15-388/688 - Practical Data Science: Deep learning

Pat Virtue
Carnegie Mellon University
Spring 2022

Slide credits: CMU AI, Zico Kolter

Outline

Recent history in machine learning

Machine learning with neural networks

Training neural networks

Specialized neural network architectures

Deep learning in data science

Outline

Recent history in machine learning

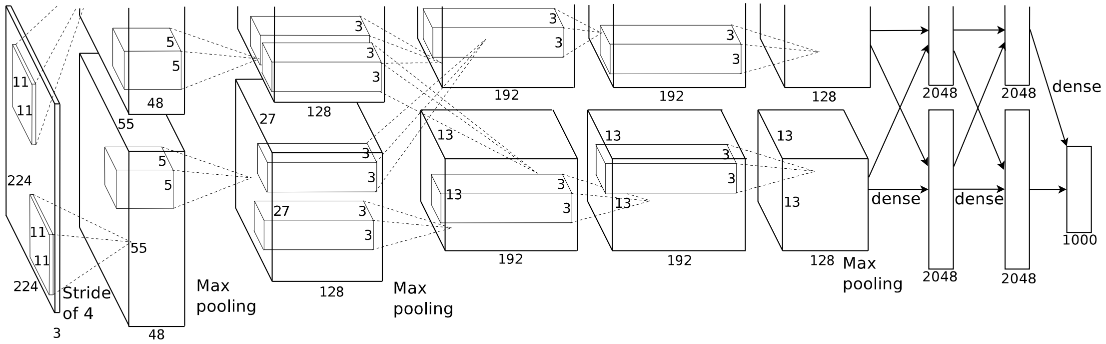
Machine learning with neural networks

Training neural networks

Specialized neural network architectures

Deep learning in data science

AlexNet

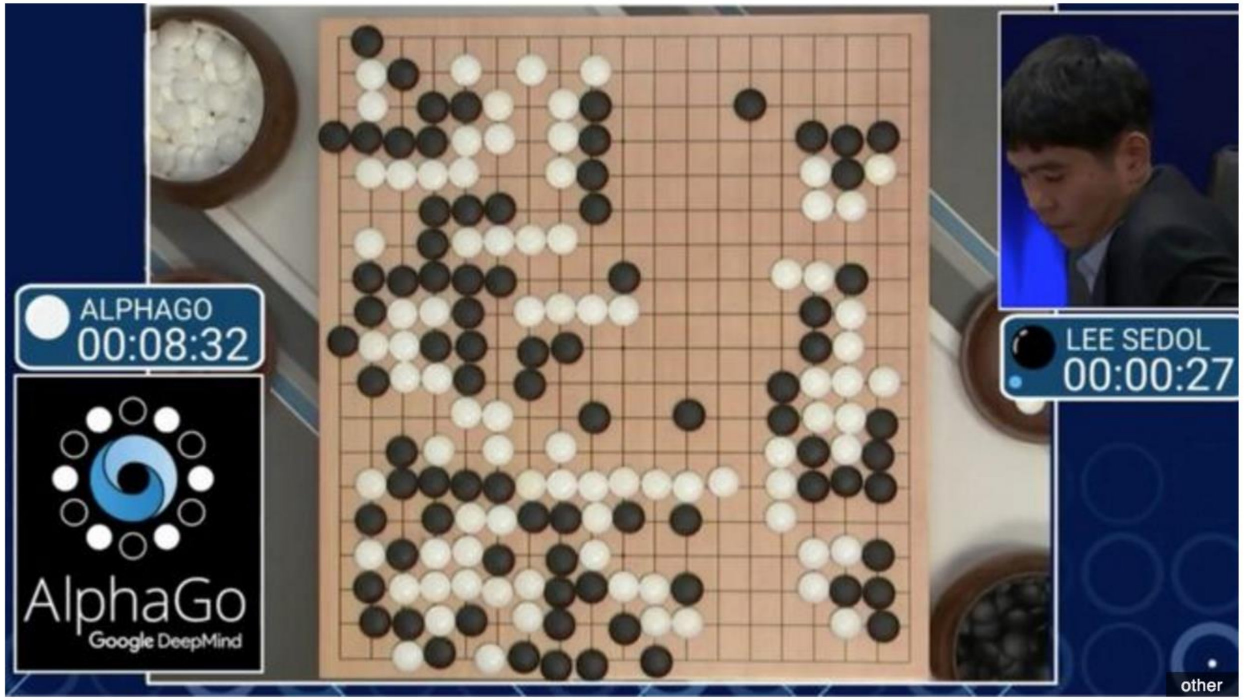


"AlexNet" (Krizhevsky et al., 2012), winning entry of ImageNet 2012 competition with a Top-5 error rate of 15.3% (next best system with highly engineered features based got 26.1% error)

AlphaGo

Artificial intelligence: Google's AlphaGo beats Go master Lee Se-dol

12 March 2016 | Technology



Google Translate

In November 2016, Google transitioned it's translation service to a deep-learning-based system, dramatically improved translation quality in many settings

Kilimanjaro is 19,710 feet of the mountain covered with snow, and it is said that the highest mountain in Africa. Top of the west, "Ngaje Ngai" in the Maasai language, has been referred to as the house of God. The top close to the west, there is a dry, frozen carcass of a leopard. Whether the leopard had what the demand at that altitude, there is no that nobody explained.

Kilimanjaro is a mountain of 19,710 feet covered with snow and is said to be the highest mountain in Africa. The summit of the west is called "Ngaje Ngai" in Masai, the house of God. Near the top of the west there is a dry and frozen dead body of leopard. No one has ever explained what leopard wanted at that altitude.

https://www.nytimes.com/2016/12/14/magazine/the-great-ai-awakening.html

Outline

Recent history in machine learning

Machine learning with neural networks

Training neural networks

Specialized neural network architectures

Deep learning in data science

Neural networks for machine learning

The term “neural network” largely refers to the hypothesis class part of a machine learning algorithm:

1. **Hypothesis:** non-linear hypothesis function, which involve compositions of multiple linear operators (e.g. matrix multiplications) and elementwise non-linear functions

2. **Loss:** “Typical” loss functions for classification and regression: logistic, softmax (multiclass logistic), hinge, squared error, absolute error

3. **Optimization:** Gradient descent, or more specifically, a variant called stochastic gradient descent we will discuss shortly

Linear hypotheses and feature learning

Until now, we have (mostly) considered machine learning algorithms that linear hypothesis class

$$h_\theta(x) = \theta^T \phi(x)$$

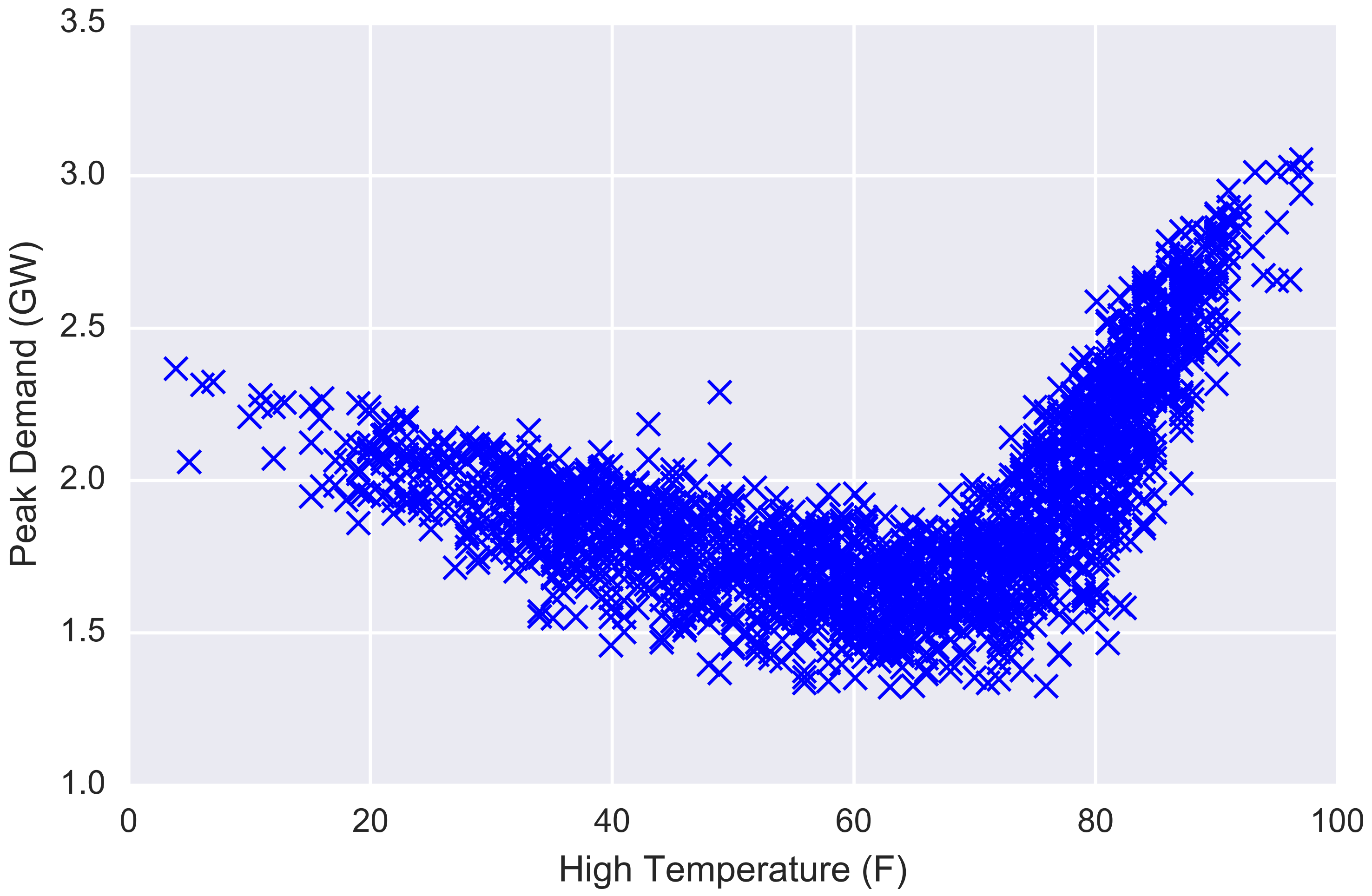
where $\phi: \mathbb{R}^n \rightarrow \mathbb{R}^k$ denotes some set of typically non-linear features

Example: polynomials, radial basis functions, custom features like TFIDF (in many domains every 10 years or so there would be new feature types)

The performance of these algorithms depends crucially on coming up with good features

Key question: can we come up with an algorithm that will automatically *learn* the features themselves?

Combining a set of linear functions



Feature learning, take one

Instead of a simple linear classifier, let's consider a two-stage hypothesis class where one linear function creates the features and another produces the final hypothesis

$$h_\theta(x) = W_2\phi(x) + b_2 = W_2(W_1 x + b_1) + b_2,$$
$$\theta = \{W_1 \in \mathbb{R}^{k\times n}, b_1 \in \mathbb{R}^k, W_2 \in \mathbb{R}^{1\times k}, b_2 \in \mathbb{R}\}$$

But there is a problem:

$$h_\theta(x) = W_2(W_1 x + b_1) + b_2 = \tilde{W} x + \tilde{b}$$

i.e., we are still just using a linear classifier (the apparent added complexity is actually not changing the underlying hypothesis function)

Neural networks

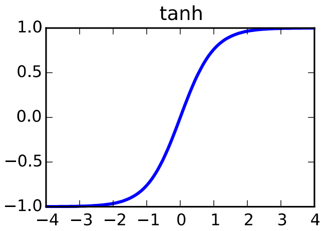
Neural networks are a simple extension of this idea, where we additionally apply a non-linear function after each linear transformation

$$h_\theta(x) = f_2(W_2 f_1(W_1 x + b_1) + b_2)$$

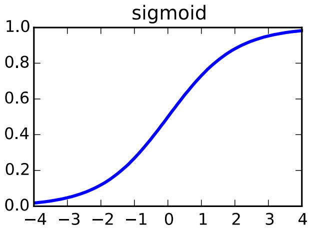
where $f_1, f_2 \colon \mathbb{R} \to \mathbb{R}$ are a non-linear function (applied elementwise)

Common choices of f_i:

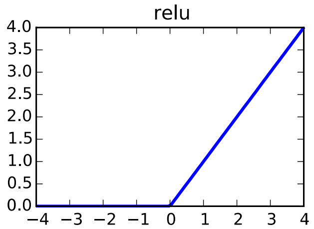
Hyperbolic tangent: $f(x) = \tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$



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

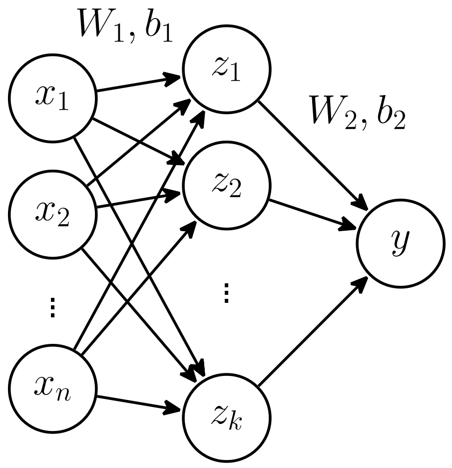


Rectified linear unit (ReLU): $f(x) = \max\{x, 0\}$



Illustrating neural networks

We can illustrate the form of neural networks using figures like the following

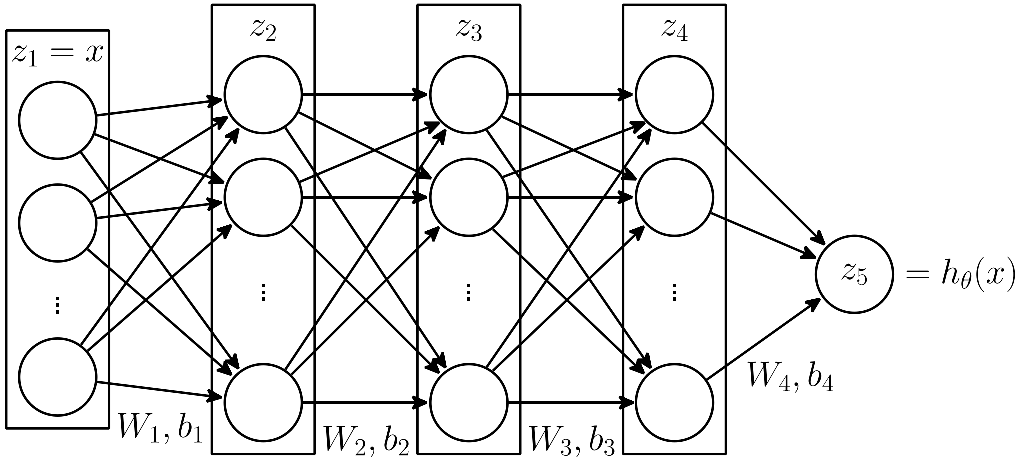


Middle layer z is referred to as the *hidden layer* or *activations*

These are the learned features, nothing in the data prescribed what values they should take, left up to algorithm to decide

Deep learning

"Deep learning" refers (almost always) to machine learning using neural network models with multiple hidden layers



Hypothesis function for k-layer network

$$z_{i+1} = f_i(W_i z_i + b_i), \qquad z_1 = x, \qquad h_\theta(x) = z_k$$

(note the z_i here refers to a vector, not an entry into vector)

Properties of neural networks

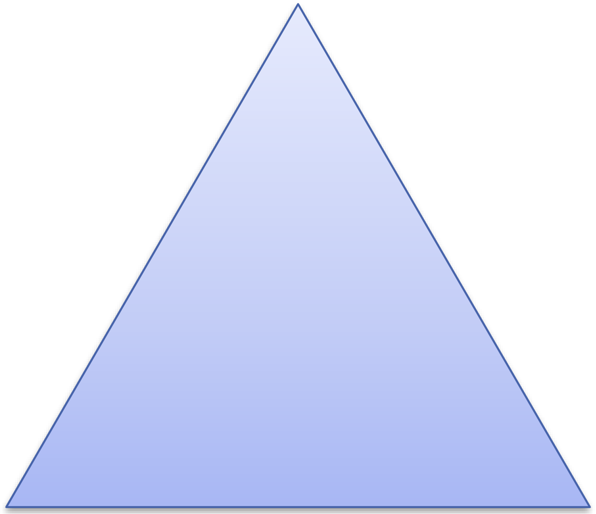
A neural network will a single hidden layers (and enough hidden units) is a *universal function approximator*, can approximate *any* function over inputs

In practice, not *that* relevant (similar to how polynomials can fit any function), and the more important aspect is that they appear to work very well in practice for many domains

The hypothesis $h_\theta(x)$ is not a convex function of parameters $\theta = \{W_i, b_i\}$, so we have possibility of local optima

Architectural choices (how many layers, how they are connected, etc), become important algorithmic design choices (i.e. hyperparameters)

Why now?



Why such rapid progress?

Outline

Recent history in machine learning

Machine learning with neural networks

Training neural networks

Specialized neural network architectures

Deep learning in data science

Neural networks for machine learning

Hypothesis function: neural network

Loss function: "traditional" loss, e.g. logistic loss for binary classification:

$$\ell(h_\theta(x), y) = \log\big(1 + \exp\big(-y \cdot h_\theta(x)\big)\big)$$

Optimization: How do we solve the optimization problem

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{m} \ell\big(h_\theta\big(x^{(i)}\big), y^{(i)}\big)$$

Just use gradient descent as normal (or rather, a version called stochastic gradient descent)

Stochastic gradient descent

Key challenge for neural networks: often have very large number of samples, computing gradients can be computationally intensive.

Traditional gradient descent computes the gradient with respect to the sum over *all examples,* then adjusts the parameters in this direction

$$\theta := \theta - \alpha \sum_{i=1}^{m} \nabla_\theta \ell(h_\theta\left(x^{(i)}, y^{(i)}\right)$$

Alternative approach, *stochastic gradient descent* (SGD): adjust parameters based upon just *one* sample

$$\theta := \theta - \alpha \nabla_\theta \ell\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$$

and then repeat these updates for all samples

Gradient descent vs. SGD

Gradient descent, repeat:

- For $i = 1, \dots, m$:

$$g^{(i)} \leftarrow \nabla_\theta \ell\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$$

- Update parameters:

$$\theta \leftarrow \theta - \alpha \sum_{i=1}^{m} g^{(i)}$$

Stochastic gradient descent, repeat:

- For $i = 1, \dots, m$:

$$\theta \leftarrow \theta - \nabla_\theta \ell\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$$

In practice, stochastic gradient descent uses a small collection of samples, not just one, called a *minibatch*

Computing gradients: backpropagation

So, how do we compute the gradient $\nabla_\theta \ell\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$?

Remember θ here denotes a set of parameters, so we're really computing gradients with respect to all elements of that set

This is accomplished via the *backpropagation* algorithm

We won't cover the algorithm in detail, but backpropagation is just an application of the (multivariate) chain rule from calculus, plus "caching" intermediate terms that, for instance, occur in the gradient of both W_1 and W_2

Training neural networks in practice

The other good news is also that you will rarely need to implement backpropagation yourself

Many libraries provides methods for you to just specify the neural network “forward” pass, and automatically compute the necessary gradients

Examples: Tensorflow, PyTorch

You’ll use one of these a bit on the homework

Outline

Recent history in machine learning

Machine learning with neural networks

Training neural networks

Specialized neural network architectures

Deep learning in data science

Specialized architectures

Very little of the current wave of enthusiasm for deep learning has actually come from the simple “fully connected” neural network model we have seen so far

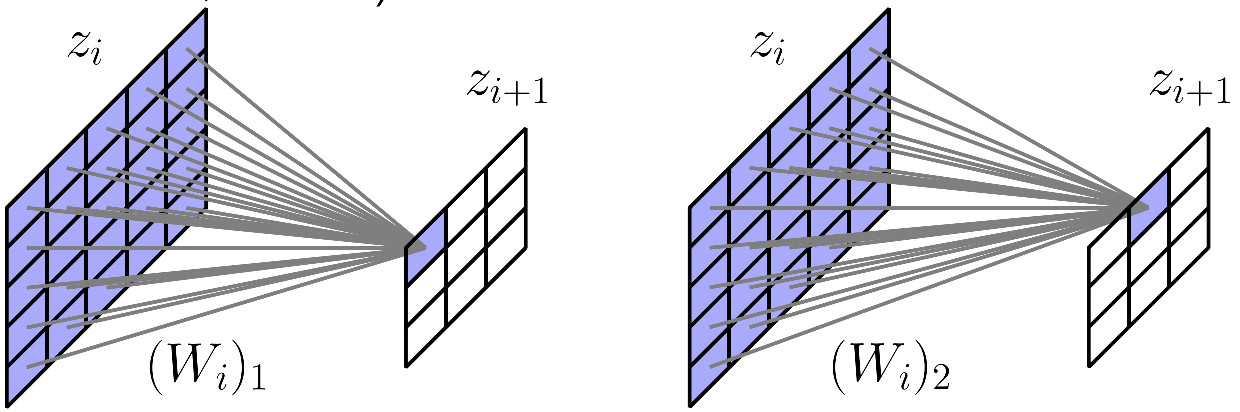
Instead, most of the excitement has come from two more specialized architectures: **convolutional neural networks**, and **recurrent neural networks**

The problem with fully-connected networks

A 256x256 (RGB) image means ~200,000 dimensional input

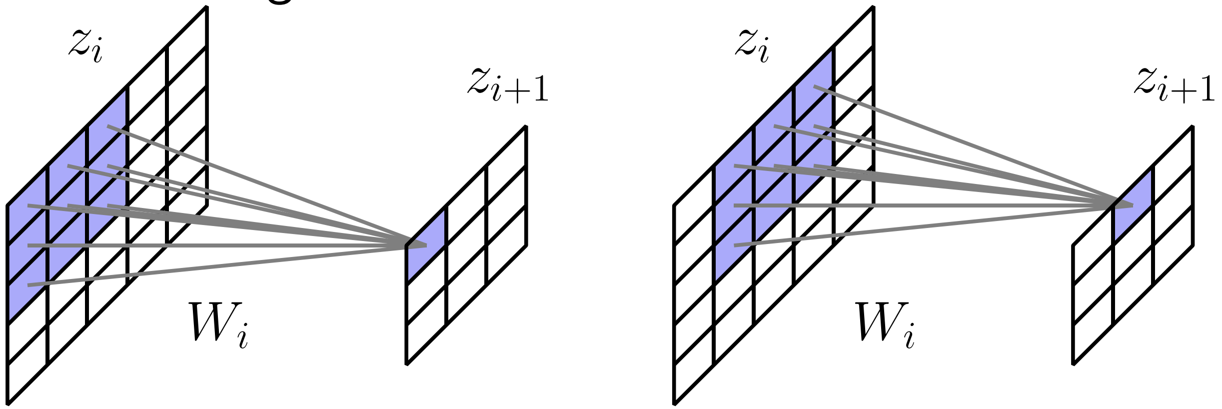
Fully connected deep network would require a huge number of parameters, very likely to overfit to data

A generic deep network also doesn't capture of the the "natural" *invariances* we expect in images (location, scale)

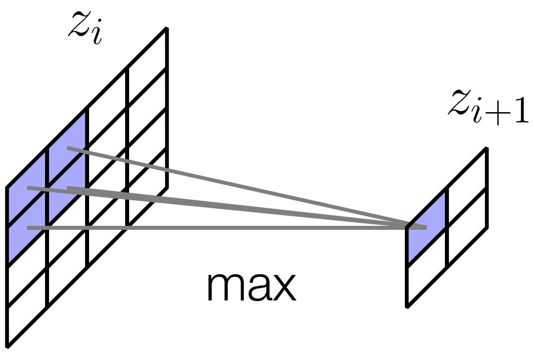


Convolutional neural networks

Constrain weights: require that activations in following layer be a “local” function of previous layer, and share weights across all locations

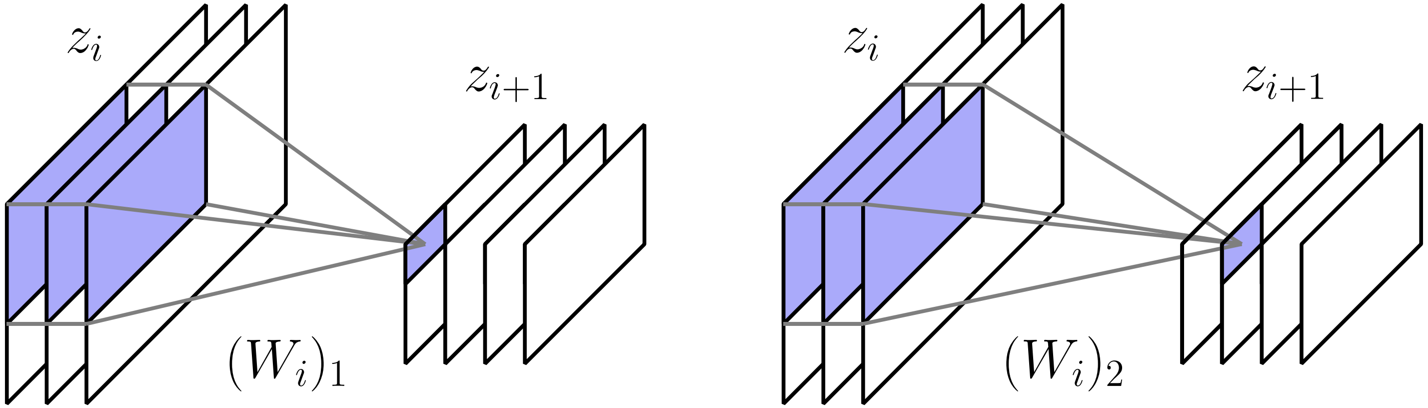


Also common to use *max-pooling* layers that take maximum over region



Convolutional networks in practice

Actually common to use "3D" convolutions to combine multiple channels, and use multiple convolutions at each layer to create different features



Convolutions are still linear operations, and we can take gradients using backpropagation in much the same manner

Predicting sequential data

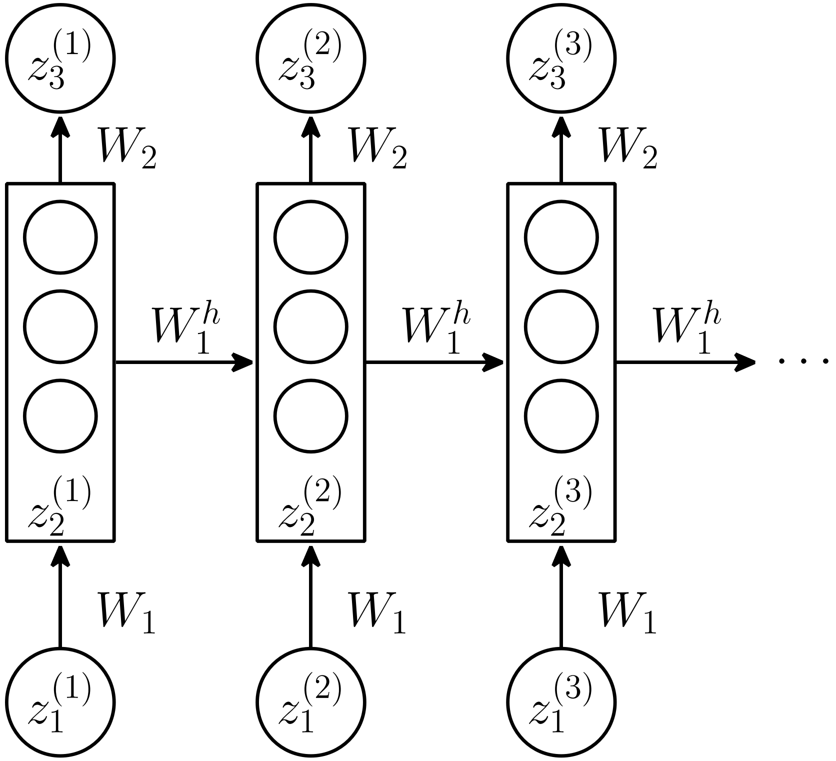
In practice, we often want to predict a *sequence* of outputs given a *sequence* of inputs

Just predicting each output independently would miss crucial information

Many examples: time series forecasting, sentence labeling, part of speech tagging, etc

Recurrent neural networks

Maintain state over time, activations are a function of current input and *previous* activations



$$z_{i+1}^{(t)} = f_i\left(W_i x^{(t)} + W_i^h z^{(t-1)} + b_i\right)$$
$$h_\theta\left(x^{(t)}\right) = z_k^{(t)}$$

Recurrent neural networks in practice

Traditional RNNs have trouble capturing long-term dependencies

More typical to use a more complex hidden unit and activations, called a long short term memory (LSTM) network

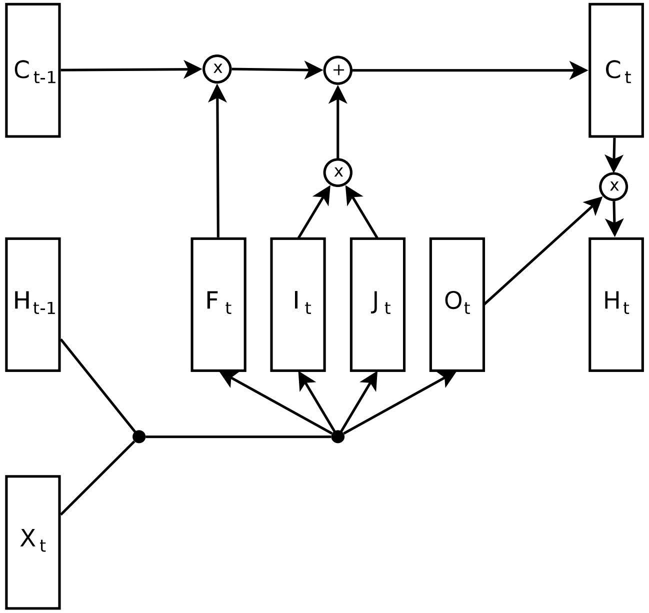


Figure from (Jozefowicz et al., 2015)

Outline

Recent history in machine learning

Machine learning with neural networks

Training neural networks

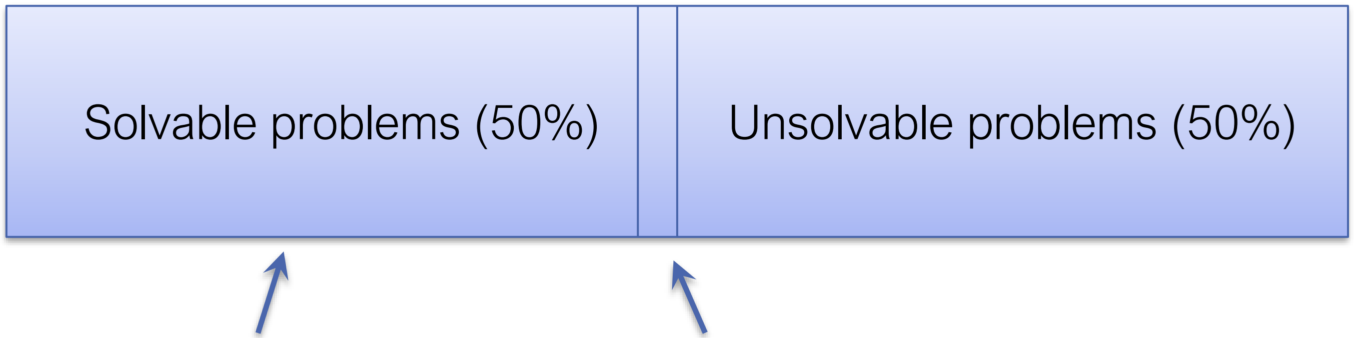
Specialized neural network architectures

Deep learning in data science

Deep learning in data science

What role does deep learning have to play in data science?

Data problems we would like to solve



Problems that can use "simple" machine learning (45%)

Problems that need, e.g. *new* deep learning (5%)

Solving data science problems with deep learning

When you come up against some machine learning problem with "traditional" features (i.e., human-interpretable characteristics of the data) do *not* try to solve it by applying deep learning methods first

Use linear regression/classification, linear regression/classification with non-linear features, or gradient boosting methods instead

If these still don't solve your problem *and* you can visualize the data in a way that lets you solve it "manually", *or* if you really want to squeeze out a 1-2% improvement in performance, *then* you can apply deep learning

The exceptions

However, it’s also undeniable that deep learning has made remarkable progress for structured data like images, audio, or text

For these types of data, you can use an *already trained* network as a *feature extractor* (i.e., a way of mapping the data to some alternatively, probably lower dimensional representation)

Example: Image processing with VGG

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure from Simonyan and Zisserman, 2015

VGG network (Simonyan and Zisserman, 2015), trained on ImageNet 1000-way classification of images

Given a new image classification problem, take pre-trained VGG network, take the last layer of weights, and use them as features

Can also “finetune” last few layers of a network to specialize to a new task

Example: text processing with word2vec

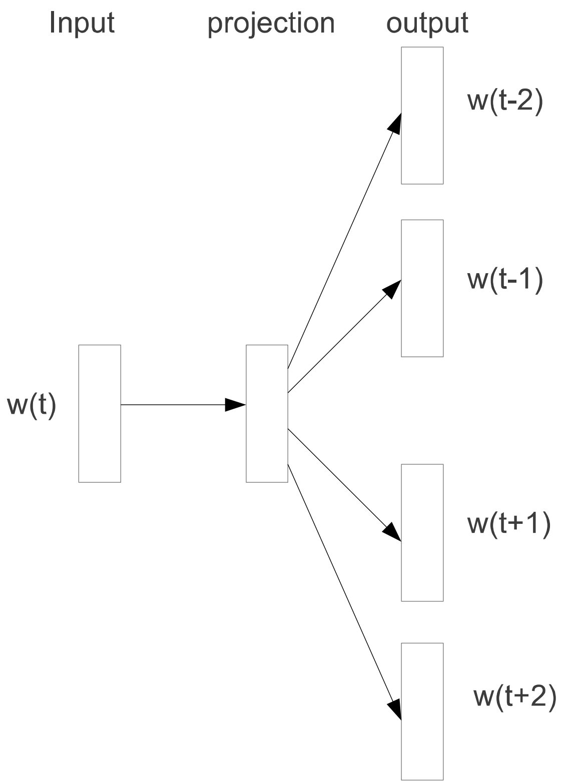


Figure from Mikolov, et al., 2013

word2vec (Mikolov, et al., 2013) is a method developed for predicting surrounding words from a given word

To do so, it creates an "embedding" for every word that acts as a good surrogate for the things this word can mean, pre-trained versions available

Bottom line: instead of using bag of words, use word2vec to get a vector representation of each word in a corpus

Example: text processing with BERT

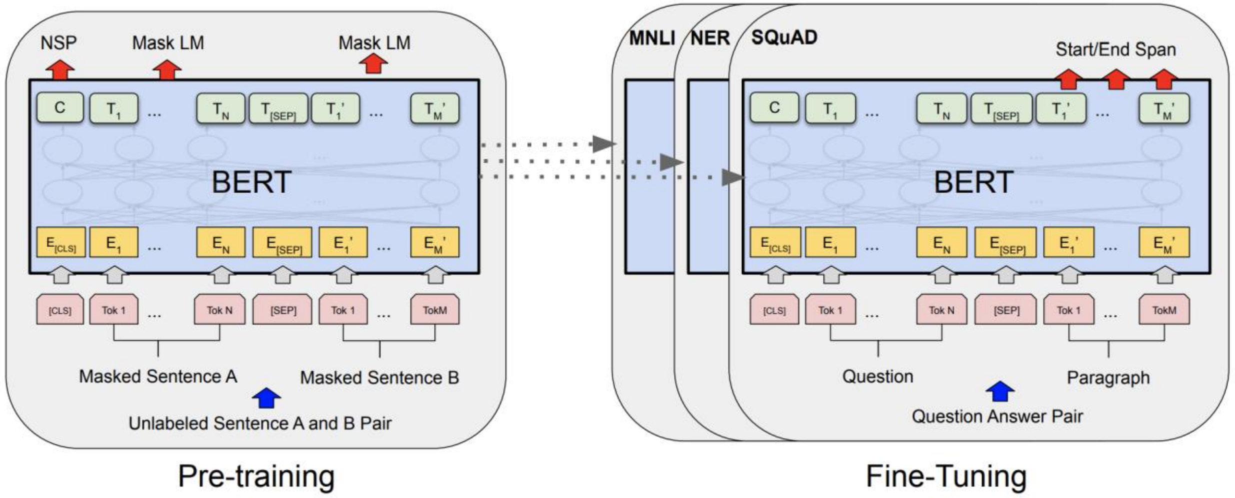


Figure from Devlin, et al., 2018

BERT (Bidirectional Encoder Representations from Transformers), (Devlin et al., 2018) trains a language model to predict missing elements of a sentence and predict one sentence from another for two sentence pairs

At application time, can fine-tune this generic model to many other possible tasks such as question answering, sentence classification, etc