Neural Networks II: Deep Learning

Mengye Ren

(Slides credit to David Rosenberg, He He, et al.)

NYU

Nov 26, 2024

Slides

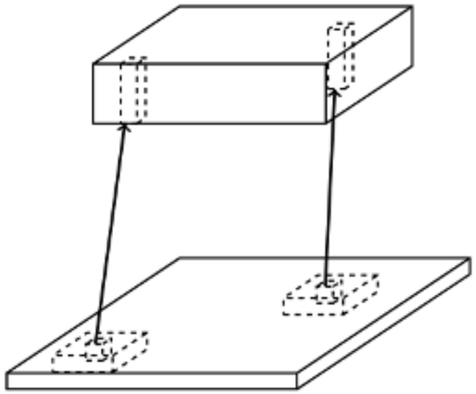
Logistics

- Homework 4 Due: Dec 3.
- Last lecture: Dec 10 Project presentation.
- Presentation order: Your assigned Group ID.
- Each group has a max of 4 minutes (hard stop) + 1 min Q&A.
- OH this week: Wednesday 1-2pm.

Dec 9. send slides.
in PDF.

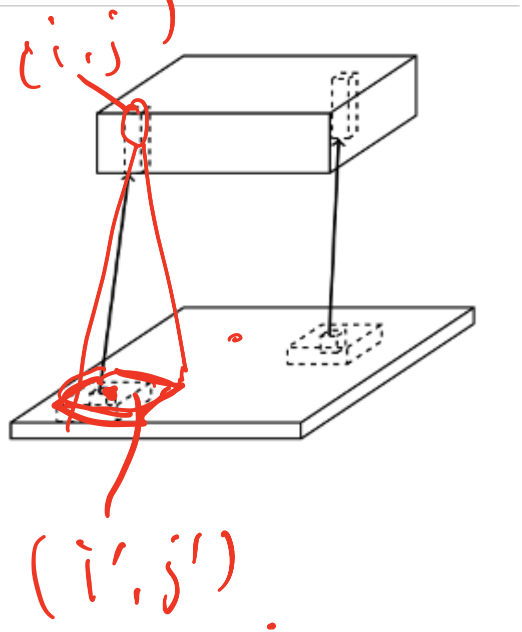
Local connection patterns

- 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).



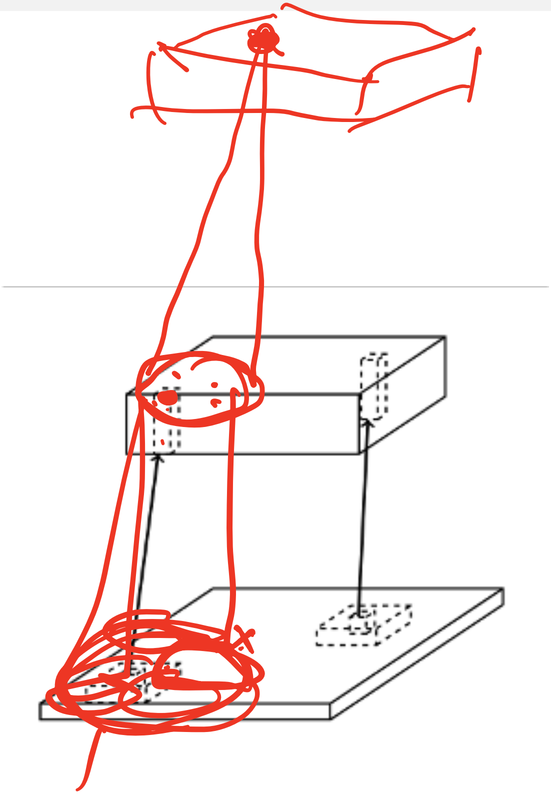
Local connection patterns

- 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.
- $z_{i,j,c} = \sum_{i' \in [i\pm k], j' \in [j \pm k], c'} x_{i'j'c'} w_{i,j,i'-i,j'-j,c',c}$ 6 dim



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.
- $z_{i,j,c} = \sum_{i' \in [i \pm k], j' \in [j \pm k], c'} x_{i'j'c'} w_{i,j,i'-i,j'-j,c',c}$
- 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

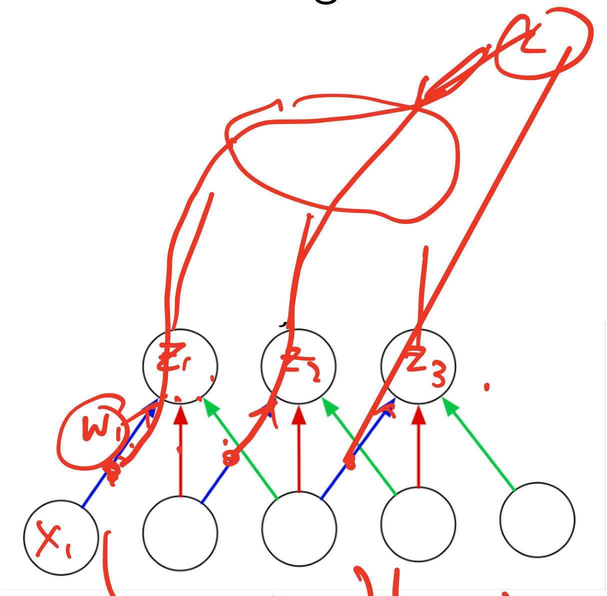
Fully Connected → Local

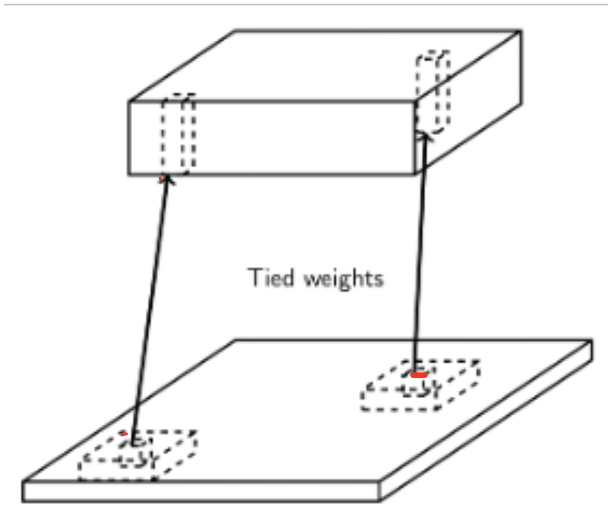
↓ weight sharing

↓ CNN.

- 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.

$\boxed{\frac{\partial L}{\partial w_1}}$

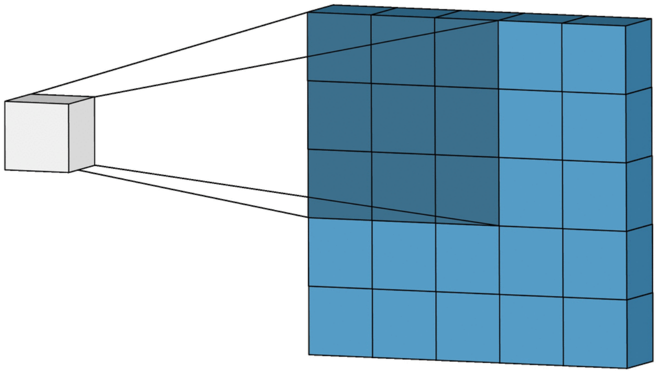




$\frac{\partial L}{\partial z_1} \cdot x_1$

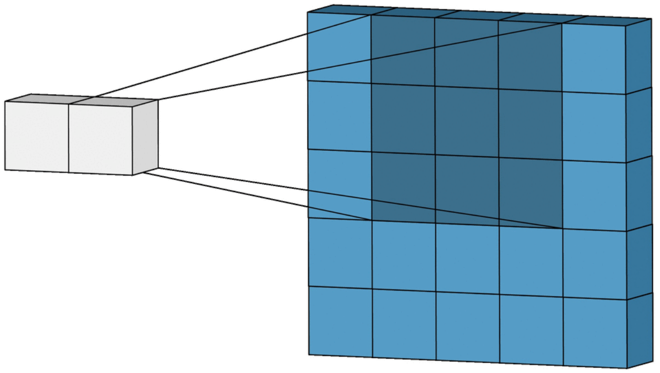
2D convolution

- 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"
- $z_{i,j,c} = \sum_{i' \in [i \pm k], j' \in [j \pm k], c'} x_{i'j'c'} {\color{red} w_{i-i',j-j',c',c}}$



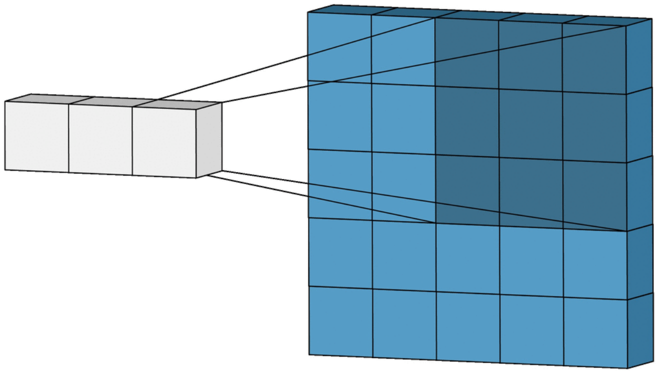
2D convolution

- 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"
- $z_{i,j,c} = \sum_{i' \in [i \pm k], j' \in [j \pm k], c'} x_{i'j'c'} \color{red}{w_{i-i',j-j',c',c}}$



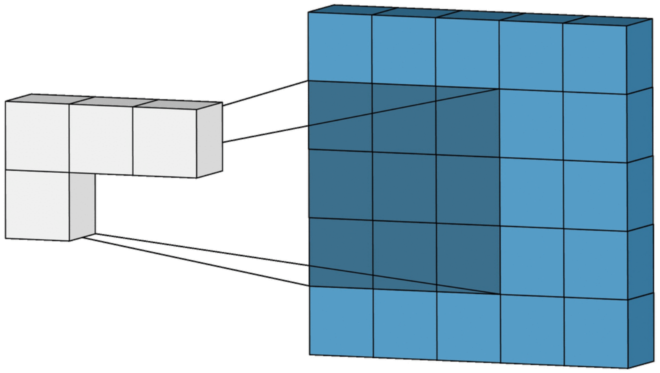
2D convolution

- 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"
- $z_{i,j,c} = \sum_{i' \in [i \pm k], j' \in [j \pm k], c'} x_{i'j'c'} \color{red}{w_{i-i', j-j', c', c}}$



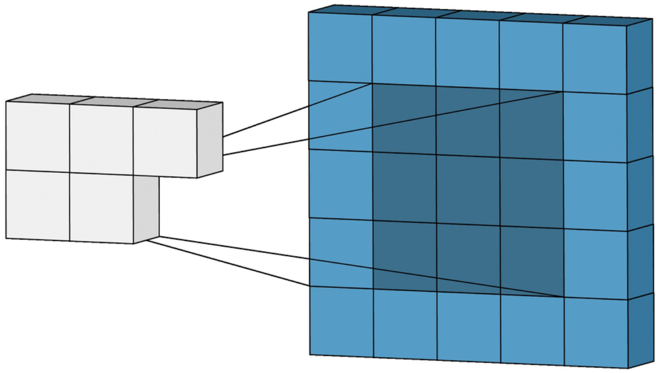
2D convolution

- 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"
- $z_{i,j,c} = \sum_{i' \in [i \pm k], j' \in [j \pm k], c'} x_{i'j'c'} w_{i-i',j-j',c',c}$



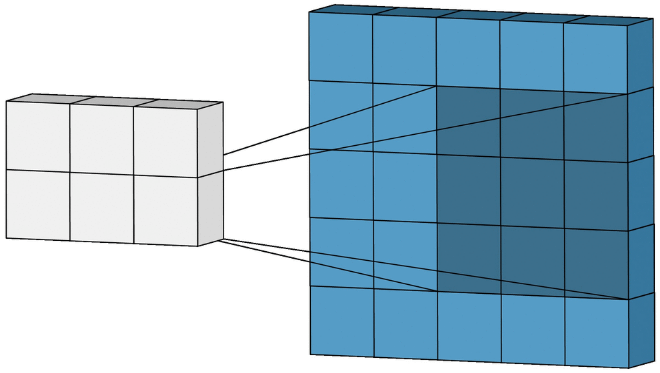
2D convolution

- 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"
- $z_{i,j,c} = \sum_{i' \in [i \pm k], j' \in [j \pm k], c'} x_{i'j'c'} \textcolor{red}{w_{i-i', j-j', c', c}}$



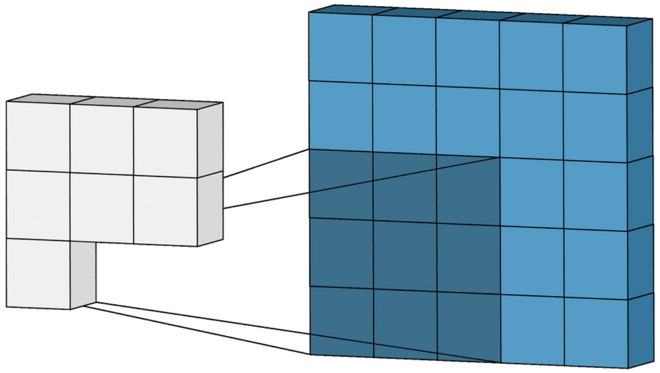
2D convolution

- 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"
- $z_{i,j,c} = \sum_{i' \in [i \pm k], j' \in [j \pm k], c'} x_{i'j'c'} {\color{red} w_{i-i',j-j',c',c}}$



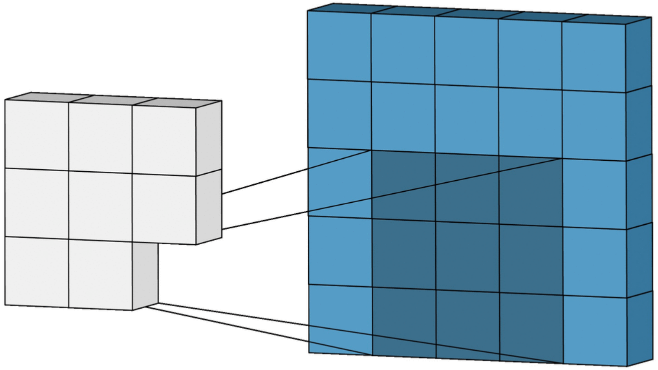
2D convolution

- 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"
- $z_{i,j,c} = \sum_{i' \in [i \pm k], j' \in [j \pm k], c'} x_{i'j'c'} w_{i-i', j-j', c', c}$



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"
- $z_{i,j,c} = \sum_{i' \in [i \pm k], j' \in [j \pm k], c'} x_{i'j'c'} w_{i-i', j-j', c', c}$

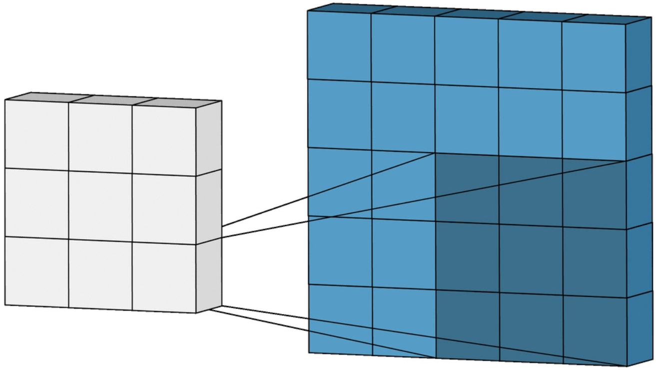


2D convolution

- 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"
- $z_{i,j,c} = \sum_{i' \in [i \pm k], j' \in [j \pm k], c'} x_{i'j'c'} w_{\underbrace{i-i',j-j'}_{\text{neighborhood index}},\underbrace{c',c}}$

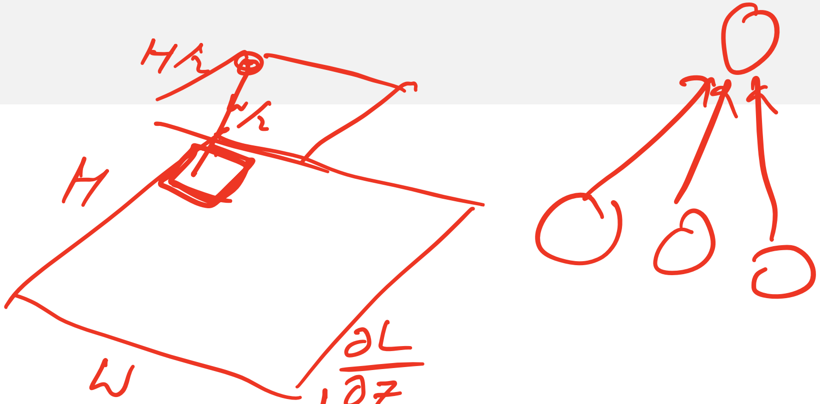
neighborhood index

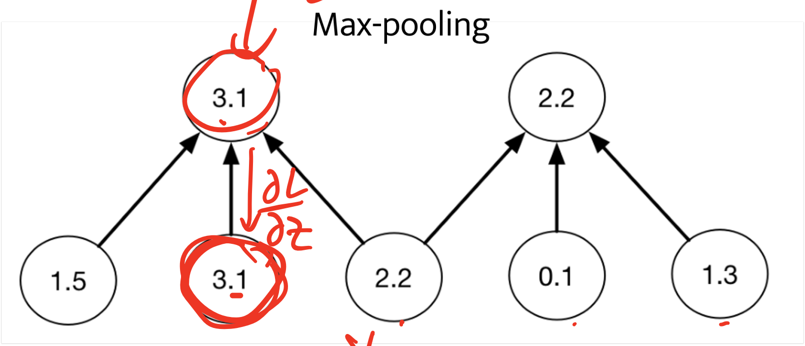
input channel
output channel

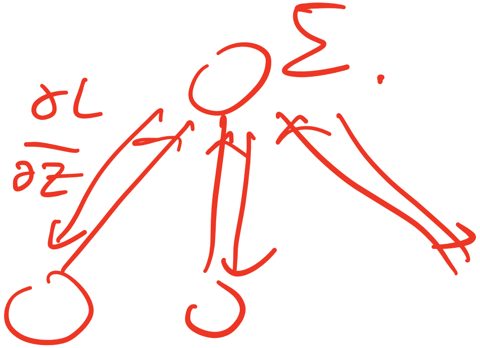


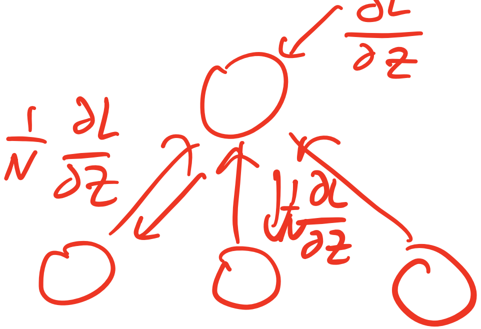
Pooling

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



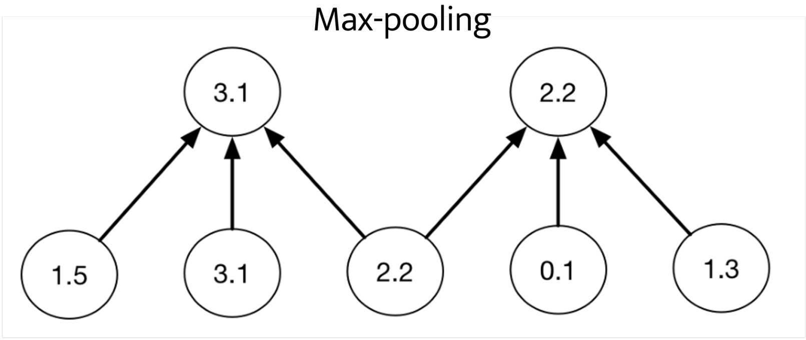


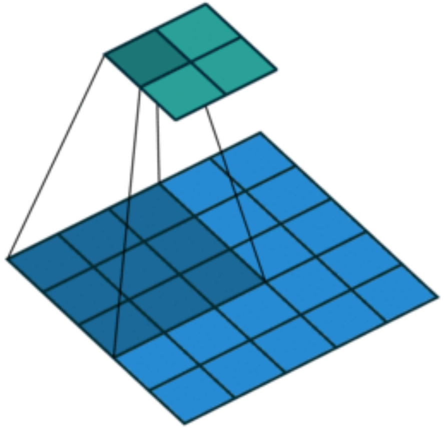




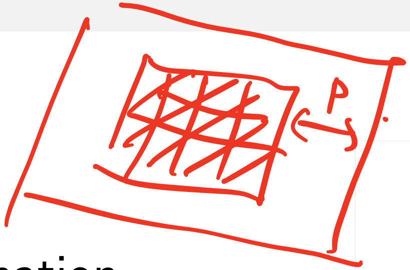
Pooling

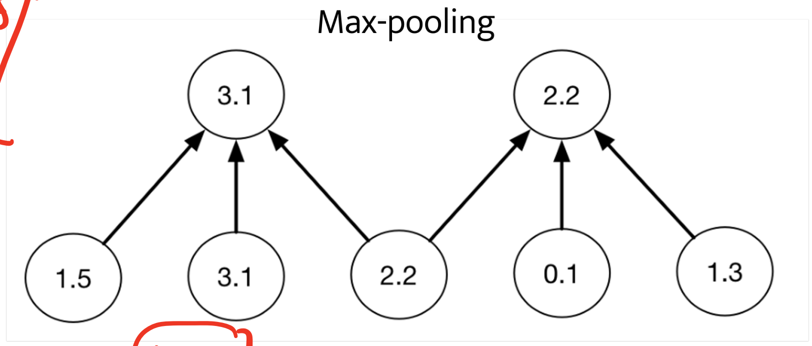
- Need to summarize global information more efficiently.
- Pooling reduces image / activation dimensions.
- Max-pooling or average-pooling
- You can also perform a "strided" convolution by jumping multiple steps.



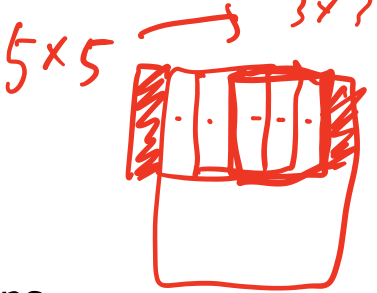


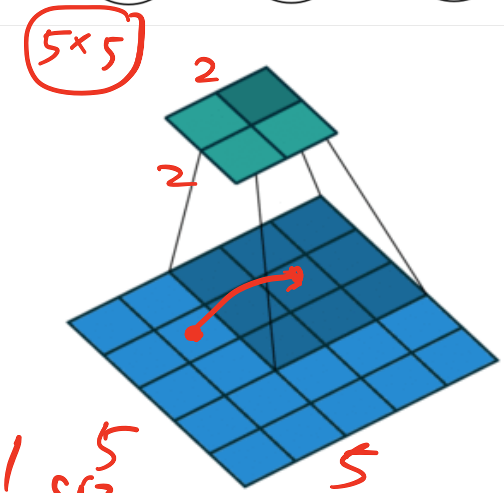
Pooling





- 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.

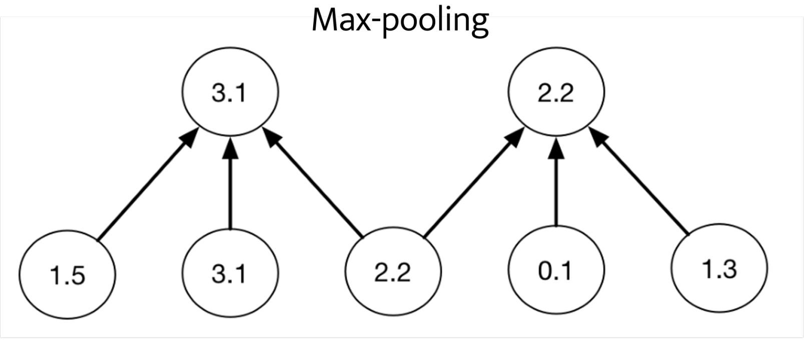


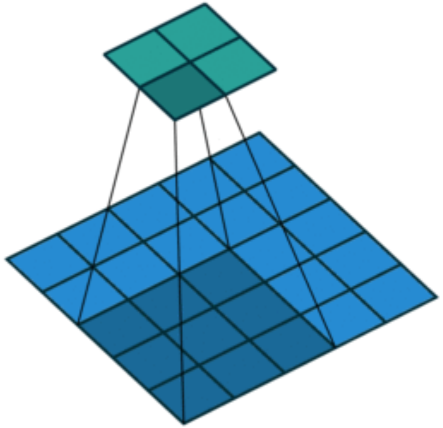


Out = function of (inp size, kernel size, stride, padding)

Pooling

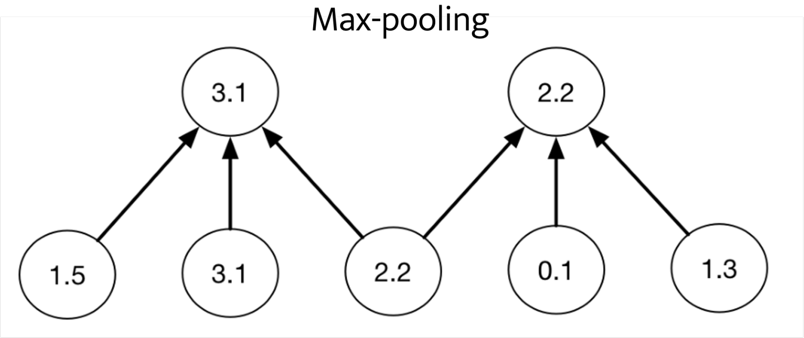
- Need to summarize global information more efficiently.
- Pooling reduces image / activation dimensions.
- Max-pooling or average-pooling
- You can also perform a “strided” convolution by jumping multiple steps.

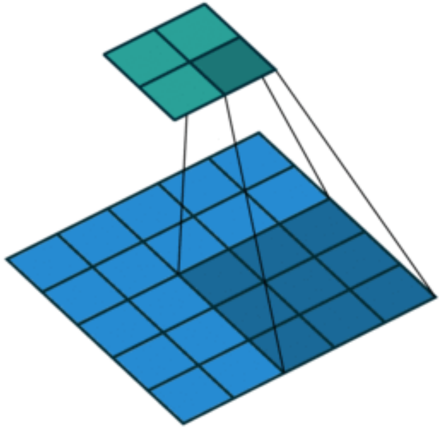




Pooling

- 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.

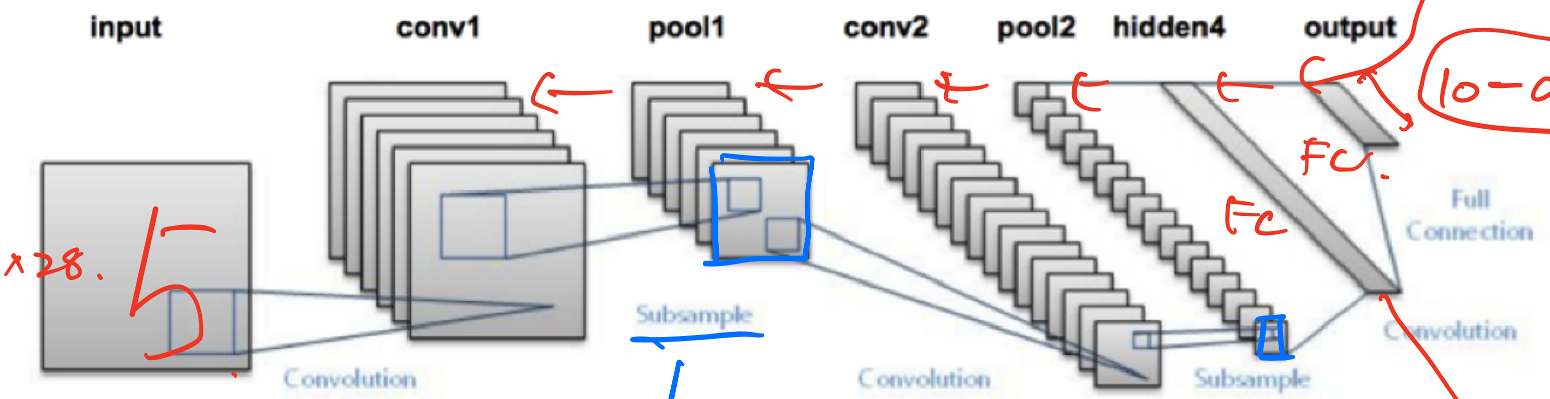




Assembling together: LeNet

cross entropy

0-9.



28×28.

10-dim.

FC.

FC

pooling.

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

7×7

4×4.

4×4×64. → 6K. avg

↪ 16×64 Concat.

vector
global
representation.

Historical development

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

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.

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.

g_{out}

- Loss often stalls or blows up.

$J_3 \quad J_2 \quad J_1 \quad \ldots =$

- Why?
  - Backpropagation: multiplying the Jacobian $\frac{\partial y}{\partial x}$ by each layer. M N g_{inp}
  - 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.

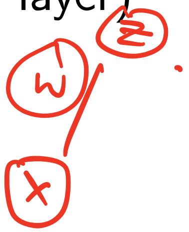
Kaiming Initialization

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

[1] He et al. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet. ICCV, 2015.

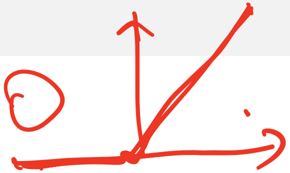
Kaiming Initialization

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



[1] He et al. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet. ICCV, 2015.

Kaiming Initialization



Var $\quad Var[x^+] + Var(x^-] $, with $\frac{1}{2}$ under $Var[x^+]$ and 0 under $Var(x^-]$

- Suppose each neuron and weight connection are sampling from a random distribution.
- At l-th layer, $Var[z_l] = n_l Var[w_l x_l]$ (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, $x_{l+1} = \frac{1}{2} Var[z_l]$.

[1] 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, $Var[z_l] = n_l Var[w_l x_l]$ (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, $x_{l+1} = \frac{1}{2} Var[z_l]$.
- Putting altogether, $Var[x_{l+1}] = \frac{1}{2} n_l Var[w_l] Var[x_l]$.

[1] 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, $Var[z_l] = n_l Var[w_l x_l]$ (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, $x_{l+1} = \frac{1}{2} Var[z_l]$.
- Putting altogether, $x_{l+1} = \frac{1}{2} n_l Var[w_l] Var[x_l]$.
- To make the variance constant, we need $\boxed{\frac{1}{2} n_l Var[w_l] = 1}$, $Std[w_l] = \sqrt{2/n_l}$[1].

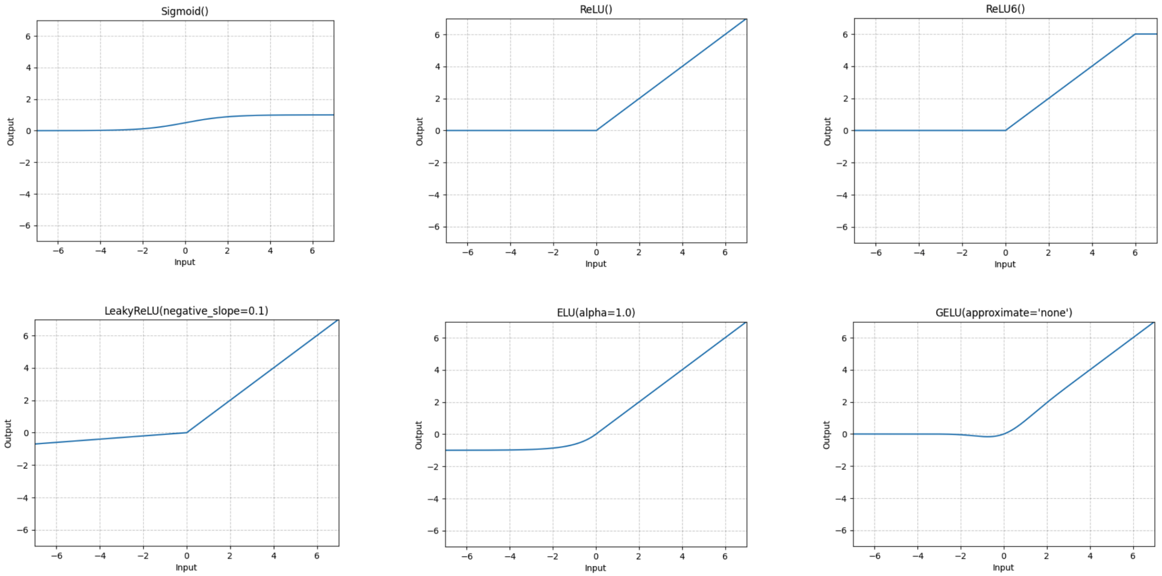
more inp neurons
small std. of weights.

$\sqrt{\frac{1}{n_{in}}}$
w/o relu.

[1] He et al. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet. ICCV, 2015.

Activation functions

- ReLU was proposed in 2009-2010[2,3], and was successfully used in AlexNet in 2012[4].
- Address the vanishing gradient issue in activations, comparing to sigmoid or tanh.



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

[2] Jarrett et al. What is the Best Multi-Stage Architecture for Object Recognition? ICCV, 2009.
[3] Nair & Hinton/ Rectified Linear Units Improve Restricted Boltzmann Machines. ICML, 2010.
[4] Krizhevsky et al. ImageNet Classification with Deep Convolutional Neural Networks. NIPS, 2012.

SGD Learning Rate

mini-batch.

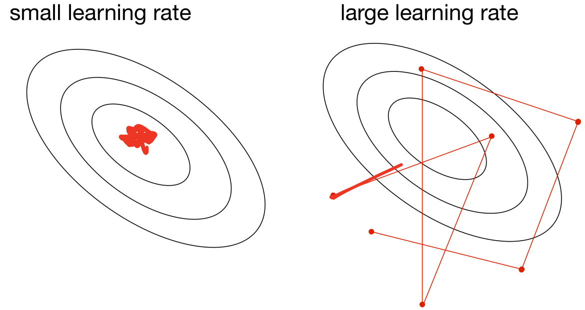
slower.

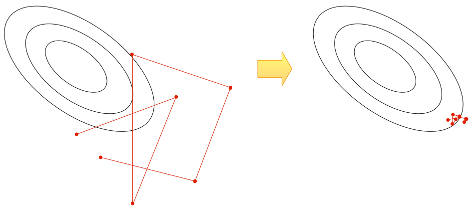
- In stochastic training, the learning rate also influences the fluctuations due to the stochasticity of the gradients.
- Typical strategy:
  - Use a large learning rate early in training so you can get close to the optimum.
  - Gradually decay the learning rate to reduce the fluctuations.

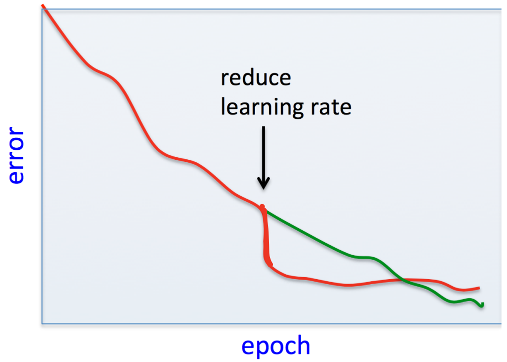
faster

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:

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

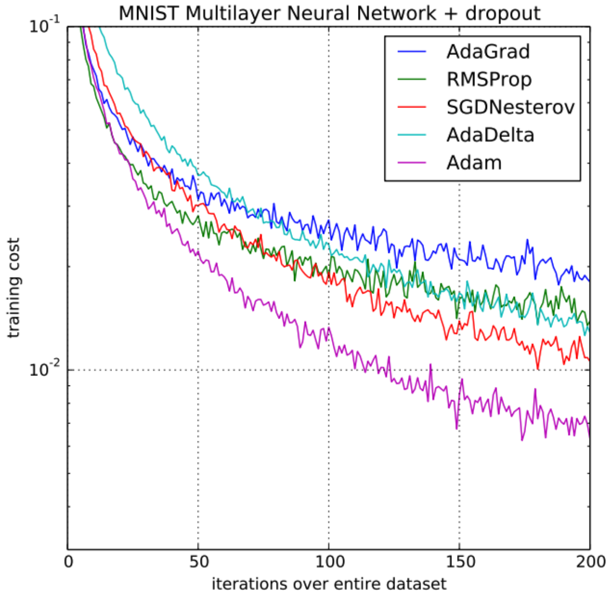
exponential moving avg.

0.0001.

if norm is small

Adam optimizer

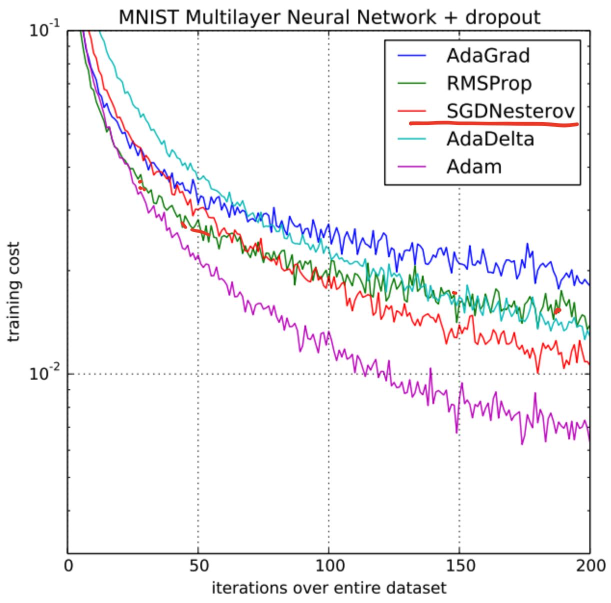
- Adam = RMSprop + momentum = Adaptive Momentum estimation
- Smoother estimate of the average gradient and gradient norm.



Adam optimizer

→ Adam W.

- 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

t

weight

time

0

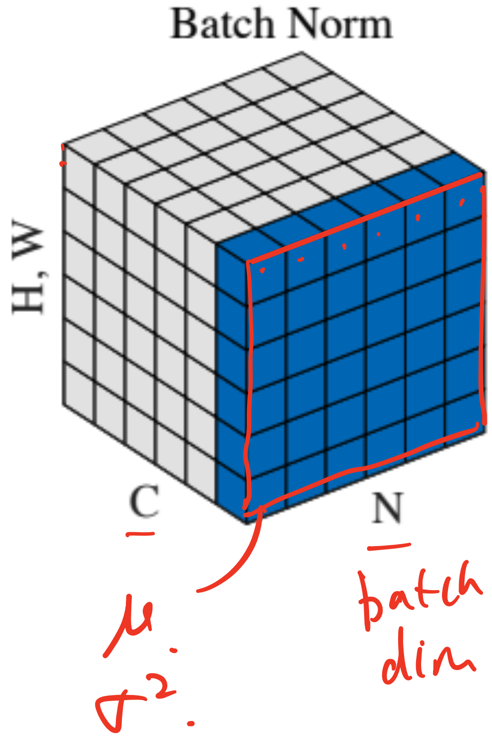
- Weight initialization is tricky, and there is no guarantee that the distribution of activations will stay the same over the learning process.
- What if the weights keep grow bigger and activation may explode?

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.

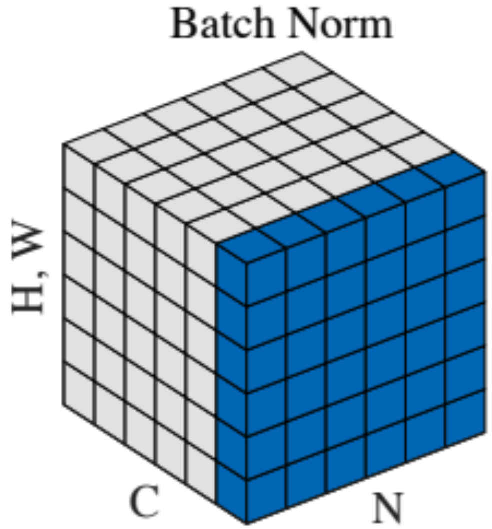
Batch Normalization (BN)

- 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)

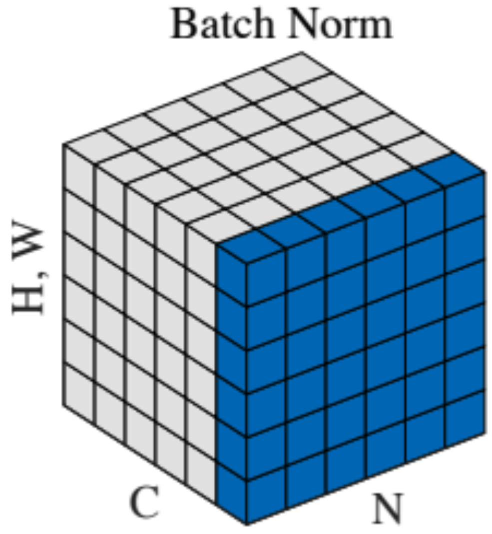
$$\tilde{x} = \frac{x - \mu}{\sigma}.$$

wrong.

$$\boxed{\frac{1}{\sigma} \cdot \frac{\partial L}{\partial \tilde{x}}.}$$

- 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.
- $\tilde{x} = \gamma \boxed{\frac{x-\mu}{\sigma}} + \beta$ — channel c.
- γ, β: learnable parameters. μ, σ: statistics from the training batch.

Batch-normalized activation.



Batch Normalization (BN)

$\mu\ \sigma.$

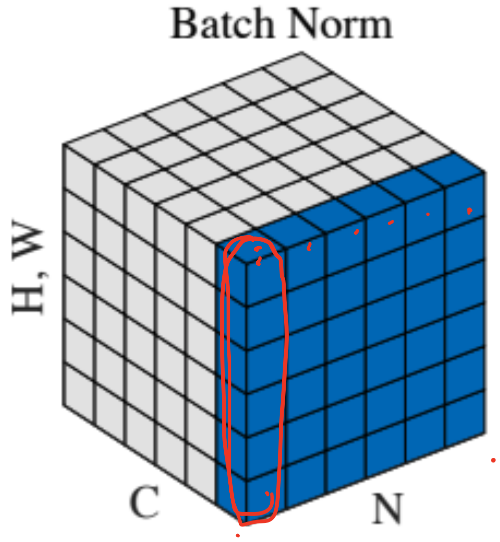
$\mu\ \sigma.$

- 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.

$\lambda \mu + (1-\lambda)\cdot \mu_{ema}.$

- $\tilde{x} = \gamma \frac{x-\mu}{\sigma + \epsilon} + \beta$ $+\epsilon$ 1e-7 μ_{ema} σ_{ema} σ.
- γ, β: learnable parameters. μ, σ: statistics from the training batch.
- Test time: using the mean and variance from the entire training set.

one image.



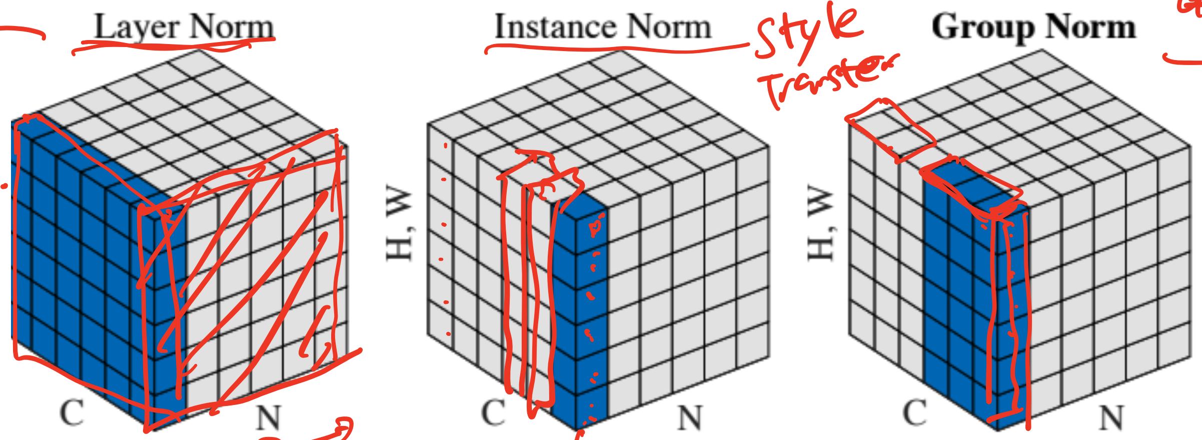
BN Alternatives

- 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.

HW·C HW GAN. HW·$\frac{C}{G}$

style Transfer

Transformer.
LLMs...



[5]

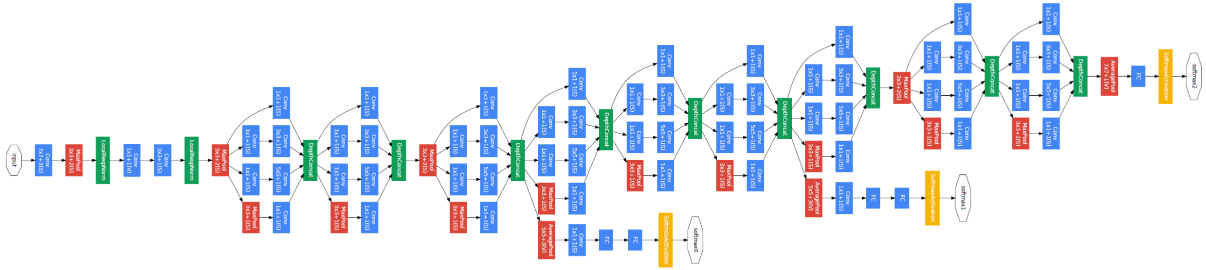
→ ResNet.
↳ BN
↳ GN.

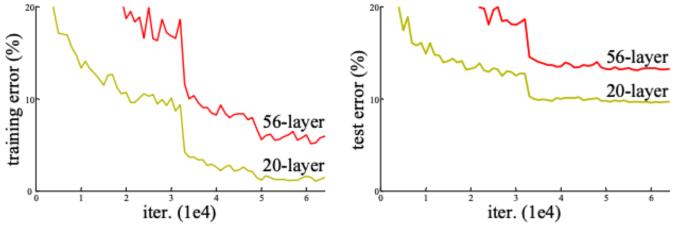
The smaller normalization field the less degree of freedom.

[5] Wu and He. Group normalization. ECCV 2018.

Going Deeper

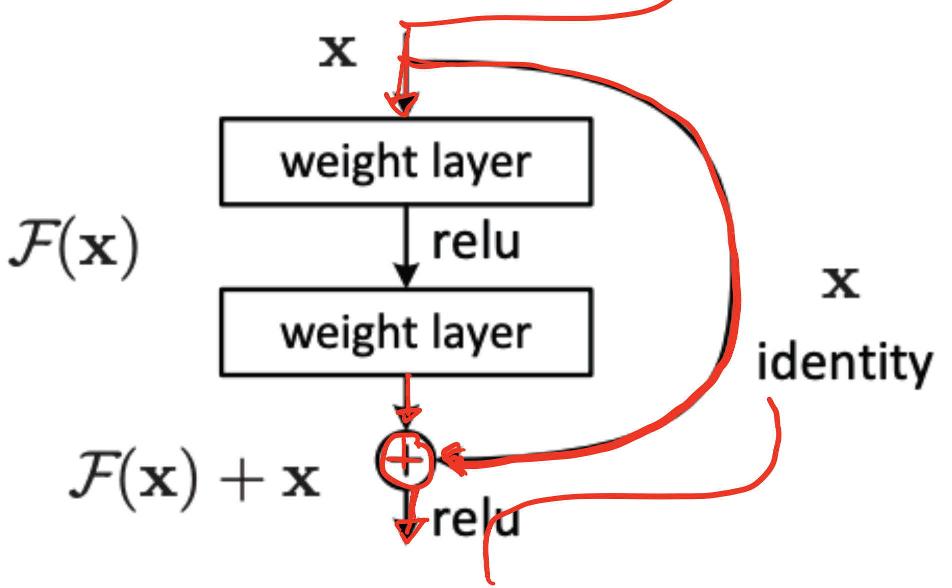
- The progress of normalization allowed us to train even deeper networks.
- The networks are no longer too sensitive with initialization.
- But the best networks were still around 20 layers and deeper results in worse performance.





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.[6]



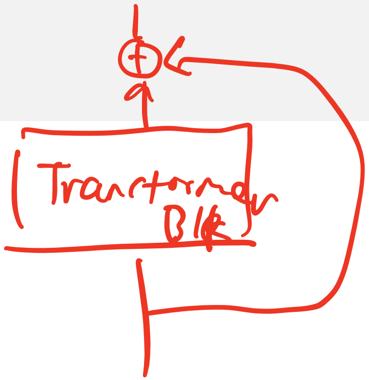
[6] He et al. Deep Residual Learning for Image Recognition. CVPR 2016.

ResNet Success

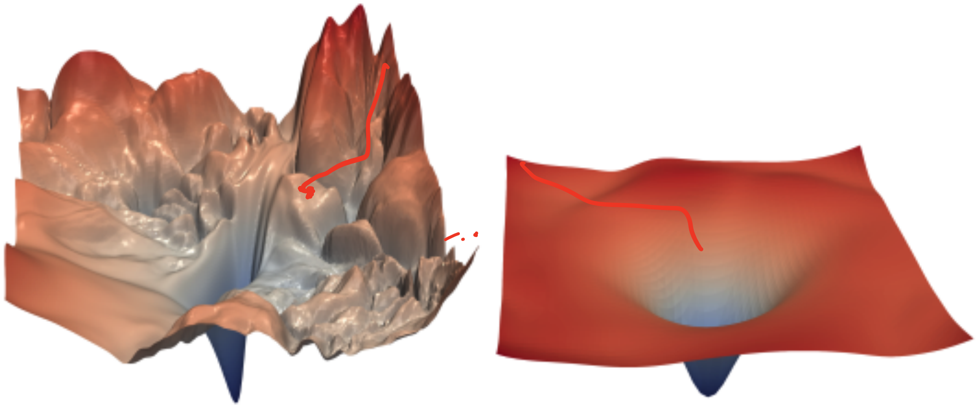
- Now able to train over 100 layers.
- One of the most important network design choices in the past decade.
- Prevalent in almost all network architectures, including Transformers.

[7] 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 [7].

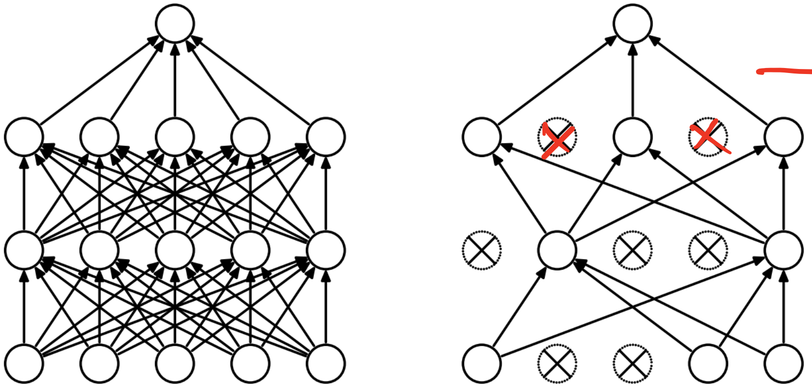


(a) without skip connections (b) with skip connections

[7] Li et al. Visualizing the Loss Landscape of Neural Nets. NIPS 2018.

Dropout[8]

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



Transformer.

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

training.

[8] Srivastava et al. A Simple Way to Prevent Neural Networks from Overfitting. JMLR, 2014.

Dropout[8]

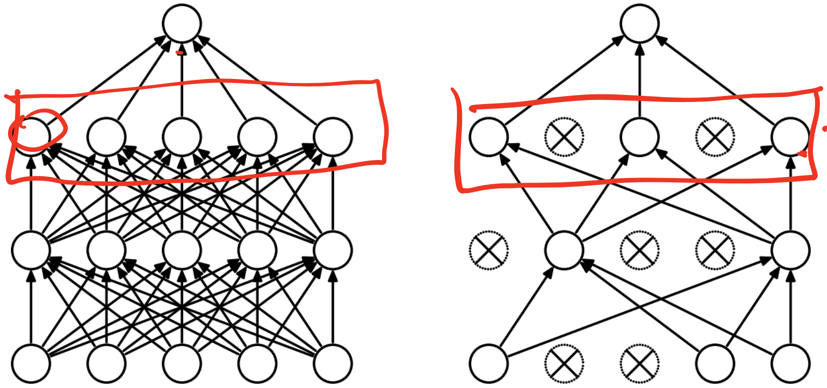
- Want to reduce overfitting in neural networks.
- Stochastically turning off neurons in propagation.
- Training to preserve redundancy.
- Test time: multiplying activations with probability. Model ensembling effect.

multiply.

$p \cdot z$

$p.$

during eval.



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

$p.$ turning on.

Bernoulli$(p) \cdot z$

Expected value same.

[8] Srivastava et al. A Simple Way to Prevent Neural Networks from Overfitting. JMLR, 2014.

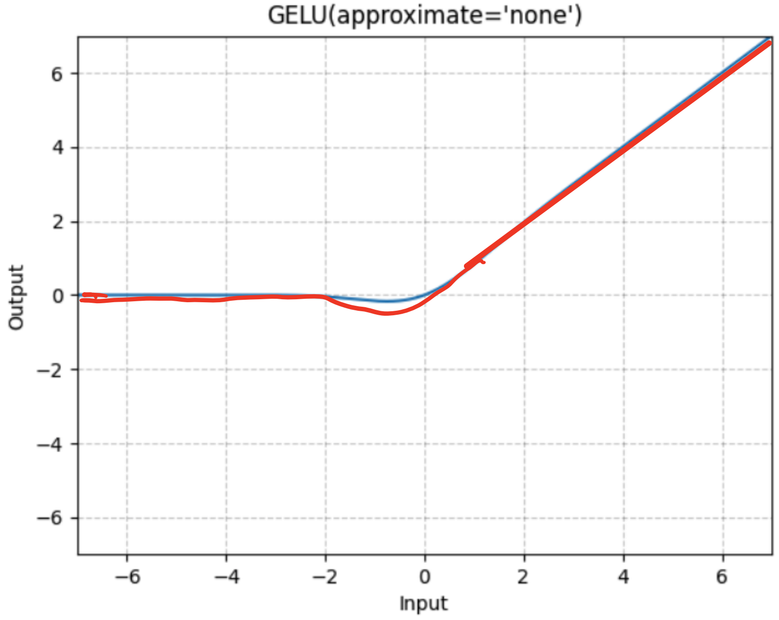
GELU[9]

500

ReLU.

dead unit.

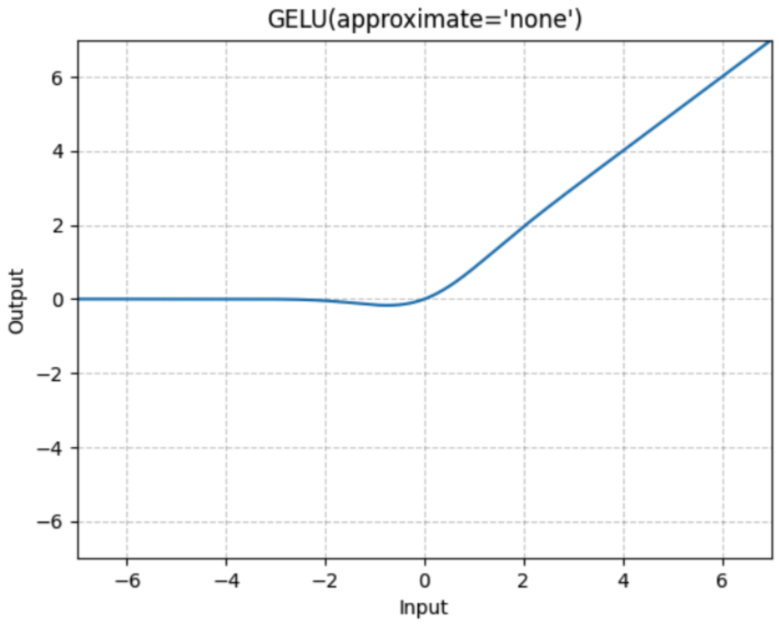
- Gaussian Error Linear Unit - A smoother activation function.
- Motivated by Dropout.



[9] Hendrycks & Gimpel. Gaussian Error Linear Unit (GELU). CoRR abs/1606.08415, 2016.

GELU[9]

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



[9] Hendrycks & Gimpel. Gaussian Error Linear Unit (GELU). CoRR abs/1606.08415, 2016.

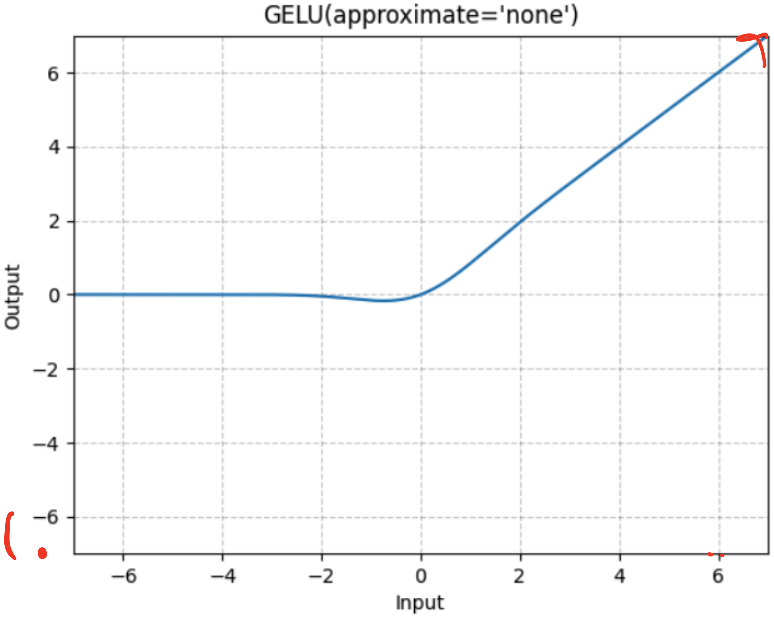
GELU[9]

Transformer.

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

$f(x) \approx x$. if x is big.

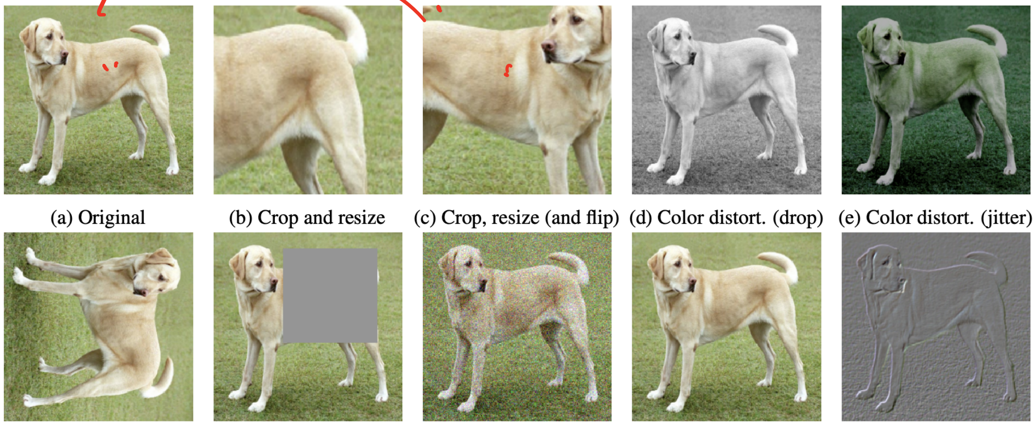
$f(x) \approx 0$ if x is small.



[9] 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



(a) Original (b) Crop and resize (c) Crop, resize (and flip) (d) Color distort. (drop) (e) Color distort. (jitter)

(f) Rotate $\{90°, 180°, 270°\}$ (g) Cutout (h) Gaussian noise (i) Gaussian blur (j) Sobel filtering

Image credit[10]

[10] 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

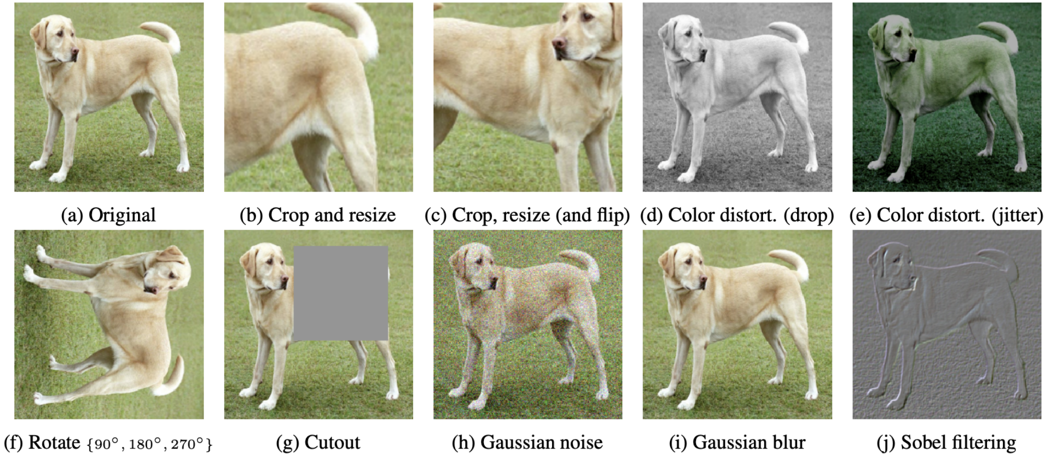


Image credit[10]

[10] 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

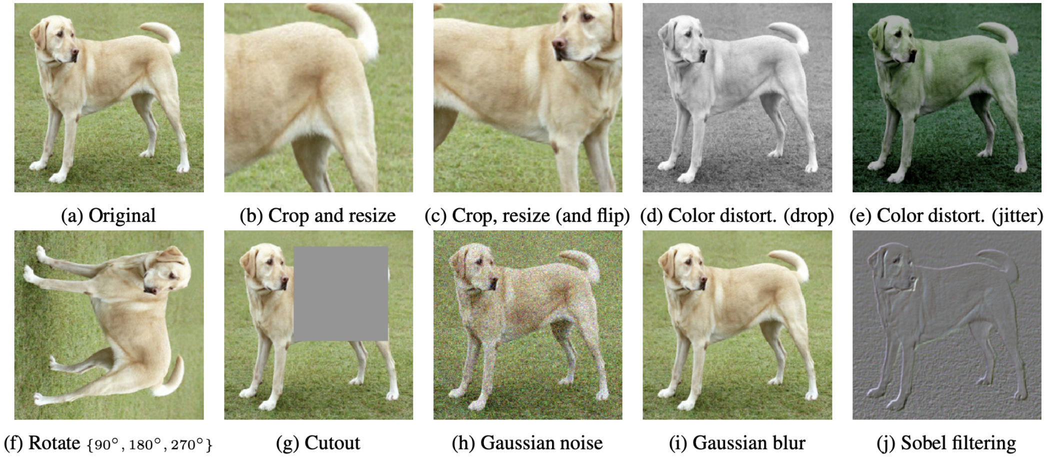


Image credit[10]

[10] 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

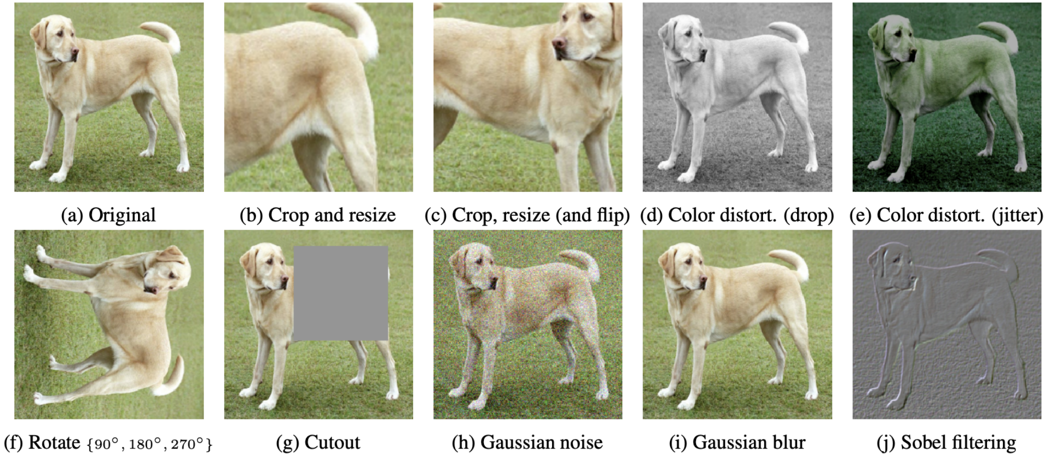


Image credit[10]

[10] 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

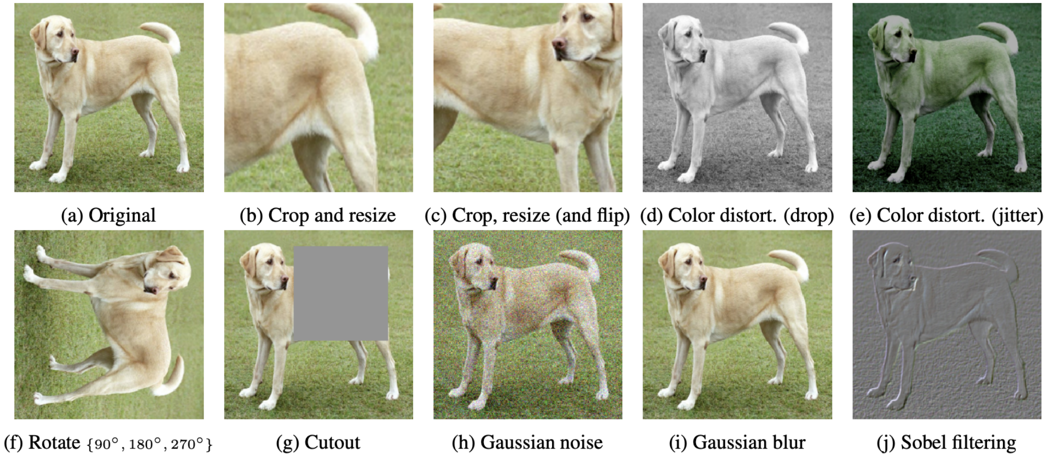
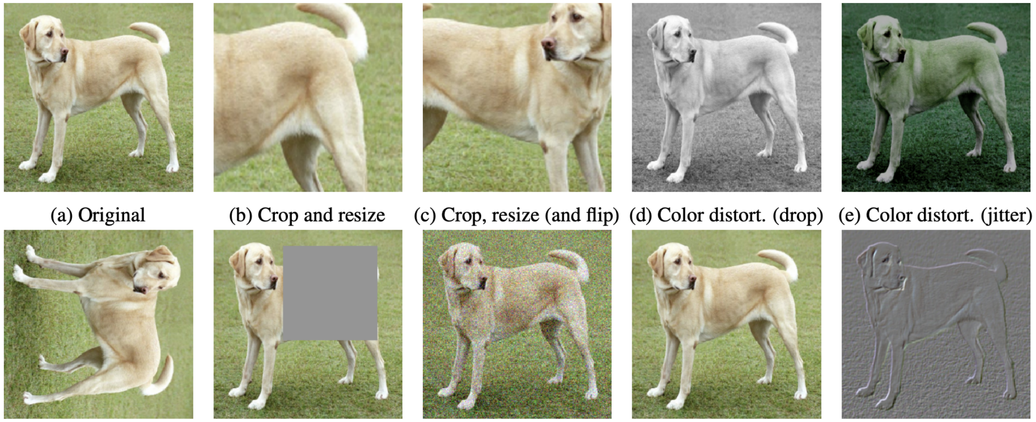


Image credit[10]

[10] 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.



(a) Original (b) Crop and resize (c) Crop, resize (and flip) (d) Color distort. (drop) (e) Color distort. (jitter)

(f) Rotate {90°, 180°, 270°} (g) Cutout (h) Gaussian noise (i) Gaussian blur (j) Sobel filtering

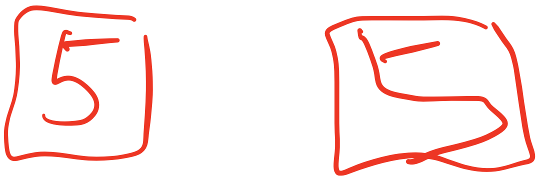


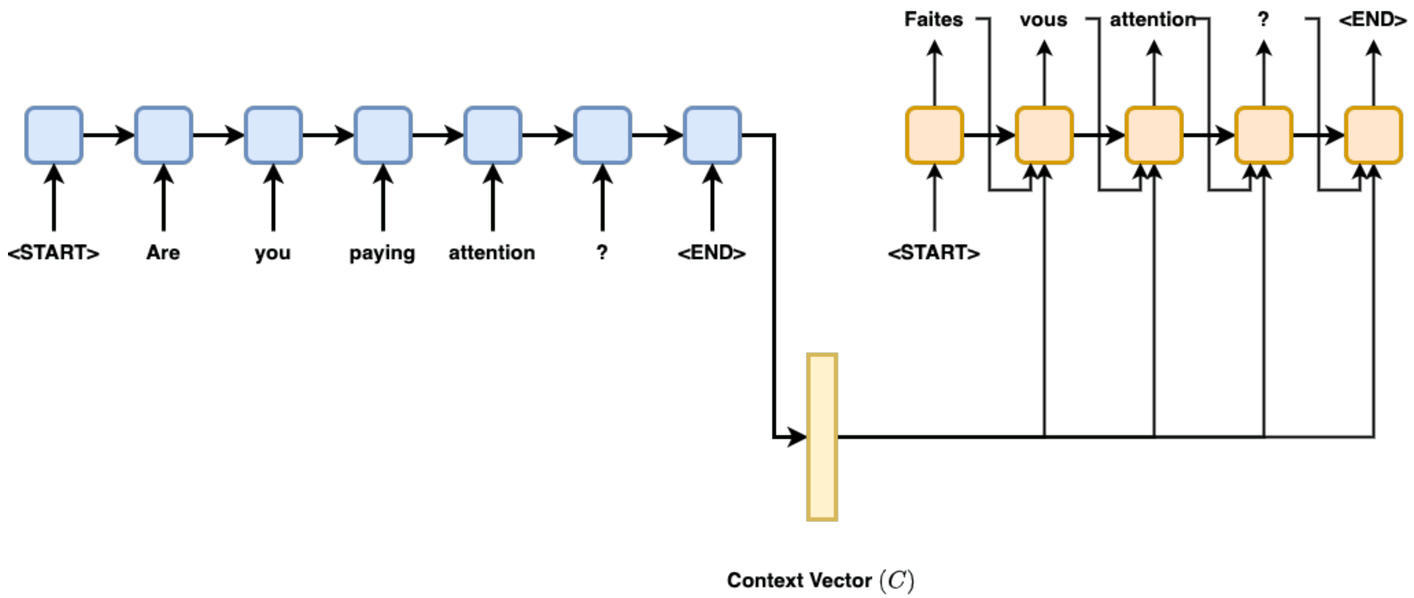
Image credit[10]

[10] Chen et al. A Simple Framework for Contrastive Learning of Visual Representations. ICML 2020.

Language and sequential signals

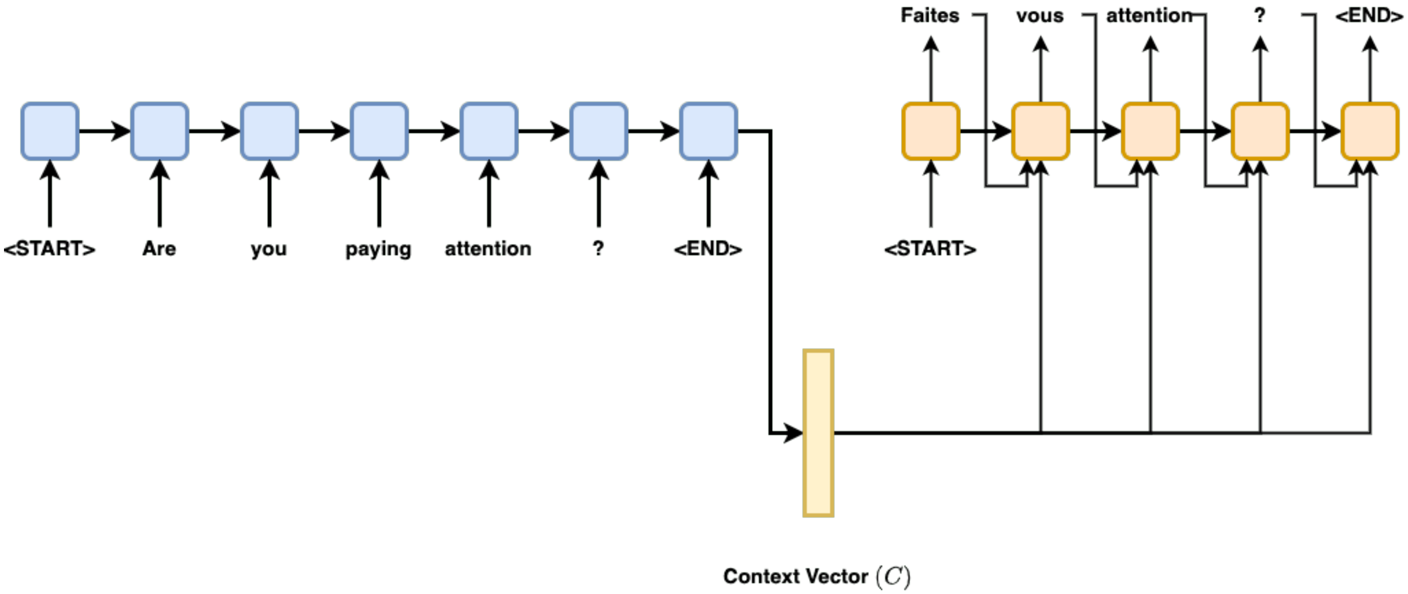
What about natural language

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



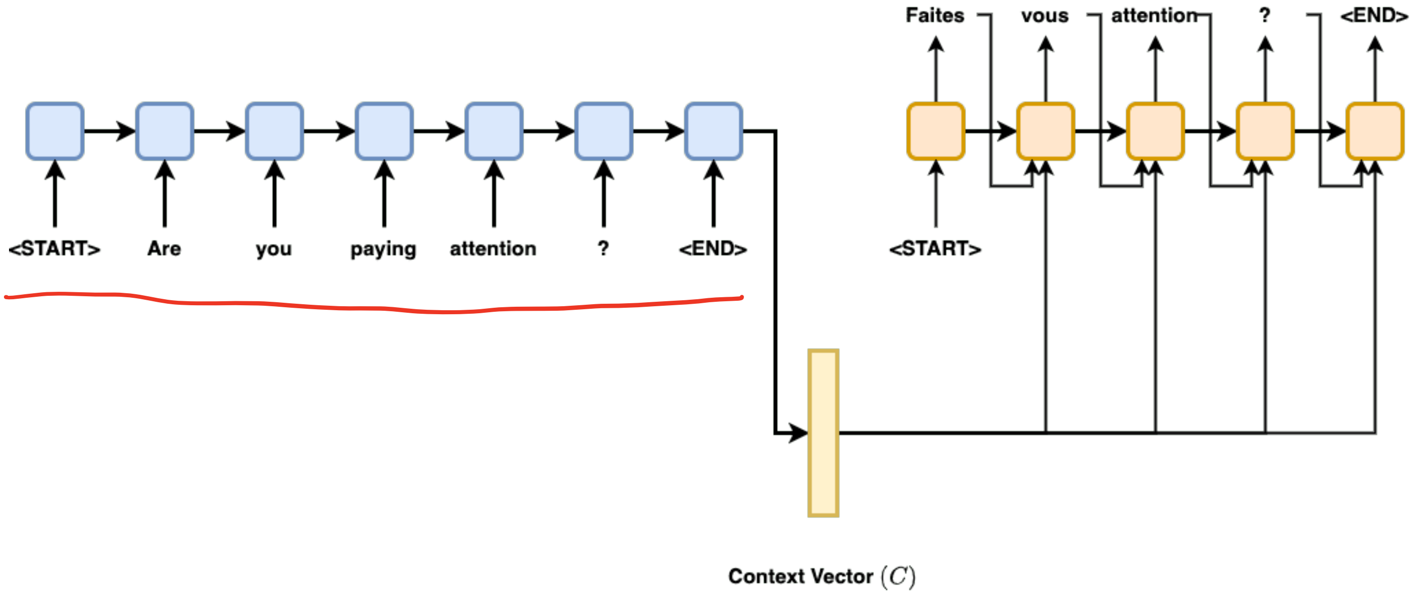
What about natural language

- 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 accomodate sequential signals such as language.



Word embeddings

0

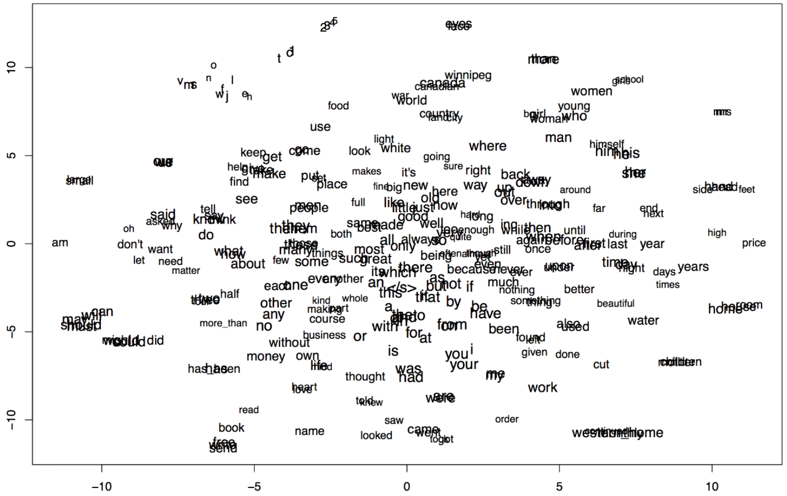
1

2.

- Neural networks are best dealing with real valued vectors.

token embedding.

vector .

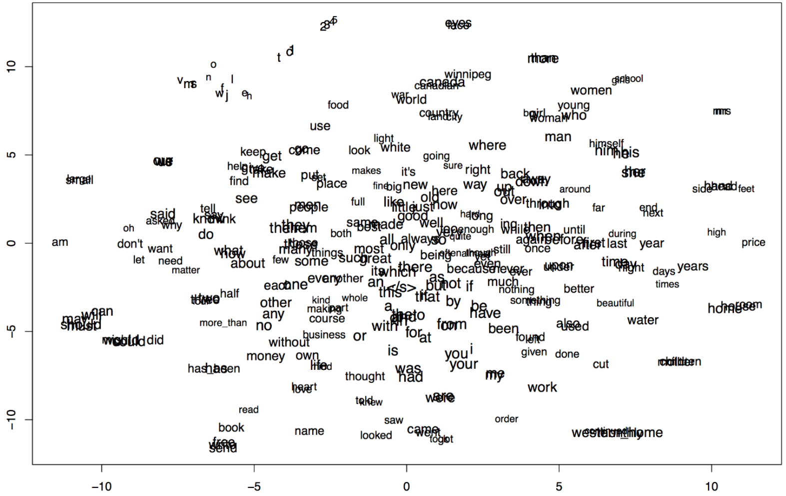


[11]

11 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).



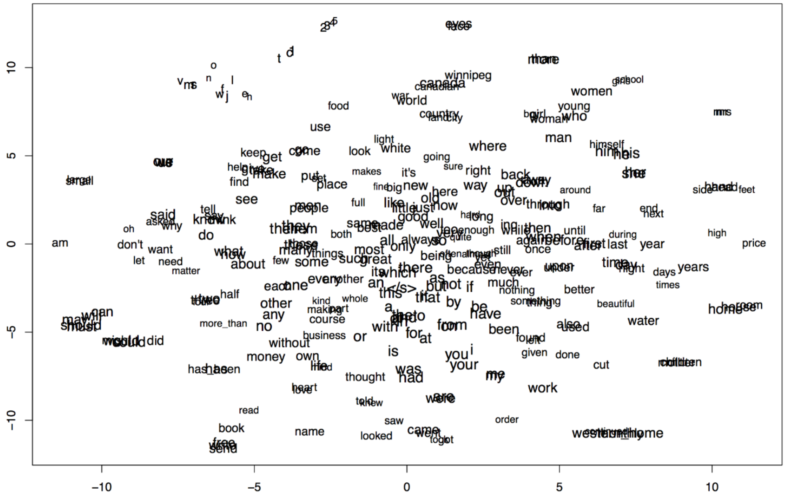
[11]

---

11 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.



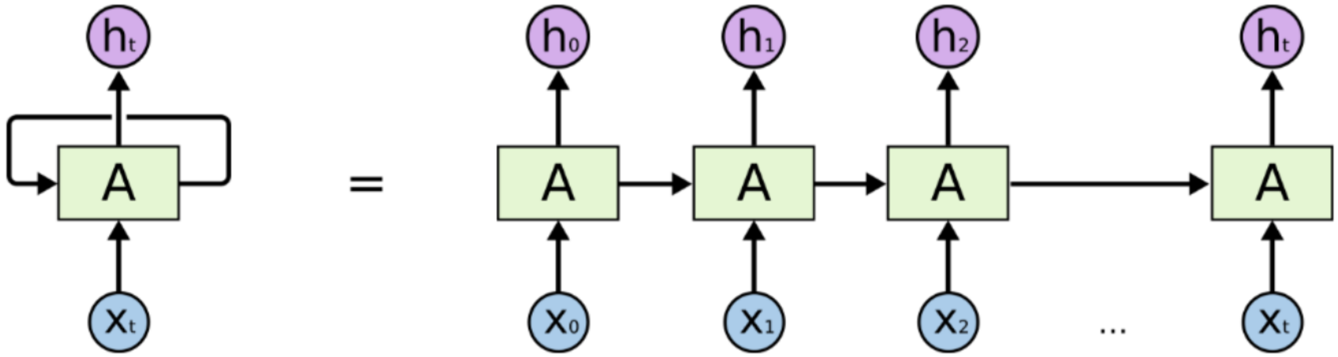
[11]

---

[11] https://aelang.github.io/word-embeddings.html

Convolutional vs. recurrent networks

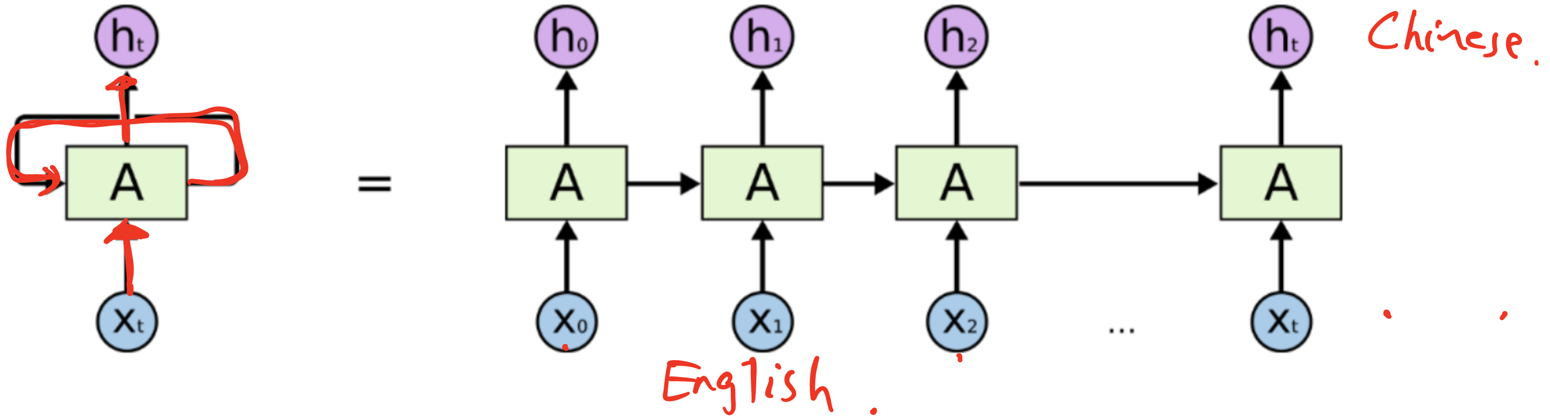
- Recall in images we used the convolution operation.
- We can also use the idea of convolution for temporal signals.



Convolutional vs. recurrent networks

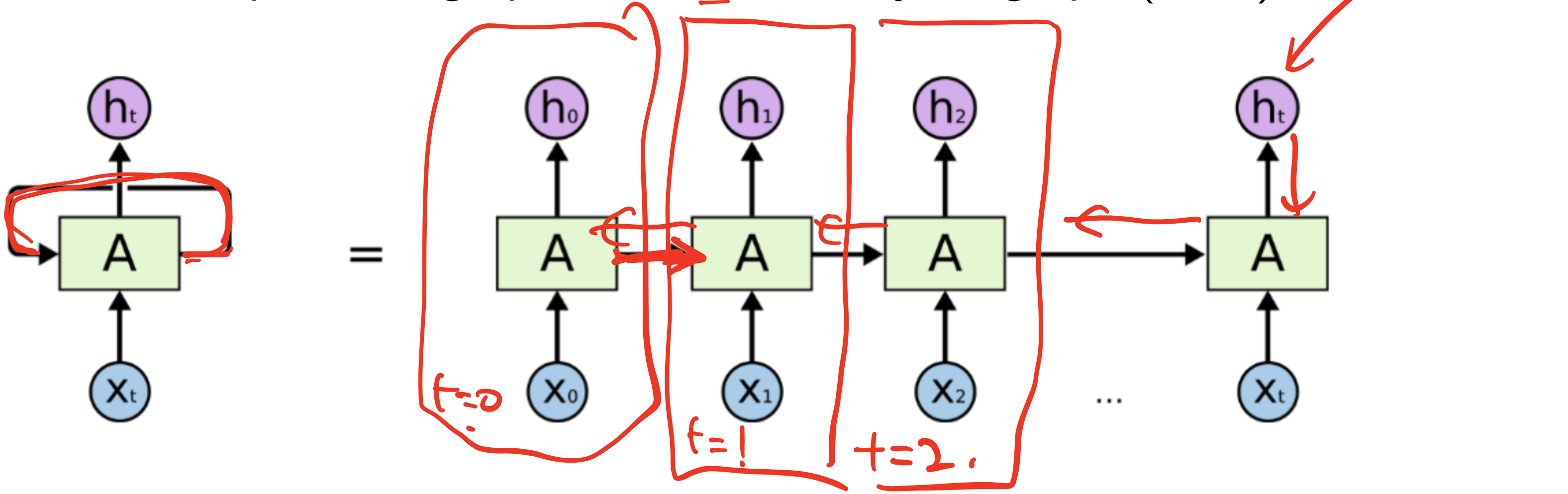
- Recall in images we used the convolution operation.
- We can also use the idea of convolution for temporal signals.
- Another alternative is to use a type of network called recurrent networks.
- Two inputs: x_t is the current input, and h_t is the historical hidden state. → RNN.

Weight sharing on temporal level.



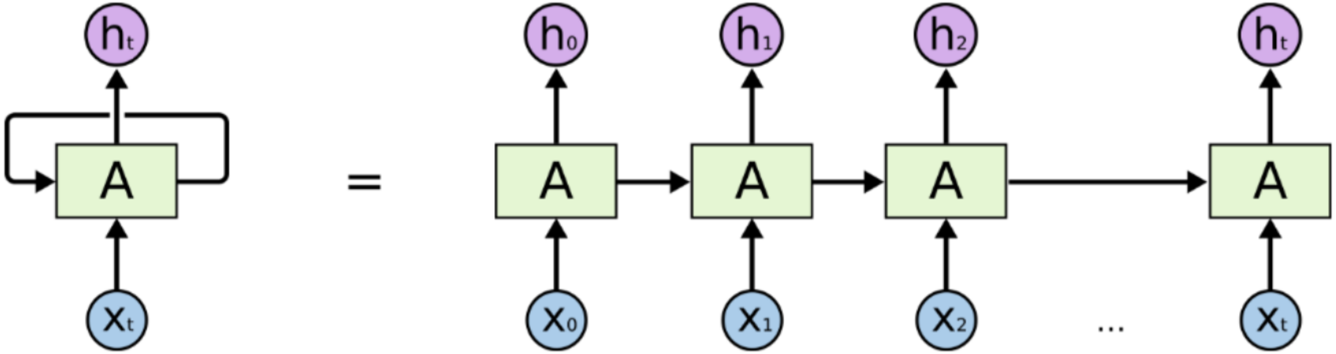
Convolutional vs. recurrent networks

- 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.



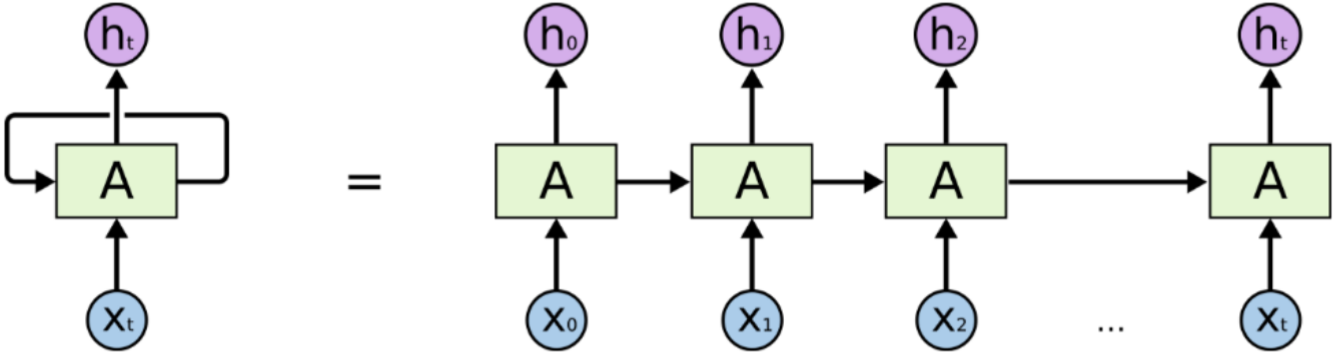
12

---

[12] Image credit: Chris Olah https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent neural networks (RNNs)

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

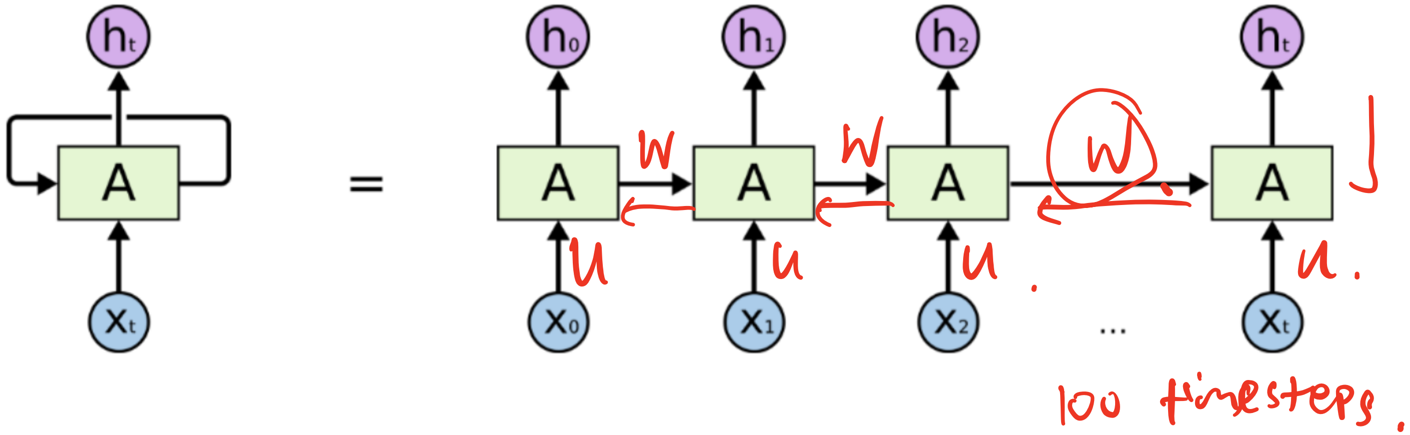


12

[12] Image credit: Chris Olah https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent neural networks (RNNs)

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

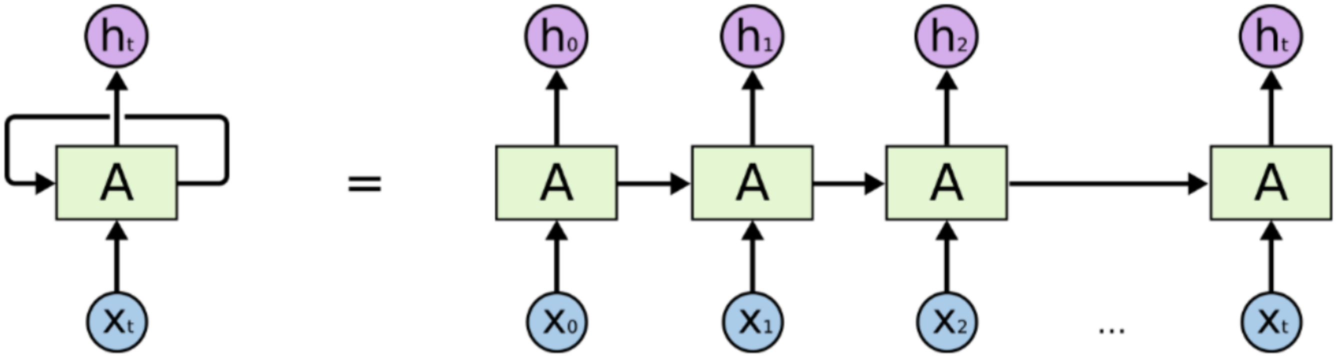


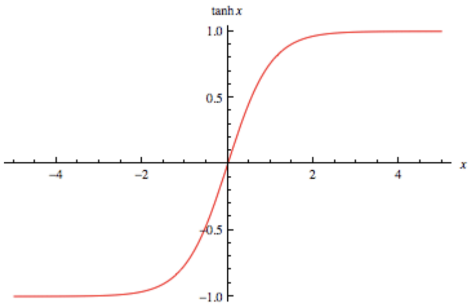
12

[12] Image credit: Chris Olah https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Gradient vanishing

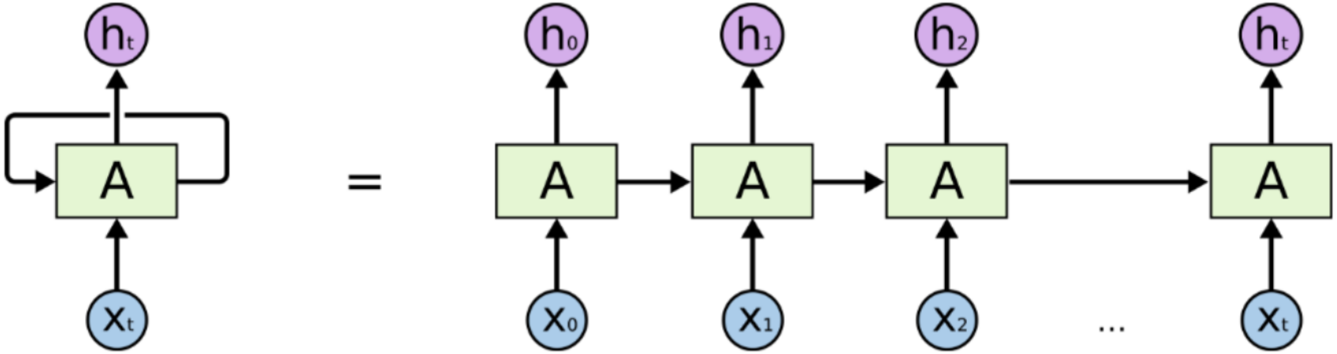
- Every iteration, we multiply the hidden state h_{t-1} from the previous iteration with the same W. Recall the definition of Jacobian.

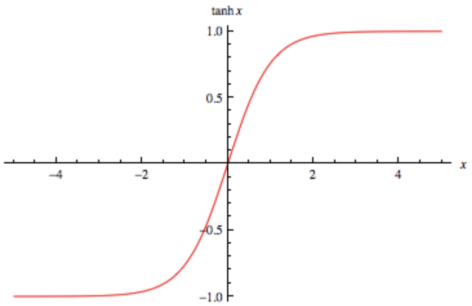




Gradient vanishing

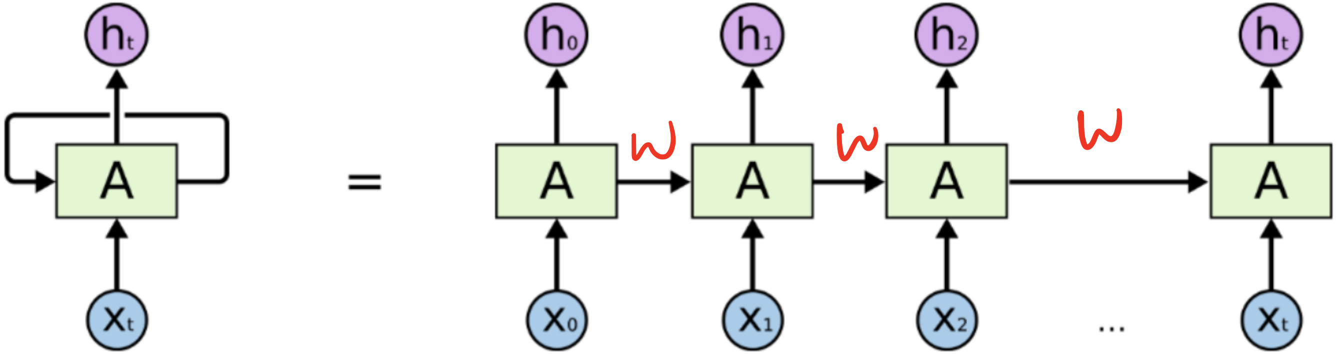
- Every iteration, we multiply the hidden state h_{t-1} from the previous iteration with the same W. Recall the definition of Jacobian.
- If the largest singular value of W is less than one then back-propagation will be attenuated.

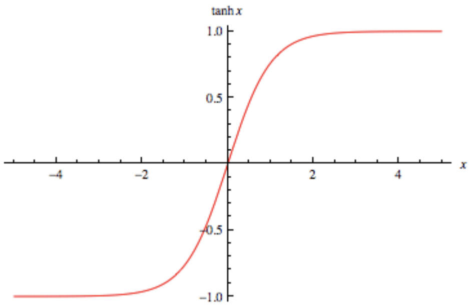




Gradient vanishing

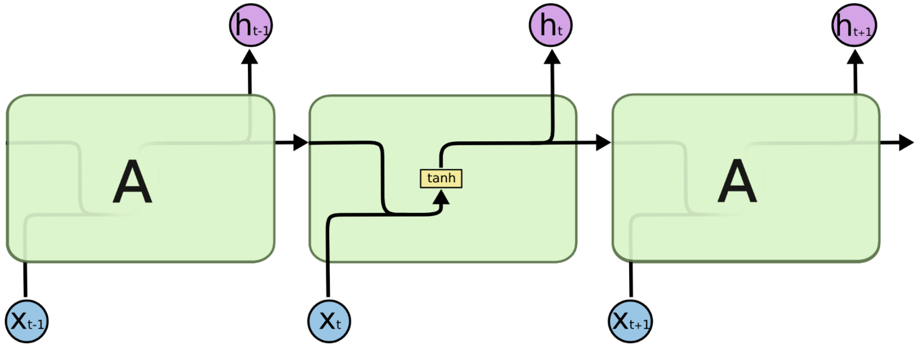
- Every iteration, we multiply the hidden state h_{t-1} from the previous iteration with the same W. Recall the definition of Jacobian.
- If the largest singular value of W is less than one then back-propagation will be attenuated.
- Similarly, we apply tanh activation every iteration – further reducing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

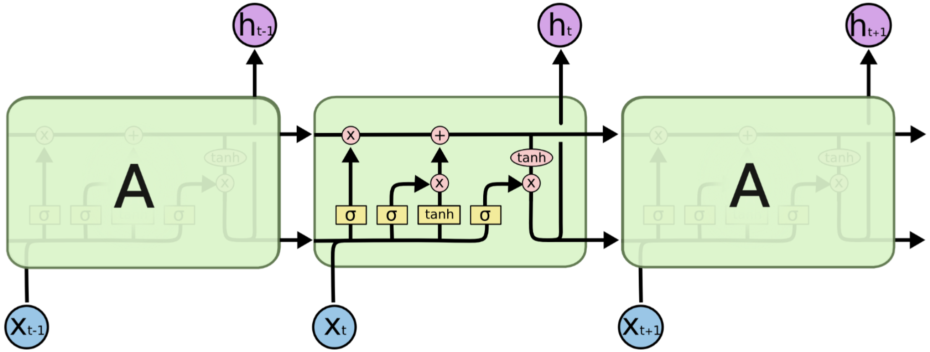




Gating functions in LSTM

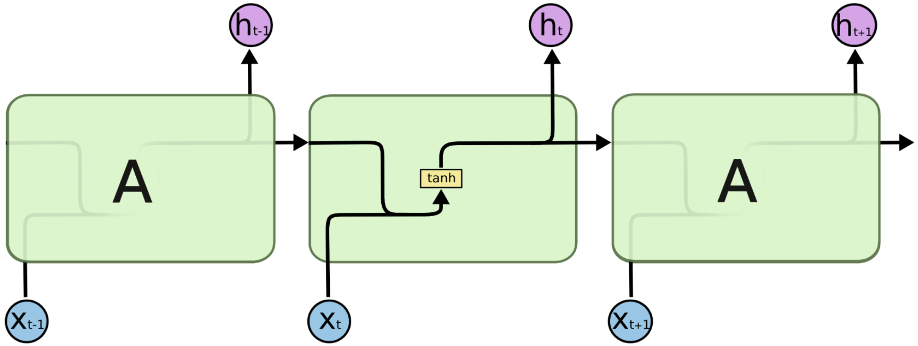
- Long short-term memory is a network that addresses the gradient vanishing problem by introducing gating functions.
- Gating functions provide "shortcuts", like ResNet.

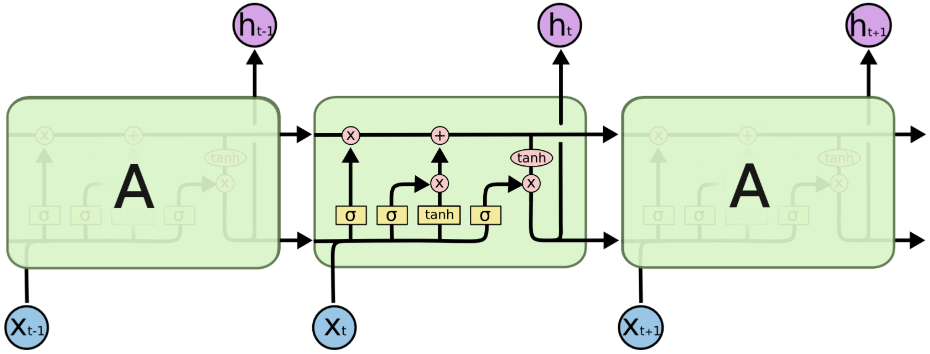




Gating functions in LSTM

- 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.



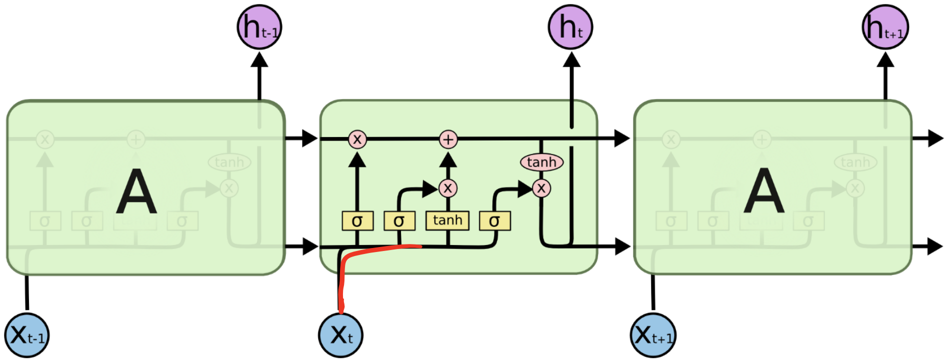


Gating functions in LSTM

information to be consumed

information get rid of.

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



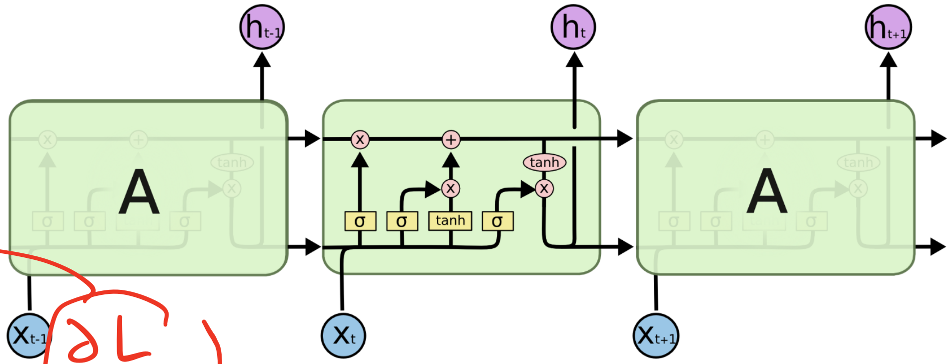
Gating functions in LSTM

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

$W c_{t-1}$

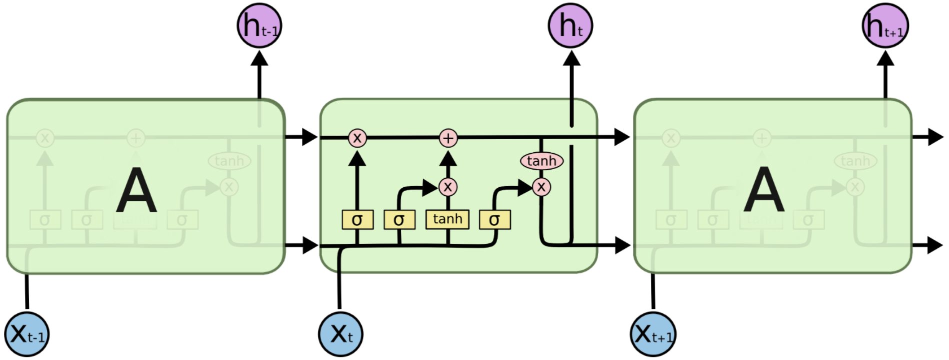
$$\frac{\partial L}{\partial c_{t-1}} = f_t \odot \frac{\partial L}{\partial c_t}.$$

$f_t \downarrow 0$



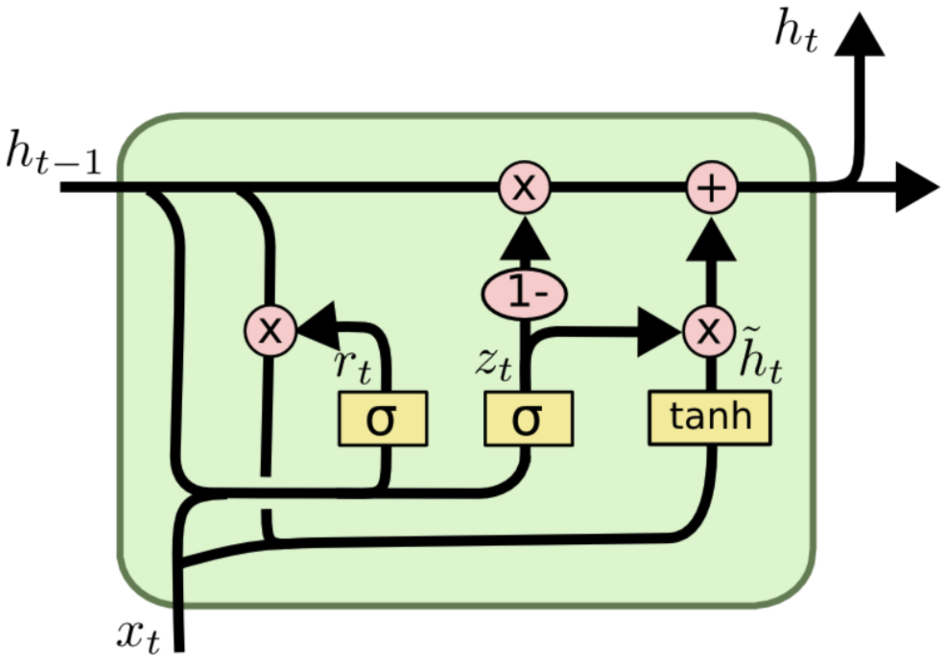
Gating functions in LSTM

- 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: $o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$.
- $h_t = o_t \odot \tanh(c_t)$.



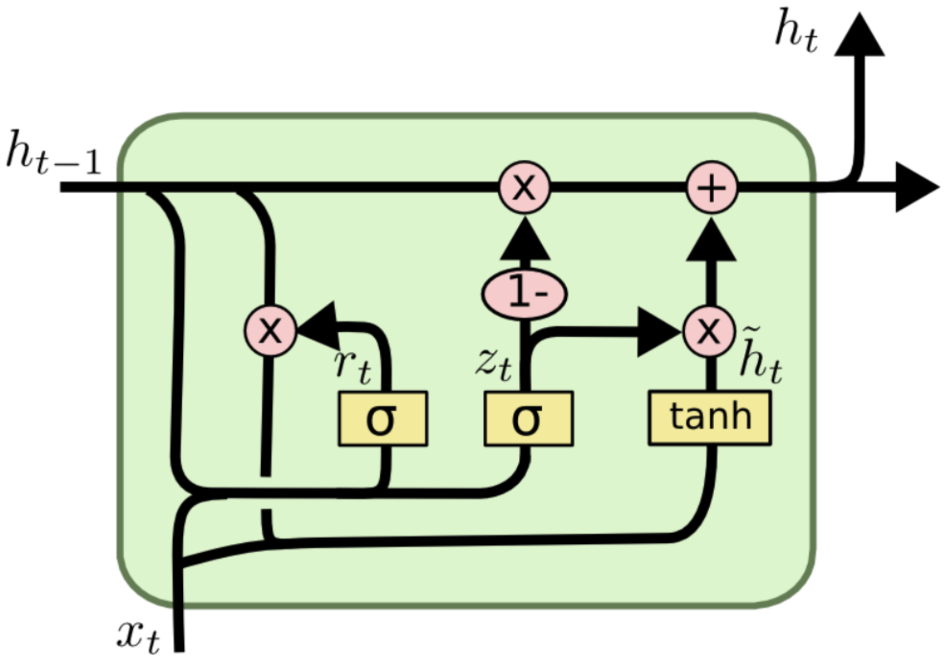
Gated Recurrent Unit

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



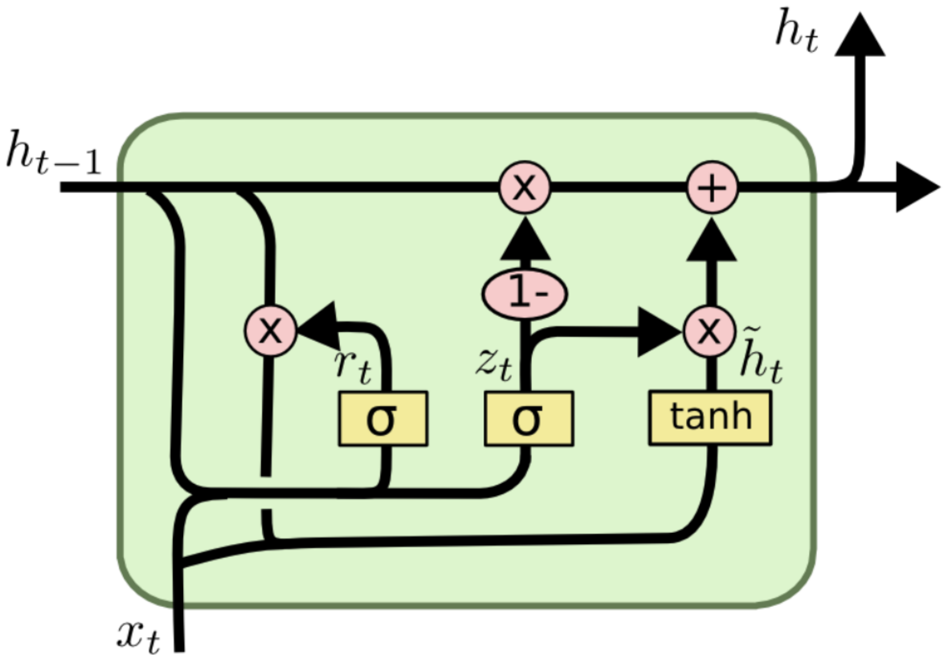
Gated Recurrent Unit

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



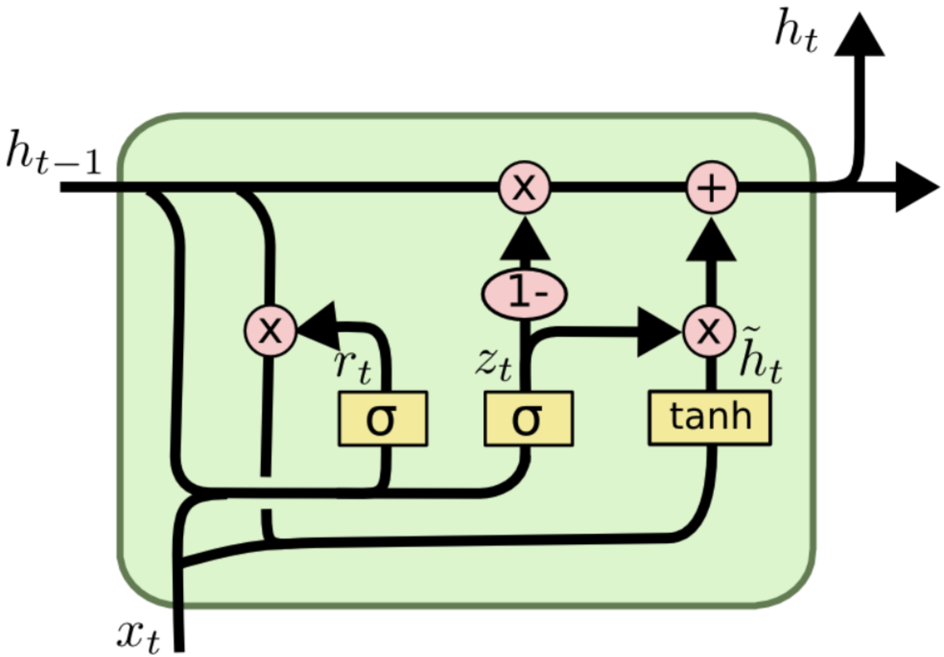
Gated Recurrent Unit

- 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

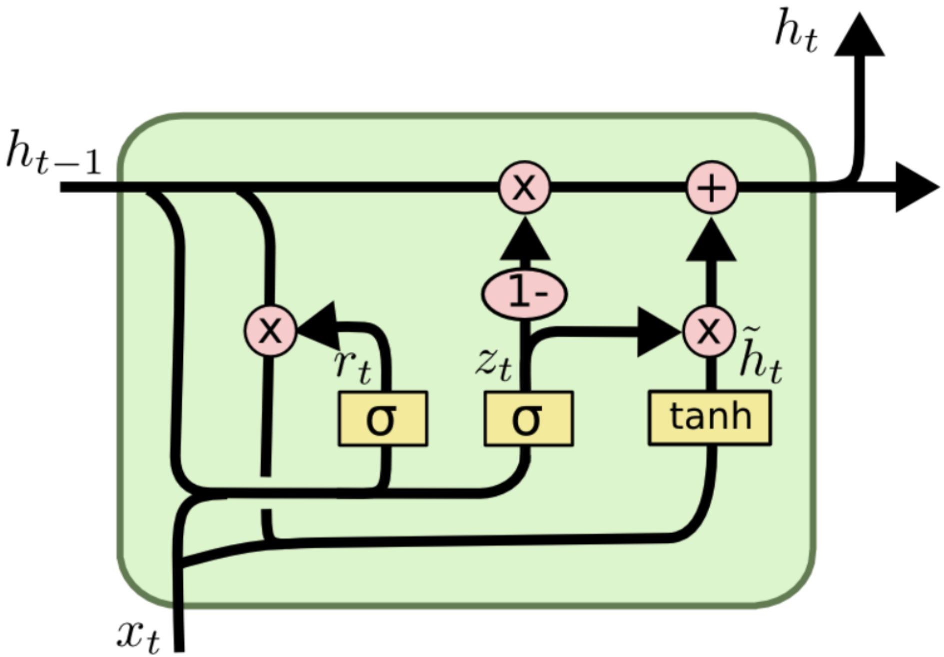
- Proposed by Chung et al. in 2015, a simplified variant compared to LSTM.
- Input gate $i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$.
- Reset gate $r_t = \sigma(W_r[h_{t-1}, x_t] + b_r)$.
- $\tilde{h}_t = \tanh(W_h[r_t \odot h_t, x_t] + b_h)$.



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 $r_t = \sigma(W_r[h_{t-1}, x_t] + b_r)$.
- $\tilde{h}_t = \tanh(W_h[r_t \odot h_t, x_t] + b_h)$.
- $h_t = \boxed{(1 - i_t)} \odot h_{t-1} + \boxed{i_t} \odot \tilde{h}_t$.

Current input if reset.



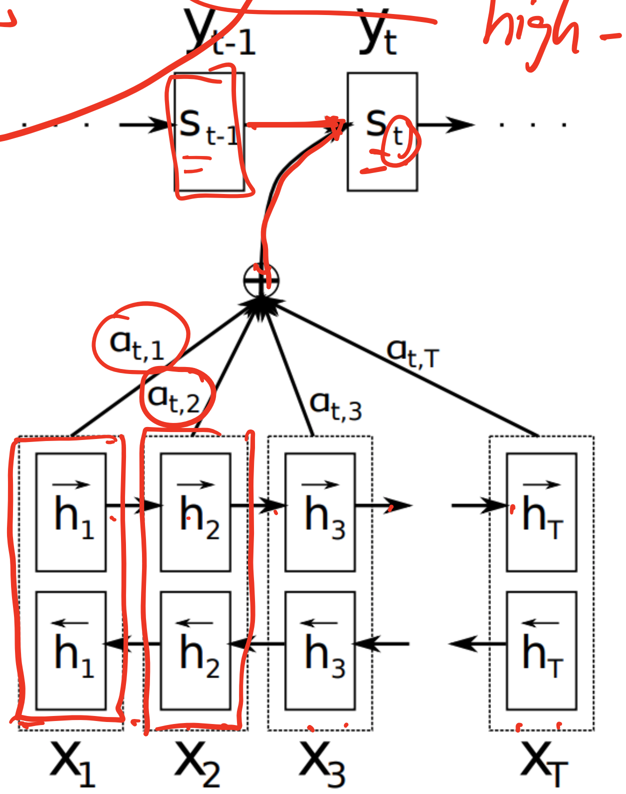
Attention Mechanisms

a b c d e f (eos)

a' b' c' d' e' f'

high-level

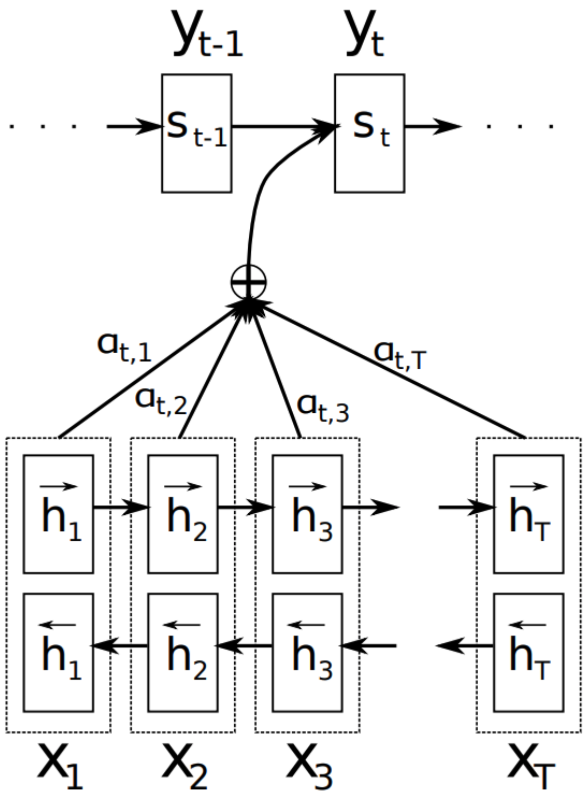
- Earlier content will decay more.
- Hard to refer back to the raw content.
- Reverse order better than forward order [abcde -> a'b'c'd'e' vs. abcde -> e'd'c'b'a'].



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.



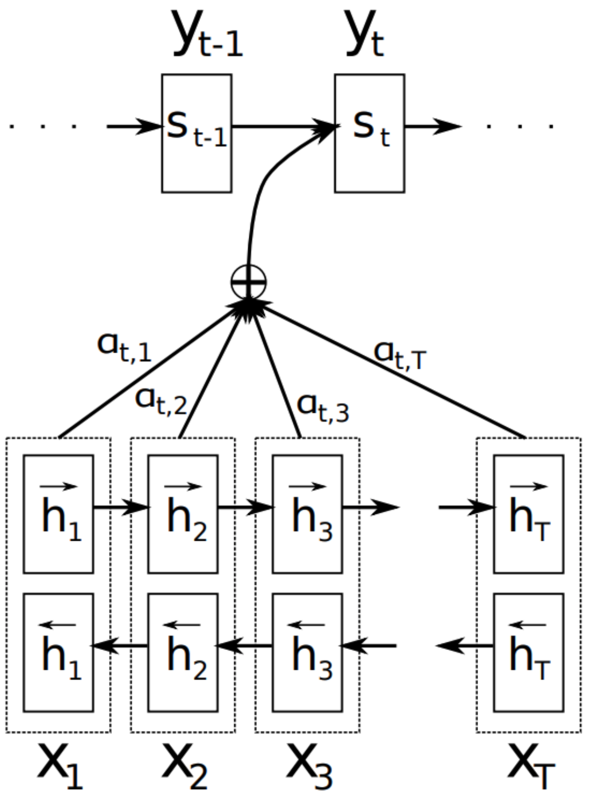
Bahdanau et al., 2014

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.
- $s_t = f(s_{t-1}, y_{t-1}, c_t)$

rnn output.

Content attends to.



Bahdanau et al., 2014

Attention Mechanisms

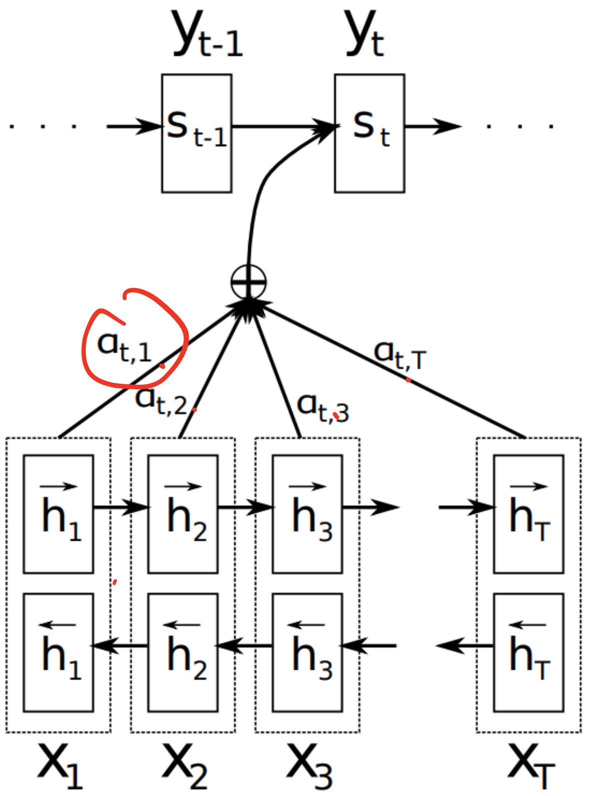
- Earlier content will decay more.
- Hard to refer back to the raw content.
- Reverse order better than forward order [abcde -> a'b'c'd'e' vs. abcde -> e'd'c'b'a'].
- Attending to arbitrary sequence tokens.
- $s_t = f(s_{t-1}, y_{t-1}, c_t)$
- $c_t = \sum_\tau \alpha_{t,\tau} h_\tau$, $\alpha_{t,\tau}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

Softmax attention.

input Content.

[0, 1, 0, 0]

[0.5, 0.5, 0 ?]



Bahdanau et al., 2014

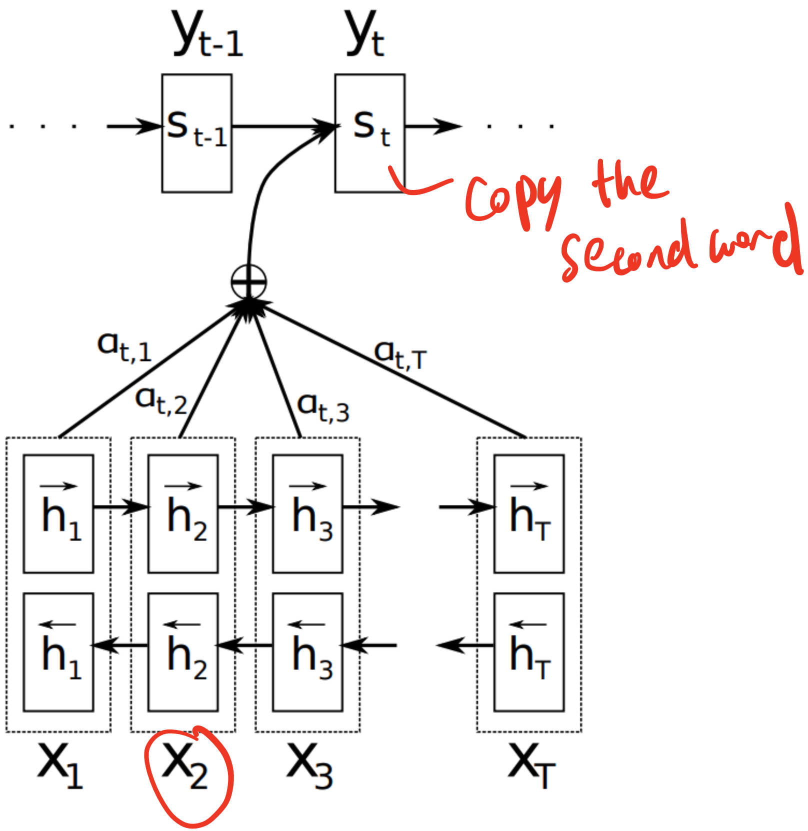
Attention Mechanisms

- 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.
- $s_t = f(s_{t-1}, y_{t-1}, c_t)$

raw attention

- $c_t = \sum_\tau \alpha_{t,\tau} h_\tau, \; \alpha_{t,\tau} = \frac{\exp(a(s_{t-1}, h_k))}{\sum_k \exp(a(s_{t-1}, h_k))}$
- $a(s_{t-1}, h_k) = v_a^\top \tanh(W_a[s_{i-1}, h_k])$ → scalar.

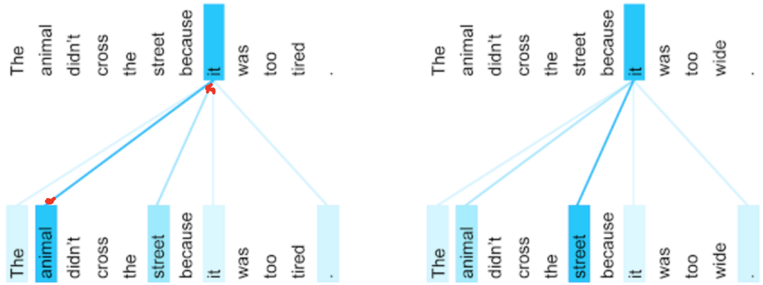
compatible.



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.

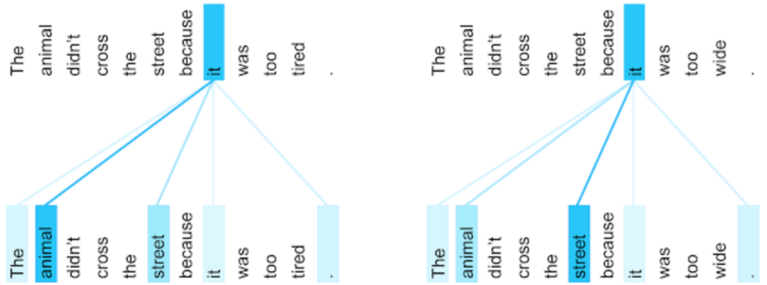


13

13 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.

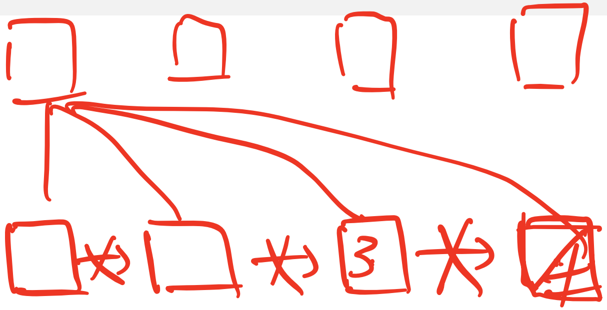


13

[13] Image credit: Google Research Blog

Positional encoding

- Attention operation is permuation equivariant.



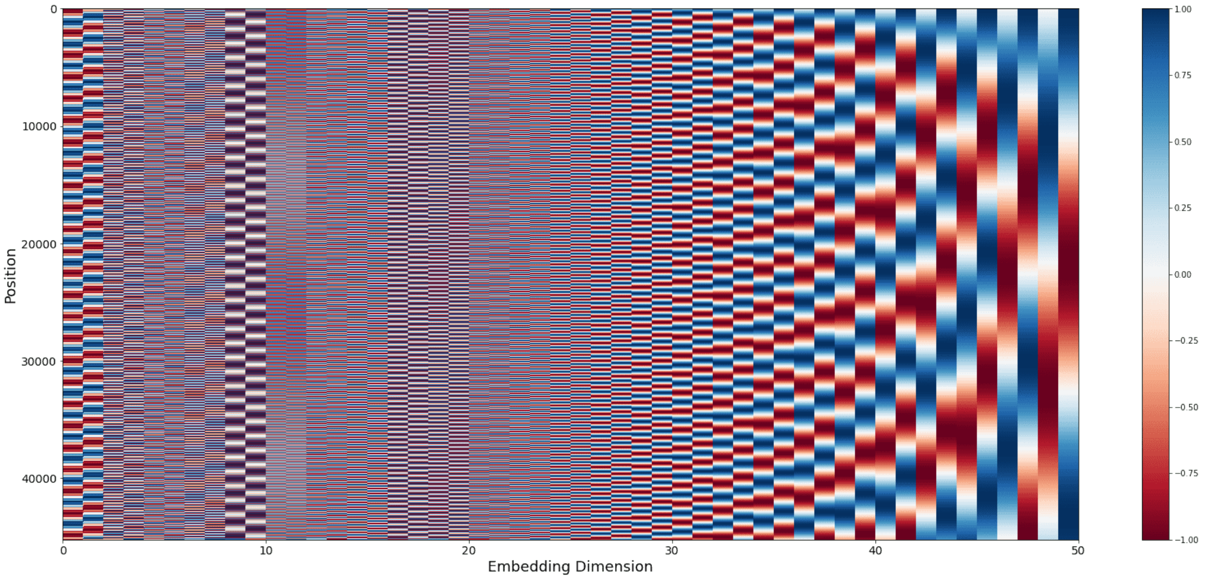
Positional encoding

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

Positional encoding

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

pos.



fixed dimensional encoding.