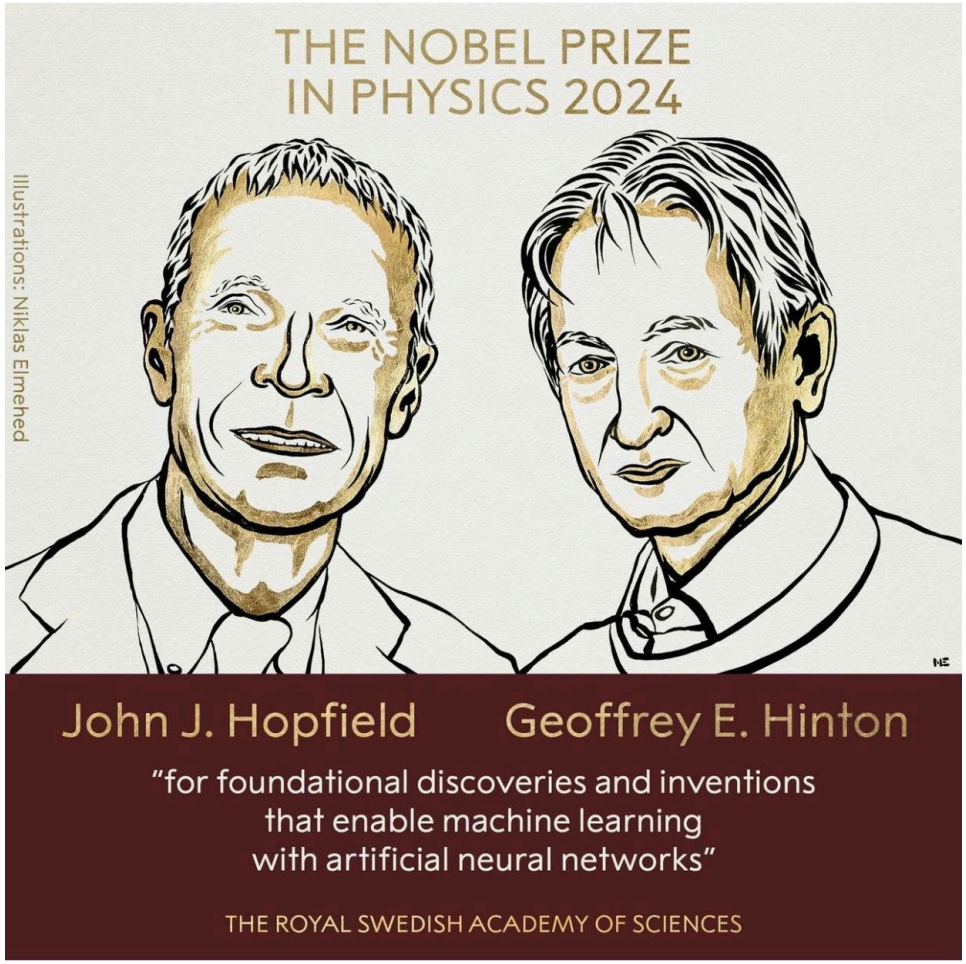
Convolution

CISC 7026: Introduction to Deep Learning

University of Macau



John Hopfield and Geoffrey Hinton used tools from physics to construct methods that helped lay the foundation for today's powerful machine learning. Machine learning based on artificial neural networks is currently revolutionising science, engineering and daily life.

John Hopfield - Hopfield networks, an older type of neural network

John Hopfield - Hopfield networks, an older type of neural network

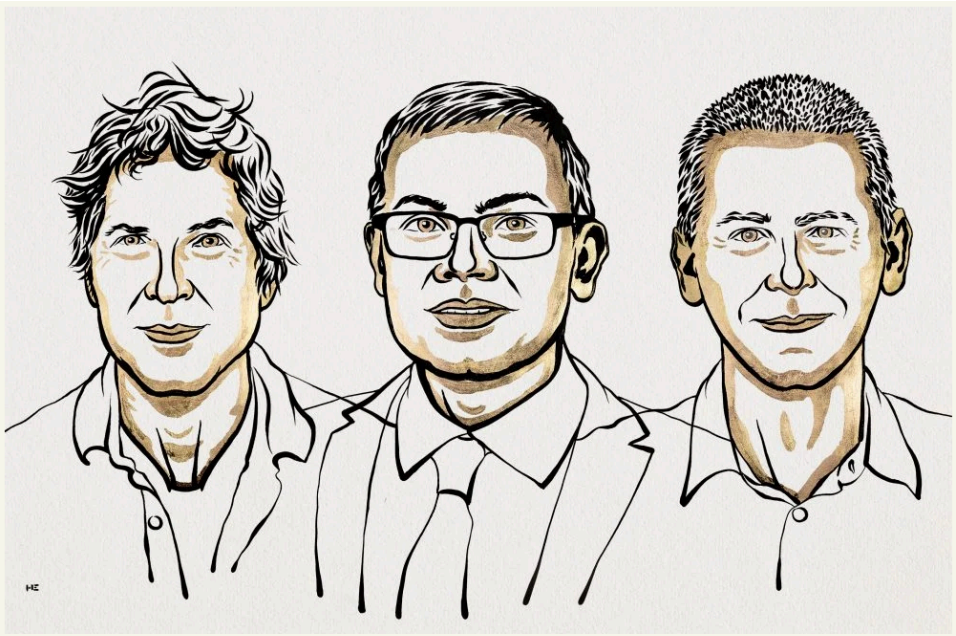
Geoffrey Hinton - Most well-known for backpropagation

John Hopfield - Hopfield networks, an older type of neural network

Geoffrey Hinton - Most well-known for backpropagation

Geoffrey Hinton's student also created AlexNet, which we will discuss today

https://www.youtube.com/watch?v=qrvK_KuIeJk



David Baker, Demis Hassabis and John Jumper. Ill. Niklas Elmehed © Nobel Prize Outreach

Demis Hassabis and John Jumper have successfully utilised artificial intelligence to predict the structure of almost all known proteins. David Baker has learned how to master life's building blocks and create entirely new proteins.

https://www.youtube.com/watch?v=gg7WjuFs8F4

https://www.youtube.com/watch?v=gg7WjuFs8F4

Alphafold solves a regression problem

https://www.youtube.com/watch?v=gg7WjuFs8F4

Alphafold solves a regression problem

$$X : \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\}^k$$

Sequence of amino acids

https://www.youtube.com/watch?v=gg7WjuFs8F4

Alphafold solves a regression problem

$$X : \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\}^k$$

Sequence of amino acids

$$Y : \mathbb{R}^{j \times 3}$$

3D coordinates (x, y, z) of each atom in the protein

https://www.youtube.com/watch?v=gg7WjuFs8F4

Alphafold solves a regression problem

$$X : \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\}^k$$

Sequence of amino acids

$$Y : \mathbb{R}^{j \times 3}$$

3D coordinates (x, y, z) of each atom in the protein

$$f : X \times \Theta \mapsto Y$$

f is a neural network known as a transformer

Hinton, Hassabis, and Jumper attended Cambridge!

Plan for makeup lecture Saturday October 26

Plan for makeup lecture Saturday October 26

Afternoon? Evening?

Plan for makeup lecture Saturday October 26

Afternoon? Evening?

I will teach Autoencoders and Generative Models

Plan for makeup lecture Saturday October 26

Afternoon? Evening?

I will teach Autoencoders and Generative Models

Prof. Qingbiao Li will teach you Graph Neural Networks on Monday October 28

Agenda

1. Review
2. Signal Processing
3. Convolution
4. Convolutional Neural Networks
5. Additional Dimensions
6. Deeper Networks
7. Coding

Agenda

1. **Review**
2. Signal Processing
3. Convolution
4. Convolutional Neural Networks
5. Additional Dimensions
6. Deeper Networks
7. Coding

Dirty secret of deep learning

Review

Dirty secret of deep learning

As networks become larger, deep learning changes

Review

Dirty secret of deep learning

As networks become larger, deep learning changes

We understand the rules that govern small networks (physics)

Review

Dirty secret of deep learning

As networks become larger, deep learning changes

We understand the rules that govern small networks (physics)

One the system becomes sufficiently complex, we lose understanding (biology)

Review

Dirty secret of deep learning

As networks become larger, deep learning changes

We understand the rules that govern small networks (physics)

One the system becomes sufficiently complex, we lose understanding (biology)

Modern deep learning is a science, where we advance through trial and error

Review

To improve neural networks, we looked at:

Review

To improve neural networks, we looked at:

- Deeper and wider networks

Review

To improve neural networks, we looked at:

- Deeper and wider networks
- New activation functions

Review

To improve neural networks, we looked at:

- Deeper and wider networks
- New activation functions
- Parameter initialization

Review

To improve neural networks, we looked at:

- Deeper and wider networks
- New activation functions
- Parameter initialization
- New optimization methods

Review

A 2-layer neural network can represent **any** continuous function to arbitrary precision

Review

A 2-layer neural network can represent **any** continuous function to arbitrary precision

$$\mid f(\boldsymbol{x}, \boldsymbol{\theta}) - g(\boldsymbol{x}) \mid < \varepsilon$$

Review

A 2-layer neural network can represent **any** continuous function to arbitrary precision

$$\mid f(\boldsymbol{x}, \boldsymbol{\theta}) - g(\boldsymbol{x}) \mid < \varepsilon$$

$$\lim_{d_h \to \infty} \varepsilon = 0$$

Review

A 2-layer neural network can represent **any** continuous function to arbitrary precision

$$\mid f(\boldsymbol{x}, \boldsymbol{\theta}) - g(\boldsymbol{x}) \mid < \varepsilon$$

$$\lim_{d_h \to \infty} \varepsilon = 0$$

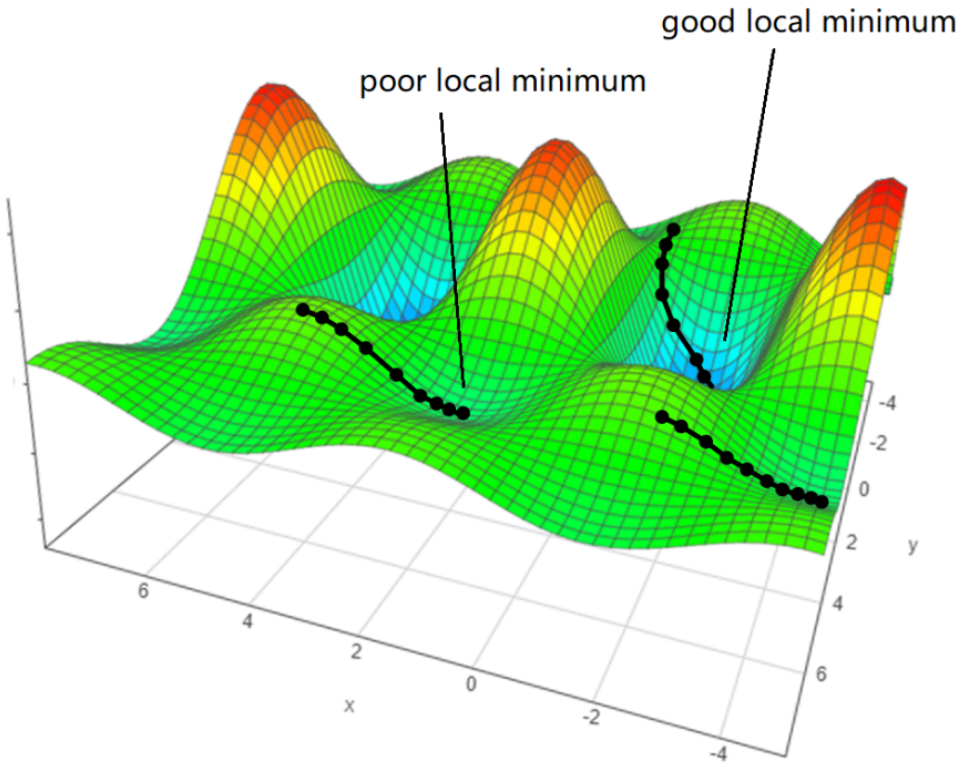
However, finding such $\boldsymbol{\theta}$ is a much harder problem

Review

Gradient descent only guarantees convergence to a **local** optima

Review

Gradient descent only guarantees convergence to a **local** optima



Review

Deeper and wider networks often perform better

Review

Deeper and wider networks often perform better

We call these **overparameterized** networks

Review

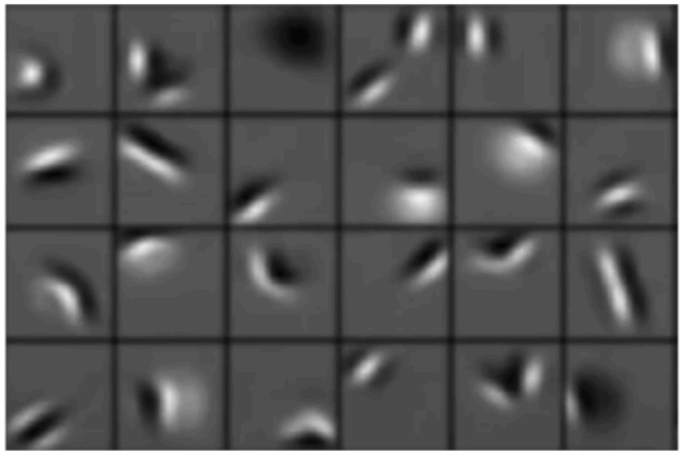
Deeper and wider networks often perform better

We call these **overparameterized** networks

"The probability of finding a "bad" (high value) local minimum is non-zero for small-size networks and decreases quickly with network size" - Choromanska, Anna, et al. *The loss surfaces of multilayer networks.* (2014)

Review

Low-level features



Edges, dark spots

Mid-level features

Eyes, ears, nose

High-level features

Facial structure

Review

```
import torch
d_x, d_y, d_h = 1, 1, 256
net = torch.nn.Sequential(
    torch.nn.Linear(d_x, d_h),
    torch.nn.Sigmoid(),
    torch.nn.Linear(d_h, d_h),
    torch.nn.Sigmoid(),
    ...
    torch.nn.Linear(d_h, d_y),
)

x = torch.ones((d_x,))
y = net(x)
```

Review

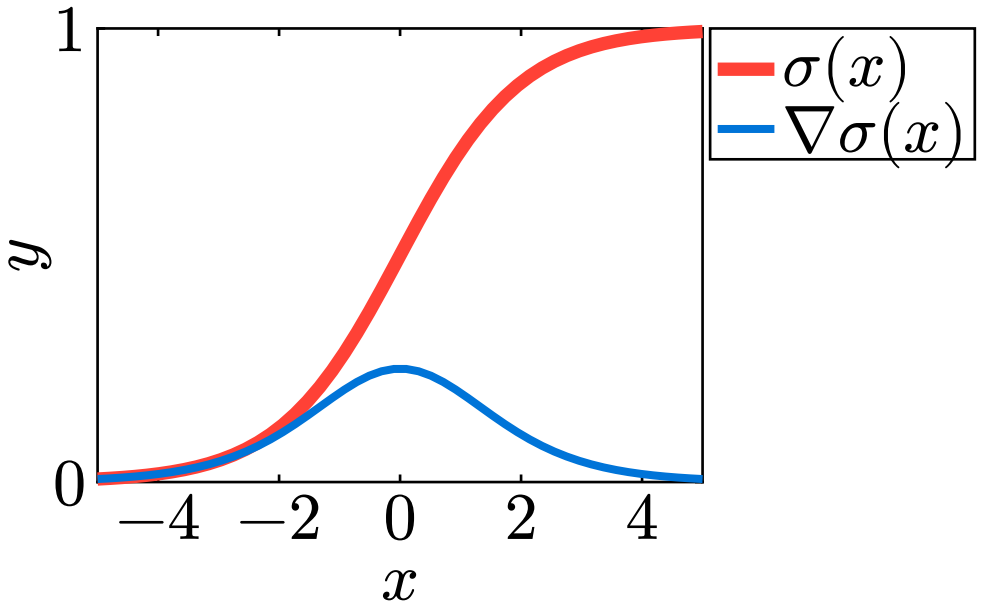
```
import jax, equinox
d_x, d_y, d_h = 1, 1, 256
net = equinox.nn.Sequential([
    equinox.nn.Linear(d_x, d_h),
    equinox.nn.Lambda(jax.nn.sigmoid),
    equinox.nn.Linear(d_h, d_h),
    equinox.nn.Lambda(jax.nn.sigmoid),
    ...
    equinox.nn.Linear(d_h, d_y),
])

x = jax.numpy.ones((d_x,))
y = net(x)
```

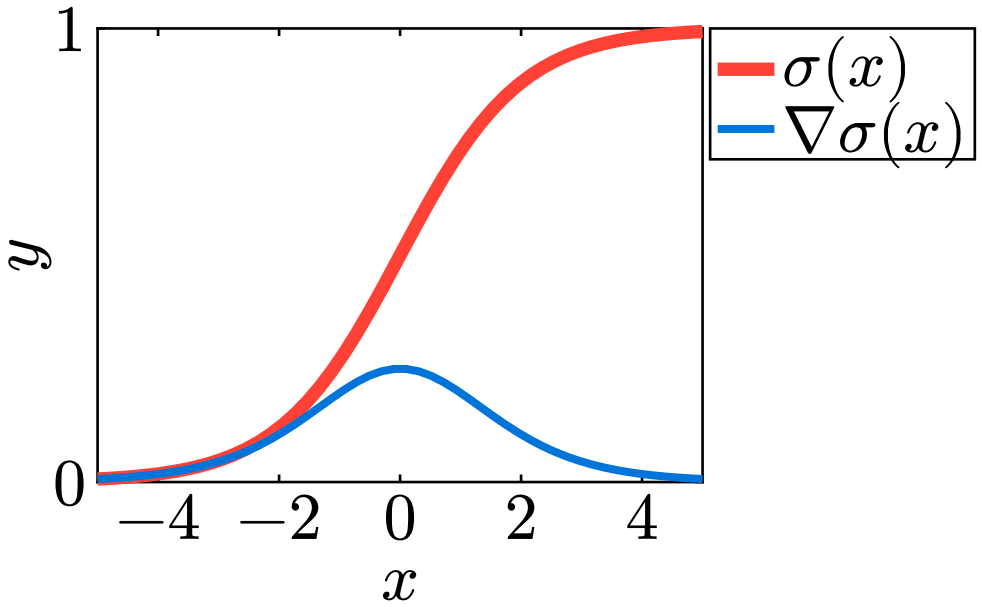
Review

Next, we looked at activation functions

Review

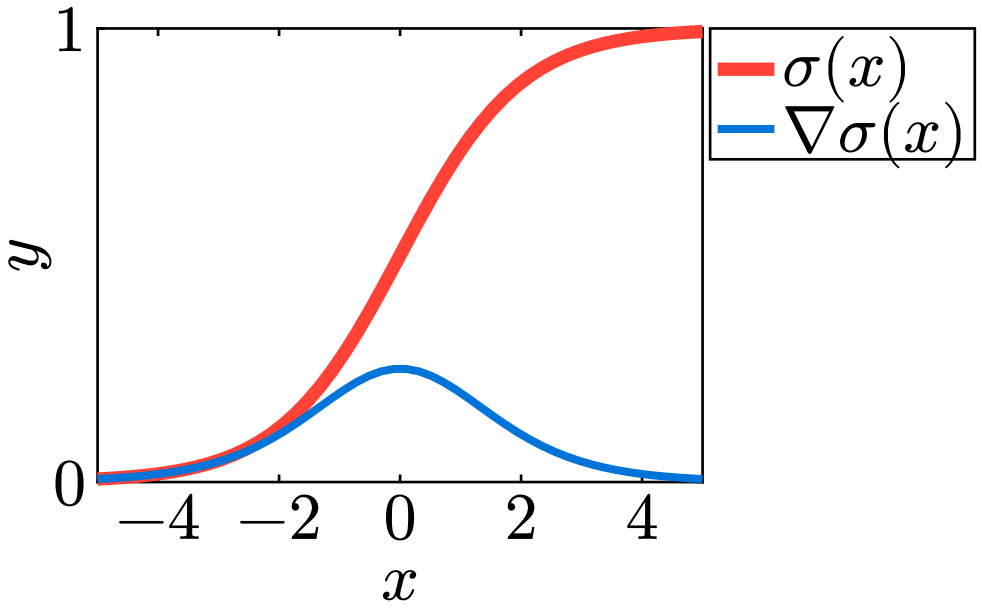


Review



The sigmoid function can result in a **vanishing gradient**

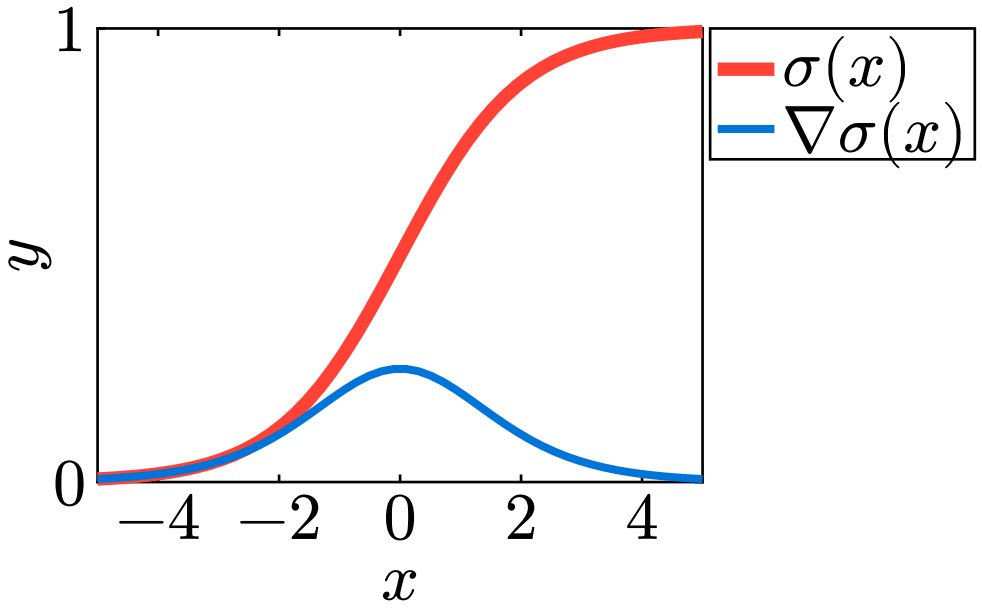
Review



The sigmoid function can result in a **vanishing gradient**

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}_3^\top \sigma(\boldsymbol{\theta}_2^\top \sigma(\boldsymbol{\theta}_1^\top \overline{\boldsymbol{x}})))$$

Review

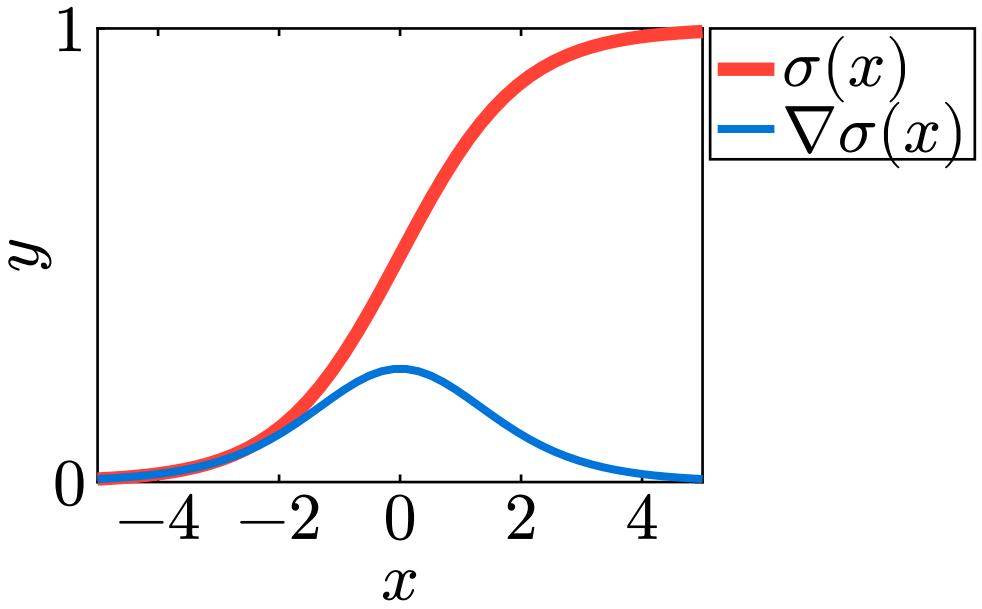


The sigmoid function can result in a **vanishing gradient**

$$f(\boldsymbol{x},\boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}_3^\top \sigma(\boldsymbol{\theta}_2^\top \sigma(\boldsymbol{\theta}_1^\top \overline{\boldsymbol{x}})))$$

$$\nabla_{\boldsymbol{\theta}_1} f(\boldsymbol{x},\boldsymbol{\theta}) = \nabla[\sigma](\boldsymbol{\theta}_3^\top \sigma(\boldsymbol{\theta}_2^\top \sigma(\boldsymbol{\theta}_1^\top \overline{\boldsymbol{x}}))) \cdot \nabla[\sigma](\boldsymbol{\theta}_2^\top \sigma(\boldsymbol{\theta}_1^\top \overline{\boldsymbol{x}})) \cdot \nabla[\sigma](\boldsymbol{\theta}_1^\top \overline{\boldsymbol{x}})$$

Review

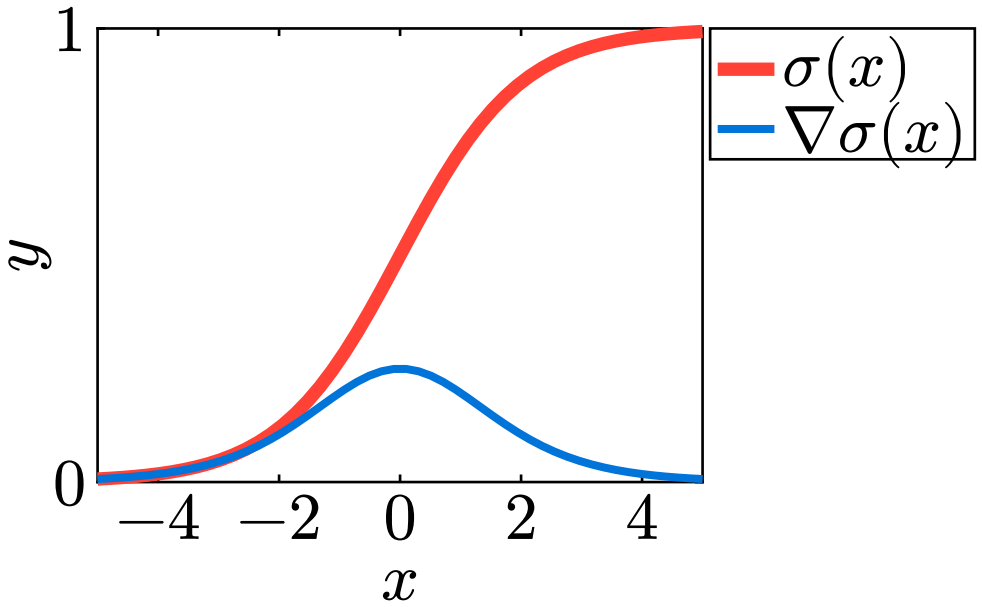


The sigmoid function can result in a **vanishing gradient**

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}_3^\top \sigma(\boldsymbol{\theta}_2^\top \sigma(\boldsymbol{\theta}_1^\top \boldsymbol{x})))$$

$$\nabla_{\boldsymbol{\theta}_1} f(\boldsymbol{x}, \boldsymbol{\theta}) = \underbrace{\nabla[\sigma](\boldsymbol{\theta}_3^\top \sigma(\boldsymbol{\theta}_2^\top \sigma(\boldsymbol{\theta}_1^\top \boldsymbol{x})))}_{<0.5} \cdot \underbrace{\nabla[\sigma](\boldsymbol{\theta}_2^\top \sigma(\boldsymbol{\theta}_1^\top \boldsymbol{x}))}_{<0.5} \cdot \underbrace{\nabla[\sigma](\boldsymbol{\theta}_1^\top \boldsymbol{x})}_{<0.5}$$

Review



The sigmoid function can result in a **vanishing gradient**

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}_3^\top \sigma(\boldsymbol{\theta}_2^\top \sigma(\boldsymbol{\theta}_1^\top \overline{\boldsymbol{x}})))$$

$$\nabla_{\boldsymbol{\theta_1}} f(\boldsymbol{x}, \boldsymbol{\theta}) = \underbrace{\nabla[\sigma](\boldsymbol{\theta}_3^\top \sigma(\boldsymbol{\theta}_2^\top \sigma(\boldsymbol{\theta}_1^\top \overline{\boldsymbol{x}})))}_{<0.5} \cdot \underbrace{\nabla[\sigma](\boldsymbol{\theta}_2^\top \sigma(\boldsymbol{\theta}_1^\top \overline{\boldsymbol{x}}))}_{<0.5} \cdot \underbrace{\nabla[\sigma](\boldsymbol{\theta}_1^\top \overline{\boldsymbol{x}})}_{<0.5}$$

$$\nabla_{\boldsymbol{\theta_1}} f(\boldsymbol{x}, \boldsymbol{\theta}) \approx 0$$

Review

To fix the vanishing gradient, use the **rectified linear unit (ReLU)**

$$\sigma(x) = \max(0, x)$$

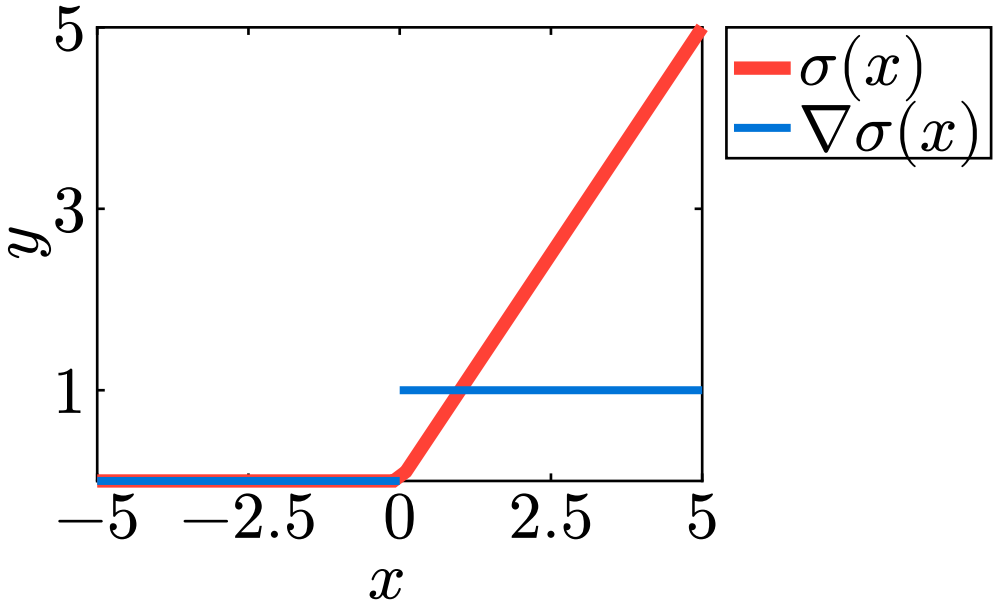
$$\nabla\sigma(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}$$

Review

To fix the vanishing gradient, use the **rectified linear unit (ReLU)**

$$\sigma(x) = \max(0, x)$$

$$\nabla\sigma(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}$$

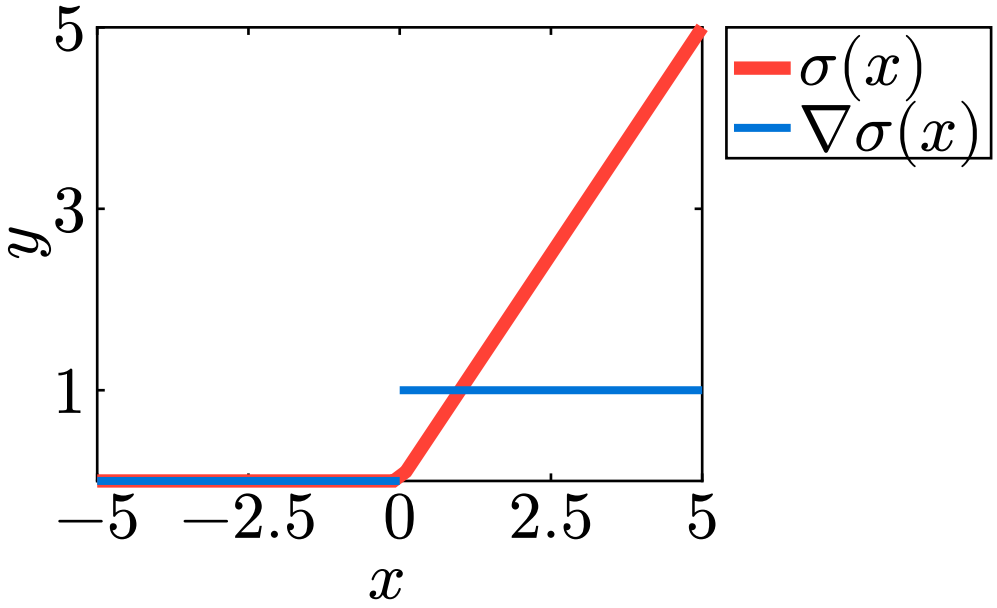


Review

To fix the vanishing gradient, use the **rectified linear unit (ReLU)**

$$\sigma(x) = \max(0, x)$$

$$\nabla\sigma(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}$$



"Dead" neurons always output 0, and cannot recover

Review

To fix dead neurons, use **leaky ReLU**

To fix dead neurons, use **leaky ReLU**

$$\sigma(x) = \max(0.1x, x)$$

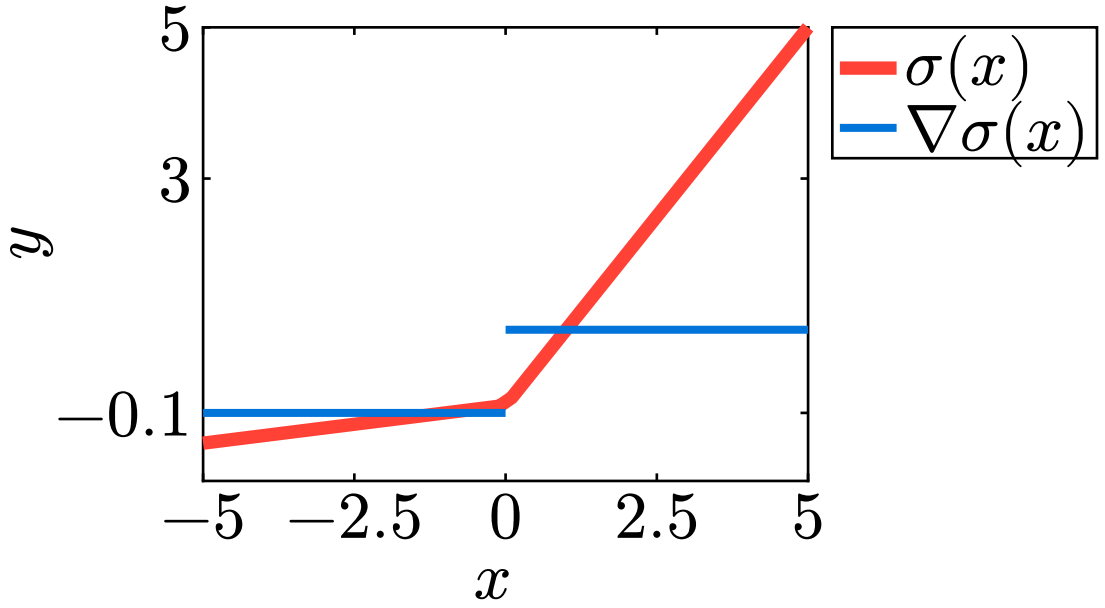
$$\nabla\sigma(x) = \begin{cases} 0.1 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}$$

Review

To fix dead neurons, use **leaky ReLU**

$$\sigma(x) = \max(0.1x, x)$$

$$\nabla\sigma(x) = \begin{cases} 0.1 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}$$

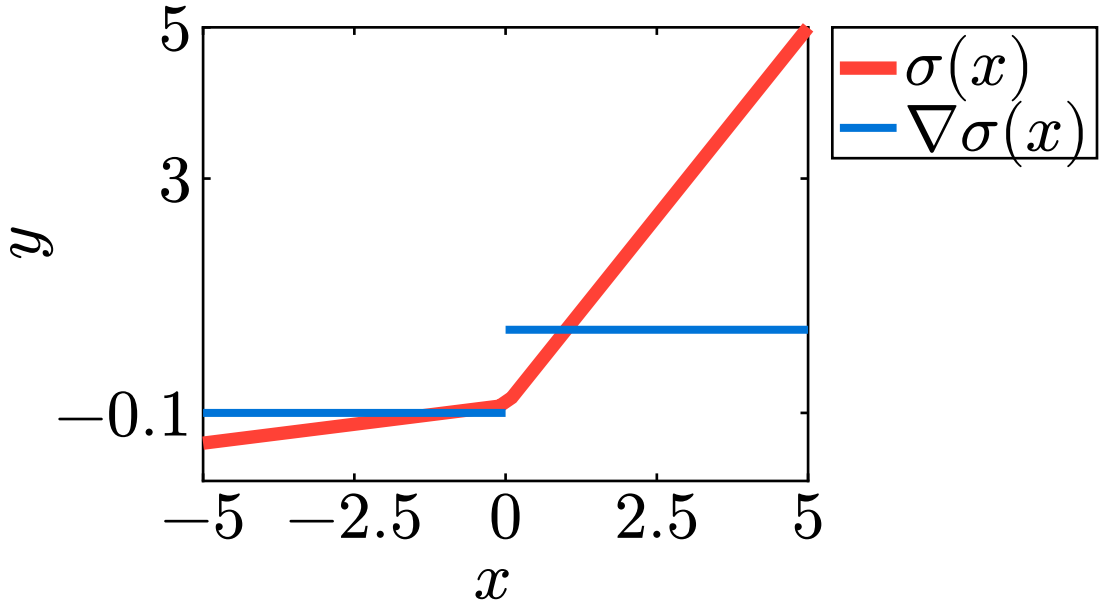


Review

To fix dead neurons, use **leaky ReLU**

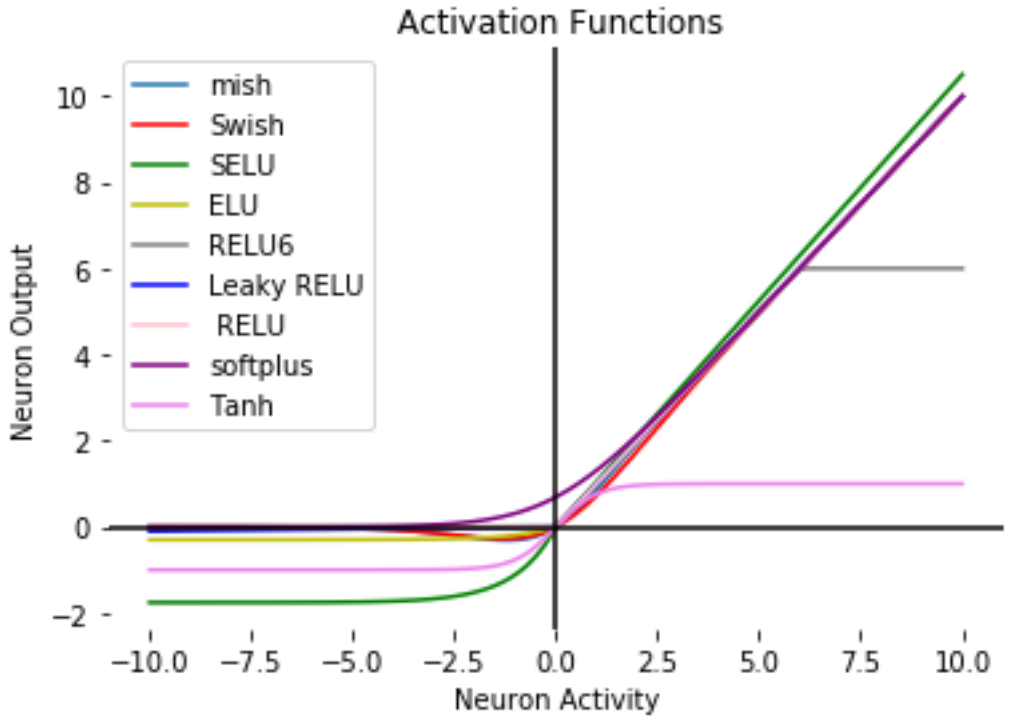
$$\sigma(x) = \max(0.1x, x)$$

$$\nabla\sigma(x) = \begin{cases} 0.1 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}$$



Small negative slope allows dead neurons to recover

Review


Activation Functions
mish
Swish
SELU
ELU
RELU6
Leaky RELU
RELU
softplus
Tanh
Neuron Output
10
8
6
4
2
0
−2
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
Neuron Activity

Review

Then, we discussed parameter initialization

Review

Then, we discussed parameter initialization

Initialization should:

Review

Then, we discussed parameter initialization

Initialization should:

1. Be random to break symmetry within a layer

Review

Then, we discussed parameter initialization

Initialization should:

1. Be random to break symmetry within a layer
2. Scale the parameters to prevent vanishing or exploding gradients

Review

Then, we discussed parameter initialization

Initialization should:

1. Be random to break symmetry within a layer
2. Scale the parameters to prevent vanishing or exploding gradients

$$\boldsymbol{\theta} \sim \mathcal{U}\left[-\frac{\sqrt{6}}{\sqrt{2d_h}}, \frac{\sqrt{6}}{\sqrt{2d_h}}\right]$$

Review

Then, we discussed parameter initialization

Initialization should:

1. Be random to break symmetry within a layer
2. Scale the parameters to prevent vanishing or exploding gradients

$$\boldsymbol{\theta} \sim \mathcal{U}\left[-\frac{\sqrt{6}}{\sqrt{2d_h}}, \frac{\sqrt{6}}{\sqrt{2d_h}}\right]$$

$$\boldsymbol{\theta} \sim \mathcal{U}\left[-\frac{\sqrt{6}}{\sqrt{d_x + d_y}}, \frac{\sqrt{6}}{\sqrt{d_x + d_y}}\right]$$

Review

Finally, we reviewed improvements to optimization:

Review

Finally, we reviewed improvements to optimization:

1. Stochastic gradient descent

Review

Finally, we reviewed improvements to optimization:

1. Stochastic gradient descent
2. Adaptive optimization

Review

Finally, we reviewed improvements to optimization:

1. Stochastic gradient descent
2. Adaptive optimization
3. Weight decay

Review

Stochastic gradient descent (SGD) reduces the memory usage

Review

Stochastic gradient descent (SGD) reduces the memory usage

Rather than compute the gradient over the entire dataset, we approximate the gradient over a subset of the data

Review

Stochastic gradient descent also inserts noise into the optimization process

Review

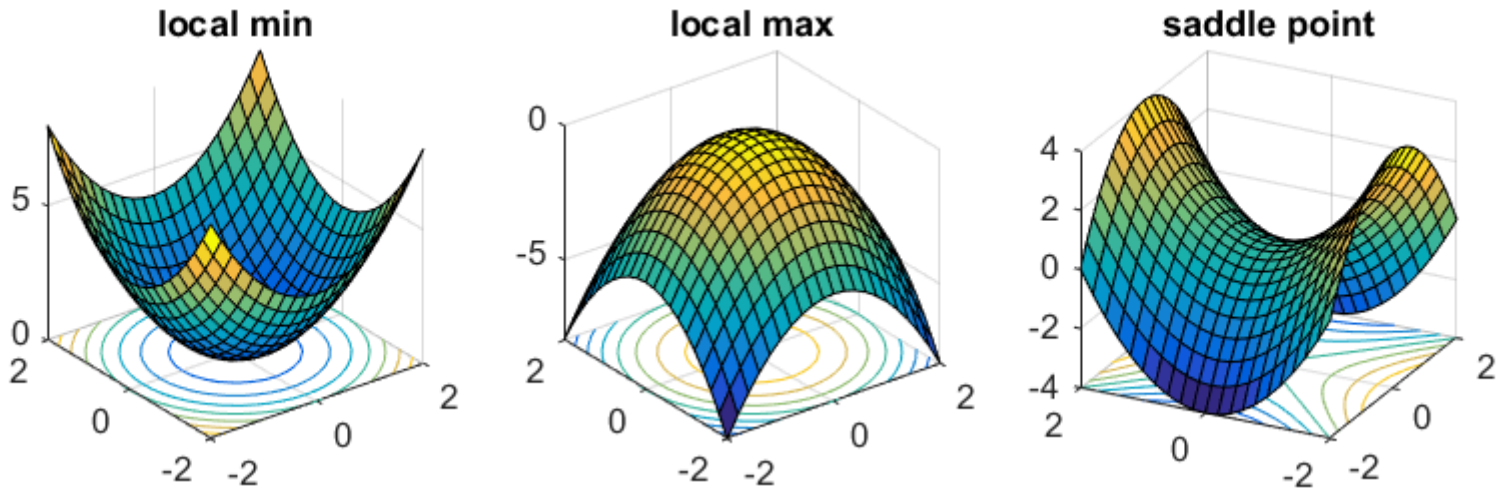
Stochastic gradient descent also inserts noise into the optimization process

This noise can prevent premature convergence to bad optima

Review

Stochastic gradient descent also inserts noise into the optimization process

This noise can prevent premature convergence to bad optima



Review

Use `torch.utils.data.DataLoader` for SGD

Review

Use `torch.utils.data.DataLoader` for SGD

```
import torch
dataloader = torch.utils.data.DataLoader(
    training_data,
    batch_size=32, # How many datapoints to sample
    shuffle=True, # Randomly shuffle each epoch
)
for epoch in number_of_epochs:
    for batch in dataloader:
        X_j, Y_j = batch
        loss = L(X_j, Y_j, theta)
        ...
```

Review

Then, we looked at adaptive variants of gradient descent

Review

Then, we looked at adaptive variants of gradient descent

function GRADIENT DESCENT($\boldsymbol{X}$, $\boldsymbol{Y}$, $\mathcal{L}$, t, α)
    ▷ Randomly initialize parameters
    $\boldsymbol{\theta} \leftarrow \text{Glorot}()$
    for $i \in 1...t$ **do**
        ▷ Compute the gradient of the loss
        $\boldsymbol{J} \leftarrow \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta})$
        ▷ Update the parameters using the negative gradient
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \cdot \boldsymbol{J}$
    return $\boldsymbol{\theta}$

Introduce **momentum** first

Review

Introduce **momentum** first

1: **function** Momentum Gradient Descent($\boldsymbol{X}$, $\boldsymbol{Y}$, $\mathcal{L}$, t, α, β)
2: $\boldsymbol{\theta} \leftarrow \text{Glorot}()$
3: $\boldsymbol{M} \leftarrow \boldsymbol{0}$ # Init momentum
4: **for** $i \in 1...t$ **do**
5: $\boldsymbol{J} \leftarrow \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta})$ # Represents acceleration
6: $\boldsymbol{M} \leftarrow \beta \cdot \boldsymbol{M} + (1 - \beta) \cdot \boldsymbol{J}$ # Momentum and acceleration
7: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \cdot \boldsymbol{M}$
8: **return** $\boldsymbol{\theta}$

Now **adaptive learning rate**

Now **adaptive learning rate**

function RMSProp($\boldsymbol{X}, \boldsymbol{Y}, \mathcal{L}, t, \alpha, \beta, \varepsilon$)
    $\boldsymbol{\theta} \leftarrow \text{Glorot}()$
    $\boldsymbol{V} \leftarrow \boldsymbol{0}$ # Init variance
    for $i \in 1...t$ **do**
        $\boldsymbol{J} \leftarrow \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta})$ # Represents acceleration
        $\boldsymbol{V} \leftarrow \beta \cdot \boldsymbol{V} + (1 - \beta) \cdot \boldsymbol{J} \odot \boldsymbol{J}$ # Magnitude
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \cdot \boldsymbol{J} \oslash \sqrt[\circ]{\boldsymbol{V} + \varepsilon}$ # Rescale grad by prev updates
    return $\boldsymbol{\theta}$

Review

Combine **momentum** and **adaptive learning rate** to create **Adam**

Review

Combine **momentum** and **adaptive learning rate** to create **Adam**

function ADAPTIVE MOMENT ESTIMATION($\boldsymbol{X}$, $\boldsymbol{Y}$, $\mathcal{L}$, t, α, β_1, β_2, ε)
    $\boldsymbol{\theta} \leftarrow \text{Glorot}()$
    $\boldsymbol{M}, \boldsymbol{V} \leftarrow \mathbf{0}$
    for $i \in 1...t$ **do**
        $\boldsymbol{J} \leftarrow \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta})$
        $\boldsymbol{M} \leftarrow \beta_1 \cdot \boldsymbol{M} + (1 - \beta_1)\boldsymbol{J}$ # Compute momentum
        $\boldsymbol{V} \leftarrow \beta_2 \cdot \boldsymbol{V} + (1 - \beta_2) \cdot \boldsymbol{J} \odot \boldsymbol{J}$ # Magnitude
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \cdot \boldsymbol{M} \oslash \sqrt[\odot]{\boldsymbol{V} + \varepsilon}$ # Adaptive param update
    return $\boldsymbol{\theta}$ # Note, we use biased $\boldsymbol{M}, \boldsymbol{V}$ for clarity

Review

Many modern optimizers also include **weight decay**

Review

Many modern optimizers also include **weight decay**

Weight decay penalizes large parameters

Review

Many modern optimizers also include **weight decay**

Weight decay penalizes large parameters

$$\mathcal{L}_{\text{decay}}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \mathcal{L}_{\text{decay}}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) + \lambda \sum_i \theta_i^2$$

Review

Many modern optimizers also include **weight decay**

Weight decay penalizes large parameters

$$\mathcal{L}_{\text{decay}}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \mathcal{L}_{\text{decay}}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) + \lambda \sum_i \theta_i^2$$

This results in a smoother, parabolic loss landscape that is easier to optimize

Agenda

1. **Review**
2. Signal Processing
3. Convolution
4. Convolutional Neural Networks
5. Additional Dimensions
6. Deeper Networks
7. Coding

Agenda

1. Review
2. **Signal Processing**
3. Convolution
4. Convolutional Neural Networks
5. Additional Dimensions
6. Deeper Networks
7. Coding

Signal Processing

In our networks, we do not consider the structure of data

Signal Processing

In our networks, we do not consider the structure of data

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{d_x} \end{bmatrix}$$

Signal Processing

In our networks, we do not consider the structure of data

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{d_x} \end{bmatrix}$$

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_{d_x} \end{bmatrix}$$

Signal Processing

In our networks, we do not consider the structure of data

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{d_x} \end{bmatrix}$$

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_{d_x} \end{bmatrix} \qquad \boldsymbol{\theta} \stackrel{?}{=} \begin{bmatrix} \theta_7 \\ \theta_4 \\ \vdots \\ \theta_1 \end{bmatrix}$$

Signal Processing

We treat images as a vector, with no relationship between nearby pixels

Signal Processing

We treat images as a vector, with no relationship between nearby pixels

These images are equivalent to a neural network

Signal Processing

We treat images as a vector, with no relationship between nearby pixels

These images are equivalent to a neural network

It is a miracle that our neural networks could classify clothing!

Signal Processing

There is structure inherent in the real world

Signal Processing

There is structure inherent in the real world

By representing this structure, we can make more efficient neural networks that learn faster and generalize better

Signal Processing

There is structure inherent in the real world

By representing this structure, we can make more efficient neural networks that learn faster and generalize better

To do so, we will think of the world as a collection of signals

Signal Processing

A **signal** represents information as a function of time, space or some other variable

Signal Processing

A **signal** represents information as a function of time, space or some other variable

$$x(t) = 2t + 1$$

Signal Processing

A **signal** represents information as a function of time, space or some other variable

$$x(t) = 2t + 1$$

$$x(u, v) = \frac{u^2}{v} - 3$$

Signal Processing

A **signal** represents information as a function of time, space or some other variable

$$x(t) = 2t + 1$$

$$x(u, v) = \frac{u^2}{v} - 3$$

$x(t), x(u, v)$ represent physical processes that we may or may not know

Signal Processing

A **signal** represents information as a function of time, space or some other variable

$$x(t) = 2t + 1$$

$$x(u, v) = \frac{u^2}{v} - 3$$

$x(t), x(u, v)$ represent physical processes that we may or may not know

In **signal processing**, we analyze the meaning of signals

Signal Processing

A **signal** represents information as a function of time, space or some other variable

$$x(t) = 2t + 1$$

$$x(u, v) = \frac{u^2}{v} - 3$$

$x(t), x(u, v)$ represent physical processes that we may or may not know

In **signal processing**, we analyze the meaning of signals

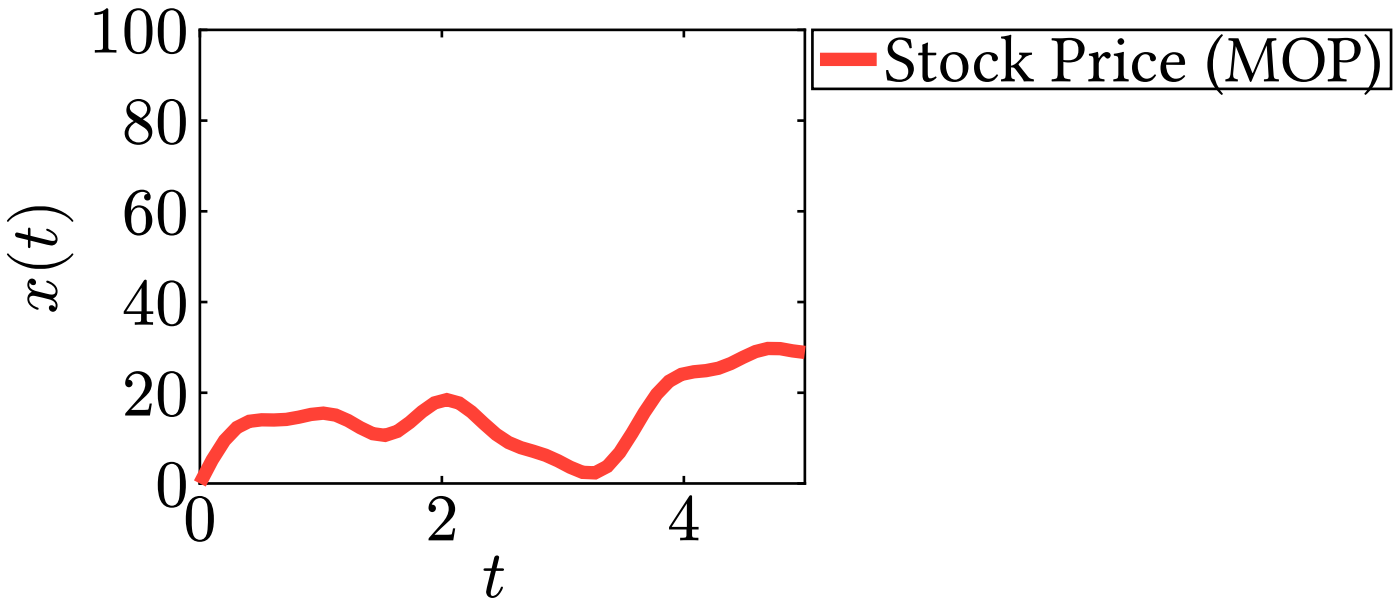
Knowing the meaning of signals is very useful

Signal Processing

$$x(t) = \text{stock price}$$

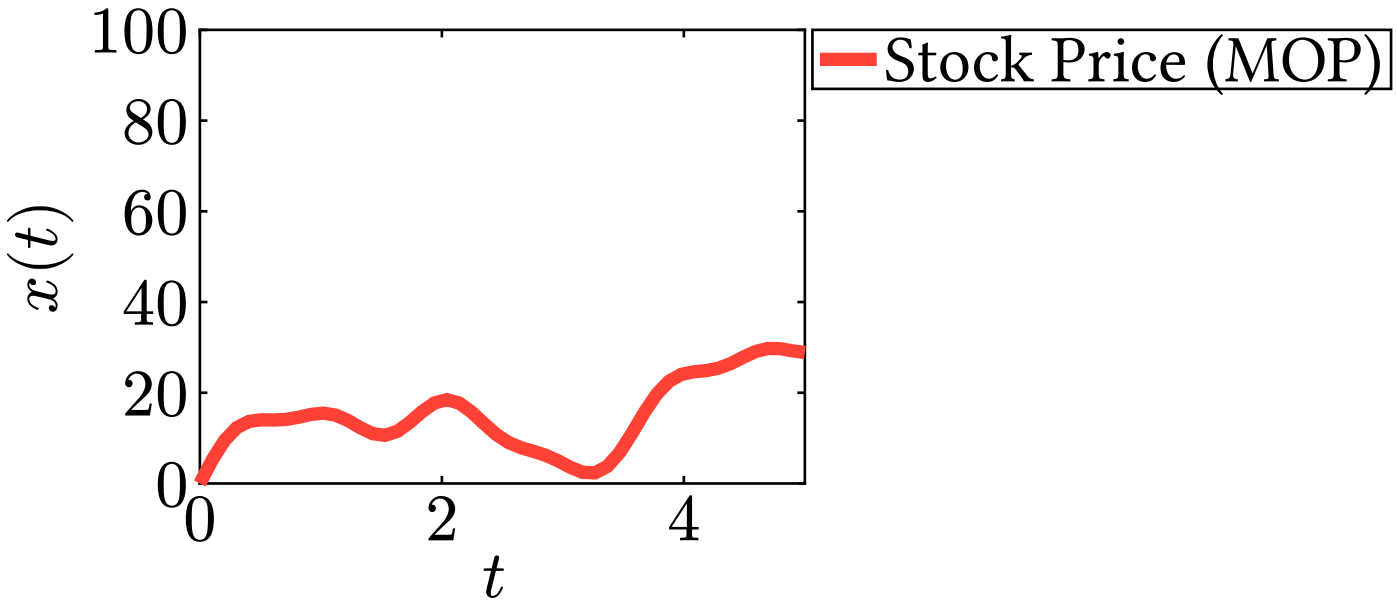
Signal Processing

$$x(t) = \text{stock price}$$



Signal Processing

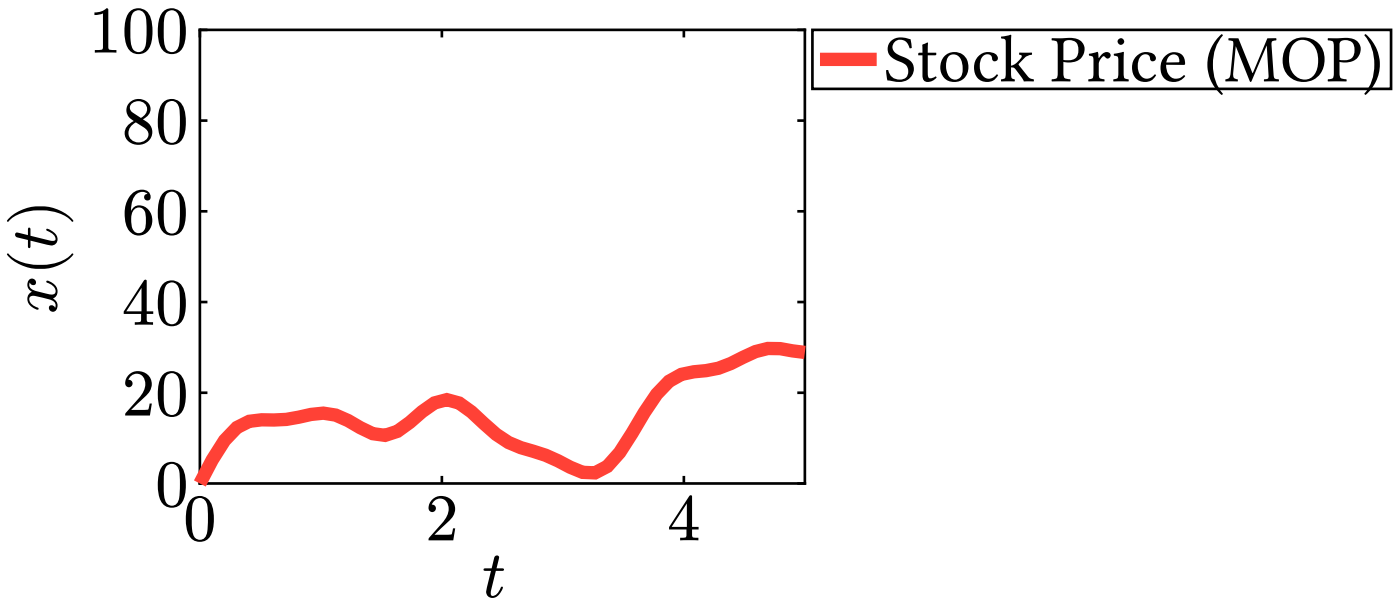
$$x(t) = \text{stock price}$$



There is an underlying structure to $x(t)$

Signal Processing

$$x(t) = \text{stock price}$$



There is an underlying structure to $x(t)$

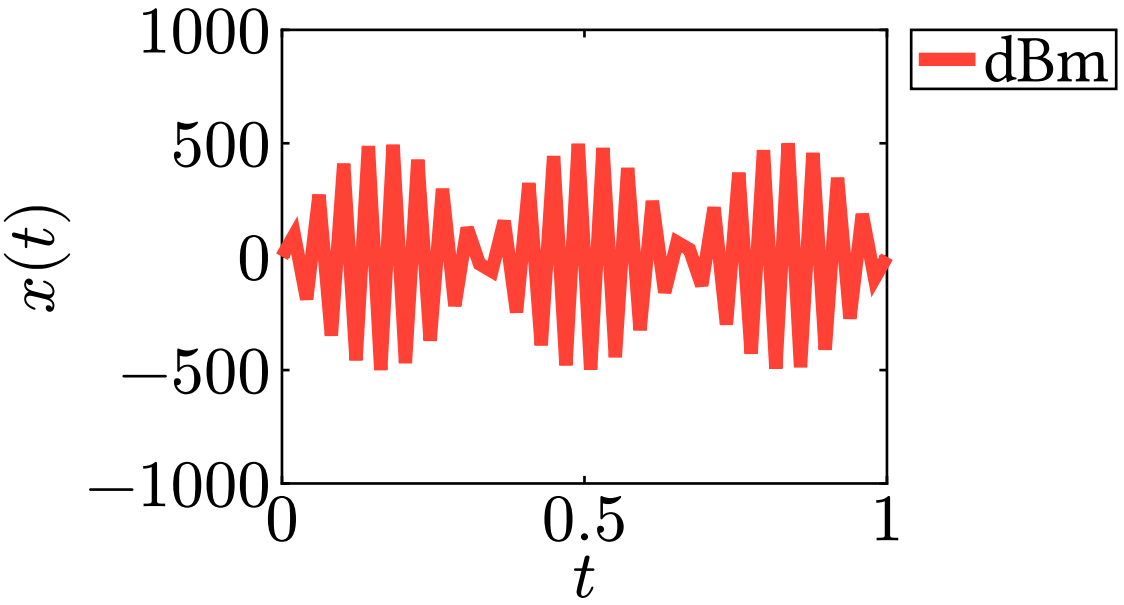
Structure: Tomorrow's stock price will be close to today's stock price

Signal Processing

$$x(t) = \text{audio}$$

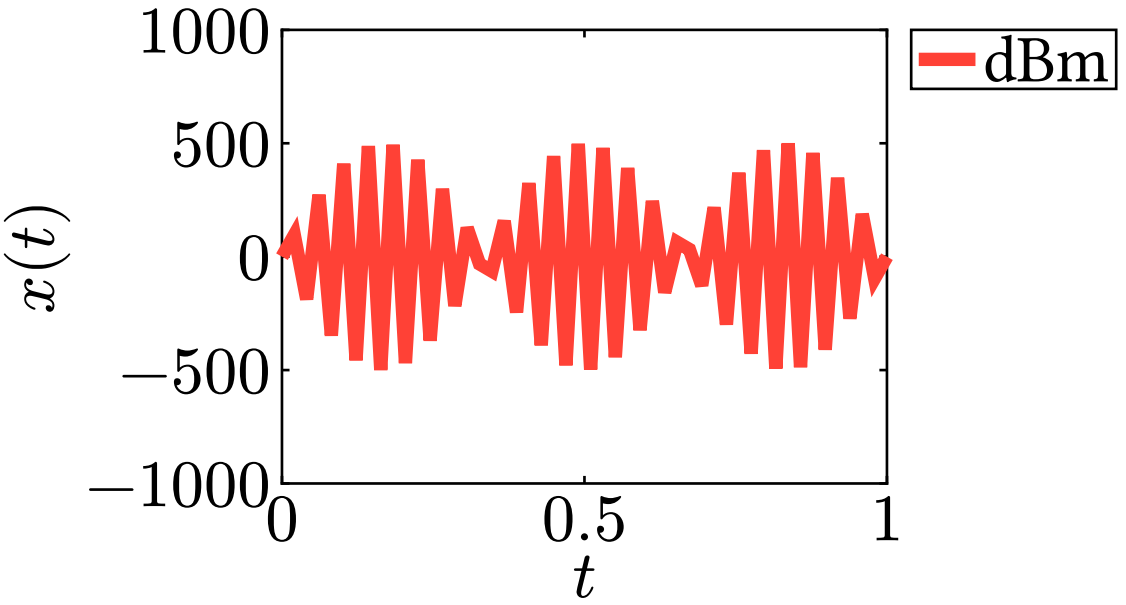
Signal Processing

$$x(t) = \text{audio}$$



Signal Processing

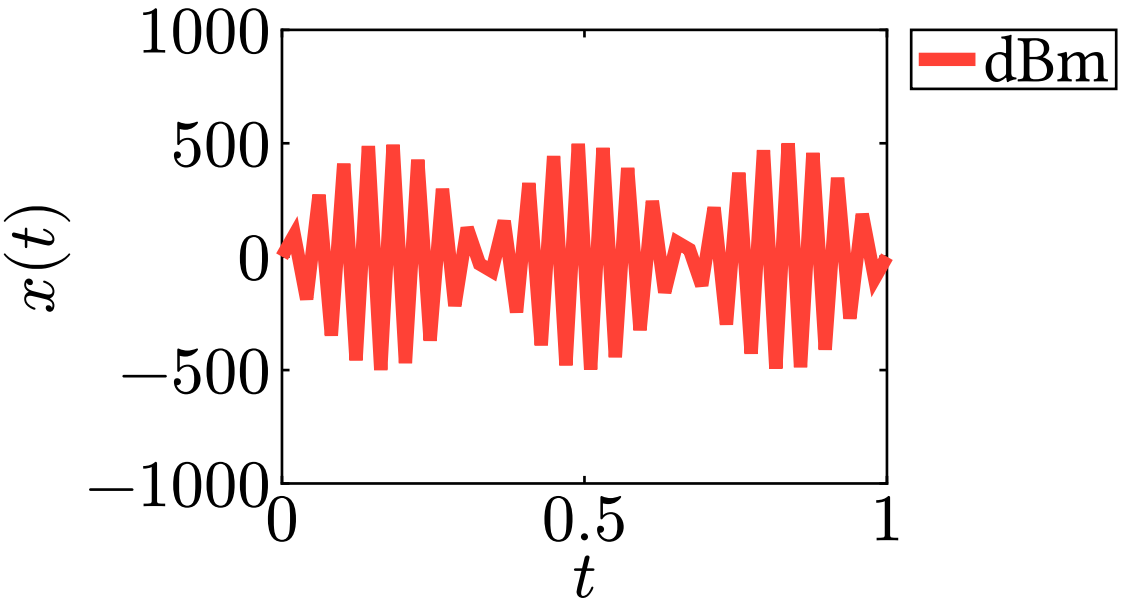
$$x(t) = \text{audio}$$



Structure: Nearby waves form syllables

Signal Processing

$$x(t) = \text{audio}$$



Structure: Nearby waves form syllables

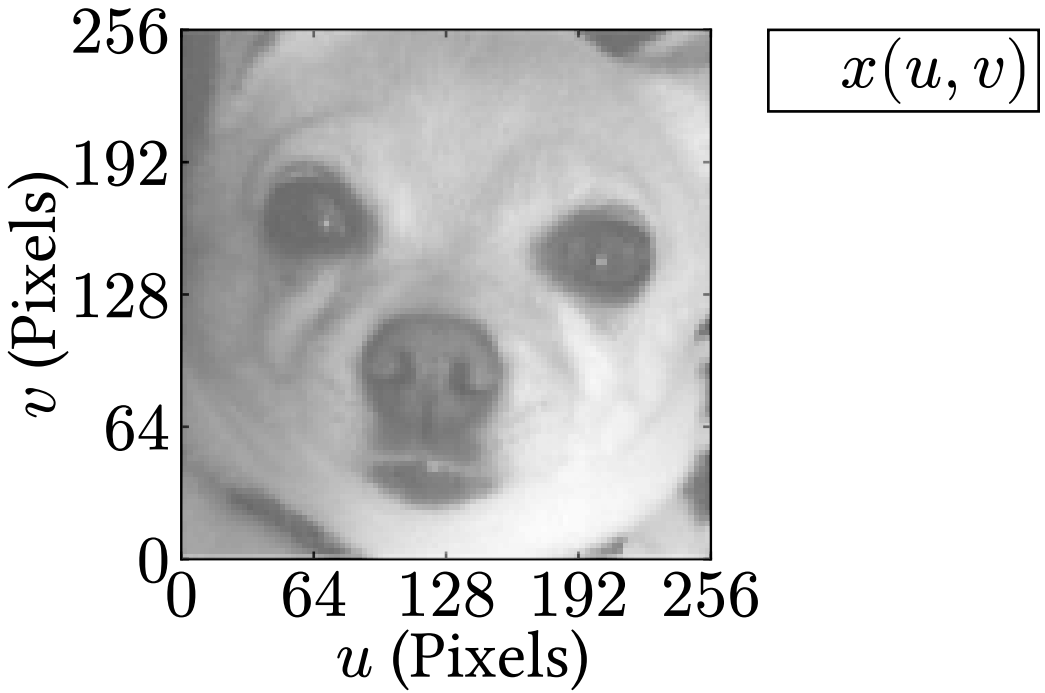
Structure: Nearby syllables combine to create meaning

Signal Processing

$$x(u, v) = \text{image}$$

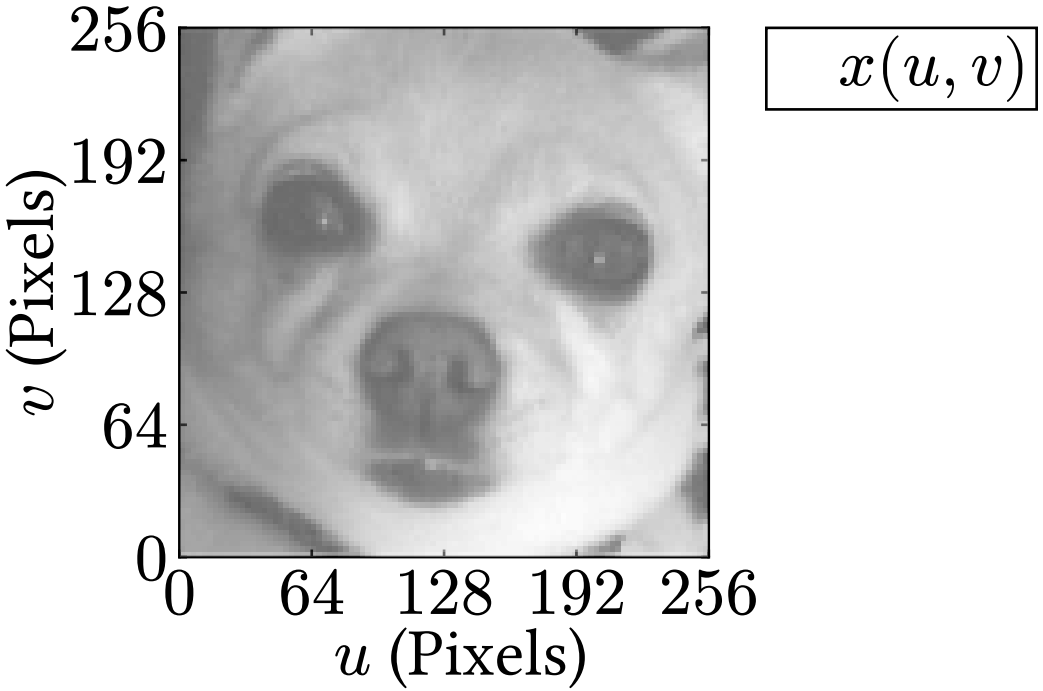
Signal Processing

$$x(u, v) = \text{image}$$



Signal Processing

$$x(u, v) = \text{image}$$



Structure: Repeated components (circles, symmetry, eyes, nostrils, etc)

Signal Processing

In signal processing, we often consider two properties:

Signal Processing

In signal processing, we often consider two properties:

- Locality

Signal Processing

In signal processing, we often consider two properties:

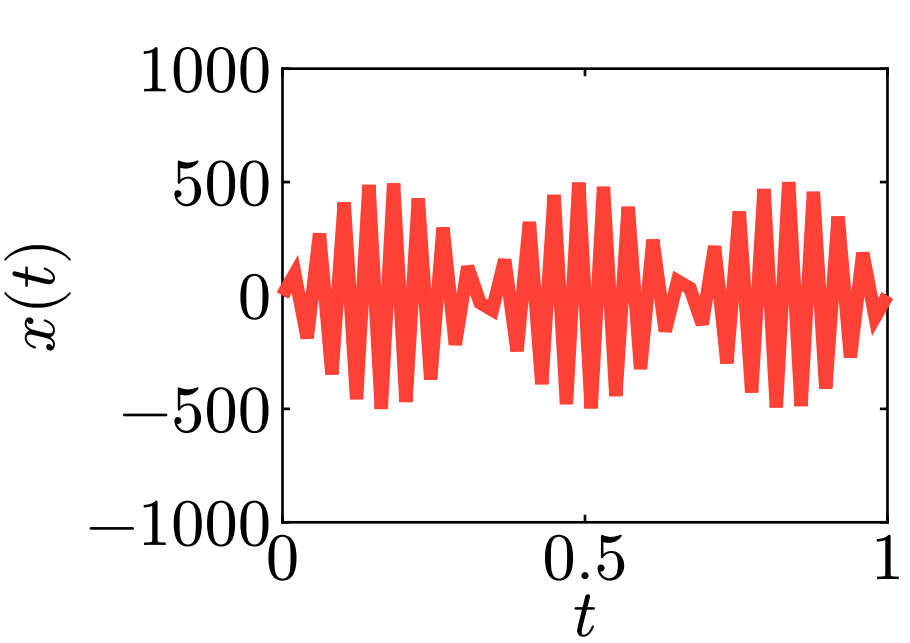
- Locality
- Translation equivariance

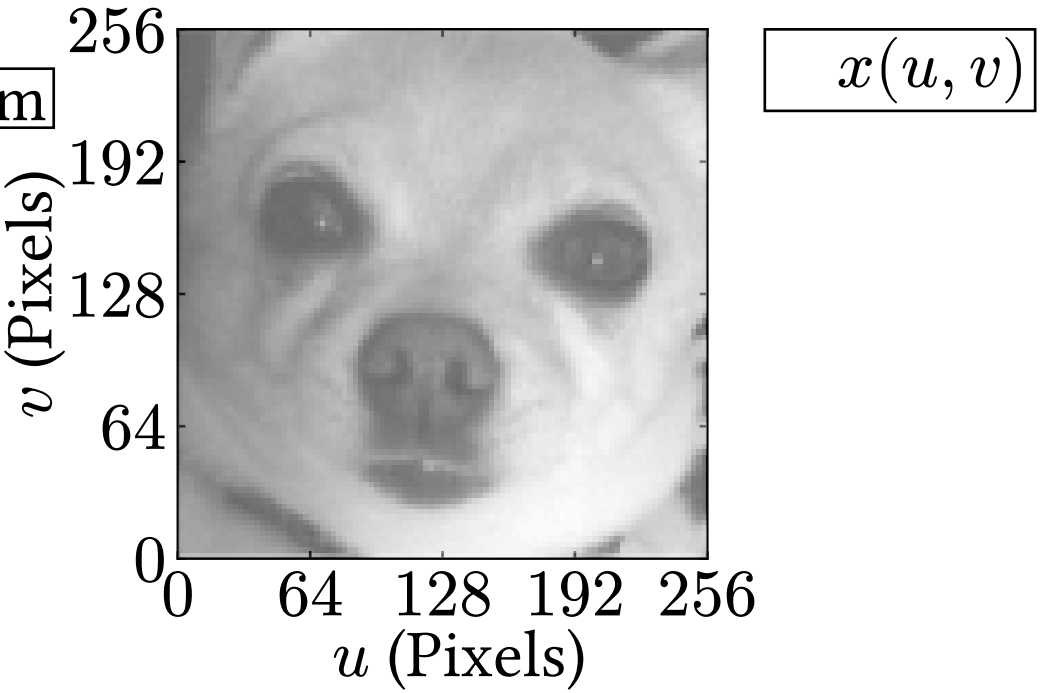
Signal Processing

Locality: Information concentrated over small regions of space/time

Signal Processing

Locality: Information concentrated over small regions of space/time



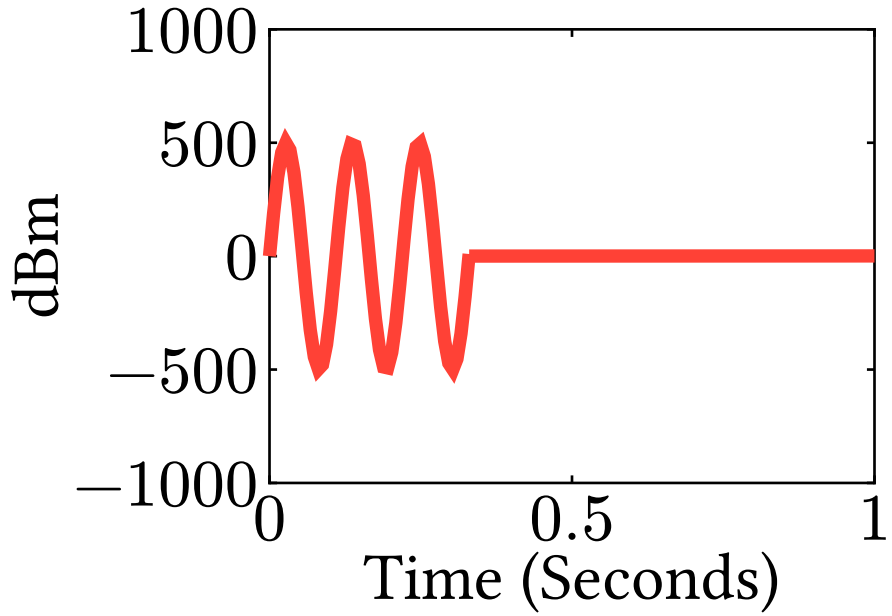


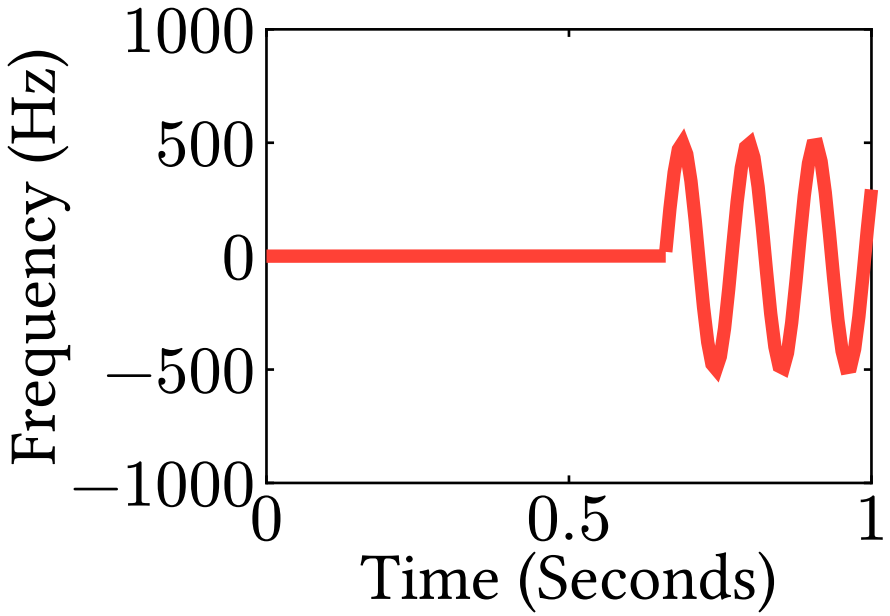
Signal Processing

Translation Equivariance: Shift in signal results in shift in output

Signal Processing

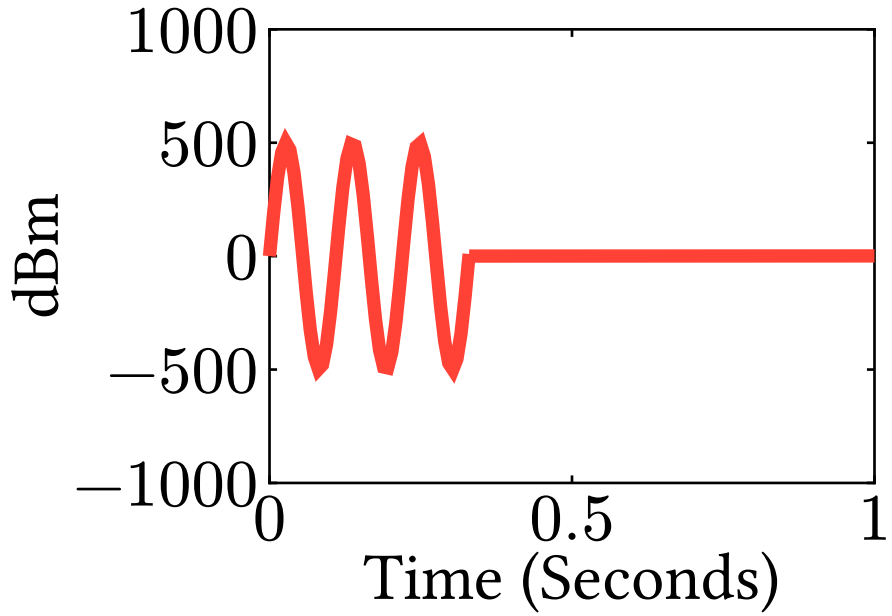
Translation Equivariance: Shift in signal results in shift in output

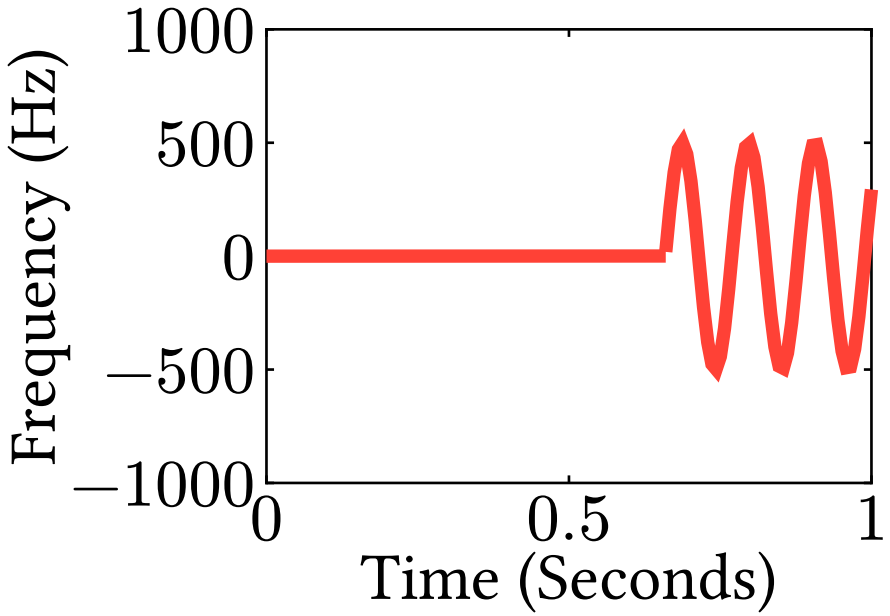




Signal Processing

Translation Equivariance: Shift in signal results in shift in output





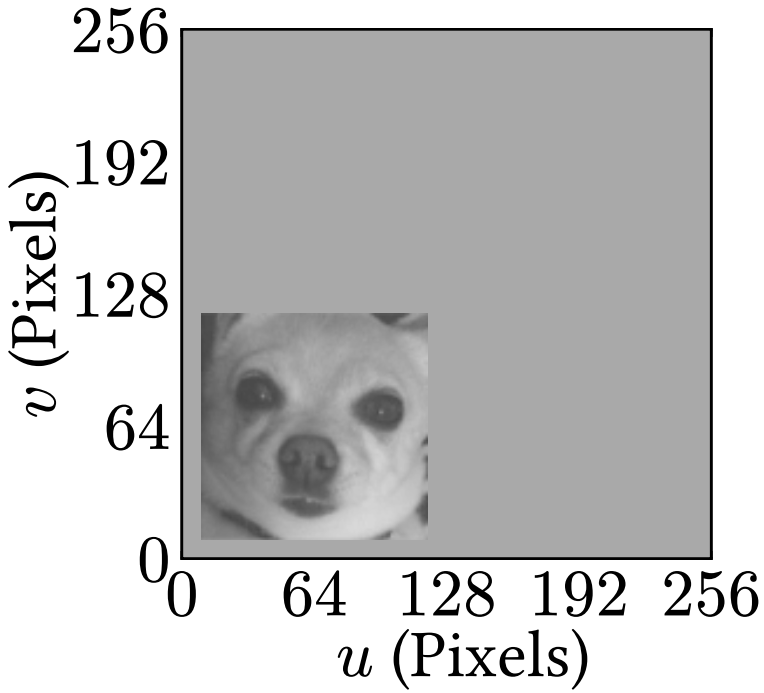
Both say “hello”

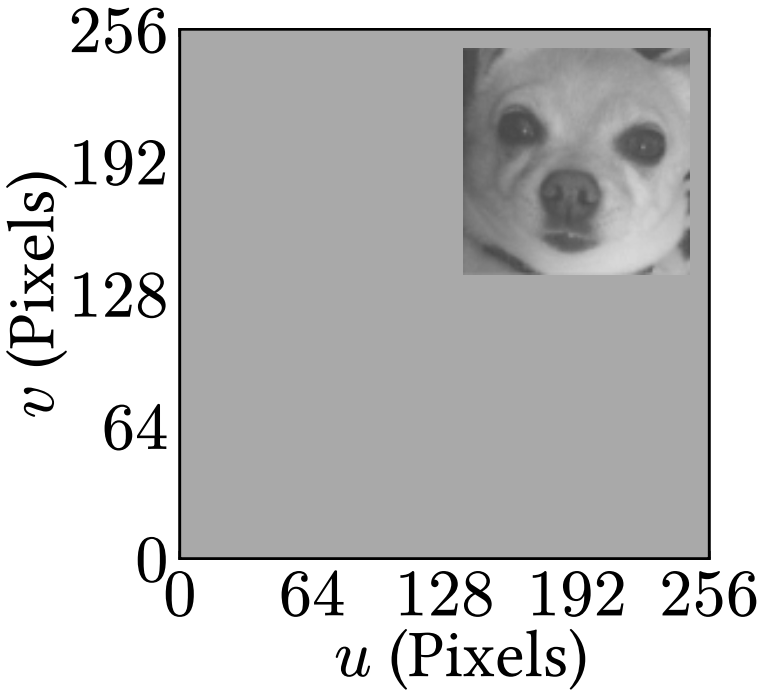
Signal Processing

Translation Equivariance: Shift in signal results in shift in output

Signal Processing

Translation Equivariance: Shift in signal results in shift in output





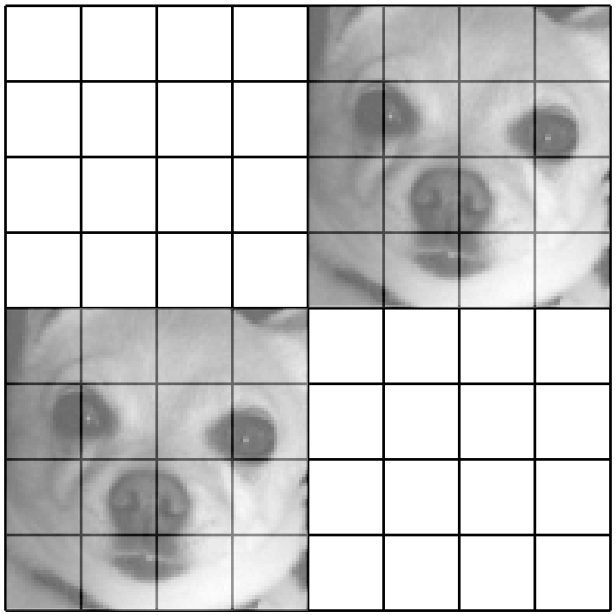
Both contain a dog

Signal Processing

Perceptrons are not local or translation equivariant, each pixel is an independent neuron

Signal Processing

Perceptrons are not local or translation equivariant, each pixel is an independent neuron

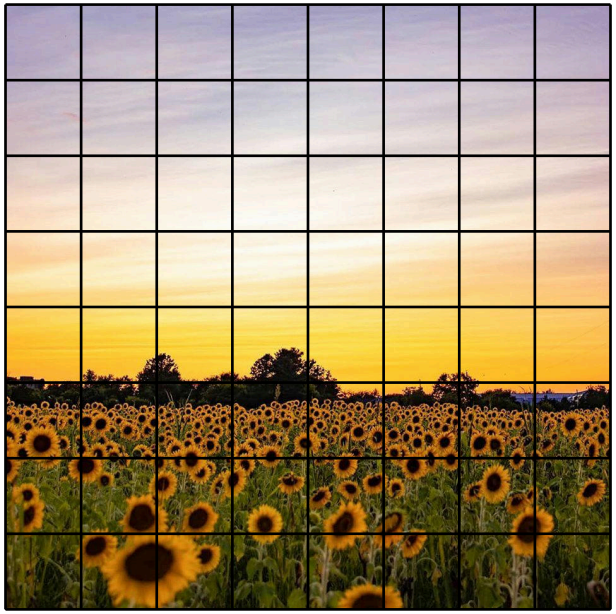


Signal Processing

A more realistic scenario of locality and translation equivariance

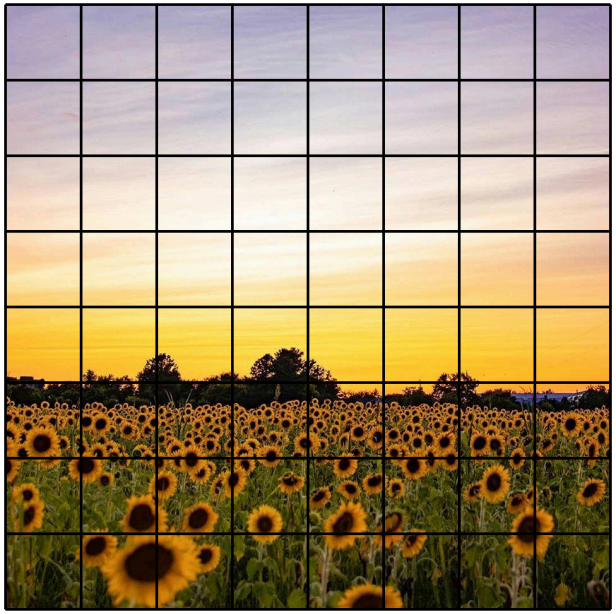
Signal Processing

A more realistic scenario of locality and translation equivariance



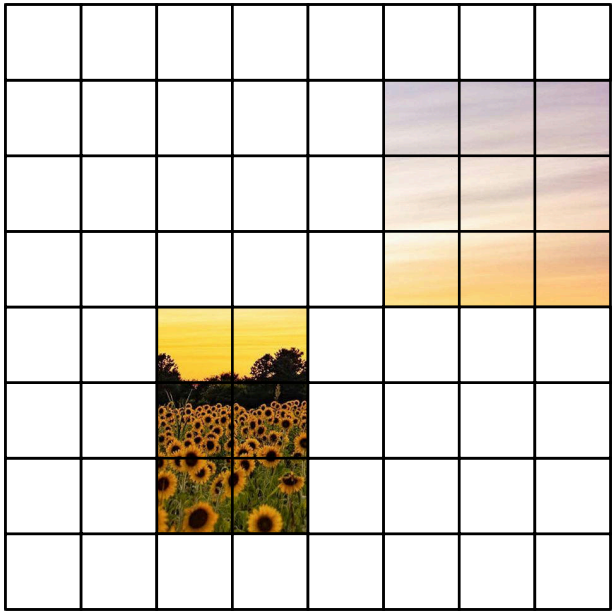
Signal Processing

A more realistic scenario of locality and translation equivariance



Signal Processing

A more realistic scenario of locality and translation equivariance



Agenda

1. Review
2. **Signal Processing**
3. Convolution
4. Convolutional Neural Networks
5. Additional Dimensions
6. Deeper Networks
7. Coding

Agenda

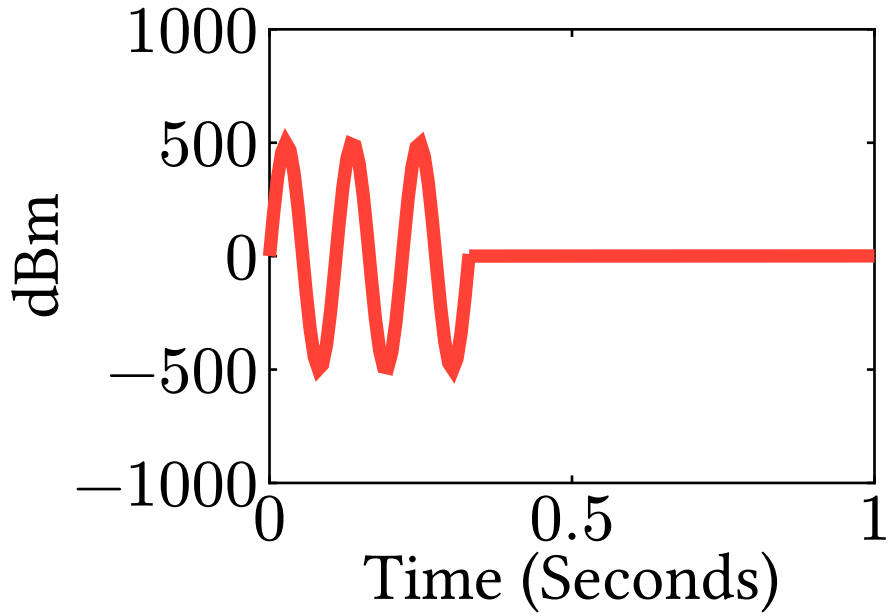
1. Review
2. Signal Processing
3. **Convolution**
4. Convolutional Neural Networks
5. Additional Dimensions
6. Deeper Networks
7. Coding

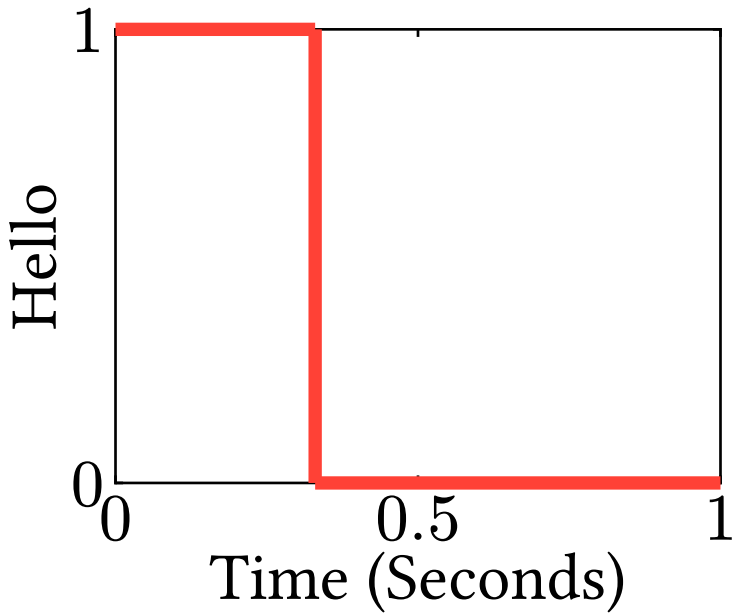
Convolution

In signal processing, we often turn signals into other signals

Convolution

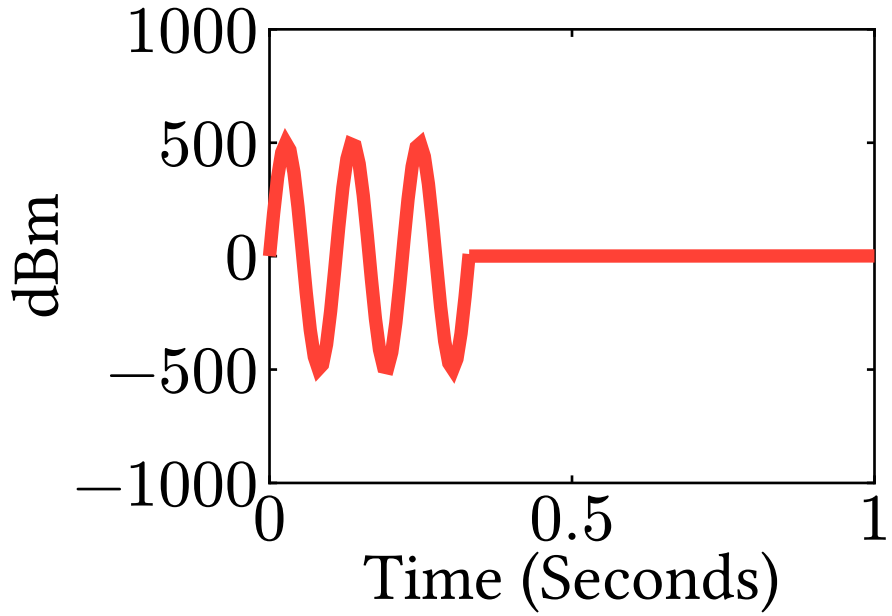
In signal processing, we often turn signals into other signals

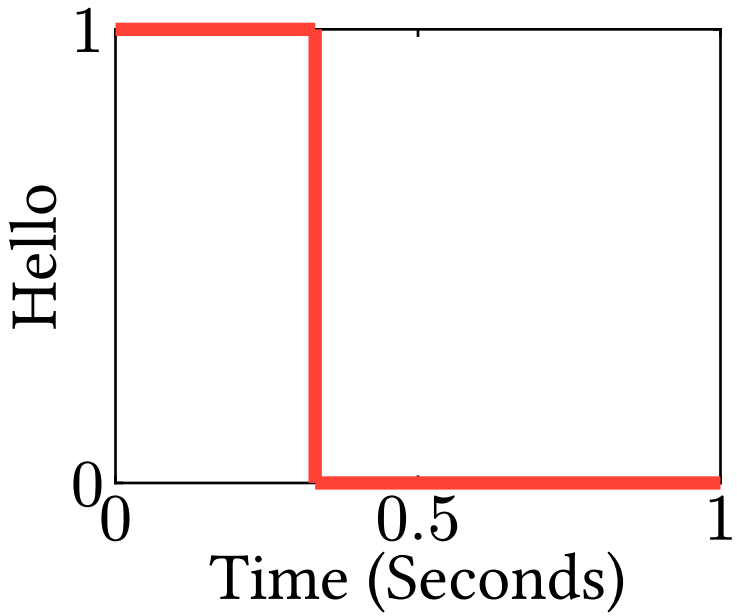




Convolution

In signal processing, we often turn signals into other signals

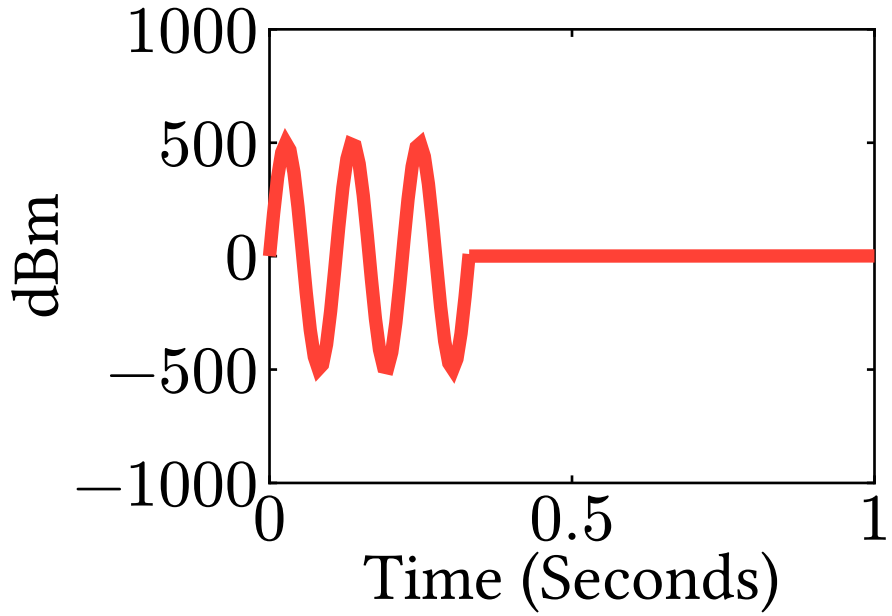


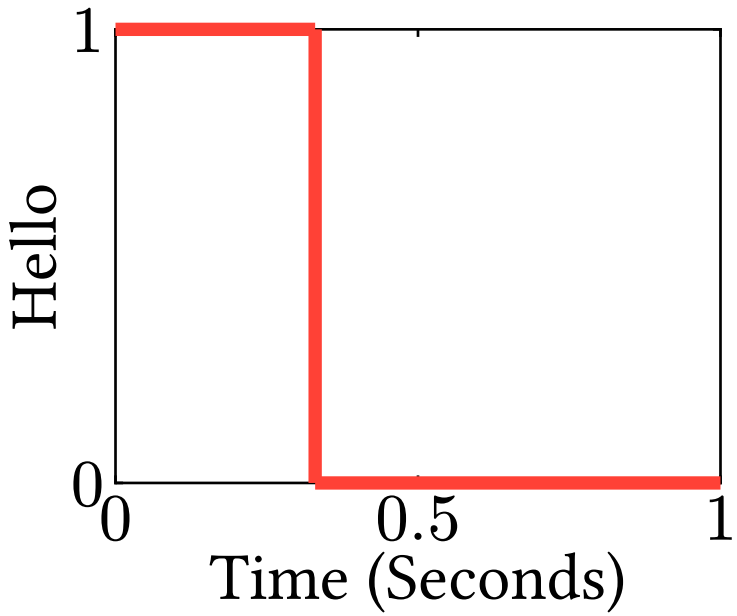


A standard way to transform signals is **convolution**

Convolution

In signal processing, we often turn signals into other signals





A standard way to transform signals is **convolution**

Convolution is translation equivariant and local

Convolution

Convolution is the sum of products of a signal $x(t)$ and a **filter** $g(t)$

Convolution

Convolution is the sum of products of a signal $x(t)$ and a **filter** $g(t)$

If the t is continuous in $x(t)$

$$x(t) * g(t) = \int_{-\infty}^{\infty} x(t-\tau)g(\tau)d\tau$$

Convolution

Convolution is the sum of products of a signal $x(t)$ and a **filter** $g(t)$

If the t is continuous in $x(t)$

$$x(t) * g(t) = \int_{-\infty}^{\infty} x(t-\tau)g(\tau)d\tau$$

If the t is discrete in $x(t)$

$$x(t) * g(t) = \sum_{\tau=-\infty}^{\infty} x(t-\tau)g(\tau)$$

Convolution

Convolution is the sum of products of a signal $x(t)$ and a **filter** $g(t)$

If the t is continuous in $x(t)$

$$x(t) * g(t) = \int_{-\infty}^{\infty} x(t-\tau)g(\tau)d\tau$$

If the t is discrete in $x(t)$

$$x(t) * g(t) = \sum_{\tau=-\infty}^{\infty} x(t-\tau)g(\tau)$$

We slide the filter $g(t)$ across the signal $x(t)$

Convolution

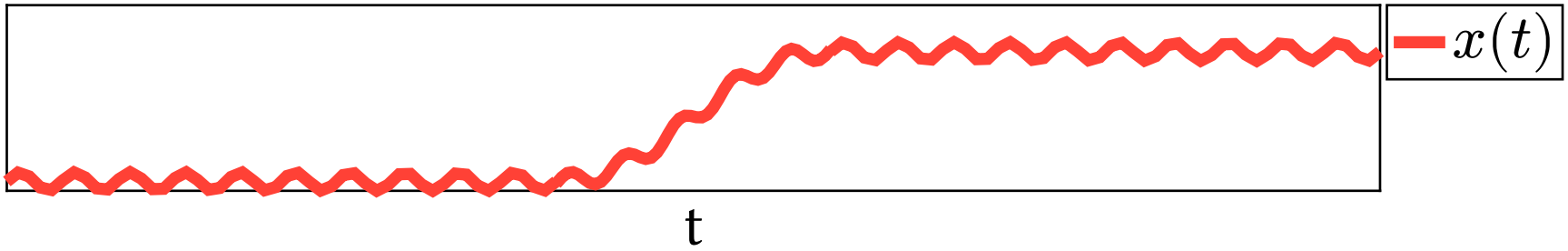
Example: Let us examine a low-pass filter

Convolution

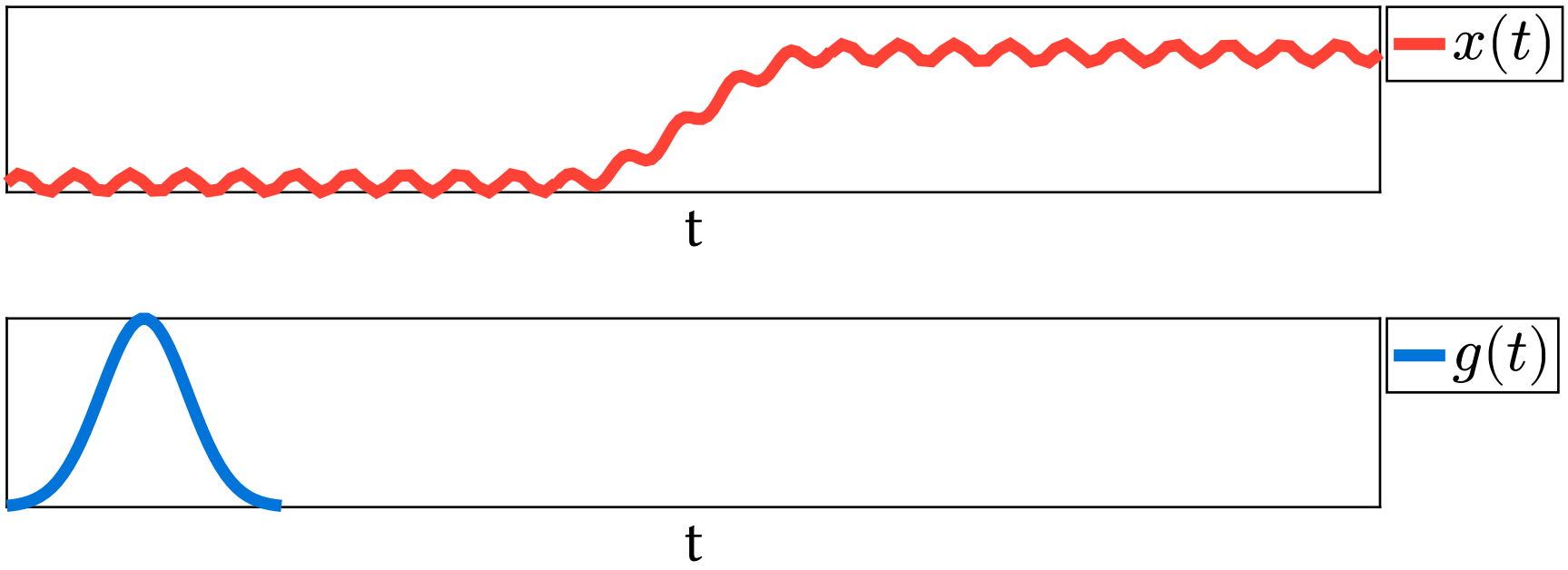
Example: Let us examine a low-pass filter

The filter will take a signal and remove noise, producing a cleaner signal

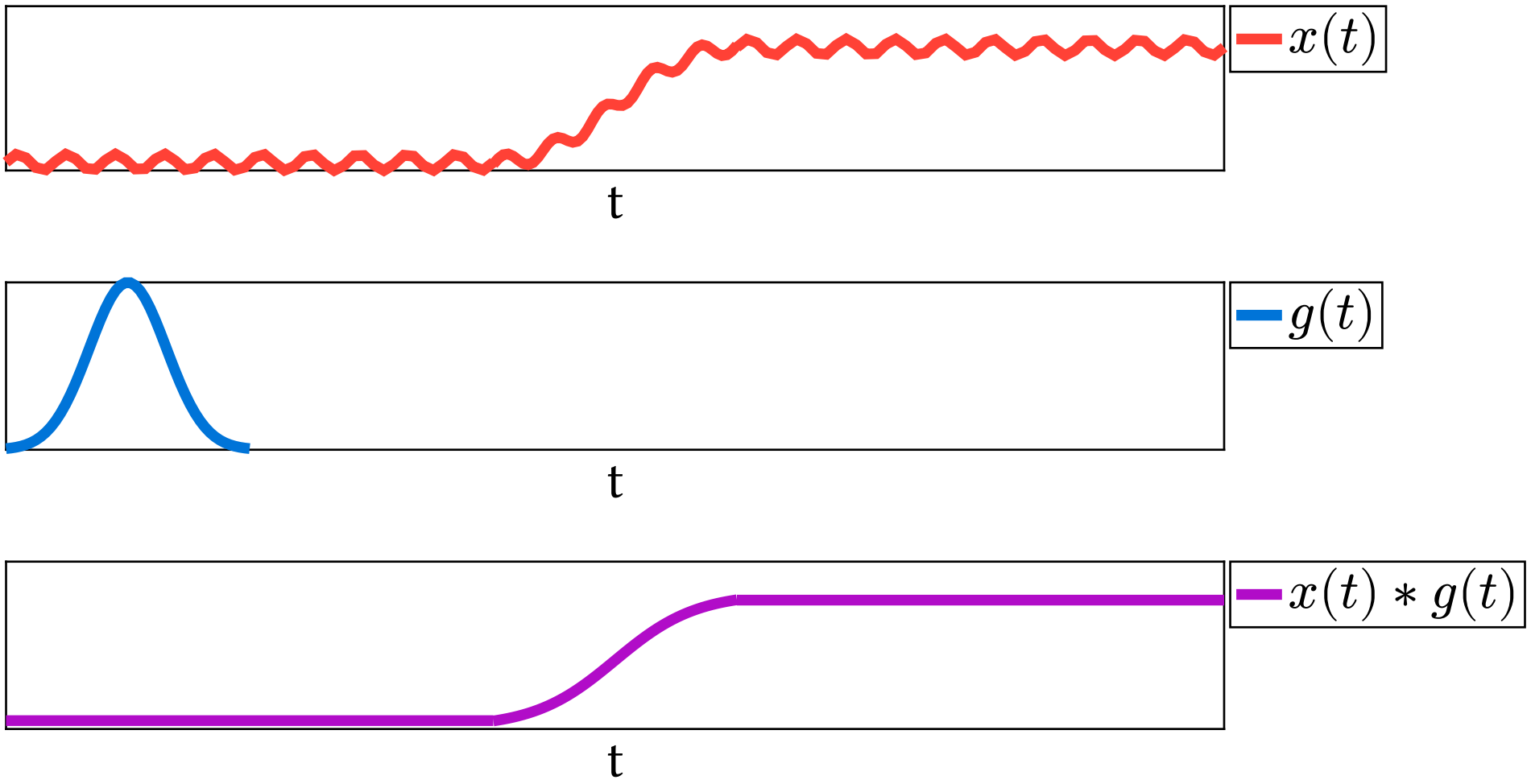
Convolution



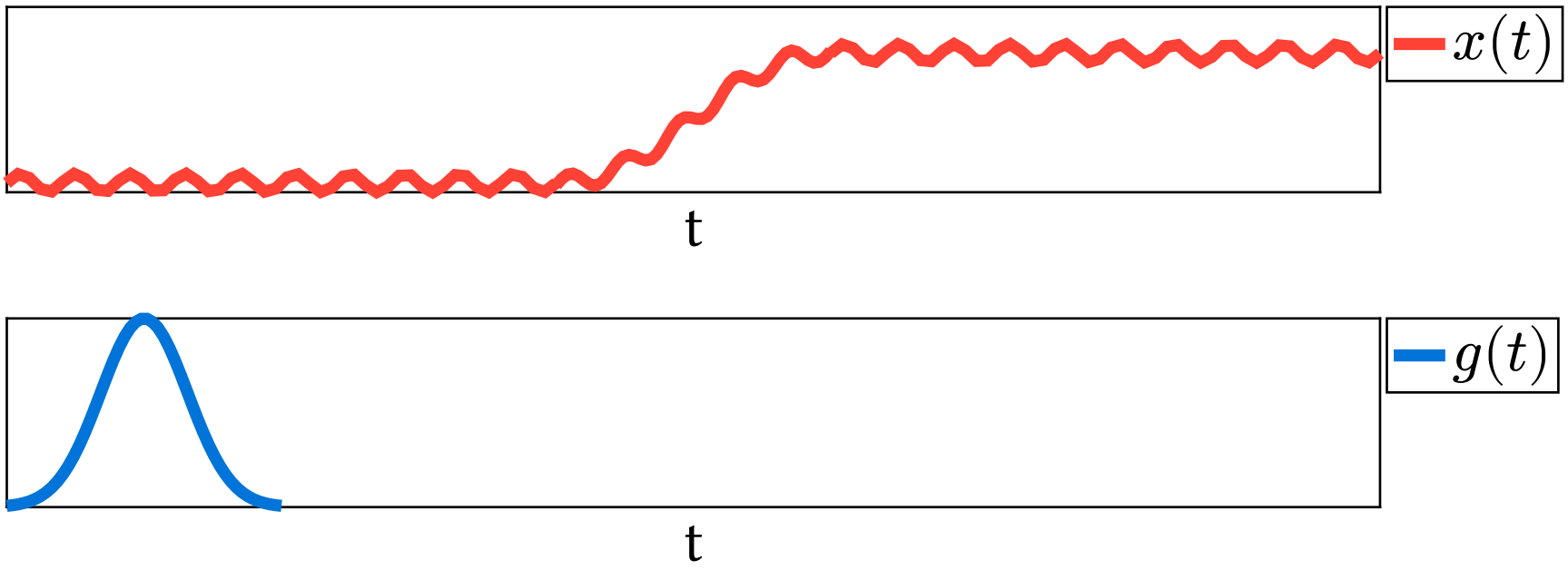
Convolution



Convolution

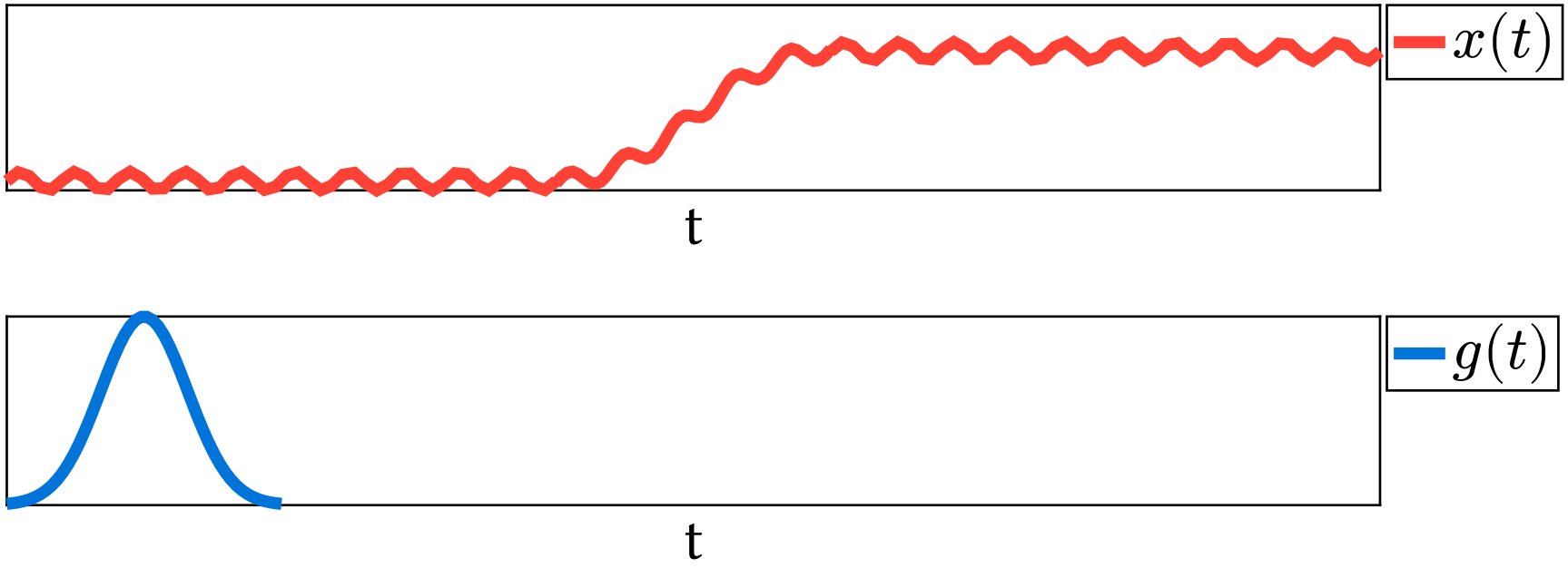


Convolution



Convolution is **local** to the filter $g(t)$

Convolution



Convolution is **local** to the filter $g(t)$

Convolution is also **equivariant** to time/space shifts

Convolution

Often, we use continuous time/space convolution for analog signals

Convolution

Often, we use continuous time/space convolution for analog signals

- Physics

Convolution

Often, we use continuous time/space convolution for analog signals

- Physics
- Control theory

Convolution

Often, we use continuous time/space convolution for analog signals

- Physics
- Control theory
- Electrical engineering

Convolution

Often, we use continuous time/space convolution for analog signals

- Physics
- Control theory
- Electrical engineering

Almost all deep learning occurs on digital hardware (discrete time/space)

Convolution

Often, we use continuous time/space convolution for analog signals

- Physics
- Control theory
- Electrical engineering

Almost all deep learning occurs on digital hardware (discrete time/space)

- Images

Convolution

Often, we use continuous time/space convolution for analog signals

- Physics
- Control theory
- Electrical engineering

Almost all deep learning occurs on digital hardware (discrete time/space)

- Images
- Recorded audio

Convolution

Often, we use continuous time/space convolution for analog signals

- Physics
- Control theory
- Electrical engineering

Almost all deep learning occurs on digital hardware (discrete time/space)

- Images
- Recorded audio
- Anything stored as bits

Convolution

Often, we use continuous time/space convolution for analog signals

- Physics
- Control theory
- Electrical engineering

Almost all deep learning occurs on digital hardware (discrete time/space)

- Images
- Recorded audio
- Anything stored as bits

But it is good to know both! Continuous variables for theory. Discrete variables for software

Convolution

$$\begin{bmatrix} x(t) \\ g(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & & & \end{bmatrix}$$

Convolution

$$\begin{bmatrix} x(t) \\ g(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & & & \\ 4 & & & & \end{bmatrix}$$

Convolution

$$\begin{bmatrix} x(t) \\ g(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ & 2 & 1 & & \\ 4 & 7 & & & \end{bmatrix}$$

Convolution

$$\begin{bmatrix} x(t) \\ g(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ & & 2 & 1 & \\ 4 & 5 & 10 & & \end{bmatrix}$$

Convolution

$$\begin{bmatrix} x(t) \\ g(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ & & & 2 & 1 \\ 4 & 5 & 10 & 13 & \end{bmatrix}$$

Convolution

$$\begin{bmatrix} x(t) \\ g(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & & & \\ 4 & 5 & 10 & 13 & \end{bmatrix}$$

Convolution

$$\begin{bmatrix} x(t) \\ g(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & & & \\ 4 & 5 & 10 & 13 & \end{bmatrix}$$

Question: Does anybody see a connection to neural networks?

Convolution

$$\begin{bmatrix} x(t) \\ g(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & & & \\ 4 & 5 & 10 & 13 & \end{bmatrix}$$

Question: Does anybody see a connection to neural networks?

Hint: What if I rewrite the filter?

Convolution

$$\begin{bmatrix} x(t) \\ g(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & & & \\ 4 & 5 & 10 & 13 & \end{bmatrix}$$

Question: Does anybody see a connection to neural networks?

Hint: What if I rewrite the filter?

$$\begin{bmatrix} x(t) \\ g(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ \theta_2 & \theta_1 & & & \\ & & & & \end{bmatrix}$$

Convolution

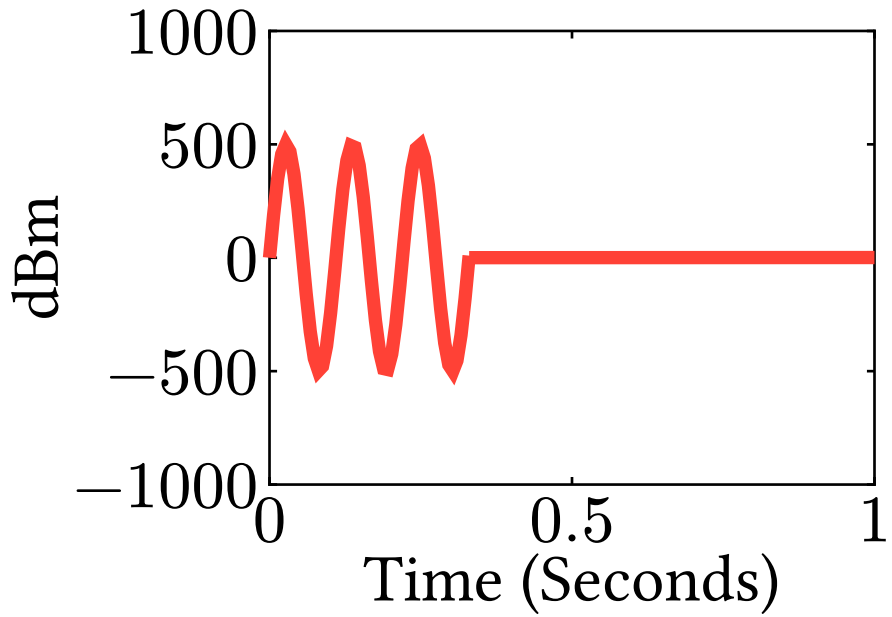
$$\begin{bmatrix} x(t) \\ g(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ \theta_2 & \theta_1 & & & \\ \theta_2 + 2\theta_1 & & & & \end{bmatrix}$$

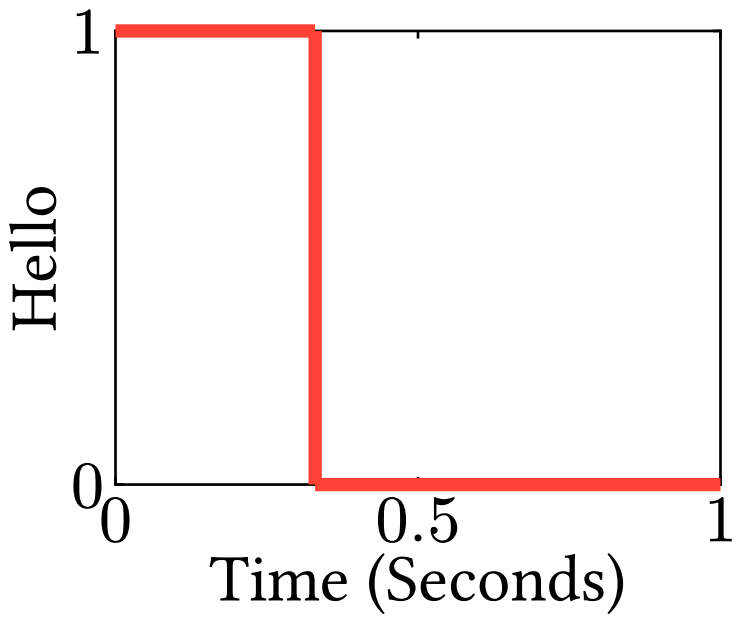
Convolution

$$\begin{bmatrix} x(t) \\ g(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ & \theta_2 & \theta_1 & & \\ \theta_2 + 2\theta_1 & 2\theta_2 + 3\theta_1 & & & \end{bmatrix}$$

Convolution

$$\begin{bmatrix} x(t) \\ g(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ & \theta_2 & \theta_1 & & \\ \theta_2 + 2\theta_1 & 2\theta_2 + 3\theta_1 & & & \end{bmatrix}$$





Convolution

$$\begin{bmatrix} x(t) \\ g(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ & \theta_2 & \theta_1 & & \\ \theta_2 + 2\theta_1 & 2\theta_2 + 3\theta_1 & & & \end{bmatrix}$$

Convolution

$$\begin{bmatrix} x(t) \\ g(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ & \theta_2 & \theta_1 & & \\ \theta_2 + 2\theta_1 & 2\theta_2 + 3\theta_1 & & & \end{bmatrix}$$

Just like neural networks, convolution is a linear operation

Convolution

$$\begin{bmatrix} x(t) \\ g(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ & \theta_2 & \theta_1 & & \\ \theta_2 + 2\theta_1 & 2\theta_2 + 3\theta_1 & & & \end{bmatrix}$$

Just like neural networks, convolution is a linear operation

It is a weighted sum of the inputs, just like a neuron

Convolution

$$\begin{bmatrix} x(t) \\ g(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ & \theta_2 & \theta_1 & & \\ \theta_2 + 2\theta_1 & 2\theta_2 + 3\theta_1 & & & \end{bmatrix}$$

Just like neural networks, convolution is a linear operation

It is a weighted sum of the inputs, just like a neuron

Question: How does convolution differ from a neuron?

Convolution

$$\begin{bmatrix} x(t) \\ g(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ & \theta_2 & \theta_1 & & \\ \theta_2 + 2\theta_1 & 2\theta_2 + 3\theta_1 & & & \end{bmatrix}$$

Just like neural networks, convolution is a linear operation

It is a weighted sum of the inputs, just like a neuron

Question: How does convolution differ from a neuron?

Answer: In a neuron, each input x_i has a different parameter θ_i. In convolution, we reuse (slide) θ_i over $x_1, x_2, \ldots$

Agenda

1. Review
2. Signal Processing
3. **Convolution**
4. Convolutional Neural Networks
5. Additional Dimensions
6. Deeper Networks
7. Coding

Agenda

1. Review
2. Signal Processing
3. Convolution
4. **Convolutional Neural Networks**
5. Additional Dimensions
6. Deeper Networks
7. Coding

Convolutional Neural Networks

Convolutional neural networks (CNNs) use convolutional layers

Convolutional Neural Networks

Convolutional neural networks (CNNs) use convolutional layers

Their translation equivariance and locality make them very efficient

Convolutional Neural Networks

Convolutional neural networks (CNNs) use convolutional layers

Their translation equivariance and locality make them very efficient

They also scale to variable-length sequences

Convolutional Neural Networks

Convolutional neural networks (CNNs) use convolutional layers

Their translation equivariance and locality make them very efficient

They also scale to variable-length sequences

Efficiently expands neural networks to images, videos, sounds, etc

Convolutional Neural Networks

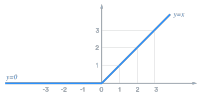
CNNs have been around since the 1980's

Convolutional Neural Networks

CNNs have been around since the 1980's

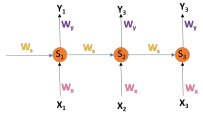
Linear Regression (1795)

Hebbian Learning (1949)



ReLU Activation Function (1969)

Convolutional Neural Network (1980)



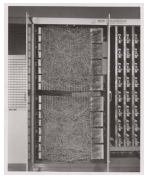
Backpropagation Through Time (1990)

Deep Q Network (2014)

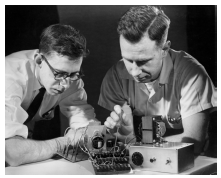
Transformer (2017)

Diffusion Model (2020)

Neural Network (1943)



Deep Neural Network (1958)



Recurrent Neural Network (1972)

Backpropagation (1982)

Alexnet (2012)

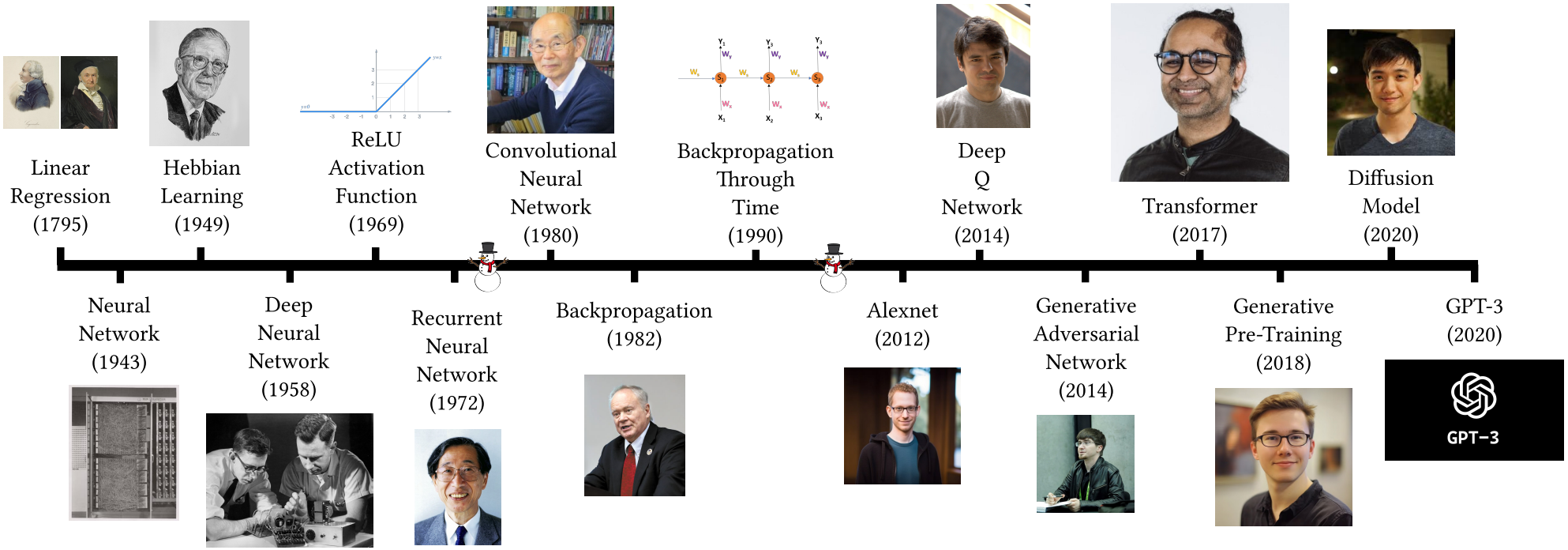
Generative Adversarial Network (2014)

Generative Pre-Training (2018)

GPT-3 (2020)

Convolutional Neural Networks

CNNs have been around since the 1980's



2012: GPU and CNN efficiency resulted in breakthroughs

Convolutional Neural Networks

So how does a convolutional neural network work?

Convolutional Neural Networks

Recall the neuron

Recall the neuron

Neuron for single x:

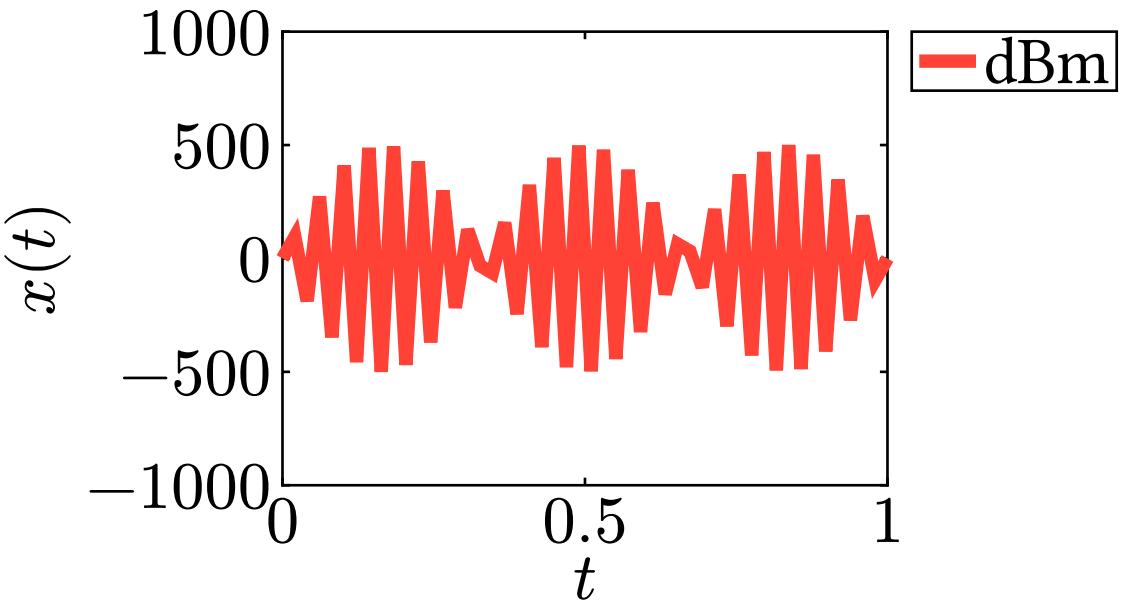
$$f(x, \boldsymbol{\theta}) = \sigma(\theta_1 x + \theta_0)$$

Convolutional Neural Networks

Recall the neuron

Neuron for single x:

$$f(x, \boldsymbol{\theta}) = \sigma(\theta_1 x + \theta_0)$$

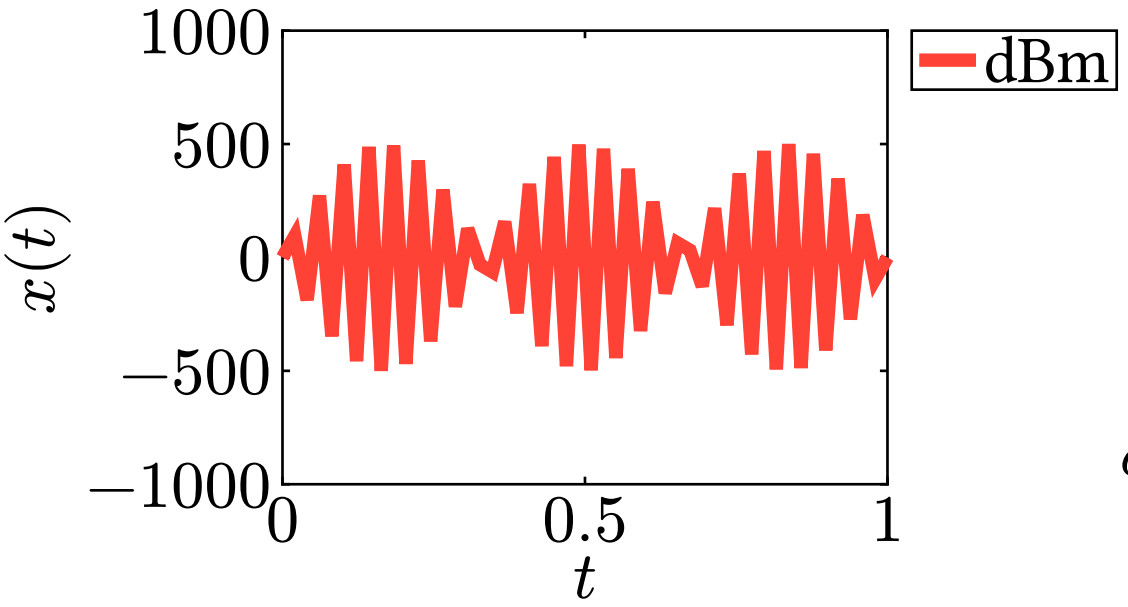


Convolutional Neural Networks

Recall the neuron

Neuron for single x:

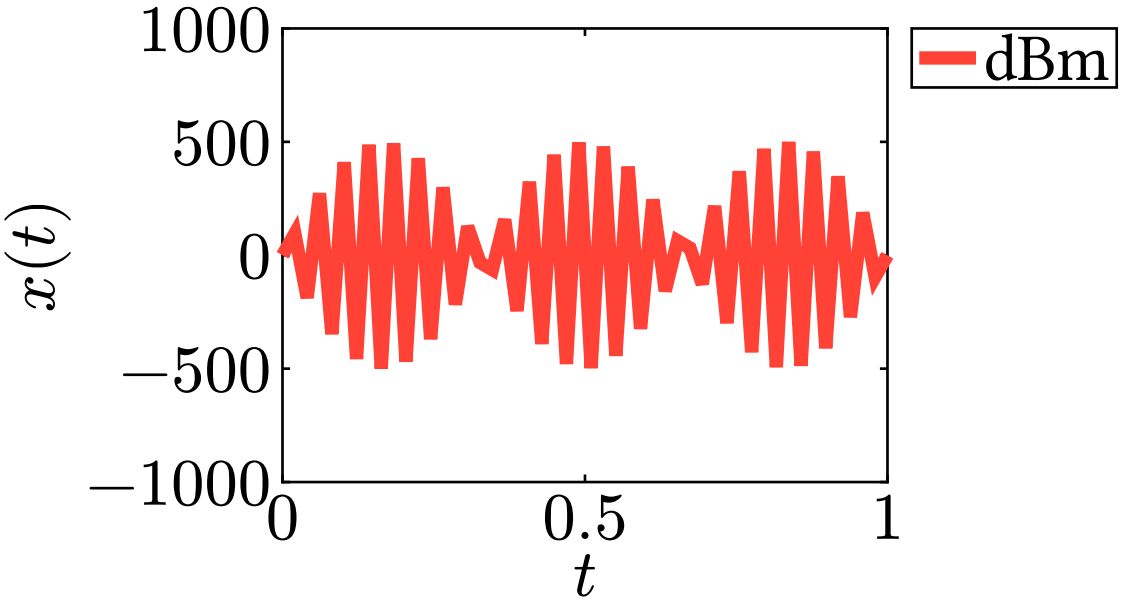
$$f(x, \boldsymbol{\theta}) = \sigma(\theta_1 x + \theta_0)$$



$$f\left(\begin{bmatrix} x(0.1) \\ x(0.2) \\ \vdots \end{bmatrix}, \boldsymbol{\theta}\right) =$$

$$\sigma(\theta_0 + \theta_1 x(0.1) + \theta_2 x(0.2) + ...)$$

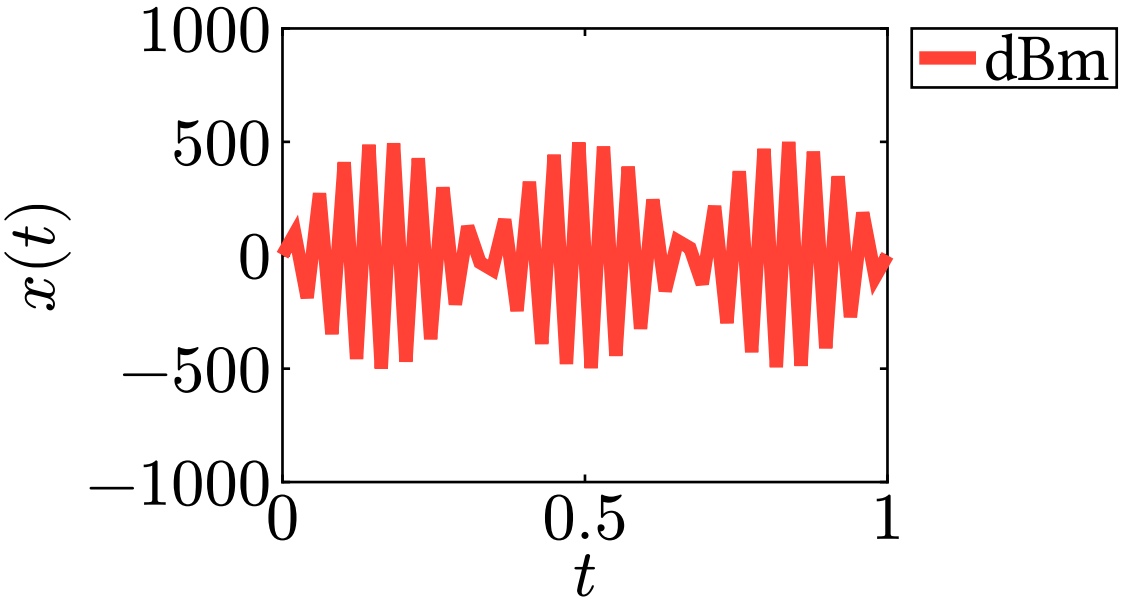
Convolutional Neural Networks



$$f\left(\begin{bmatrix} x(0.1) \\ x(0.2) \\ \vdots \end{bmatrix}, \boldsymbol{\theta}\right) =$$

$$\theta_0 + \theta_1 x(0.1) + \theta_2 x(0.2) + \ldots$$

Convolutional Neural Networks

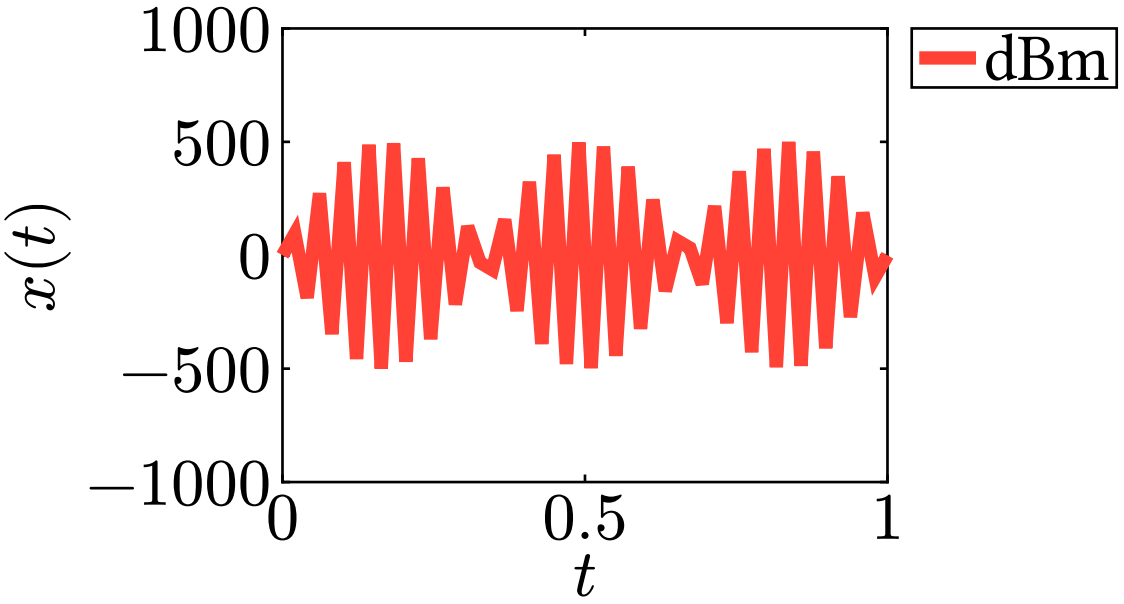


$$f\left(\begin{bmatrix} x(0.1) \\ x(0.2) \\ \vdots \end{bmatrix}, \boldsymbol{\theta}\right) =$$

$$\theta_0 + \theta_1 x(0.1) + \theta_2 x(0.2) + \ldots$$

Question: How many parameters do we need?

Convolutional Neural Networks



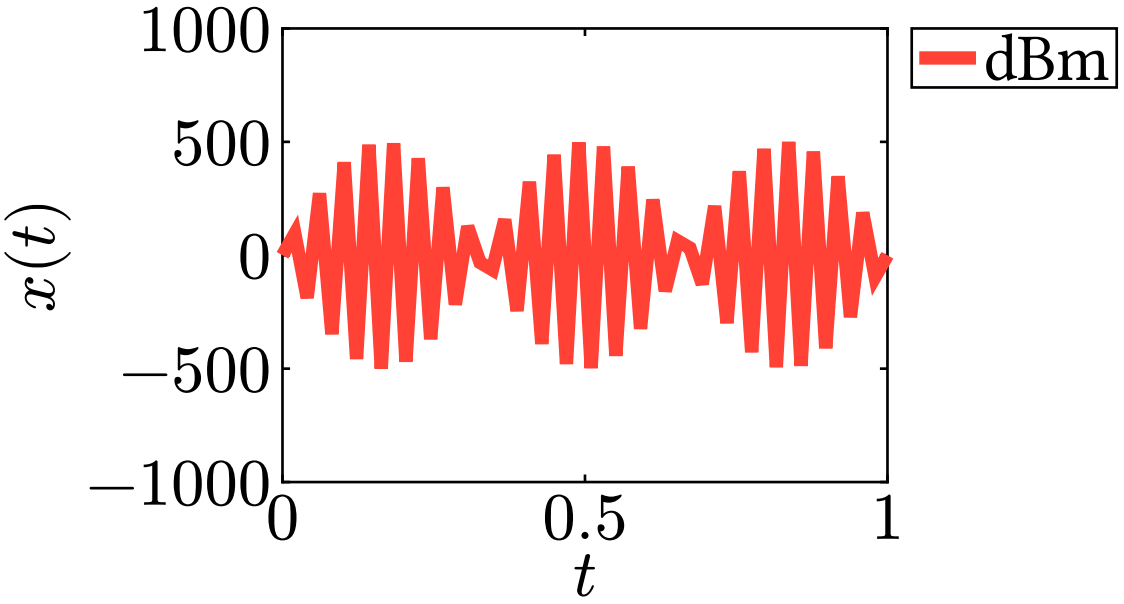
$$f\left(\begin{bmatrix} x(0.1) \\ x(0.2) \\ \vdots \end{bmatrix}, \boldsymbol{\theta}\right) =$$

$$\theta_0 + \theta_1 x(0.1) + \theta_2 x(0.2) + \ldots$$

Question: How many parameters do we need?

Answer 1: 10, parameters scale with sequence length

Convolutional Neural Networks



$$f\left(\begin{bmatrix} x(0.1) \\ x(0.2) \\ \vdots \end{bmatrix}, \boldsymbol{\theta}\right) =$$

$$\theta_0 + \theta_1 x(0.1) + \theta_2 x(0.2) + \ldots$$

Question: How many parameters do we need?

Answer 1: 10, parameters scale with sequence length

Question: What if our sequence is 1.1 seconds long?

Convolutional Neural Networks

One parameter for each timestep does not work well

Convolutional Neural Networks

One parameter for each timestep does not work well

$$f\left(\begin{bmatrix} x(0.1) \\ x(0.2) \\ \vdots \end{bmatrix}, \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \end{bmatrix}\right)$$

$$= \sigma(\theta_0 + \theta_1 x(0.1) + \theta_2 x(0.2) + \theta_3 x(0.3) + \theta_4 x(0.4) + \theta_5 x(0.5) + ...)$$

Convolutional Neural Networks

One parameter for each timestep does not work well

$$f\left(\begin{bmatrix} x(0.1) \\ x(0.2) \\ \vdots \end{bmatrix}, \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \end{bmatrix}\right)$$

$$= \sigma(\theta_0 + \theta_1 x(0.1) + \theta_2 x(0.2) + \theta_3 x(0.3) + \theta_4 x(0.4) + \theta_5 x(0.5) + ...)$$

Question: How many parameters if speech is 2 hours long?

Convolutional Neural Networks

One parameter for each timestep does not work well

$$f\left(\begin{bmatrix} x(0.1) \\ x(0.2) \\ \vdots \end{bmatrix}, \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \end{bmatrix}\right)$$

$$= \sigma(\theta_0 + \theta_1 x(0.1) + \theta_2 x(0.2) + \theta_3 x(0.3) + \theta_4 x(0.4) + \theta_5 x(0.5) + ...)$$

Question: How many parameters if speech is 2 hours long? 72,000

Question: What if speech is 2 hours and 0.1 seconds long?

Convolutional Neural Networks

One parameter for each timestep does not work well

$$f\left(\begin{bmatrix} x(0.1) \\ x(0.2) \\ \vdots \end{bmatrix}, \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \end{bmatrix}\right)$$

$$= \sigma(\theta_0 + \theta_1 x(0.1) + \theta_2 x(0.2) + \theta_3 x(0.3) + \theta_4 x(0.4) + \theta_5 x(0.5) + ...)$$

Question: How many parameters if speech is 2 hours long? 72,000

Question: What if speech is 2 hours and 0.1 seconds long?

Answer: Train a new neural network

Convolutional Neural Networks

$$f\left(\begin{bmatrix} x(0.1) \\ x(0.2) \\ \vdots \end{bmatrix}, \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \end{bmatrix}\right)$$

$$= \sigma(\theta_0 + \theta_1 x(0.1) + \theta_2 x(0.2) + \theta_3 x(0.3) + \theta_4 x(0.4) + \theta_5 x(0.5) + ...)$$

Convolutional Neural Networks

$$f\left(\begin{bmatrix} x(0.1) \\ x(0.2) \\ \vdots \end{bmatrix}, \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \end{bmatrix}\right)$$

$$= \sigma(\theta_0 + \theta_1 x(0.1) + \theta_2 x(0.2) + \theta_3 x(0.3) + \theta_4 x(0.4) + \theta_5 x(0.5) + ...)$$

What if we reuse parameters?

$$f\left(\begin{bmatrix} x(0.1) \\ x(0.2) \\ \vdots \end{bmatrix}, \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}\right) = \begin{bmatrix} \sigma(\theta_0 + \theta_1 x(0.1) + \theta_2 x(0.2)) \\ \sigma(\theta_0 + \theta_1 x(0.2) + \theta_2 x(0.3)) \\ \sigma(\theta_0 + \theta_1 x(0.3) + \theta_2 x(0.4)) \\ \vdots \end{bmatrix}$$

Convolutional Neural Networks

$$f\left(\begin{bmatrix} x(0.1) \\ x(0.2) \\ \vdots \end{bmatrix}, \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \end{bmatrix}\right)$$

$$= \sigma(\theta_0 + \theta_1 x(0.1) + \theta_2 x(0.2) + \theta_3 x(0.3) + \theta_4 x(0.4) + \theta_5 x(0.5) + ...)$$

What if we reuse parameters?

$$f\left(\begin{bmatrix} x(0.1) \\ x(0.2) \\ \vdots \end{bmatrix}, \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}\right) = \begin{bmatrix} \sigma(\theta_0 + \theta_1 x(0.1) + \theta_2 x(0.2)) \\ \sigma(\theta_0 + \theta_1 x(0.2) + \theta_2 x(0.3)) \\ \sigma(\theta_0 + \theta_1 x(0.3) + \theta_2 x(0.4)) \\ \vdots \end{bmatrix}$$

Convolutional Neural Networks

$$f\left(\begin{bmatrix} x(0.1) \\ x(0.2) \\ \vdots \end{bmatrix}, \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}\right) = \begin{bmatrix} \sigma(\theta_0 + \theta_1 x(0.1) + \theta_2 x(0.2)) \\ \sigma(\theta_0 + \theta_1 x(0.2) + \theta_2 x(0.3)) \\ \sigma(\theta_0 + \theta_1 x(0.3) + \theta_2 x(0.4)) \\ \vdots \end{bmatrix}$$

Convolutional Neural Networks

$$f\left(\begin{bmatrix} x(0.1) \\ x(0.2) \\ \vdots \end{bmatrix}, \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}\right) = \begin{bmatrix} \sigma(\theta_0 + \theta_1 x(0.1) + \theta_2 x(0.2)) \\ \sigma(\theta_0 + \theta_1 x(0.2) + \theta_2 x(0.3)) \\ \sigma(\theta_0 + \theta_1 x(0.3) + \theta_2 x(0.4)) \\ \vdots \end{bmatrix}$$

This is convolution, we slide our filter $g(t) = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}$ over the input $x(t)$

Convolutional Neural Networks

We can write both a perceptron and convolution in vector form

$$f(x(t), \boldsymbol{\theta}) = \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.1) \\ x(0.2) \\ \vdots \end{bmatrix}\right)$$

Convolutional Neural Networks

We can write both a perceptron and convolution in vector form

$$f(x(t), \boldsymbol{\theta}) = \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.1) \\ x(0.2) \\ \vdots \end{bmatrix}\right) \qquad f(x(t), \boldsymbol{\theta}) = \begin{bmatrix} \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.1) \\ x(0.2) \end{bmatrix}\right) \\ \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.2) \\ x(0.3) \end{bmatrix}\right) \\ \vdots \end{bmatrix}$$

A convolution layer applies a "mini" perceptron to every few timesteps

Convolutional Neural Networks

$$f(x(t), \boldsymbol{\theta}) = \begin{bmatrix} \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.1) \\ x(0.2) \end{bmatrix}\right) \\ \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.2) \\ x(0.3) \end{bmatrix}\right) \\ \vdots \end{bmatrix}$$

Convolutional Neural Networks

$$f(x(t), \boldsymbol{\theta}) = \begin{bmatrix} \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.1) \\ x(0.2) \end{bmatrix}\right) \\ \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.2) \\ x(0.3) \end{bmatrix}\right) \\ \vdots \end{bmatrix}$$

Local, only considers a few nearby inputs at a time

Convolutional Neural Networks

$$f(x(t), \boldsymbol{\theta}) = \begin{bmatrix} \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.1) \\ x(0.2) \end{bmatrix}\right) \\ \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.2) \\ x(0.3) \end{bmatrix}\right) \\ \vdots \end{bmatrix}$$

Local, only considers a few nearby inputs at a time

Translation equivariant, each output corresponds to input

Convolutional Neural Networks

$$f(x(t), \boldsymbol{\theta}) = \begin{bmatrix} \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.1) \\ x(0.2) \end{bmatrix}\right) \\ \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.2) \\ x(0.3) \end{bmatrix}\right) \\ \vdots \end{bmatrix}$$

Local, only considers a few nearby inputs at a time

Translation equivariant, each output corresponds to input

Convolutional Neural Networks

$$f(x(t), \boldsymbol{\theta}) = \begin{bmatrix} \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.1) \\ x(0.2) \end{bmatrix}\right) \\ \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.2) \\ x(0.3) \end{bmatrix}\right) \\ \vdots \end{bmatrix}$$

Convolutional Neural Networks

$$f(x(t), \boldsymbol{\theta}) = \begin{bmatrix} \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.1) \\ x(0.2) \end{bmatrix}\right) \\ \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.2) \\ x(0.3) \end{bmatrix}\right) \\ \vdots \end{bmatrix}$$

Question: What is the shape of the results?

Convolutional Neural Networks

$$f(x(t), \boldsymbol{\theta}) = \begin{bmatrix} \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.1) \\ x(0.2) \end{bmatrix}\right) \\ \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.2) \\ x(0.3) \end{bmatrix}\right) \\ \vdots \end{bmatrix}$$

Question: What is the shape of the results?

Answer 1: Depends on sequence length and filter size!

Convolutional Neural Networks

$$f(x(t), \boldsymbol{\theta}) = \begin{bmatrix} \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.1) \\ x(0.2) \end{bmatrix}\right) \\ \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.2) \\ x(0.3) \end{bmatrix}\right) \\ \vdots \end{bmatrix}$$

Question: What is the shape of the results?

Answer 1: Depends on sequence length and filter size!

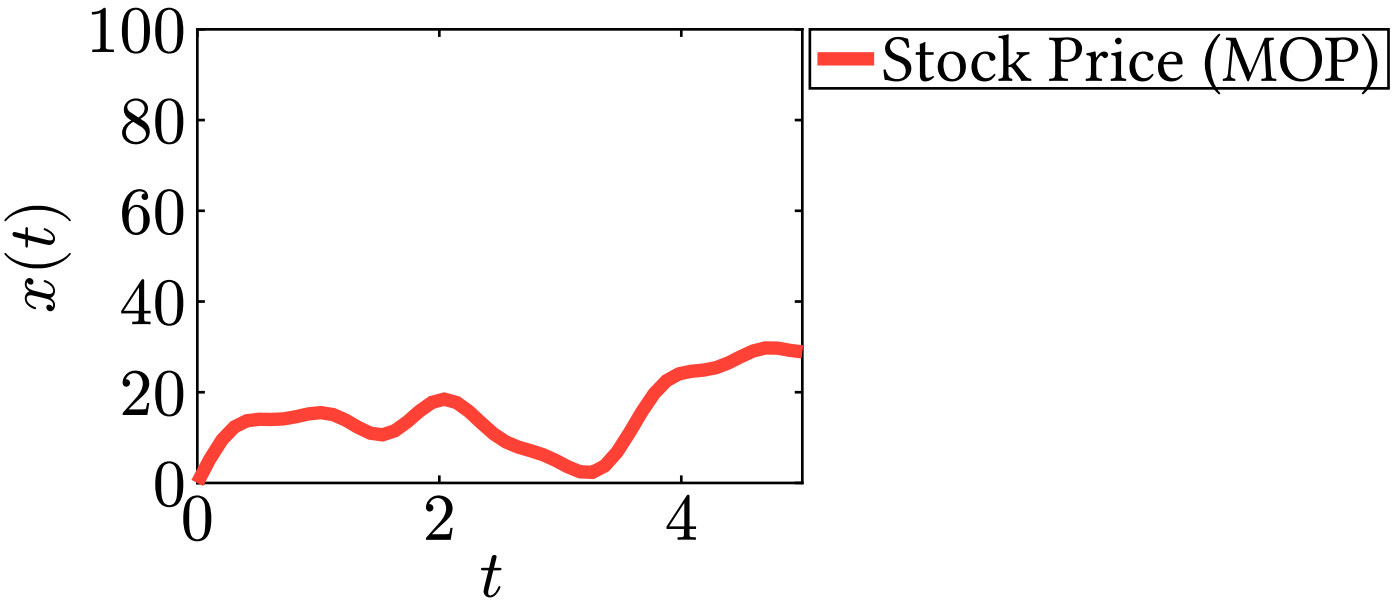
Answer 2: $T - k$, where T is sequence length and k is filter length

Convolutional Neural Networks

Maybe we want to predict a single output for a sequence

Convolutional Neural Networks

Maybe we want to predict a single output for a sequence



Convolutional Neural Networks

If we want a single output, we should **pool**

Convolutional Neural Networks

If we want a single output, we should **pool**

$$z(t) = f(x(t), \boldsymbol{\theta}) = \begin{bmatrix} \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.1) \\ x(0.2) \end{bmatrix}\right) & \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.2) \\ x(0.3) \end{bmatrix}\right) & \dots \end{bmatrix}^\top$$

Convolutional Neural Networks

If we want a single output, we should **pool**

$$z(t) = f(x(t), \boldsymbol{\theta}) = \begin{bmatrix} \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.1) \\ x(0.2) \end{bmatrix}\right) & \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.2) \\ x(0.3) \end{bmatrix}\right) & \dots \end{bmatrix}^\top$$

$$\text{SumPool}(z(t)) = \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.1) \\ x(0.2) \end{bmatrix}\right) + \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.2) \\ x(0.3) \end{bmatrix}\right) + \dots$$

Convolutional Neural Networks

If we want a single output, we should **pool**

$$z(t) = f(x(t), \boldsymbol{\theta}) = \left[\sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.1) \\ x(0.2) \end{bmatrix} \right) \quad \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.2) \\ x(0.3) \end{bmatrix} \right) \quad \ldots \right]^\top$$

$$\text{SumPool}(z(t)) = \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.1) \\ x(0.2) \end{bmatrix} \right) + \sigma\left(\boldsymbol{\theta}^\top \begin{bmatrix} 1 \\ x(0.2) \\ x(0.3) \end{bmatrix} \right) + \ldots$$

$$\text{MeanPool}(z(t)) = \frac{1}{T-k}\,\text{SumPool}(z(t)); \quad \text{MaxPool}(z(t)) = \max(z(t))$$

Question: $x(t)$ is a function, what is the function signature?

Question: $x(t)$ is a function, what is the function signature?

Answer:

$$x : \mathbb{R}_+ \mapsto \mathbb{R}$$

Question: $x(t)$ is a function, what is the function signature?

Answer:

$$x : \mathbb{R}_+ \mapsto \mathbb{R}$$

So far, we have considered:

Question: $x(t)$ is a function, what is the function signature?

Answer:

$$x : \mathbb{R}_+ \mapsto \mathbb{R}$$

So far, we have considered:

- 1 dimensional variable t

Question: $x(t)$ is a function, what is the function signature?

Answer:

$$x : \mathbb{R}_+ \mapsto \mathbb{R}$$

So far, we have considered:

- 1 dimensional variable t
- 1 dimensional output/channel $x(t)$

Question: $x(t)$ is a function, what is the function signature?

Answer:

$$x : \mathbb{R}_+ \mapsto \mathbb{R}$$

So far, we have considered:

- 1 dimensional variable t
- 1 dimensional output/channel $x(t)$
- 1 filter

Question: $x(t)$ is a function, what is the function signature?

Answer:

$$x : \mathbb{R}_+ \mapsto \mathbb{R}$$

So far, we have considered:

- 1 dimensional variable t
- 1 dimensional output/channel $x(t)$
- 1 filter

We must consider a more general case

Question: $x(t)$ is a function, what is the function signature?

Answer:

$$x : \mathbb{R}_+ \mapsto \mathbb{R}$$

So far, we have considered:

- 1 dimensional variable t
- 1 dimensional output/channel $x(t)$
- 1 filter

We must consider a more general case

Things will get more complicated, but the core idea is exactly the same

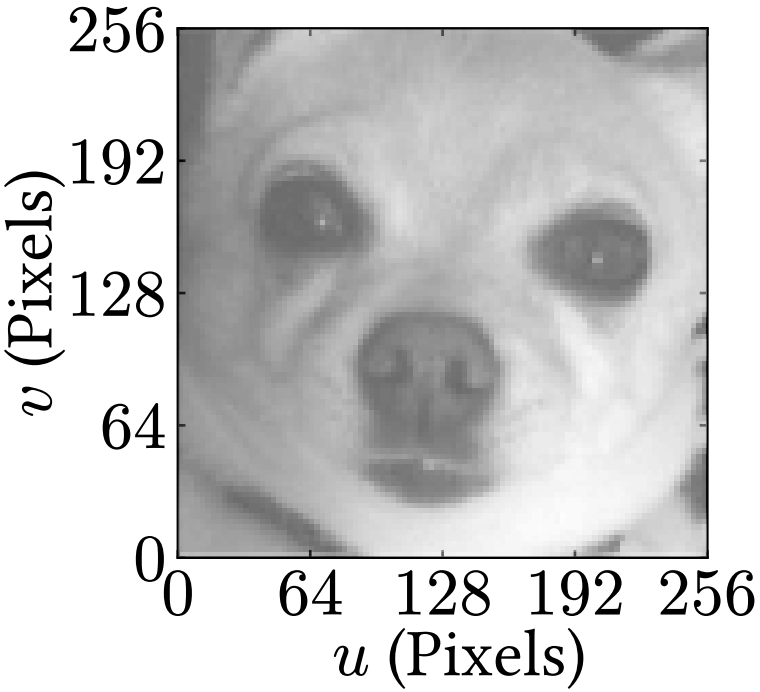
Agenda

1. Review
2. Signal Processing
3. Convolution
4. **Convolutional Neural Networks**
5. Additional Dimensions
6. Deeper Networks
7. Coding

Agenda

1. Review
2. Signal Processing
3. Convolution
4. Convolutional Neural Networks
5. **Additional Dimensions**
6. Deeper Networks
7. Coding

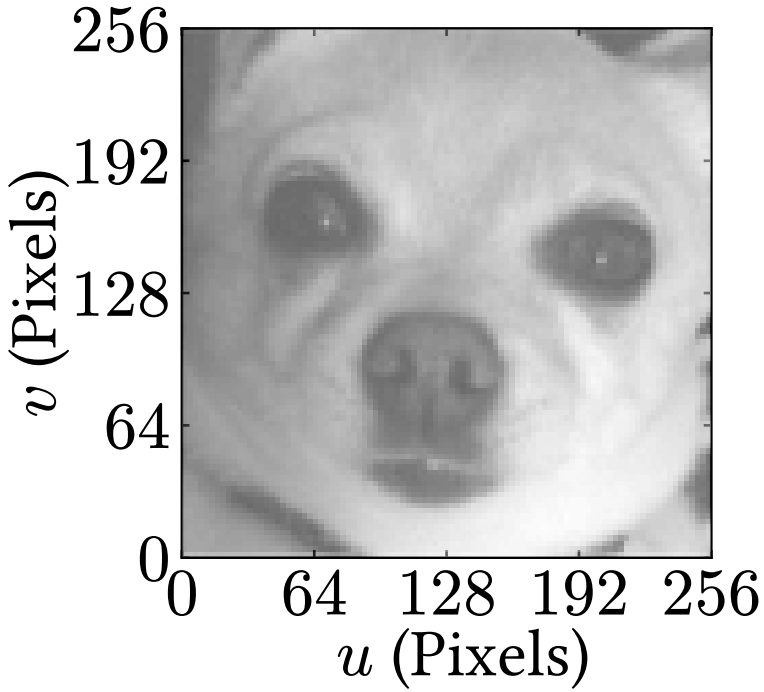
Additional Dimensions



$x(u, v)$

Question: How many input dimensions for x?

Additional Dimensions



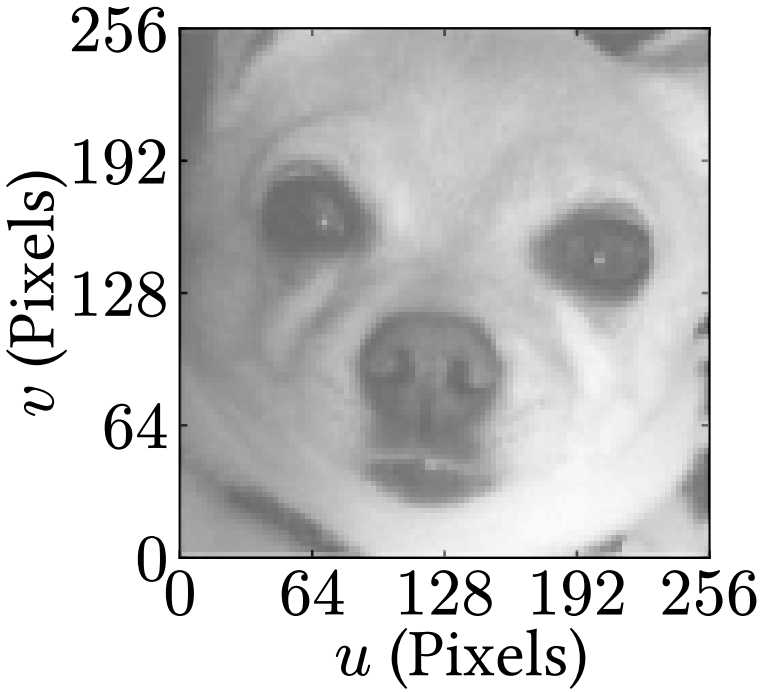
$x(u, v)$

Question: How many input dimensions for x?

Answer: 2: u, v

Question: How many output dimensions/channels for x?

Additional Dimensions



$$x(u, v)$$

Question: How many input dimensions for x?

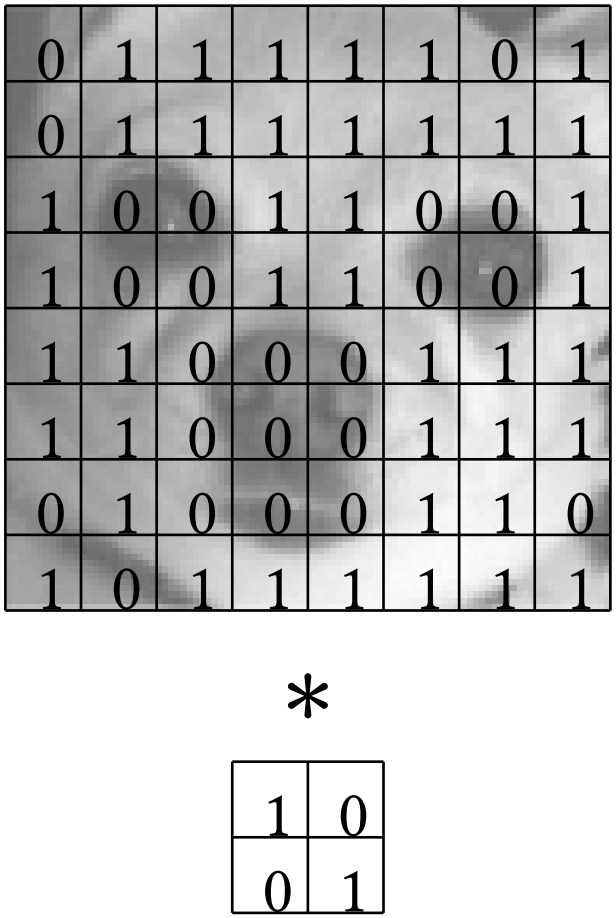
Answer: 2: u, v

Question: How many output dimensions/channels for x?

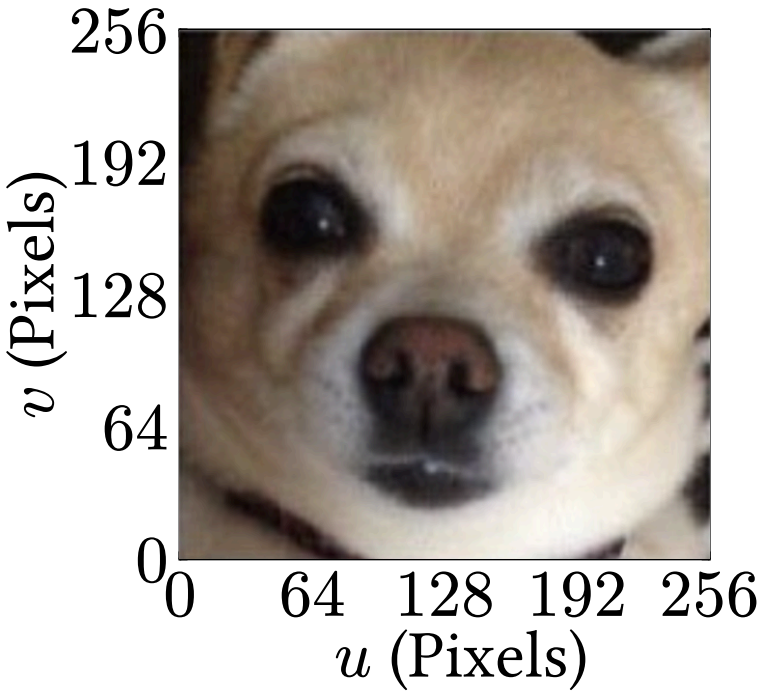
Answer: 1, black/white value

$$x : \underbrace{\mathbb{Z}_{0,255}}_{\text{width}} \times \underbrace{\mathbb{Z}_{0,255}}_{\text{height}} \mapsto \underbrace{[0, 1]}_{\text{Color values}}$$

Additional Dimensions



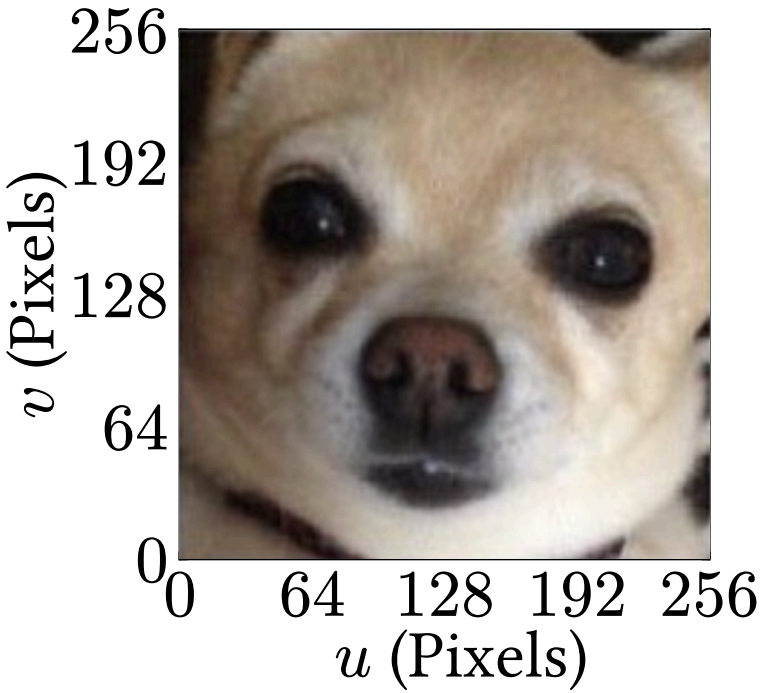
Additional Dimensions



$$x(u, v)$$

Question: How many input dimensions for x?

Additional Dimensions



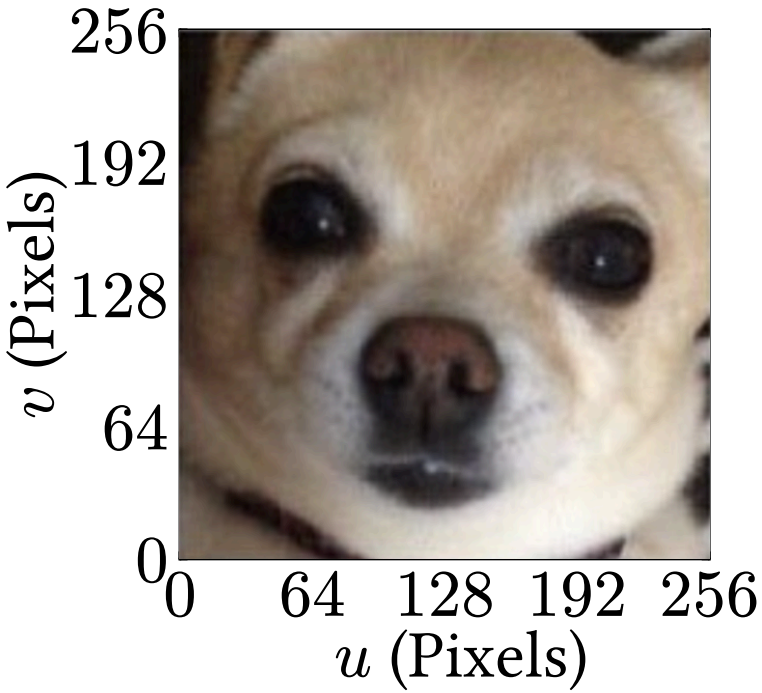
$x(u, v)$

Question: How many input dimensions for x?

Answer: 2: u, v

Question: How many output dimensions for x?

Additional Dimensions



$$x(u, v)$$

Question: How many input dimensions for x?

Answer: 2: u, v

Question: How many output dimensions for x?

Answer: 3 – red, green, and blue channels

$$x : \underbrace{\mathbb{Z}_{0,255}}_{\text{width}} \times \underbrace{\mathbb{Z}_{0,255}}_{\text{height}} \mapsto \underbrace{[0,1]^3}_{\text{Color values}}$$

Additional Dimensions

Additional Dimensions

Computers represent 3 color channel each with 256 integer values

Additional Dimensions

Computers represent 3 color channel each with 256 integer values

But we usually convert the colors $x(u, v) \in [0, 1]$ for scale reasons

Additional Dimensions

Computers represent 3 color channel each with 256 integer values

But we usually convert the colors $x(u, v) \in [0, 1]$ for scale reasons

$$\left[\frac{R}{255} \quad \frac{G}{255} \quad \frac{B}{255}\right]$$

Additional Dimensions

The pixels extend in 2 directions (input) $x(u, v)$

Additional Dimensions

The pixels extend in 2 directions (input) $x(u, v)$

Each pixel contains 3 colors (channels) $x(u, v) = \begin{bmatrix} r & g & b \end{bmatrix}^\top$

Additional Dimensions

The pixels extend in 2 directions (input) $x(u, v)$

Each pixel contains 3 colors (channels) $x(u, v) = \begin{bmatrix} r & g & b \end{bmatrix}^\top$

And the pixels extend in 2 directions (variables)

Additional Dimensions

The pixels extend in 2 directions (input) $x(u, v)$

Each pixel contains 3 colors (channels) $x(u, v) = \begin{bmatrix} r & g & b \end{bmatrix}^\top$

And the pixels extend in 2 directions (variables)

$$\boldsymbol{x}(u, v) = \left[\underbrace{\begin{bmatrix} .13 & .14 & .12 & .10 \\ .80 & .40 & .20 & .05 \\ .15 & .08 & .75 & .16 \\ .21 & .38 & .90 & .78 \end{bmatrix}}_{\text{red}} \underbrace{\begin{bmatrix} .15 & .08 & .75 & .16 \\ .80 & .40 & .20 & .05 \\ .21 & .38 & .90 & .78 \\ .13 & .14 & .12 & .10 \end{bmatrix}}_{\text{green}} \underbrace{\begin{bmatrix} .80 & .40 & .20 & .05 \\ .21 & .38 & .90 & .78 \\ .15 & .08 & .75 & .16 \\ .13 & .14 & .12 & .10 \end{bmatrix}}_{\text{blue}} \right]^\top$$

Additional Dimensions

The pixels extend in 2 directions (input) $x(u, v)$

Each pixel contains 3 colors (channels) $x(u, v) = \begin{bmatrix} r & g & b \end{bmatrix}^\top$

And the pixels extend in 2 directions (variables)

$$\boldsymbol{x}(u, v) = \left[\underbrace{\begin{bmatrix} .13 & .14 & .12 & .10 \\ .80 & .40 & .20 & .05 \\ .15 & .08 & .75 & .16 \\ .21 & .38 & .90 & .78 \end{bmatrix}}_{\text{red}} \underbrace{\begin{bmatrix} .15 & .08 & .75 & .16 \\ .80 & .40 & .20 & .05 \\ .21 & .38 & .90 & .78 \\ .13 & .14 & .12 & .10 \end{bmatrix}}_{\text{green}} \underbrace{\begin{bmatrix} .80 & .40 & .20 & .05 \\ .21 & .38 & .90 & .78 \\ .15 & .08 & .75 & .16 \\ .13 & .14 & .12 & .10 \end{bmatrix}}_{\text{blue}} \right]^\top$$

This form is called CHW (channel, height, width) format

Additional Dimensions

The pixels extend in 2 directions (input) $x(u, v)$

Each pixel contains 3 colors (channels) $x(u, v) = \begin{bmatrix} r & g & b \end{bmatrix}^\top$

And the pixels extend in 2 directions (variables)

$$\boldsymbol{x}(u, v) = \left[\underbrace{\begin{bmatrix} .13 & .14 & .12 & .10 \\ .80 & .40 & .20 & .05 \\ .15 & .08 & .75 & .16 \\ .21 & .38 & .90 & .78 \end{bmatrix}}_{\text{red}} \underbrace{\begin{bmatrix} .15 & .08 & .75 & .16 \\ .80 & .40 & .20 & .05 \\ .21 & .38 & .90 & .78 \\ .13 & .14 & .12 & .10 \end{bmatrix}}_{\text{green}} \underbrace{\begin{bmatrix} .80 & .40 & .20 & .05 \\ .21 & .38 & .90 & .78 \\ .15 & .08 & .75 & .16 \\ .13 & .14 & .12 & .10 \end{bmatrix}}_{\text{blue}} \right]^\top$$

This form is called CHW (channel, height, width) format

Additional Dimensions

0	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1
1	0	0	1	1	0	0	1
1	0	0	1	1	0	0	1
1	1	0	0	0	1	1	1
1	1	0	0	0	1	1	1
0	1	0	0	0	1	1	0
1	0	1	1	1	1	1	1

$*$

2	0
0	1

$+$

0	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1
1	0	0	1	1	0	0	1
1	0	0	1	1	0	0	1
1	1	0	0	0	1	1	1
1	1	0	0	0	1	1	1
0	1	0	0	0	1	1	0
1	0	1	1	1	1	1	1

$*$

1	1
0	1

$+$

0	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1
1	0	0	1	1	0	0	1
1	0	0	1	1	0	0	1
1	1	0	0	0	1	1	1
1	1	0	0	0	1	1	1
0	1	0	0	0	1	1	0
1	0	1	1	1	1	1	1

$*$

1	0
0	0

Additional Dimensions

I will not bore you with the full equations

Additional Dimensions

I will not bore you with the full equations

Question: What is the shape of $\boldsymbol{\theta}$ for a single layer?

Additional Dimensions

I will not bore you with the full equations

Question: What is the shape of $\boldsymbol{\theta}$ for a single layer?

Answer:

$$\boldsymbol{\theta} \in \mathbb{R}^{c_x \times c_y \times k \times k + c_y}$$

- Input channels: c_x
- Output channels: c_y
- Filter u (height): $k + 1$
- Filter v (width): k

Additional Dimensions

I will not bore you with the full equations

Question: What is the shape of $\boldsymbol{\theta}$ for a single layer?

Answer:

$$\boldsymbol{\theta} \in \mathbb{R}^{c_x \times c_y \times k \times k + c_y}$$

- Input channels: c_x
- Output channels: c_y
- Filter u (height): $k + 1$
- Filter v (width): k

Convolve c_y filters of size $k \times k$ across c_x channels, bias for each output

Additional Dimensions

One last thing, **stride** allows you to "skip" cells during convolution

Additional Dimensions

One last thing, **stride** allows you to "skip" cells during convolution

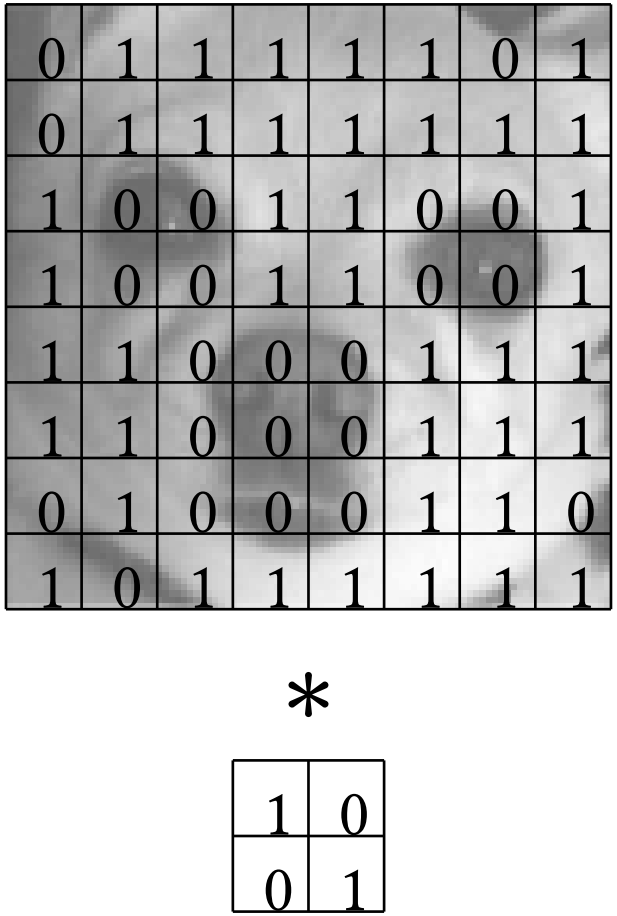
This can decrease the size of image without pooling

Additional Dimensions

One last thing, **stride** allows you to "skip" cells during convolution

This can decrease the size of image without pooling

Padding adds zero pixels to the image to increase the output size



Agenda

1. Review
2. Signal Processing
3. Convolution
4. Convolutional Neural Networks
5. **Additional Dimensions**
6. Deeper Networks
7. Coding

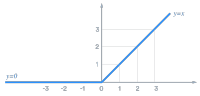
Agenda

1. Review
2. Signal Processing
3. Convolution
4. Convolutional Neural Networks
5. Additional Dimensions
6. **Deeper Networks**
7. Coding

Deeper Networks

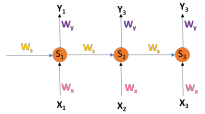
Linear Regression (1795)

Hebbian Learning (1949)



ReLU Activation Function (1969)

Convolutional Neural Network (1980)



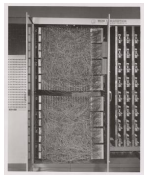
Backpropagation Through Time (1990)

Deep Q Network (2014)

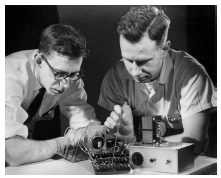
Transformer (2017)

Diffusion Model (2020)

Neural Network (1943)



Deep Neural Network (1958)



Recurrent Neural Network (1972)

Backpropagation (1982)

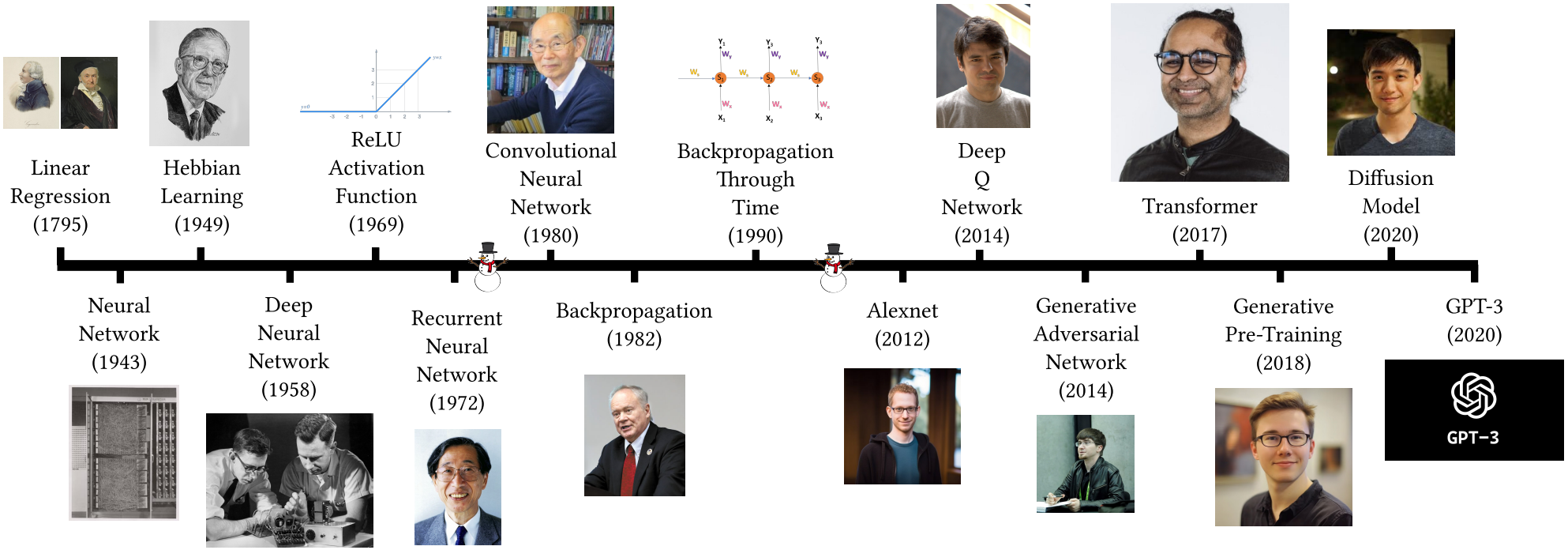
Alexnet (2012)

Generative Adversarial Network (2014)

Generative Pre-Training (2018)

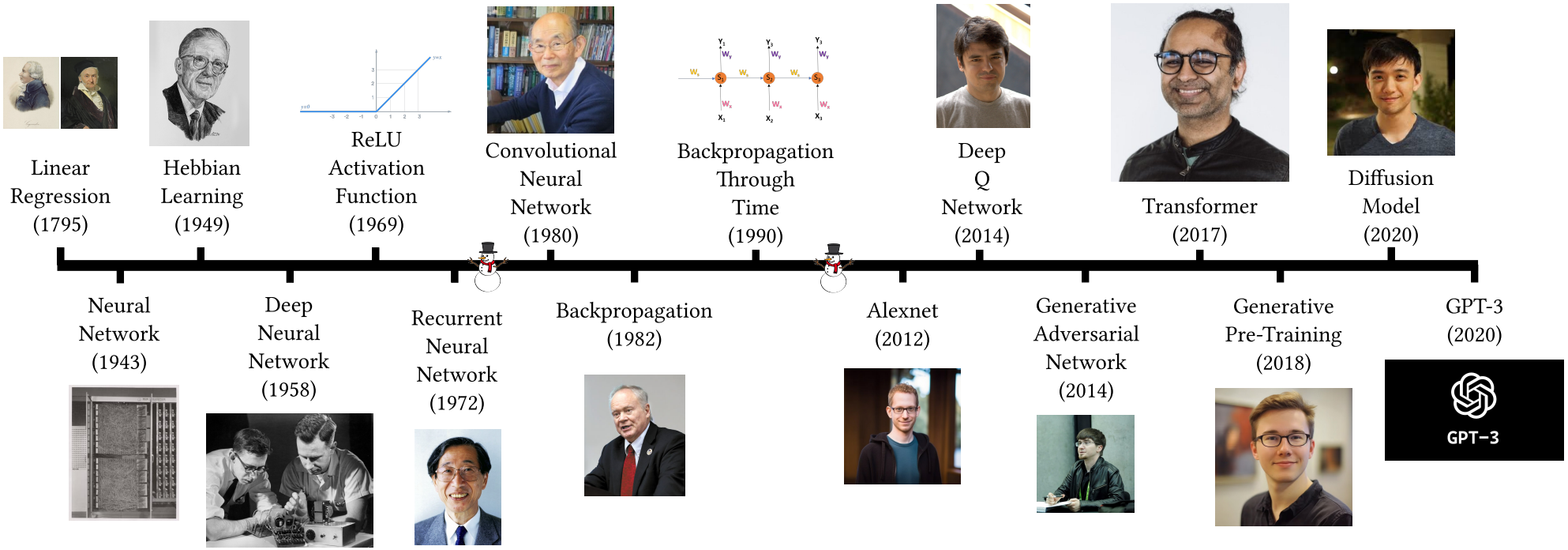
GPT-3 (2020)

Deeper Networks



AlexNet: Train a neural network on a GPU ($n = 1.2m$, 6 days)

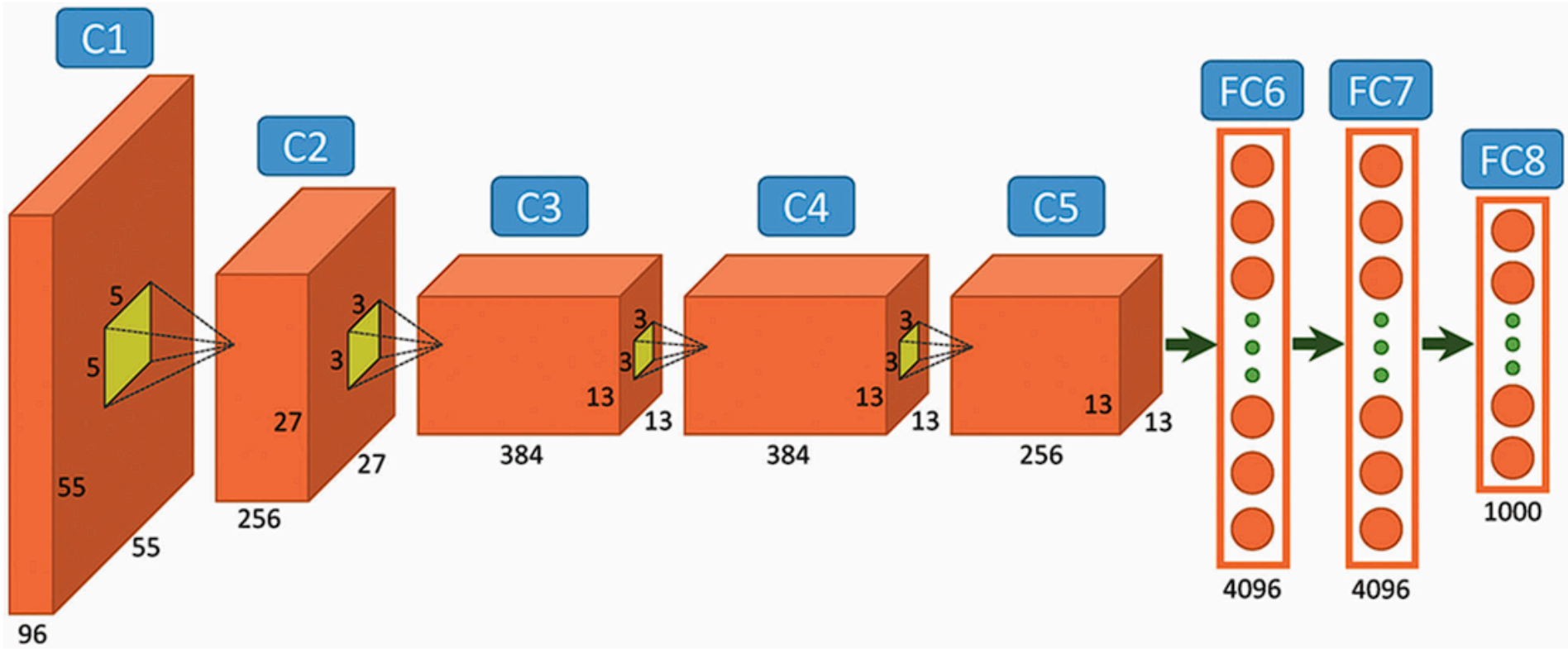
Deeper Networks



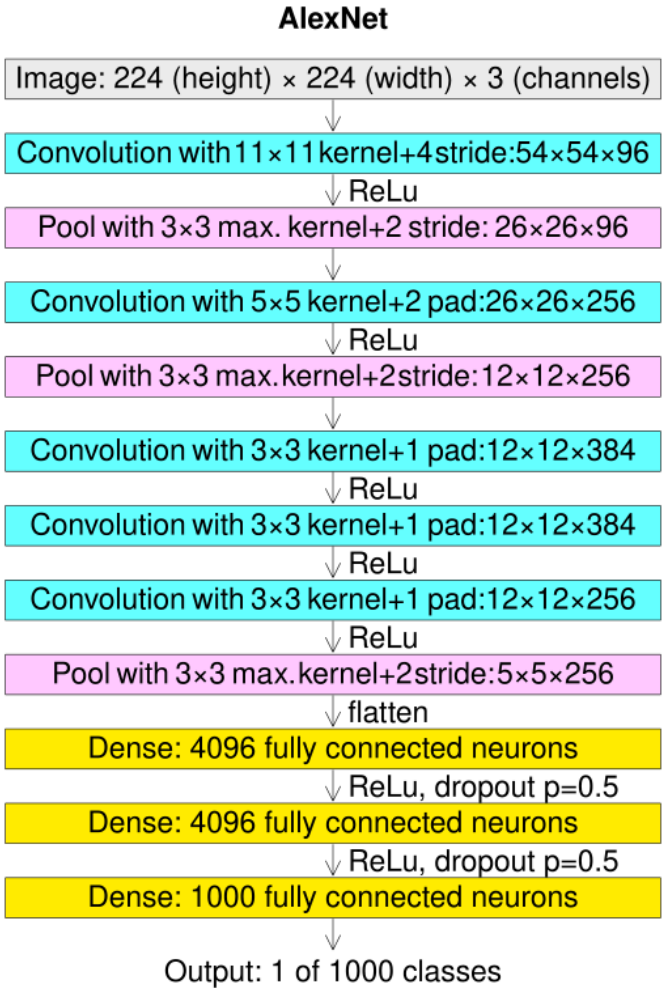
AlexNet: Train a neural network on a GPU ($n = 1.2m$, 6 days)

Paper by Krizhevsky, Sutskever (OpenAI CSO), and Hinton (Nobel)

Deeper Networks



Deeper Networks



Deeper Networks

Since AlexNet, there have been larger and better models

Deeper Networks

Since AlexNet, there have been larger and better models

- VGG

Deeper Networks

Since AlexNet, there have been larger and better models

- VGG
- ResNet

Deeper Networks

Since AlexNet, there have been larger and better models

- VGG
- ResNet
- DenseNet

Deeper Networks

Since AlexNet, there have been larger and better models

- VGG
- ResNet
- DenseNet
- MobileNet

Deeper Networks

Since AlexNet, there have been larger and better models

- VGG
- ResNet
- DenseNet
- MobileNet
- ResNeXt

Deeper Networks

Since AlexNet, there have been larger and better models

- VGG
- ResNet
- DenseNet
- MobileNet
- ResNeXt

ResNet, MobileNet, and ResNeXt are still used today!

Deeper Networks

Since AlexNet, there have been larger and better models

- VGG
- ResNet
- DenseNet
- MobileNet
- ResNeXt

ResNet, MobileNet, and ResNeXt are still used today!

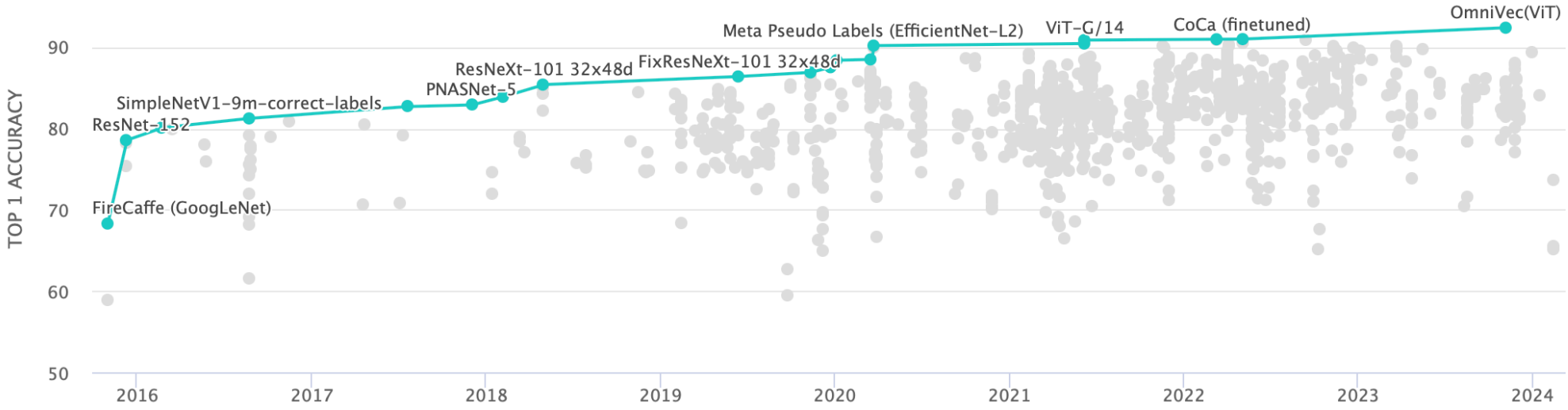
Each one introduces a few tricks to obtain better results

Deeper Networks

After ResNet, marginal improvements

Deeper Networks

After ResNet, marginal improvements



Agenda

1. Review
2. Signal Processing
3. Convolution
4. Convolutional Neural Networks
5. Additional Dimensions
6. **Deeper Networks**
7. Coding

Agenda

1. Review
2. Signal Processing
3. Convolution
4. Convolutional Neural Networks
5. Additional Dimensions
6. Deeper Networks
7. **Coding**

Coding

```
import torch
c_x = 3 # Number of colors
c_y = 32
k = 2 # Filter size
h, w = 128, 128 # Image size

conv1 = torch.nn.Conv2d(
  in_channels=c_x,
  out_channels=c_y,
  kernel_size=2
)
image = torch.rand((1, c_x, h, w)) # Torch requires BCHW
out = conv1(image) # Shape(1, c_y, h - k, w - k)
```

Coding

```
import jax, equinox
c_x = 3 # Number of colors
c_y = 32
k = 2 # Filter size
h, w = 128, 128 # Image size
conv1 = equinox.nn.Conv2d(
  in_channels=c_x,
  out_channels=c_y,
  kernel_size=2,
  key=jax.random.key(0)
)
image = jax.random.uniform(jax.random.key(1), (c_x, h, w))
out = conv1(image) # Shape(c_y, h - k, w - k)
```

Coding

```
import torch
conv1 = torch.nn.Conv2d(3, c_h, 2)
pool1 = torch.nn.AdaptiveAvgPool2d((a, a))
conv2 = torch.nn.Conv2d(c_h, c_y, 2)
pool2 = torch.nn.AdaptiveAvgPool2d((b, b))
linear = torch.nn.Linear(c_y * b * b)
z_1 = conv1(image)
z_1 = torch.nn.functional.leaky_relu(z_1)
z_1 = pool1(z_1) # Shape(1, c_h, a, a)
z_2 = conv1(z1)
z_2 = torch.nn.functional.leaky_relu(z_2)
z_2 = pool2(z_2) # Shape(1, c_y, b, b)
z_3 = linear(z_2.flatten())
```

Coding

```
import jax, equinox
conv1 = equinox.nn.Conv2d(3, c_h, 2)
pool1 = equinox.nn.AdaptiveAvgPool2d((a, a))
conv2 = equinox.nn.Conv2d(c_h, c_y, 2)
pool2 = equinox.nn.AdaptiveAvgPool2d((b, b))
linear = equinox.nn.Linear(c_y * b * b)
z_1 = conv1(image,(3, h, w))
z_1 = jax.nn.leaky_relu(z_1)
z_1 = pool1(z_1) # Shape(c_h, a, a)
z_2 = conv1(z1)
z_2 = jax.nn.leaky_relu(z_2)
z_2 = pool2(z_2) # Shape(c_y, b, b)
z_3 = linear(z_2.flatten())
```

Coding

https://colab.research.google.com/drive/1IRRSsvdeC4a5AEWF_1WX9iveBqGSppn1#scrollTo=YVkCyz78x4Rp