Neural Networks Workshop

21st January 2017, NYC WiMLDS Meetup

Laura Graesser

Objectives for today

- Understand what neural networks are, what they are used for, and why they are powerful
- Understand their structure and why it matters
- Develop a solid foundation for learning more about deep learning
- Learn how to build and train your own neural networks in Keras
- Have fun

Part 1

Agenda

- **10:00 - 11:15:** Theory: Deep feed-forward neural networks
- **11:15 - 11:30:** Break
- **11:30 - 12:45:** Practice: Classifying hand written digits with Keras
- **12:45 - 13:00:** Review and wrap up

What is a neural network?

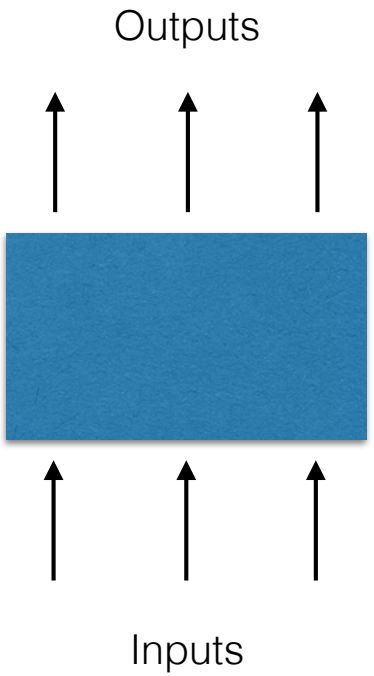
A family of algorithms

that excel at making

predictions about

unseen data

A function



Strengths

- Flexible and general
- Powerful - universal function approximators
- Good at generalizing
- Learn their own features

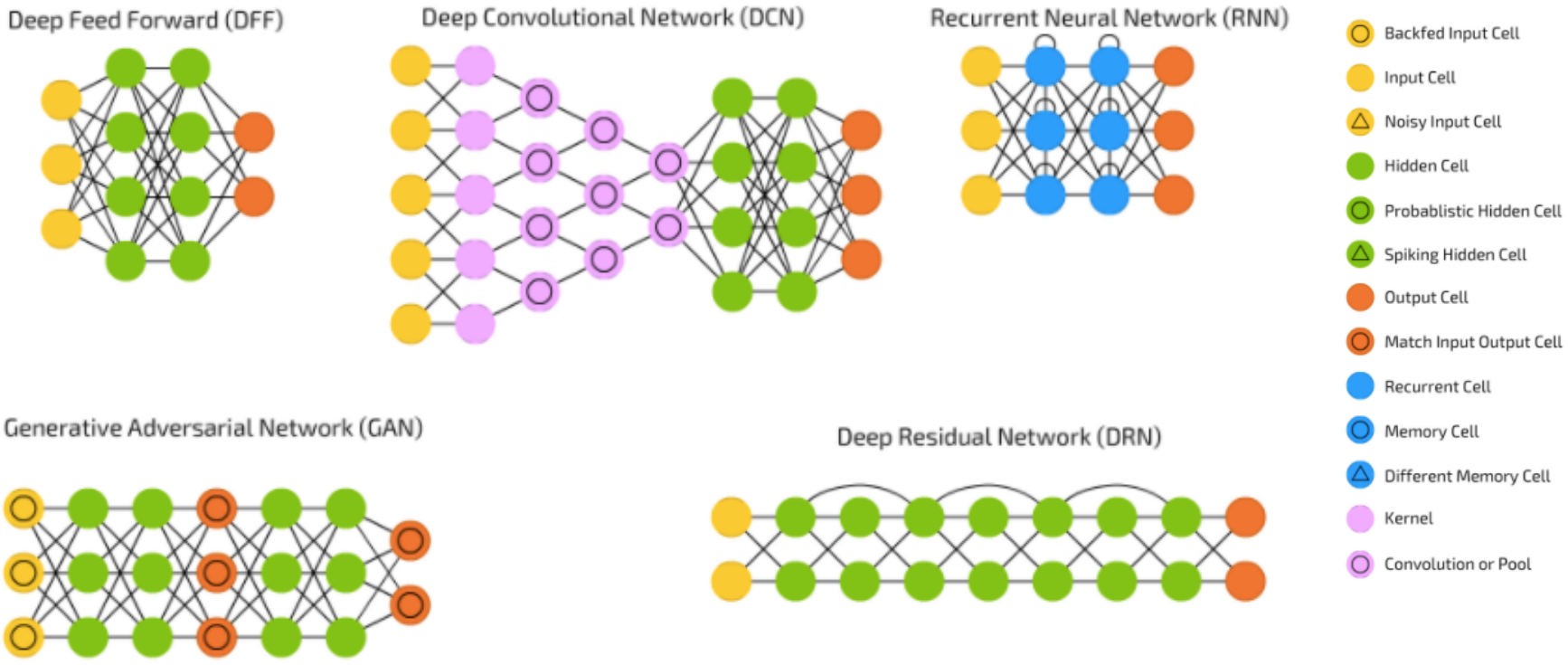
A brief history of neural networks

- 1957: The perceptron algorithm invented by Frank Rosenblatt
- 1986: Backpropagation invented by Rumelhart, Hinton, and Williams
- 1989: Yann LeCun successfully trains a neural network to recognize hand written digits
- 2012: AlexNet (convolutional network) wins ImageNet smashing existing benchmarks, Google trains a neural network to recognize cats without ever being given a labelled example
- 2013: Algorithm for efficient estimation of word vectors invented by Mikolov, Chen, Corrado and Dean; significant impact on NLP
- 2016: Google Translate switches to neural network model and improves significantly, speech recognition approaches human level performance, AlphaGo, driverless cars, …

Applications

- Computer vision: object recognition, image captioning, image generation, self driving cars, sorting cucumbers
- Audio: speech transcription, music transcription, music composition
- Natural language processing: machine translation, question answering
- Computer games: Go, Atari
- Approximating NP-hard problems: traveling salesman

Number of architectures is increasing



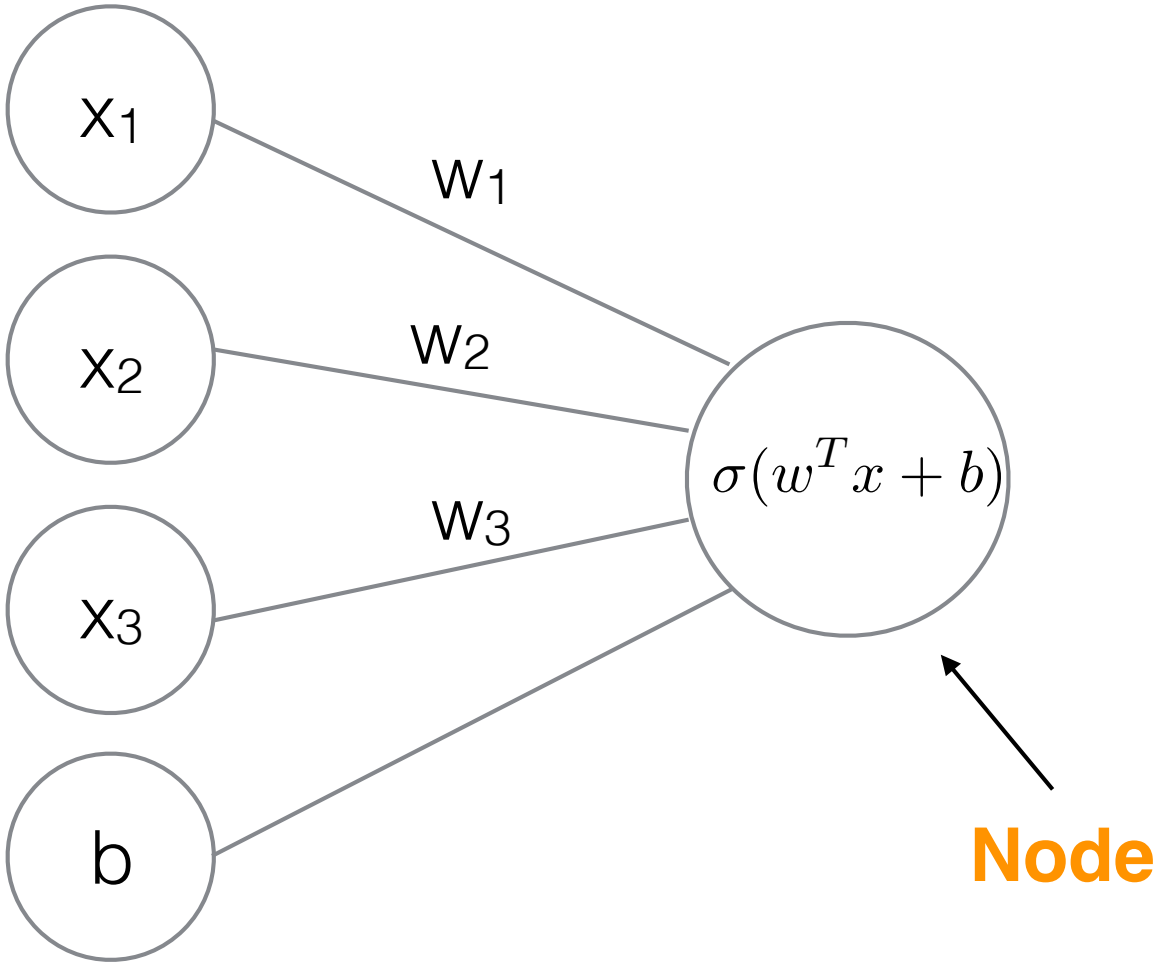
We are focusing on deep feed-forward neural networks

Source: The Neural Network Zoo, The Asimov Institute

Components of a machine learning model

- Model
- Optimizer
- Loss function
- Data

Nodes are the fundamental building blocks of neural networks



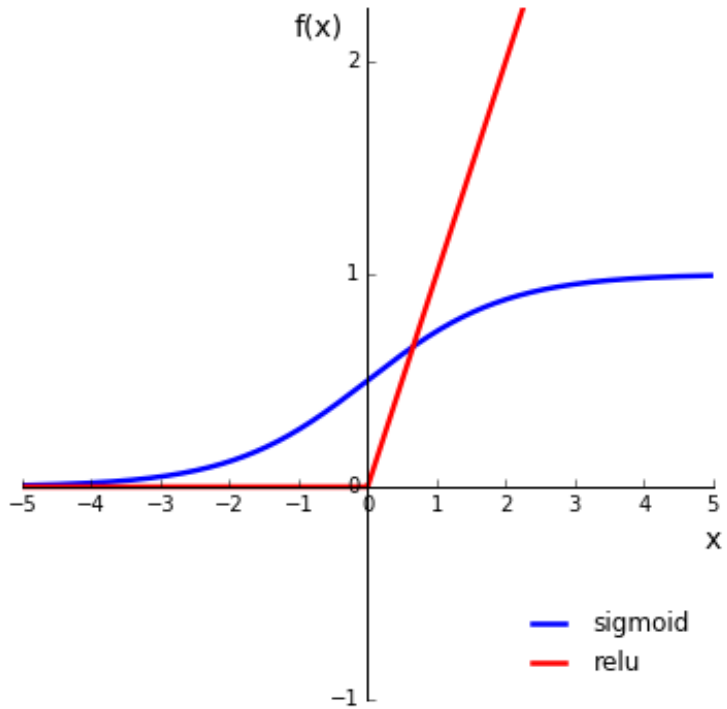
Activation functions: sigmoid and rectified linear unit (ReLU)

Sigmoid

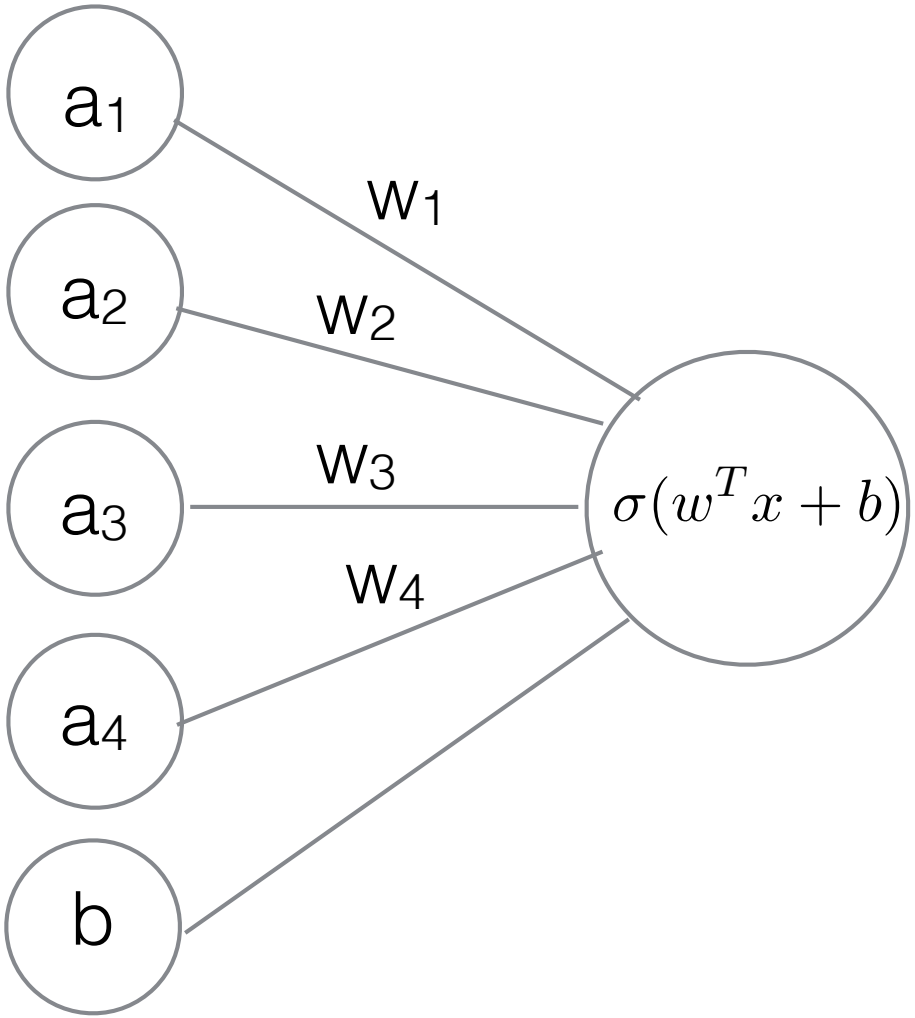
$$\sigma(z) = \frac{1}{1+e^{-z}}$$

ReLU

$$ReLU(x) = max(0, x)$$



Example



$$a = \begin{pmatrix} 0.8 \\ 0.5 \\ 0.2 \\ 0.5 \end{pmatrix}, \quad w = \begin{pmatrix} 5 \\ -3 \\ 4 \\ 0.75 \end{pmatrix} \quad b = 2$$

$$\begin{aligned} y &= \sigma(w^T a + b) \\ a^T w &= 3.675 \\ y &= \sigma(5.675) \\ &= \frac{1}{1 + e^{-5.675}} \\ &\approx 0.997 \end{aligned}$$

Structure of a deep feed-forward neural network

Output layer

Hidden layers

- 1 - n layers, k_1,.., k_n nodes per layer
- Non linear activation function

Input layer

Source: The Neural Network Zoo, The Asimov Institute

The feed-forward step

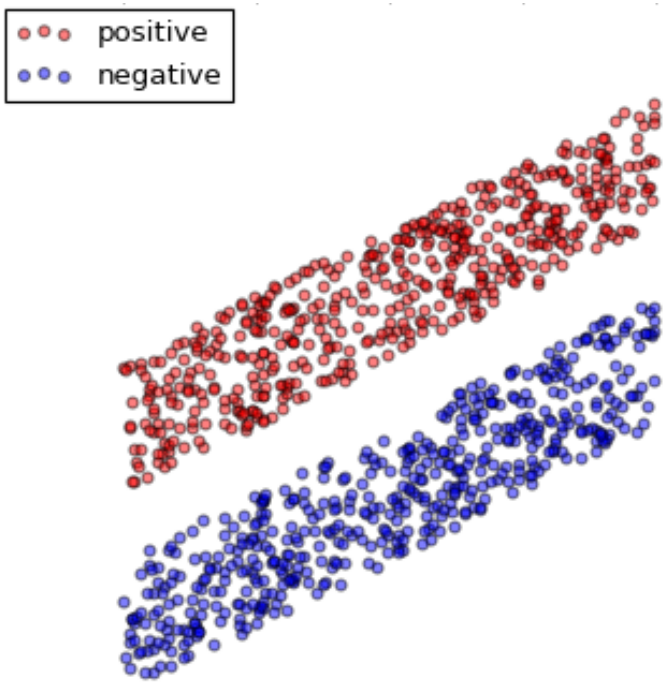
Data flows forward through a network from input to output

Source: The Neural Network Zoo, The Asimov Institute

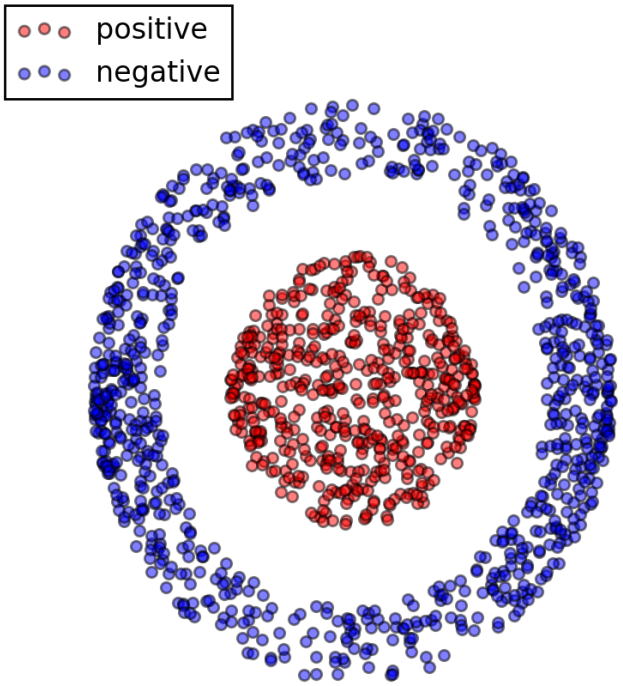
Why is this structure powerful?

Motivating example: two class classification problem

A. Data linearly separable



B. Data not linearly separable



How can we make B like A?

How to find the right transformation?

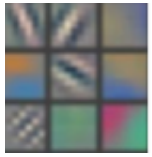
- Previous approaches
    - Specify explicitly by manually transforming the input features
    - Pick an extremely general transformation
- Neural networks *learn* the correct transformation

Non-linear activation function + depth = power

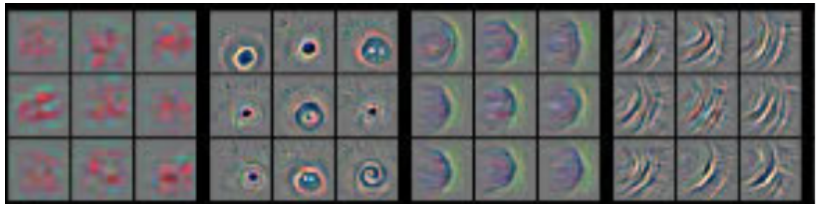
- Non linear activation function + hidden layers.
    - Each layers creates new features. These are non-linear transformations of their inputs.
    - Hierarchy of features, progressively more abstract layer by layer.
- Output layer: computes linear separating boundary (classification) from final hidden layer features.

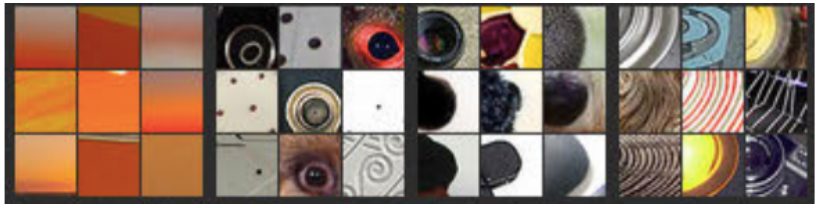
The features can be visualized

Layer 1

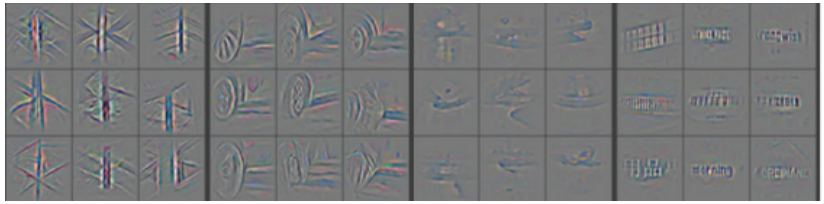


Layer 2

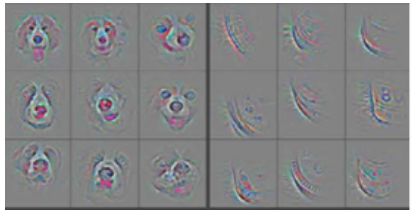


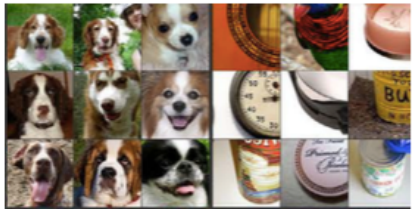


Layer 3

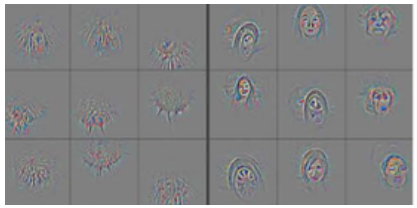


Layer 4





Layer 5



Source: Visualizing and Understanding Convolutional Networks, Zeiler & Fergus, 2014

How a network learns

- Learning is the process of adjusting weights and biases so that the network produces the correct outputs
- Start with randomly initialized values
- Show model many examples
- Propagate error between model's output and the correct value backwards through the model
- Need a measure of error

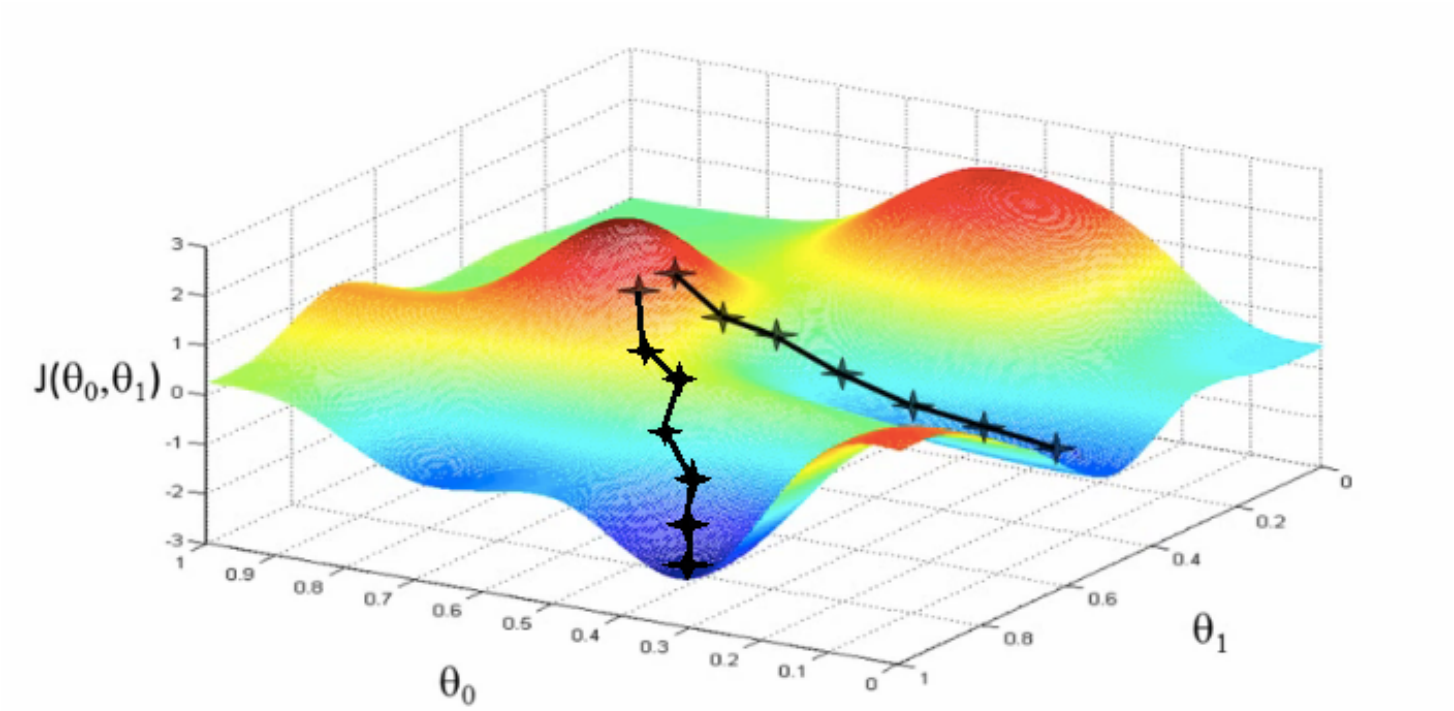
Loss functions: different ways of measuring error

	Quadratic loss (mean squared error)	Categorical cross-entropy
f(y, a)	$\frac{1}{2} \parallel y - a \parallel_2^2$	$-\left(y^T \ln(a) + (1-y)^T \ln(1-a)\right)$
f'(y, a)	$(a - y)$	$-\left(\frac{y}{a} - \frac{(1-y)}{(1-a)}\right)$

Loss functions: examples

1. a = 0.99
   - Quadratic loss = $(1 - 0.99)^2 = 0.0001$
   - Cross Entropy loss = $-(1 * \ln(0.99) + 0 * \ln(0.01)) \approx 0.01$
2. a = 0.5
   - Quadratic loss = $(1 - 0.5)^2 = 0.25$
   - Cross Entropy loss = $-(1 * \ln(0.5) + 0 * \ln(0.5)) \approx 0.69$
3. a = 0.01
   - Quadratic loss = $(1 - 0.01)^2 = 0.9801$
   - Cross Entropy loss = $-(1 * \ln(0.01) + 0 * \ln(0.99)) \approx 4.61$

Gradient descent



Learning rate controls the size of the steps

Source: Tuning the learning rate in gradient descent, Vasilis Vryniotis, Datumbox blog

Stochastic gradient descent

- Weights and biases are updated after processing a random subset (batches) of the data instead of the entire dataset
- Batches typically contain 16 - 256 examples
- Estimation of the cost function
- More efficient, faster convergence

Recap

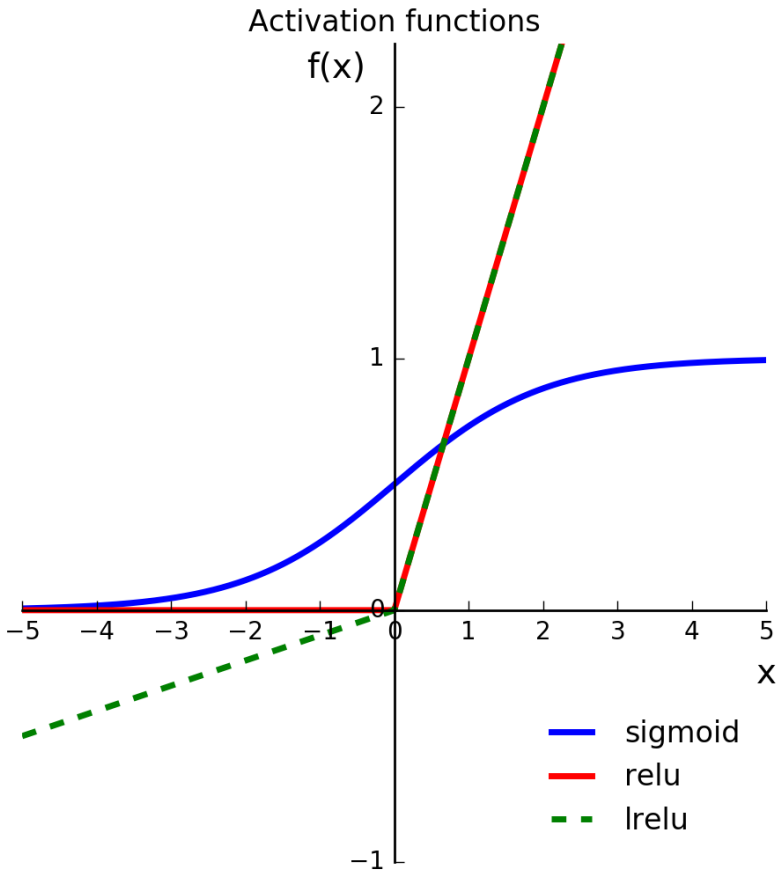
- Model: deep feed-forward network
- Optimizer: stochastic gradient descent
- Loss function: quadratic loss, categorical cross-entropy
- Data: MNIST

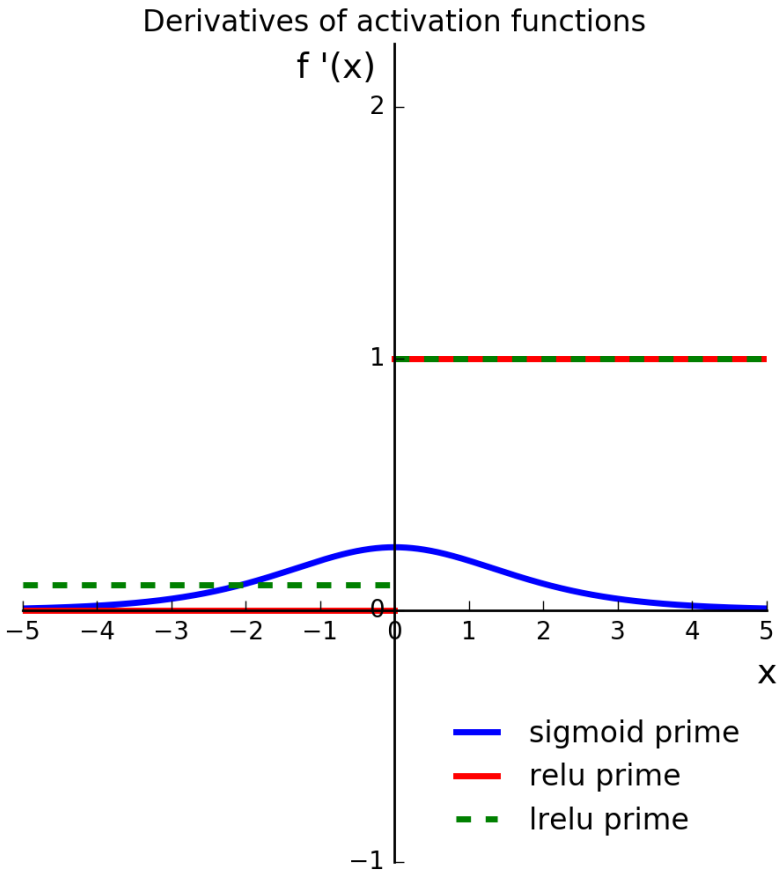
Classifying handwritten digits with Keras

Questions

- What happens when you increase / decrease the batch size?
- What happens when you increase the size of a layer?
- What happens when you increase the number of layers?
- What happens when you change the loss function?
- What happens when you change the activation function?
- What happens to the difference between the training and validation accuracy?

Training issues: nodes can saturate or die





Part 1 Appendix: Categorical cross-entropy vs. Quadratic loss

Terminology

- C: any loss function
  - CQ: quadratic loss
  - CCE: categorical cross-entropy
- a: vector of outputs for all nodes in the output layer
- z: vector of values before the activation function is applied in the output layer
- delta: error of a node (defined to the right)
- f(x): activation function at output layer
- f'(x): derivative of the activation function

$$\nabla_a C = \left[\frac{\partial C}{\partial a_1}, \frac{\partial C}{\partial a_2}, ..., \frac{\partial C}{\partial a_n}\right]$$

$$\nabla_a C^Q = (a - y)$$

$$\nabla_a C^{CE} = -\left(\frac{y}{a} - \frac{(1-y)}{(1-a)}\right)$$

$$\begin{aligned}\delta &= \nabla_a C \odot f\,'(z) \\ &= \left[\frac{\partial C}{\partial a_1} f\,'(z_1),\ \frac{\partial C}{\partial a_2} f\,'(z_2),\ ....,\ \frac{\partial C}{\partial a_n} f\,'(z_n)\right] \\ &= \left[\frac{\partial C}{\partial a_1}\frac{\partial a_1}{\partial z_1},\ \frac{\partial C}{\partial a_2}\frac{\partial a_2}{\partial z_2},\ ....,\ \frac{\partial C}{\partial a_n}\frac{\partial a_n}{\partial z_n}\right]\end{aligned}$$

Categorical cross-entropy "undoes" sigmoid gradient in output layer

$$\delta = \nabla_a C^{CE} \sigma\,'(x) = -\left(\frac{y}{a} - \frac{(1-y)}{(1-a)}\right)\sigma'(z)$$

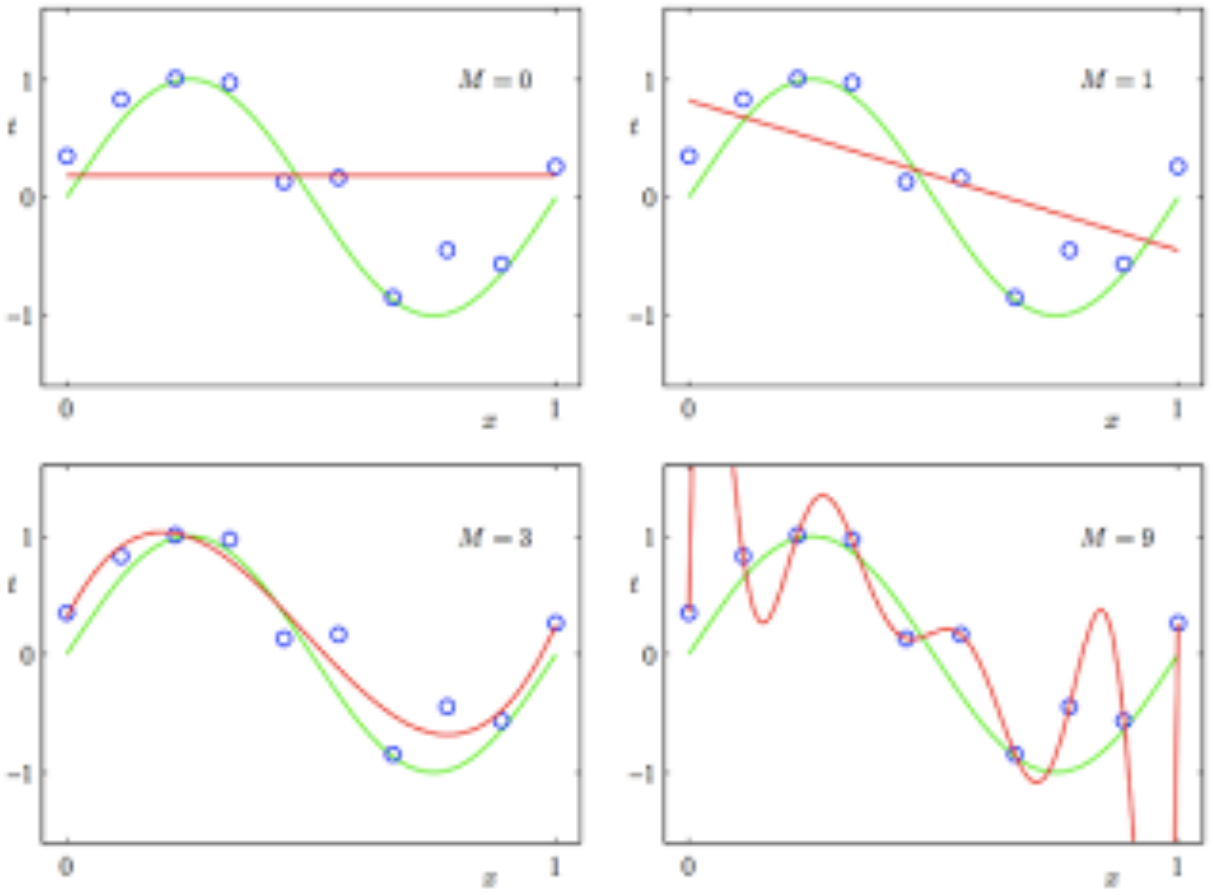
$$\begin{aligned}
\delta &= -\left(\frac{y}{\sigma(z)} - \frac{(1-y)}{(1-\sigma(z))}\right)\sigma\,'(z) \\
&= -\left(\frac{y(1-\sigma(z)-(1-y)\sigma(z)}{\sigma(z)(1-\sigma(z))}\right)\sigma\,'(z) \\
&= -\left(\frac{y-\sigma(z)y-\sigma(z)+y\sigma(z)}{\sigma(z)(1-\sigma(z))}\right)\sigma\,'(z) \\
&= -\left(\frac{y-\sigma(z)}{\sigma(z)(1-\sigma(z))}\right)\sigma\,'(z) \\
&= \left(\frac{\sigma(z)-y}{\sigma(z)(1-\sigma(z))}\right)\sigma\,'(z) \\
&= \sigma(z) - y \quad , \; since\; \sigma\,'(z) = \sigma(z)(1-\sigma(z))
\end{aligned}$$

Part 2

Agenda

- **14:00 - 14:40:** Theory: Regularization
- **14:40 - 15:10:** Practice: Classifying airplanes and cats
- **15:10 - 15:20:** Break
- **15:20 - 15:50:** Theory: Convolutional Networks
- **15:50 - 16:20:** Practice: Building convolutional networks
- **16:20 - 16:30:** Break
- **16:30 - 17:00:** Theory: Improving the way networks learn

Bias vs. variance



High bias
"Underfitting"

High variance
"Overfitting"

Source: Pattern Recognition and Machine Learning, Figure 1.4, Christopher Bishop, via KDNuggets

Neural networks are susceptible to overfitting

- Very expressive models
    - Can represent up to $O(2^N)$ input regions using only O(N) parameters
- Tendency to make marginal weight updates
    - Particularly for 0 - 1 classification with sigmoid or softmax outputs
    - Leads to large weights. This makes the output of a network much more sensitive to small changes in the inputs

Regularization

Regularization is an umbrella term given to any technique that helps to prevent a neural network from overfitting the training data

Regularizing neural networks can make it more likely that the network approximates the "true" function, i.e. the one which maps the underlying distribution of data from inputs to outputs, not just the training data

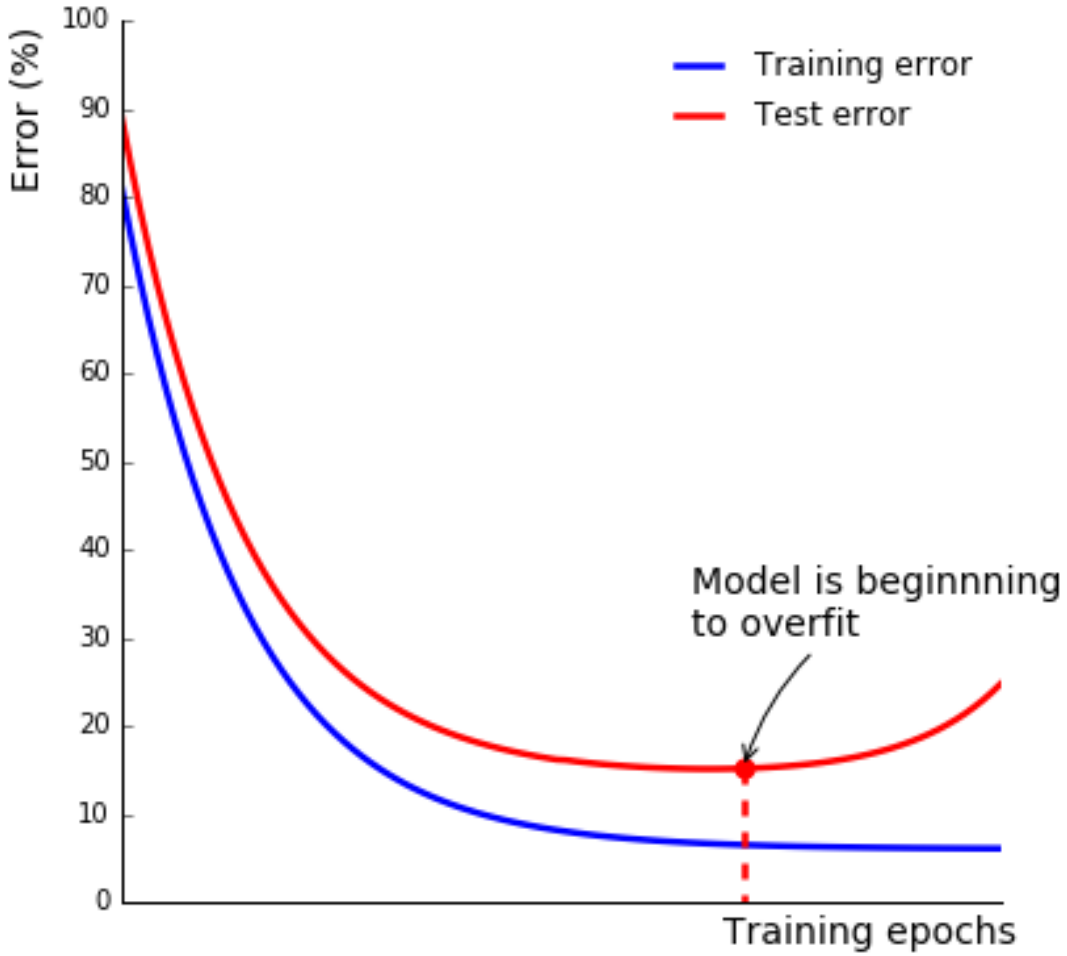
Regularization assumptions

- Smoothness
    - Smaller weights are better than large weights
    - ***or*** fewer features are better than more features
- Features that are good in many situations are better than features are good in few situations

Regularization techniques

- Smoothness
    - **L2 weight regularization:** Smaller weights are better than large weights
    - **L1 weight regularization:** Fewer features are better than more features
    - **Early stopping:** Assumes that models tend to learn more significant features first
- **Dropout:** Features that are good in many situations are better than features are good in few situations

Early stopping



Weight regularization

L1

$$C = \frac{1}{n}\left(\sum_{i=1}^{n} C_i + \lambda \sum_{w \in W} |w|\right)$$

$$\frac{\partial C}{\partial w} = \lambda \; sign(w)$$

Regularization component of the derivative only

L2

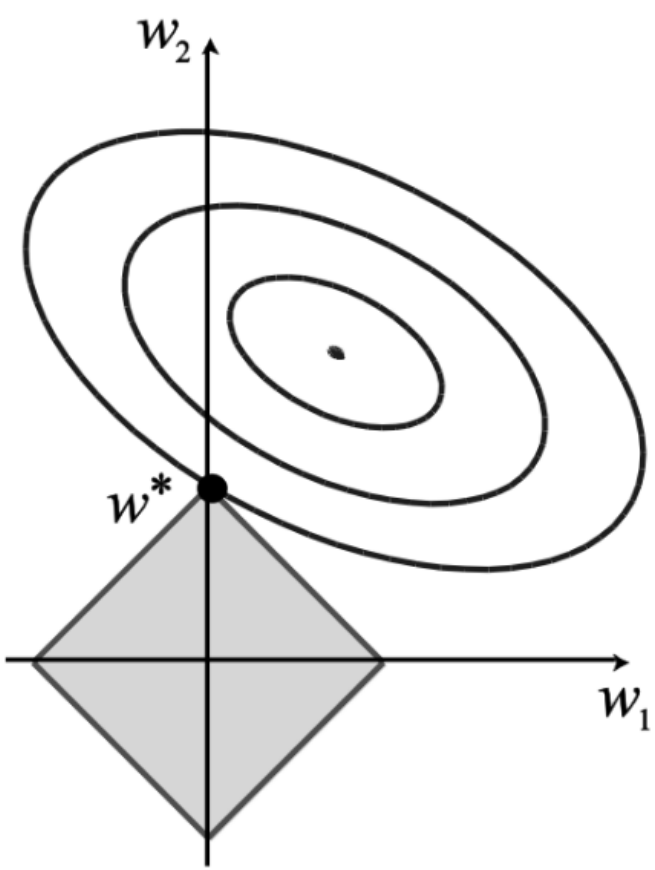
$$C = \frac{1}{n}\left(\sum_{i=1}^{n} C_i + \frac{\lambda}{2} \sum_{w \in W} w^T w\right)$$

$$\frac{\partial C}{\partial w} = \lambda w$$

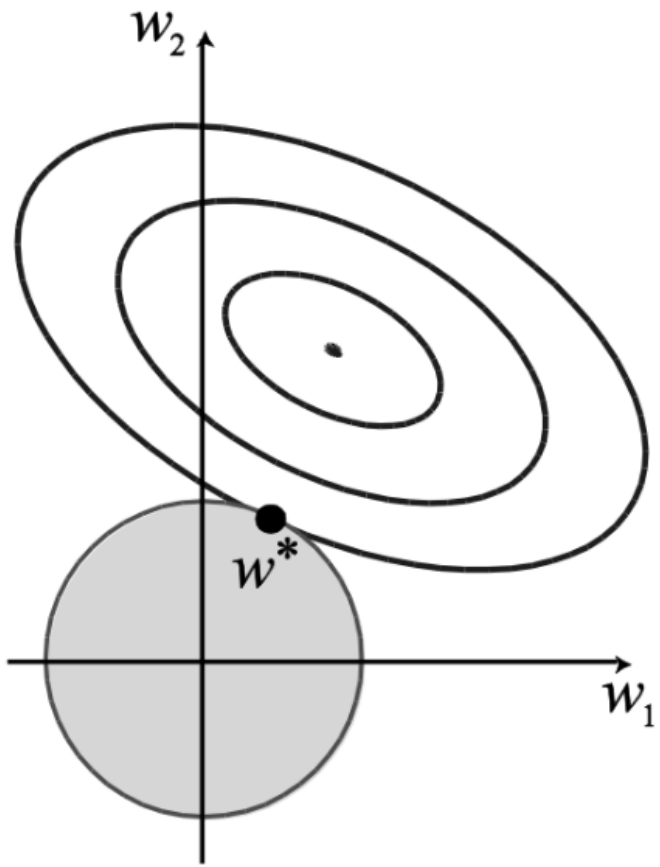
Regularization component of the derivative only

Weight regularization

L1



L2

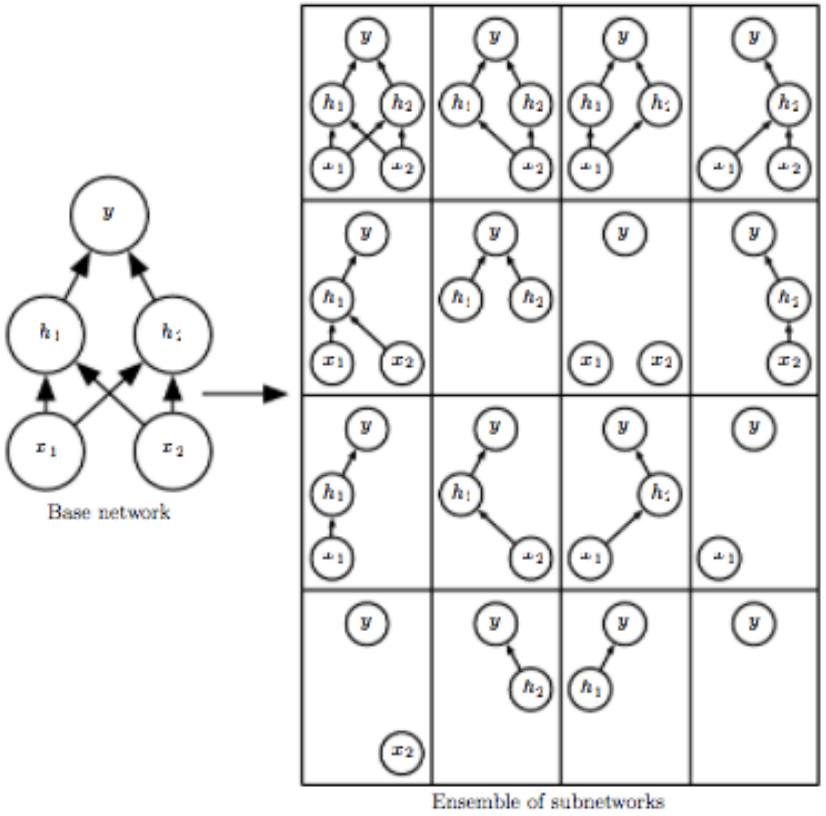


Source: Logistic Regression, Vibhav Gogate, The University of Texas at Dallas

Dropout

- Ensemble method: a collection of independent models vote on the correct outcome
    - Independent models are likely to make different errors
- Efficient technique for building a very large ensemble of independent neural networks

Dropout



- Each time a batch is processed, each node is turned off with probability *p*
- This randomly generates a subnetwork
- Many many subnetworks are trained with different data
- Information is shared across subnetworks
- Once trained, networks vote on the correct outcome using the weight scaling inference rule (Hinton)

Source: Deep Learning Book, Ian Goodfellow and Yoshua Bengio and Aaron Courville

Dropout - intuition

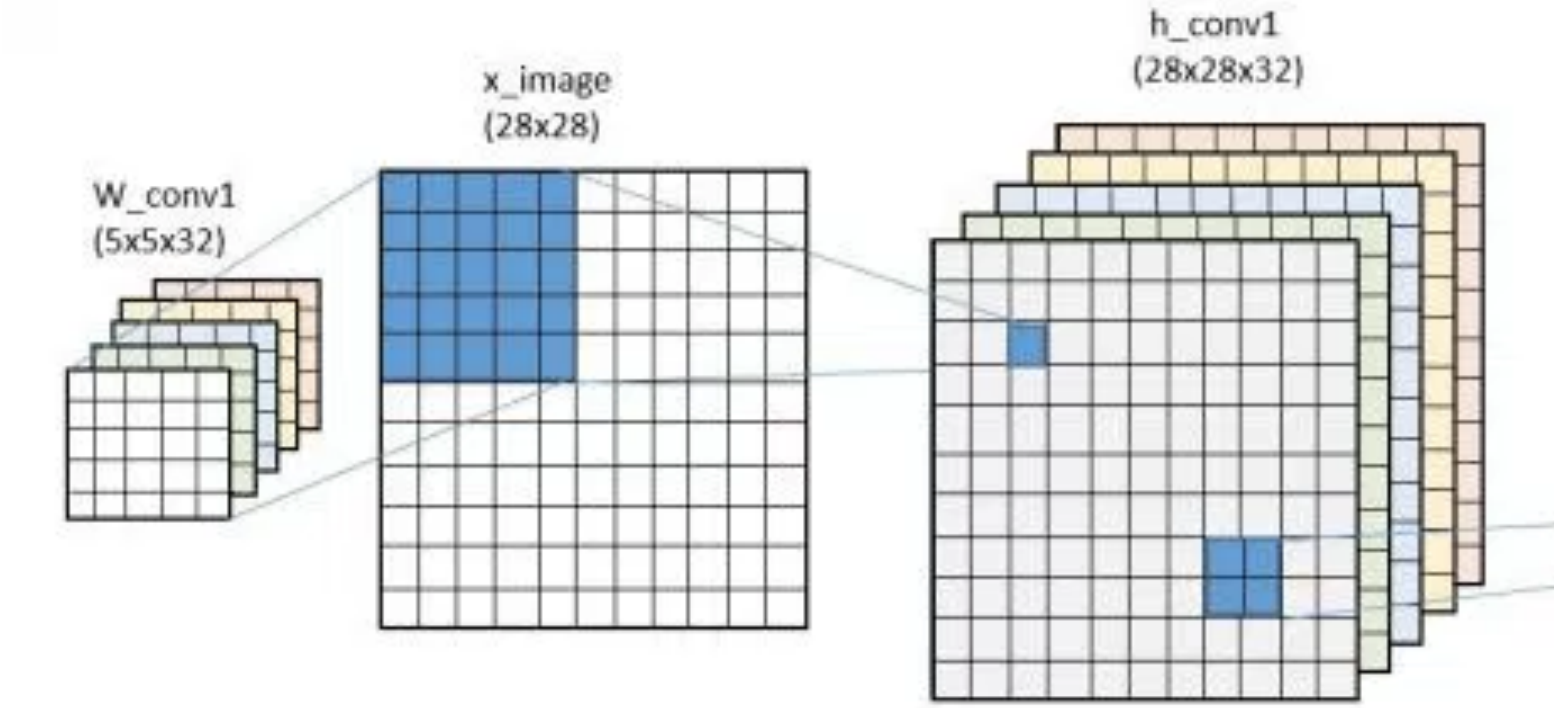
- Good features are those that work well in different contexts.
- No node can rely on any one of its inputs to be present
- This builds robustness in feature detectors

Classifying airplanes and cats

Convolutional networks

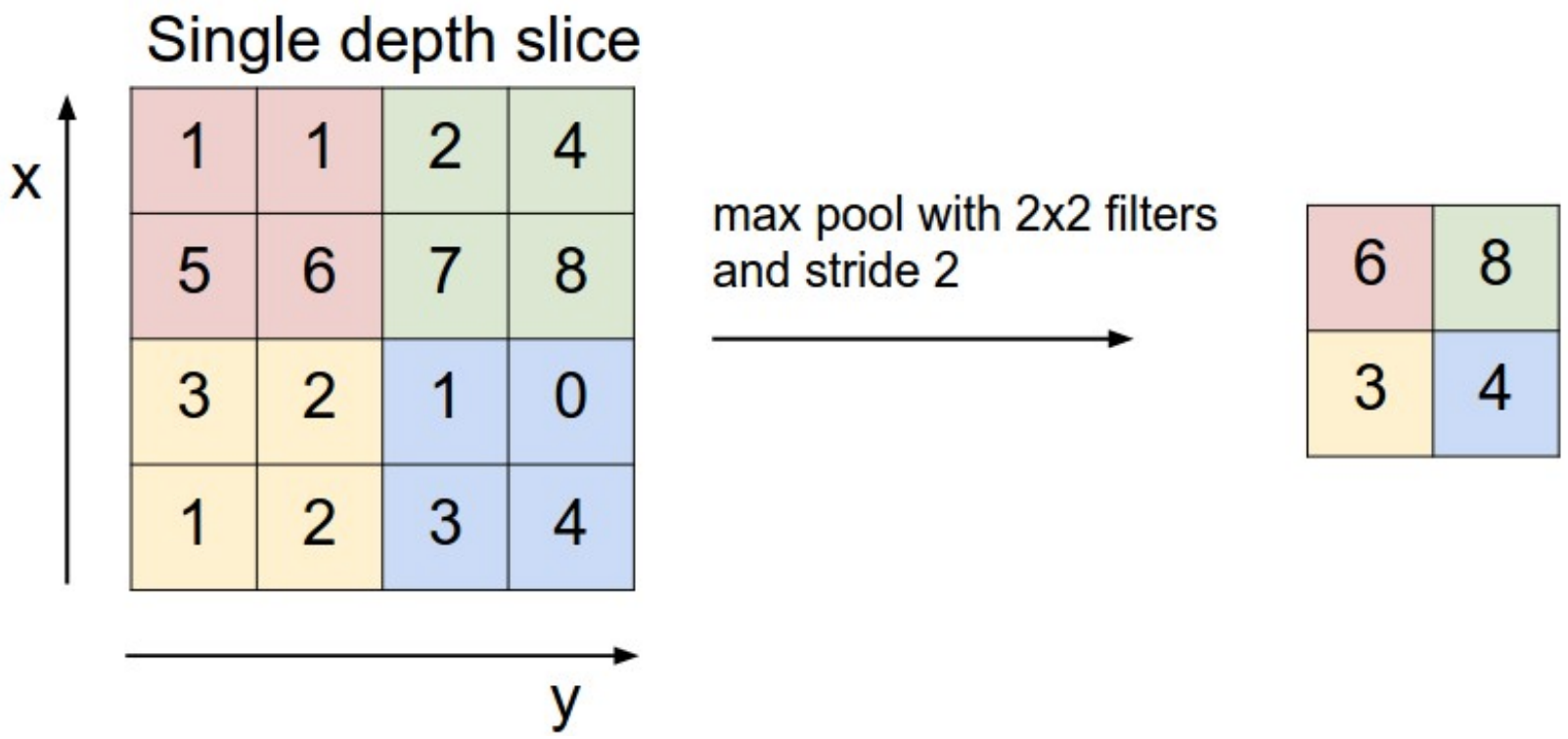
- Excels at computer vision tasks
- Specifically designed to take advantage of the spatial structure of images - *shares information across space*
- Up until now we have only discussed *Dense* layers
- Convolutional networks typically include three more types of layers
    - Convolutional
    - Pooling
    - Flatten

Convolutional layer



Source: The eScience cloud, TensorFlow Meets Microsoft CNTK

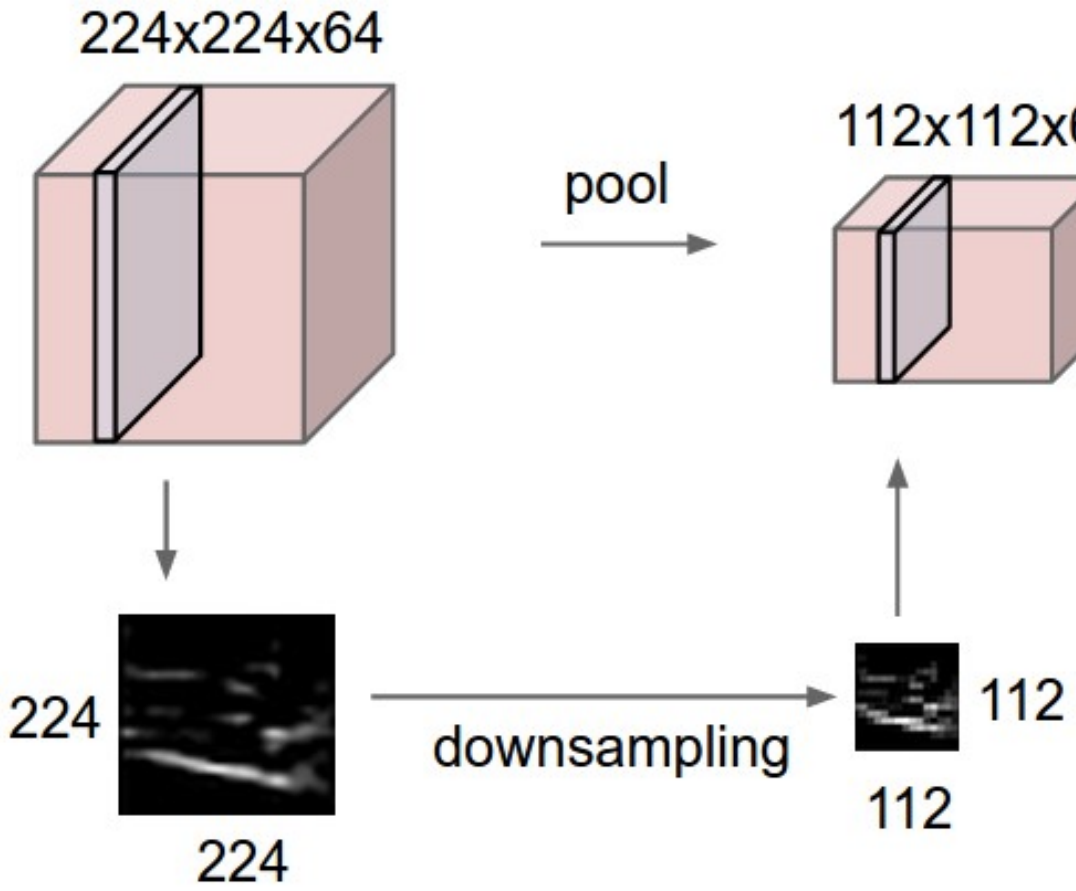
Pooling layer (I)



Pooling step carried out for each feature in a convolutional layer

Source: Cs231n Convolutional Neural Networks for Visual Recognition

Pooling layer (II)



Source: Cs231n Convolutional Neural Networks for Visual Recognition

Flatten layer

Converts three dimensional output of a convolutional or pooling layer into a one dimensional vector to be provided as input into a dense layer

Convolutional networks: airplanes and cats revisited

Improving the way networks learn

- Optimizers: alternatives / augmentations to stochastic gradient descent
- Weight initialization
- Dataset augmentation
- Adversarial training

Momentum

SGD weight update

$$w \to w' = w - \eta \nabla C$$

SGD with momentum weight update

$$v \to v' = \mu v - \eta \nabla C$$

$$w \to w' = w + v'$$

Weight initialization

- Balance between
  - Regularization -> small initial weights
  - Breaking symmetry -> large initial weights
  - Avoid saturation -> small initial weights
  - Propagating information -> large initial weights

Two strategies

Simple

$$w \sim U\left(\frac{-1}{\sqrt{m}}, \frac{1}{\sqrt{m}}\right)$$

Normalized[1]

$$w \sim U\left(\frac{-6}{\sqrt{(m+n)}}, \frac{6}{\sqrt{(m+n)}}\right)$$

m = number of inputs to a node
n = number of outputs from a node

1. Glorot and Bengio, 2010
Source: Deep Learning Book, Ian Goodfellow and Yoshua Bengio and Aaron Courville

Some guidelines for building your own networks

- Find out what a decent baseline is
- Start simple and small, get a signal first
- Then check your model can overfit the training data
- Then add regularization
- After the first layer, decrease the number of nodes in a layer, ideally by no more than a factor of 2 compared to the layer above
- Size of first layer is an art, related to complexity of problem and degrees of freedom your model needs
- “Don’t be a hero”[1] - use existing pre-trained networks and / or their hyper-parameter settings as a starting point

Source: Andrej Karpathy

Thank you