CS 474/574 Machine Learning
8. Deep Learning

Prof. Dr. Forrest Sheng Bao
Dept. of Computer Science
Iowa State University
Ames, IA, USA

April 21, 2021

Images that were not created by the instructor but from the web are cited in the Markdown source code in the syntax:

```
![a web image](URL)
```

Deep learning (DL)

- Deep neural networks (DNNs): extensive amounts of layers.

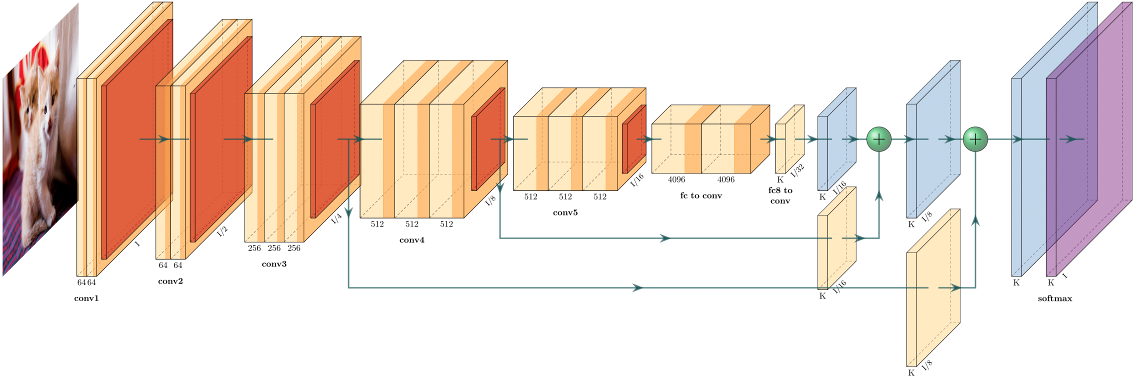


Figure 1: a web image

Why is DL something relatively new?

- DNNs use the same algorithms as any (shallow) ANNs: feedforward and backpropagation

Why is DL something relatively new?

- DNNs use the same algorithms as any (shallow) ANNs: feedforward and backpropagation
- But DL needs two new sauces: “Big Data” and “Big Processor”

Why is DL something relatively new?

- DNNs use the same algorithms as any (shallow) ANNs: feedforward and backpropagation
- But DL needs two new sauces: “Big Data” and “Big Processor”
- Extensive amounts of layers means extensive amounts of transfer matrixes (weights) that need Big Data to train

Why is DL something relatively new?

- DNNs use the same algorithms as any (shallow) ANNs: feedforward and backpropagation
- But DL needs two new sauces: "Big Data" and "Big Processor"
- Extensive amounts of layers means extensive amounts of transfer matrixes (weights) that need Big Data to train
- It wasn't feasible until massive data was digitized (how many pictures were digital before iPhone?)

Why is DL something relatively new?

- DNNs use the same algorithms as any (shallow) ANNs: feedforward and backpropagation
- But DL needs two new sauces: “Big Data” and “Big Processor”
- Extensive amounts of layers means extensive amounts of transfer matrixes (weights) that need Big Data to train
- It wasn’t feasible until massive data was digitized (how many pictures were digital before iPhone?)
- Extensive amounts of layers also needs lots of computational power in training and prediction

Why is DL something relatively new?

- DNNs use the same algorithms as any (shallow) ANNs: feedforward and backpropagation
- But DL needs two new sauces: “Big Data” and “Big Processor”
- Extensive amounts of layers means extensive amounts of transfer matrixes (weights) that need Big Data to train
- It wasn’t feasible until massive data was digitized (how many pictures were digital before iPhone?)
- Extensive amounts of layers also needs lots of computational power in training and prediction
- It wasn’t feasible until the raise of general-purpose graphic processing unit (GPGPU) computing

Why is DL something relatively new?

- DNNs use the same algorithms as any (shallow) ANNs: feedforward and backpropagation
- But DL needs two new sauces: “Big Data” and “Big Processor”
- Extensive amounts of layers means extensive amounts of transfer matrixes (weights) that need Big Data to train
- It wasn’t feasible until massive data was digitized (how many pictures were digital before iPhone?)
- Extensive amounts of layers also needs lots of computational power in training and prediction
- It wasn’t feasible until the raise of general-purpose graphic processing unit (GPGPU) computing
- New techniques have been developed to speed up the training of DNNs and/or to avoid overfitting: dropout, batch normalization, stochastic pooling, etc. They are used for other DNNs as well.

Deep learning: automated feature extraction

- Conventional ML: manually craft a set of features in a process called **feature engineering**.

Deep learning: automated feature extraction

- Conventional ML: manually craft a set of features in a process called **feature engineering**.
- Limitation: Sometimes features are too difficult to be manually designed, e.g., from I/O system log.

Deep learning: automated feature extraction

- Conventional ML: manually craft a set of features in a process called **feature engineering**.
- Limitation: Sometimes features are too difficult to be manually designed, e.g., from I/O system log.
- A (no-brainer) solution: let computers find it for us, even by brutal force.

Deep learning: automated feature extraction

- Conventional ML: manually craft a set of features in a process called **feature engineering**.
- Limitation: Sometimes features are too difficult to be manually designed, e.g., from I/O system log.
- A (no-brainer) solution: let computers find it for us, even by brutal force.
- Many DL tasks are end-to-end and hence are more black-box.

Deep learning introduces new ways to use ANNs

- There are more tasks that need function fitting beyond conventional classification and regression.

Deep learning introduces new ways to use ANNs

- There are more tasks that need function fitting beyond conventional classification and regression.
- DL can be used to generate data (e.g, generating the translation of a sentence, turning a picture into Van Gogh style)

Deep learning introduces new ways to use ANNs

- There are more tasks that need function fitting beyond conventional classification and regression.
- DL can be used to generate data (e.g, generating the translation of a sentence, turning a picture into Van Gogh style)
- Sometimes we use the network to get something else useful, such as word embedding, but discard the network in the end.

Deep learning introduces new ways to use ANNs

- There are more tasks that need function fitting beyond conventional classification and regression.
- DL can be used to generate data (e.g, generating the translation of a sentence, turning a picture into Van Gogh style)
- Sometimes we use the network to get something else useful, such as word embedding, but discard the network in the end.
- Maybe weights of only a small set of layers are what we need from training.

Deep learning introduces new ways to use ANNs

- There are more tasks that need function fitting beyond conventional classification and regression.
- DL can be used to generate data (e.g, generating the translation of a sentence, turning a picture into Van Gogh style)
- Sometimes we use the network to get something else useful, such as word embedding, but discard the network in the end.
- Maybe weights of only a small set of layers are what we need from training.
- Using DL requires creative ways to prepare training data (e.g., negative sampling), not straightforward pairs of feature/input vectors and labels/output vectors).

CNN, ConvNet, Convolutional Neural Networks

- Convolutional layers:

CNN, ConvNet, Convolutional Neural Networks

- Convolutional layers:
  - Convolution is a math operation that characterizes a signal (text, image, audio) using its spatial similarity with a template (called a **filter** in CNNs)

CNN, ConvNet, Convolutional Neural Networks

- Convolutional layers:
  - Convolution is a math operation that characterizes a signal (text, image, audio) using its spatial similarity with a template (called a **filter** in CNNs)
  - A CNN usually trains more than one filters, forming a **filter bank**.

CNN, ConvNet, Convolutional Neural Networks

- Convolutional layers:
  - Convolution is a math operation that characterizes a signal (text, image, audio) using its spatial similarity with a template (called a **filter** in CNNs)
  - A CNN usually trains more than one filters, forming a **filter bank**.
  - Conv layers are NOT fully connected.

CNN, ConvNet, Convolutional Neural Networks

- Convolutional layers:
  - Convolution is a math operation that characterizes a signal (text, image, audio) using its spatial similarity with a template (called a **filter** in CNNs)
  - A CNN usually trains more than one filters, forming a **filter bank**.
  - Conv layers are NOT fully connected.
- Pooling layers: downsample a matrix into one sample which contains the most useful information

CNN, ConvNet, Convolutional Neural Networks

- Convolutional layers:
  - Convolution is a math operation that characterizes a signal (text, image, audio) using its spatial similarity with a template (called a **filter** in CNNs)
  - A CNN usually trains more than one filters, forming a **filter bank**.
  - Conv layers are NOT fully connected.
- Pooling layers: downsample a matrix into one sample which contains the most useful information
- ReLU layers: as activations, faster to train than $\tanh$ or logistic.

CNN, ConvNet, Convolutional Neural Networks

- Convolutional layers:
  - Convolution is a math operation that characterizes a signal (text, image, audio) using its spatial similarity with a template (called a **filter** in CNNs)
  - A CNN usually trains more than one filters, forming a **filter bank**.
  - Conv layers are NOT fully connected.
- Pooling layers: downsample a matrix into one sample which contains the most useful information
- ReLU layers: as activations, faster to train than $\tanh$ or logistic.
- Fully connected layer or dense layers (basically this is the regular ANN): usually toward the end of a network.

CNN, ConvNet, Convolutional Neural Networks

- Convolutional layers:
  - Convolution is a math operation that characterizes a signal (text, image, audio) using its spatial similarity with a template (called a **filter** in CNNs)
  - A CNN usually trains more than one filters, forming a **filter bank**.
  - Conv layers are NOT fully connected.
- Pooling layers: downsample a matrix into one sample which contains the most useful information
- ReLU layers: as activations, faster to train than $\tanh$ or logistic.
- Fully connected layer or dense layers (basically this is the regular ANN): usually toward the end of a network.
- Softmax layers: common but not mandatory for output.

CNN, ConvNet, Convolutional Neural Networks

- Convolutional layers:
  - Convolution is a math operation that characterizes a signal (text, image, audio) using its spatial similarity with a template (called a **filter** in CNNs)
  - A CNN usually trains more than one filters, forming a **filter bank**.
  - Conv layers are NOT fully connected.
- Pooling layers: downsample a matrix into one sample which contains the most useful information
- ReLU layers: as activations, faster to train than $\tanh$ or logistic.
- Fully connected layer or dense layers (basically this is the regular ANN): usually toward the end of a network.
- Softmax layers: common but not mandatory for output.
- Usual architecture: seveveral stacks of $\{$conv, pool, Relu$\}$, then several FC layers, finally/optionally softmax.

CNN, ConvNet, Convolutional Neural Networks

- Convolutional layers:
  - Convolution is a math operation that characterizes a signal (text, image, audio) using its spatial similarity with a template (called a **filter** in CNNs)
  - A CNN usually trains more than one filters, forming a **filter bank**.
  - Conv layers are NOT fully connected.
- Pooling layers: downsample a matrix into one sample which contains the most useful information
- ReLU layers: as activations, faster to train than $\tanh$ or logistic.
- Fully connected layer or dense layers (basically this is the regular ANN): usually toward the end of a network.
- Softmax layers: common but not mandatory for output.
- Usual architecture: seveveral stacks of {conv, pool, Relu}, then several FC layers, finally/optionally softmax.
- Visualization of the output at layers: http://cs231n.github.io/convolutional-networks/

CNN, ConvNet, Convolutional Neural Networks

- Convolutional layers:
  - Convolution is a math operation that characterizes a signal (text, image, audio) using its spatial similarity with a template (called a **filter** in CNNs)
  - A CNN usually trains more than one filters, forming a **filter bank**.
  - Conv layers are NOT fully connected.
- Pooling layers: downsample a matrix into one sample which contains the most useful information
- ReLU layers: as activations, faster to train than $\tanh$ or logistic.
- Fully connected layer or dense layers (basically this is the regular ANN): usually toward the end of a network.
- Softmax layers: common but not mandatory for output.
- Usual architecture: seveveral stacks of {conv, pool, Relu}, then several FC layers, finally/optionally softmax.
- Visualization of the output at layers: http://cs231n.github.io/convolutional-networks/
- Too many hyperparameters (especially the size of filters and the stacking of layers). So some pre-configued architectures are commonly used, e.g., AlexNet, VGGNet, ResNet.

CNN, ConvNet, Convolutional Neural Networks

- Convolutional layers:
  - Convolution is a math operation that characterizes a signal (text, image, audio) using its spatial similarity with a template (called a **filter** in CNNs)
  - A CNN usually trains more than one filters, forming a **filter bank**.
  - Conv layers are NOT fully connected.
- Pooling layers: downsample a matrix into one sample which contains the most useful information
- ReLU layers: as activations, faster to train than $\tanh$ or logistic.
- Fully connected layer or dense layers (basically this is the regular ANN): usually toward the end of a network.
- Softmax layers: common but not mandatory for output.
- Usual architecture: seveveral stacks of {conv, pool, Relu}, then several FC layers, finally/optionally softmax.
- Visualization of the output at layers: http://cs231n.github.io/convolutional-networks/
- Too many hyperparameters (especially the size of filters and the stacking of layers). So some pre-configued architectures are commonly used, e.g., AlexNet, VGGNet, ResNet.
- Applications: matrix-like data, images, audios, 3D scans, time series, etc.

CNN, ConvNet, Convolutional Neural Networks

- Convolutional layers:
  - Convolution is a math operation that characterizes a signal (text, image, audio) using its spatial similarity with a template (called a **filter** in CNNs)
  - A CNN usually trains more than one filters, forming a **filter bank**.
  - Conv layers are NOT fully connected.
- Pooling layers: downsample a matrix into one sample which contains the most useful information
- ReLU layers: as activations, faster to train than $\tanh$ or logistic.
- Fully connected layer or dense layers (basically this is the regular ANN): usually toward the end of a network.
- Softmax layers: common but not mandatory for output.
- Usual architecture: seveveral stacks of {conv, pool, Relu}, then several FC layers, finally/optionally softmax.
- Visualization of the output at layers: http://cs231n.github.io/convolutional-networks/
- Too many hyperparameters (especially the size of filters and the stacking of layers). So some pre-configued architectures are commonly used, e.g., AlexNet, VGGNet, ResNet.
- Applications: matrix-like data, images, audios, 3D scans, time series, etc.
- Filters trained for a task can be reused or fine-tuned for another task. (**transfer learning**)

Hyperparameters of convolutional layers

See Tensorflow Conv1D, Conv2D, Conv3D

- How many filters, e.g., 32

Hyperparameters of convolutional layers

See Tensorflow Conv1D, Conv2D, Conv3D

- How many filters, e.g., 32
- Dimensions of filters, e.g., 3x3

Hyperparameters of convolutional layers

See Tensorflow Conv1D, Conv2D, Conv3D

- How many filters, e.g., 32
- Dimensions of filters, e.g., 3x3
- strides: how many pixels to shift after one convolution. 1D if 1D conv, 2D if 2D conv.

Hyperparameters of convolutional layers

See Tensorflow Conv1D, Conv2D, Conv3D

- How many filters, e.g., 32
- Dimensions of filters, e.g., 3x3
- strides: how many pixels to shift after one convolution. 1D if 1D conv, 2D if 2D conv.
- padding: to make use of samples around the edges Ref

Additional resources about CNNs

- A very informative presentation by Yann Le Cun at CERN in 2016

Additional resources about CNNs

- A very informative presentation by Yann Le Cun at CERN in 2016
- DeepVis, a CNN visualization tool by Jason Yosinski at ICML DL 2015

Additional resources about CNNs

- A very informative presentation by Yann Le Cun at CERN in 2016
- DeepVis, a CNN visualization tool by Jason Yosinski at ICML DL 2015
- TF_CNNVis, a similar CNN visualization tool based on TF

Additional resources about CNNs

- A very informative presentation by Yann Le Cun at CERN in 2016
- DeepVis, a CNN visualization tool by Jason Yosinski at ICML DL 2015
- TF_CNNVis, a similar CNN visualization tool based on TF
- https://towardsdatascience.com/how-to-visualize-convolutional-features-in-40-lines-of-code-70b7d87b0030

Recurrent Neural Networks (RNNs)

- Very often, the output of a system that we want to model depends partially on the previous output. E.g., next-word prediction or machine translation.

Recurrent Neural Networks (RNNs)

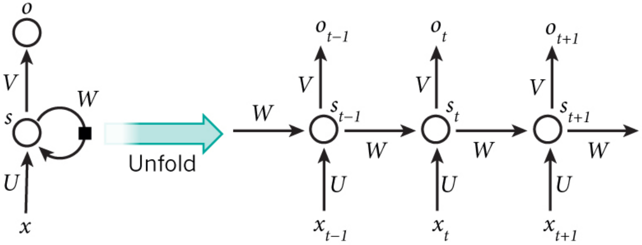
- Very often, the output of a system that we want to model depends partially on the previous output. E.g., next-word prediction or machine translation.
- RNNs is the kind of networks. "recurrent" means using the same information again.

Recurrent Neural Networks (RNNs)

- Very often, the output of a system that we want to model depends partially on the previous output. E.g., next-word prediction or machine translation.
- RNNs is the kind of networks. “recurrent” means using the same information again.
- An RNN’s input has two parts: $\mathbf{x}_t$, the “fresh” input at step t, and $\mathbf{o}_{t-1}$ the output at previous step $t-1$.

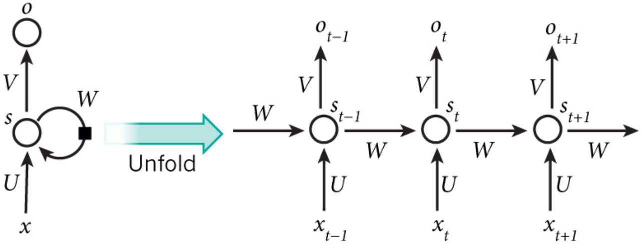
Recurrent Neural Networks (RNNs)

- Very often, the output of a system that we want to model depends partially on the previous output. E.g., next-word prediction or machine translation.
- RNNs is the kind of networks. "recurrent" means using the same information again.
- An RNN's input has two parts: $\mathbf{x}_t$, the "fresh" input at step t, and $\mathbf{o}_{t-1}$ the output at previous step $t-1$.
- Unrolling/unfolding an RNN unit:



Recurrent Neural Networks (RNNs)

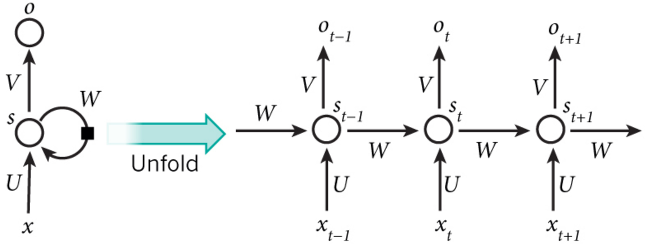
- Very often, the output of a system that we want to model depends partially on the previous output. E.g., next-word prediction or machine translation.
- RNNs is the kind of networks. "recurrent" means using the same information again.
- An RNN's input has two parts: $\mathbf{x}_t$, the "fresh" input at step t, and $\mathbf{o}_{t-1}$ the output at previous step $t-1$.
- Unrolling/unfolding an RNN unit:



- Examples: Elman network 1 2, Hopfiled Network

Recurrent Neural Networks (RNNs)

- Very often, the output of a system that we want to model depends partially on the previous output. E.g., next-word prediction or machine translation.
- RNNs is the kind of networks. "recurrent" means using the same information again.
- An RNN's input has two parts: $\mathbf{x}_t$, the "fresh" input at step t, and $\mathbf{o}_{t-1}$ the output at previous step $t-1$.
- Unrolling/unfolding an RNN unit:



- Examples: Elman network 1 2, Hopfiled Network
- An RNN is just an IIR filter (are you also a EE major?):

$$y[n] = \sum_{l=1}^{N} a_l y[n-l] + \sum_{k=0}^{M} b_k x[n-k]$$

where $x[i]$ (or $y[i]$) is the i-th element (a scalar) in the input (or output) sequence.

LTSM and GRU networks: two special kind of RNNs

- Vanilla (basic, plain) RNNs are not good at modeling long-term dependencies which is especially common in text. E.g., in the sentence, “Unlike what he said, I did NOT eat the pizza”, “Unlike” and “not”, two words separated far, need to be jointly considered to infer whether the subject ate the pizza.

LTSM and GRU networks: two special kind of RNNs

- Vanilla (basic, plain) RNNs are not good at modeling long-term dependencies which is especially common in text. E.g., in the sentence, “Unlike what he said, I did NOT eat the pizza”, “Unlike” and “not”, two words separated far, need to be jointly considered to infer whether the subject ate the pizza.
- A solution is to maintain a state. E.g., when “unlike” is scanned, set a state, later use that state to double-negate “eat” with “not”. Reset the state after “not” is scanned.

LTSM and GRU networks: two special kind of RNNs

- Vanilla (basic, plain) RNNs are not good at modeling long-term dependencies which is especially common in text. E.g., in the sentence, "Unlike what he said, I did NOT eat the pizza", "Unlike" and "not", two words separated far, need to be jointly considered to infer whether the subject ate the pizza.
- A solution is to maintain a state. E.g., when "unlike" is scanned, set a state, later use that state to double-negate "eat" with "not". Reset the state after "not" is scanned.
- Hence LSTM (an architecture) was invented. Another motiviation is that gradient vanishing is significant for a deep RNN.

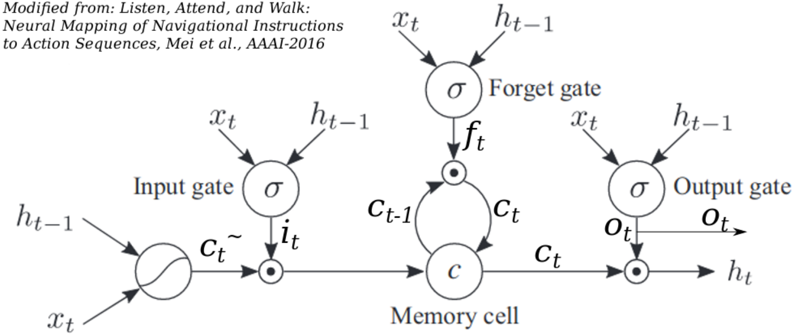
LTSM and GRU networks: two special kind of RNNs

- Vanilla (basic, plain) RNNs are not good at modeling long-term dependencies which is especially common in text. E.g., in the sentence, "Unlike what he said, I did NOT eat the pizza", "Unlike" and "not", two words separated far, need to be jointly considered to infer whether the subject ate the pizza.
- A solution is to maintain a state. E.g., when "unlike" is scanned, set a state, later use that state to double-negate "eat" with "not". Reset the state after "not" is scanned.
- Hence LSTM (an architecture) was invented. Another motiviation is that gradient vanishing is significant for a deep RNN.
- Gated Recurrent Unit (GRU): simpler, just reset gate and update gate.

LSTM gates

An LSTM neuron is a **cell** or **unit**. Each LSTM cell has 3 **gates**: forget gate, input gate, and output gate. The 3 gates are all computed from current input $\mathbf{x}(t)$ and hidden state vector (NOT output) from previous time-step $\mathbf{h}(t-1)$, but with different transfer matrixes/weights.

Modified from: Listen, Attend, and Walk: Neural Mapping of Navigational Instructions to Action Sequences, Mei et al., AAAI-2016



$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$$
$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$$
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$$
$$\tilde{c}_t = \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$
$$h_t = o_t \circ \sigma_h(c_t)$$

Note that the activation functions $\sigma_g, \sigma_c, \sigma_h$ may not all be the same. Usually σ_c and σ_h are $\tanh$ and gate activations are logistic.

Units vs. Layers

- Each LSTM unit at a time-step produces one scalar output (o_t). Some ML frameworks allow returning the outputs (e.g., `return_sequences` in `tf.keras.layers.LSTM`) at all time-steps, hence a sequence.

Units vs. Layers

- Each LSTM unit at a time-step produces one scalar output (o_t). Some ML frameworks allow returning the outputs (e.g., `return_sequences` in `tf.keras.layers.LSTM`) at all time-steps, hence a sequence.
- To train an LSTM unit, a sample is usually a sequence of inputs, each of them matches one time-step t.

Units vs. Layers

- Each LSTM unit at a time-step produces one scalar output (o_t). Some ML frameworks allow returning the outputs (e.g., `return_sequences` in `tf.keras.layers.LSTM`) at all time-steps, hence a sequence.
- To train an LSTM unit, a sample is usually a sequence of inputs, each of them matches one time-step t.
- Depending your application, you may parallelize several units together. They run independently and "remember" different things. E.g., to predict the stock price of more than two companies. See demo

Units vs. Layers

- Each LSTM unit at a time-step produces one scalar output (o_t). Some ML frameworks allow returning the outputs (e.g., `return_sequences` in `tf.keras.layers.LSTM`) at all time-steps, hence a sequence.
- To train an LSTM unit, a sample is usually a sequence of inputs, each of them matches one time-step t.
- Depending your application, you may parallelize several units together. They run independently and "remember" different things. E.g., to predict the stock price of more than two companies. See demo
- You could also stack LSTM layer along time-steps so your network can PROBABLY "remember" more complex things, e.g., Lattic GRU or context-dependent language models in NLP. It differs from giving longer input sequences.

Units vs. Layers

- Each LSTM unit at a time-step produces one scalar output (o_t). Some ML frameworks allow returning the outputs (e.g., `return_sequences` in `tf.keras.layers.LSTM`) at all time-steps, hence a sequence.
- To train an LSTM unit, a sample is usually a sequence of inputs, each of them matches one time-step t.
- Depending your application, you may parallelize several units together. They run independently and "remember" different things. E.g., to predict the stock price of more than two companies. See demo
- You could also stack LSTM layer along time-steps so your network can PROBABLY "remember" more complex things, e.g., Lattic GRU or context-dependent language models in NLP. It differs from giving longer input sequences.
- The term "cell" in literature is a unit. But in many ML frameworks, including Tensorflow, it could mean a layer of units.

Additional information about LSTM

- Time series prediction from Tensorflow tutorial

Additional information about LSTM

- Time series prediction from Tensorflow tutorial
- http://www.wildml.com/2015/10/recurrent-neural-network-tutorial-part-4-implementing-a-grulstm-rnn-with-python-and-theano/

Additional information about LSTM

- Time series prediction from Tensorflow tutorial
- http://www.wildml.com/2015/10/recurrent-neural-network-tutorial-part-4-implementing-a-grulstm-rnn-with-python-and-theano/
- http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Neural language models in Elman network

- Jeff Elman, UCSD, “Finding Structure in Time,’’ Cognitive Science, 14:179-211, 1990

Neural language models in Elman network

- Jeff Elman, UCSD, “Finding Structure in Time,’ ’ Cognitive Science, 14:179-211, 1990
- A language model predicts the Probability of a sequence of (words, characters, etc.)

Neural language models in Elman network

- Jeff Elman, UCSD, "Finding Structure in Time,' ' Cognitive Science, 14:179-211, 1990
- A language model predicts the Probability of a sequence of (words, characters, etc.)
- Because of the properties of conditional probability, we want the probability of next word, given a short history of the sequence: $P(w_{t+1}|w_i, i \in [t-k..t])$

Neural language models in Elman network

- Jeff Elman, UCSD, "Finding Structure in Time,'' Cognitive Science, 14:179-211, 1990
- A language model predicts the Probability of a sequence of (words, characters, etc.)
- Because of the properties of conditional probability, we want the probability of next word, given a short history of the sequence: $P(w_{t+1}|w_i, i \in [t-k..t])$
- Elman network/simple RNN. Three layers:

Neural language models in Elman network

- Jeff Elman, UCSD, "Finding Structure in Time,' ' Cognitive Science, 14:179-211, 1990
- A language model predicts the Probability of a sequence of (words, characters, etc.)
- Because of the properties of conditional probability, we want the probability of next word, given a short history of the sequence: $P(w_{t+1}|w_i, i \in [t-k..t])$
- Elman network/simple RNN. Three layers:
  - Input layer is the concatenation $\mathbf{x}(t)$ of two parts: the current **sequence** (not just one element!!!) $\mathbf{w}(t) = [w_{t-k}, \ldots, w_t]$, plus output from the hidden layer in previous step $\mathbf{s}(t-1)$.

Neural language models in Elman network

- Jeff Elman, UCSD, "Finding Structure in Time,' ' Cognitive Science, 14:179-211, 1990
- A language model predicts the Probability of a sequence of (words, characters, etc.)
- Because of the properties of conditional probability, we want the probability of next word, given a short history of the sequence: $P(w_{t+1}|w_i, i \in [t-k..t])$
- Elman network/simple RNN. Three layers:
  - Input layer is the concatenation $\mathbf{x}(t)$ of two parts: the current **sequence** (not just one element!!!) $\mathbf{w}(t) = [w_{t-k}, \ldots, w_t]$, plus output from the hidden layer in previous step $\mathbf{s}(t-1)$.
  - hidden/context layer: $\mathbf{s}(t) = f\left(\mathbb{X}\mathbf{x}(t)\right)$ where $\mathbb{X}$ is the matrix of weights from the input layer to hidden layer.

Neural language models in Elman network

- Jeff Elman, UCSD, "Finding Structure in Time,'' Cognitive Science, 14:179-211, 1990
- A language model predicts the Probability of a sequence of (words, characters, etc.)
- Because of the properties of conditional probability, we want the probability of next word, given a short history of the sequence: $P(w_{t+1}|w_i, i \in [t-k..t])$
- Elman network/simple RNN. Three layers:
  - Input layer is the concatenation $\mathbf{x}(t)$ of two parts: the current **sequence** (not just one element!!!) $\mathbf{w}(t) = [w_{t-k}, \ldots, w_t]$, plus output from the hidden layer in previous step $\mathbf{s}(t-1)$.
  - hidden/context layer: $\mathbf{s}(t) = f(\mathbb{X}\mathbf{x}(t))$ where $\mathbb{X}$ is the matrix of weights from the input layer to hidden layer.
  - Output layer: multiple neurons, one of which of the highest activation corresponds to the best prediction. Each neuron corresponds to one element in the sequence, e.g., word/character/etc.

Neural language models in Elman network

- Jeff Elman, UCSD, "Finding Structure in Time,'' Cognitive Science, 14:179-211, 1990
- A language model predicts the Probability of a sequence of (words, characters, etc.)
- Because of the properties of conditional probability, we want the probability of next word, given a short history of the sequence: $P(w_{t+1}|w_i, i \in [t-k..t])$
- Elman network/simple RNN. Three layers:
  - Input layer is the concatenation $\mathbf{x}(t)$ of two parts: the current **sequence** (not just one element!!!) $\mathbf{w}(t) = [w_{t-k}, \ldots, w_t]$, plus output from the hidden layer in previous step $\mathbf{s}(t-1)$.
  - hidden/context layer: $\mathbf{s}(t) = f\left(\mathbb{X}\mathbf{x}(t)\right)$ where $\mathbb{X}$ is the matrix of weights from the input layer to hidden layer.
  - Output layer: multiple neurons, one of which of the highest activation corresponds to the best prediction. Each neuron corresponds to one element in the sequence, e.g., word/character/etc.
- The new language model: $P(w_{t+1}|\mathbf{w}(t), \mathbf{s}(t-1))$, predicting the next output word given a short history $\mathbf{w}(t)$ up to current step t and the hidden layer up to previous step $t-1$.

Neural language models – additional reading

- Feedforward: “A neural Probabilistic Language Model”, Beigio et al., JMLR, 3:1137–1155, 2003

Neural language models – additional reading

- Feedforward: “A neural Probabilistic Language Model”, Beigio et al., JMLR, 3:1137–1155, 2003
- “Recurrent neural network based language model”, Mikolov et al., Interspeech 2010

Neural language models – additional reading

- Feedforward: "A neural Probabilistic Language Model", Beigio et al., JMLR, 3:1137–1155, 2003
- "Recurrent neural network based language model", Mikolov et al., Interspeech 2010
- Multiplicative RNN: "Generating Text with Recurrent Neural Networks", Sutskever, ICML 2011

Neural language models – additional reading

- Feedforward: “A neural Probabilistic Language Model”, Beigio et al., JMLR, 3:1137–1155, 2003
- “Recurrent neural network based language model”, Mikolov et al., Interspeech 2010
- Multiplicative RNN: “Generating Text with Recurrent Neural Networks”, Sutskever, ICML 2011
- CBOW vs. skip-gram models

Embedding

- Many objects (e.g., words, categories) are discrete. Not suitable for NNs. See Block2Vec

Embedding

- Many objects (e.g., words, categories) are discrete. Not suitable for NNs. See Block2Vec
- Embedding uses a vector of floats to represent a discrete object. Ideally, similar objects, e.g., “cat” and “tiger” have similar vectors, whereas irrelavant objects, e.g., “cat” and “hat” have orthorgonal vectors.

Embedding

- Many objects (e.g., words, categories) are discrete. Not suitable for NNs. See Block2Vec
- Embedding uses a vector of floats to represent a discrete object. Ideally, similar objects, e.g., "cat" and "tiger" have similar vectors, whereas irrelavant objects, e.g., "cat" and "hat" have orthorgonal vectors.
- An embedding layer is a look-up table, initially random. See demo

Embedding

- Many objects (e.g., words, categories) are discrete. Not suitable for NNs. See Block2Vec
- Embedding uses a vector of floats to represent a discrete object. Ideally, similar objects, e.g., "cat" and "tiger" have similar vectors, whereas irrelavant objects, e.g., "cat" and "hat" have orthorgonal vectors.
- An embedding layer is a look-up table, initially random. See demo
- Embeddings are obtained by backpropagation from downstream tasks.

Embedding

- Many objects (e.g., words, categories) are discrete. Not suitable for NNs. See Block2Vec
- Embedding uses a vector of floats to represent a discrete object. Ideally, similar objects, e.g., “cat” and “tiger” have similar vectors, whereas irrelavant objects, e.g., “cat” and “hat” have orthorgonal vectors.
- An embedding layer is a look-up table, initially random. See demo
- Embeddings are obtained by backpropagation from downstream tasks.
- Word embedding (static): Word2vec, GloVe

Embedding

- Many objects (e.g., words, categories) are discrete. Not suitable for NNs. See Block2Vec
- Embedding uses a vector of floats to represent a discrete object. Ideally, similar objects, e.g., “cat” and “tiger” have similar vectors, whereas irrelavant objects, e.g., “cat” and “hat” have orthorgonal vectors.
- An embedding layer is a look-up table, initially random. See demo
- Embeddings are obtained by backpropagation from downstream tasks.
- Word embedding (static): Word2vec, GloVe
- How to create a downstream task for word embedding? Negative sampling in CBOW and skip-gram.

Embedding

- Many objects (e.g., words, categories) are discrete. Not suitable for NNs. See Block2Vec
- Embedding uses a vector of floats to represent a discrete object. Ideally, similar objects, e.g., “cat” and “tiger” have similar vectors, whereas irrelavant objects, e.g., “cat” and “hat” have orthorgonal vectors.
- An embedding layer is a look-up table, initially random. See demo
- Embeddings are obtained by backpropagation from downstream tasks.
- Word embedding (static): Word2vec, GloVe
- How to create a downstream task for word embedding? Negative sampling in CBOW and skip-gram.
- Dynamic/contextual word embedding: AI^2 Elmo, biLSTM, Transformer, OpenAI GPT.

Embedding

- Many objects (e.g., words, categories) are discrete. Not suitable for NNs. See Block2Vec
- Embedding uses a vector of floats to represent a discrete object. Ideally, similar objects, e.g., "cat" and "tiger" have similar vectors, whereas irrelavant objects, e.g., "cat" and "hat" have orthorgonal vectors.
- An embedding layer is a look-up table, initially random. See demo
- Embeddings are obtained by backpropagation from downstream tasks.
- Word embedding (static): Word2vec, GloVe
- How to create a downstream task for word embedding? Negative sampling in CBOW and skip-gram.
- Dynamic/contextual word embedding: AI^2 Elmo, biLSTM, Transformer, OpenAI GPT.
- Sentence embedding: Google Universal Sentence Encoder, Google BERT, Facebook InferSent

Embedding

- Many objects (e.g., words, categories) are discrete. Not suitable for NNs. See Block2Vec
- Embedding uses a vector of floats to represent a discrete object. Ideally, similar objects, e.g., “cat” and “tiger” have similar vectors, whereas irrelavant objects, e.g., “cat” and “hat” have orthorgonal vectors.
- An embedding layer is a look-up table, initially random. See demo
- Embeddings are obtained by backpropagation from downstream tasks.
- Word embedding (static): Word2vec, GloVe
- How to create a downstream task for word embedding? Negative sampling in CBOW and skip-gram.
- Dynamic/contextual word embedding: AI^2 Elmo, biLSTM, Transformer, OpenAI GPT.
- Sentence embedding: Google Universal Sentence Encoder, Google BERT, Facebook InferSent
- See also: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Embedding

Encoder-decoder architecutures: NNs go generative

- The next/context-word prediction is an example that DL is not about classification or regression in conventional ML sense.

Encoder-decoder architecutures: NNs go generative

- The next/context-word prediction is an example that DL is not about classification or regression in conventional ML sense.
- The output of the neural network is not a handful of discrete labels representing classes of objects, nor numerical value to approximate, but a kind of information in the same domain of the input that can be perceived by humans directly, e.g., word(s) to word(s).

Encoder-decoder architecutures: NNs go generative

- The next/context-word prediction is an example that DL is not about classification or regression in conventional ML sense.
- The output of the neural network is not a handful of discrete labels representing classes of objects, nor numerical value to approximate, but a kind of information in the same domain of the input that can be perceived by humans directly, e.g., word(s) to word(s).
- Because the neural network first transforms human-readable information into a human-non-readable form, e.g., convolution or hidden states of an RNN, we call this process *encoding*. Then, such informtion, deeply processed, is converted by to a human-readable form, in the *decoding* stage.

Encoder-decoder architecutures: NNs go generative

- The next/context-word prediction is an example that DL is not about classification or regression in conventional ML sense.
- The output of the neural network is not a handful of discrete labels representing classes of objects, nor numerical value to approximate, but a kind of information in the same domain of the input that can be perceived by humans directly, e.g., word(s) to word(s).
- Because the neural network first transforms human-readable information into a human-non-readable form, e.g., convolution or hidden states of an RNN, we call this process *encoding*. Then, such informtion, deeply processed, is converted by to a human-readable form, in the *decoding* stage.
- Examples: from grayscale images to RGB images, from photos to Van Gogh style paintings, machine translation, from rotated objects to upright objects, from blurry/out-of-focus images to clear images, from text descriptions to animations, from videos to captions (video captioning), etc.

Encoder-decoder architecutures: NNs go generative

- The next/context-word prediction is an example that DL is not about classification or regression in conventional ML sense.
- The output of the neural network is not a handful of discrete labels representing classes of objects, nor numerical value to approximate, but a kind of information in the same domain of the input that can be perceived by humans directly, e.g., word(s) to word(s).
- Because the neural network first transforms human-readable information into a human-non-readable form, e.g., convolution or hidden states of an RNN, we call this process *encoding*. Then, such informtion, deeply processed, is converted by to a human-readable form, in the *decoding* stage.
- Examples: from grayscale images to RGB images, from photos to Van Gogh style paintings, machine translation, from rotated objects to upright objects, from blurry/out-of-focus images to clear images, from text descriptions to animations, from videos to captions (video captioning), etc.
- Although still supervised, the expected output is not manually provided but given from data automatically – semi-supervised.

Seq-to-seq learning

- Seq(uence)-to-Seq(uence) learning is a good example of encoder-decoder architecture.

Seq-to-seq learning

- Seq(uence)-to-Seq(uence) learning is a good example of encoder-decoder architecture.
- A set of approaches belong to sequence-to-sequence (seq-to-seq) learning.

Seq-to-seq learning

- Seq(uence)-to-Seq(uence) learning is a good example of encoder-decoder architecture.
- A set of approaches belong to sequence-to-sequence (seq-to-seq) learning.
- Two RNNs: encoder and decoder, the former encodes sequantial inputs into information hidden in weights of an NN while the latter generates outputs, usually one each timestep, based on the hidden information. See "I am a student" figure in Tensorflow NMT tutorial

Seq-to-seq learning

- Seq(uence)-to-Seq(uence) learning is a good example of encoder-decoder architecture.
- A set of approaches belong to sequence-to-sequence (seq-to-seq) learning.
- Two RNNs: encoder and decoder, the former encodes sequantial inputs into information hidden in weights of an NN while the latter generates outputs, usually one each timestep, based on the hidden information. See "I am a student" figure in Tensorflow NMT tutorial
- "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation", Cho et al., EMNLP 2014

Seq-to-seq learning

- Seq(uence)-to-Seq(uence) learning is a good example of encoder-decoder architecture.
- A set of approaches belong to sequence-to-sequence (seq-to-seq) learning.
- Two RNNs: encoder and decoder, the former encodes sequantial inputs into information hidden in weights of an NN while the latter generates outputs, usually one each timestep, based on the hidden information. See "I am a student" figure in Tensorflow NMT tutorial
- "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation", Cho et al., EMNLP 2014
- Other applications: seizure prediction, speech recognition or synthesis, e.g., Boss Wang et al., 2017, Tacotron: Towards End-to-end Speech Synthesis

Autoencoders: representation learning

- Let's bring DL-based generation to another level: can we re-produce the input data from the output layer?

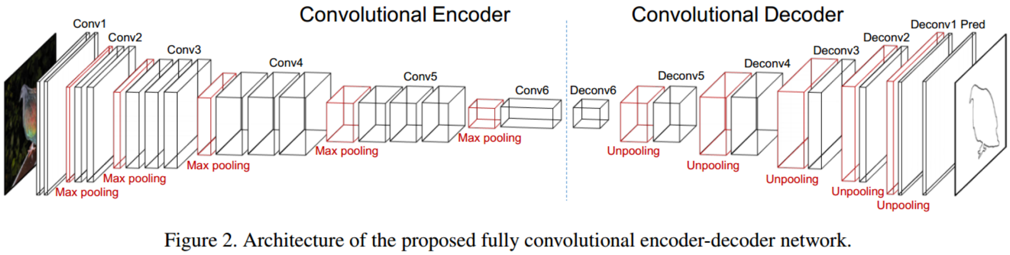


Figure 2: Object Contour Detection with a Fully Convolutional Encoder-Decoder Network, Yang et al., CVPR 2016

Autoencoders: representation learning

- Let's bring DL-based generation to another level: can we re-produce the input data from the output layer?

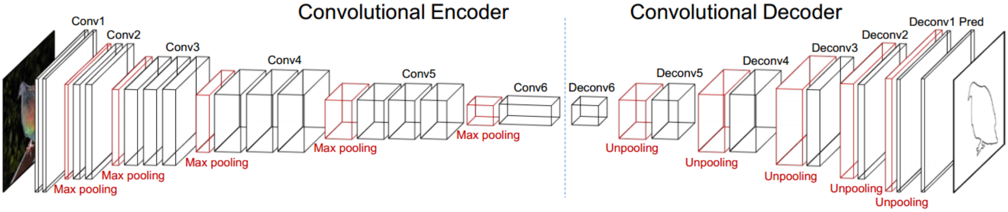


Figure 2. Architecture of the proposed fully convolutional encoder-decoder network.

Figure 2: Object Contour Detection with a Fully Convolutional Encoder-Decoder Network, Yang et al., CVPR 2016

- Applications: data compression. More applications: image deblurring. Lots of illegal things: courterfeit signatures/voices (don't worry, we will talk about how to catch them in GANs)

Generative adversarial networks (GANs)

- A GAN has two NNs: generator and discriminator/classifier. They push the limit of each other to find the essence of data.

See Google's tutorial

Further reading on word and sentence embedding

- Generalized Language Models by Lilian Weng

Further reading on word and sentence embedding

- Generalized Language Models by Lilian Weng
- https://stats.stackexchange.com/questions/421935/what-exactly-are-keys-queries-and-values-in-attention-mechanisms

Further reading on word and sentence embedding

- Generalized Language Models by Lilian Weng
- https://stats.stackexchange.com/questions/421935/what-exactly-are-keys-queries-and-values-in-attention-mechanisms
- Deconstructing BERT, Part 2: Visualizing the Inner Workings of Attention, by Jesse Vig

Further reading on word and sentence embedding

- Generalized Language Models by Lilian Weng
- https://stats.stackexchange.com/questions/421935/what-exactly-are-keys-queries-and-values-in-attention-mechanisms
- Deconstructing BERT, Part 2: Visualizing the Inner Workings of Attention, by Jesse Vig
- Implementing Transformer in Tensorflow, by Tensorflow team

Further reading on word and sentence embedding

- Generalized Language Models by Lilian Weng
- https://stats.stackexchange.com/questions/421935/what-exactly-are-keys-queries-and-values-in-attention-mechanisms
- Deconstructing BERT, Part 2: Visualizing the Inner Workings of Attention, by Jesse Vig
- Implementing Transformer in Tensorflow, by Tensorflow team
- Implementing Transformer in PyTorch, by Harvard NLP

Transfer learning

- Multitask learning

Transfer learning

- Multitask learning
- Long et al., Conditional Adversarial Domain Adaptation, NIPS 2018

Frameworks

There are several DL frameworks:

- Tensorflow (Keras), PyTorch, Flux, MXnet

Frameworks

There are several DL frameworks:

- Tensorflow (Keras), PyTorch, Flux, MXnet
- No longer active dev: Torch, Theano

Frameworks

There are several DL frameworks:

- Tensorflow (Keras), PyTorch, Flux, MXnet
- No longer active dev: Torch, Theano
- They share similar UI, e.g., Torch's layer stacking is very similar to that of Keras.

Frameworks

There are several DL frameworks:

- Tensorflow (Keras), PyTorch, Flux, MXnet
- No longer active dev: Torch, Theano
- They share similar UI, e.g., Torch's layer stacking is very similar to that of Keras.
- ONNX

DL on small devices

- DL is cool. But a DNN easily involve hundreds of thousands of parameters. How to pack them into a device of only a few MB of memory and powered by a battery?

DL on small devices

- DL is cool. But a DNN easily involve hundreds of thousands of parameters. How to pack them into a device of only a few MB of memory and powered by a battery?
- Solution 1, specific hardware/co-processor, mostly are for vision tasks: Google Coral Edge TPU, Intel Movidius, ARM Ethos (UK), Ambarella CVFLow, nVidia Tegra with CUDA (see also Jetson), QualComm Zeroth, Cambricon (China), Huawei NPU in Kirin (China)

DL on small devices

- DL is cool. But a DNN easily involve hundreds of thousands of parameters. How to pack them into a device of only a few MB of memory and powered by a battery?
- Solution 1, specific hardware/co-processor, mostly are for vision tasks: Google Coral Edge TPU, Intel Movidius, ARM Ethos (UK), Ambarella CVFLow, nVidia Tegra with CUDA (see also Jetson), QualComm Zeroth, Cambricon (China), Huawei NPU in Kirin (China)
- Using special hardware for operations like convolution is not new. DSPs have been there for decades, e.g., TI TMS320.

DL on small devices

- DL is cool. But a DNN easily involve hundreds of thousands of parameters. How to pack them into a device of only a few MB of memory and powered by a battery?
- Solution 1, specific hardware/co-processor, mostly are for vision tasks: Google Coral Edge TPU, Intel Movidius, ARM Ethos (UK), Ambarella CVFLow, nVidia Tegra with CUDA (see also Jetson), QualComm Zeroth, Cambricon (China), Huawei NPU in Kirin (China)
- Using special hardware for operations like convolution is not new. DSPs have been there for decades, e.g., TI TMS320.
- Solution 2, tailored libraries:

DL on small devices

- DL is cool. But a DNN easily involve hundreds of thousands of parameters. How to pack them into a device of only a few MB of memory and powered by a battery?
- Solution 1, specific hardware/co-processor, mostly are for vision tasks: Google Coral Edge TPU, Intel Movidius, ARM Ethos (UK), Ambarella CVFLow, nVidia Tegra with CUDA (see also Jetson), QualComm Zeroth, Cambricon (China), Huawei NPU in Kirin (China)
- Using special hardware for operations like convolution is not new. DSPs have been there for decades, e.g., TI TMS320.
- Solution 2, tailored libraries:
    - Google Tensorflow Lite, its supported microcontrollers, and an example on Arduino Nano 33 Sense

DL on small devices

- DL is cool. But a DNN easily involve hundreds of thousands of parameters. How to pack them into a device of only a few MB of memory and powered by a battery?
- Solution 1, specific hardware/co-processor, mostly are for vision tasks: Google Coral Edge TPU, Intel Movidius, ARM Ethos (UK), Ambarella CVFLow, nVidia Tegra with CUDA (see also Jetson), QualComm Zeroth, Cambricon (China), Huawei NPU in Kirin (China)
- Using special hardware for operations like convolution is not new. DSPs have been there for decades, e.g., TI TMS320.
- Solution 2, tailored libraries:
  - Google Tensorflow Lite, its supported microcontrollers, and an example on Arduino Nano 33 Sense
  - Google MobileNet

DL on small devices

- DL is cool. But a DNN easily involve hundreds of thousands of parameters. How to pack them into a device of only a few MB of memory and powered by a battery?
- Solution 1, specific hardware/co-processor, mostly are for vision tasks: Google Coral Edge TPU, Intel Movidius, ARM Ethos (UK), Ambarella CVFLow, nVidia Tegra with CUDA (see also Jetson), QualComm Zeroth, Cambricon (China), Huawei NPU in Kirin (China)
- Using special hardware for operations like convolution is not new. DSPs have been there for decades, e.g., TI TMS320.
- Solution 2, tailored libraries:
  - Google Tensorflow Lite, its supported microcontrollers, and an example on Arduino Nano 33 Sense
  - Google MobileNet
- Solution 3, a new math:

DL on small devices

- DL is cool. But a DNN easily involve hundreds of thousands of parameters. How to pack them into a device of only a few MB of memory and powered by a battery?
- Solution 1, specific hardware/co-processor, mostly are for vision tasks: Google Coral Edge TPU, Intel Movidius, ARM Ethos (UK), Ambarella CVFLow, nVidia Tegra with CUDA (see also Jetson), QualComm Zeroth, Cambricon (China), Huawei NPU in Kirin (China)
- Using special hardware for operations like convolution is not new. DSPs have been there for decades, e.g., TI TMS320.
- Solution 2, tailored libraries:
  - Google Tensorflow Lite, its supported microcontrollers, and an example on Arduino Nano 33 Sense
  - Google MobileNet
- Solution 3, a new math:
  - quantization/integer neural networks 1 by Mathworks 2 by QualComm

DL on small devices

- DL is cool. But a DNN easily involve hundreds of thousands of parameters. How to pack them into a device of only a few MB of memory and powered by a battery?
- Solution 1, specific hardware/co-processor, mostly are for vision tasks: Google Coral Edge TPU, Intel Movidius, ARM Ethos (UK), Ambarella CVFLow, nVidia Tegra with CUDA (see also Jetson), QualComm Zeroth, Cambricon (China), Huawei NPU in Kirin (China)
- Using special hardware for operations like convolution is not new. DSPs have been there for decades, e.g., TI TMS320.
- Solution 2, tailored libraries:
  - Google Tensorflow Lite, its supported microcontrollers, and an example on Arduino Nano 33 Sense
  - Google MobileNet
- Solution 3, a new math:
  - quantization/integer neural networks 1 by Mathworks 2 by QualComm
  - analog neural networks: Hopfield 1990, IBM Analog AI using phase-change memory

DL on small devices

- DL is cool. But a DNN easily involve hundreds of thousands of parameters. How to pack them into a device of only a few MB of memory and powered by a battery?
- Solution 1, specific hardware/co-processor, mostly are for vision tasks: Google Coral Edge TPU, Intel Movidius, ARM Ethos (UK), Ambarella CVFLow, nVidia Tegra with CUDA (see also Jetson), QualComm Zeroth, Cambricon (China), Huawei NPU in Kirin (China)
- Using special hardware for operations like convolution is not new. DSPs have been there for decades, e.g., TI TMS320.
- Solution 2, tailored libraries:
  - Google Tensorflow Lite, its supported microcontrollers, and an example on Arduino Nano 33 Sense
  - Google MobileNet
- Solution 3, a new math:
  - quantization/integer neural networks 1 by Mathworks 2 by QualComm
  - analog neural networks: Hopfield 1990, IBM Analog AI using phase-change memory
- "Cross training" (compared to cross-compilation): train on the cloud/workstation and predict on the edge, e.g., Amazon DeepLens