# Supervised Learning: Neural Networks and Deep Learning

Noah Simon & Ali Shojaie

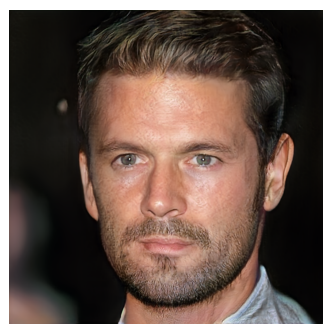
Aug 1-3, 2022
Summer Institute in Statistics for Big Data
University of Washington

# Neural Networks/Deep Learning

Recent interest in NNs and Deep NNs...

For supervised/unsupervised/reinforcement learning

- computer vision;
- speech recognition;
- natural language processing;
- strategy games.



# Neural Networks/Deep Learning

We will focus on supervised learning applications...

# 80/20 rule

In my experience, facility with linear/logistic/cox regression...

and their high dimensional extensions...

get near best performance on most high dimensional problems

---

Problems with highly structured features are the exception to this.

# [Deep] Neural Networks

What is a neural network?

Most generally, a NN is a function...

- determined by a set of parameters (or weights)...
- that takes in input "features"
- and outputs a "prediction" (or multiple predictions)

---

That sounds like the definition for linear regression!

What is special about NNs?

# [Deep] Neural Networks

What is special about NNs?

---

NNs can be extremely "expressive":

With a relatively small number of parameters...

  can approximate a rich set of functions

---

What does this mean for bias/variance tradeoff?

# Simple Neural Network

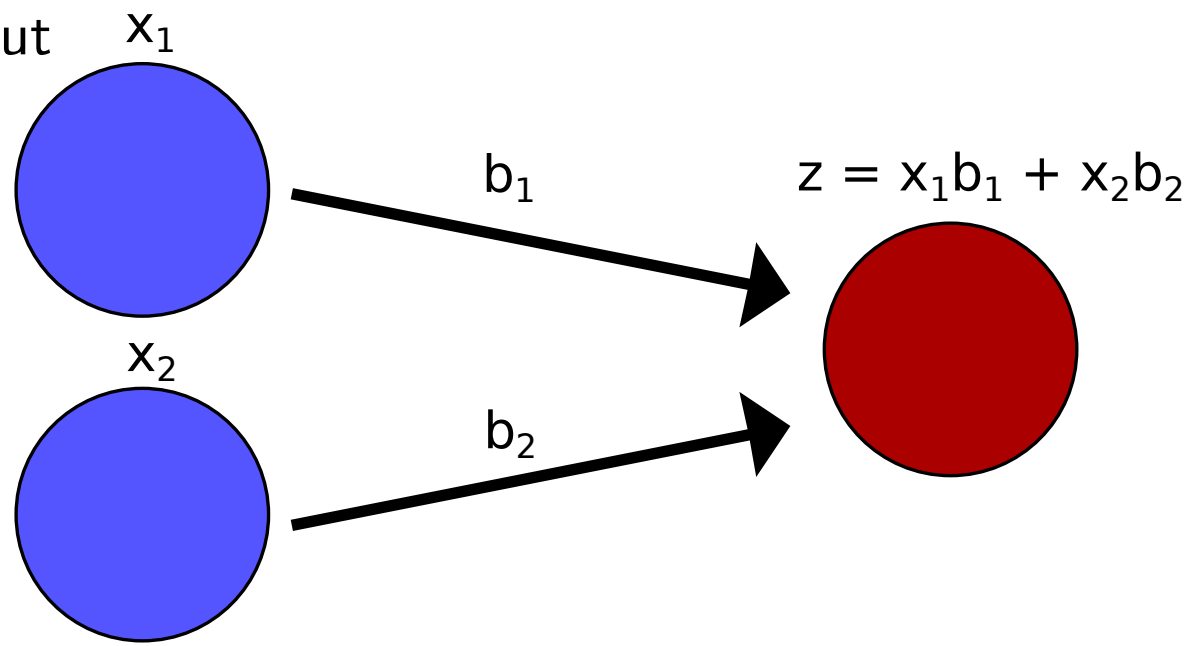
The simplest neural nets (often not called neural networks) have

- an "input layer": The features...
- an "output layer"

And use a simple linear map from input to output

# Simple Example with 2 Features

- $x_1, x_2$ - inputs
- $z$ - output




$$f_{b_1,b_2}(x_1, x_2) = x_1 b_1 + x_2 b_2$$

# Simple Neural Network

Often, a nonlinear transform is applied after linear combination

This non-linear transform is known as an "activation function"

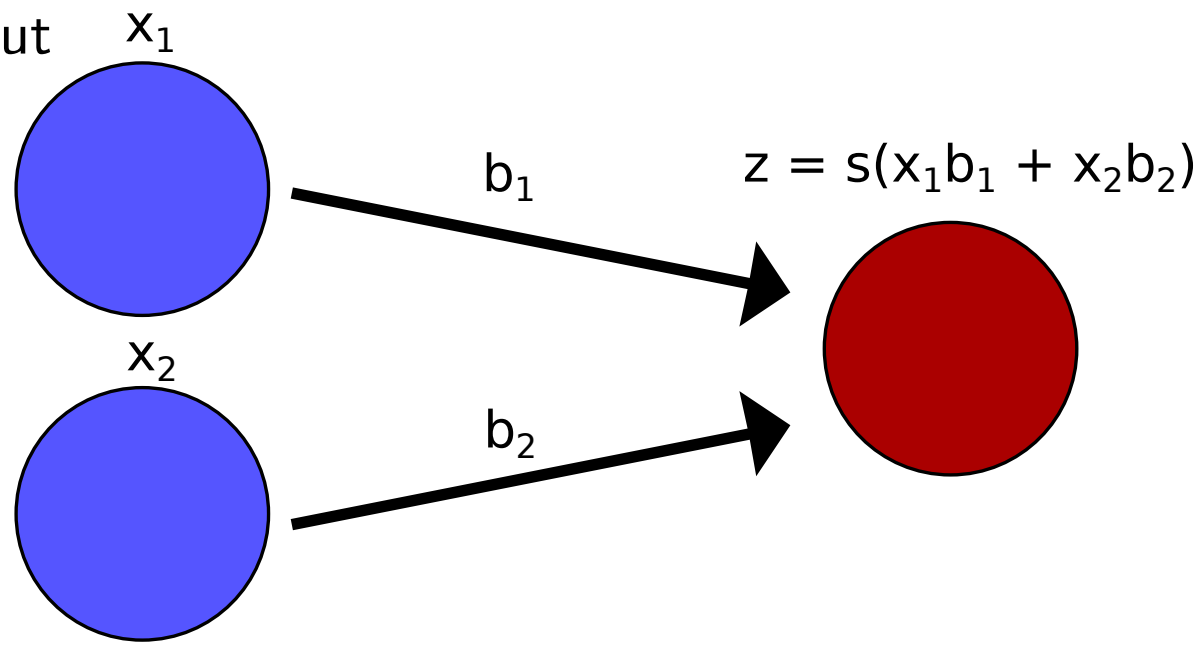
$$f_{b_1,b_2}(x_1, x_2) = S(x_1 b_1 + x_2 b_2)$$

most common activation function is `expit`

$$S(z) \equiv \frac{e^z}{1 + e^z}$$

# Simple example with 2 features

- $x_1, x_2$ - inputs
- $z$ - output




$$f_{b_1,b_2}(x_1, x_2) = S(x_1 b_1 + x_2 b_2)$$

# Hidden Layers

These simple examples connect NNs back to simple objects

(linear/logistic regression)

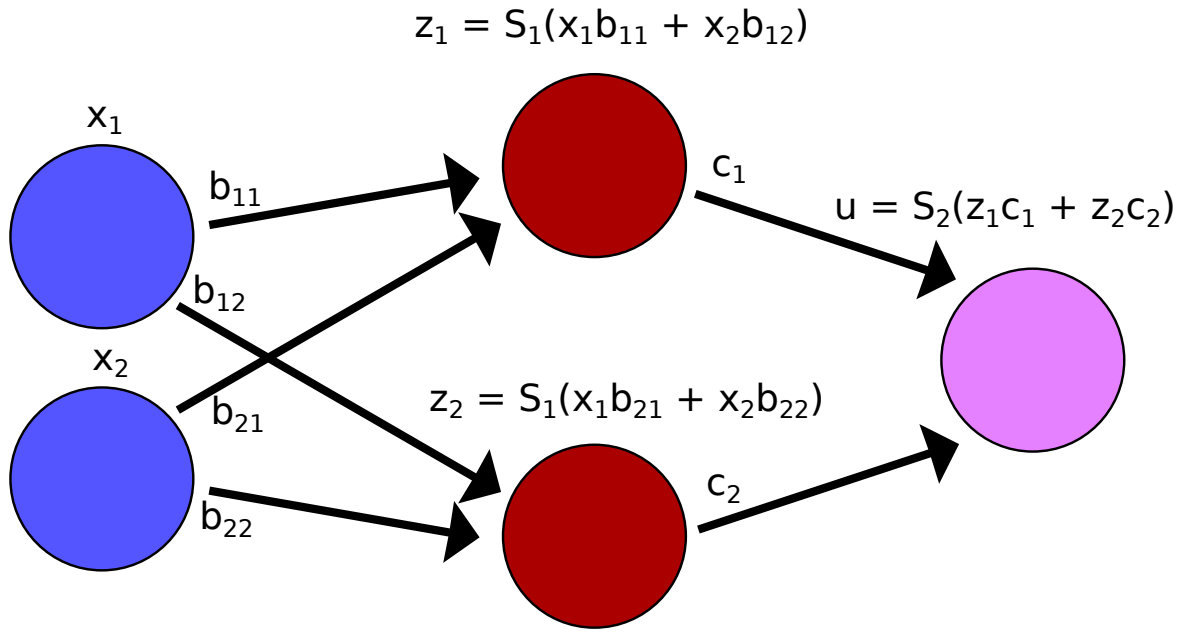
But do not harness the power of neural networks.

This power comes through hidden layers.

These hidden layers allow NNs to be extremely expressive

# Example with 2 Features, 1 Hidden Layer

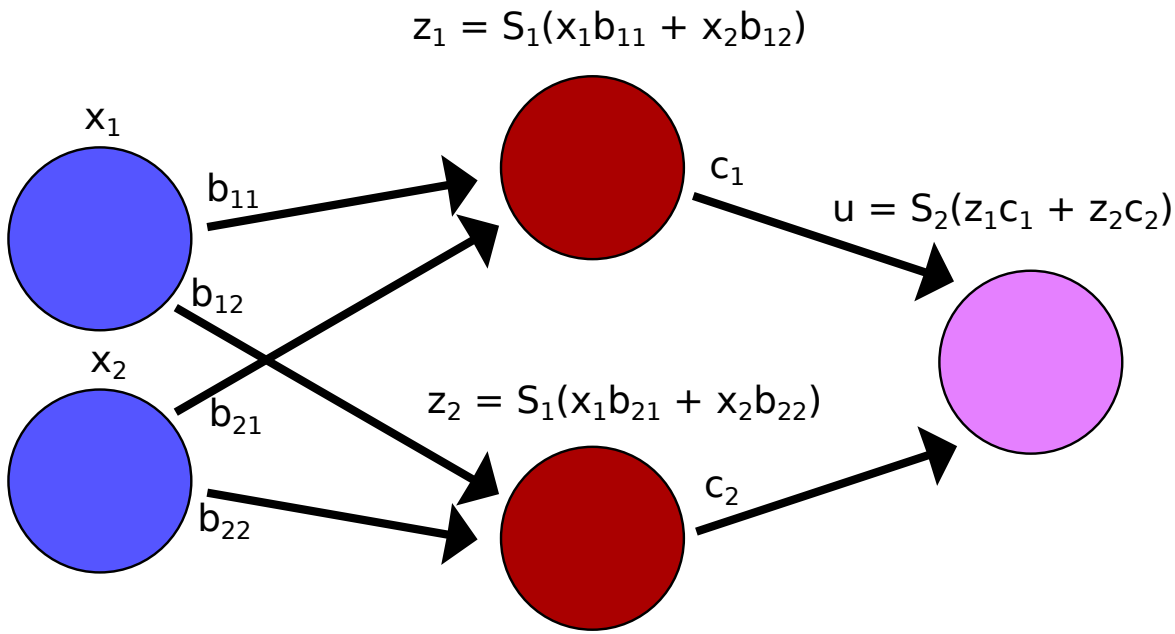
- $x_1, x_2$ - inputs
- $z_1, z_2$ - hidden llayer
- $u$ - output




---

$$f_{\mathbf{b},\mathbf{c}}(x_1, x_2) = S_2\Big[c_1 * S_1(x_1 b_{11} + x_2 b_{12}) + c_2 * S_1(x_1 b_{21} + x_2 b_{22})\Big]$$

# Fully Connected Multilayer NN




---

At a given layer, each node...

takes as input, a linear combination of **all** nodes on previous layer

applies a pre-specified activation function

and outputs that value to **all** nodes on next layer

# Training a NN

Given a fixed topology (width/depth of hidden layers)...

How do we select the coefficient values in our linear combination...

to build a predictive model?

---

For a given topology, let $\theta$ denote the vector of coefficient values

Then we can thing of the NN as the function $f_\theta(\mathbf{x})$.

We choose the $\theta$-value that best fits our data!

$$\hat{\theta} = \text{argmin}_\theta \sum_{i=1}^{n} \left(y_i - f_\theta\left(\mathbf{x}_i\right)\right)^2$$

or the equivalent for binary/survival data.

# Training a NN

$$\hat{\theta} = \text{argmin}_\theta \sum_{i=1}^{n} \left(y_i - f_\theta\left(\mathbf{x}_i\right)\right)^2$$

---

Ok... but how do we get to the minimizer?

We use gradient descent!

AI people renamed the chain rule here...

and call it back propagation.

---

There are a bunch of other optimization tricks, that are very important in practice, relatively heuristic, but somewhat unimportant for big picture understanding of NNs

# Topology of a Neural Network

When using a NN, one must select:

- Number of hidden layers
- Number of nodes per layer
- Type of activation function to use

# Topology of a Neural Network

Selecting depth/width of network determines bias/variance tradeoff

Additional layers exponentially increase expressiveness of network

In some sense, adding layers increases order of interaction...

Not exactly true with non-linear $S$ — sufficiently wide 3 layer network is universal approximator

How do we tune/evaluate topology??

# Topology of a Neural Network

In practice, even modest NNs with 1 hidden layer and a handful of hidden nodes can require thousands of observations to fit

Effective use of deeper (more hidden layer) networks requires...

- huge numbers of observations (10 million +) or,
- structured topology (to be discussed shortly)

# Activation Functions

A few activation functions

- Identity: $S(z) = z$

  generally used in final layer for continuous response

---

- expit/soft-max: $S(z) = e^z/(1 + e^z)$

  generally used in final layer for binomial/multinomial

  classical choice in hidden layers

---

- linear rectifier unit (reLU): $S(z) = (z)_+$

  modern choice for hidden layers

---

Often use different activation function for hidden vs output layers

# Regularization/Penalization

Penalization is used for tweaking the bias/variance tradeoff

Generally a ridge penalty is used

$$\hat{\theta} = \text{argmin}_\theta \sum_{i=1}^{n} \left(y_i - f_\theta\left(\mathbf{x}_i\right)\right)^2 + \lambda \left\|\theta\right\|_2^2$$

There is recent work on using lasso penalties, and different tuning parameters for each level.

How do we select $\lambda$??

# Basic NN Summary

NNs are a method for creating expressive estimators

Topology of the network determines bias/variance tradeoff

Can be combined other regression tools eg. regularization

# Convolutional Neural Networks

In many predictive applications, data has additional spatial structure to leverage

Image recognition is one important example.

Here, relative location of pixels is more important than absolute

e.g. a cat in the upper left should be similarly informative to a cat in the center

Can encode this in NNs with a very simple idea:

Convolutional filters/Convolutional NNs

# Convolutional Neural Networks

Standard NNs are made up of multiple fully connected (FC) layers...

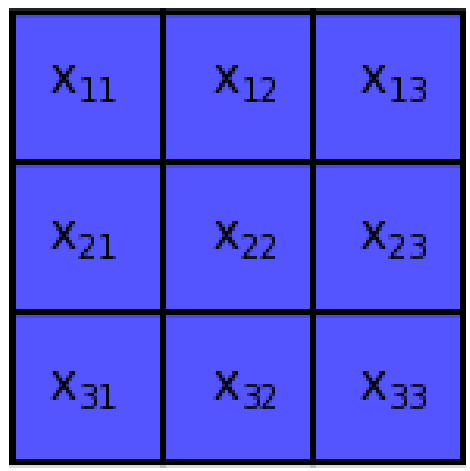
Convolutional Neural Networks (CNNs) combine FC layers...

- with convolutional layers

We give a simple picture to illustrate

# Convolutional Layer

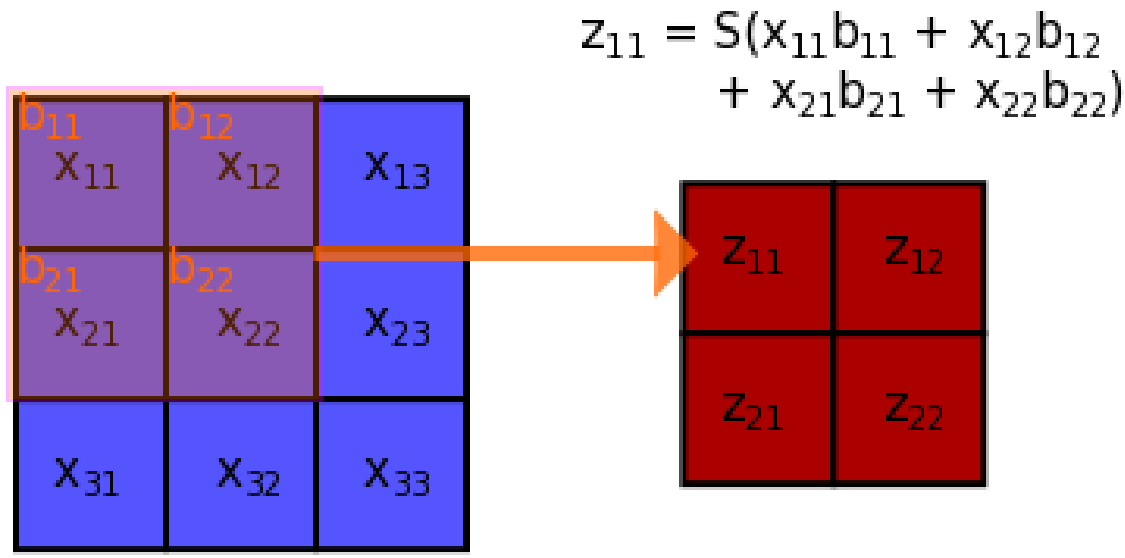
Consider data from an image with 9 pixels



# Convolutional Layer

Here we use a $2 \times 2$ filter...

resulting in 4 hidden nodes

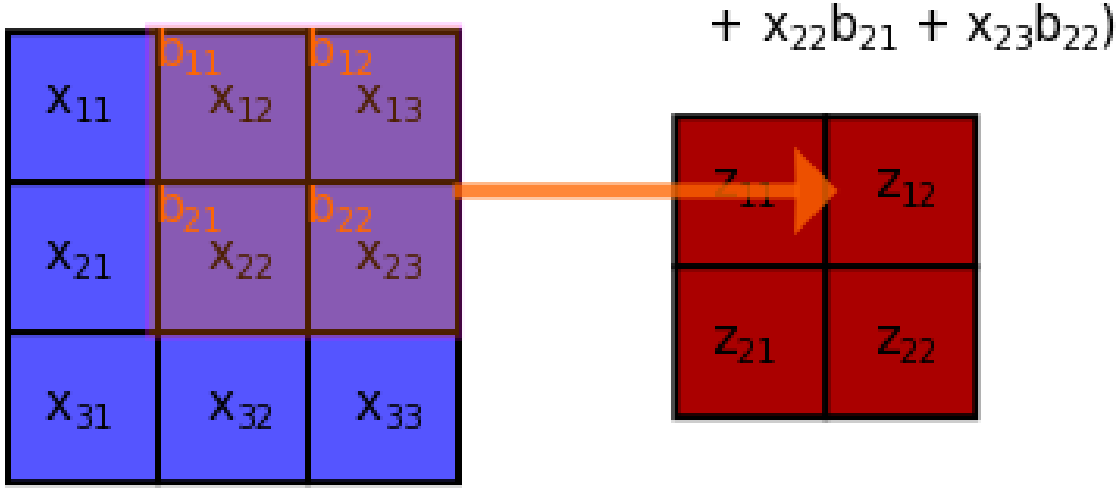


# Convolutional Layer

The input pixels change for each hidden node...

but the coefficients stay the same

$$z_{12} = S(x_{12}b_{11} + x_{13}b_{12} + x_{22}b_{21} + x_{23}b_{22})$$

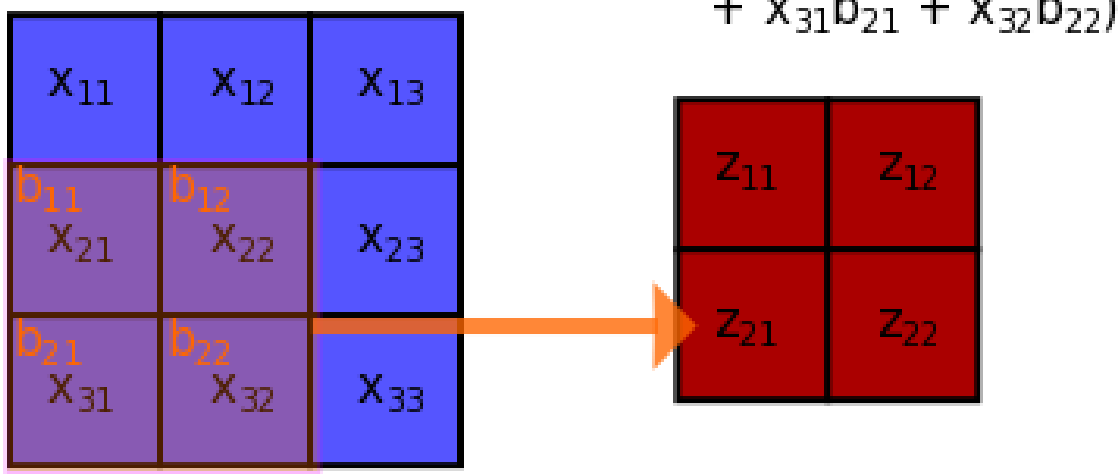


# Convolutional Layer

The input pixels change for each hidden node...

but the coefficients stay the same

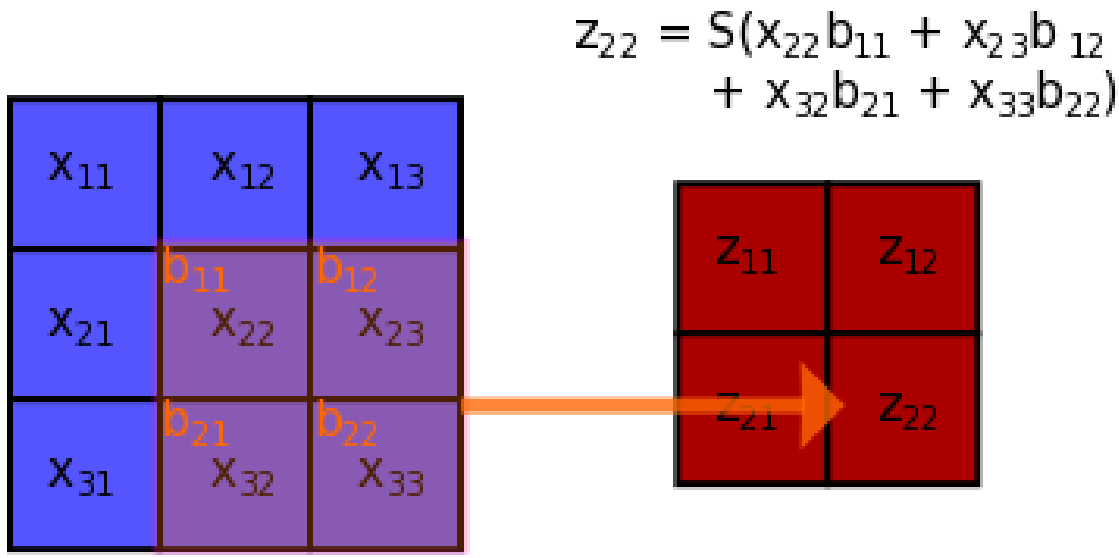
$$z_{21} = S(x_{21}b_{11} + x_{22}b_{12} + x_{31}b_{21} + x_{32}b_{22})$$



# Convolutional Layer

The input pixels change for each hidden node...

but the coefficients stay the same



# Pooling Layer

"Layers" are also included that aggregate adjacent nodes in a prespecified way.

Often this is done by taking the simple `average`

Also sometimes the `max`

---

Mostly for computation

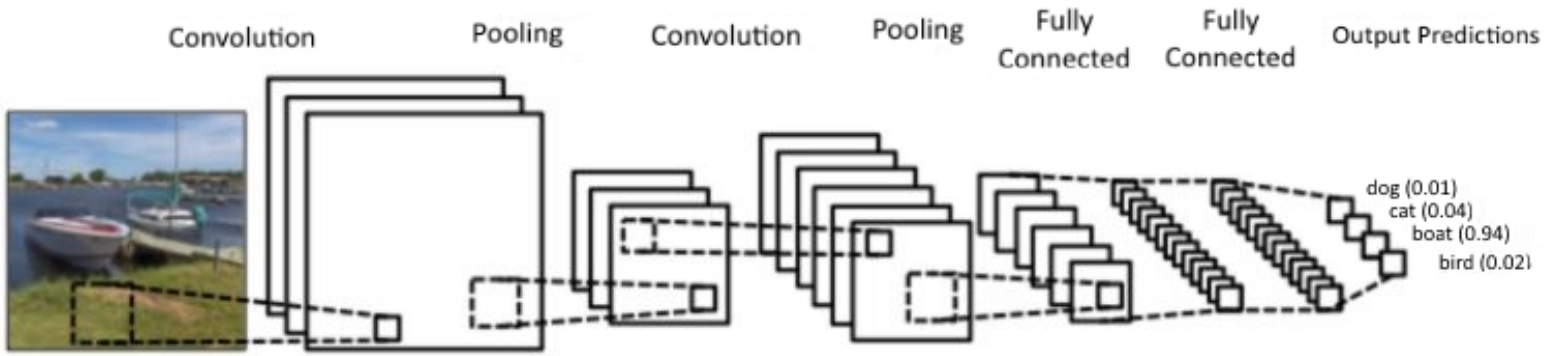
pooling layer quarters number of future inputs

# Convolutional Networks

To flexibly design a NN can combine...

- Convolutional Layers
- Pooling Layers
- Fully Connected Layers
- and an output layer

# CNN Architecture




---

taken from `Neural Networks with R by Balaji Venkateswaran, Giuseppe Ciaburro`

# CNN Overview

CNNs are a version of NNs that assume special input structure:

Local combinations of features are important...

and the information they contain is *locationally equivariant*

---

Has been *extremely* useful in image/speech classification

# Recurrent Neural Networks

One final widely used NN infrastructure...

Particularly useful for sequentially ordered data of irregular length

eg. measurements over time on a patient

# Recurrent Neural Networks - example

Suppose we aim to predict heart-attack within 1 month...

For each patient, we measure lab values + outcome each month

Different patients have different lengths of followup

---

How do we build a predictive model?

Recurrent Neural Nets (RNNs) are one attractive approach

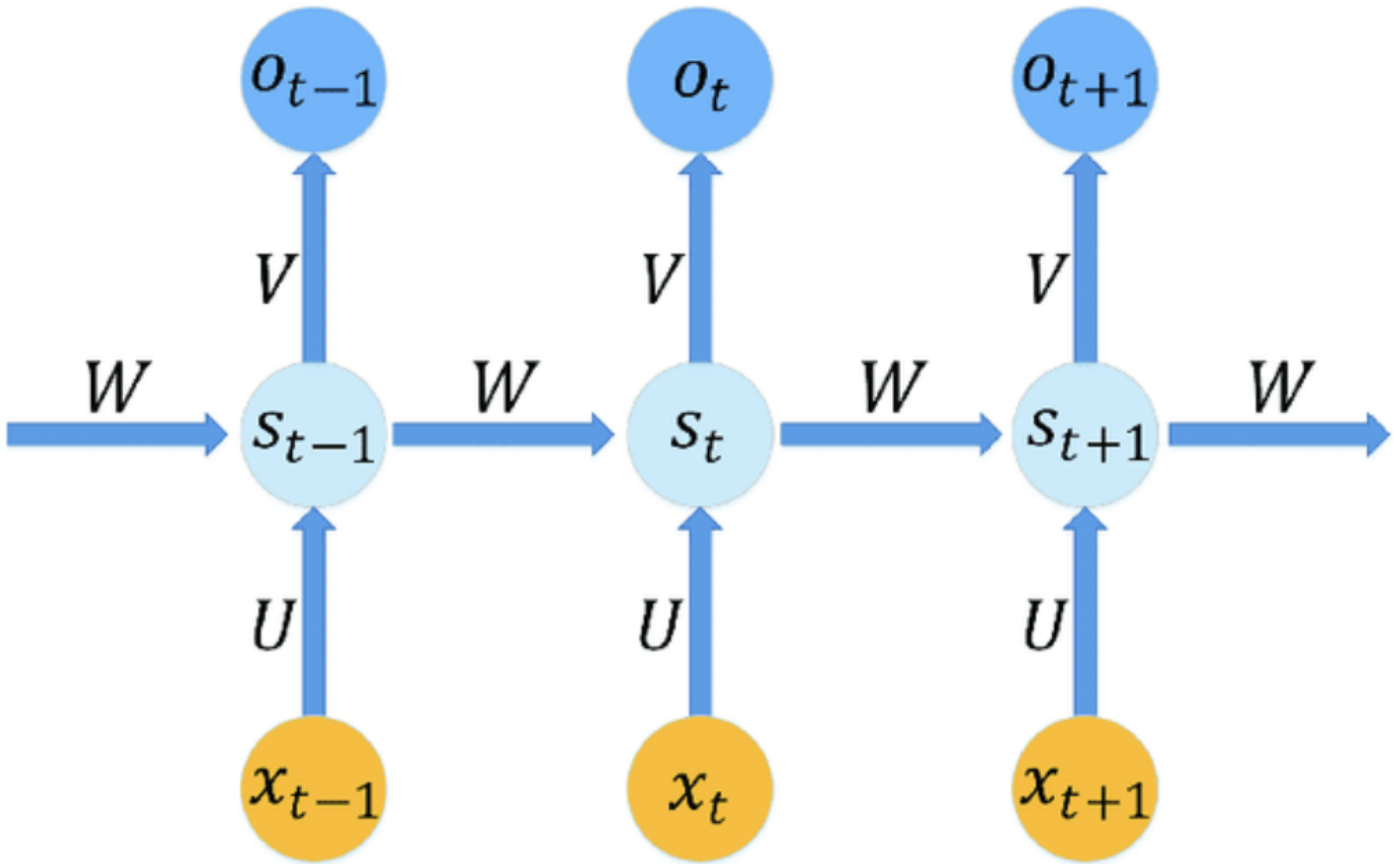
# Recurrent Neural Network

Like the other NNs we talked about...

RNNs are just a way of specifying a flexible function that takes an input, and gives a prediction

RNNs can take an input sequence of variable length

# RNNs




borrowed from *A deep learning framework for financial time series using stacked autoencoders and long-short term memory* by Bao et al

# RNN uses

RNNs are increasingly used, eg. in

- Natural Language Processing
- Audio Processing
- Protein binding
- DNA sequence alignment

# NN Big Picture

NNs are very flexible function estimators

with cool names...

---

Is that necessarily a good thing???