Introduction to Neural Networks with Keras

27th June 2017, O'Reilly Artificial Intelligence Conference

Laura Graesser

About me

- Student, Masters in Computer Science, NYU
- Deep Learning Intern, Autonomous Driving, NVIDIA
- Website: learningmachinelearning.org
- Twitter: lgraesser3
- Email: lhg256@nyu.edu

Logistics

- Python, Keras, Theano or Tensorflow, Matplotlib and Git already installed
- Materials: https://github.com/lgraesser/Intro-to-Neural-Networks-O-Reilly-AI
- ‘git clone https://github.com/lgraesser/Intro-to-Neural-Networks-O-Reilly-AI.git‘ to download materials
- ‘git pull’ to update
- TAs: Felipe Ducau, Anant Gupta

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

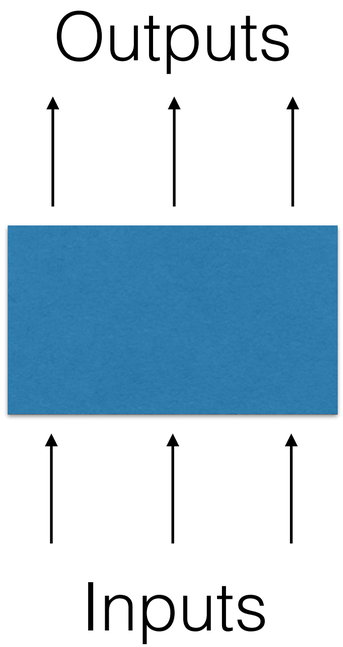
Agenda

- **13:30 - 14:45:** Theory: Deep feed-forward neural networks
- **14:45 - 15:00:** Practice: Classifying hand written digits
- **15:00 - 15:30:** Break
- **15:30 - 15:45:** Practice: Classifying hand written digits
- **15:45 - 16:15:** Theory: Overfitting and regularization
- **16:15 - 16:45:** Practice: Implementing regularization
- **16:45 - 17:00:** Wrap up

What are neural networks?

A family of algorithms that excel at making predictions about unseen data

Function approximators



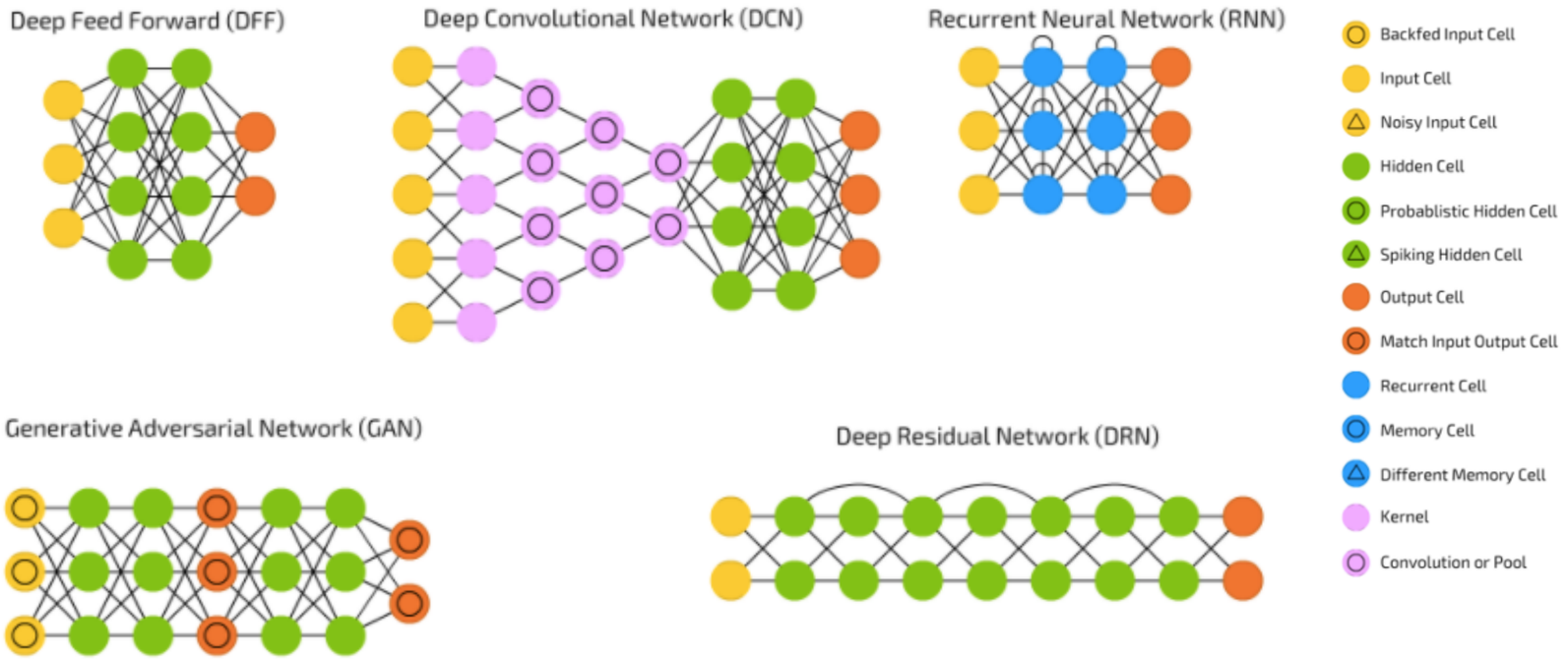
Strengths

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

A brief history of neural networks

- 1957: Perceptron algorithm invented by Frank Rosenblatt
- 1986: Backpropagation invented by Rumelhart, Hinton, and Williams
- 1989: Yann LeCun et al successfully train 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
- 2016: Neural machine translation, speech recognition, AlphaGo, GANs, autonomous vehicles, …

Many possible architectures



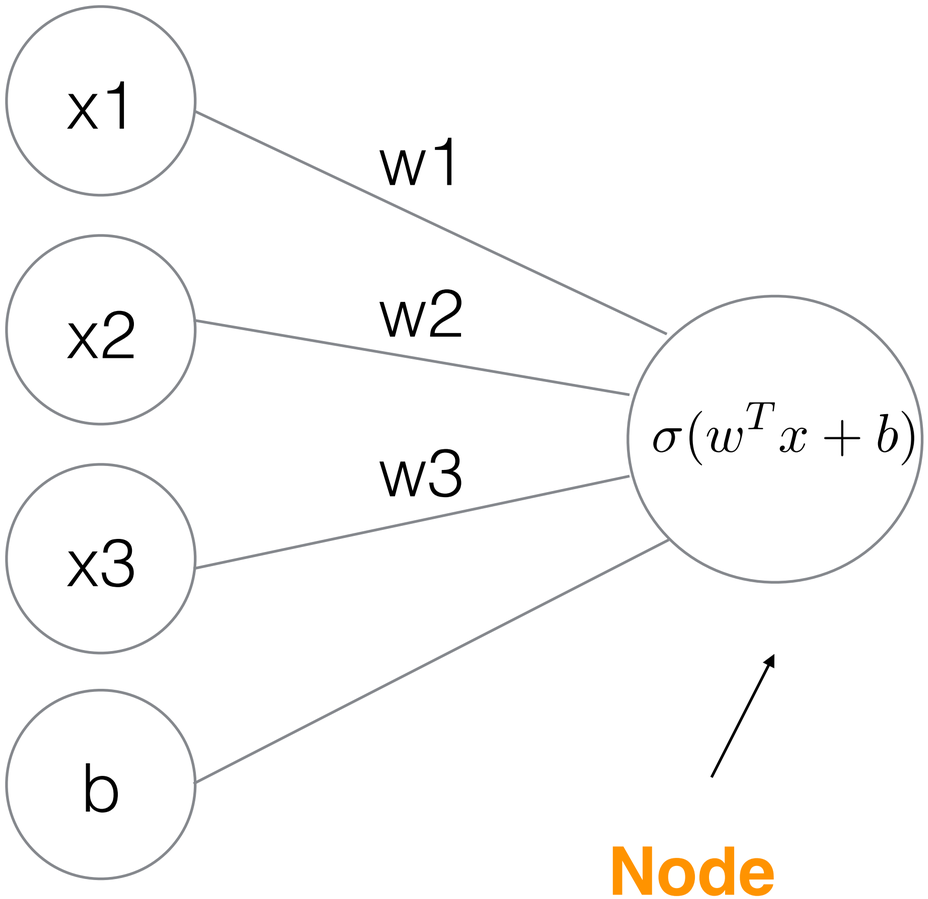
We are focusing on deep feed-forward neural networks

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



Notation

$$w^T x + b =$$

$$w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

$$\hat{w}^T = (w_1 \ w_2 \ w_3 \ b)$$

$$\hat{x}^T = (x_1 \ x_2 \ x_3 \ 1)$$

$$w^T x + b = \hat{w}^T \hat{x}$$

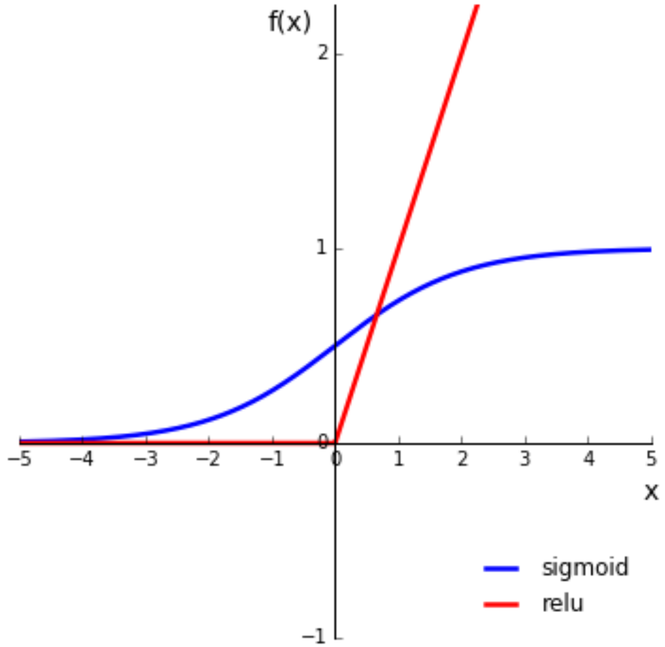
Activation functions

Sigmoid

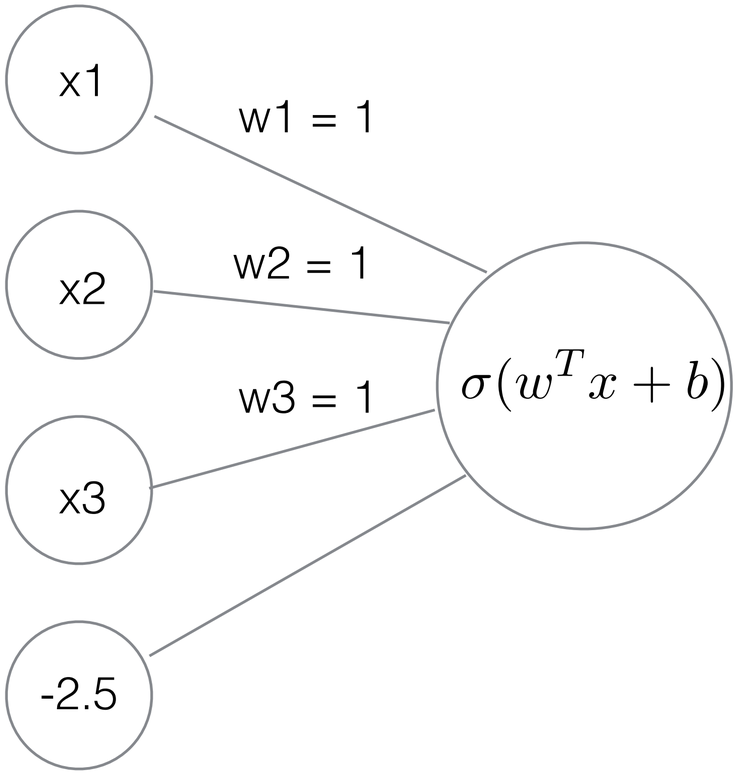
$$\sigma(z) = \frac{1}{1+e^{-z}}$$

ReLU

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



Example - AND



x_1	x_2	x_3	$w^T x + b$	$\sigma(w^T x + b)$
0	0	0	-2.5	0.076
0	0	1	-1.5	0.182
0	1	0	-1.5	0.182
0	1	1	-0.5	0.378
1	0	0	-1.5	0.182
1	0	1	-0.5	0.378
1	1	1	0.5	0.622

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

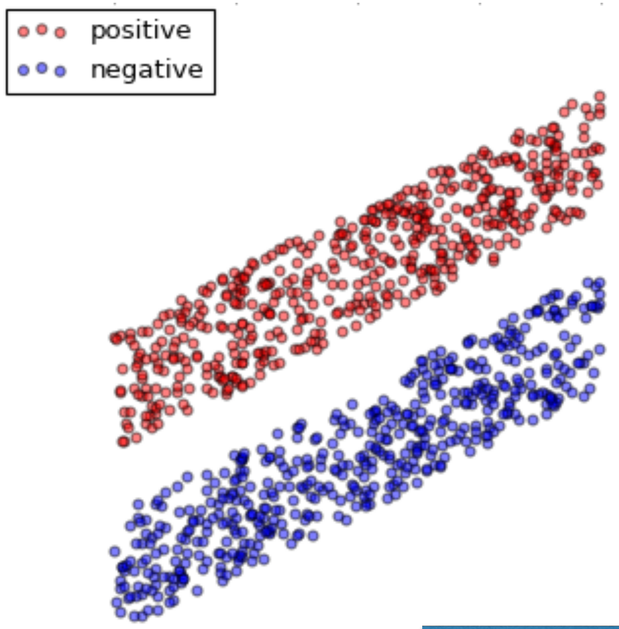
Data flows forward through a network from input to output

Source: The Neural Network Zoo, The Asimov Institute

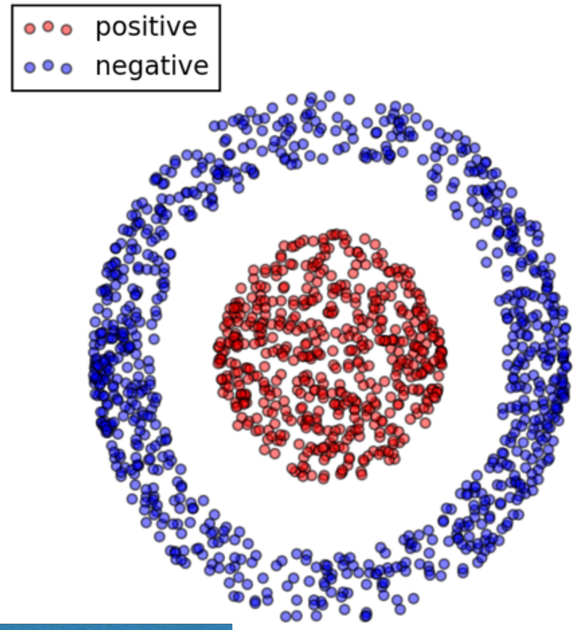
Why is this structure powerful?

Motivating example: two class classification problem

A. Data linearly separable



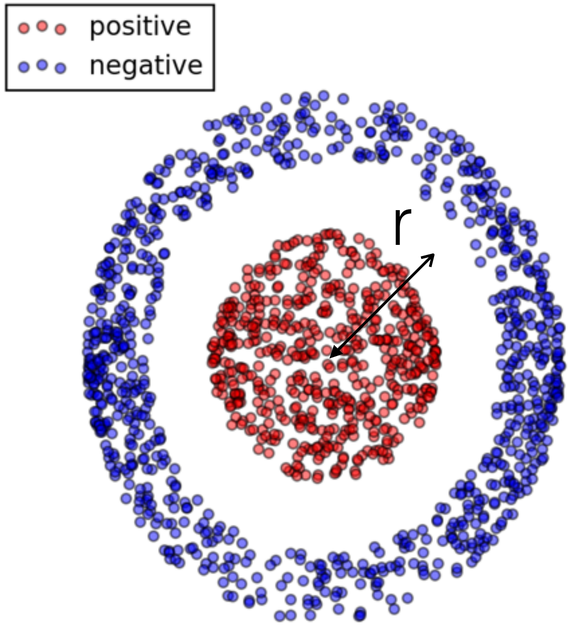
B. Data not linearly separable



How can we make B like A?

Transforming feature space

- Assume both circles centered at (0, 0), r = 1
- Positive class where $x^2 + y^2 < r^2$
- Add two new features, x^2,y^2 —> (x, y, x^2,y^2, b) = v
- Let b (bias) = r^2
- Let weights = (0, 0, 1, 1, -1)
- Then $w^T v = x^2 + y^2 - b$
- Assign positive class when $w^T v < 0$
- Data is linearly separable



How to find the right transformation?

- Previous approaches
    - Specify explicitly by manually transforming the input features
    - Pick an extremely general transformation (e.g RBF kernels)
- Neural networks *learn* the correct transformation

Non-linear activation function + depth = power

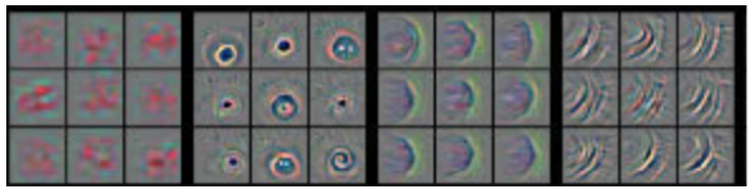
- Non linear activation function + hidden layers.
    - Each layers creates new features. These are non-linear transformations of their inputs.
    - Hierarchy of features, progressively more abstract layer by layer.
- Output layer: computes linear separating boundary (classification) from final hidden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

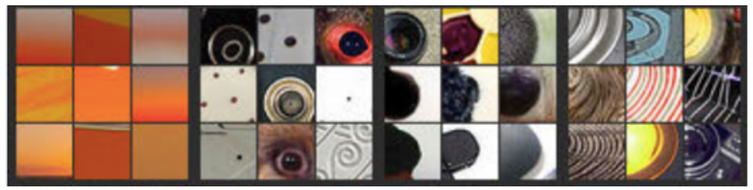
Example features by layer

Node activation projected into pixel space

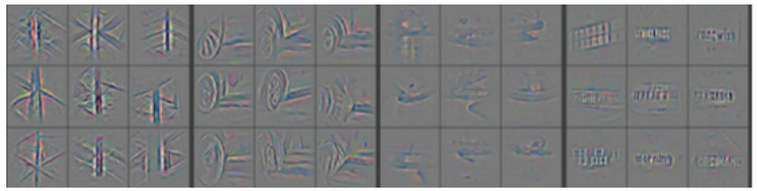
Corresponding images

Layer 2

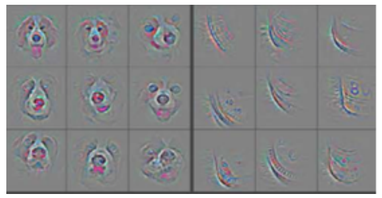




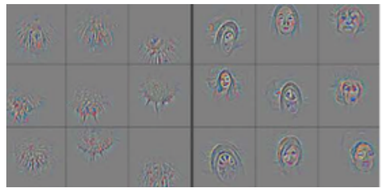
Layer 3



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
- Supervised learning - assumes we have the correct answer per datapoint
- Start with randomly initialized values
- Show model many examples, calculate error
- Propagate error between model’s output and the correct value backwards through the model

How a network learns

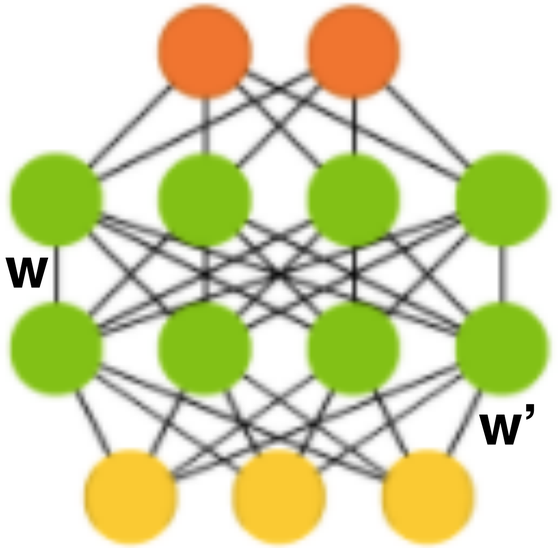
- Network inputs: I_1, I_2, I_3
- Network outputs: a_1, a_2
- Correct answer: y_1, y_2
- Error: $J(y_1, a_1) + J(y_2, a_2)$
- Need a measure of error

Loss functions: different ways of measuring error

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

Backpropagation

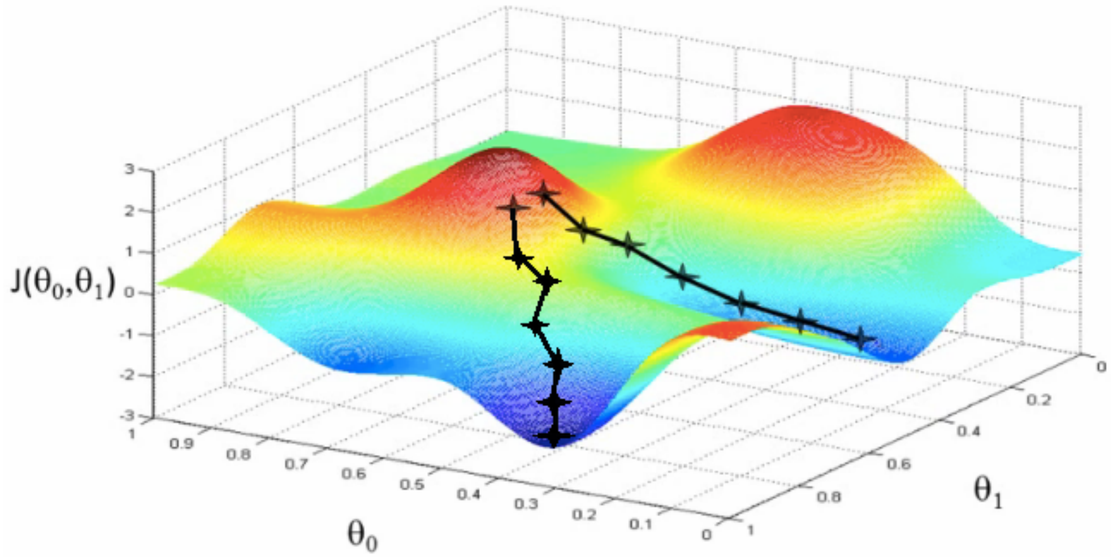
- Method of calculating the gradient of the loss with respect to the weights and biases using the chain rule
- Makes it possible to learn through gradient descent (weight update algorithm)
- Gradient calculated layer wise, each layer receives the gradient of the layer above
- Weights are updated in proportion to their contribution to the loss



Gradient descent

$\theta'_0 \leftarrow \theta_0 - \alpha \nabla_{\theta_0} J(\theta_0, \theta_1)$

$\theta'_1 \leftarrow \theta_1 - \alpha \nabla_{\theta_1} J(\theta_0, \theta_1)$



Learning rate controls the size of the steps

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

Stochastic gradient descent

- Large datasets and models make gradient descent expensive
- Weights and biases are updated after processing a random subset (minibatch) of the data instead of the entire dataset
- Batches typically contain 16 - 256 examples
- Estimation of the loss function
- More efficient, faster convergence

Recap

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

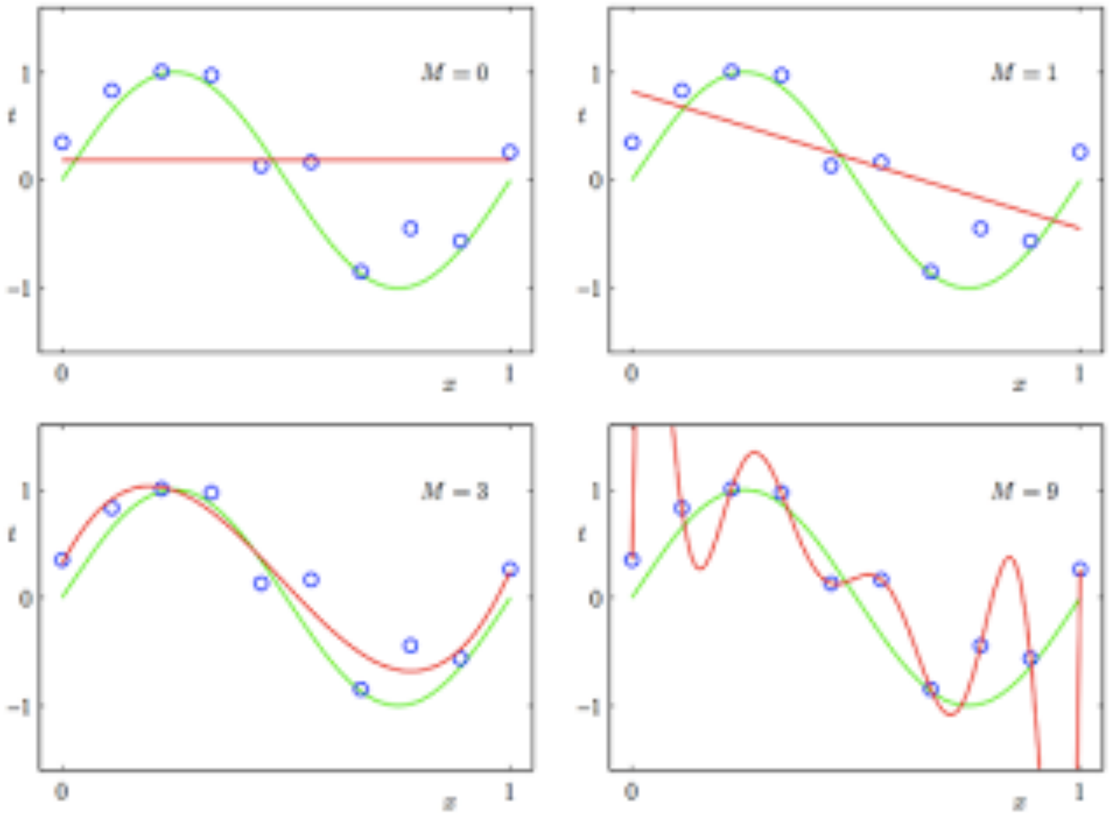
Classifying handwritten digits with Keras

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?
- Are training and validation accuracy the same?

Overfitting and regularization

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
- 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 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

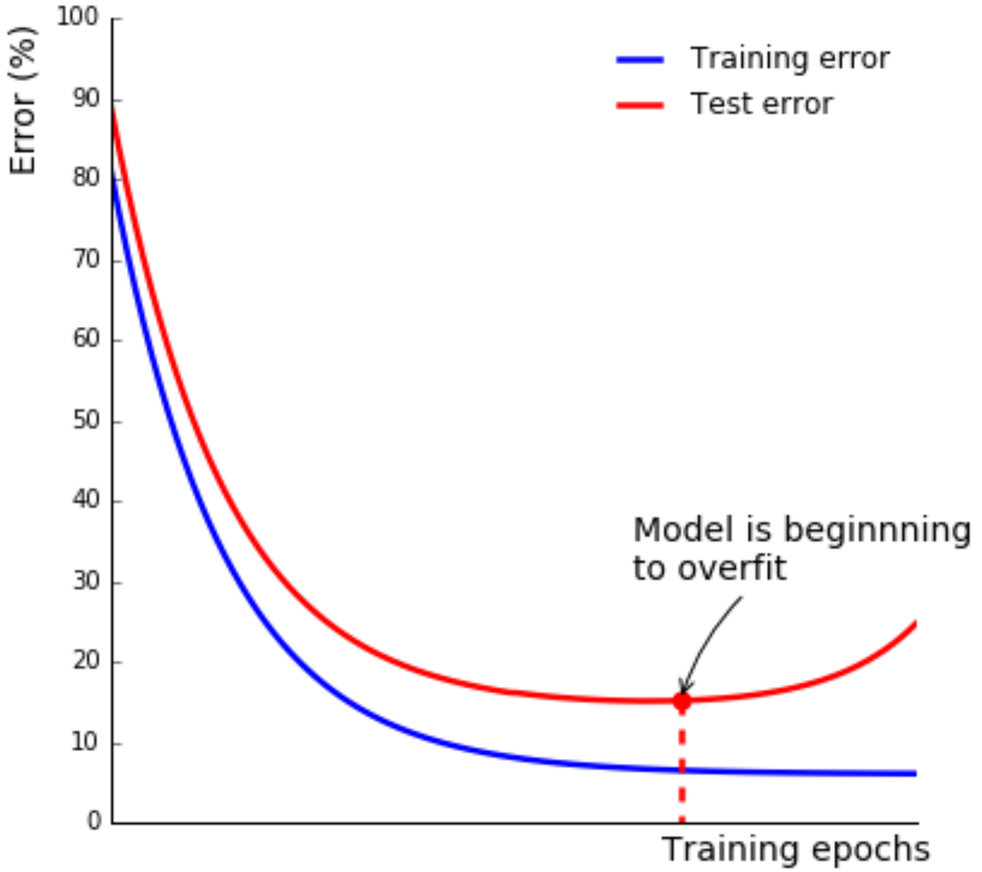
Desirable properties for a machine learning model

- Smoothness - smaller weights are better than large weights
- Simplicity - fewer features are better than more features
- Generality - features that are good in many situations are better than features that are good in few situations

Regularization techniques

- Smoothness: **L2 weight regularization**
- Simplicity: **L1 weight regularization**
- Generality: L1, L2, **early stopping, dropout**

Early stopping



Weight regularization

L1

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

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

Regularization component of the derivative only

L2

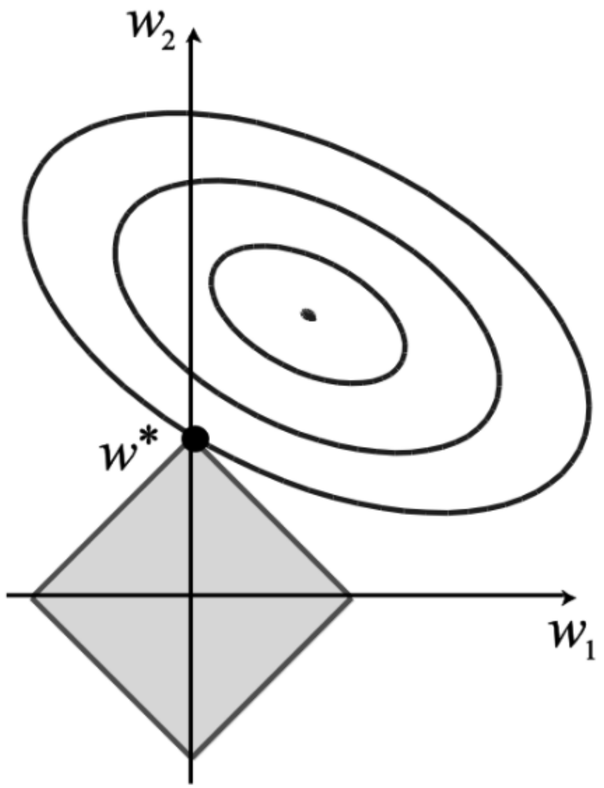
$$J = \frac{1}{n}\left(\sum_{i=1}^{n} J_i + \frac{\lambda}{2} \sum_{w \in W} w^T w\right)$$

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

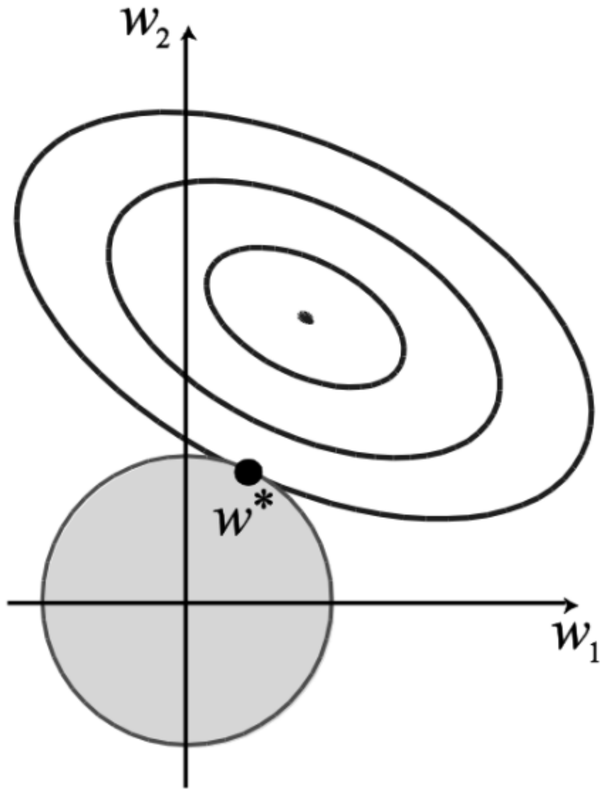
Regularization component of the derivative only

Weight regularization

L1



L2

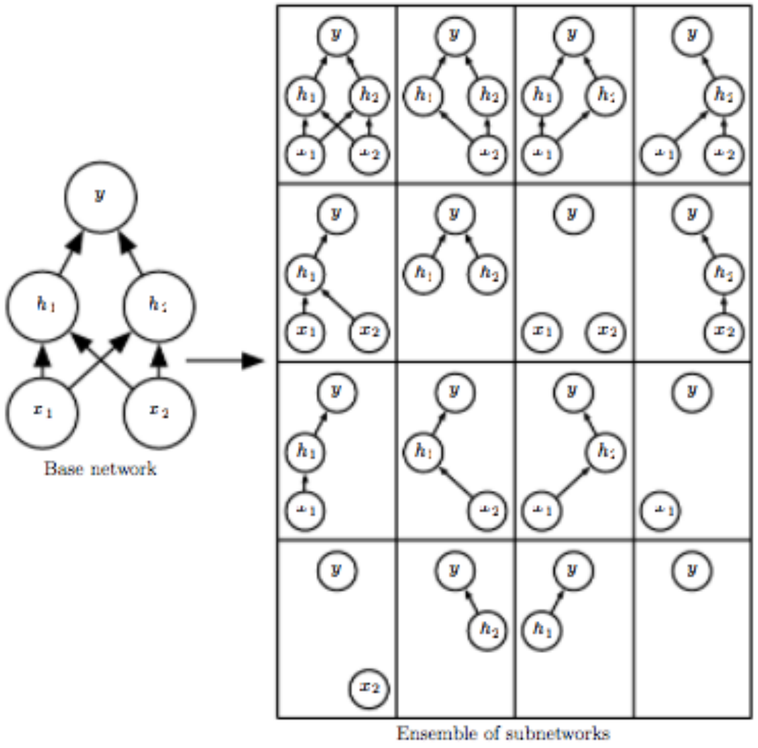


Source: Logistic Regression, Vibhav Gogate, The University of Texas at Dallas

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

Implementing regularization

Tips and tricks

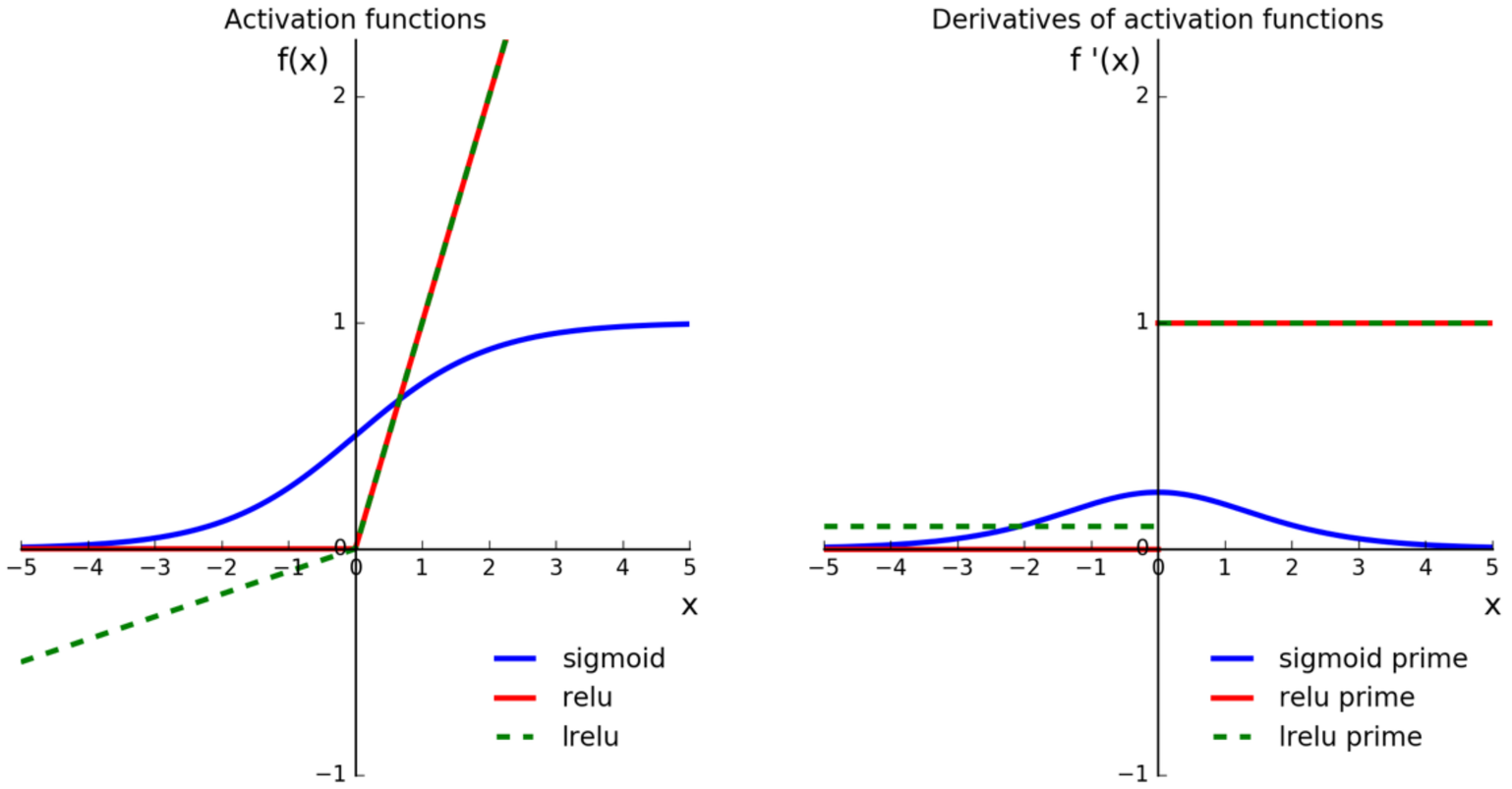
- Find out what a decent baseline is
- Start simple and small, get a signal first
- Check your model can overfit the training data
- Add regularization
- Use ReLU family of activation functions
- After the first layer, decrease the number of nodes in a layer, ideally by no more than a factor of 2 compared 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

Appendix

Training issues: nodes can saturate or die



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

Loss functions: examples

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

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

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

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