

# Neural Networks II: Deep Learning

Mengye Ren

NYU

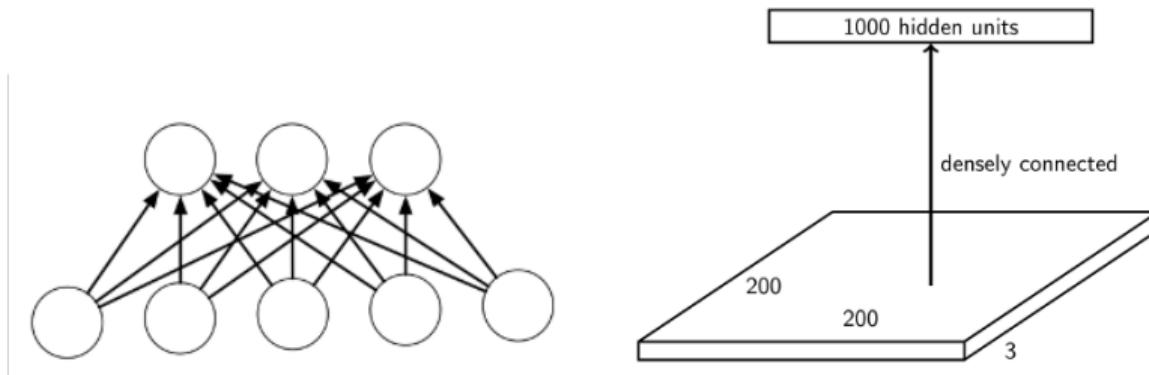
Nov 28, 2023

## Fully connected vs. locally connected

- So far we apply a layer where all output neurons are connected to all input neurons.
- In matrix form,  $z = Wx$ .
- This is also called a fully connected layer or a dense layer or a linear layer.

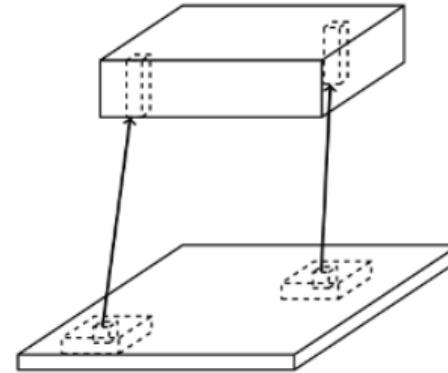
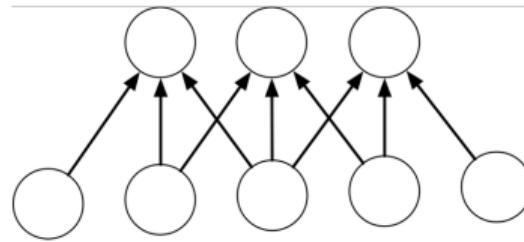
## Fully connected vs. locally connected

- So far we apply a layer where all output neurons are connected to all input neurons.
- In matrix form,  $z = Wx$ .
- This is also called a fully connected layer or a dense layer or a linear layer.
- For  $200 \times 200$  image and 1000 hidden units, the matrix of a single layer will have 40M parameters!



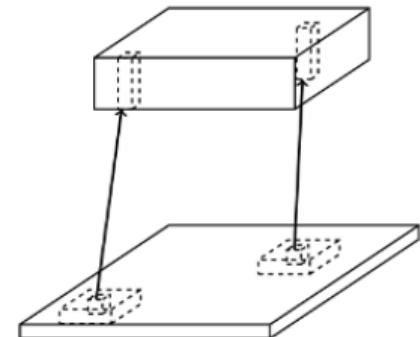
## Fully connected vs. locally connected

- An alternative strategy is to use local connection.
- For neuron  $i$ , only connects to its neighborhood (e.g.  $[i+k, i-k]$ )
- For images, we index neurons with three dimensions  $i$ ,  $j$ , and  $c$ .
- $i$  = vertical index,  $j$  = horizontal index,  $c$  = channel index.



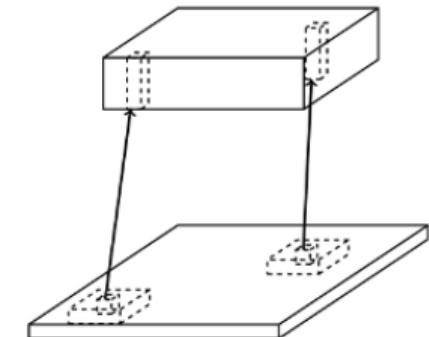
## Local connection patterns

- The typical image input layer has 3 channels R G B for color or 1 channel for grayscale.
- The hidden layers may have  $C$  channels, at each spatial location  $(i, j)$ .



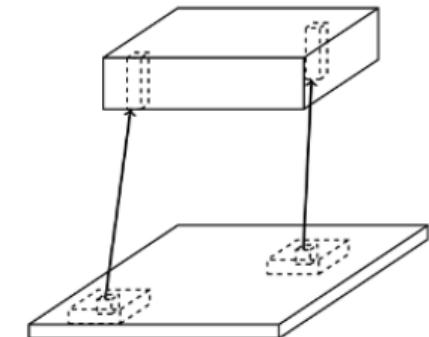
## Local connection patterns

- The typical image input layer has 3 channels R G B for color or 1 channel for grayscale.
- The hidden layers may have  $C$  channels, at each spatial location  $(i, j)$ .
- Now each hidden neuron  $z_{i,j,c}$  receives inputs from  $x_{i \pm k, j \pm k, \cdot}$ .
- $k$  is the “kernel” size - do not confuse with the other kernel we learned.
- $$z_{i,j,c} = \sum_{i' \in [i \pm k], j' \in [j \pm k], c'} x_{i'j'c'} w_{i,j,i'-i,j'-j,c',c}$$



## Local connection patterns

- The typical image input layer has 3 channels R G B for color or 1 channel for grayscale.
- The hidden layers may have  $C$  channels, at each spatial location  $(i, j)$ .
- Now each hidden neuron  $z_{i,j,c}$  receives inputs from  $x_{i \pm k, j \pm k, \cdot}$ .
- $k$  is the “kernel” size - do not confuse with the other kernel we learned.
- $$z_{i,j,c} = \sum_{i' \in [i \pm k], j' \in [j \pm k], c'} x_{i'j'c'} w_{i,j,i'-j,j'-j,c',c}$$
- The spatial awareness (receptive field) of the neighborhood grows bigger as we go deeper.



## Weight sharing

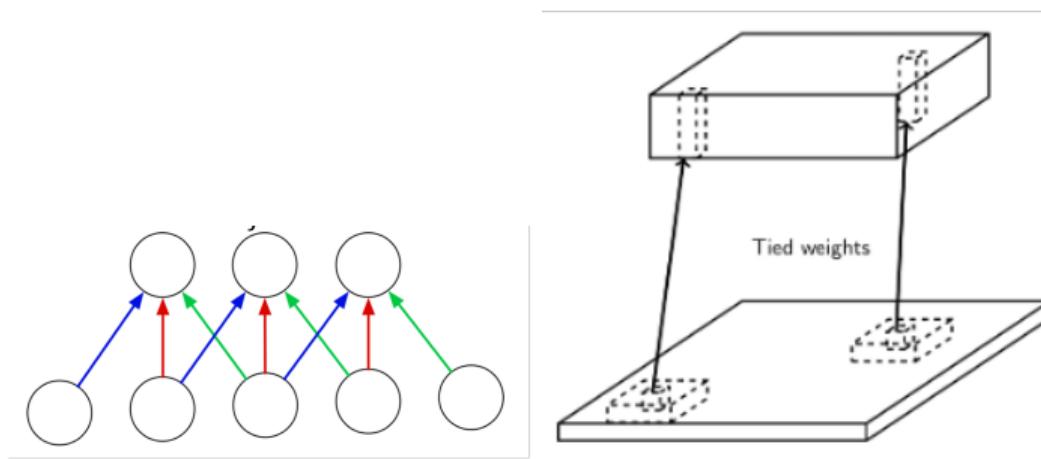
- Still a lot of weights: If we have 100 channels in the second layer, then  
 $200 \times 200 \times 3 \times 100 = 12M$

## Weight sharing

- Still a lot of weights: If we have 100 channels in the second layer, then  $200 \times 200 \times 3 \times 100 = 12M$
- Local information is the same regardless of the position of an element.

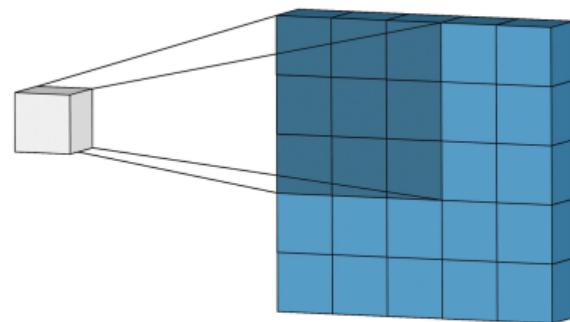
## Weight sharing

- Still a lot of weights: If we have 100 channels in the second layer, then  $200 \times 200 \times 3 \times 100 = 12M$
- Local information is the same regardless of the position of an element.
- Solution: We can tie the weights at different locations.



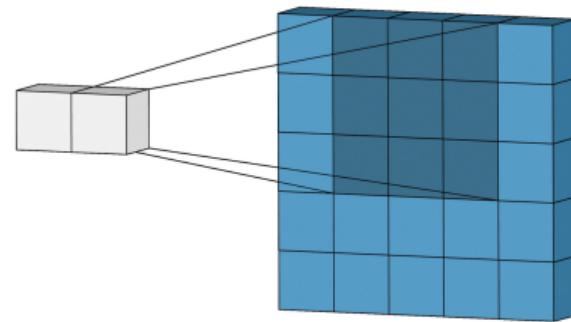
## 2D convolution

- Using the same weight connections for each activation spatial location works like the “filtering operation” or “convolution”
- The neighborhood window is the filter window.
- The weight connection is called “convolution filter”
- $$z_{i,j,c} = \sum_{i' \in [i \pm k], j' \in [j \pm k], c'} x_{i'j'c'} w_{i-i', j-j', c'}$$



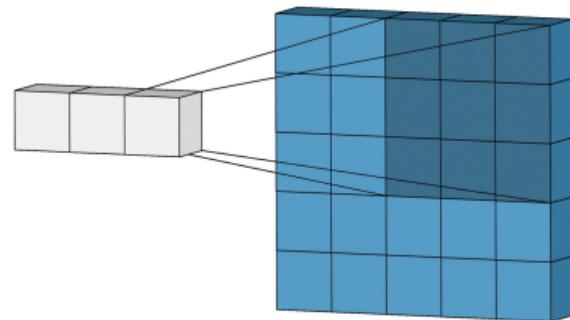
## 2D convolution

- Using the same weight connections for each activation spatial location works like the “filtering operation” or “convolution”
- The neighborhood window is the filter window.
- The weight connection is called “convolution filter”
- $$z_{i,j,c} = \sum_{i' \in [i \pm k], j' \in [j \pm k], c'} x_{i'j'c'} w_{i-i', j-j', c'}$$



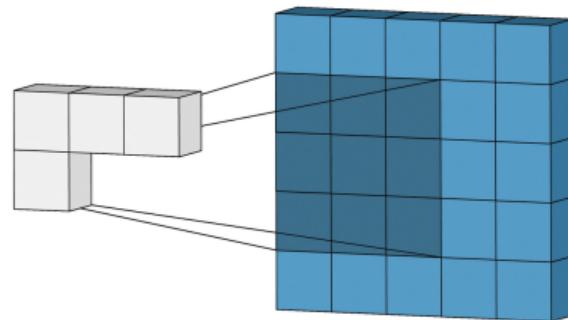
## 2D convolution

- Using the same weight connections for each activation spatial location works like the “filtering operation” or “convolution”
- The neighborhood window is the filter window.
- The weight connection is called “convolution filter”
- $$z_{i,j,c} = \sum_{i' \in [i \pm k], j' \in [j \pm k], c'} x_{i'j'c'} w_{i-i', j-j', c'}$$



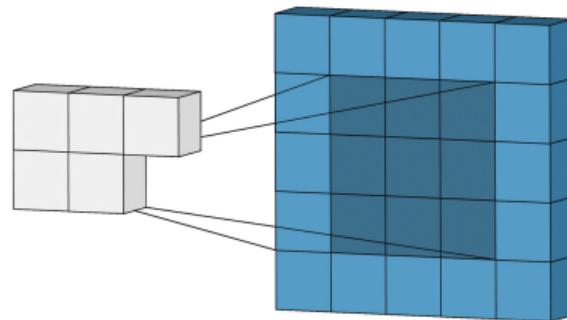
## 2D convolution

- Using the same weight connections for each activation spatial location works like the “filtering operation” or “convolution”
- The neighborhood window is the filter window.
- The weight connection is called “convolution filter”
- $$z_{i,j,c} = \sum_{i' \in [i \pm k], j' \in [j \pm k], c'} x_{i'j'c'} w_{i-i', j-j', c'}$$



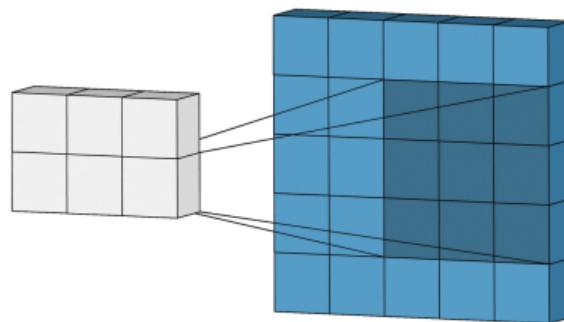
## 2D convolution

- Using the same weight connections for each activation spatial location works like the “filtering operation” or “convolution”
- The neighborhood window is the filter window.
- The weight connection is called “convolution filter”
- $$z_{i,j,c} = \sum_{i' \in [i \pm k], j' \in [j \pm k], c'} x_{i'j'c'} w_{i-i', j-j', c'}$$



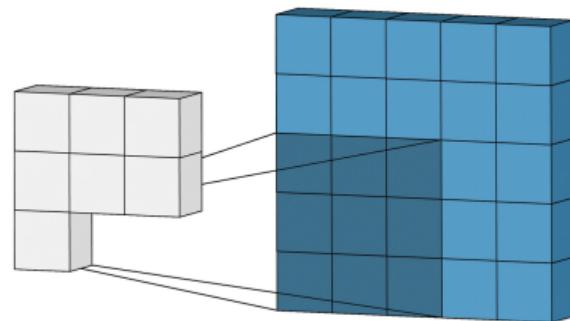
## 2D convolution

- Using the same weight connections for each activation spatial location works like the “filtering operation” or “convolution”
- The neighborhood window is the filter window.
- The weight connection is called “convolution filter”
- $$z_{i,j,c} = \sum_{i' \in [i \pm k], j' \in [j \pm k], c'} x_{i'j'c'} w_{i-i', j-j', c'}$$



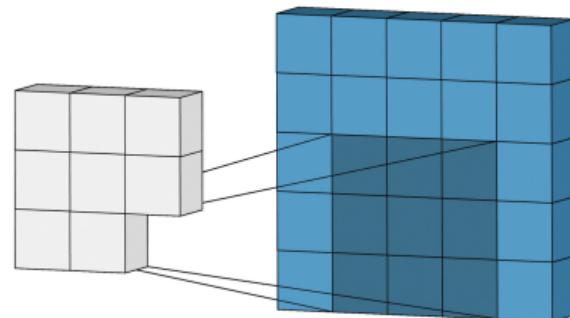
## 2D convolution

- Using the same weight connections for each activation spatial location works like the “filtering operation” or “convolution”
- The neighborhood window is the filter window.
- The weight connection is called “convolution filter”
- $$z_{i,j,c} = \sum_{i' \in [i \pm k], j' \in [j \pm k], c'} x_{i'j'c'} w_{i-i', j-j', c'}$$



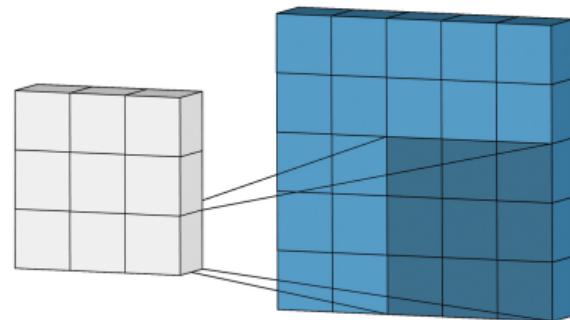
## 2D convolution

- Using the same weight connections for each activation spatial location works like the “filtering operation” or “convolution”
- The neighborhood window is the filter window.
- The weight connection is called “convolution filter”
- $$z_{i,j,c} = \sum_{i' \in [i \pm k], j' \in [j \pm k], c'} x_{i'j'c'} w_{i-i', j-j', c'}$$



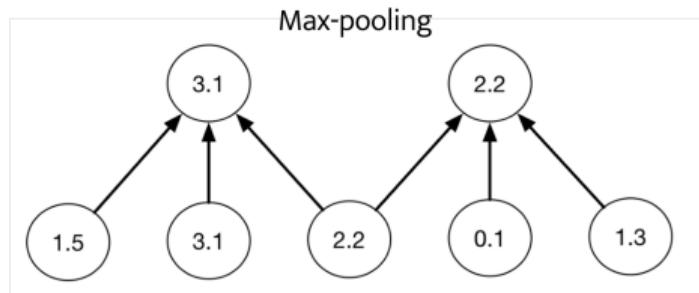
## 2D convolution

- Using the same weight connections for each activation spatial location works like the “filtering operation” or “convolution”
- The neighborhood window is the filter window.
- The weight connection is called “convolution filter”
- $$z_{i,j,c} = \sum_{i' \in [i \pm k], j' \in [j \pm k], c'} x_{i'j'c'} w_{i-i', j-j', c'}$$



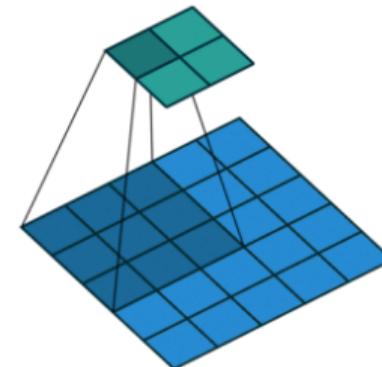
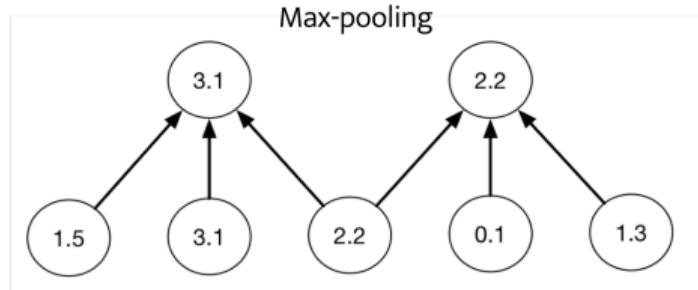
# Pooling

- Need to summarize global information more efficiently.
- Pooling reduces image / activation dimensions.
- Max-pooling or average-pooling



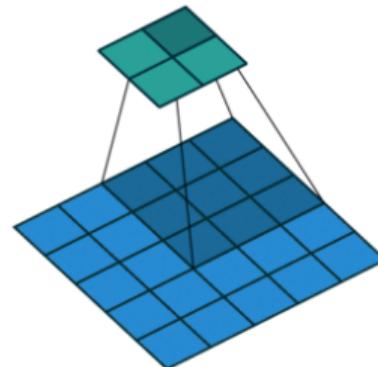
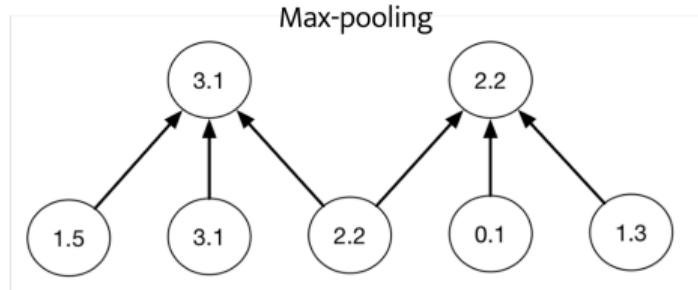
# Pooling

- Need to summarize global information more efficiently.
- Pooling reduces image / activation dimensions.
- Max-pooling or average-pooling
- You can also perform a “strided” convolution by jumping multiple steps.



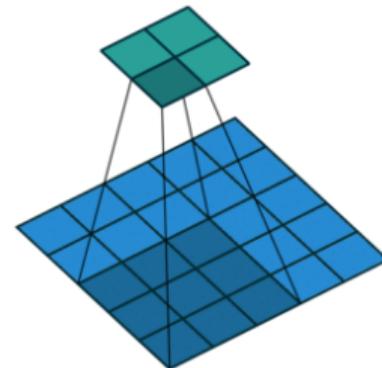
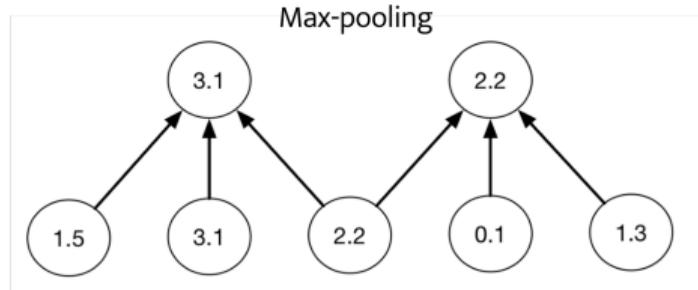
# Pooling

- Need to summarize global information more efficiently.
- Pooling reduces image / activation dimensions.
- Max-pooling or average-pooling
- You can also perform a “strided” convolution by jumping multiple steps.



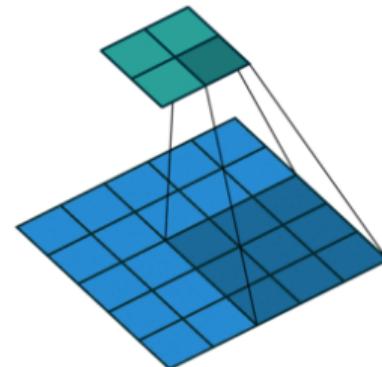
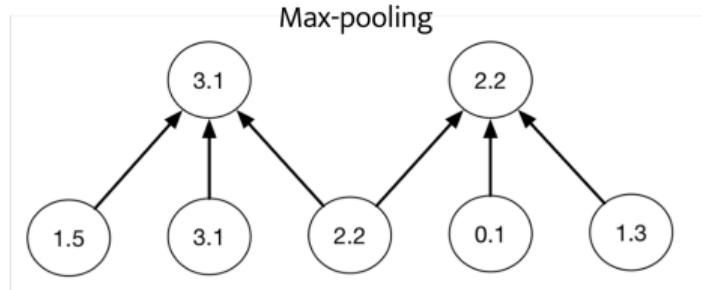
# Pooling

- Need to summarize global information more efficiently.
- Pooling reduces image / activation dimensions.
- Max-pooling or average-pooling
- You can also perform a “strided” convolution by jumping multiple steps.

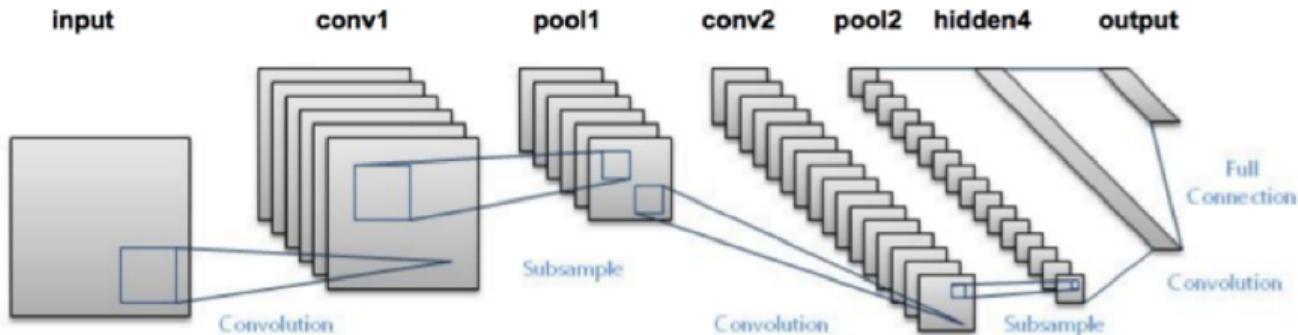


# Pooling

- Need to summarize global information more efficiently.
- Pooling reduces image / activation dimensions.
- Max-pooling or average-pooling
- You can also perform a “strided” convolution by jumping multiple steps.



# Assembling together: LeNet



- Used by USPS to read post code in the 90s.

## Historical development

- LeNet has worked and being put to practice in the 1990s.

## Historical development

- LeNet has worked and been put to practice in the 1990s.
- Neural networks for images start to dominate in the last 10 years (starting 2012) for understanding general high resolution natural images.

## Historical development

- LeNet has worked and being put to practice in the 1990s.
- Neural networks for images start to dominate in the last 10 years (starting 2012) for understanding general high resolution natural images.
- During the years:
  - Neural networks were difficult to work
  - People focused on feature engineering
  - Then apply SVM or random forest (e.g. AdaBoost face detector)
  - What has changed?

## Gradient learning conditioning

# Optimization challenges

- Larger images require deeper networks (more stages of processing at different resolutions)
- Optimizing deeper layers of networks is not trivial.
- Loss often stalls or blows up.

# Optimization challenges

- Larger images require deeper networks (more stages of processing at different resolutions)
- Optimizing deeper layers of networks is not trivial.
- Loss often stalls or blows up.
- Why?

# Optimization challenges

- Larger images require deeper networks (more stages of processing at different resolutions)
- Optimizing deeper layers of networks is not trivial.
- Loss often stalls or blows up.
- Why?
  - Backpropagation: multiplying the Jacobian  $\frac{\partial y}{\partial x}$  by each layer.
  - If the maximum singular value of each layer of Jacobian is less than 1: then the gradient will converge to 0 with more layers.
  - If the greater than 1: then the gradient will explode with more layers.
  - The bottom (input) layer may get 0 or infinite gradients.

# Weight initialization

- Even with a few layers ( $>3$ ), optimization is still hard.

## Weight initialization

- Even with a few layers ( $>3$ ), optimization is still hard.
- If weight initialization is bad (too small or too big), then optimization is hard to kick off.

## Weight initialization

- Even with a few layers ( $>3$ ), optimization is still hard.
- If weight initialization is bad (too small or too big), then optimization is hard to kick off.
- Consider the distribution of whole dataset in the activation space.
  - Intuition: upon initialization, the variance of the activations should stay the same across every layer.

## Kaiming Initialization

- Suppose each neuron and weight connection are sampling from a random distribution.

## Kaiming Initialization

- Suppose each neuron and weight connection are sampling from a random distribution.
- At  $l$ -th layer,  $\text{Var}[z_l] = n_l \text{Var}[w_l x_l]$  ( $n_l$  = num. input neurons to  $l$ -th layer)

---

<sup>1</sup>He et al. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet. ICCV, 2015.

## Kaiming Initialization

- Suppose each neuron and weight connection are sampling from a random distribution.
- At  $l$ -th layer,  $\text{Var}[z_l] = n_l \text{Var}[w_l x_l]$  ( $n_l$  = num. input neurons to  $l$ -th layer)
- If we suppose that ReLU is used as the activation, and  $w_l$  is symmetric and zero-mean,  
 $x_{l+1} = \frac{1}{2} \text{Var}[z_l]$ .

---

<sup>1</sup>He et al. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet. ICCV, 2015.

## Kaiming Initialization

- Suppose each neuron and weight connection are sampling from a random distribution.
- At  $l$ -th layer,  $\text{Var}[z_l] = n_l \text{Var}[w_l x_l]$  ( $n_l$  = num. input neurons to  $l$ -th layer)
- If we suppose that ReLU is used as the activation, and  $w_l$  is symmetric and zero-mean,  
 $x_{l+1} = \frac{1}{2} \text{Var}[z_l]$ .
- Putting altogether,  $x_{l+1} = \frac{1}{2} n_l \text{Var}[w_l] \text{Var}[x_l]$ .

---

<sup>1</sup>He et al. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet. ICCV, 2015.

## Kaiming Initialization

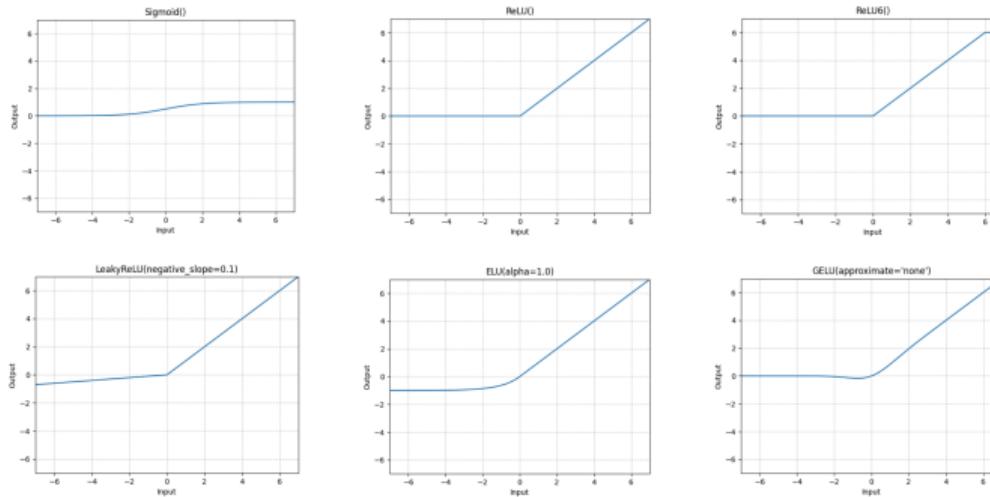
- Suppose each neuron and weight connection are sampling from a random distribution.
- At  $l$ -th layer,  $\text{Var}[z_l] = n_l \text{Var}[w_l x_l]$  ( $n_l$  = num. input neurons to  $l$ -th layer)
- If we suppose that ReLU is used as the activation, and  $w_l$  is symmetric and zero-mean,  $x_{l+1} = \frac{1}{2} \text{Var}[z_l]$ .
- Putting altogether,  $x_{l+1} = \frac{1}{2} n_l \text{Var}[w_l] \text{Var}[x_l]$ .
- To make the variance constant, we need  $\frac{1}{2} n_l \text{Var}[w_l] = 1$ ,  $\text{Std}[w_l] = \sqrt{2/n_l}$ <sup>1</sup>.

---

<sup>1</sup>He et al. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet. ICCV, 2015.

# Activation functions

- ReLU was proposed in 2009-2010<sup>23</sup>, and was successfully used in AlexNet in 2012<sup>4</sup>.
- Address the vanishing gradient issue in activations, comparing to sigmoid or tanh.



<sup>2</sup>Jarrett et al. What is the Best Multi-Stage Architecture for Object Recognition? ICCV, 2009.

<sup>3</sup>Nair & Hinton/ Rectified Linear Units Improve Restricted Boltzmann Machines. ICML, 2010.

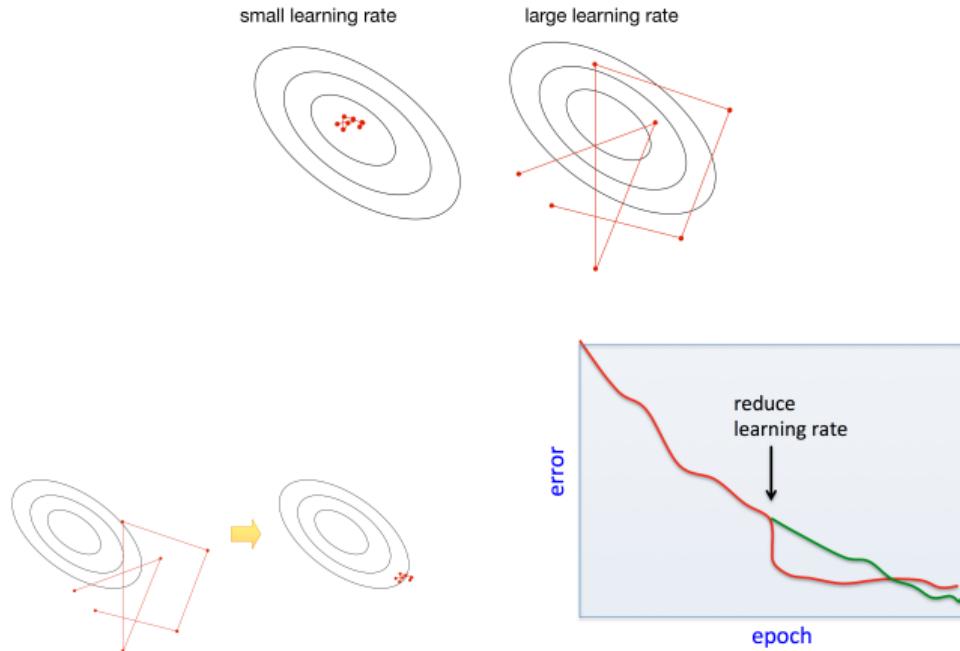
<sup>4</sup>Krizhevsky et al. ImageNet Classification with Deep Convolutional Neural Networks. NIPS, 2012.

## SGD Learning Rate

- In stochastic training, the learning rate also influences the **fluctuations** due to the stochasticity of the gradients.
- Typical strategy:
  - Use a large learning rate early in training so you can get close to the optimum.
  - Gradually decay the learning rate to reduce the fluctuations.

# Learning Rate Decay

- We also need to be aware about the impact of learning rate due to the stochasticity.



## RMSprop and Adam

- Recall: SGD takes large steps in directions of high curvature and small steps in directions of low curvature.
- **RMSprop** is a variant of SGD which rescales each coordinate of the gradient to have norm 1 on average. It does this by keeping an exponential moving average  $s_j$  of the squared gradients.

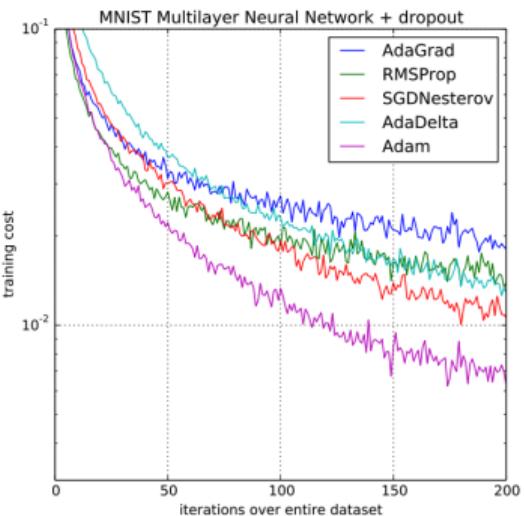
## RMSprop and Adam

- Recall: SGD takes large steps in directions of high curvature and small steps in directions of low curvature.
- **RMSprop** is a variant of SGD which rescales each coordinate of the gradient to have norm 1 on average. It does this by keeping an exponential moving average  $s_j$  of the squared gradients.
- The following update is applied to each coordinate  $j$  independently:

$$\begin{aligned}s_j &\leftarrow (1 - \gamma)s_j + \gamma[\frac{\partial L}{\partial \theta_j}]^2 \\ \theta_j &\leftarrow \theta_j - \frac{\alpha}{\sqrt{s_j + \epsilon}} \frac{\partial L}{\partial \theta_j}\end{aligned}$$

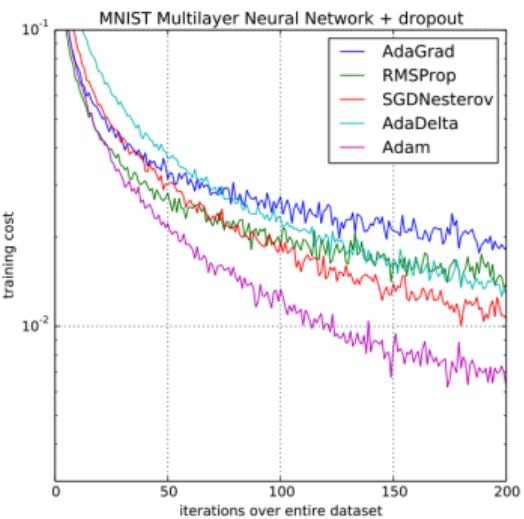
# Adam optimizer

- Adam = RMSprop + momentum = Adaptive Momentum estimation
- Smoother estimate of the average gradient and gradient norm.



# Adam optimizer

- Adam = RMSprop + momentum = Adaptive Momentum estimation
- Smoother estimate of the average gradient and gradient norm.
- $m_t$ : exponential moving average of gradient.
- $v_t$ : exponential moving average of gradient squared.
- $\hat{m}_t, \hat{v}_t$ : Bias correction.
- $\theta_t \leftarrow \theta_{t-1} - \alpha \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$
- The “default” optimizer for modern networks.



# Normalization

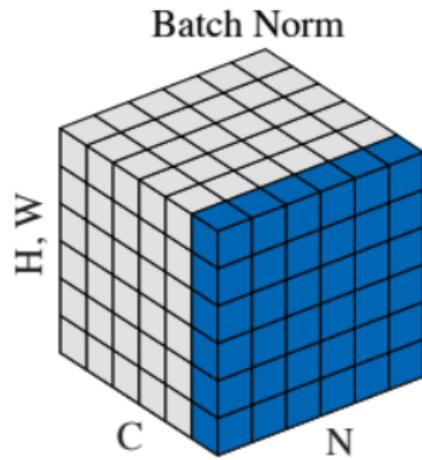
- Weight initialization is tricky, and there is no guarantee that the distribution of activations will stay the same over the learning process.
- What if the weights keep grow bigger and activation may explode?

# Normalization

- Weight initialization is tricky, and there is no guarantee that the distribution of activations will stay the same over the learning process.
- What if the weights keep grow bigger and activation may explode?
- We can “normalize” the activations.
- The idea is to control the activation within a normal range: zero-mean, uni-variance.

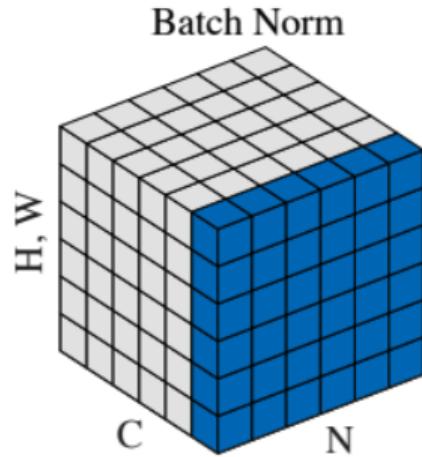
# Batch Normalization (BN)

- In CNNs, neurons across different spatial locations are also samples of the same feature channel.



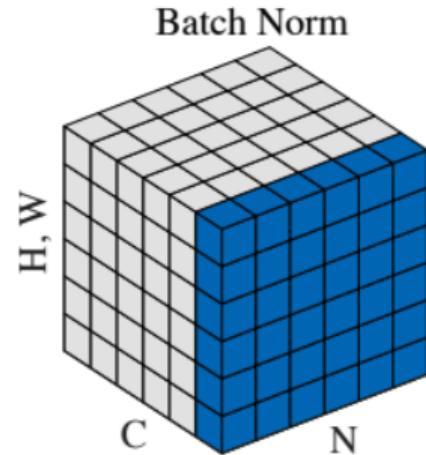
# Batch Normalization (BN)

- In CNNs, neurons across different spatial locations are also samples of the same feature channel.
- Batch norm: Normalize across  $N \ H \ W$  dimensions, leaving  $C$  channels.



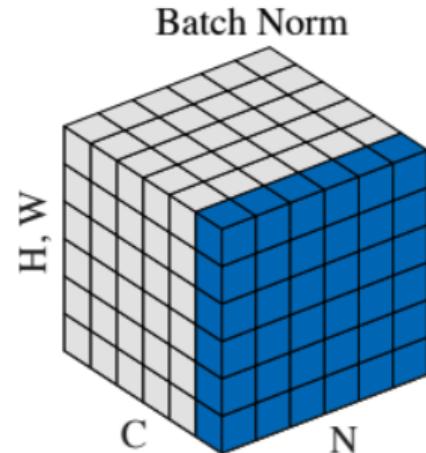
# Batch Normalization (BN)

- In CNNs, neurons across different spatial locations are also samples of the same feature channel.
- Batch norm: Normalize across N H W dimensions, leaving C channels.
- $\tilde{x} = \gamma \frac{x - \mu}{\sigma} + \beta$
- $\gamma, \beta$ : learnable parameters.  $\mu, \sigma$ : statistics from the training batch.



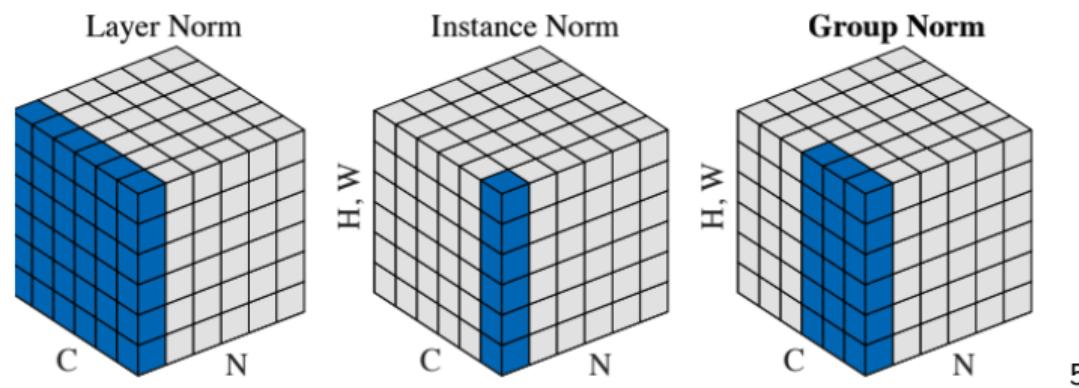
# Batch Normalization (BN)

- In CNNs, neurons across different spatial locations are also samples of the same feature channel.
- Batch norm: Normalize across N H W dimensions, leaving C channels.
- $\tilde{x} = \gamma \frac{x - \mu}{\sigma} + \beta$
- $\gamma, \beta$ : learnable parameters.  $\mu, \sigma$ : statistics from the training batch.
- Test time: using the mean and variance from the entire training set.



## BN Alternatives

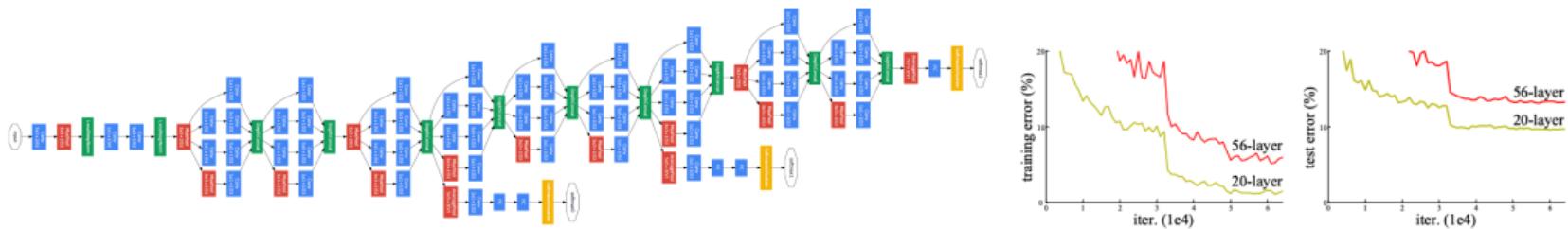
- Need a considerable batch size to estimate mean and variance correctly.
- Training is different from testing.
- Alternatives consider the C channel dimension instead of N batch dimension.



<sup>5</sup>Wu and He. Group normalization. ECCV 2018.

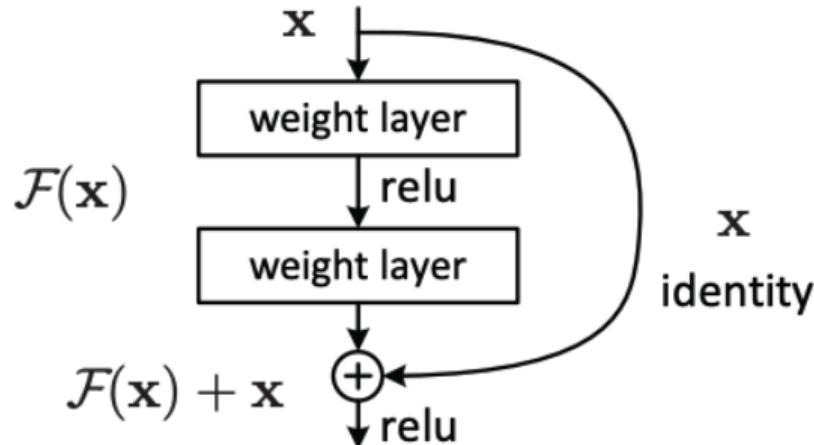
# Going Deeper

- The progress of normalization allowed us to train even deeper networks.
- The networks are no longer too sensitive with initialization.
- But the best networks were still around 20 layers and deeper results in worse performance.



# Residual Networks (ResNet)

- Recall in gradient boosting, we are iteratively adding a function to the model to expand the capacity.
- Residual connection: Skip connection to prevent gradient vanishing.<sup>6</sup>



<sup>6</sup>He et al. Deep Residual Learning for Image Recognition. CVPR 2016.

## ResNet Success

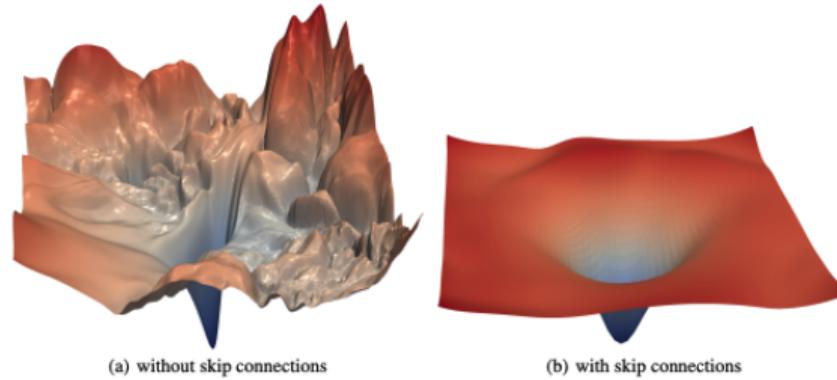
- Now able to train over 100 layers.
- One of the most important network design choices in the past decade.
- Prevalent in almost all network architectures, including Transformers.

---

<sup>7</sup>Li et al. Visualizing the Loss Landscape of Neural Nets. NIPS 2018.

# ResNet Success

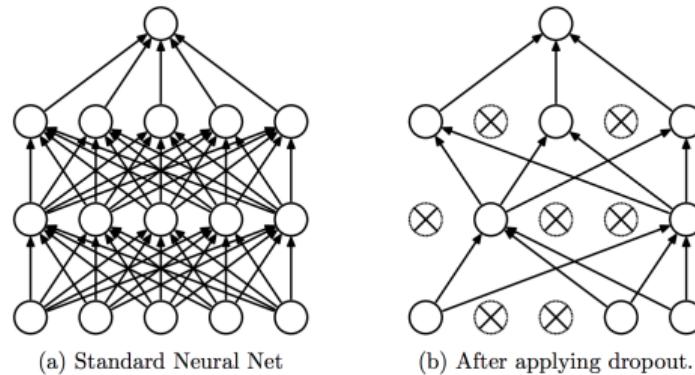
- Now able to train over 100 layers.
- One of the most important network design choices in the past decade.
- Prevalent in almost all network architectures, including Transformers.
- Loss landscape view: Skip connections makes loss smoother -> easier to optimize <sup>7</sup>.



<sup>7</sup>Li et al. Visualizing the Loss Landscape of Neural Nets. NIPS 2018.

# Dropout<sup>8</sup>

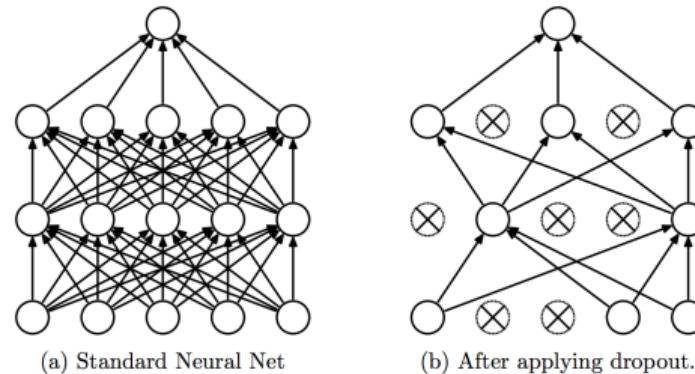
- Want to reduce overfitting in neural networks.
- Stochastically turning off neurons in propagation.



<sup>8</sup>Srivastava et al. A Simple Way to Prevent Neural Networks from Overfitting. JMLR, 2014.

# Dropout<sup>8</sup>

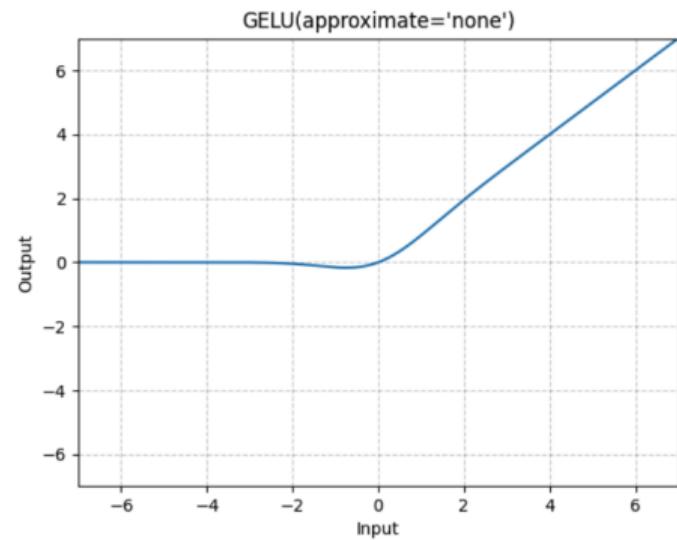
- Want to reduce overfitting in neural networks.
- Stochastically turning off neurons in propagation.
- Training to preserve redundancy.
- Test time: multiplying activations with probability. Model ensembling effect.



<sup>8</sup>Srivastava et al. A Simple Way to Prevent Neural Networks from Overfitting. JMLR, 2014.

# GELU<sup>9</sup>

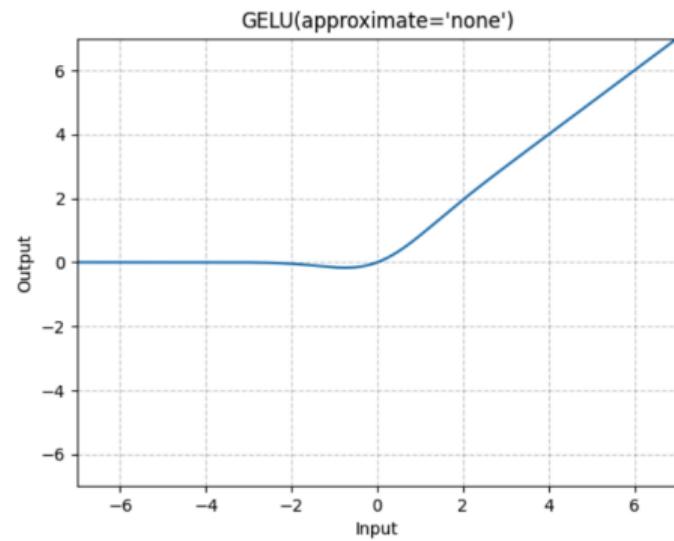
- Gaussian Error Linear Unit - A smoother activation function.
- Motivated by Dropout.



<sup>9</sup>Hendrycks & Gimpel. Gaussian Error Linear Unit (GELU). CoRR abs/1606.08415, 2016.

# GELU<sup>9</sup>

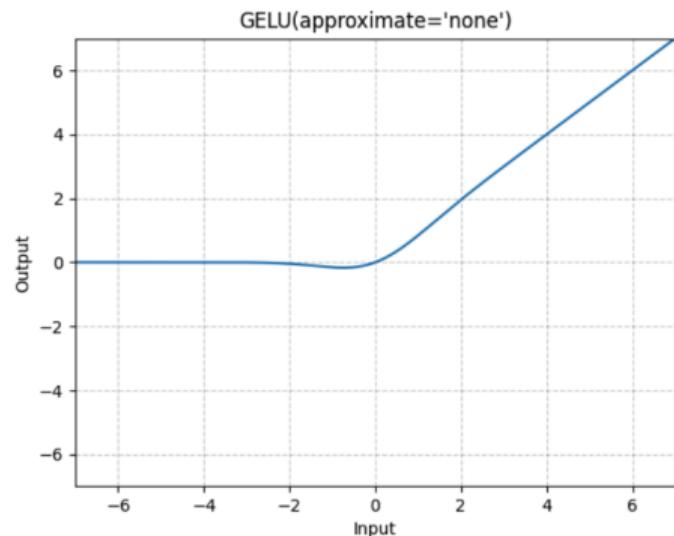
- Gaussian Error Linear Unit - A smoother activation function.
- Motivated by Dropout.
- $f(x) = \mathbb{E}[x \cdot m]$ .



<sup>9</sup>Hendrycks & Gimpel. Gaussian Error Linear Unit (GELU). CoRR abs/1606.08415, 2016.

# GELU<sup>9</sup>

- Gaussian Error Linear Unit - A smoother activation function.
- Motivated by Dropout.
- $f(x) = \mathbb{E}[x \cdot m]$ .
- $m \sim \text{Bernoulli}(\Phi(x))$ .
- $\Phi(x) = P(X \leq x)$ .
- $X \sim \mathcal{N}(0, 1)$ .



<sup>9</sup>Hendrycks & Gimpel. Gaussian Error Linear Unit (GELU). CoRR abs/1606.08415, 2016.

# Data augmentation

- Leverage the invariances of images
- Create more data points for free

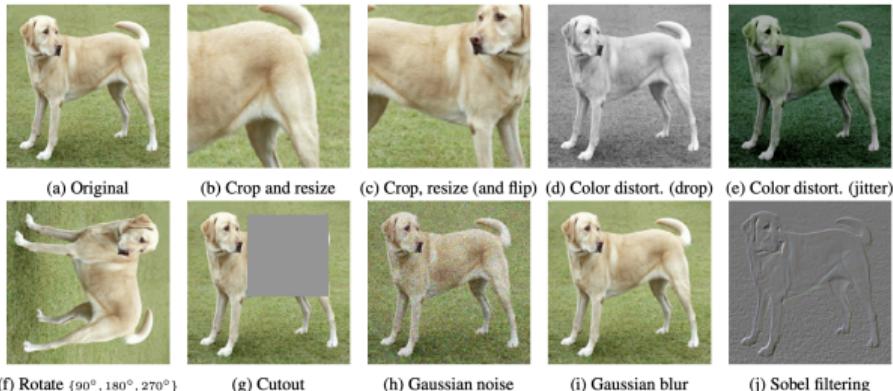


Image credit<sup>10</sup>

<sup>10</sup>Chen et al. A Simple Framework for Contrastive Learning of Visual Representations. ICML 2020.

# Data augmentation

- Leverage the invariances of images
- Create more data points for free
  - Random cropping

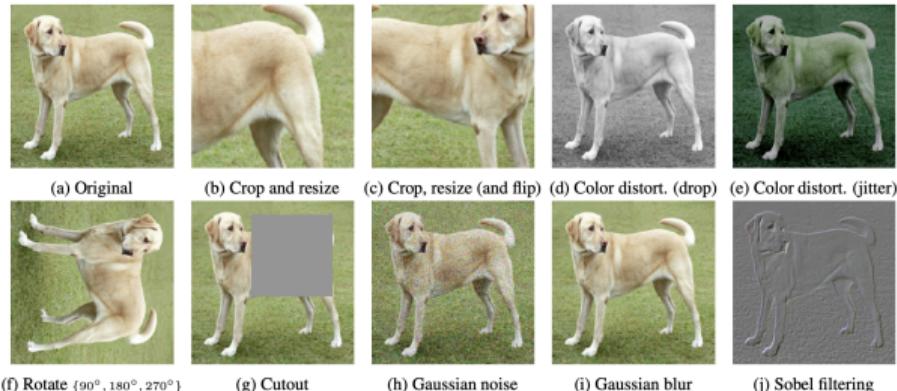


Image credit<sup>10</sup>

<sup>10</sup>Chen et al. A Simple Framework for Contrastive Learning of Visual Representations. ICML 2020.

# Data augmentation

- Leverage the invariances of images
- Create more data points for free
  - Random cropping
  - Left+right flipping

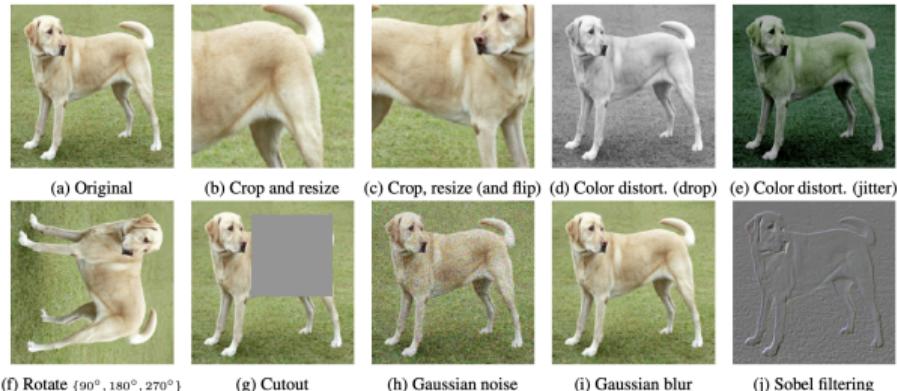


Image credit<sup>10</sup>

<sup>10</sup>Chen et al. A Simple Framework for Contrastive Learning of Visual Representations. ICML 2020.

# Data augmentation

- Leverage the invariances of images
- Create more data points for free
  - Random cropping
  - Left+right flipping
  - Random color jittering

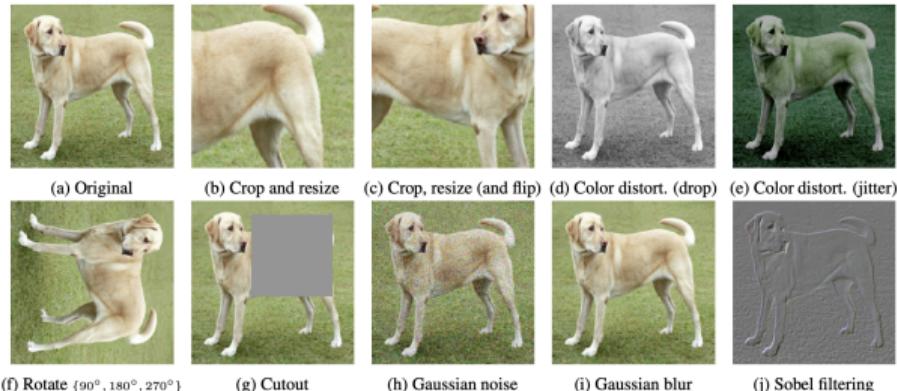


Image credit<sup>10</sup>

<sup>10</sup>Chen et al. A Simple Framework for Contrastive Learning of Visual Representations. ICML 2020.

# Data augmentation

- Leverage the invariances of images
- Create more data points for free
  - Random cropping
  - Left+right flipping
  - Random color jittering
  - Random blurring

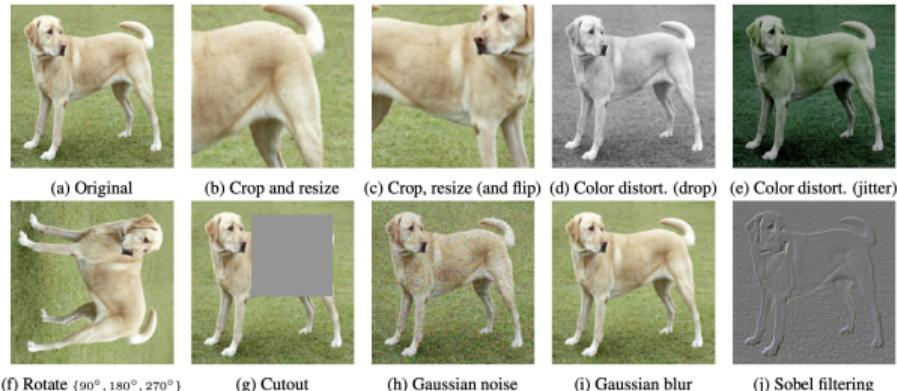


Image credit<sup>10</sup>

<sup>10</sup>Chen et al. A Simple Framework for Contrastive Learning of Visual Representations. ICML 2020.

# Data augmentation

- Leverage the invariances of images
- Create more data points for free
  - Random cropping
  - Left+right flipping
  - Random color jittering
  - Random blurring
  - Affine warping
  - Etc.

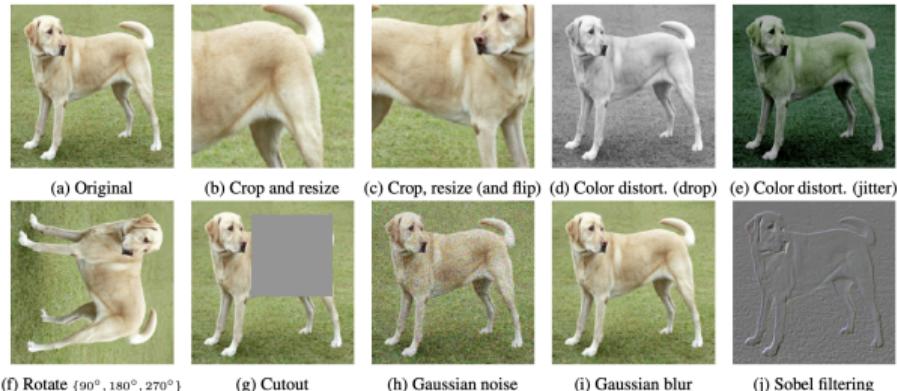


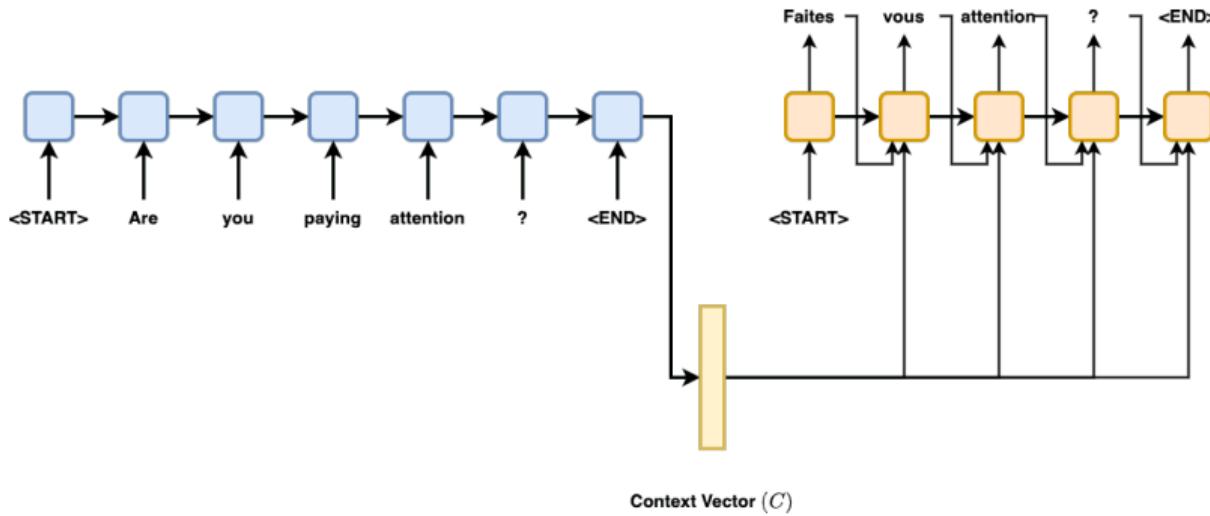
Image credit<sup>10</sup>

<sup>10</sup>Chen et al. A Simple Framework for Contrastive Learning of Visual Representations. ICML 2020.

## Language and sequential signals

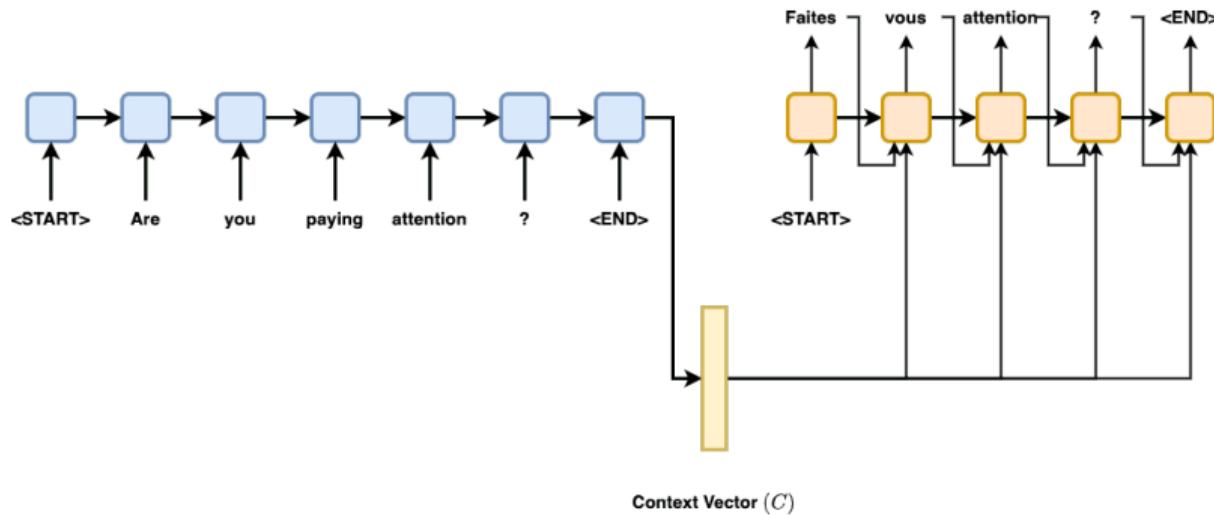
# What about natural language

- Neural networks are great for dealing with naturalistic and unstructured signals.



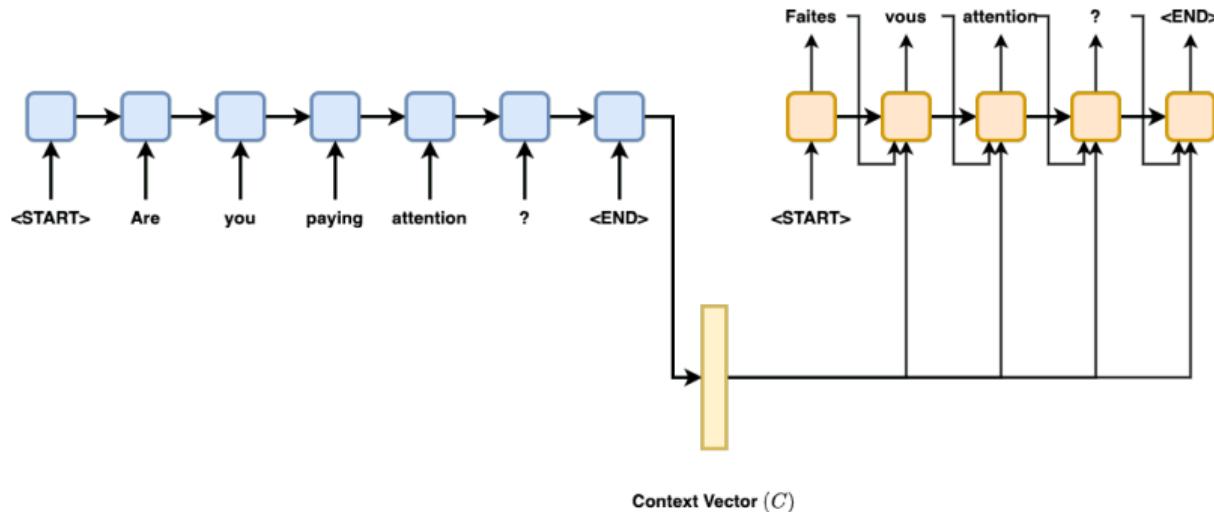
# What about natural language

- Neural networks are great for dealing with naturalistic and unstructured signals.
- Past lectures: Feature functions in structured models, but still primitive.



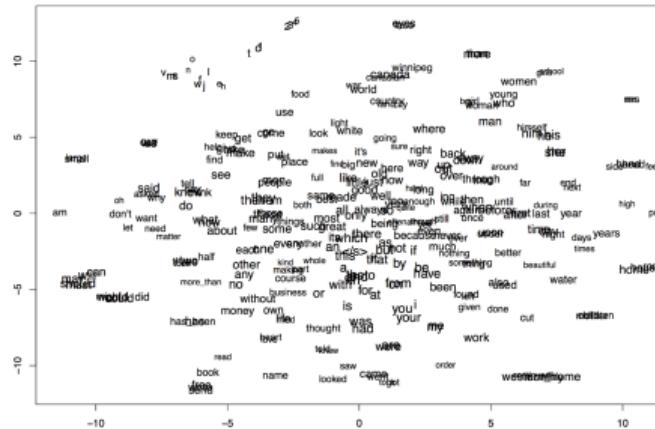
# What about natural language

- Neural networks are great for dealing with naturalistic and unstructured signals.
- Past lectures: Feature functions in structured models, but still primitive.
- Design neural networks to accommodate sequential signals such as language.



# Word embeddings

- Neural networks are best dealing with real valued vectors.

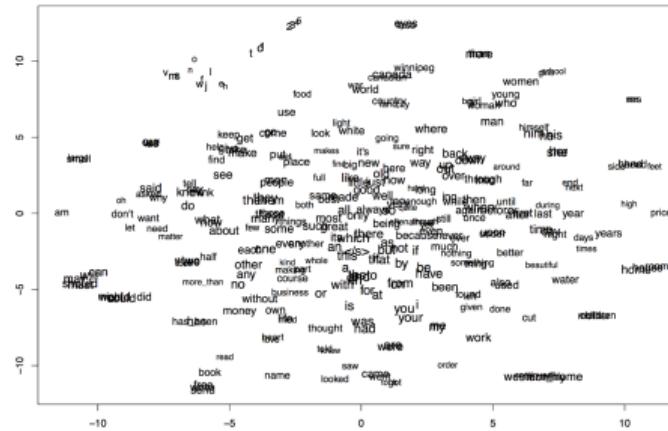


11

<sup>11</sup><https://aelang.github.io/word-embeddings.html>

# Word embeddings

- Neural networks are best dealing with real valued vectors.
- Need to convert words (discrete) into vectors (continuous).

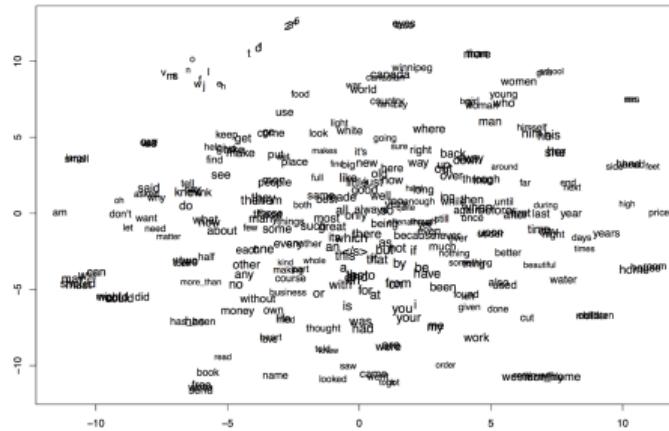


11

<sup>11</sup><https://aelang.github.io/word-embeddings.html>

# Word embeddings

- Neural networks are best dealing with real valued vectors.
- Need to convert words (discrete) into vectors (continuous).
- A large matrix of  $V \times D$ .  $V$  = vocab size,  $D$  = network embedding size.

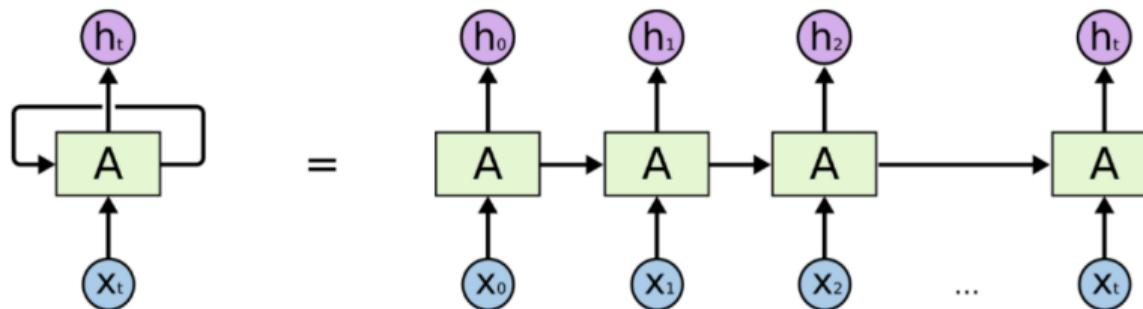


11

<sup>11</sup><https://aelang.github.io/word-embeddings.html>

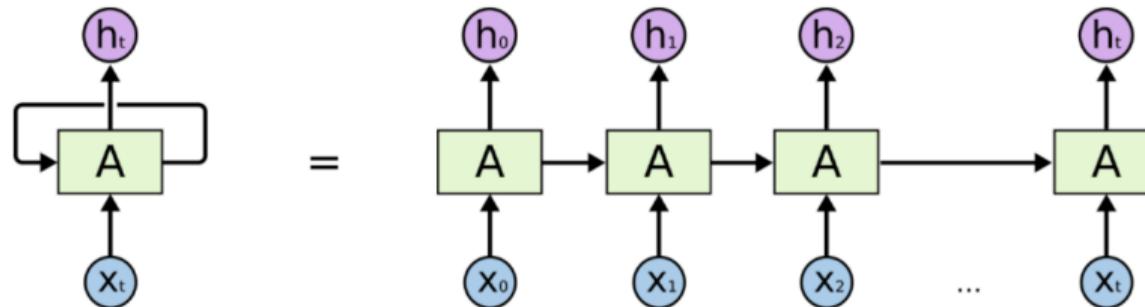
# Convolutional vs. recurrent networks

- Recall in images we used the convolution operation.
- We can also use the idea of convolution for temporal signals.



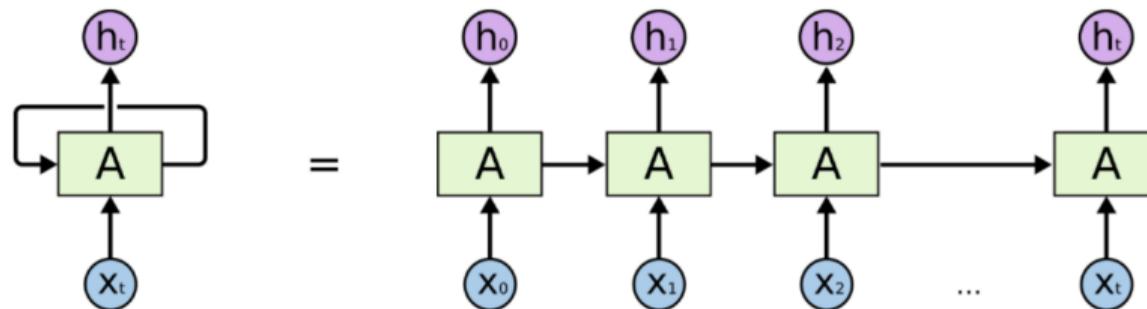
## Convolutional vs. recurrent networks

- Recall in images we used the convolution operation.
- We can also use the idea of convolution for temporal signals.
- Another alternative is to use a type of network called recurrent networks.
- Two inputs:  $x_t$  is the current input, and  $h_t$  is the historical hidden state.



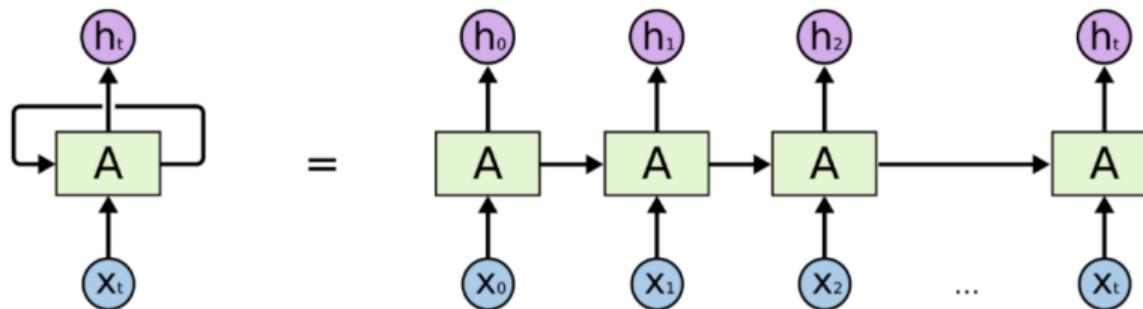
## Convolutional vs. recurrent networks

- Recall in images we used the convolution operation.
- We can also use the idea of convolution for temporal signals.
- Another alternative is to use a type of network called recurrent networks.
- Two inputs:  $x_t$  is the current input, and  $h_t$  is the historical hidden state.
- We can unroll the computation graph into a direct acyclic graph (DAG).



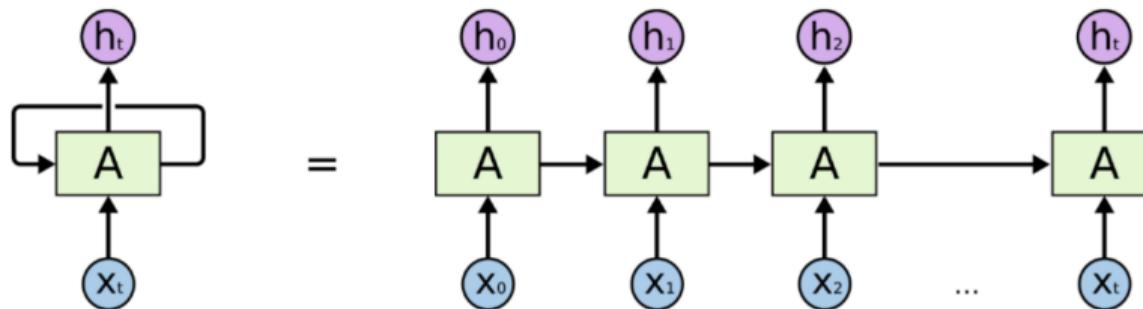
# Recurrent neural networks (RNNs)

- A simple RNN can be made similar to a standard NN with one hidden layer.



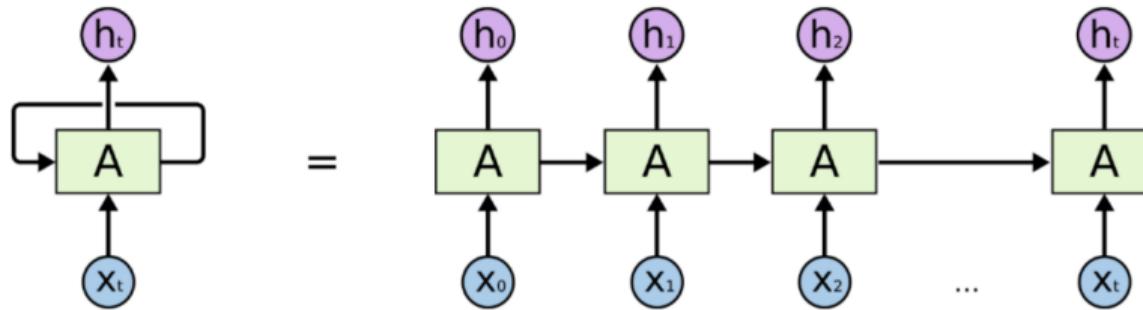
# Recurrent neural networks (RNNs)

- A simple RNN can be made similar to a standard NN with one hidden layer.
- $h_t = \tanh(Wh_{t-1} + Ux_t)$ .



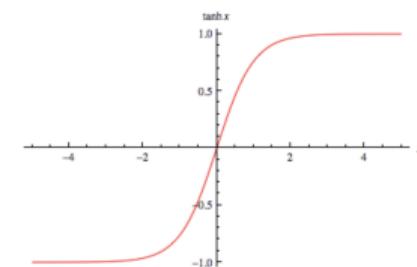
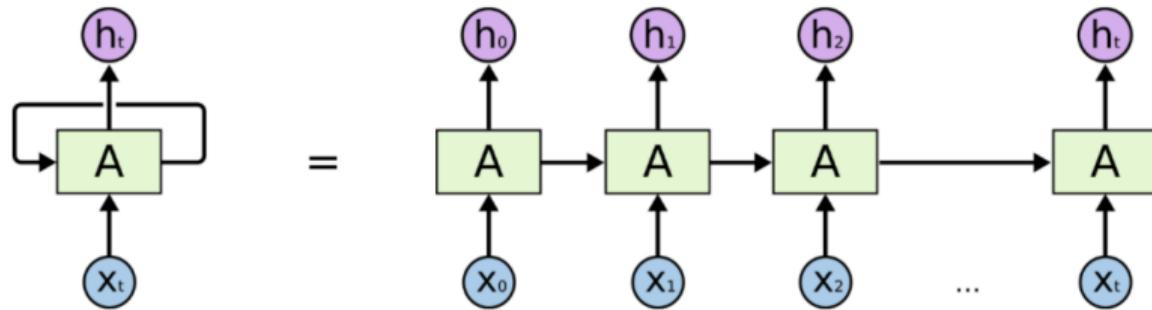
# Recurrent neural networks (RNNs)

- A simple RNN can be made similar to a standard NN with one hidden layer.
- $h_t = \tanh(Wh_{t-1} + Ux_t)$ .
- $y_t = \text{Softmax}(Vh_t)$ .



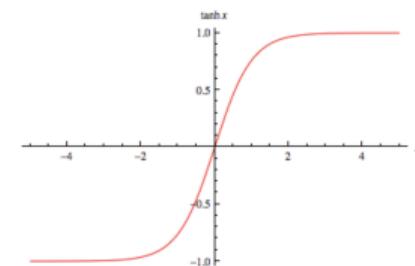
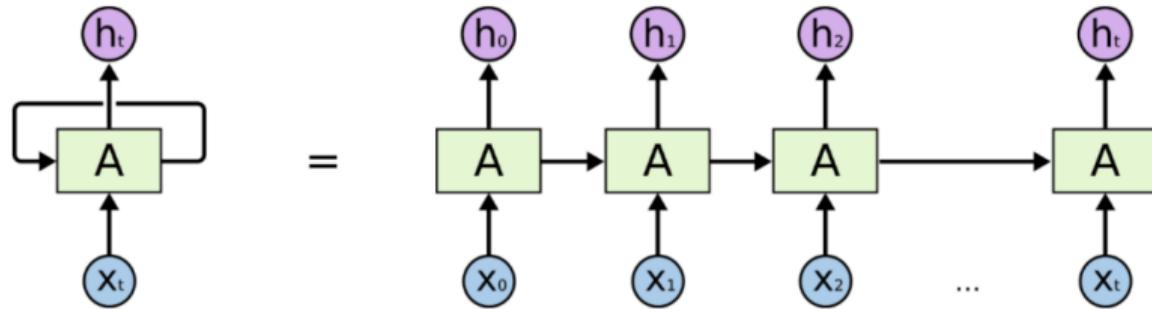
# Gradient vanishing

- Every iteration, we multiply the hidden state  $h_{t-1}$  from the previous iteration with the same  $W$ . Recall the definition of Jacobian.



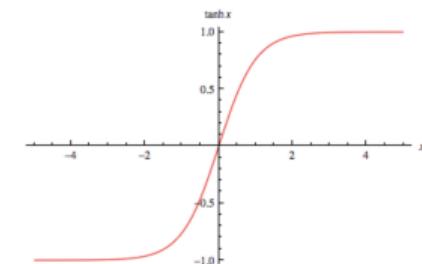
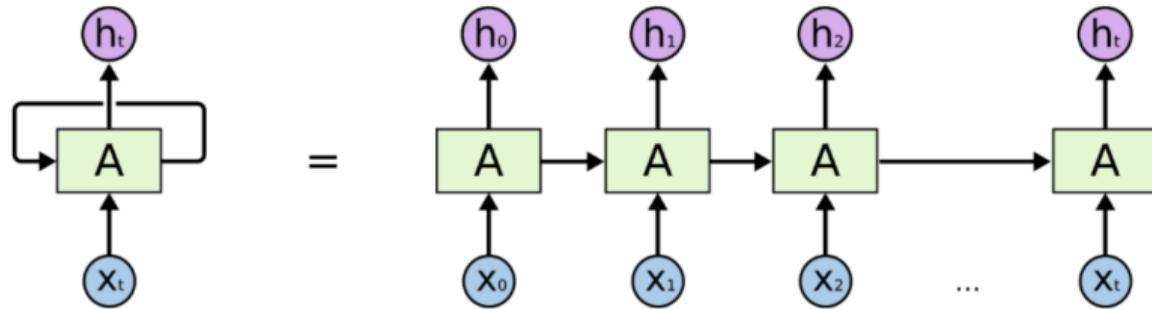
# Gradient vanishing

- Every iteration, we multiply the hidden state  $h_{t-1}$  from the previous iteration with the same  $W$ . Recall the definition of Jacobian.
- If the largest singular value of  $W$  is less than one then back-propagation will be attenuated.



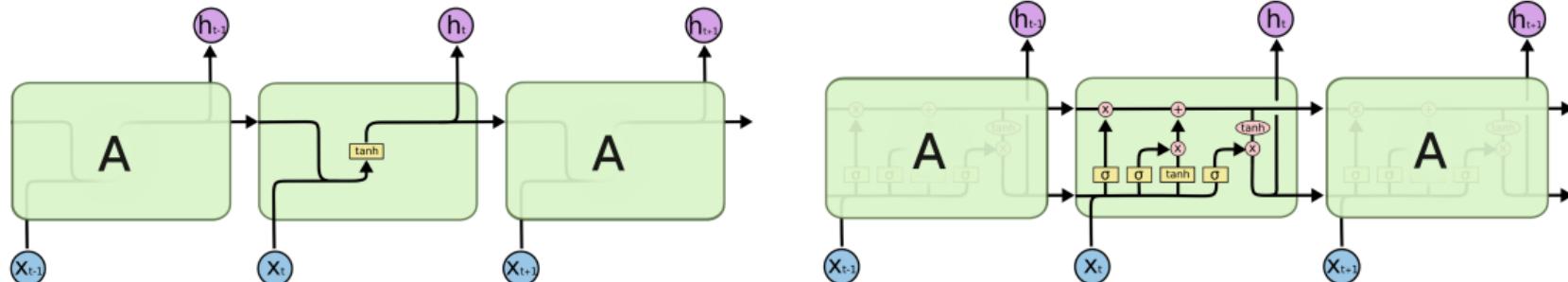
# Gradient vanishing

- Every iteration, we multiply the hidden state  $h_{t-1}$  from the previous iteration with the same  $W$ . Recall the definition of Jacobian.
- If the largest singular value of  $W$  is less than one then back-propagation will be attenuated.
- Similarly, we apply tanh activation every iteration – further reducing gradient flow.



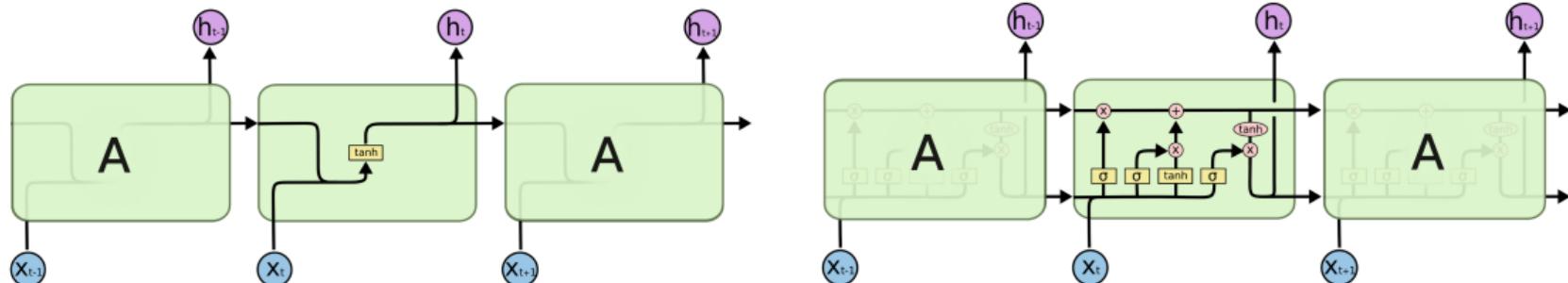
# Gating functions in LSTM

- Long short-term memory is a network that addresses the gradient vanishing problem by introducing gating functions.
- Gating functions provide “shortcuts”, like ResNet.



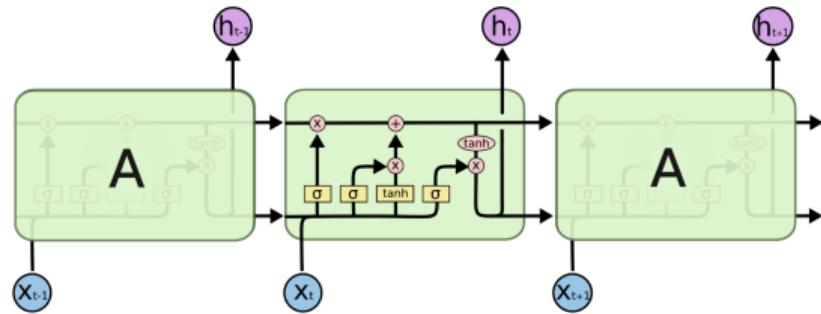
# Gating functions in LSTM

- Long short-term memory is a network that addresses the gradient vanishing problem by introducing gating functions.
- Gating functions provide “shortcuts”, like ResNet.
- Originally proposed by Hochreiter and Schmidhuber in 1997.



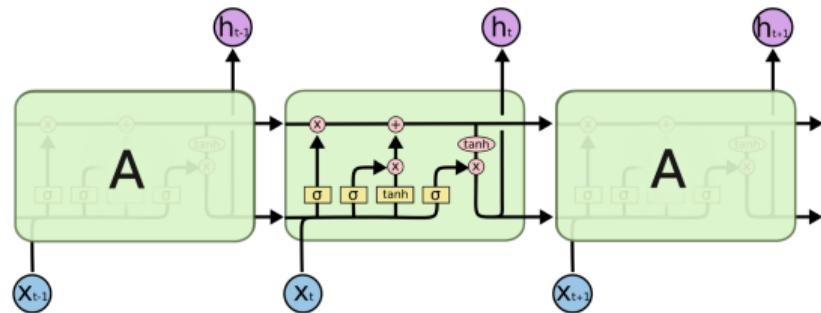
# Gating functions in LSTM

- Input gate:  $i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$ .
- Forget gate:  $f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$ .
- $z_t = \tanh(w_z[h_{t-1}x_t] + b_z)$ .



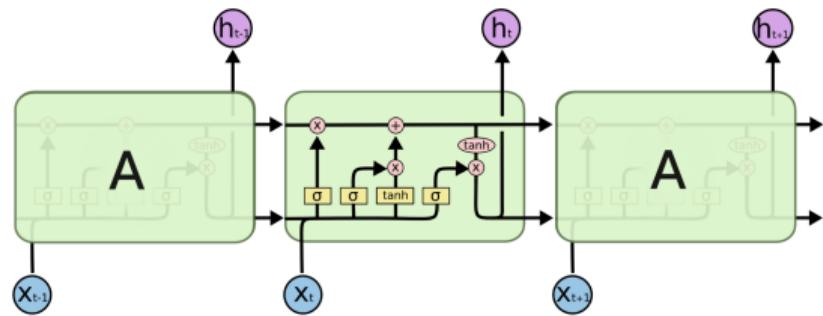
# Gating functions in LSTM

- Input gate:  $i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$ .
- Forget gate:  $f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$ .
- $z_t = \tanh(w_z[h_{t-1}x_t] + b_z)$ .
- $c_t = f_t \odot c_{t-1} + i_t \odot z_t$ .



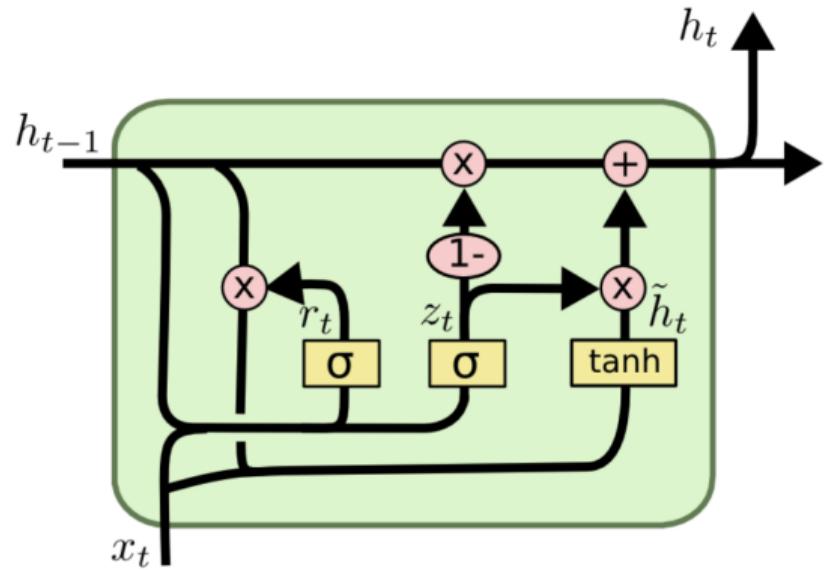
# Gating functions in LSTM

- Input gate:  $i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$ .
- Forget gate:  $f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$ .
- $z_t = \tanh(w_z[h_{t-1}x_t] + b_z)$ .
- $c_t = f_t \odot c_{t-1} + i_t \odot z_t$ .
- Output gate:  $o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$ .
- $h_t = o_t \odot \tanh(c_t)$ .



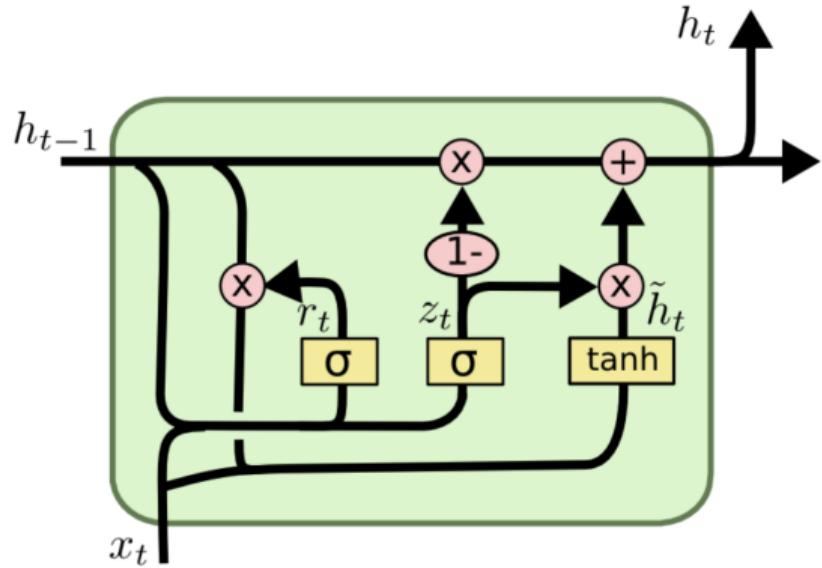
# Gated Recurrent Unit

- Proposed by Chung et al. in 2015, a simplified variant compared to LSTM.



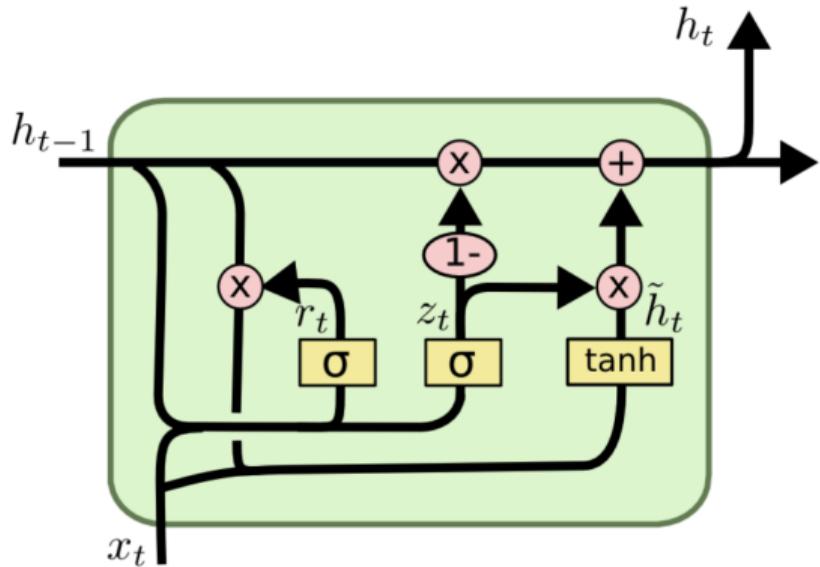
# Gated Recurrent Unit

- Proposed by Chung et al. in 2015, a simplified variant compared to LSTM.
- Input gate  $i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$ .



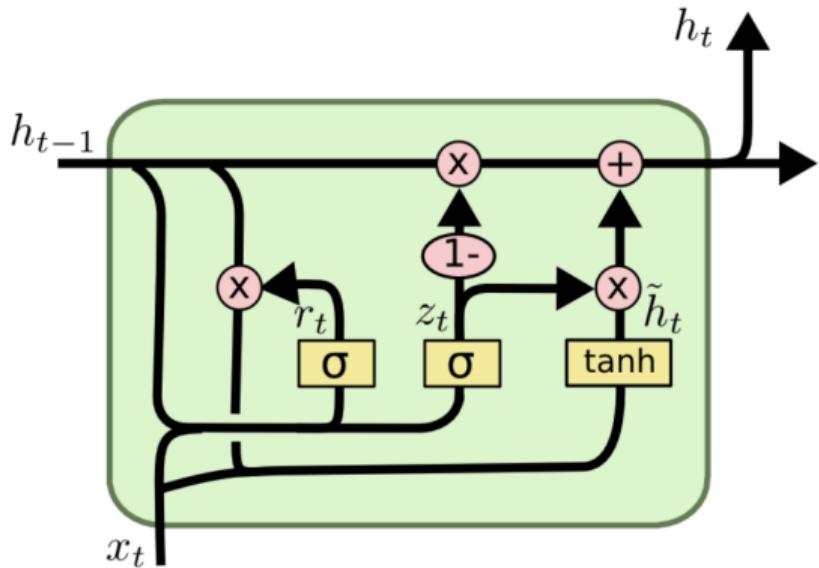
# Gated Recurrent Unit

- Proposed by Chung et al. in 2015, a simplified variant compared to LSTM.
- Input gate  $i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$ .
- Reset gate  $r_t = \sigma(W_r[h_{t-1}, x_t] + b_r)$ .



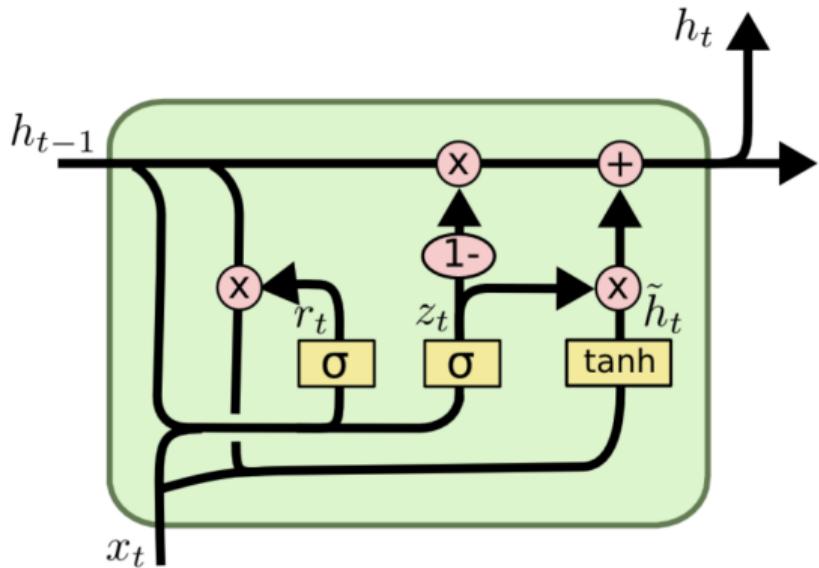
# Gated Recurrent Unit

- Proposed by Chung et al. in 2015, a simplified variant compared to LSTM.
- Input gate  $i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$ .
- Reset gate  $r_t = \sigma(W_r[h_{t-1}, x_t] + b_r)$ .
- $\tilde{h}_t = \tanh(W_h[r_t \odot h_{t-1}, x_t] + b_h)$ .



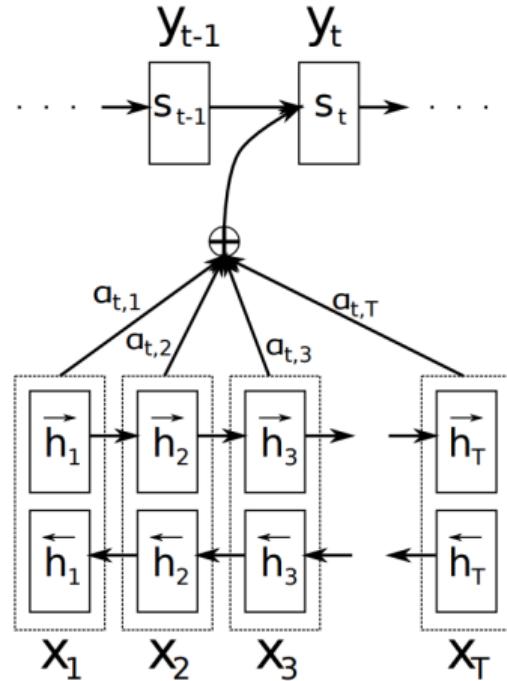
# Gated Recurrent Unit

- Proposed by Chung et al. in 2015, a simplified variant compared to LSTM.
- Input gate  $i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$ .
- Reset gate  $r_t = \sigma(W_r[h_{t-1}, x_t] + b_r)$ .
- $\tilde{h}_t = \tanh(W_h[r_t \odot h_{t-1}, x_t] + b_h)$ .
- $h_t = (1 - i_t) \odot h_{t-1} + i_t \odot \tilde{h}_t$ .



# Attention Mechanisms

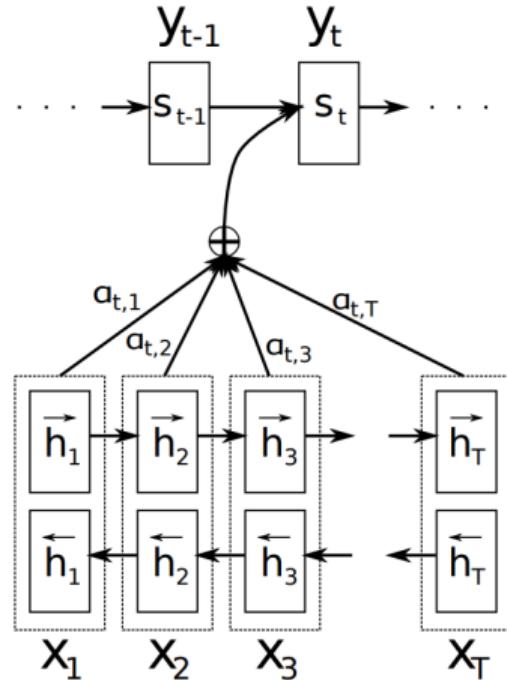
- Earlier content will decay more.
- Hard to refer back to the raw content.
- Reverse order better than forward order  
[abcde -> a'b'c'd'e' vs. abcde -> e'd'c'b'a'].



Bahdanau et al., 2014

# Attention Mechanisms

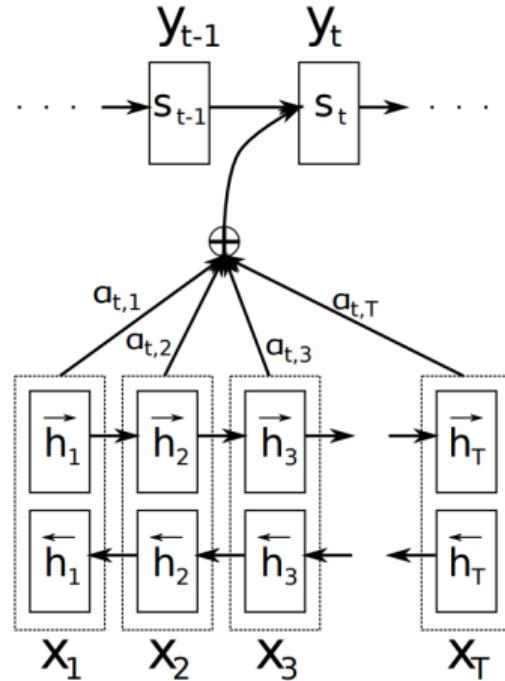
- Earlier content will decay more.
- Hard to refer back to the raw content.
- Reverse order better than forward order  
[abcde -> a'b'c'd'e' vs. abcde -> e'd'c'b'a'].
- Attending to arbitrary sequence tokens.



Bahdanau et al., 2014

# Attention Mechanisms

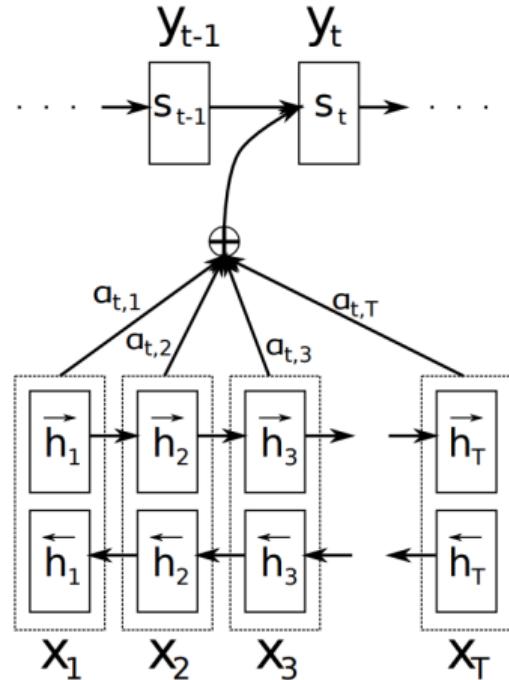
- Earlier content will decay more.
- Hard to refer back to the raw content.
- Reverse order better than forward order  
[abcde -> a'b'c'd'e' vs. abcde -> e'd'c'b'a'].
- Attending to arbitrary sequence tokens.
- $s_t = f(s_{t-1}, y_{t-1}, c_t)$



Bahdanau et al., 2014

# Attention Mechanisms

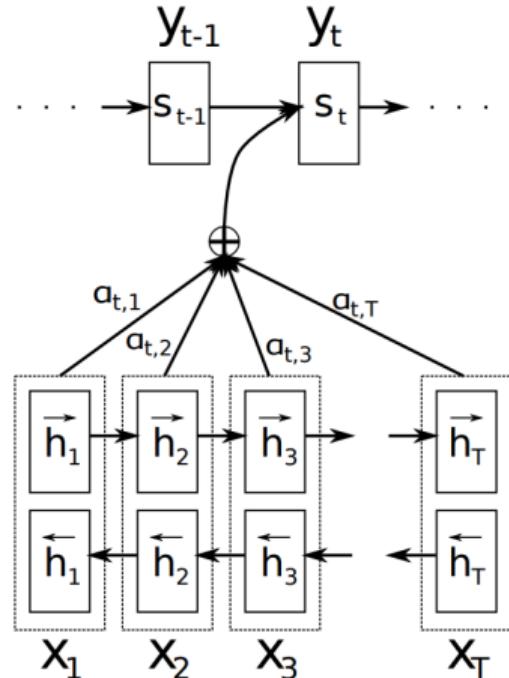
- Earlier content will decay more.
- Hard to refer back to the raw content.
- Reverse order better than forward order [abcde -> a'b'c'd'e' vs. abcde -> e'd'c'b'a'].
- Attending to arbitrary sequence tokens.
- $s_t = f(s_{t-1}, y_{t-1}, c_t)$
- $c_t = \sum_{\tau} \alpha_{t,\tau} h_{\tau}, \alpha_{t,\tau} = \frac{\exp(a(s_{t-1}, h_{\tau}))}{\sum_k \exp(a(s_{t-1}, h_k))}$



Bahdanau et al., 2014

# Attention Mechanisms

- Earlier content will decay more.
- Hard to refer back to the raw content.
- Reverse order better than forward order  
[abcde -> a'b'c'd'e' vs. abcde -> e'd'c'b'a'].
- Attending to arbitrary sequence tokens.
- $s_t = f(s_{t-1}, y_{t-1}, c_t)$
- $c_t = \sum_{\tau} \alpha_{t,\tau} h_{\tau}, \alpha_{t,\tau} = \frac{\exp(a(s_{t-1}, h_k))}{\sum_k \exp(a(s_{t-1}, h_k))}$
- $a(s_{t-1}, h_k) = v_a^T \tanh(W_a[s_{t-1}, h_k])$



Bahdanau et al., 2014

# Transformers ("Attention is All You Need")

- The previous architecture is very complicated.
  - 1 RNN for encoding the tokens.
  - Attention mechanisms for accessing content
  - 1 RNN for combining attended tokens.



13

<sup>13</sup>Image credit: Google Research Blog

# Transformers ("Attention is All You Need")

- The previous architecture is very complicated.
  - 1 RNN for encoding the tokens.
  - Attention mechanisms for accessing content
  - 1 RNN for combining attended tokens.
- RNNs have the ability to incorporate past information, so does attention.



13

<sup>13</sup>Image credit: Google Research Blog

# Positional encoding

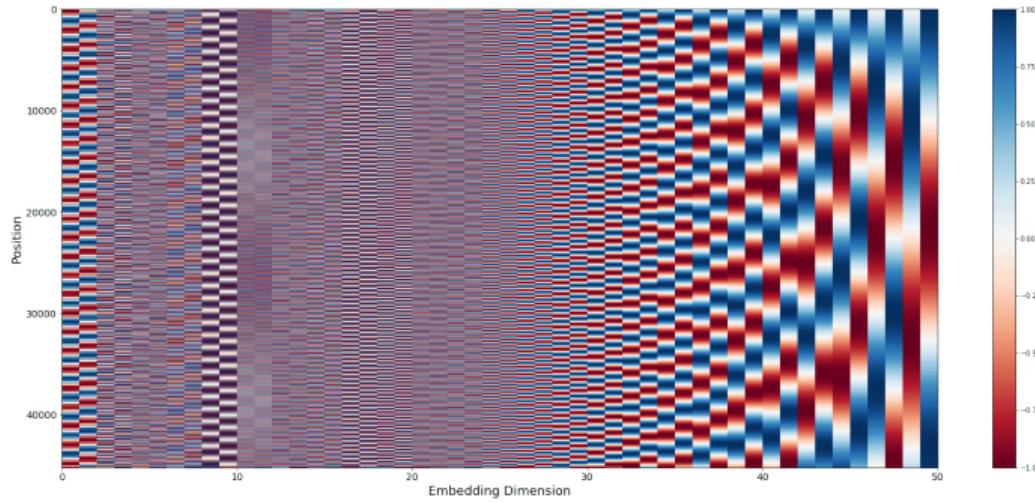
- Attention operation is permutation equivariant.

## Positional encoding

- Attention operation is permutation equivariant.
- Solution: Encode the position of each token.

# Positional encoding

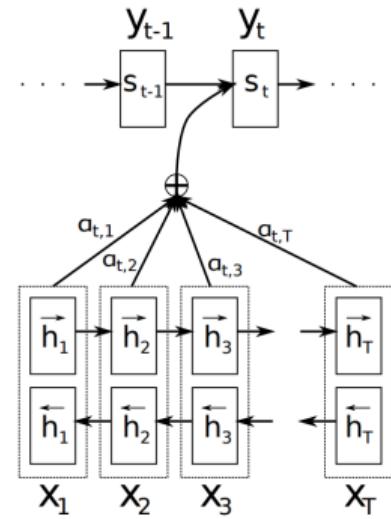
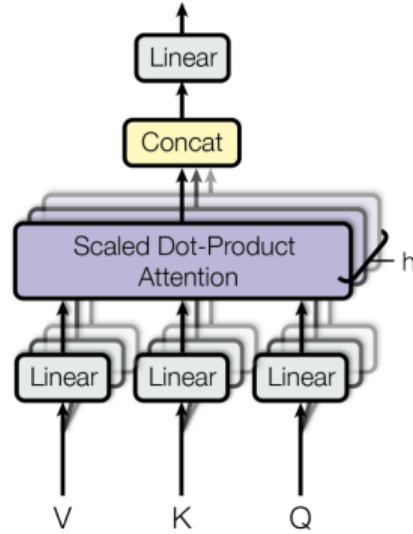
- Attention operation is permutation equivariant.
- Solution: Encode the position of each token.
- $PE(pos, 2i) = \sin(p/k^{2i/d})$ ,  $PE(pos, 2i+1) = \cos(p/k^{2i/d})$ .



# Multi-headed attention

- Map tokens into query, key, and value.

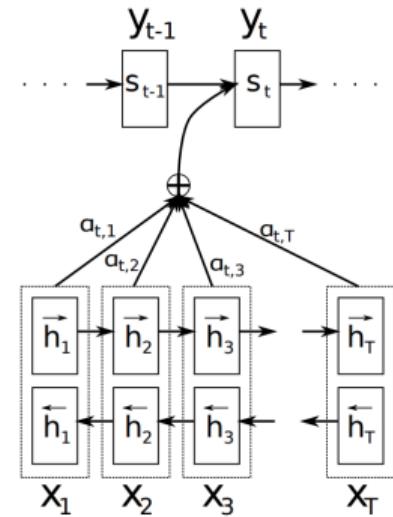
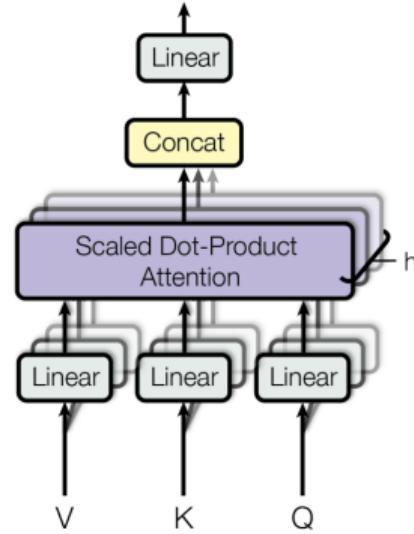
Multi-Head Attention



# Multi-headed attention

- Map tokens into query, key, and value.
- $\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$ .

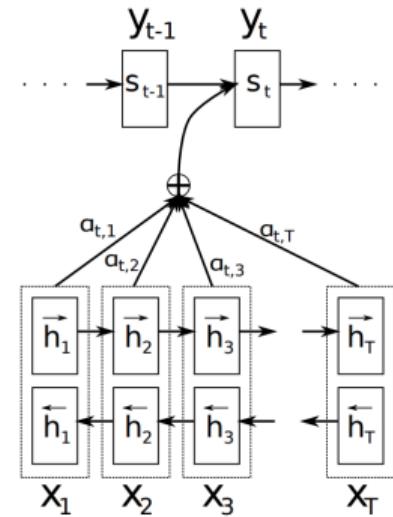
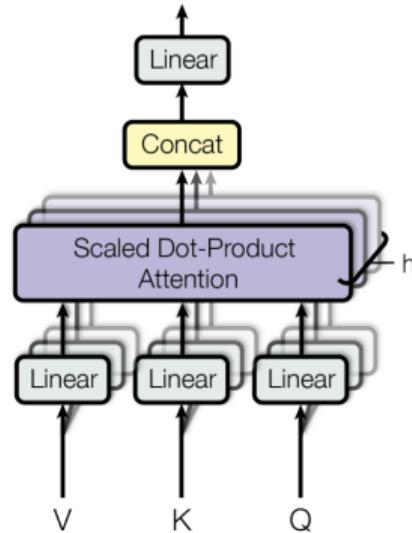
Multi-Head Attention



# Multi-headed attention

- Map tokens into query, key, and value.
- $\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$ .
- $H_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$ .

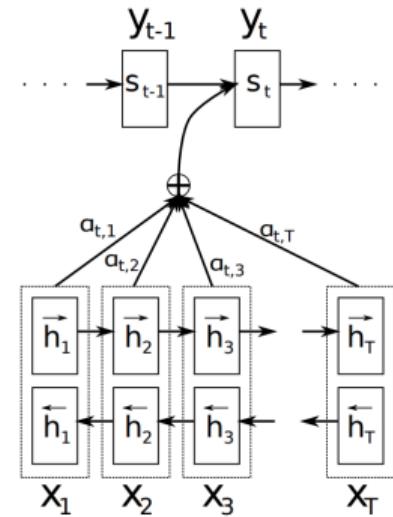
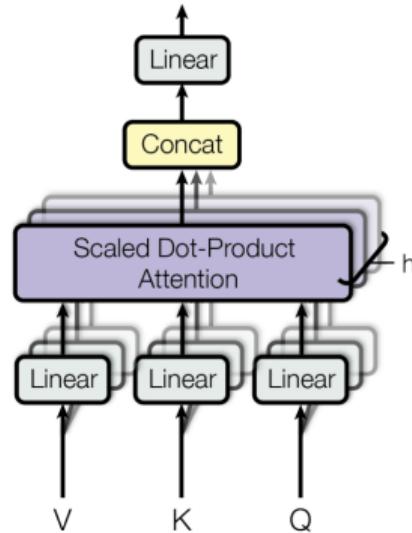
Multi-Head Attention



# Multi-headed attention

- Map tokens into query, key, and value.
- $\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$ .
- $H_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$ .
- $\text{MultiHead}(Q, K, V) = [H_1, \dots, H_n]W^O$

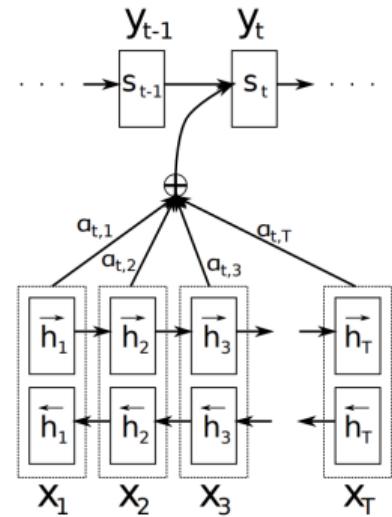
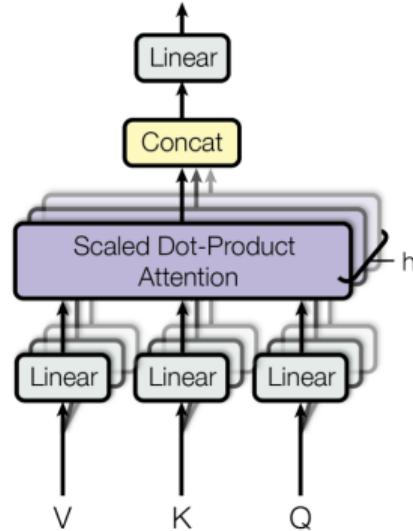
Multi-Head Attention



# Multi-headed attention

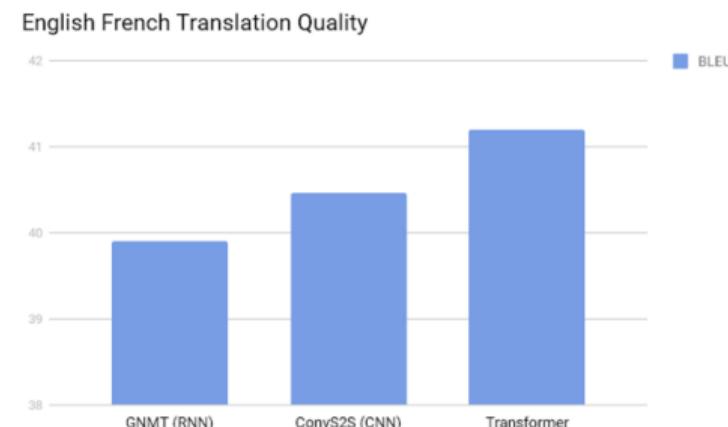
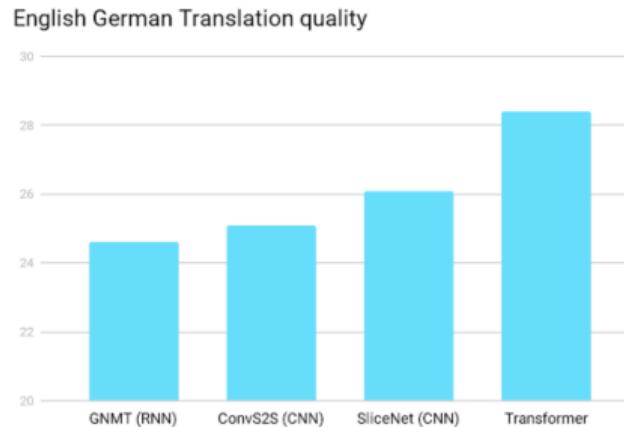
- Map tokens into query, key, and value.
- $\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$ .
- $H_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$ .
- $\text{MultiHead}(Q, K, V) = [H_1, \dots, H_n]W^O$
- More advantageous to have multiple set of attentions for each token, so it can more efficiently incorporate information from multiple sources.

Multi-Head Attention



# Machine Translation

- Achieved superior performance on machine translation.
- Animation [link](#)



# Autoregressive modeling

- Recall the chain rule on joint distribution:

$$p(x_{1:t}) = p(x_1, \dots, x_t) = p(x_1)p(x_2|x_1)\dots p(x_t|x_{t-1}) = p(x_1) \prod_i p(x_i|x_{1:i-1}).$$

## Autoregressive modeling

- Recall the chain rule on joint distribution:

$$p(x_{1:t}) = p(x_1, \dots, x_t) = p(x_1)p(x_2|x_1)\dots p(x_t|x_{t-1}) = p(x_1) \prod_i p(x_i|x_{1:i-1}).$$

- In Naive Bayes, we treat each variable as independent, but this cannot perform sequence generation.

## Autoregressive modeling

- Recall the chain rule on joint distribution:

$$p(x_{1:t}) = p(x_1, \dots, x_t) = p(x_1)p(x_2|x_1)\dots p(x_t|x_{t-1}) = p(x_1) \prod_i p(x_i|x_{1:i-1}).$$

- In Naive Bayes, we treat each variable as independent, but this cannot perform sequence generation.
- How do we model a conditional distribution  $p(x_i|x_{1:i-1})$  using an RNN or a Transformer?

# Autoregressive modeling

- Recall the chain rule on joint distribution:

$$p(x_{1:t}) = p(x_1, \dots, x_t) = p(x_1)p(x_2|x_1)\dots p(x_t|x_{t-1}) = p(x_1) \prod_i p(x_i|x_{1:i-1}).$$

- In Naive Bayes, we treat each variable as independent, but this cannot perform sequence generation.
- How do we model a conditional distribution  $p(x_i|x_{1:i-1})$  using an RNN or a Transformer?
- RNN is naturally autoregressive:  $h_t$  contains all information up to time t.

## Autoregressive modeling

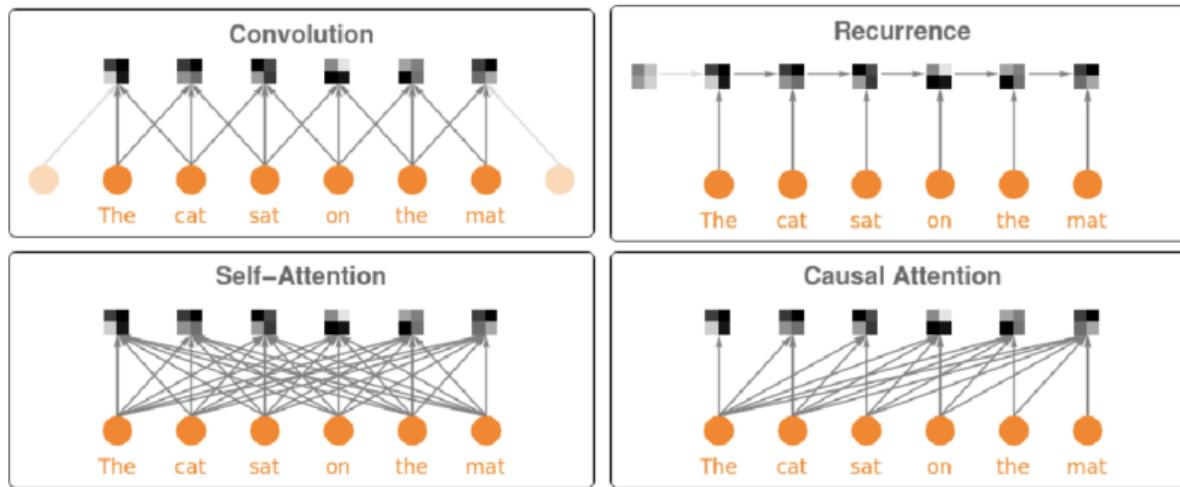
- Recall the chain rule on joint distribution:

$$p(x_{1:t}) = p(x_1, \dots, x_t) = p(x_1)p(x_2|x_1)\dots p(x_t|x_{t-1}) = p(x_1) \prod_i p(x_i|x_{1:i-1}).$$

- In Naive Bayes, we treat each variable as independent, but this cannot perform sequence generation.
- How do we model a conditional distribution  $p(x_i|x_{1:i-1})$  using an RNN or a Transformer?
- RNN is naturally autoregressive:  $h_t$  contains all information up to time t.
- For Transformers,  $h_t$  contains information about the future.

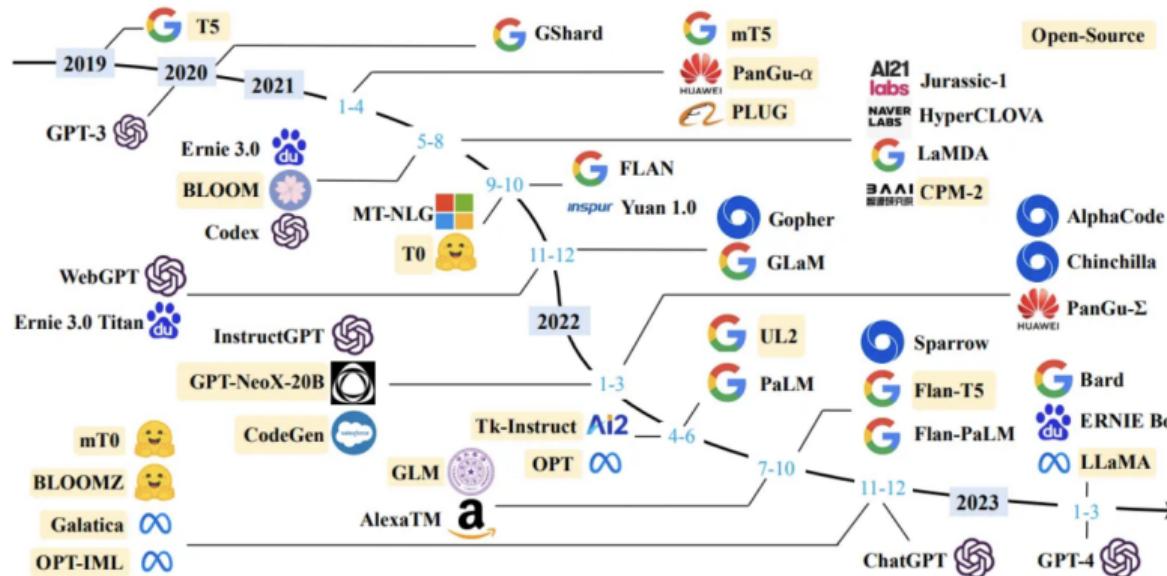
# Causal Attention

- For Transformers, we need to “mask” the attention so that each token can only attend to tokens prior to itself.
- This is called “causal attention”.



# Large Language Models

- Most LLMs today are large-scale decoder-only autoregressive (causal) Transformers (>1B parameters).



15

15 Image credit: [Medium.com](https://Medium.com)

# Interim Summary

- Optimization: Learning rate, initialization, activation functions, normalization, shortcut skip connection, attention, etc.

# Interim Summary

- Optimization: Learning rate, initialization, activation functions, normalization, shortcut skip connection, attention, etc.
- Overfitting: Dropout, Data augmentation, etc.

# Interim Summary

- Optimization: Learning rate, initialization, activation functions, normalization, shortcut skip connection, attention, etc.
- Overfitting: Dropout, Data augmentation, etc.
- Architecture Motifs: MLP, CNN, RNN, Transformers, etc.

# Interim Summary

- Optimization: Learning rate, initialization, activation functions, normalization, shortcut skip connection, attention, etc.
- Overfitting: Dropout, Data augmentation, etc.
- Architecture Motifs: MLP, CNN, RNN, Transformers, etc.
- Why deep learning works? Data, optimization, compute.

# Interim Summary

- Optimization: Learning rate, initialization, activation functions, normalization, shortcut skip connection, attention, etc.
- Overfitting: Dropout, Data augmentation, etc.
- Architecture Motifs: MLP, CNN, RNN, Transformers, etc.
- Why deep learning works? Data, optimization, compute.
- Still many open questions: Interpretability, fairness, uncertainty, data efficiency, energy efficiency, theory, etc.

# Interpretability in Deep Neural Networks

# ML Interpretability

- Linear regression: Weights represent feature selection strength.

# ML Interpretability

- Linear regression: Weights represent feature selection strength.
- SVMs: Dual weights represent sample selection.

# ML Interpretability

- Linear regression: Weights represent feature selection strength.
- SVMs: Dual weights represent sample selection.
- Bayesian methods: Model the generative process as a probabilistic model, fully transparent.

# ML Interpretability

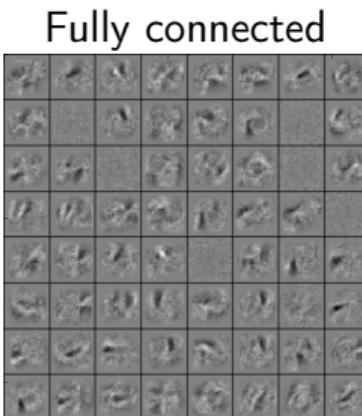
- Linear regression: Weights represent feature selection strength.
- SVMs: Dual weights represent sample selection.
- Bayesian methods: Model the generative process as a probabilistic model, fully transparent.
- Decision trees: If-else decision making process.

# ML Interpretability

- Linear regression: Weights represent feature selection strength.
- SVMs: Dual weights represent sample selection.
- Bayesian methods: Model the generative process as a probabilistic model, fully transparent.
- Decision trees: If-else decision making process.
- Neural networks: ?

# Feature Visualization

- Recall: we can understand what first-layer features are doing by visualizing the weight matrices.



Zeiler and Fergus, Visualizing and understanding convolutional networks, ECCV 2014.

- The better the input matches these weights, the more the feature activates.
- Higher-level weight matrices are hard to interpret.
  - Obvious generalization: visualize higher-level features by seeing what inputs activate them.

## Feature Visualization

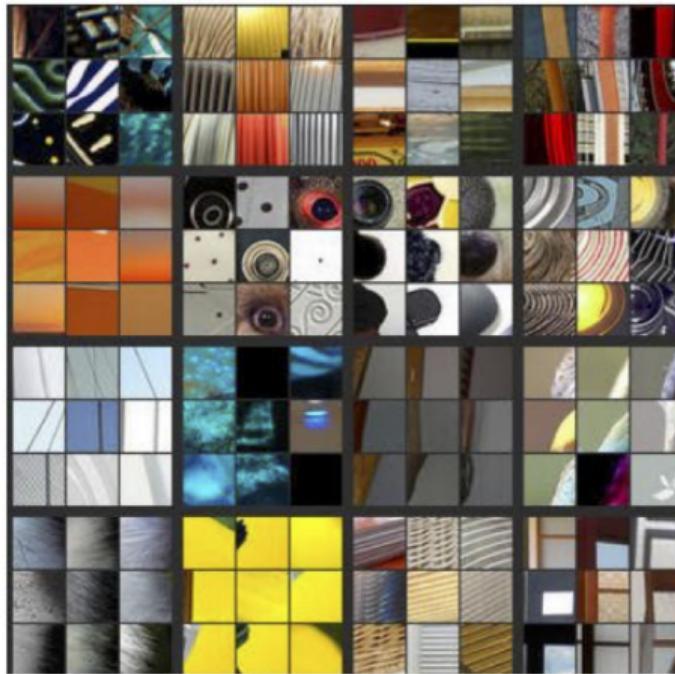
- One way to formalize: pick the images in the training set which activate a unit most strongly.
- Here's the visualization for layer 1:



Zeiler and Fergus, Visualizing and understanding convolutional networks, ECCV 2014.

# Feature Visualization

- Layer 3:



Zeiler and Fergus, Visualizing and understanding convolutional networks, ECCV 2014.

# Feature Visualization

- Layer 4:



## Feature Visualization

- Layer 5:



# Feature Visualization

- Higher layers seem to pick up more abstract, high-level information.

# Feature Visualization

- Higher layers seem to pick up more abstract, high-level information.
- Problems?

## Feature Visualization

- Higher layers seem to pick up more abstract, high-level information.
- Problems?
  - Can't tell what the unit is actually responding to in the image.

# Feature Visualization

- Higher layers seem to pick up more abstract, high-level information.
- Problems?
  - Can't tell what the unit is actually responding to in the image.
  - We may read too much into the results, e.g. a unit may detect red, and the images that maximize its activation will all be stop signs.

# Feature Visualization

- Higher layers seem to pick up more abstract, high-level information.
- Problems?
  - Can't tell what the unit is actually responding to in the image.
  - We may read too much into the results, e.g. a unit may detect red, and the images that maximize its activation will all be stop signs.
- Can use input gradients to diagnose what the unit is responding to.

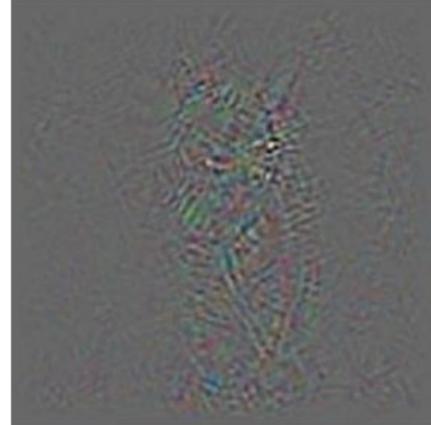
# Feature Visualization

- Input gradients can be hard to interpret.
- Take a good object recognition conv net (Alex Net) and compute the gradient of  $\log p(y = \text{"cat"}|x)$ :

Original image



Gradient for “cat”

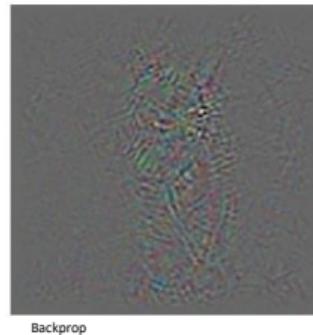


# Feature Visualization

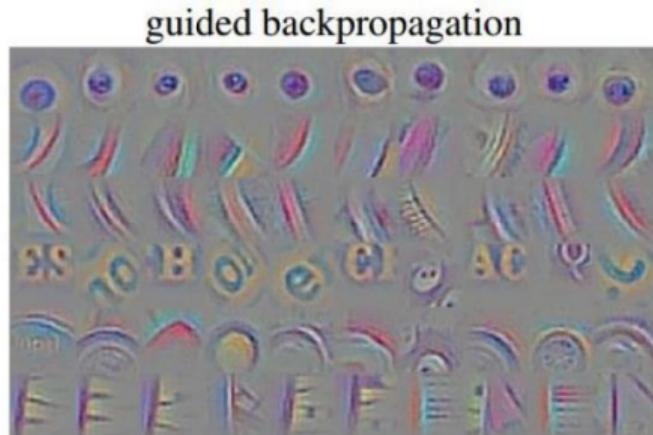
- Guided backprop is a total hack to prevent this cancellation.
- Do the backward pass as normal, but apply the ReLU nonlinearity to all the activation error signals.

$$y = \text{ReLU}(z) \quad \bar{z} = \begin{cases} \bar{y} & \text{if } z > 0 \text{ and } \bar{y} > 0 \\ 0 & \text{otherwise} \end{cases}$$

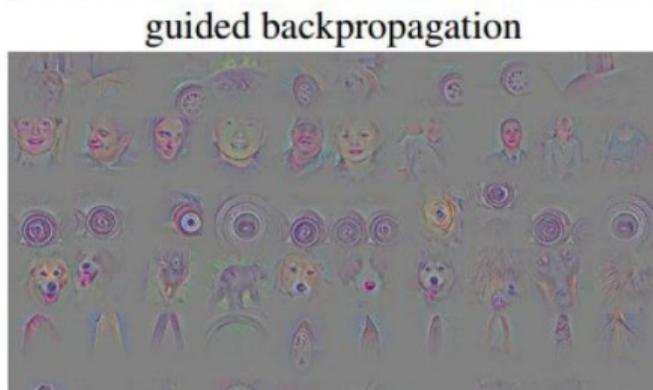
- We want to visualize what excites given unit, not what suppresses it.



# Guided Backprop



corresponding image crops

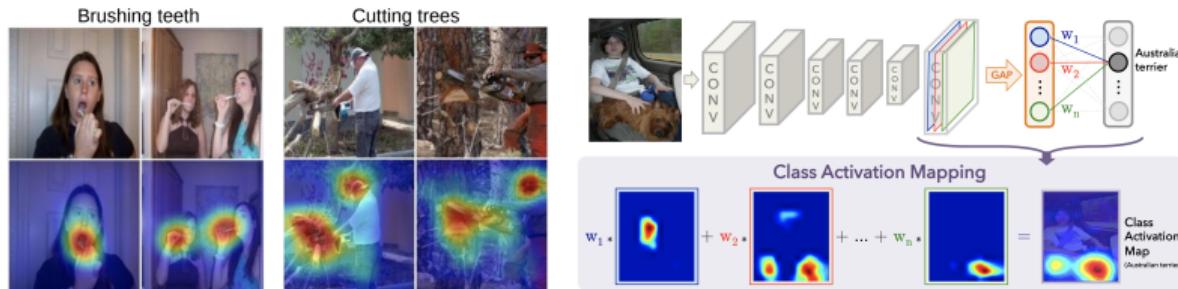


corresponding image crops



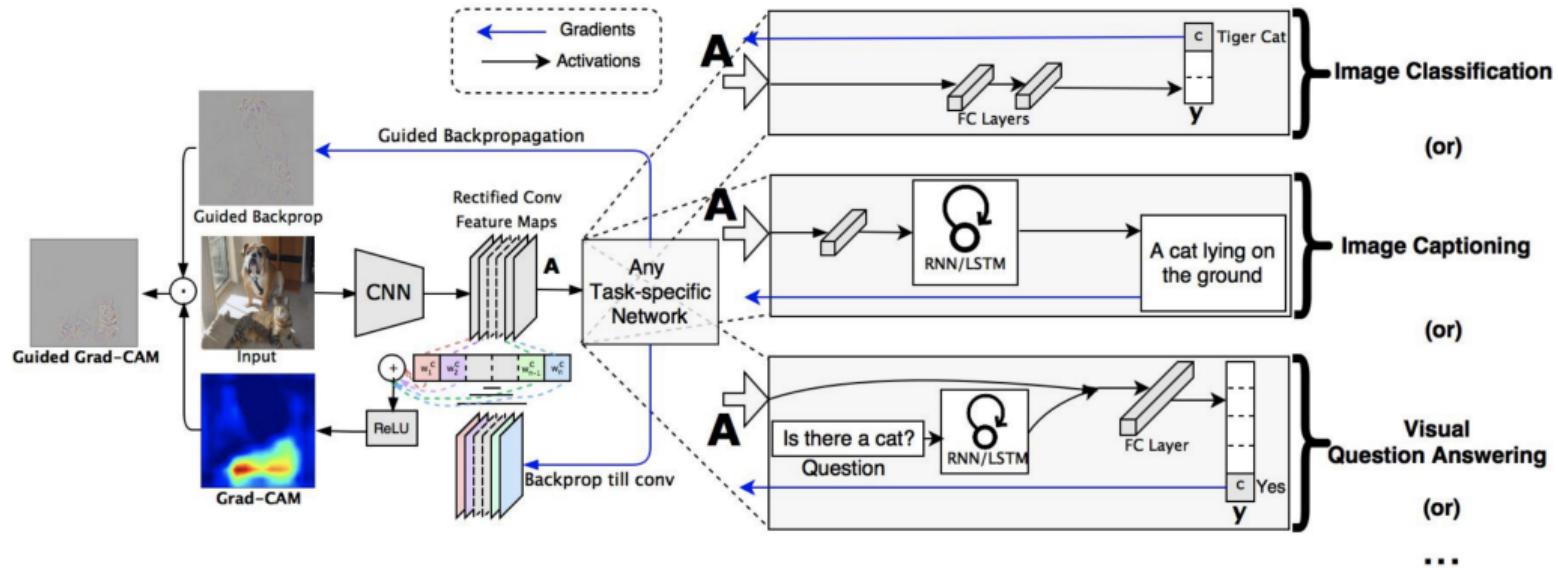
# Class activation map (CAM)

- Classification networks typically use global avg pooling before the final layer.
- This pooling layer can already contain semantic information.
- We can visualize a heat map



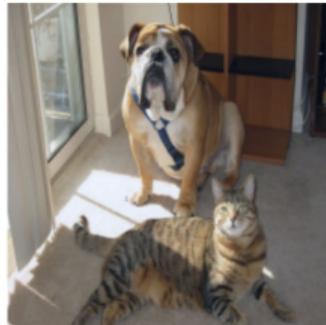
Zhou et al. Learning deep features for discriminative localization. CVPR 2016.

# GradCAM



Selvaraju et al. Grad-CAM: Visual explanations from deep networks via gradient-based localization. ICCV 2017.

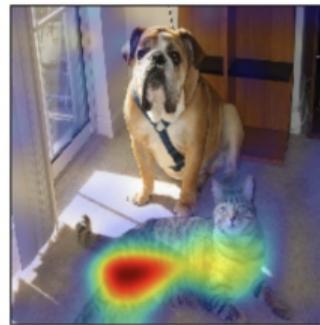
# GradCAM



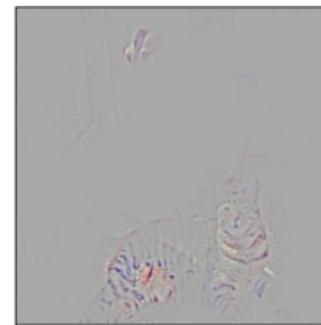
(a) Original Image



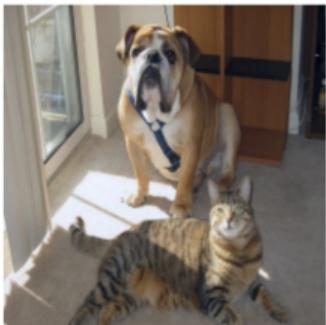
(b) Guided Backprop ‘Cat’



(c) Grad-CAM ‘Cat’



(d) Guided Grad-CAM ‘Cat’



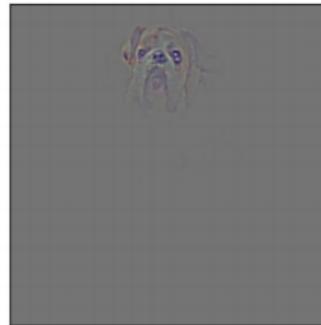
(g) Original Image



(h) Guided Backprop ‘Dog’



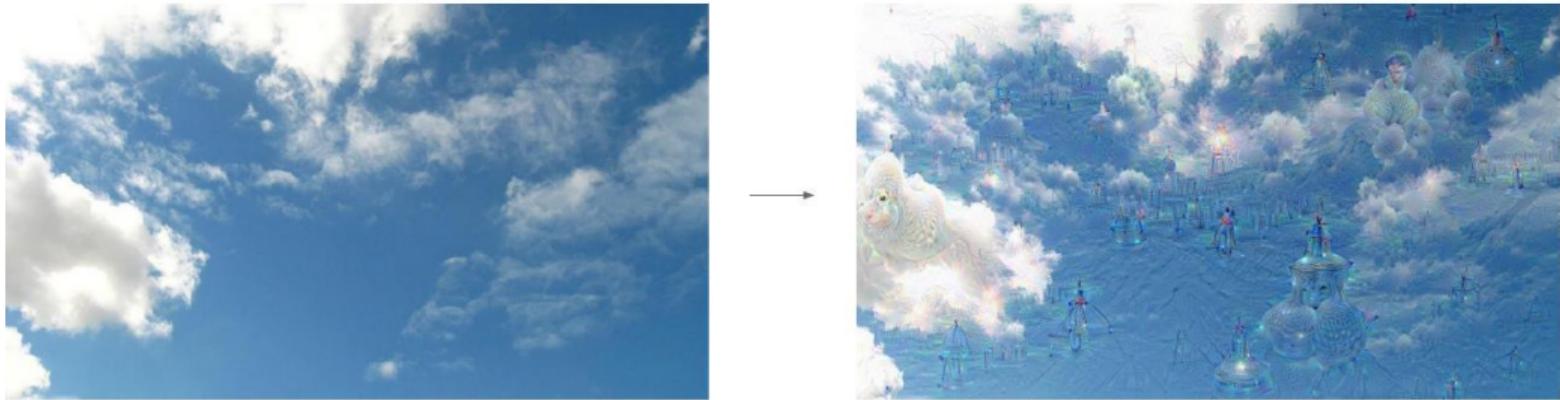
(i) Grad-CAM ‘Dog’



(j) Guided Grad-CAM ‘Dog’

# DeepDream<sup>16</sup>

- Start with an image, and run a conv net on it.
- Change the image such that units which were already highly activated get activated even more strongly. “Rich get richer.”



<sup>16</sup>Google Research Blog

# DeepDream



"Admiral Dog!"



"The Pig-Snail"



"The Camel-Bird"



"The Dog-Fish"

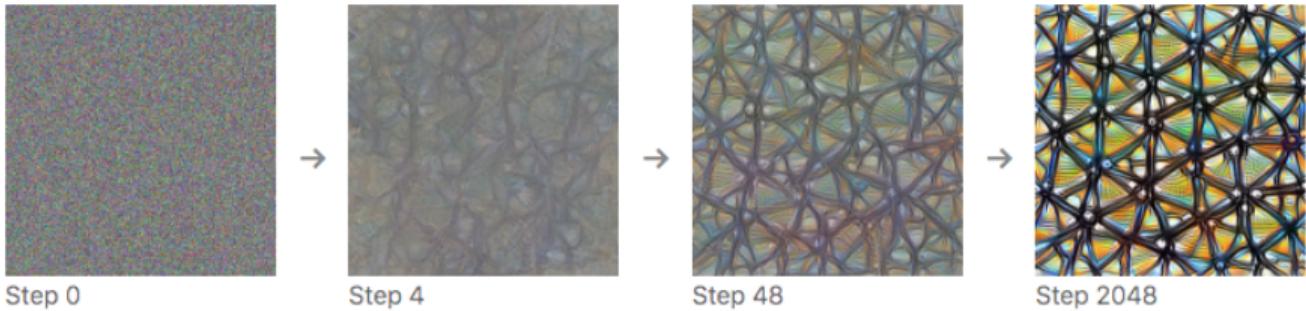
# DeepDream



# Gradient Ascent on Images

- Doing gradient ascent on an image to maximize the activation of a given neuron.

Starting from random noise, we optimize an image to activate a particular neuron (layer mixed4a, unit 11).



<https://distill.pub/2017/feature-visualization/>

# Gradient Ascent on Images

**Dataset Examples** show us what neurons respond to in practice



**Optimization** isolates the causes of behavior from mere correlations. A neuron may not be detecting what you initially thought.



Baseball—or stripes?  
*mixed4a, Unit 6*

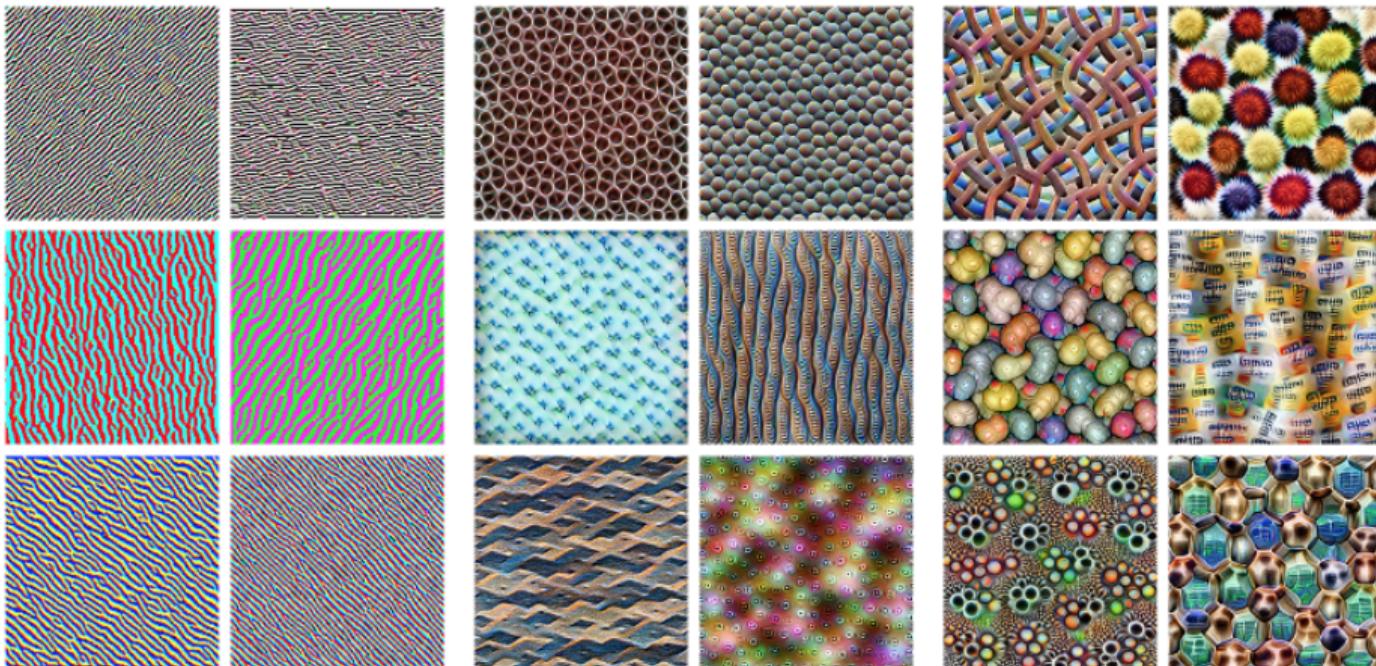
Animal faces—or snouts?  
*mixed4a, Unit 240*

Clouds—or fluffiness?  
*mixed4a, Unit 453*

Buildings—or sky?  
*mixed4a, Unit 492*

# Gradient Ascent on Images

- Higher layers in the network often learn higher-level, more interpretable representations



<https://distill.pub/2017/feature-visualization/>

# Gradient Ascent on Images

- Higher layers in the network often learn higher-level, more interpretable representations



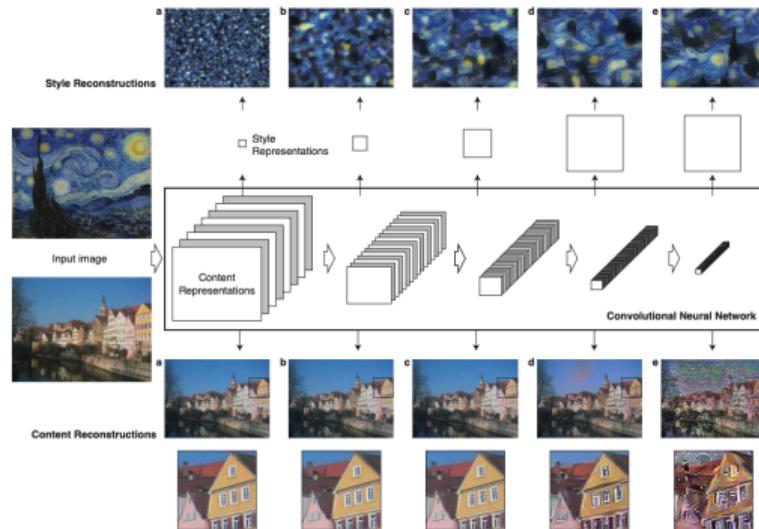
**Parts** (layers mixed4b & mixed4c)

**Objects** (layers mixed4d & mixed4e)

<https://distill.pub/2017/feature-visualization/>

# Artistic style transfer

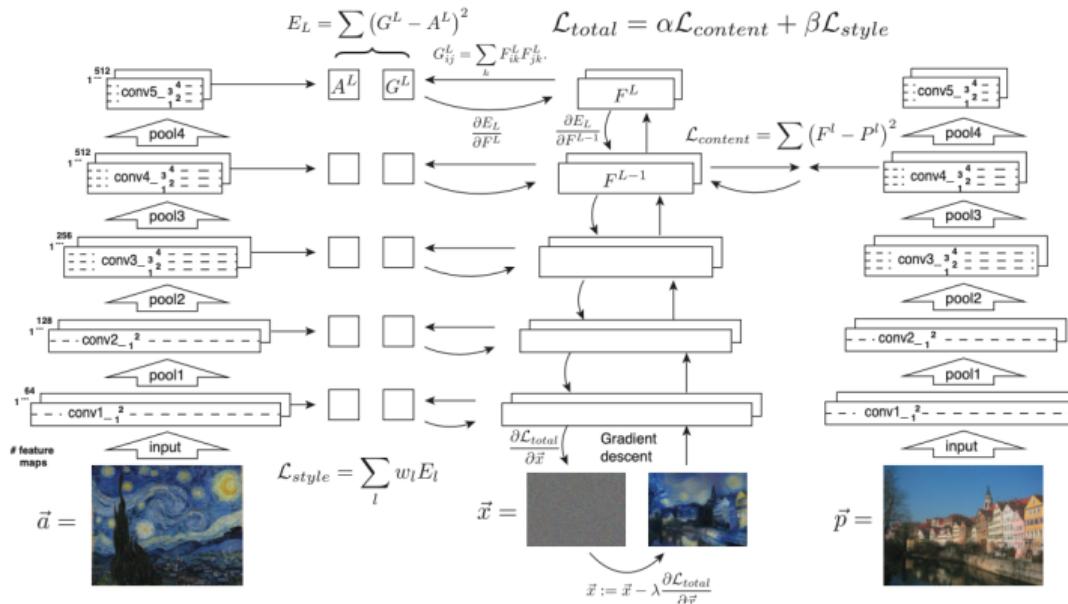
- Activations store content information
- Activation correlation stores style/textture information:  $G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l$



Gatys et al., Image style transfer using convolutional neural networks, CVPR 2016.

# Artistic style transfer

- Optimizing both content & style from random noise



Gatys et al., Image style transfer using convolutional neural networks, CVPR 2016.

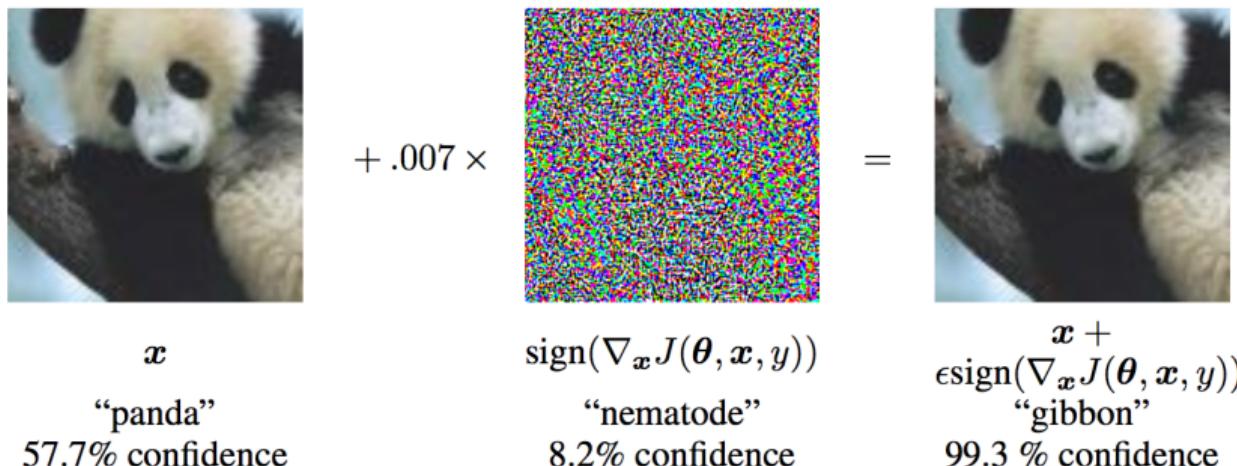
# Artistic style transfer



Gatys et al., Image style transfer using convolutional neural networks, CVPR 2016.

# Adversarial Examples

- One of the most surprising findings about neural nets has been the existence of **adversarial inputs**, i.e. inputs optimized to fool an algorithm.



Goodfellow et al., Explaining and harnessing adversarial examples, ICLR 2015.

# Adversarial Examples

- The following adversarial examples are misclassified as ostriches. (  $10 \times$  perturbation visualized in middle.)



Szegedy et al., Intriguing properties of neural networks, ICLR 2014.

# Adversarial Examples

- You can print out an adversarial image and take a picture of it, and it still works!



(a) Printout



(b) Photo of printout



(c) Cropped image

Kurakin et al., Adversarial examples in the physical world, ICLR workshop 2017.

# Adversarial Examples

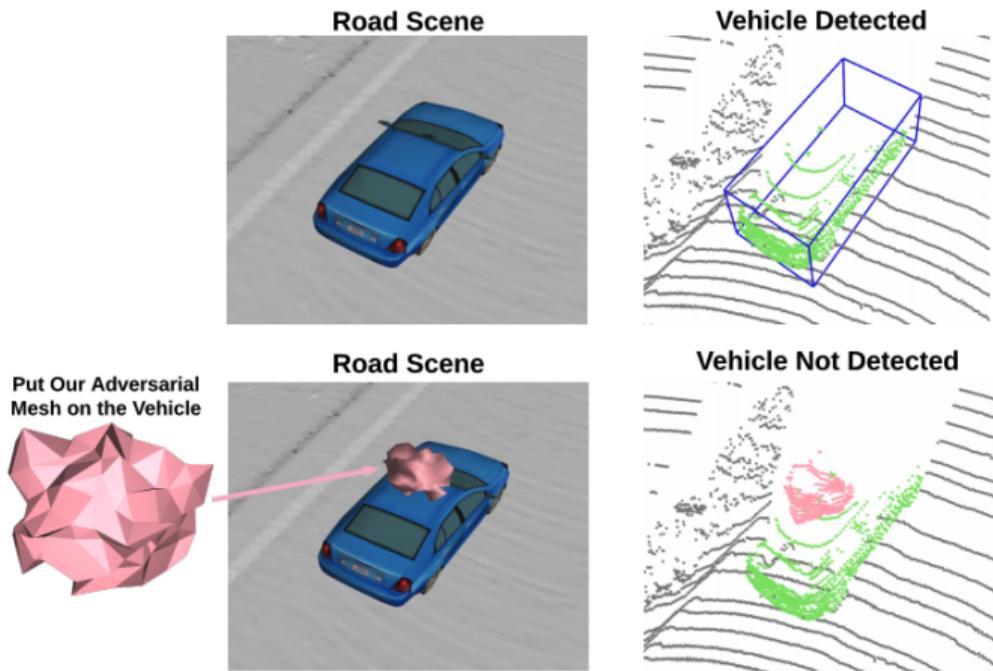
- An adversarial example in the physical world (network thinks it's a gun, from a variety of viewing angles!)



Athalye et al., Synthesizing robust adversarial examples, ICML 2018.

# Adversarial Examples

- An adversarial mesh object that can hide cars from LiDAR detector



Tu et al., Physically realizable adversarial examples for LiDAR object detection, CVPR 2020.

# Adversarial Defense

- How to defend from adversarial perturbation is still an active research area.

# Adversarial Defense

- How to defend from adversarial perturbation is still an active research area.
- Blackbox vs. whitebox attacks.

## Adversarial Defense

- How to defend from adversarial perturbation is still an active research area.
- Blackbox vs. whitebox attacks.
- One common approach is to train with millions of adversarial examples.
- Needs to train much longer, and also suffers a drop in accuracy.

## Adversarial Defense

- How to defend from adversarial perturbation is still an active research area.
- Blackbox vs. whitebox attacks.
- One common approach is to train with millions of adversarial examples.
- Needs to train much longer, and also suffers a drop in accuracy.
- Data augmentation and label smoothing also help.

# Summary

- Interpretability - ways to open up the black box of neural networks

## Summary

- Interpretability - ways to open up the black box of neural networks
- Knowing what each neuron does is like studying a “brain” with perfect observation and measurement.

# Summary

- Interpretability - ways to open up the black box of neural networks
- Knowing what each neuron does is like studying a “brain” with perfect observation and measurement.
- Still very open research area.

## Summary

- Interpretability - ways to open up the black box of neural networks
- Knowing what each neuron does is like studying a “brain” with perfect observation and measurement.
- Still very open research area.
- Adversarial examples are safety vulnerabilities of deep neural networks.

## Summary

- Interpretability - ways to open up the black box of neural networks
- Knowing what each neuron does is like studying a “brain” with perfect observation and measurement.
- Still very open research area.
- Adversarial examples are safety vulnerabilities of deep neural networks.
- Need more data and innovations in more robust learning objectives.