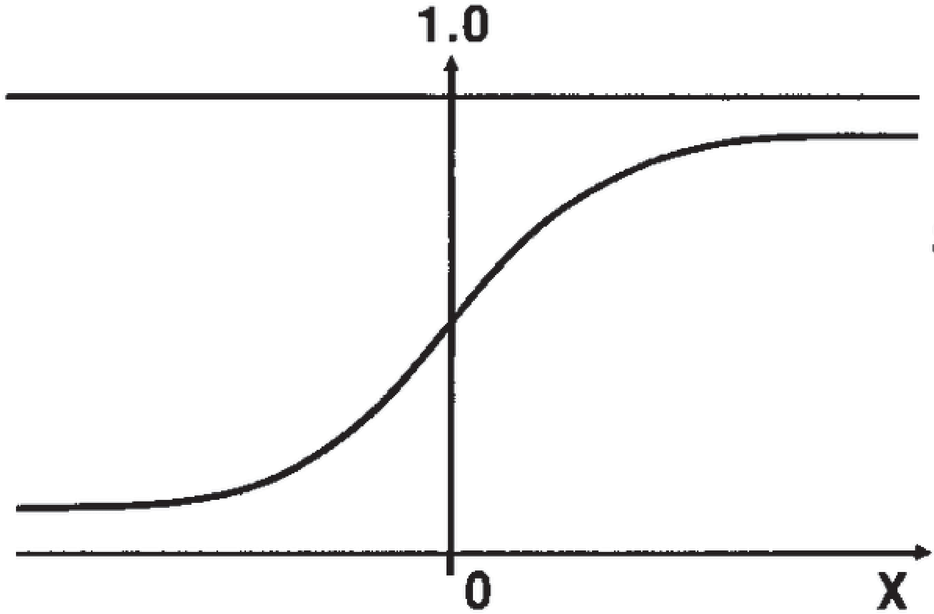
$$y = w_2(max(0, w_1 x + b)) + b_2$$

no activation functions:

$$y = Wx + b$$

with activation functions:

$$y \approx whatever(x)$$



Sigmoid

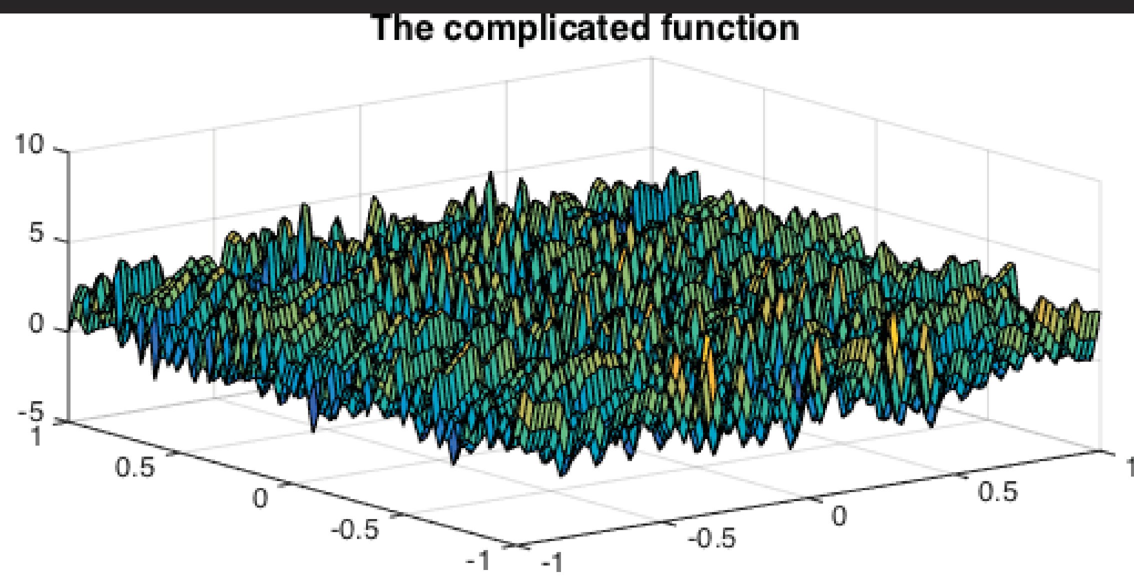
$$f(x) = \frac{1}{1+e^{-x}}$$

Universal Approximation Theorem:

a deep neural network of arbitrary with and a non-polynomial activation function can approximate **any** function

there exists a neural network NN for any function f such that

$$\forall x, f(x) \approx NN(x)$$



Big whoop

Big ~~whoop~~ ***MONEY***

AI triumphs against the world’s top pro team in strategy game Dota 2

It’s the first time an AI has beat a world champion e-sports team.

By Kelsey Piper | Apr 13, 2019, 6:30pm EDT

SHARE

r/MachineLearning

Posts

Posted by u/mippie_moe 8 months ago

437

[D] GPT-3, The $4,600,000 Language Model

Discussion

OpenAI's GPT-3 Language Model Explained

Some interesting take-aways:

- GPT-3 demonstrates that a language model trained on enough data can solve NLP tasks that it has never seen. That is, GPT-3 studies the model as a general solution for many downstream jobs **without fine-tuning**.
- It would take **355 years** to train GPT-3 on a Tesla V100, the fastest GPU on the market.
- It would cost **~$4,600,000** to train GPT-3 on using the lowest cost GPU cloud provider.

216 Comments Give Award Share Save Hide Report 96% Upvoted

Item	GPU years (Volta)	Electricity (MWh)
Initial exploration	20.25	58.94
Paper exploration	13.71	31.49
FFHQ config F	0.23	0.68
Other runs in paper	7.20	16.77
Backup runs left out	4.73	12.08
Video, figures, etc.	0.31	0.82
Public release	4.62	10.82
Total	51.05	131.61

Table 5. Computational effort expenditure and electricity consumption data for this project. The unit for computation is GPU-years on a single NVIDIA V100 GPU — it would have taken approximately 51 years to execute this project using a single GPU.

$$y = max(0, x) = ReLU(x)$$

$$y = max(0, x) = ReLU(x)$$

derivative is cheap to compute

$$y = max(0, x) = ReLU(x)$$

derivative is cheap to compute

$$\frac{\delta ReLU}{\delta x} = 1 \text{ if } x > 0 \text{ else } 0$$

$$y = max(0, x) = ReLU(x)$$

derivative is cheap to compute

$$\frac{\delta ReLU}{\delta x} = 1 \text{ if } x > 0 \text{ else } 0$$

Fast convergence

$y = max(0, x) = ReLU(x)$

derivative is cheap to compute

$\frac{\delta ReLU}{\delta x}$ = 1 if $x > 0$ else 0

Fast convergence

sparsely activated

dying ReLU

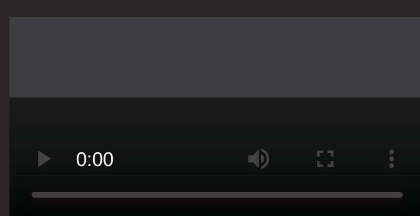
dying ReLU

neurons that only output negative values are never heard from again

dying ReLU

neurons that only output negative values are never heard from again

different versions of ReLU exist that combat this, like LeakyReLU backpropogation



video link

backprop $\neq$ deep learning

backprop $\subset$ deep learning

automatic differentiation

```
deep learning =
backpropagation +
optimization +
stochastic gradient descent +
hyperparameter tuning +
architecture +
loss functions +
weight initialization +
scaling +
...
```

architecture and scale

convolutional neural networks

Yann LeCun

convolutional neural networks

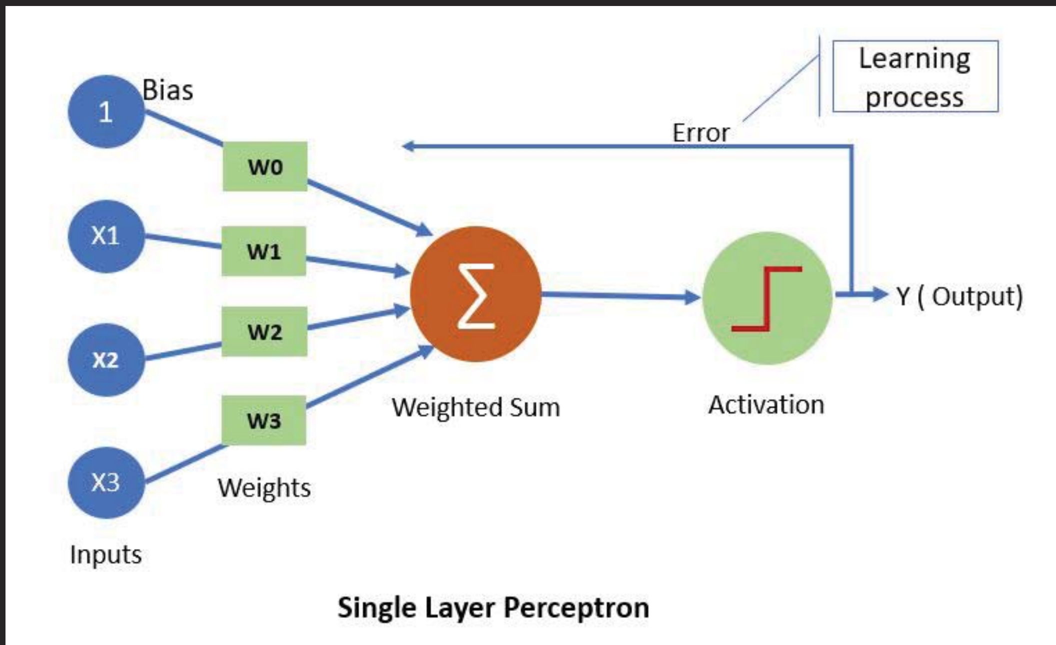
convolutional neural networks

based off mammalian eye

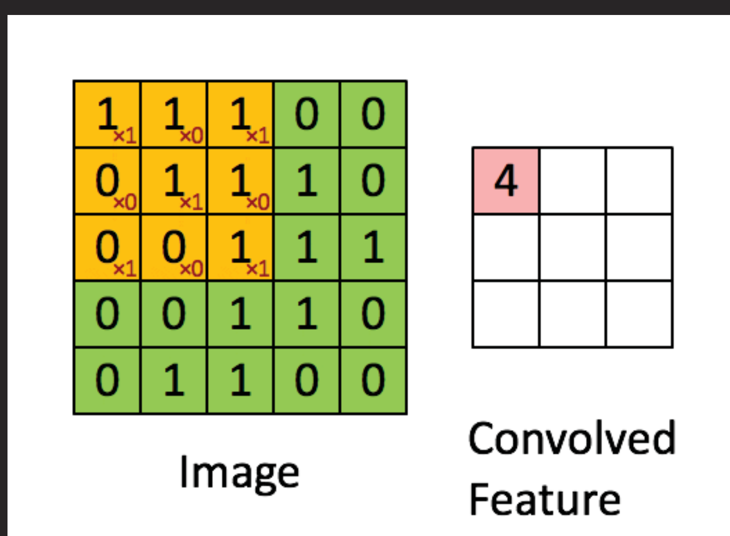
convolutional neural networks

based off mammalian eye

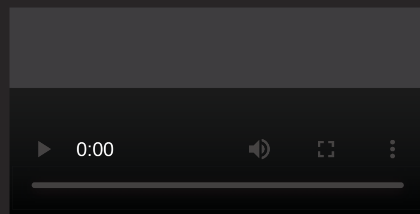
used for computer vision



convolution

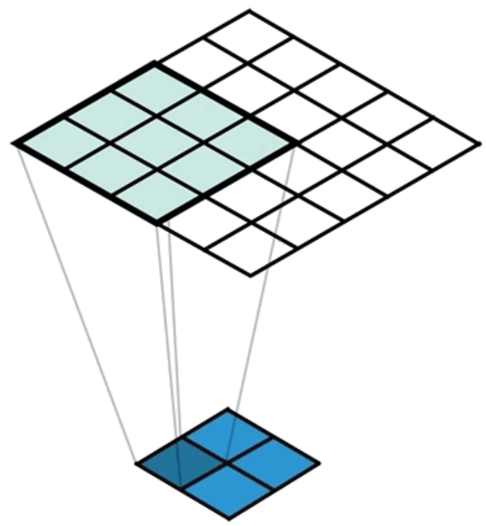


gif link



video link

```python
def make_conv_net():
    inputs = layers.Input(shape=(50, 50, 4))
    layer1 = layers.Conv2D(32, 8, strides=4, activation="relu")(i
    layer2 = layers.Conv2D(64, 4, strides=2, activation="relu")(l
    layer3 = layers.Conv2D(64, 3, strides=1, activation="relu")(l
    layer4 = layers.Flatten()(layer3)
    layer5 = layers.Dense(512, activation="relu")(layer4)
    out = layers.Dense(2, activation="linear")(layer5)
    model = keras.Model(inputs=inputs, outputs=out)
    return model
```



convolutional stride

gif link

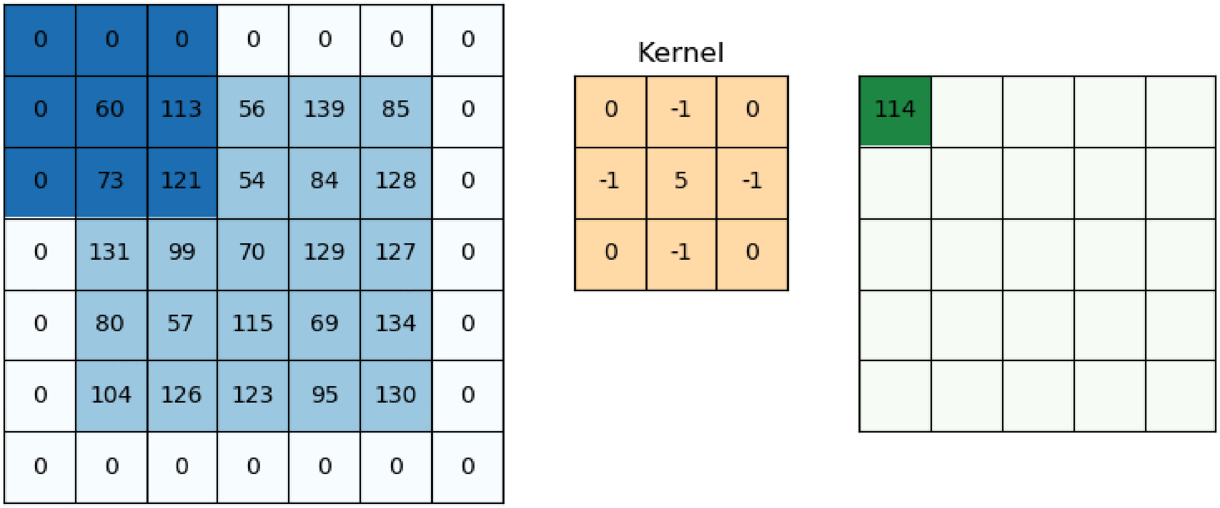
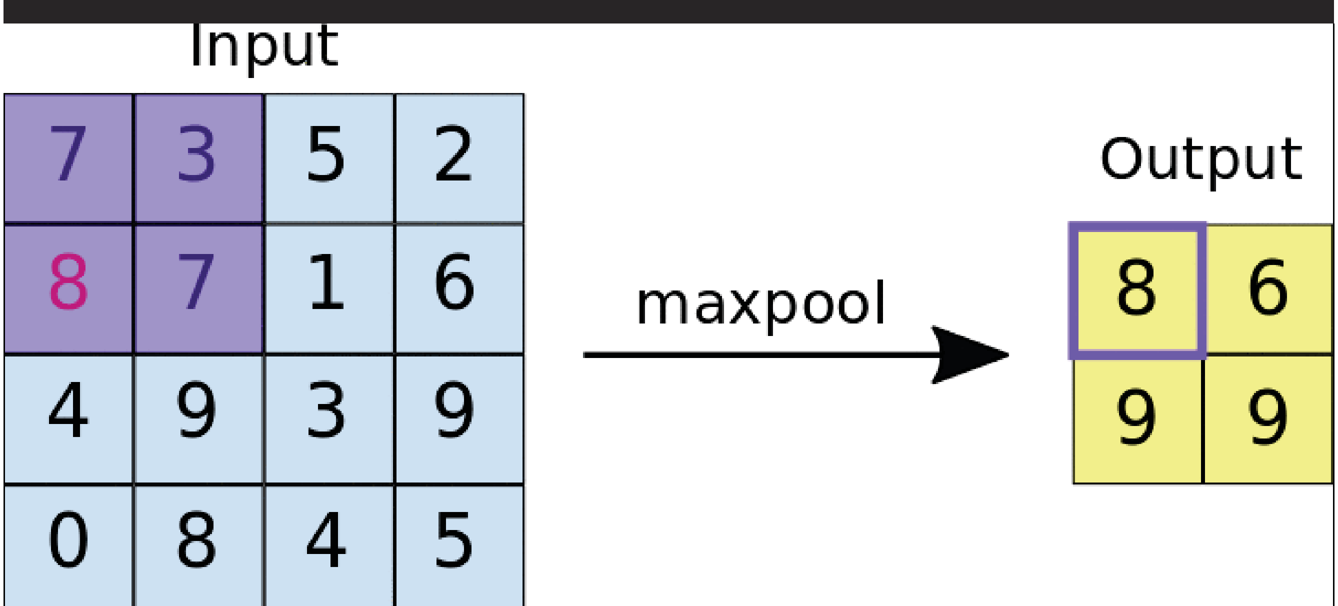


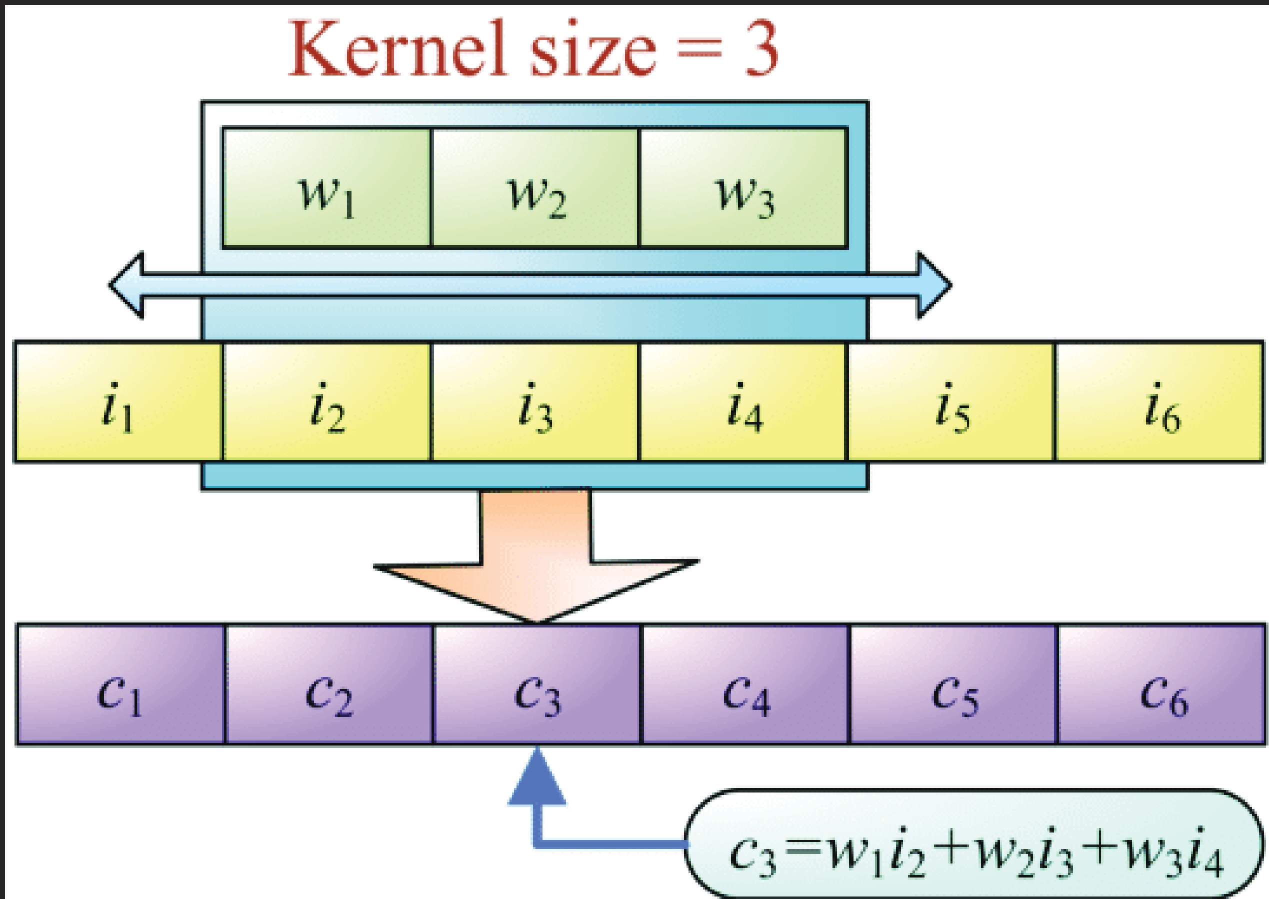
image padding

gif link



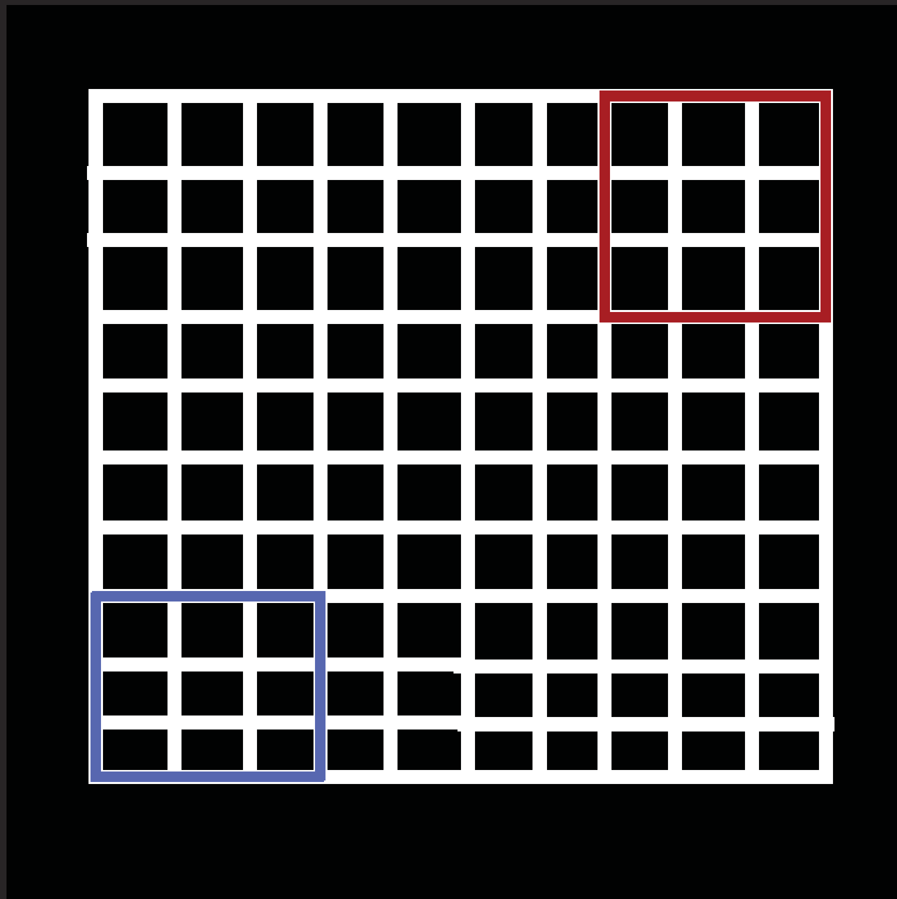
pooling (max)

gif link



highly parallelizable

highly parallelizable
highly parallelizable
highly parallelizable
highly parallelizable
highly parallelizable
highly parallelizable
highly parallelizable



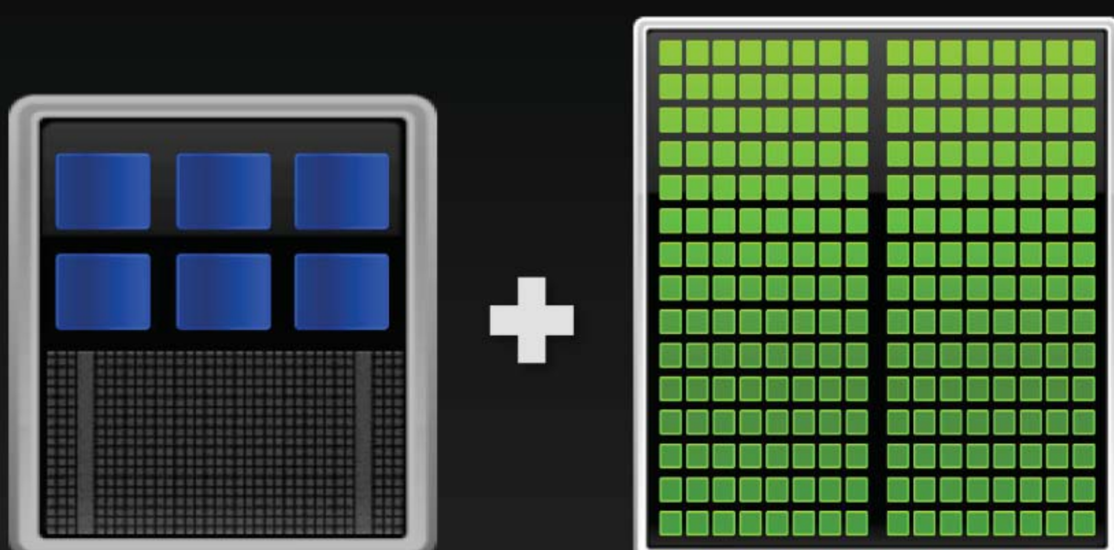
years → months

NVIDIA GeForce RTX 3090

GA102	10496	328	112	24 GB	GDDR6X
GRAPHICS PROCESSOR	CORES	TMUS	ROPS	MEMORY SIZE	MEMORY TYPE

ARM or x86 CPU
Optimized for Few Serial Tasks

GPU Accelerator
Optimized for Many Parallel Tasks



NVIDIA®

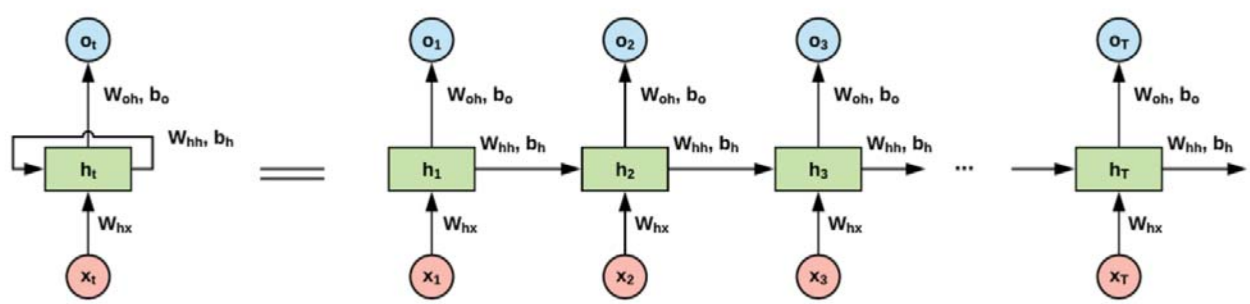
CUDA®

0:00

video link

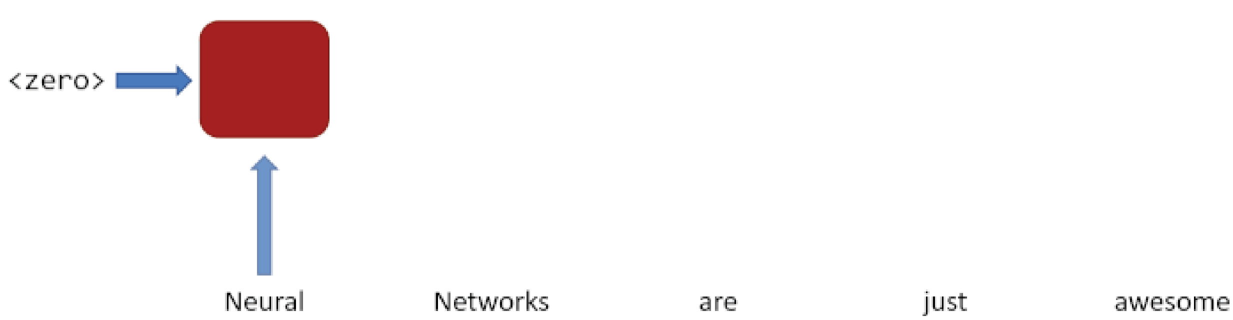
memory and attention

recurrent neural network / elman design

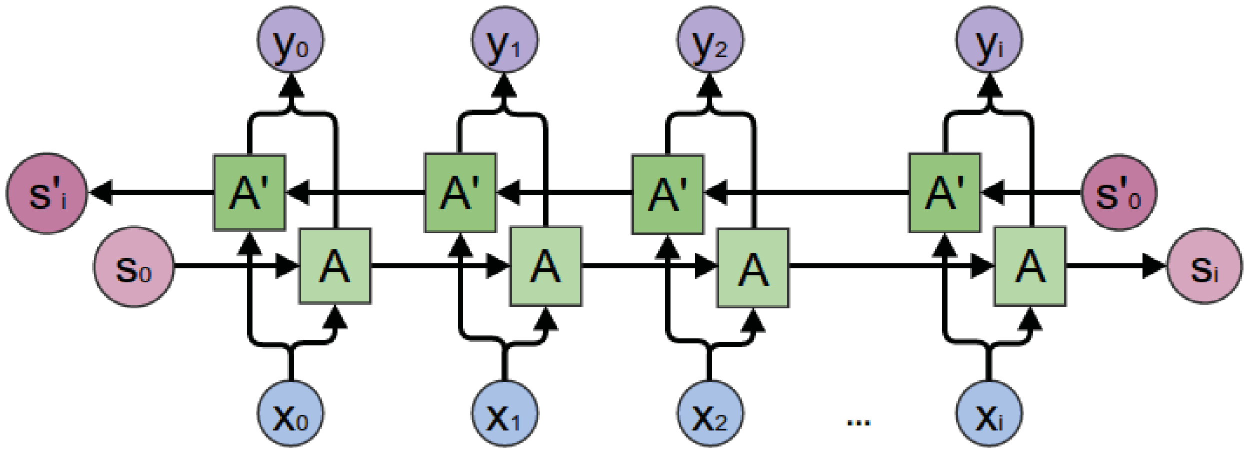


$$h_t = \sigma(W_{hx}x_t + W_{hh}h_{t-1} + b_h)$$

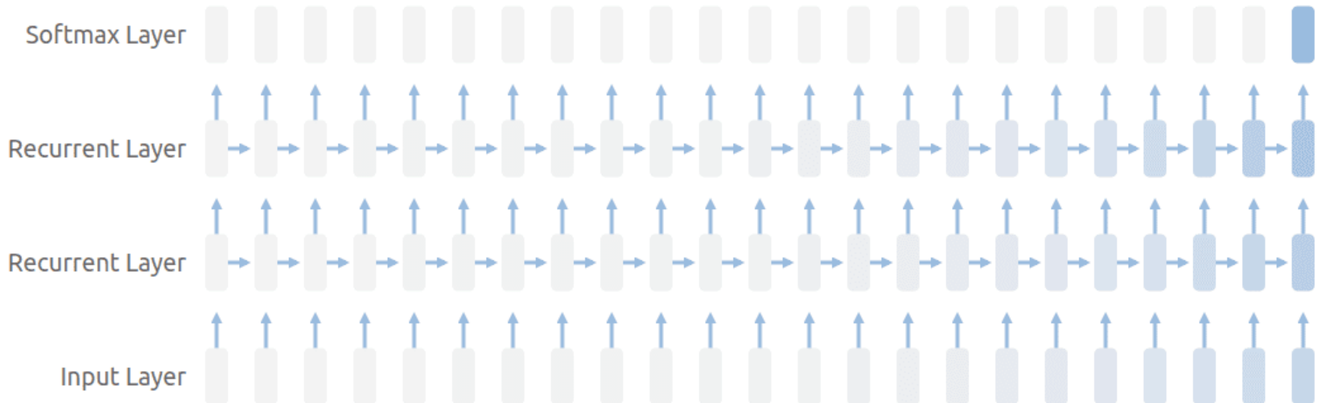
$$o_t = W_{oh}h_t + b_o$$



gif link



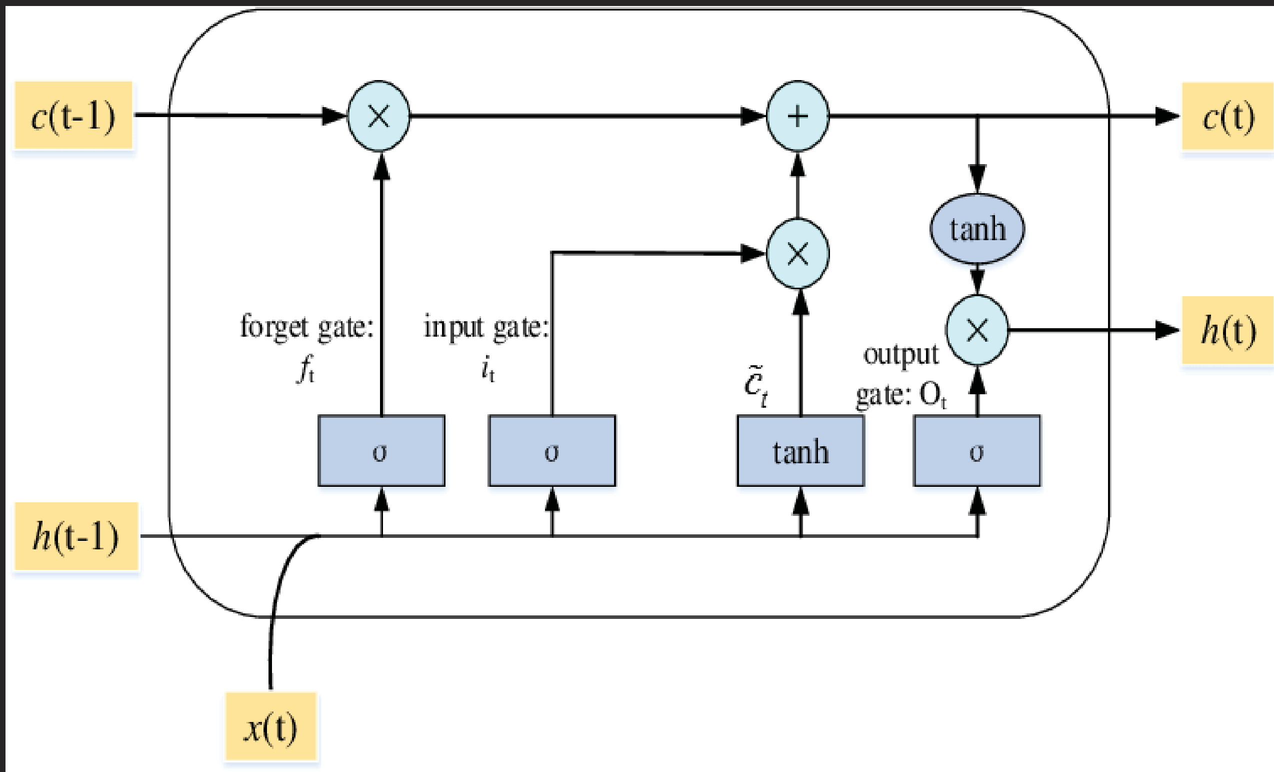
vanishing gradient problem



Vanishing Gradient: where the contribution from the earlier steps becomes insignificant in the gradient for the vanilla RNN unit.

long short-term memory

long short-term memory



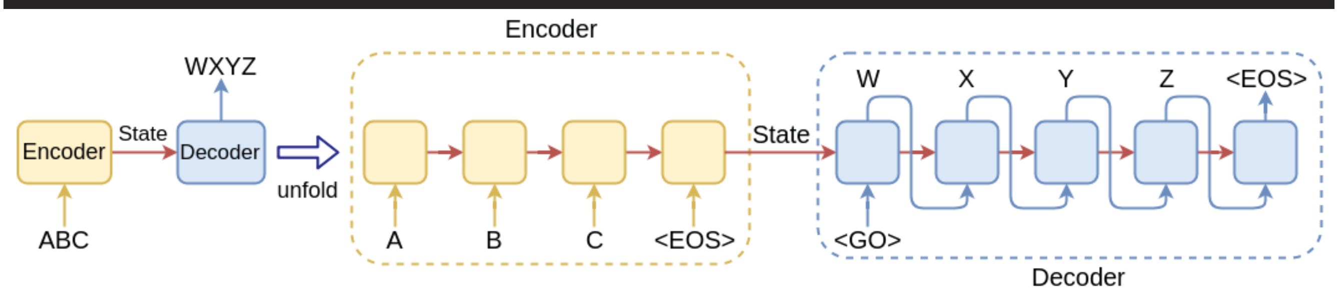
- forget gate: chooses what memory to keep

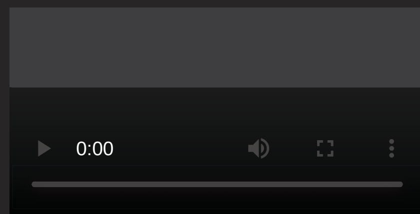
- forget gate: chooses what memory to keep
- input gate: chooses what new memory to add to the current memory

- forget gate: chooses what memory to keep
- input gate: chooses what new memory to add to the current memory
- output gate: produces hidden vector for the next state using memory, current hidden vector, and input

- forget gate: chooses what memory to keep
- input gate: chooses what new memory to add to the current memory
- output gate: produces hidden vector for the next state using memory, current hidden vector, and input

LSTMs are RNNs that read/write to a memory vector





video link

downsides of RNNs/LSTMS

downsides of RNNs/LSTMS

- can only access memory and hidden vector

downsides of RNNs/LSTMS

- can only access memory and hidden vector
- information unlikely to be remembered for a long period of time

downsides of RNNs/LSTMS

- can only access memory and hidden vector
- information unlikely to be remembered for a long period of time
- can't detect different features

attention networks

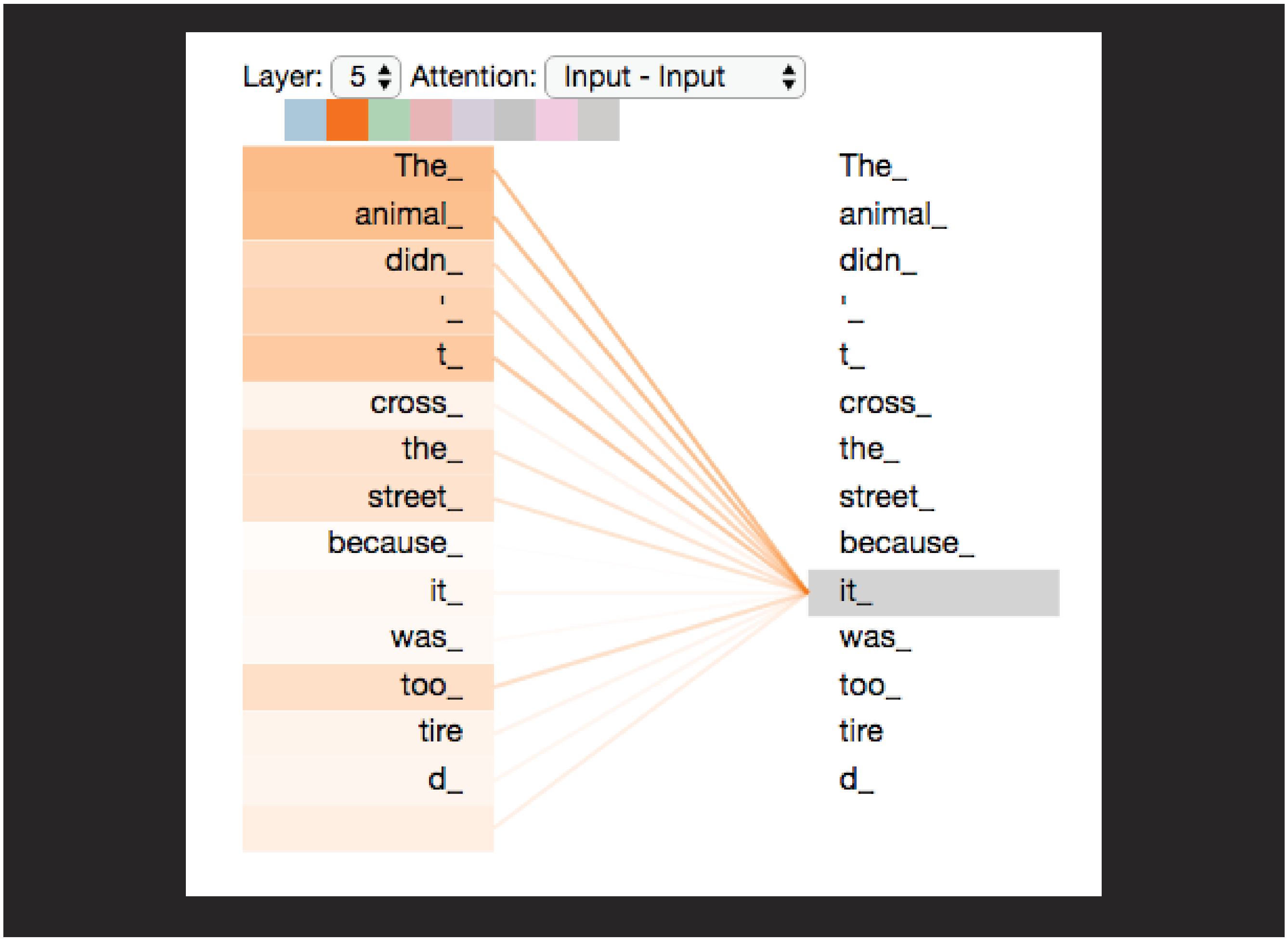
Attention Is All You Need

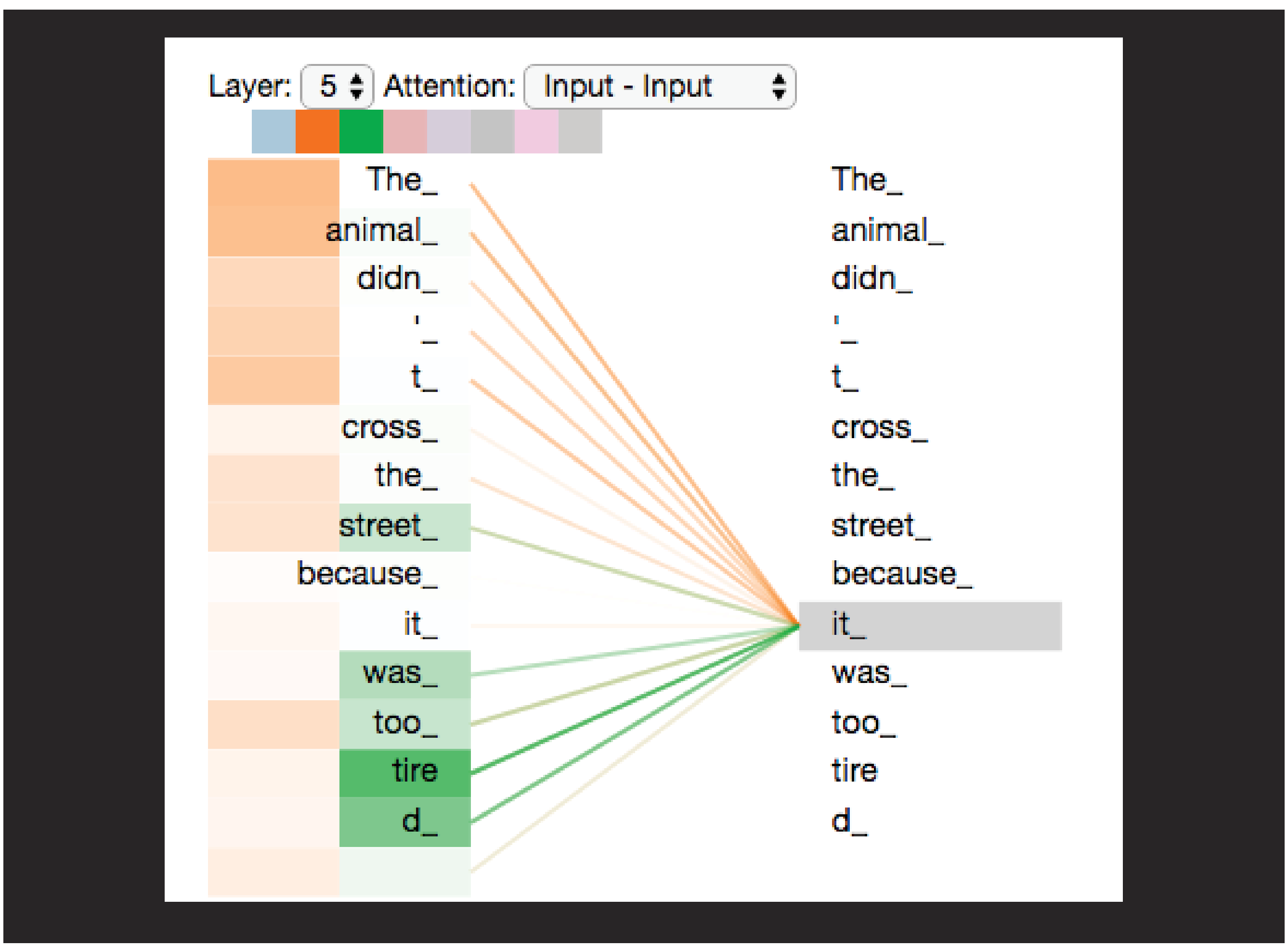
Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

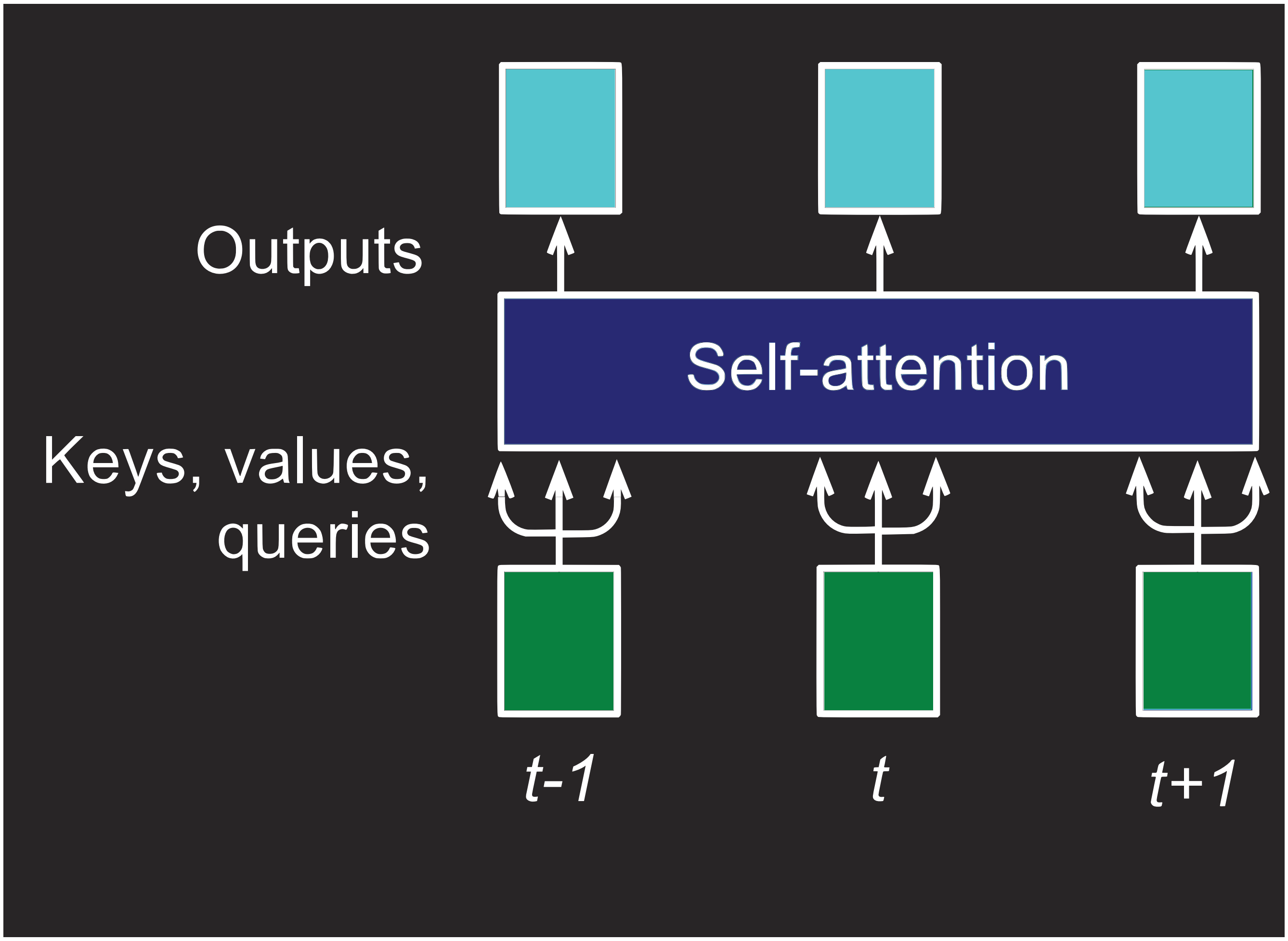
Abstract

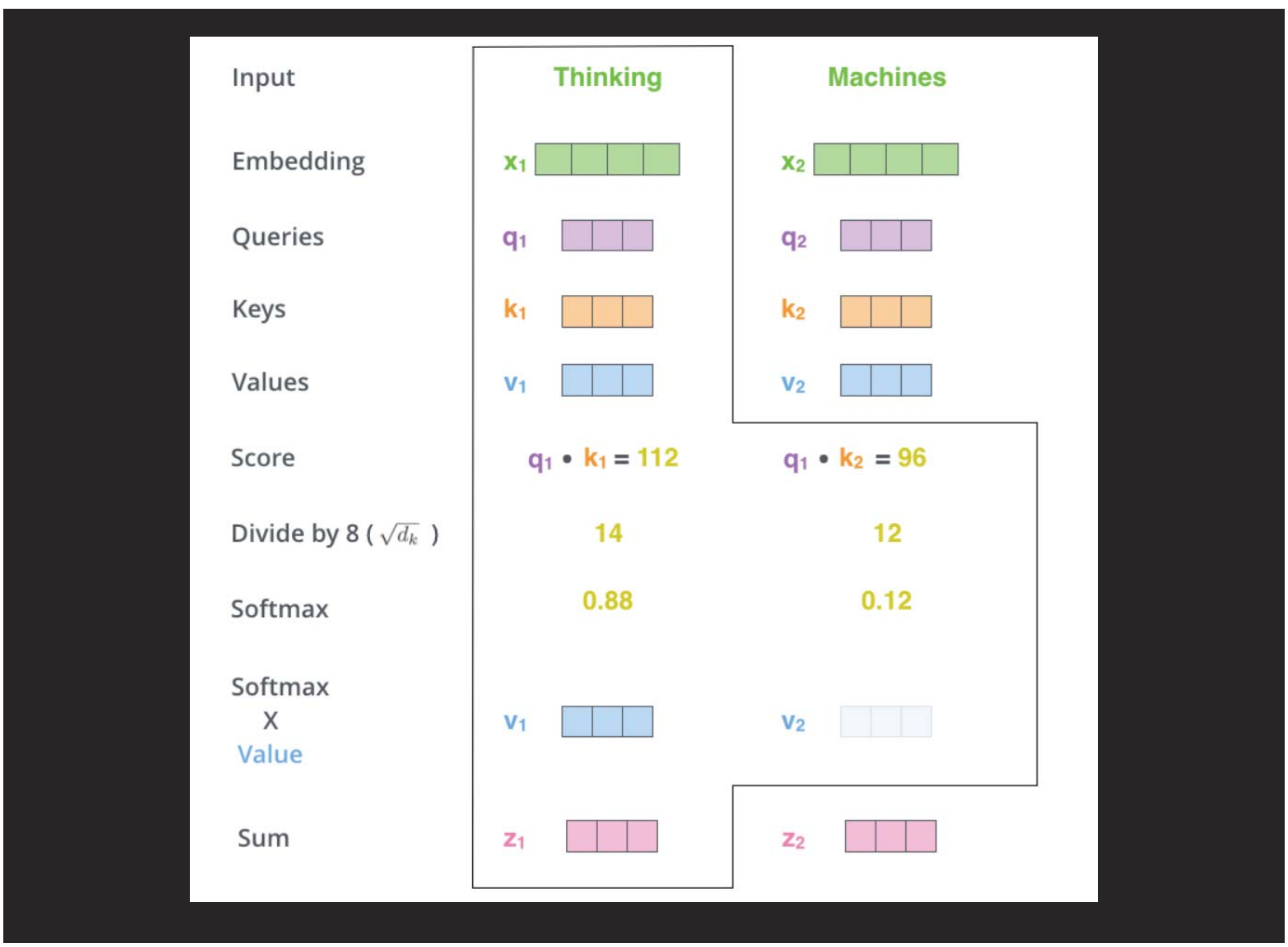
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

self-attention uses the entire input instead of remembering certain information


Layer: 5
Attention: Input - Input
The_
animal_
didn_
'_
t_
cross_
the_
street_
because_
it_
was_
too_
tire
d_


Layer: 5
Attention: Input - Input
The_
animal_
didn_
'_
t_
cross_
the_
street_
because_
it_
was_
too_
tire
d_

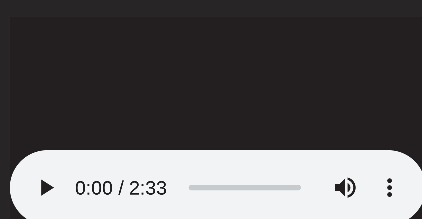

Outputs
Self-attention
Keys, values, queries
t-1
t
t+1


Input
Thinking
Machines
Embedding
x1
x2
Queries
q1
q2
Keys
k1
k2
Values
v1
v2
Score
q1 • k1 = 112
q1 • k2 = 96
Divide by 8 ($\sqrt{d_k}$)
14
12
Softmax
0.88
0.12
Softmax
X
Value
v1
v2
Sum
z1
z2

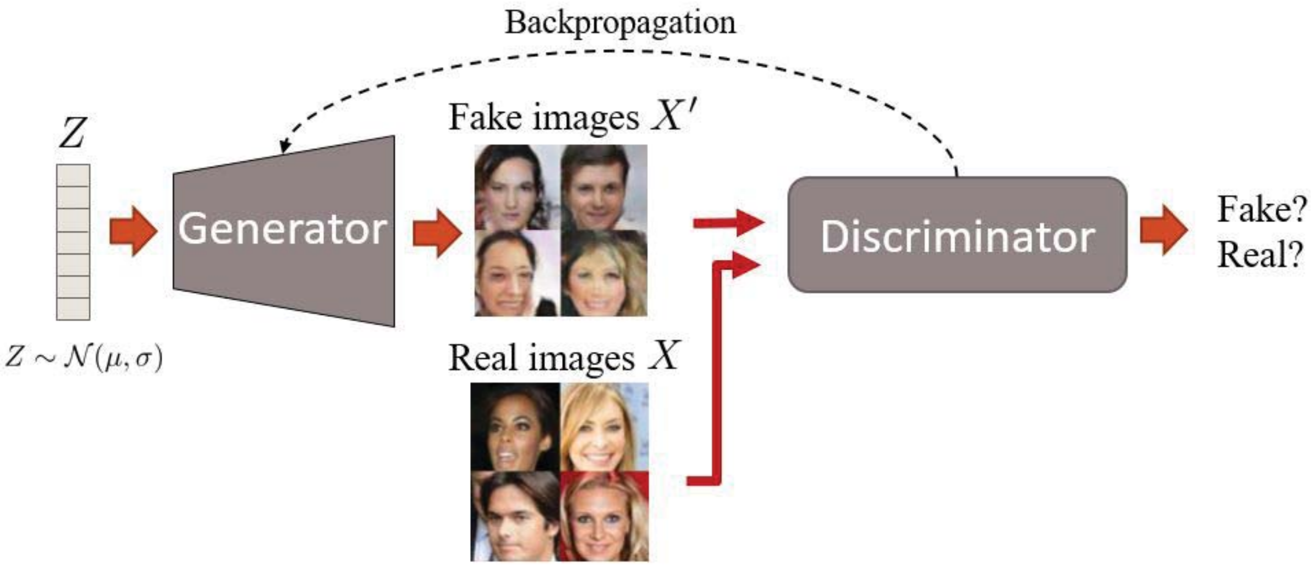
five minute break

generative adversarial networks (GANs)

thispersondoesnotexist.com



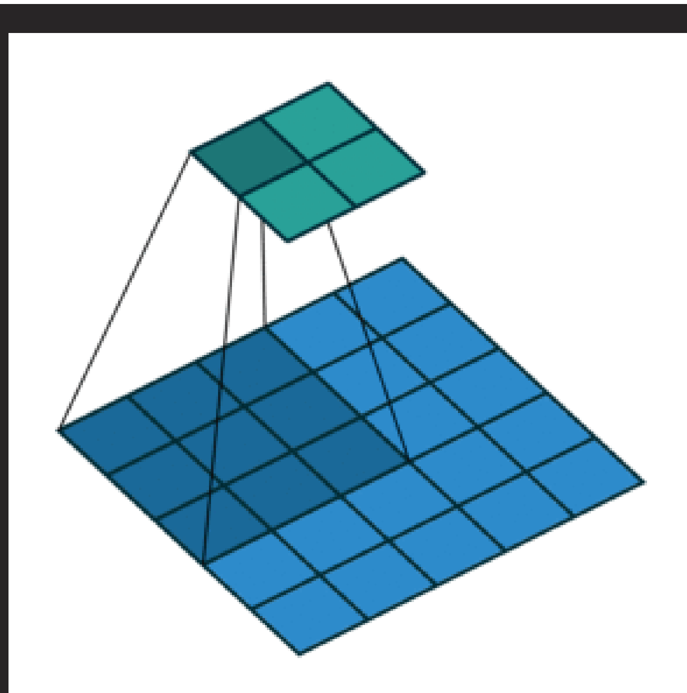
video link


Backpropagation
Z
Generator
Fake images X'
Discriminator
Fake?
Real?
$Z \sim \mathcal{N}(\mu, \sigma)$
Real images X


CHARACTER CREATION
APPEARANCE
Presets
Skin
Face
Brow
Eyes
Ears
Nose
Mouth
Jaw
Hair
Facial Hair
Decals
NOSE
Width
0.00
Height
-1.00
Depth
-1.00
Bridge
0.00
RESET ALL
ADJUST
BACK
Nose shapes and adjustments.
ENV
ZOOM
ROTATE
RANDOMIZE
Captured on PS5

```
class Generator(nn.Module):
    def __init__(self, ngpu):
        super(Generator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is Z, going into a convolution
            nn.ConvTranspose2d( nz, ngf * 8, 4, 1, 0, bias=False),
            nn.BatchNorm2d(ngf * 8),
            nn.ReLU(True),
            # state size. (ngf*8) x 4 x 4
            nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 4),
            nn.ReLU(True),
            # state size. (ngf*4) x 8 x 8
            nn.ConvTranspose2d( ngf * 4, ngf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 2),
            nn.ReLU(True),
            # state size. (ngf*2) x 16 x 16
            nn.ConvTranspose2d( ngf * 2, ngf, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf),
            nn.ReLU(True),
            # state size. (ngf) x 32 x 32
            nn.ConvTranspose2d( ngf, nc, 4, 2, 1, bias=False),
            nn.Tanh()
```

[redacted]



transposed convolution

gif link

```
class Discriminator(nn.Module):
    def __init__(self, nc, ndf):
        super(Discriminator, self).__init__()

        self.main = nn.Sequential(

            # state size. (ndf) x 64 x 64
            nn.Conv2d(ndf, ndf * 2, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.25),
            nn.BatchNorm2d(ndf * 2),

            # state size. (ndf) x 32 x 32
            nn.Conv2d(ndf * 2, ndf * 4, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            nn.BatchNorm2d(ndf * 4),
            nn.Dropout(0.25),

            # state size. (ndf*2) x 16 x 16
            nn.Conv2d(ndf * 4, ndf * 8, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            nn.BatchNorm2d(ndf * 8),
            nn.Dropout(0.25),
```

```
# Use Binary Cross Entropy Loss
criterion = nn.BCELoss()

for epoch in range(num_epochs):
    for iteration, batch in enumerate(dataloader):

        real_data = sample["data"]
        sample_size = len(real_data)
        latent_noise = torch.randn(sample_size, nz, 1, 1)
        real_labels = torch.ones(sample_size)
        fake_labels = torch.zeros(sample_size)

        # Generate Fake Data
        latent_noise = torch.randn(sample_size, nz, 1, 1)
        fake_data = G(latent_noise)
        # Test how well the generator fooled the discriminator
        gen_loss = criterion(D(fake_data), real_labels)
        # Update Generator
        gen_loss.backward()
        optimizerG.step()

        # Test Discriminator on real data and fake data
        d_real, d_fake = D(real_data), D(fake_data)
        real_loss = criterion(d_real, real_labels)
```

recap

deep neural networks

deep neural networks

- can approximate any function

deep neural networks

- can approximate any function
- are naturally parallel

current dominant architectures (convolution & attention) use different weights for different features in parallel

convolutional neural networks use a small set of weights to find local patterns over the entire input convolutional neural networks use a small set of weights to find local patterns over the entire input

- computer vision, natural language processing

recurrent neural networks use x_t and h_t to compute h_{t+1}

recurrent neural networks use x_t and h_t to compute h_{t+1}

- can learn temporal relations

recurrent neural networks use x_t and h_t to compute h_{t+1}

- can learn temporal relations
- sequence processing

attention networks combine relevant parts of a input sequence to create an output sequence of contexts

attention networks combine relevant parts of a input sequence to create an output sequence of contexts

- better performance than recurrent models attention networks combine relevant parts of a input sequence to create an output sequence of contexts

- better performance than recurrent models
- different attention heads look for different features in parallel

- GANs generate new data given a compressed representation

- GANs generate new data given a compressed representation
- deep learning is neat