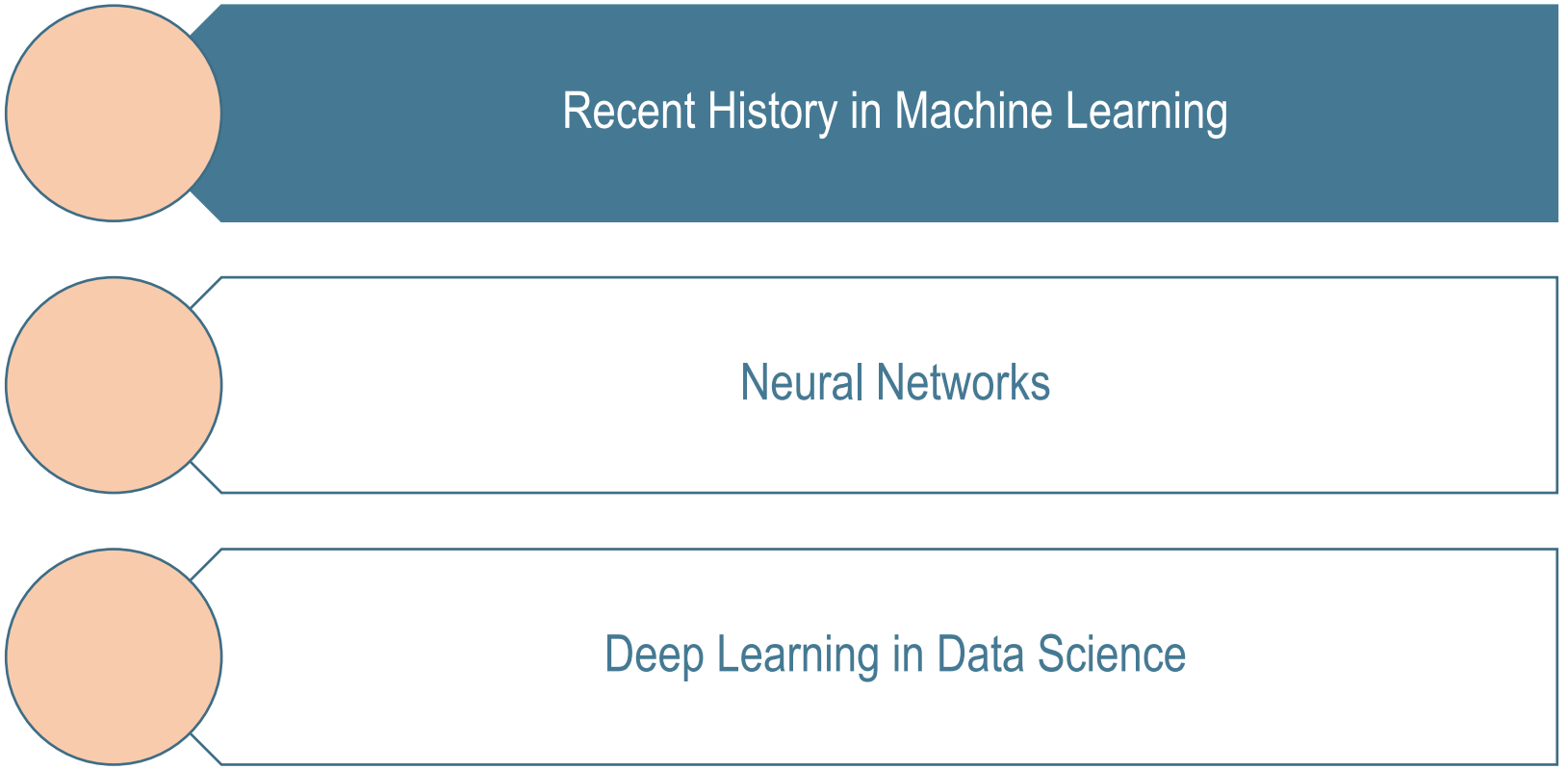
Lecture 8 – Supervised Learning

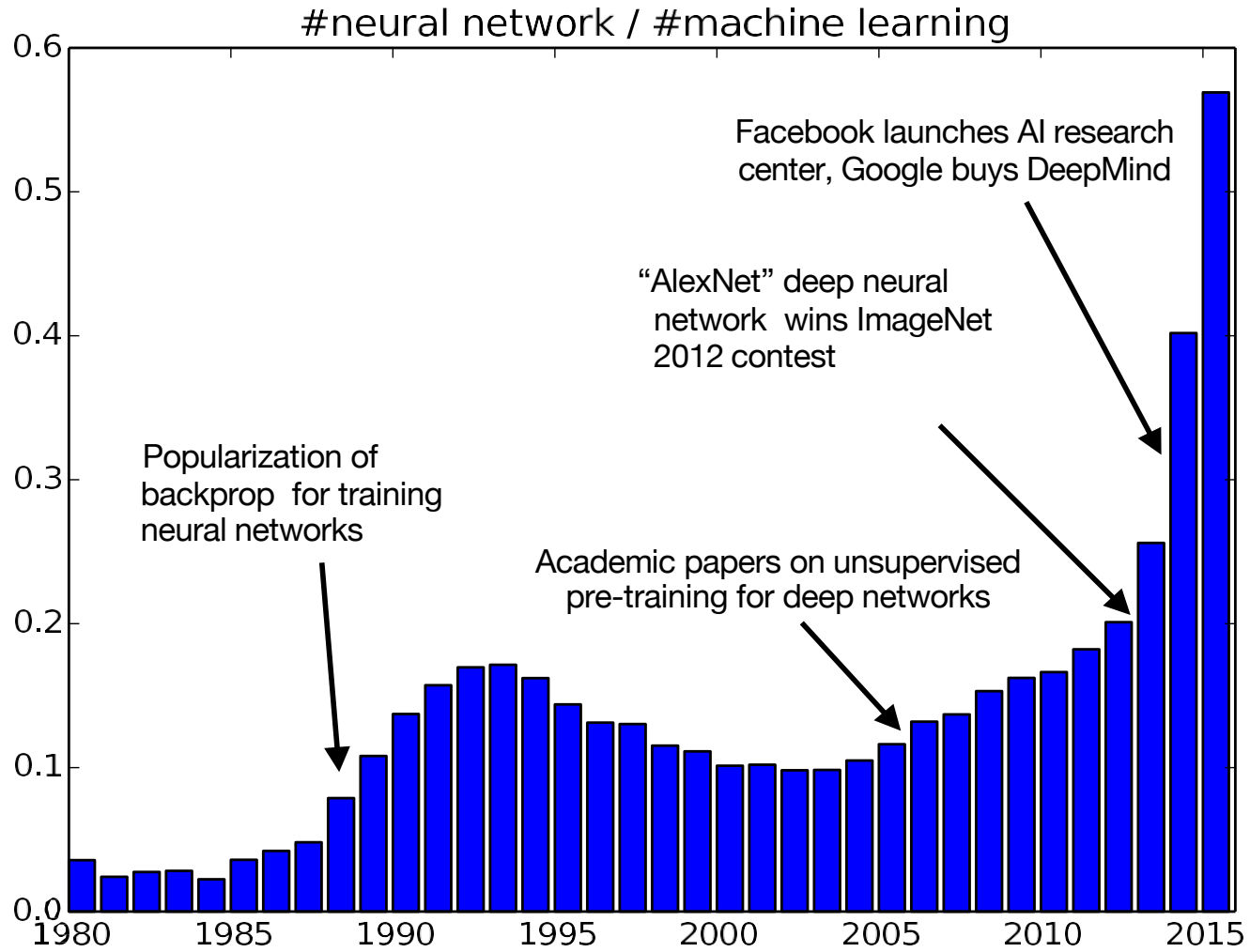
Neural Networks

Information Systems for Sustainable Society (is3) | WiSo Faculty | Univ.-Prof. Dr. Wolfgang Ketter | 11/30/22

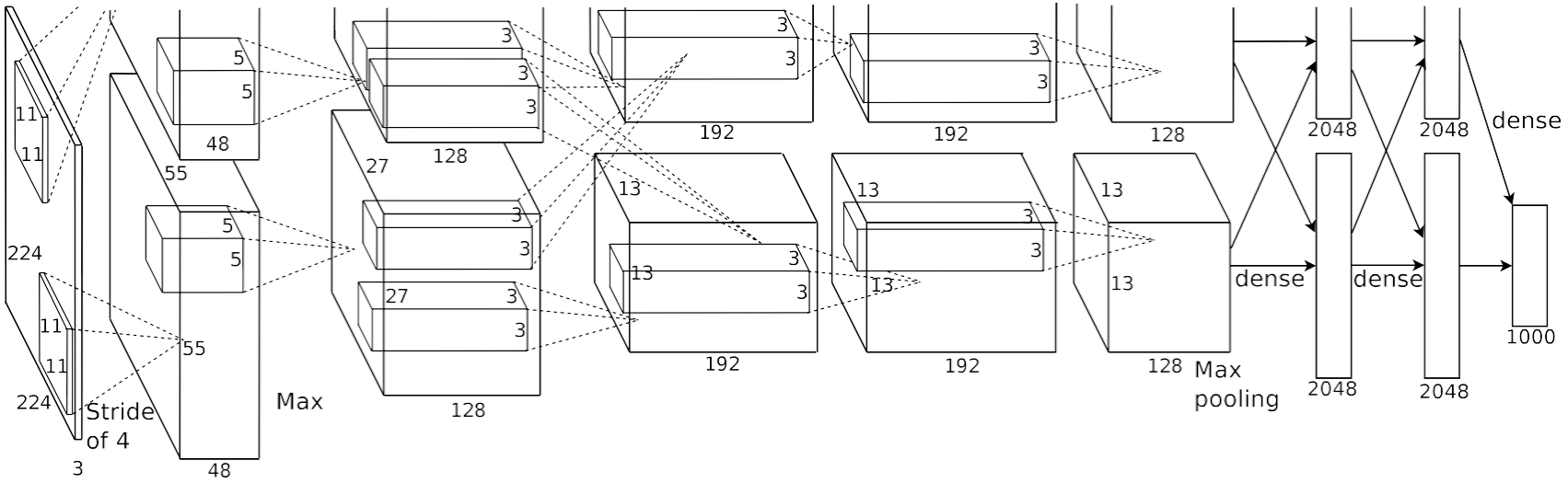
Agenda


Recent History in Machine Learning
Neural Networks
Deep Learning in Data Science

Recent history in machine learning

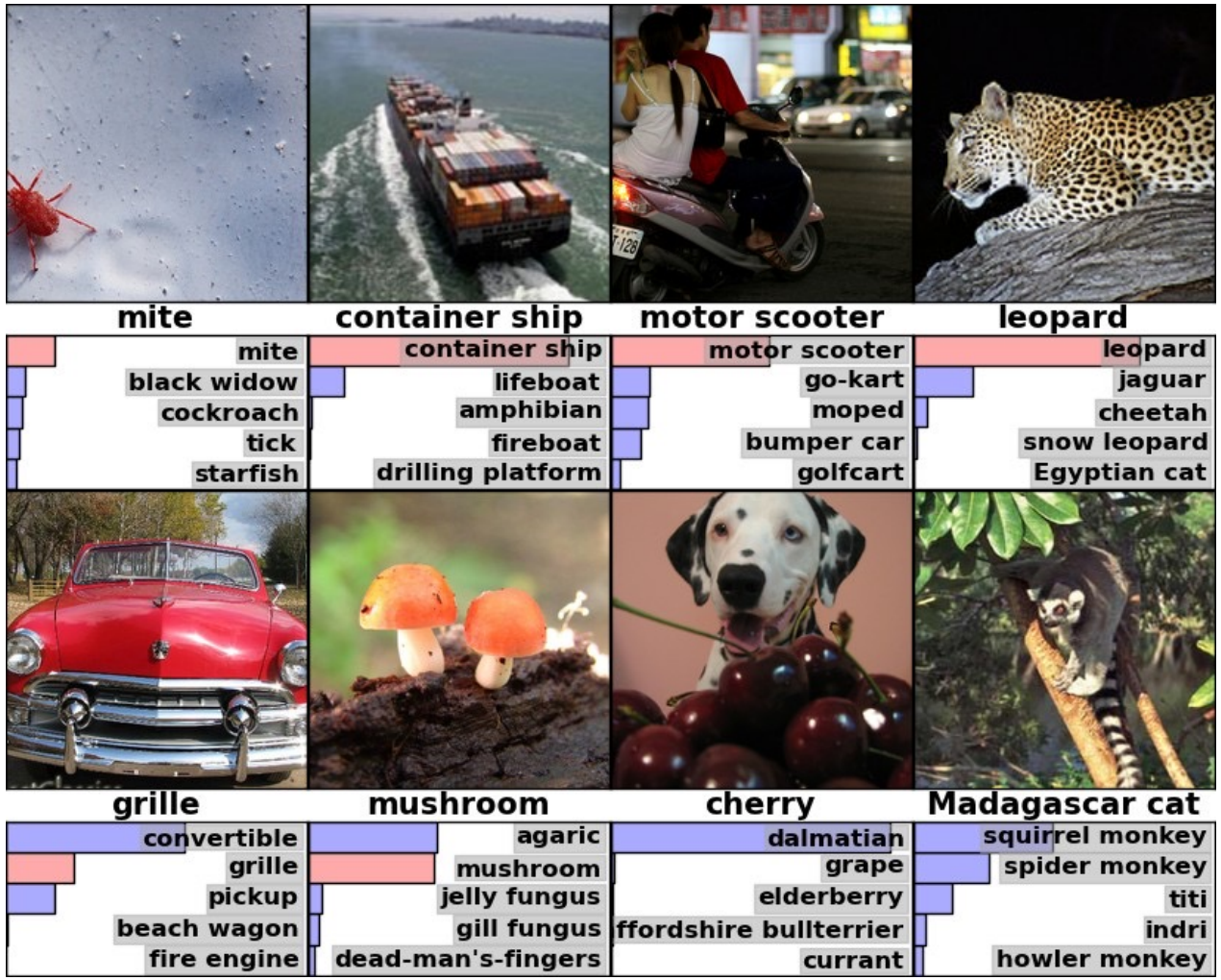


AlexNet



"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)

ImageNet classification



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.

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

AlphaGo

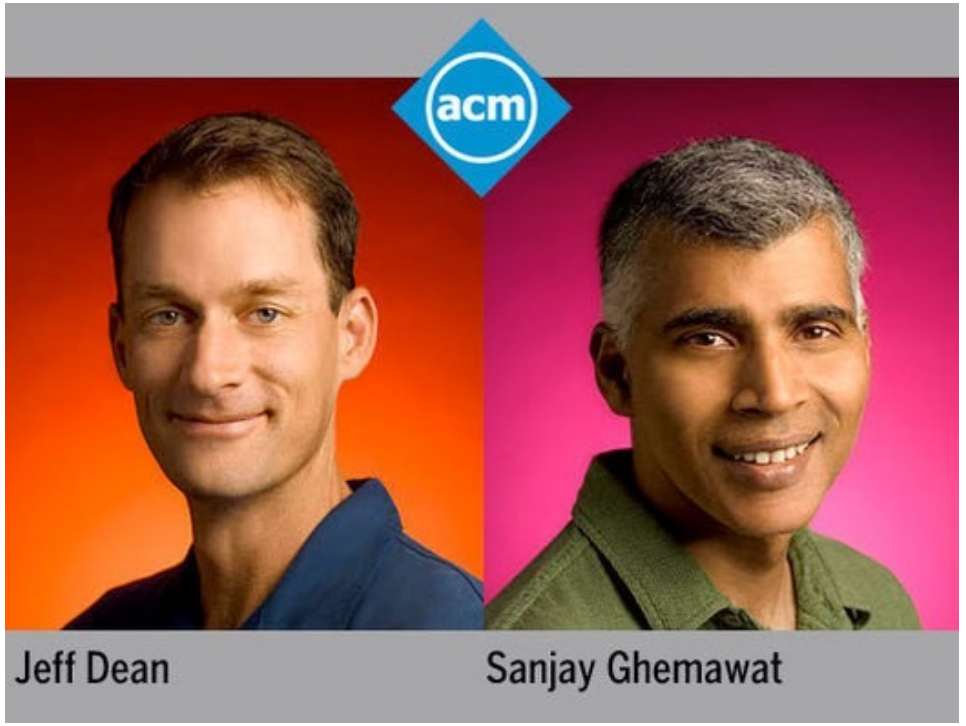
Artificial intelligence: Google's AlphaGo beats Go master Lee Se-dol

12 March 2016 | Technology

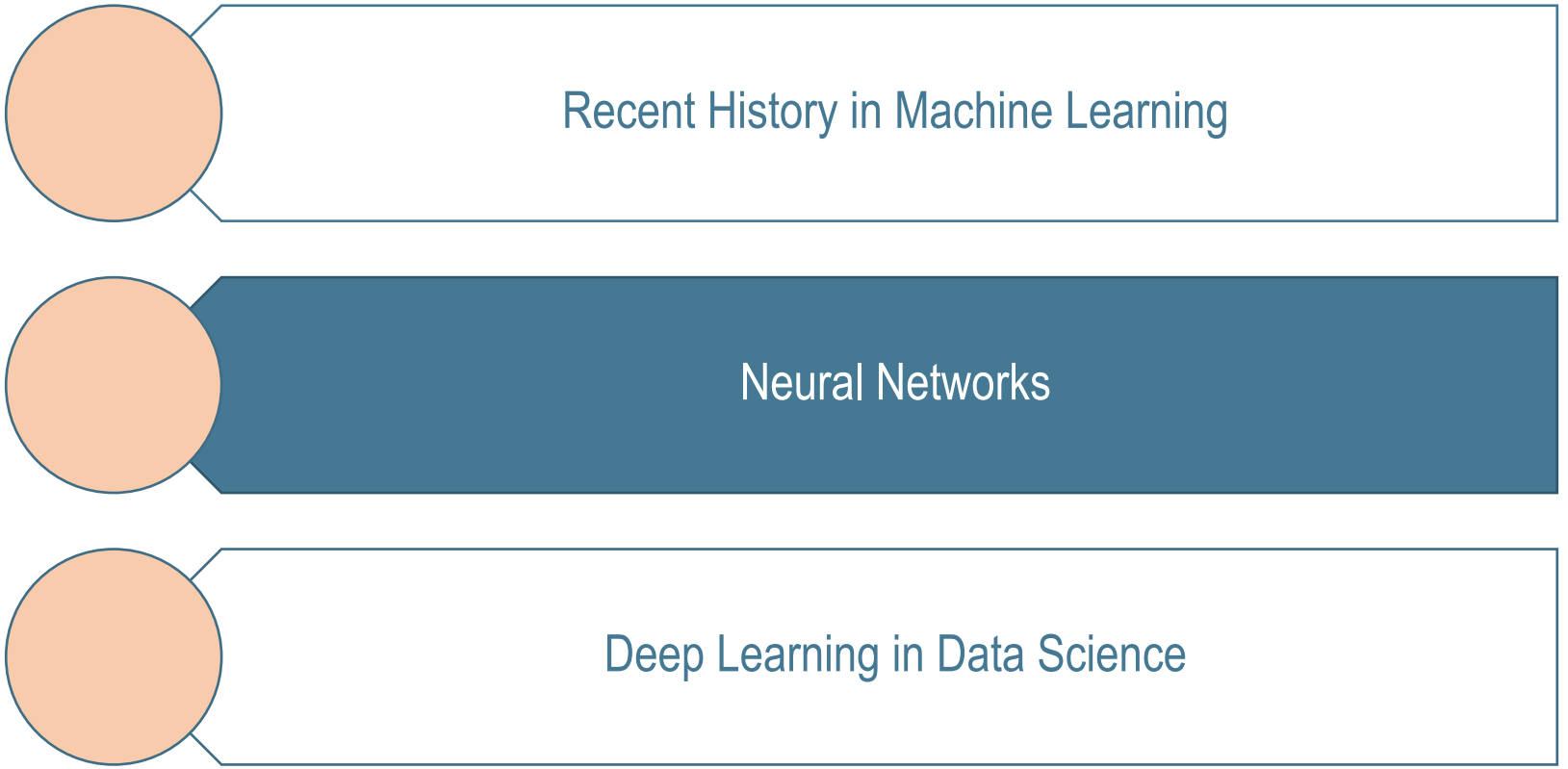
The Friendship That Made Google Huge

Coding together at the same computer, Jeff Dean and Sanjay Ghemawat changed the course of the company—and the Internet.

By James Somers,
The New Yorker
December 10, 2018



Agenda


Recent History in Machine Learning
Neural Networks
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

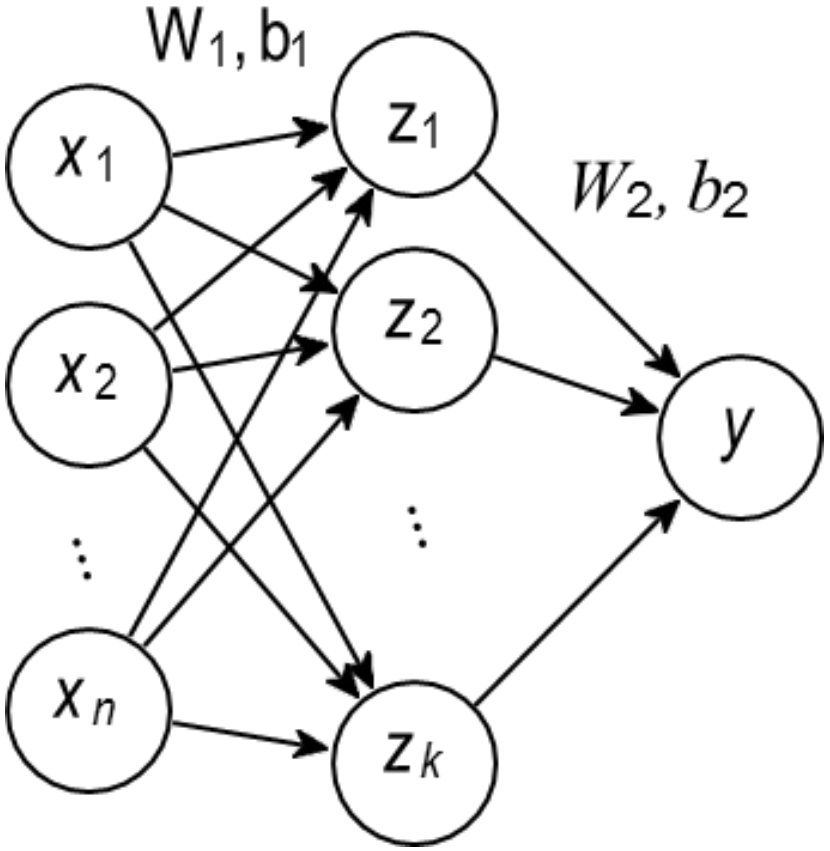
- Until now, we have (mostly) considered machine learning algorithms that use a linear hypothesis class
  - $h_\theta(x) = \theta^T \phi(x)$
  - where $\phi : \mathbb{R}^n \rightarrow \mathbb{R}^k$ denotes some set of typically non-linear features
- Example: polynomials, radial basis functions
- The 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 by itself?

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
  - $h_\theta(x) = W_2\phi(x) + b_2 = W_2(W_1 x + b_1) + b_2$
  - where $\theta = \{W_1 \in \mathbb{R}^{k \times n}, b_1 \in \mathbb{R}^k, W_2 \in \mathbb{R}^{1 \times k}, b_2 \in \mathbb{R}\}$
- By convention, we are going to separate out the "constant feature" into the b terms

Issue with linear feature learning

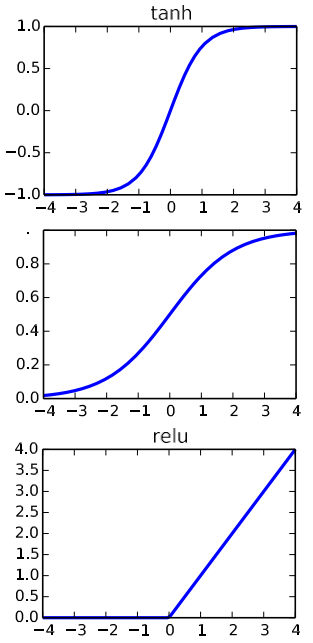
- We can depict the above network graphically using the following figure:



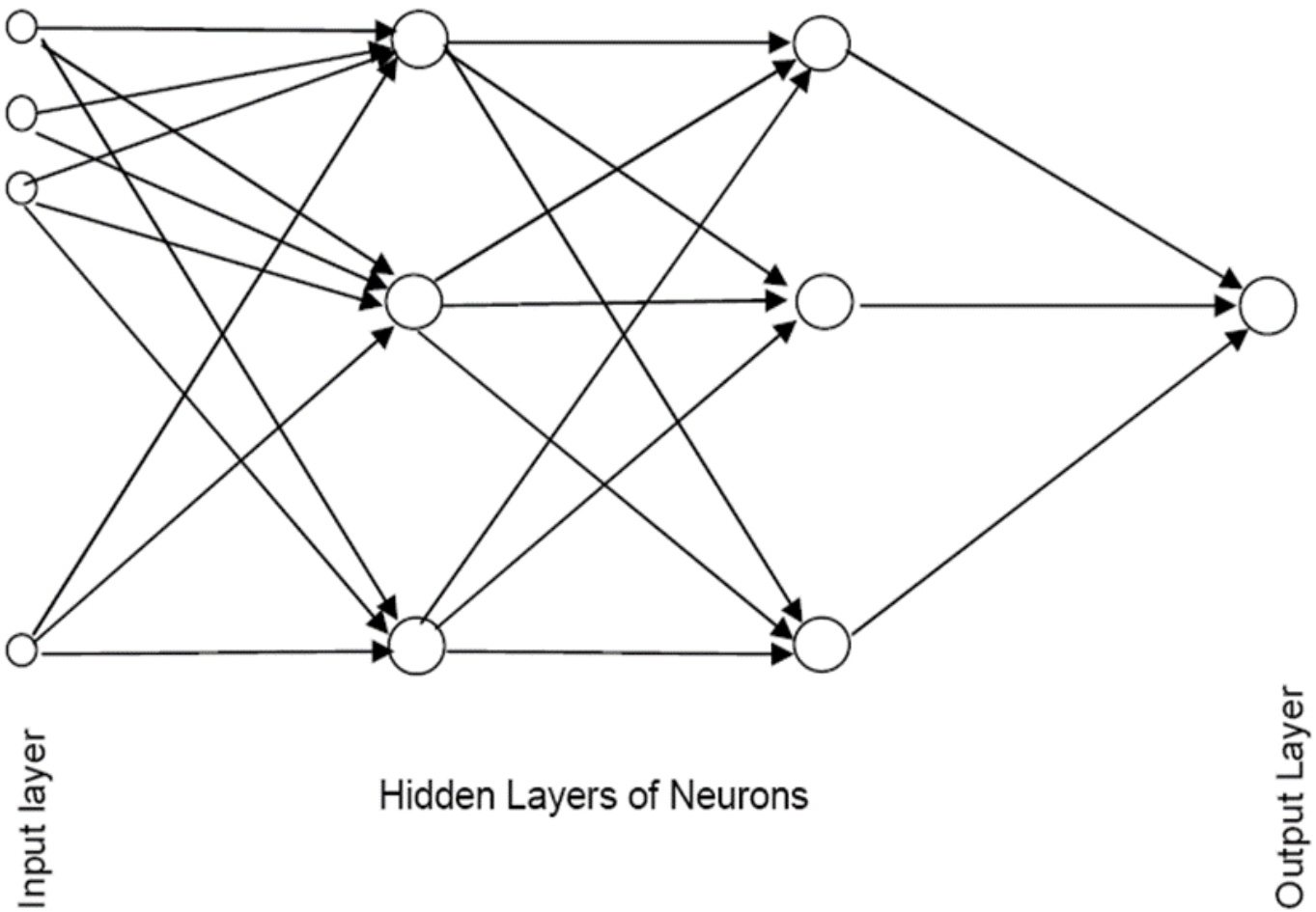
- But there is a problem:
  - $h_\theta(x)$
    $= W_2(W_1 x + b_1) + b_2$
    $= \tilde{W} x + \tilde{b}$
  - i.e., we are still just using a linear classifier (the apparent added complexity is actually not changing the underlying hypothesis function)

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 $f_1, f_2 : \mathbb{R} \to \mathbb{R}$ 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):** $f(x) = \max\{x, 0\}$

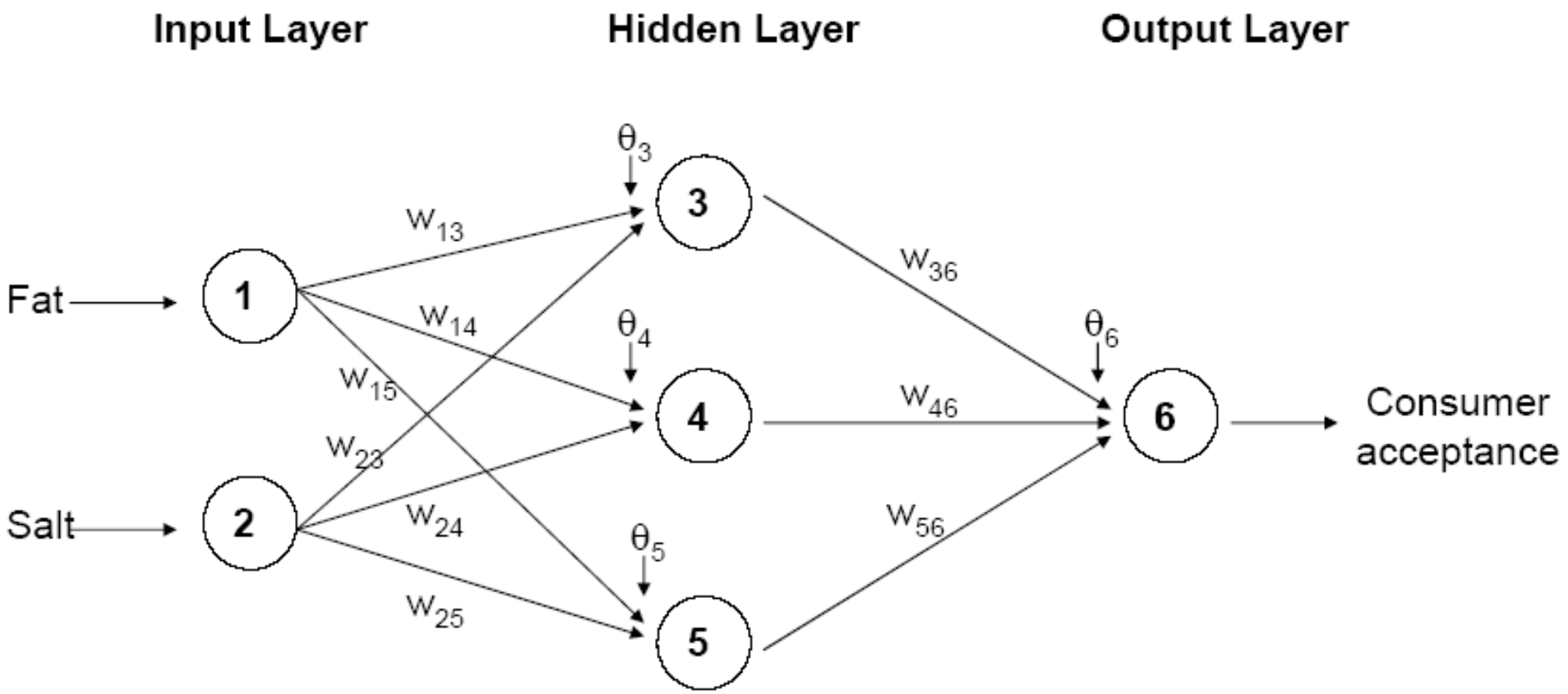


Network Structure



- Multiple layers
  - Input layer (raw observations)
  - Hidden layers
  - Output layer
- Nodes
- Weights (like coefficients, subject to iterative adjustment)
- Bias values (also like coefficients, but not subject to iterative adjustment)

Example: Using fat & salt content to predict consumer acceptance of cheese



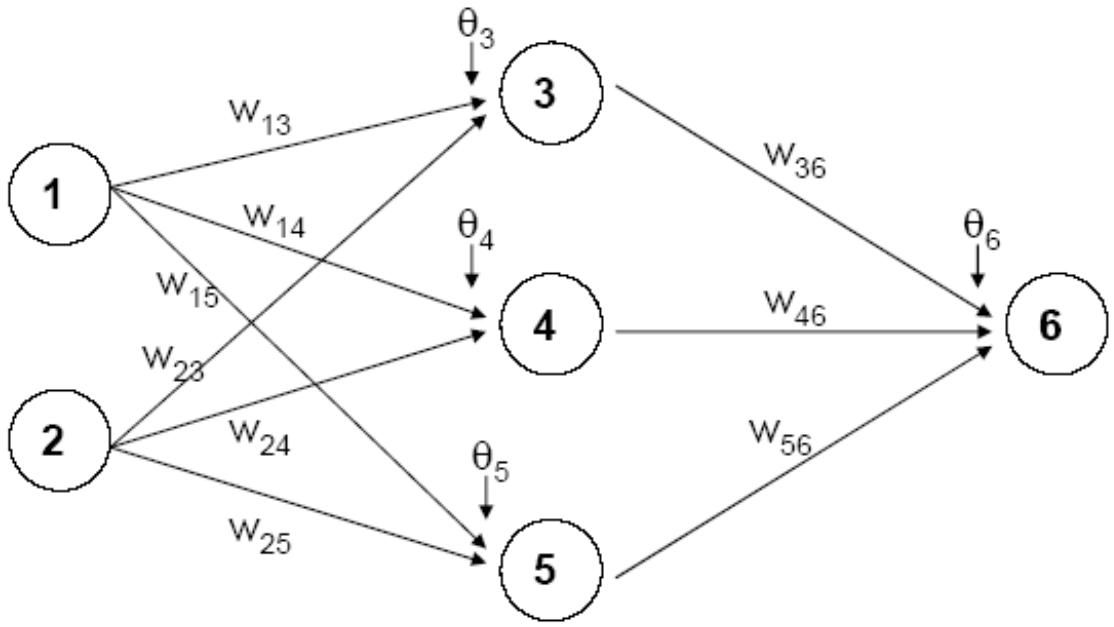
The Input Layer

- Input of input layer = output of input layer nodes (i.e. no transformation)
- E.g., for observation 1:
  - Fat input = output = 0.2
  - Salt input = output = 0.9
- Output of input layer = input into hidden layer

Obs.	*Fat Score*	*Salt Score*	*Acceptance*
1	0.2	0.9	1
2	0.1	0.1	0
3	0.2	0.4	0
4	0.2	0.5	0
5	0.4	0.5	1
6	0.3	0.8	1

The Hidden Layer

- In this example, the hidden layer has 3 nodes
- Each node receives as input the output of all input nodes
- Output of each hidden node is a function of the weighted sum of inputs
  - $output_j = g\left(\theta_j + \sum_{i=1}^{p} w_{ij} x_i\right)$

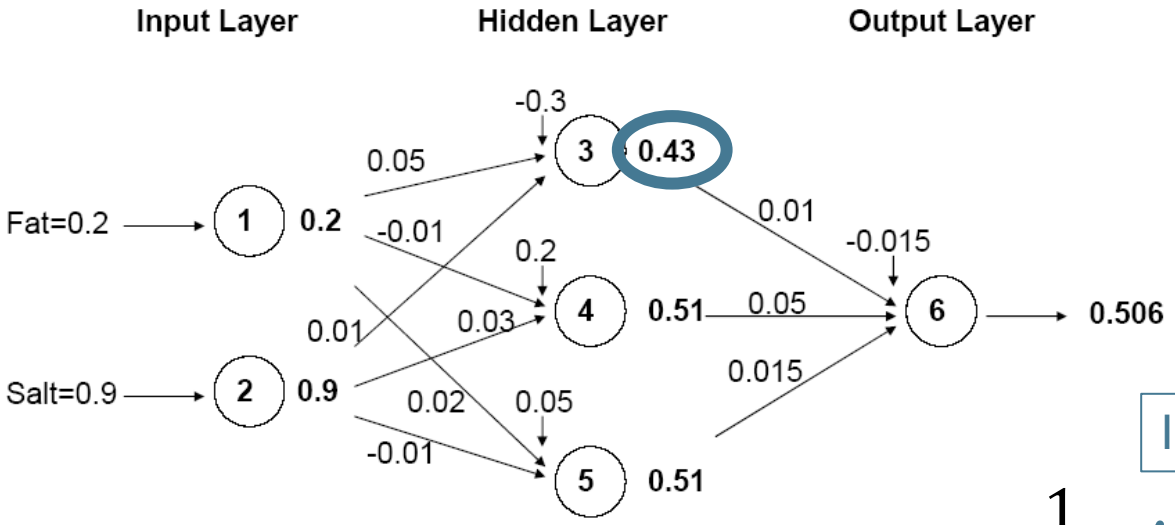


The Weights

- The Bias value (θ) and Weights (*w)* are typically initialized to random values in the range -0.05 to +0.05, which is equivalent to a model with random prediction. In other words, the model has no predictive value
- These initial weights are then used in the first round of training and are adjusted at the end based on Error Scores (i.e., Back Propagation, which we will cover in a bit)

Initial Pass of the Network: Node 3

- If g is a logistic function, the output of node 3 can be calculated as follows:



$$output_j = g\left(\theta_j + \sum_{i=1}^{p} w_{ij}x_i\right)$$

$$output_3 = \frac{1}{1 + e^{-[-0.3+(0.05)\times(0.2)+(0.01)\times(0.9)]}} = 0.43$$

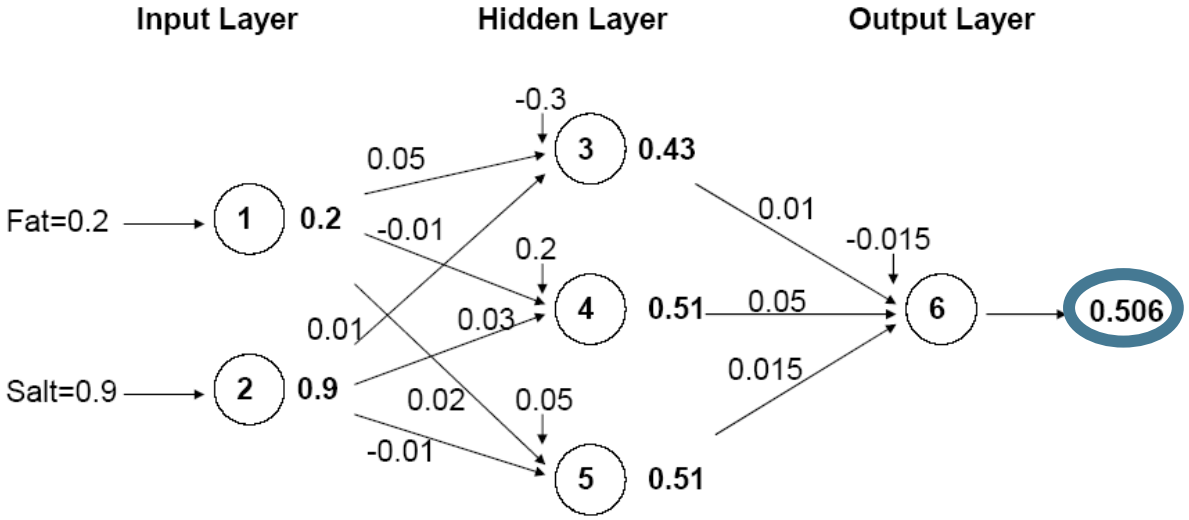
Input Values

Bias Value (random)

Initial Weights (random)

Initial Pass of the Network: Output layer

- The output of the last hidden layer becomes input for the output layer.
- Mapping the output to a classification: if cutoff for a “1” is 0.5, then we classify as “1”



$$output_j = g\left(\theta_j + \sum_{i=1}^{p} w_{ij}x_i\right)$$

$$output_6 = \frac{1}{1 + e^{-[-0.015+(0.01)(0.43)+(0.05)(0.507)+(0.015)(0.511)]}} = 0.506$$

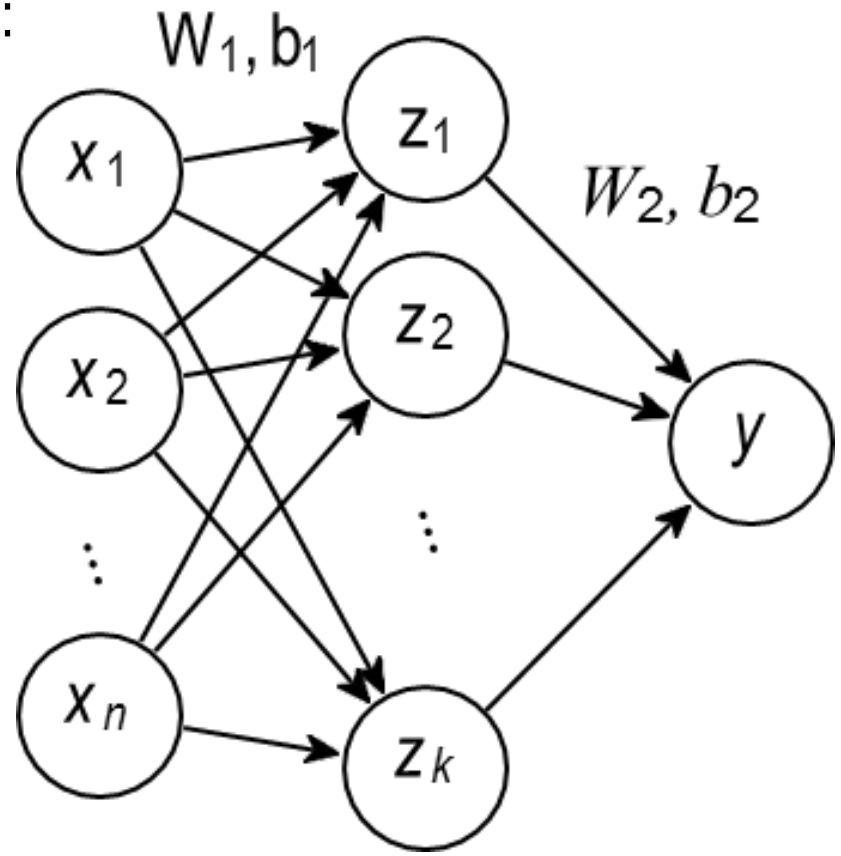
Input Values
Output from last node)

Bias Value (random)

Initial Weights
(random)

Illustrating neural networks

- We draw neural networks using the same graphic as before (the non- linear function are always in implied in the neural network setting):



- 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

Properties of neural networks

- A neural network with a single hidden layer (and enough hidden units) is a universal function approximator and 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)

Specify Network Architecture – There is a multitude of hyperparameters in Neural Networks (1/2)

Hyperparameters in Neural Networks

- **Number** of **hidden layers**:
  - Most popular – one hidden layer
- **Number** of **nodes** in hidden layer(s) :
  - More nodes capture complexity, but increase chances of overfit
- Number of **output nodes**
  - For classification, one node per class (in binary case can also use one)
  - For numerical prediction, one single node

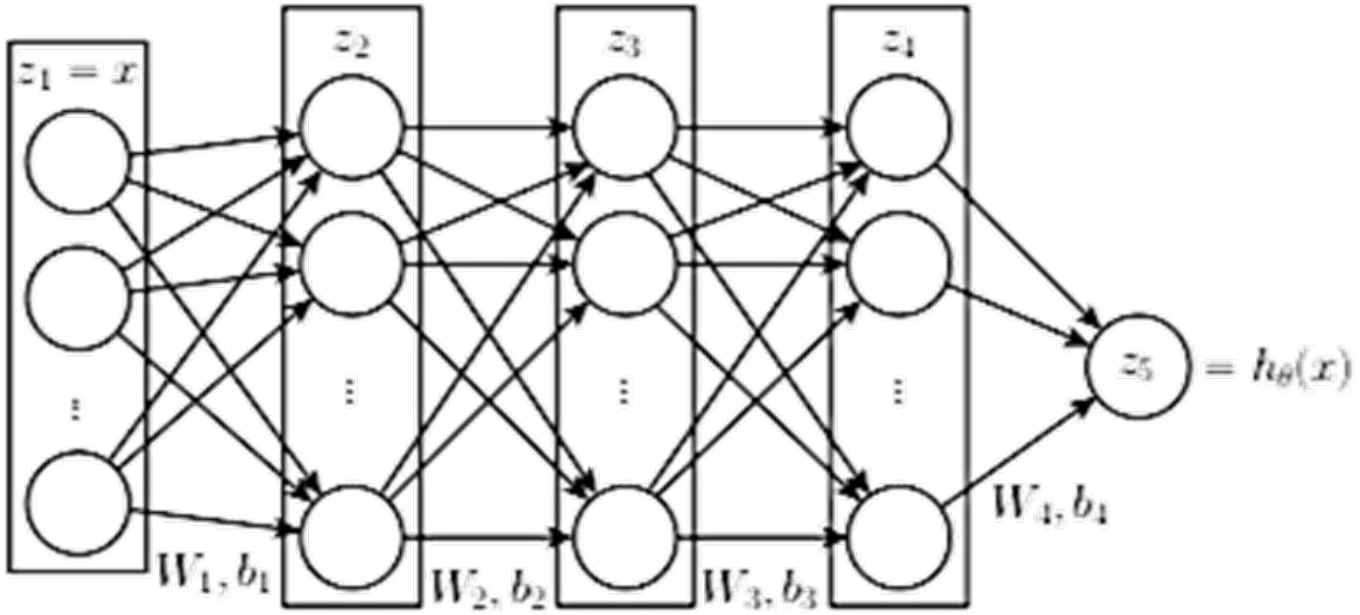
Specify Network Architecture – There is a multitude of hyperparameters in Neural Networks (1/2)

Hyperparameters in Neural Networks

- **Learning Rate** (l)
  - Low values “downweight” the new information from errors at each iteration
  - Low values slow learning, but reduce tendency to overfit to local structure
- **Momentum**
  - High values keep weights changing in same direction as previous iteration
  - Likewise, this helps avoid overfitting to local structure, but also slows learning

Deep Learning

- "Deep learning" refers (almost always) to machine learning using neural network models with multiple hidden layers (larger or equal to three):

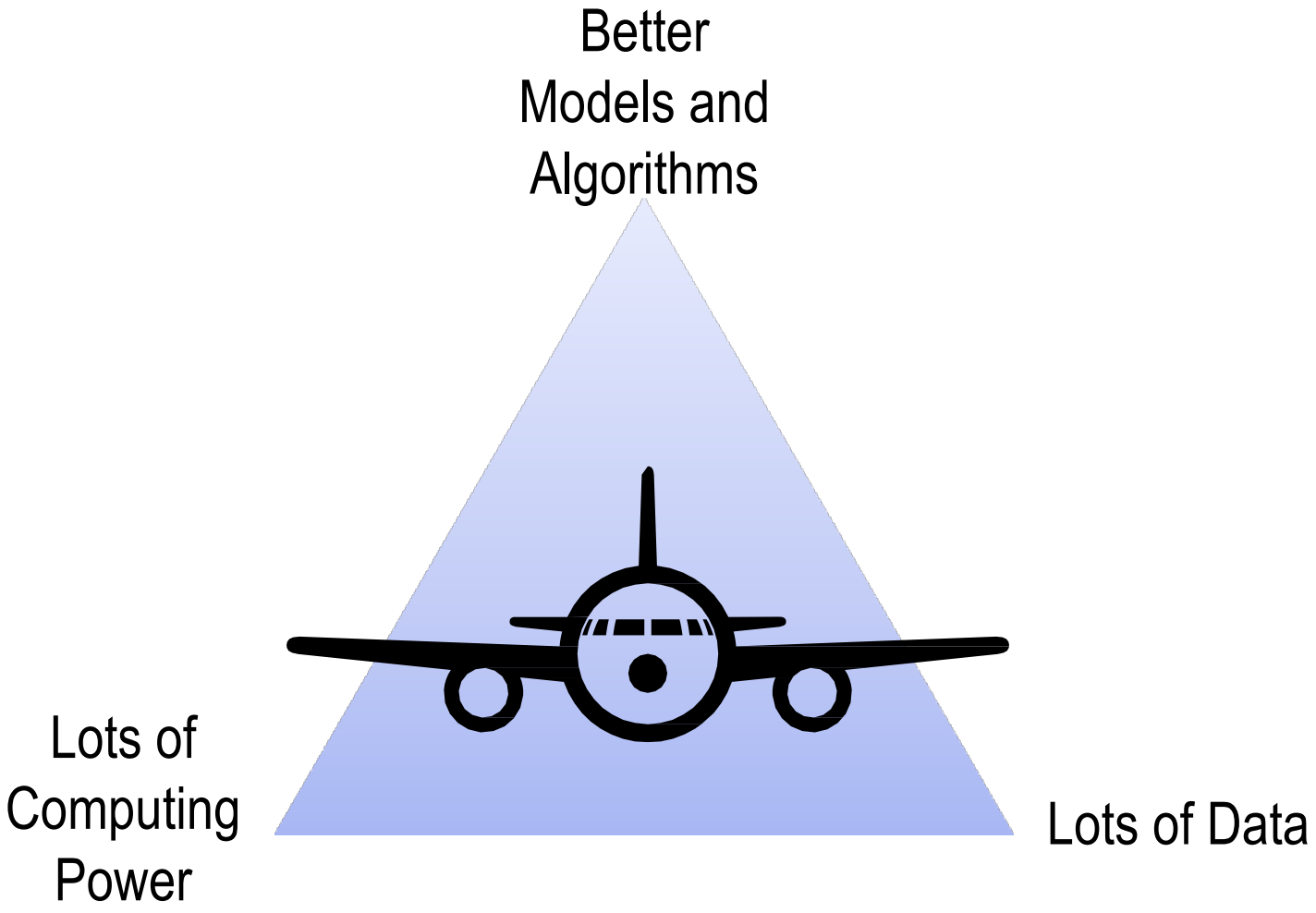


- Hypothesis function for k-layer network
  - $z_{i+1} = f_i(W_i z_i + b_i)$, $z_1 = x$ $h_\theta(x) = z_k$
  - Note: the z_i here refers to a vector, not an entry into vector

Why use deep neural networks

- **Motivation from circuit theory:** many functions can be represented more efficiently using deep networks (e.g., parity function requires $O(2^n)$ hidden units with single hidden layer, $O(n)$ with $O(\log n)$ layers
  - But not clear if deep learning really learns these types of network
- **Motivation from biology:** brain appears to use multiple levels of interconnected neurons
  - But despite the name, the connection between neural networks and biology is quite weak
- **In practice:** works much better for many domains
  - Hard to argue with good results

Why Deep Learning has taken off in recent years



Neural networks for machine learning

- **Hypothesis function:** neural network
- **Loss function:** “traditional” loss, e.g. logistic loss for binary classification:
  - $\ell(h_\theta(x), y) = \log(1 + \exp(-y.\, h_\theta(x)))$
- **Optimization:** How do we solve the optimization problem?
  - $\underset{\theta}{minimize} \sum_{i=1}^{m} \ell\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$
  - Just use gradient descent as normal (or rather, a version called stochastic gradient descent)

Optimizing NN loss functions – Stochastic gradient descent

- Key challenge: Neural networks are often trained on a very large number of samples and 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
  - $\theta \leftarrow \theta - \alpha \nabla_\theta \sum_{i=1}^{m} \ell\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) = \theta - \alpha \sum_{i=1}^{m} \nabla_\theta \ell\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$
- Alternative approach, stochastic gradient descent (SGD): adjust parameters based upon just one selected sample
  - $\theta \leftarrow \theta - \alpha \nabla_\theta \ell\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$ and then repeats 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 $i = 1, \dots, m$:
    - $\theta \leftarrow \theta - \alpha \nabla_\theta \ell\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$
  - 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 are 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 $W1$ and $W2$

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: Scikit Learn, Tensorflow, Keras, PyTorch

Poll: Benefits of deep networks

What **advantages** would you expect when applying a **deep neural network** to some machine learning problem **versus a (pure) linear classifier**? (multiple answers)

a) Less chance of overfitting

b) Can capture more complex prediction functions

c) Better training set performance

d) Better test set performance when the number of samples is small

e) Better test set performance when number of samples is large

Poll: Benefits of deep networks

What **advantages** would you expect when applying a **deep neural network** to some machine learning problem **versus a (pure) linear classifier**? (multiple answers)

a) Less chance of overfitting

b) Can capture more complex prediction functions

c) Better training set performance

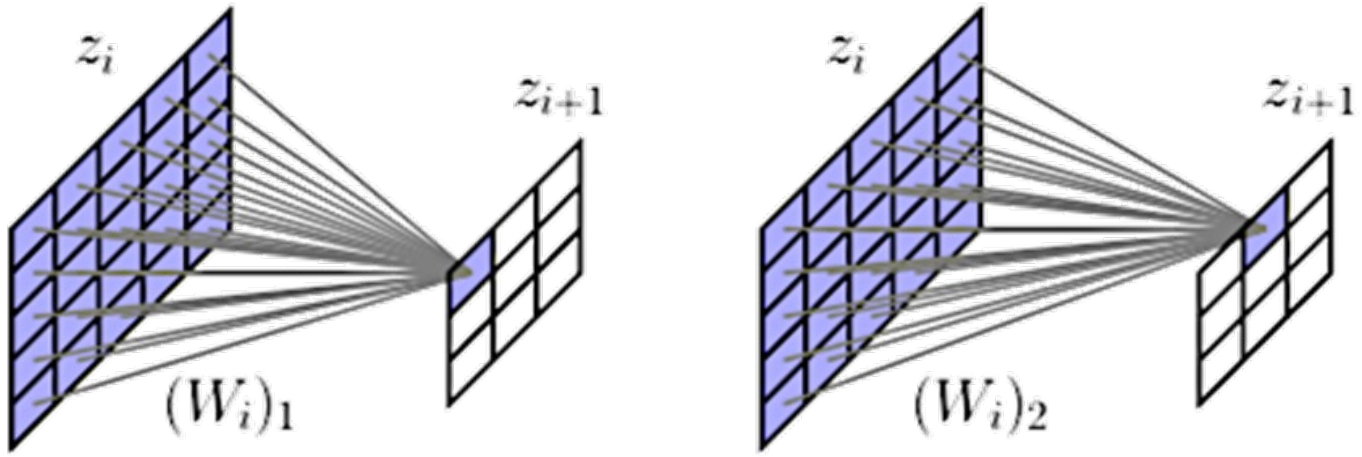
d) Better test set performance when the number of samples is small

e) Better test set performance when number of samples is large

Specialized architectures

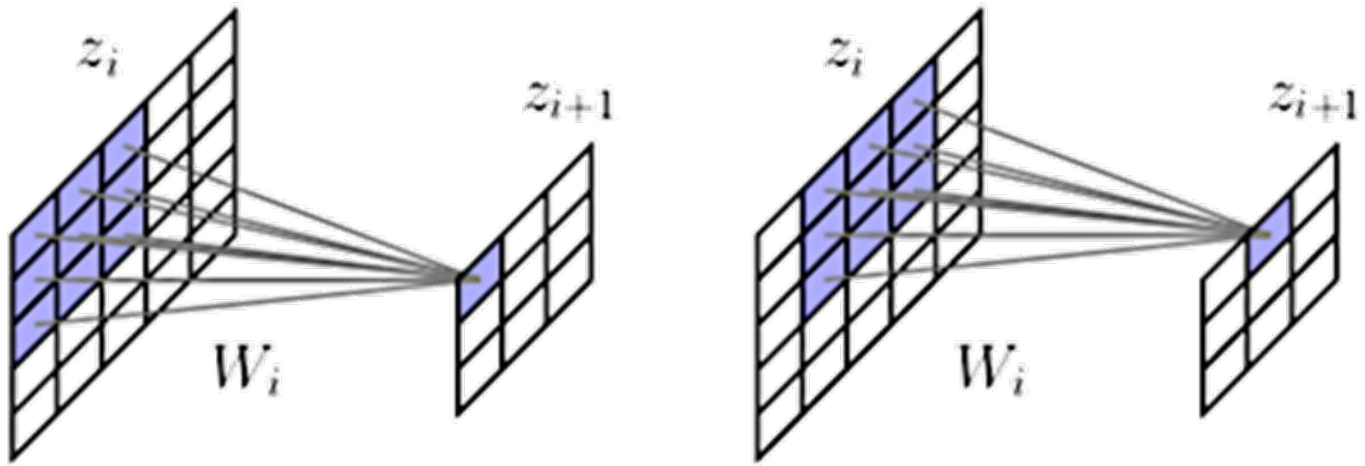
- Very little of the current wave of enthusiasm for deep learning has actually come from the simple “fully connected” neural network model we have seen so far
- Instead, most of the excitement has come from two more specialized architectures: **convolutional neural networks**, and **recurrent neural network**

The problem with fully-connected networks

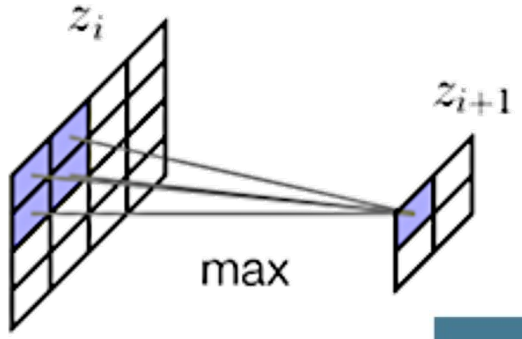


- A 256x256 (RGB) image means ~200,000 dimensional input
- Fully connected deep network would require a huge number of parameters, that are very likely to overfit to data
- A generic deep network also doesn't capture the "natural" **invariances** we expect in images (location, scale)

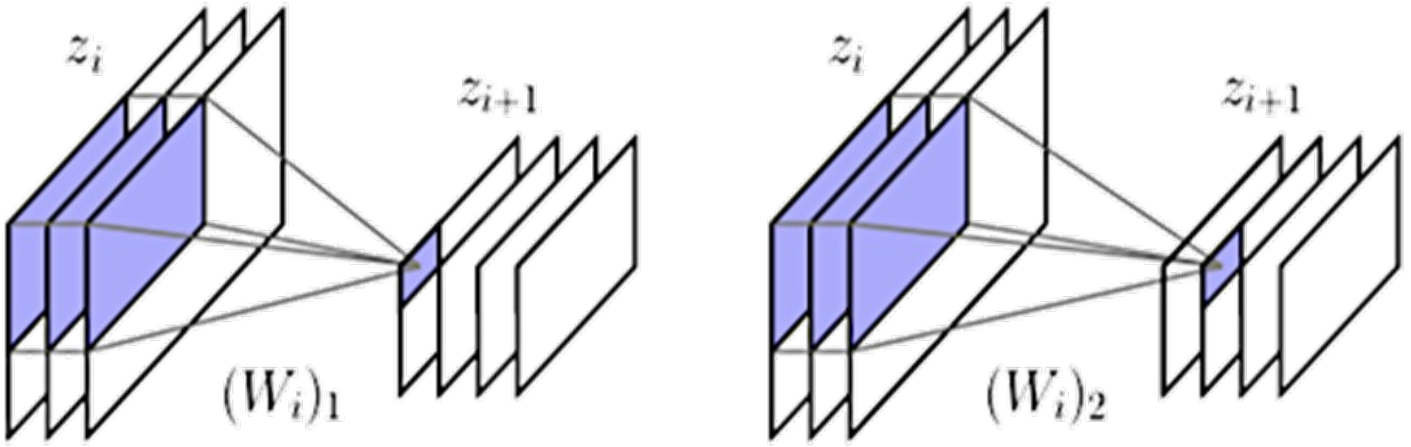
Convolutional neural networks



- 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

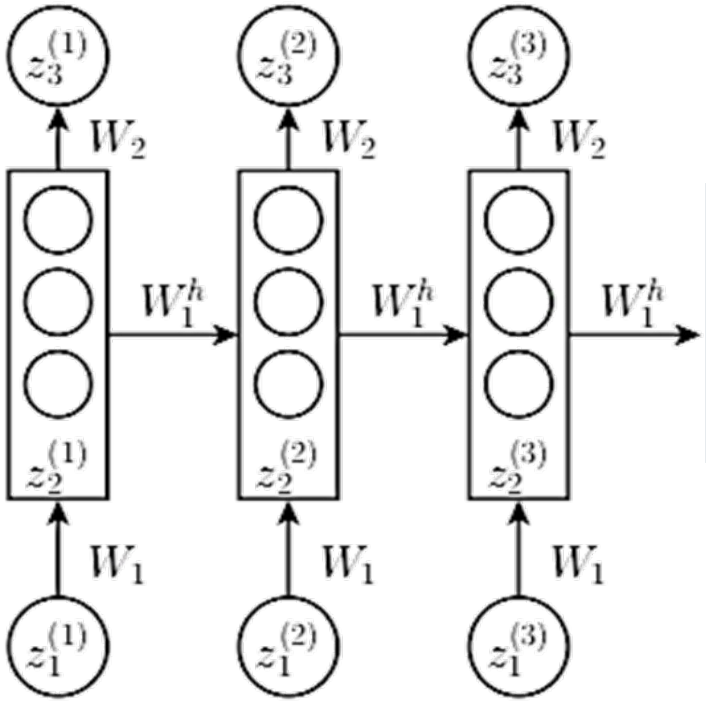


- Actually common to use “3D” convolutions to combine multiple channels, and use multiple convolutions at each layer to create different features
- Convolutions are still linear operations, and we can take gradients using backpropagation in much the same manner

Predicting sequential data (e.g., time series)

- 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



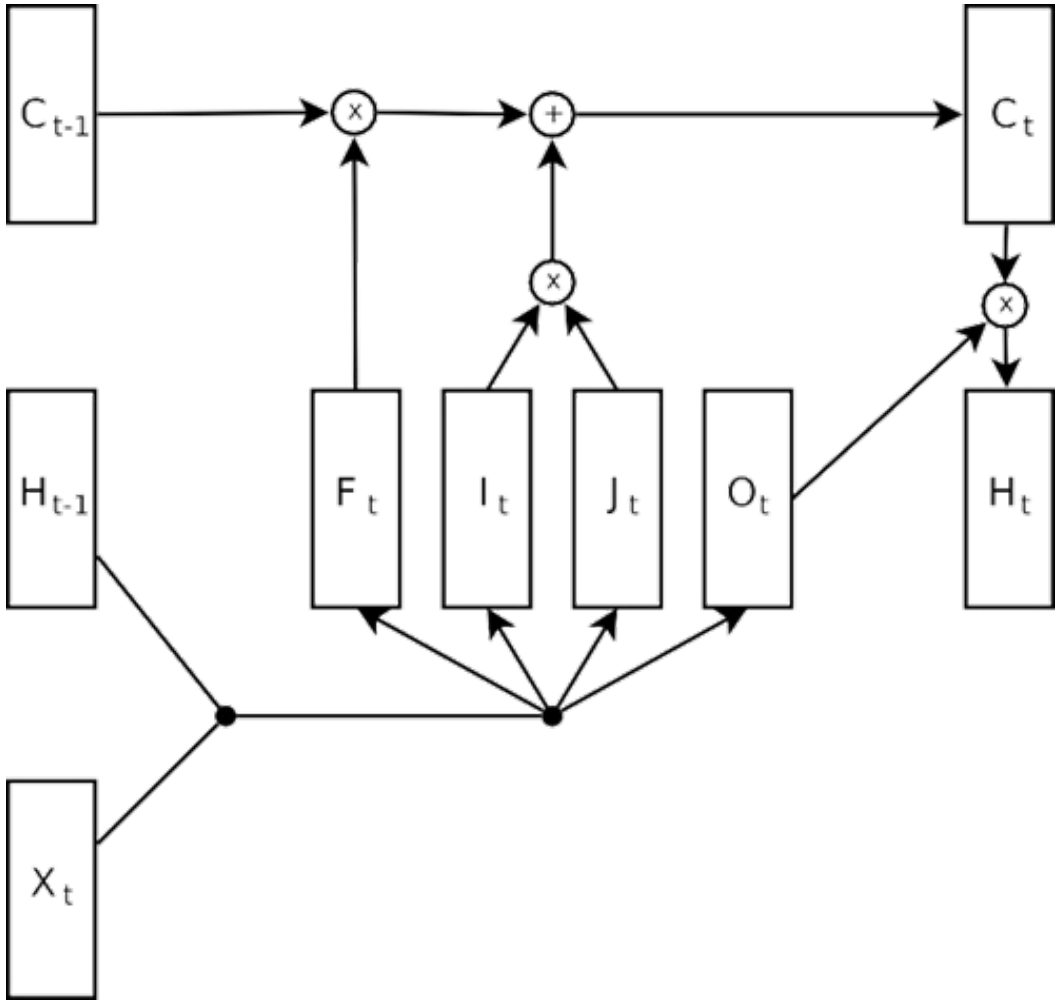
$$z_{i+1}^{(t)} = f_i\left(W_i x^{(t)} + W_i^h z^{(t-1)} + b_i\right)$$

$$\ldots$$

$$h_\theta\left(x^{(t)}\right) = z_k^{(t)}$$

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

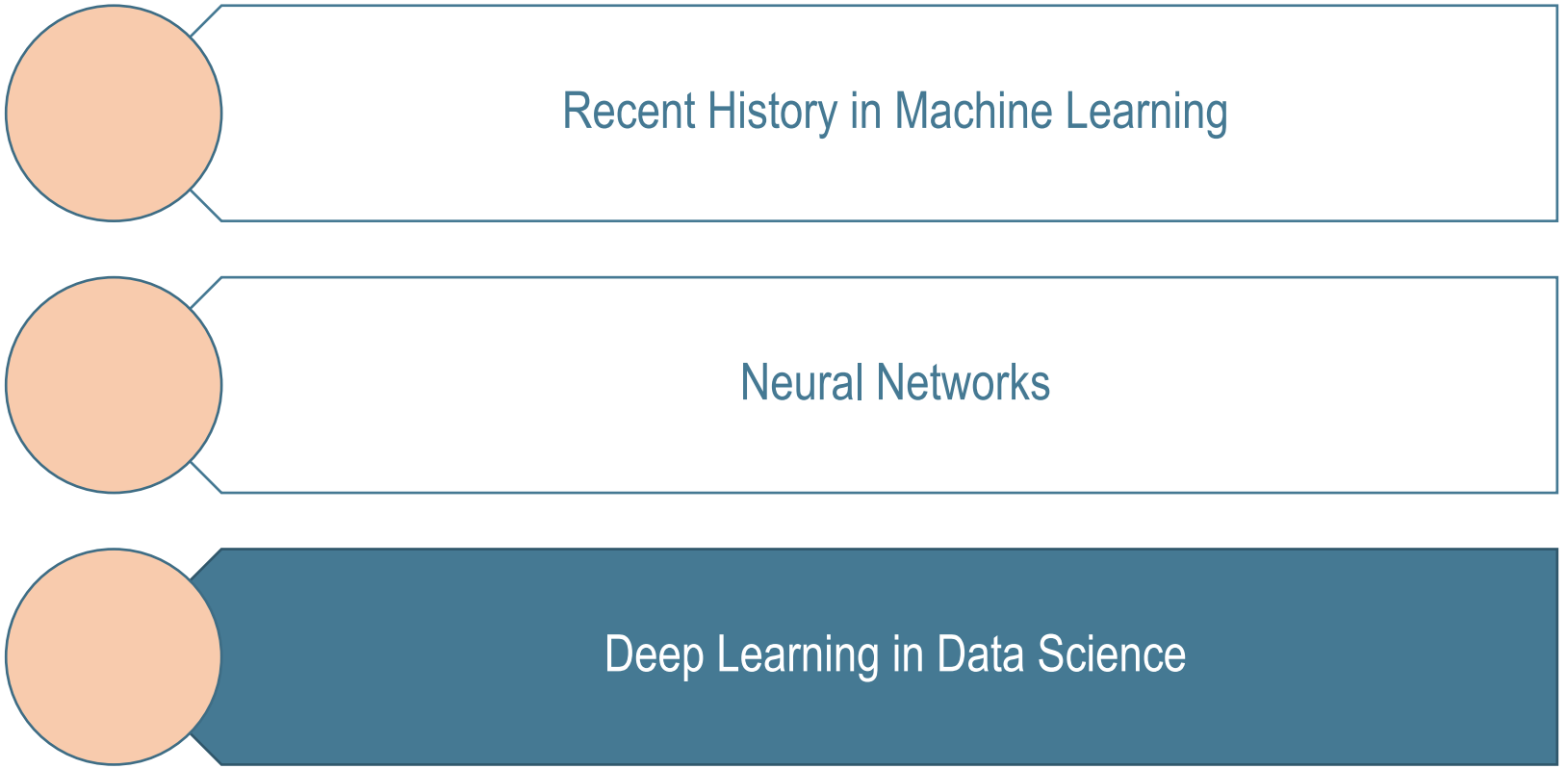
Recurrent neural networks in practice



- 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

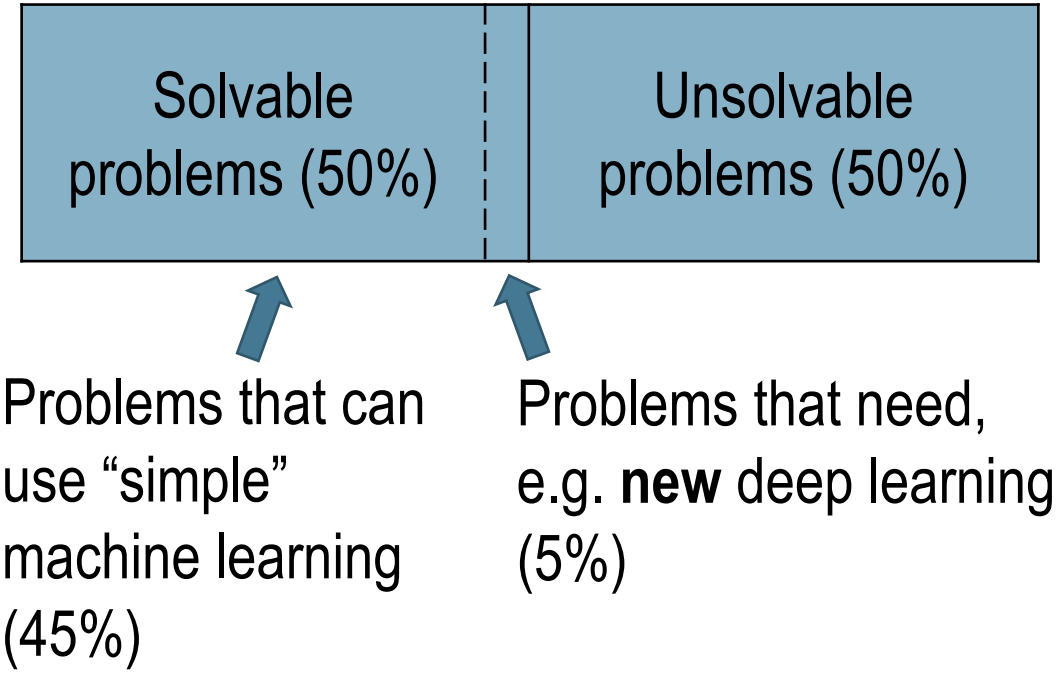
Source: Jozefowicz et al., 2015

Agenda


Recent History in Machine Learning
Neural Networks
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:



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

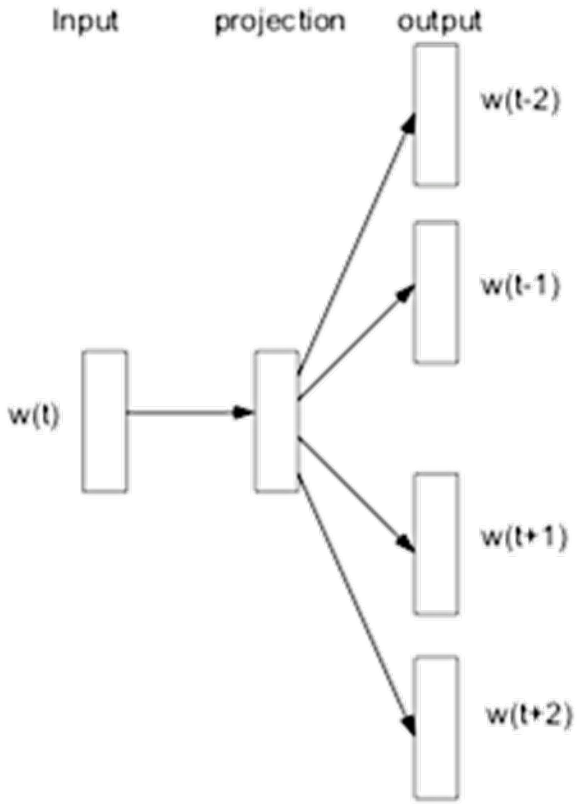
- VGG network, trained on ImageNet 1000-way classification of images
- Given a new image classification problem, take pre-trained VGG network, take the last layer of weights, and use them as features
- Can also “finetune” last few layers of a network to specialize to a new task

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					

Source: Simonyan and Zisserman, 2015

Example: text processing with word2vec

- word2vec 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



Source: Mikolov et al., 2013

Dangers of Deep Learning (because of missing theory)

Intelligent Machines

The Dark Secret at the Heart of AI

No one really knows how the most advanced algorithms do what they do. That could be a problem.

by Will Knight April 11, 2017

Last year, a strange self-driving car was released onto the quiet roads of Monmouth County, New Jersey. The experimental vehicle, developed by researchers at the chip maker Nvidia, didn't look different from other autonomous cars, but it was unlike anything demonstrated by Google, Tesla, or General Motors, and it showed the rising power of artificial intelligence. The car didn't follow a single instruction provided by an engineer or programmer. Instead, it relied entirely on an algorithm that had taught itself to drive by watching a human do it.

Summary: Advantages and Disadvantages of NN

Advantages

- Good predictive ability
- Can capture complex relationships
- No need to specify a model

Disadvantages

- Considered a “black box” prediction machine, with no insight into relationships between predictors and outcome
- No variable-selection mechanism, so you have to exercise care in selecting variables
- Heavy computational requirements if there are many variables (additional variables dramatically increase the number of weights to calculate)

Contact

For general questions and enquiries on **research**, **teaching**, **job openings** and new **projects** refer to our website at www.is3.uni-koeln.de

For specific enquiries regarding this course contact us by sending an email to the **IS3 teaching** address at is3-teaching@wiso.uni-koeln.de