TECHNISCHE UNIVERSITÄT DRESDEN

SEMANTIC COMPUTING

Lecture 8: Introduction to Deep Learning

Dagmar Gromann

International Center For Computational Logic

TU Dresden, 7 December 2018

Overview

- Introduction Deep Learning
- General Neural Networks
- Feedforward Neural Networks

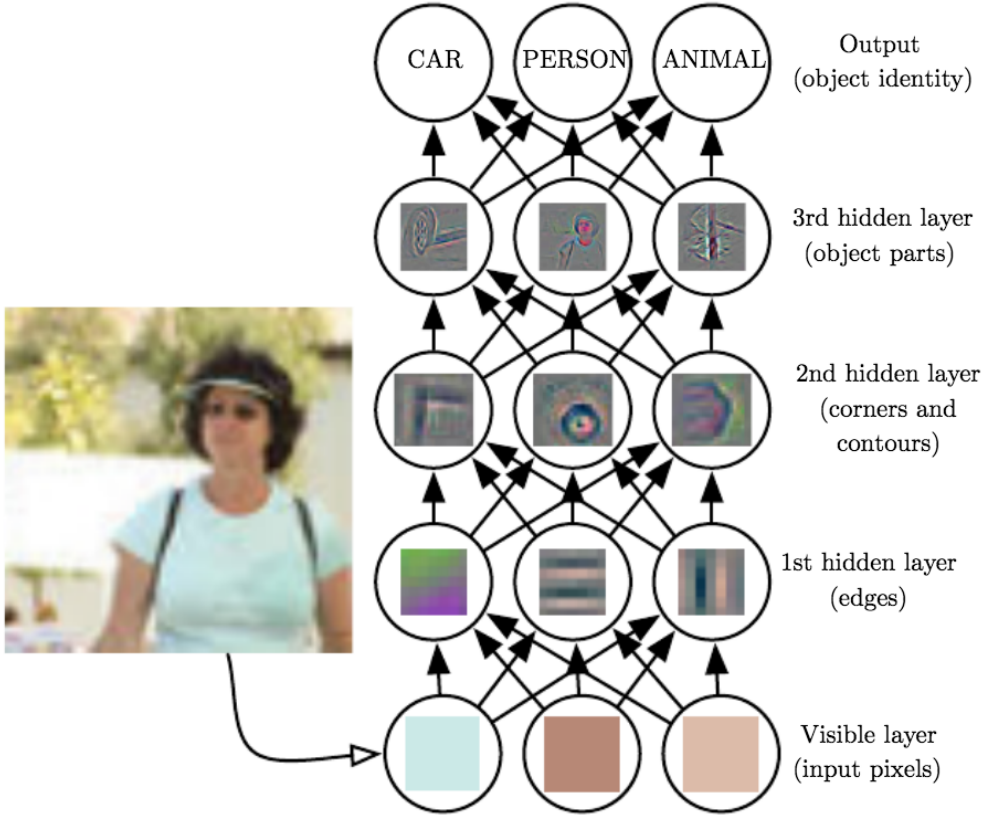
Introduction Deep Learning

Machine Learning Refresher

- Statistical machine learning algorithms rely on human-crafted representations of raw data and hand-designed features (e.g. POS tag, previous word, next word, TF-IDF, character n-gram, etc.)
- Main goal is to discover a mapping from the representation to the output
- Optimization of weights in the target function to optimize final prediction
- Most time has to be invested into finding the optimal features for your task and optimizing the parameter settings of your algorithm

Deep Learning

- A subfield of machine learning
- Representation learning attempts to automatically learn good features from raw data
- Multiple levels of representations (here: 3 layers)
- Builds complex representations from simpler ones



Representation Learning

Image source: Goodfellow et al. (2016) Deep Learning, MIT Press.
http://www.deeplearningbook.org/

Brief History

- Speech recognition breakthrough in 2010
  - Hinton, G. et al. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal Processing Magazine, 29(6), pp. 82-97.
- Computer vision breakthrough in 2012 (halving the error rate to 16%)
  - Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems, pp. 1097-1105.
- Today: vast amounts of papers published on a daily basis; some help: `http://www.arxiv-sanity.com/`

Application Examples of Deep Learning

- Neural Machine Translation
- Text summarization
- Text generation preserving the style
- Linguistic analysis (syntactic parsing, semantic parsing, etc.)
- ...

Image Captioning

Automatically generating image descriptions:

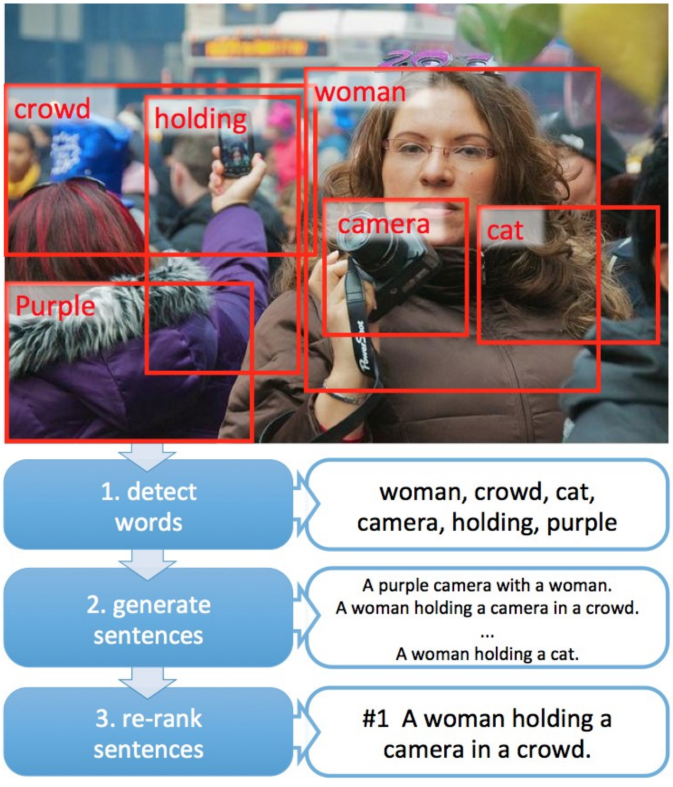
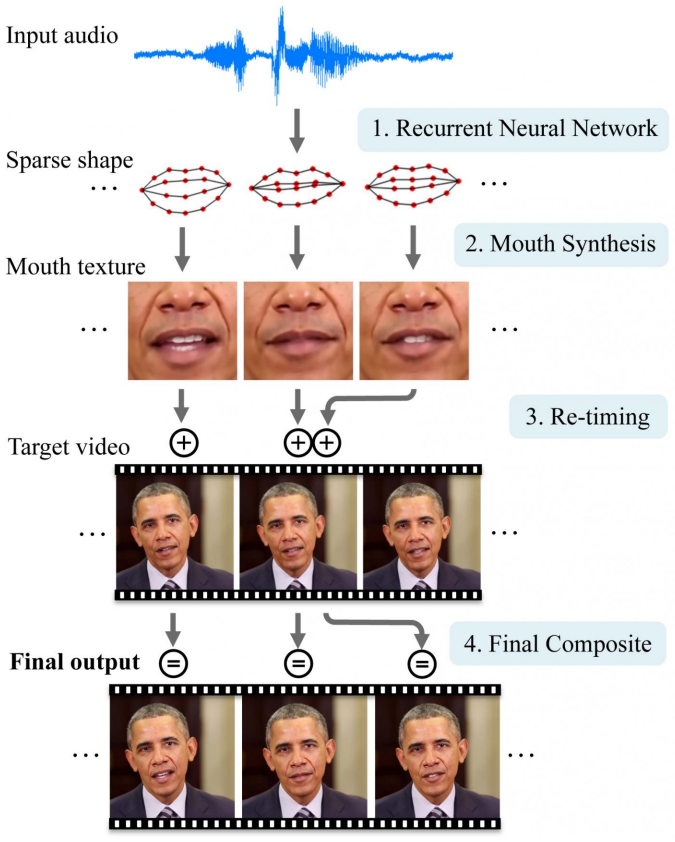


Image source: Fang, H. et al. (2015). From captions to visual concepts and back. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1473-1482).

Lip Sync from Audio



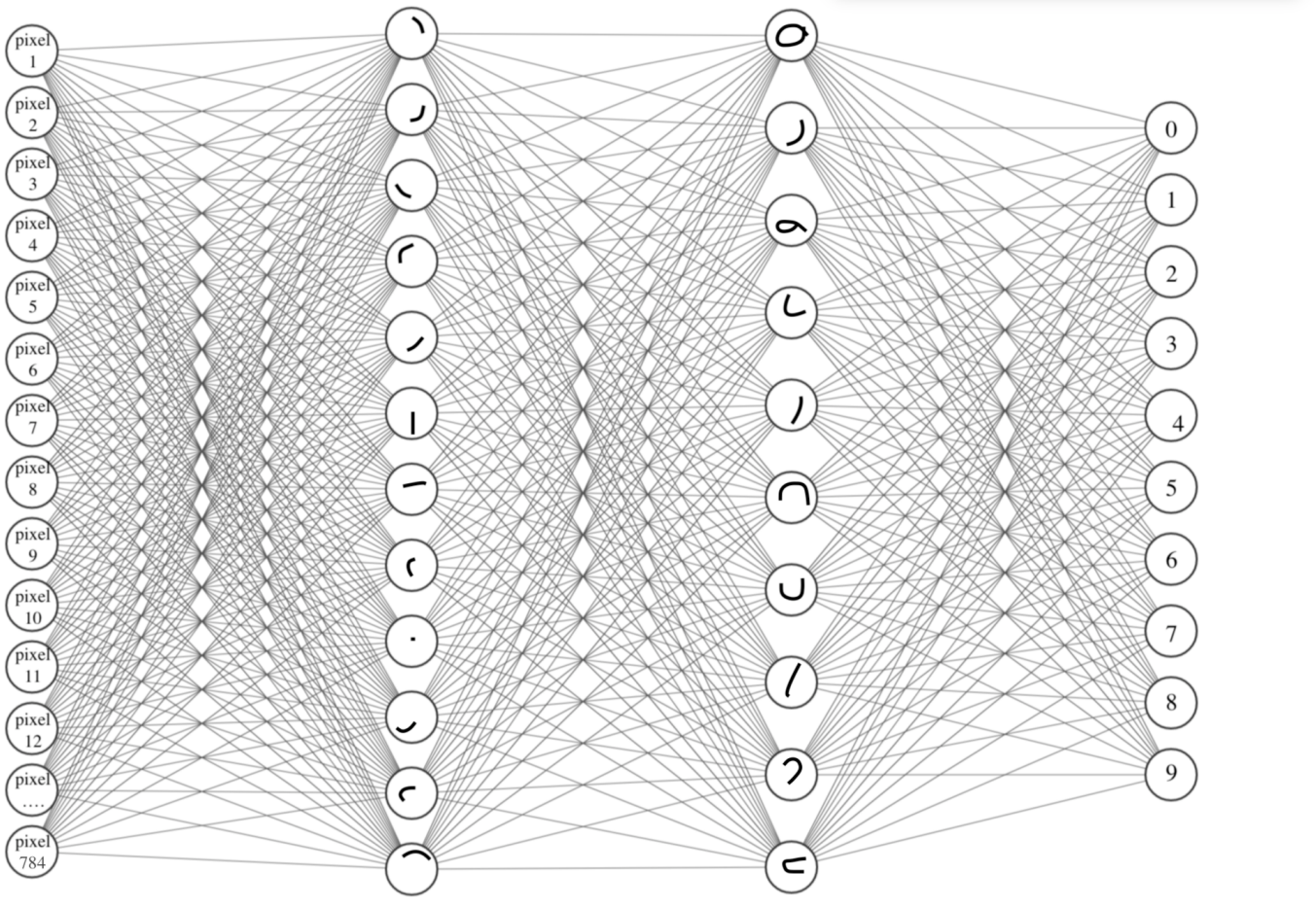
Paper: Suwajanakorn, S., Seitz, S. M., & Kemelmacher-Shlizerman, I. (2017). Synthesizing obama: learning lip sync from audio. ACM Transactions on Graphics (TOG), 36(4), 95.

Winning Atari Breakout

Google's DeepMind learns to play the Atari game Breakout without any prior knowledge based on deep reinforcement learning.

General Neural Network Example

Architecture: 2-Layer Feedforward NN



Why “neural”?

- inspired by neuroscience
- neurons: highly connected and perform computations by combining signals from other neurons
- deep learning: densely interconnected set of simple units (e.g. sigmoid units depending on **activation function** computed based on inputs from other units)
- vector-based representation:
  - X features space: (vector of) continuous or discrete variables
  - Y output space: (vector of) continuous or discrete variables
  - many layers of vector-valued representations, each unit: vector-to-scalar function
- more function approximation machine than model of the brain

Why “network”?

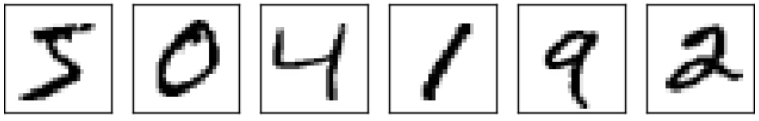
- compose together many different activation functions across different layers in a chain format (directed, acyclic graph)
- connection of layers: **input** - **hidden** (number of layers specified) - **output**
- training algorithm decides itself how to use the neurons of each layer between input and output, which is always called **hidden** layer
- **depth** of a network is specified by the number of its layers
- **width** of a layer is specified by the predefined number of units (can only be defined for a hidden layer

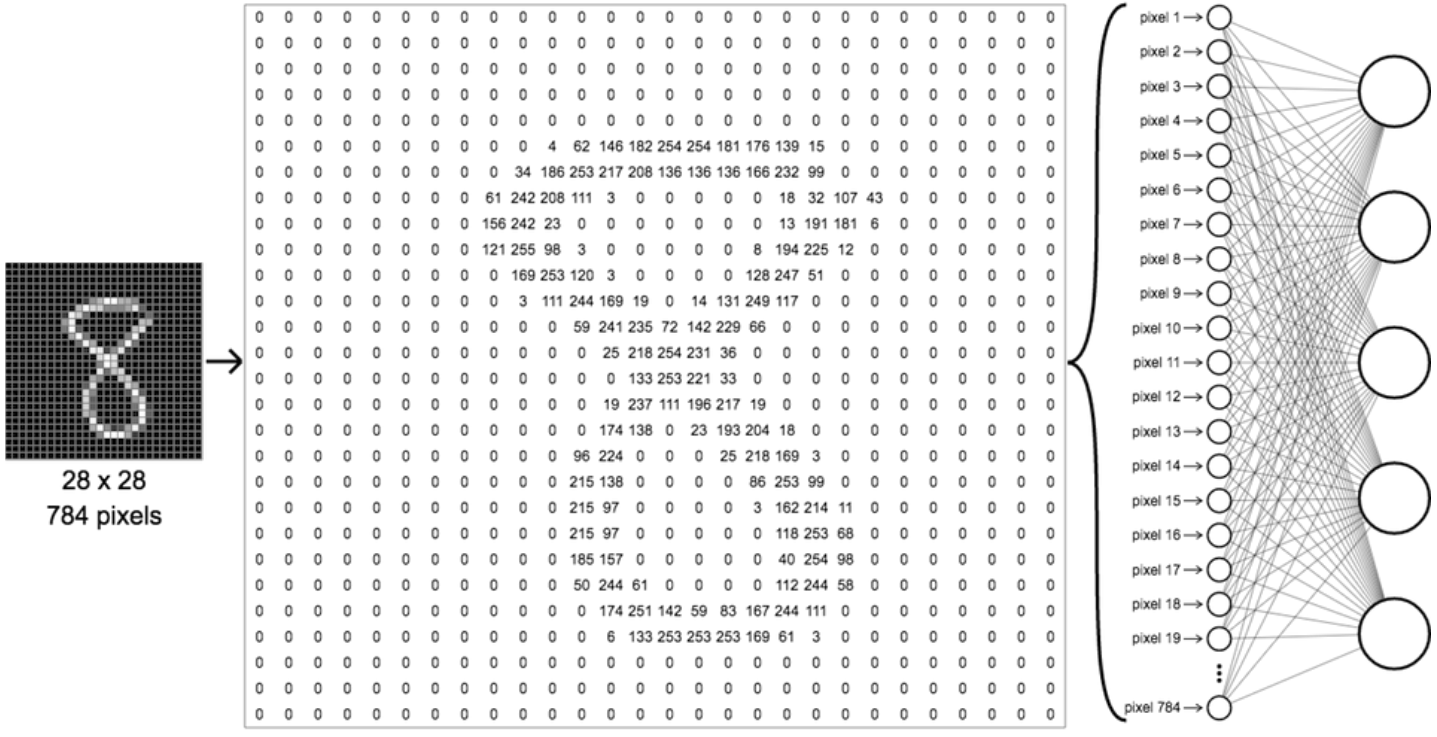
Neural Network: Main Design Decisions

You have to decide on the selection of or number of the following when building a neural network:

- form of units
- cost or loss function
- optimizer
- architecture design:
  - number of units in each layer
  - number of layers
  - type of connection between layers

Input: Handwritten Digits (MNIST dataset)

Input images: 



Dataset: `http://yann.lecun.com/exdb/mnist/`

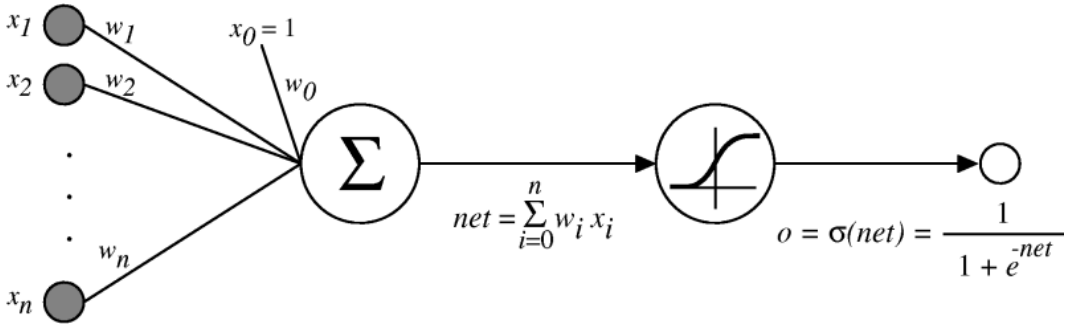
Form of Hidden Units

The type of unit is defined by its activation function;
Activation functions decide whether a neuron should be activated (“fire”) or not, that is, whether the received information is relevant or should be ignored (= nonlinear transformation over the input signal)

- Sigmoid
- Tanh
- Rectified Linear Unit (ReLU)
- Leaky ReLU
- Softmax
- ...

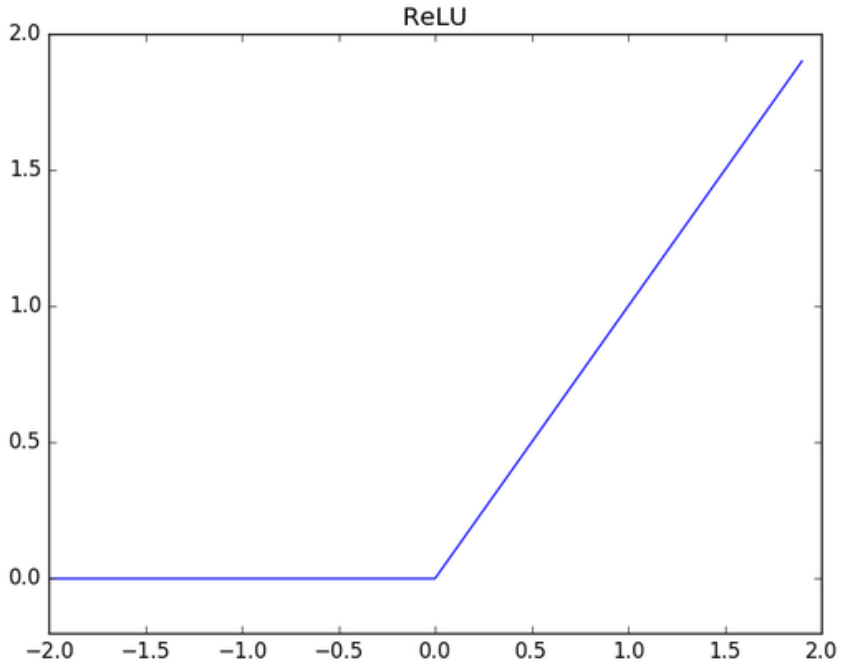
Hidden units: frequently ReLU

Sigmoid Unit



- sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$
- non-linear activation function (0 to 1 in the S-Shape)
- $g(x) = \frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$ => high between values of -3 and 3: in this range small changes of x would bring about large changes of y (highly desirable property)
- with non-linear functions we can backpropagate (update weights) and have several layers (no difference with linear function)

Rectified Linear Unit (ReLU)



- function: $f(x) = max(0, x)$
- not all neurons are activated at the same time
- if input is negative => conversion to zero and neuron inactive
- advantage: more efficient computation

Overview Output Layer

Output Type	Output Distribu-tion	Output Layer	Cost Function
Continuous	Gaussian	Linear	Gaussian cross-entropy (MSE)
Binary	Bernoulli (binary variable)	Sigmoid	Binary cross-entropy
Discrete	Multinoulli	Softmax	Discrete cross-entropy
...			

Details Softmax Output Units

- As output unit: squishes a general vector z of arbitrary real values to a vector $\sigma(z)$ of real values where each entry is in the range of [0,1] and entries add up to 1
- Objective: all neurons representing important features for a specific input (e.g. loop and line for a “9”) should be activated, i.e., fire, when that input is passed through the network
- Notation: $\sigma(z)_j = \frac{exp(z_j)}{\sum_j exp(z_j)}$
- Used for: **multiclass** classification, e.g. multinomial logistic regression and neural networks

Forward Propagation

- input to hidden layer 1 : $H_1 = W_1^\top \boldsymbol{x} + b_1$
- hidden 1 to hidden 2: $H_2 = W_2^\top H_1 + b_2$
- in other words: weighted sum of activation of previous layer and weights of current layer:
  $w_1 h_1^{(1)} + w_2 h_2^{(1)} + w_3 h_3^{(1)} + \ldots + w_n h_n^{(1)}$ where $h_{1-n}^{(1)} \in H_1$ and $w_{1-n} \in W_2$
- all bright pixels are set to a positive value, direct neighbors to negative and the rest to zero
- maybe we only want to consider weighted sums that are greater than 10: such constraints are introduced by means of the bias (b_2) $w_1 h_1^{(1)} + w_2 h_2^{(1)} + w_3 h_3^{(1)} + \ldots + w_n h_n^{(1)} - 10$
- weights: what pixel pattern the layer is picking up on
- bias: how high the weighted sum needs to be before the neuron starts getting active

Cost or loss function

- similar to linear models: our model defines a distribution $p(y|x;\theta)$ and we use the principle of maximum likelihood (negative log-likelihood)
- cost function is frequently cross-entropy between training data and predictions

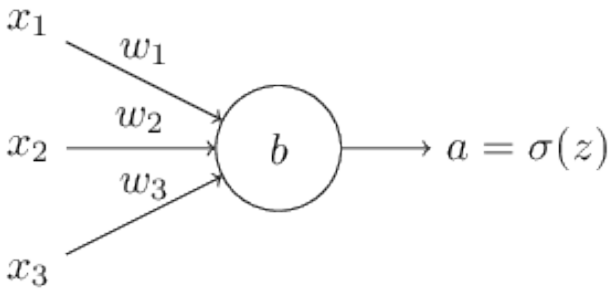
$$binary : -(y\ log(\hat{y}) + (1-y)\ log(1-\hat{y}))$$

$$multiclass : -\sum_{i=1}^{M} y_i\ log(\hat{y}_i)$$

M ... number of classes; y ... correct label; $\hat{y}$ .... prediction

Cross-entropy cost function

Example:



Of this neuron, the output = $a = (z)$ where $z = \sum_j w_j x_j + b$ is the weighted sum of inputs. Then the cross entropy cost function for this neuron is

$$C = -\sum y \; log(a)$$

Backpropagation

Backpropagation

Backpropagation, also called backprop, allows information from the cost to flow backward through the network to compute the gradient. NOT a learning method, but a very efficient method to compute gradients that are used in learning methods (optimizers).

- "Just the chain rule" of derivatives ($z = W^\top h + b$ and $h^l = \sigma(z^l)$ and $J = (h^l - \hat{y})^2$):
  $\frac{\partial J}{\partial w^l} = \frac{\partial z^l}{\partial w^l} \frac{\partial h^l}{\partial z^l} \frac{\partial J}{\partial h^l} = h^{l-1} \sigma'(z^l) 2(h^l - \hat{y})$
- average over all training examples: $\frac{\partial J}{\partial w^l} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial J_k}{\partial w^l}$ = one value of the vector that represent the gradient of the cost function (repeat for all weights and biases)

What is optimization?

If we knew the distribution of our data, the training would be an optimization problem. However, we only have a sample of training data and do not know the full distribution of the data in machine learning.

Task: minimize the expected loss on the training set
Usual algorithm: (Mini-batch) Gradient Descent

Optimization

Optimization

Minimizing or maximizing some objective function $f(\boldsymbol{x})$ by altering $\boldsymbol{x}$; when minimizing, the function is also called cost function, loss function, or error function; value that minimizes or maximizes a function usually denoted with *, e.g. x^*

How to optimize? Many different options. Most important: take the derivative $f'(\boldsymbol{x})$ because it gives you the slope of $f(\boldsymbol{x})$ at the point $\boldsymbol{x}$; specifies which small change is needed in the input to obtain a small improvement in the prediction

Gradient Descent

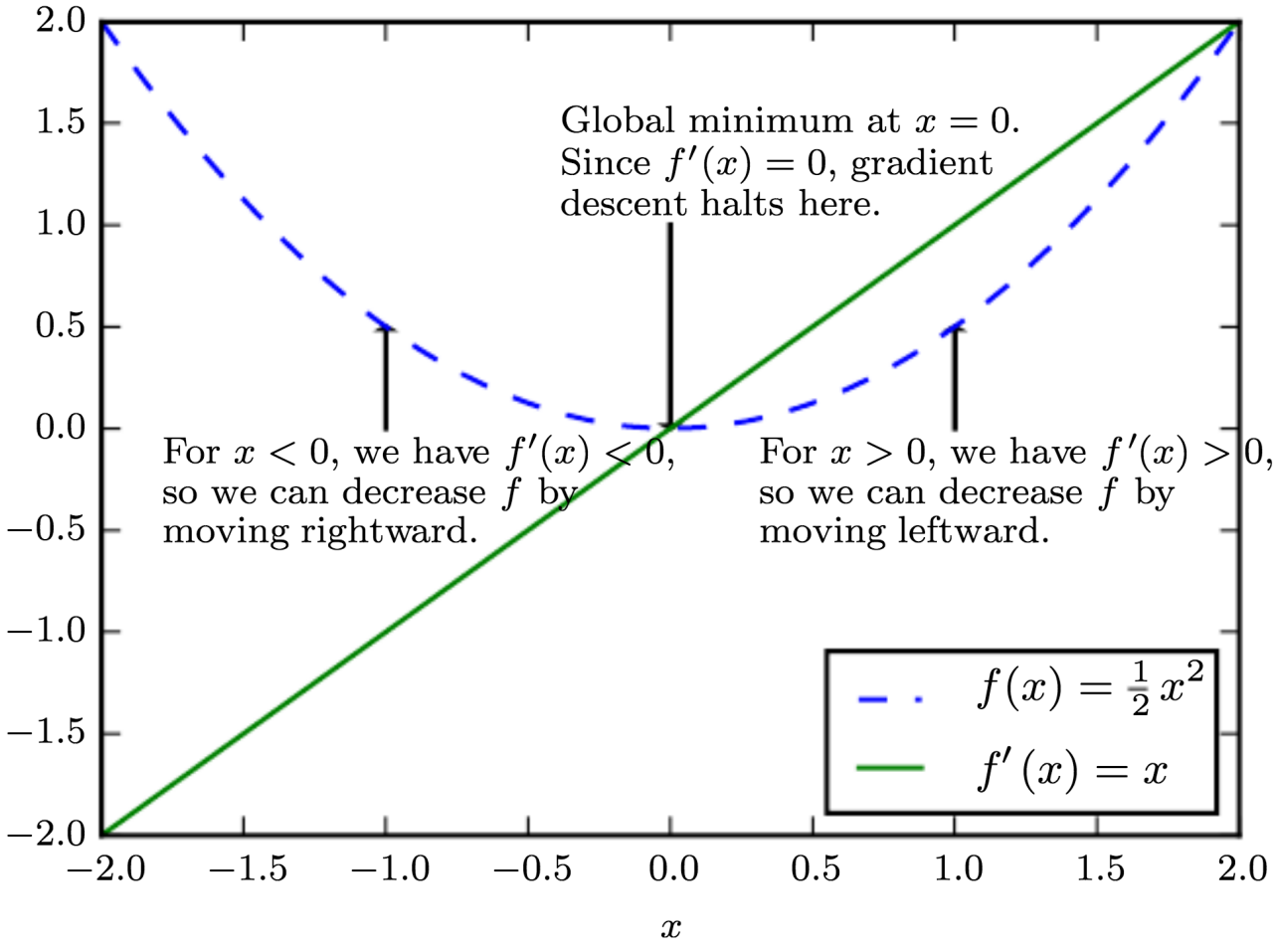


Image source: Goodfellow et al. (2016) Deep Learning, MIT Press. `http://www.deeplearningbook.org/`

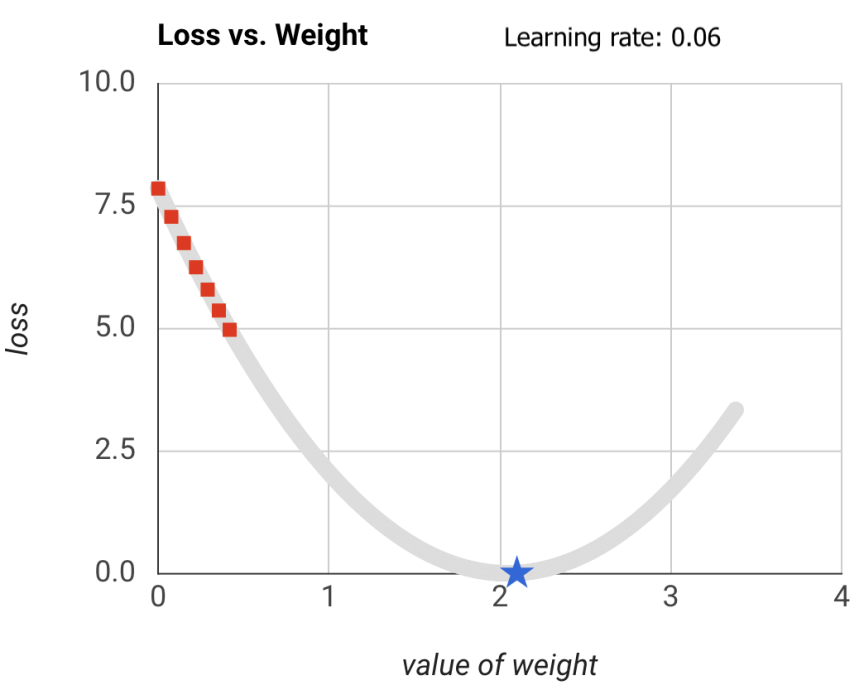
Gradient

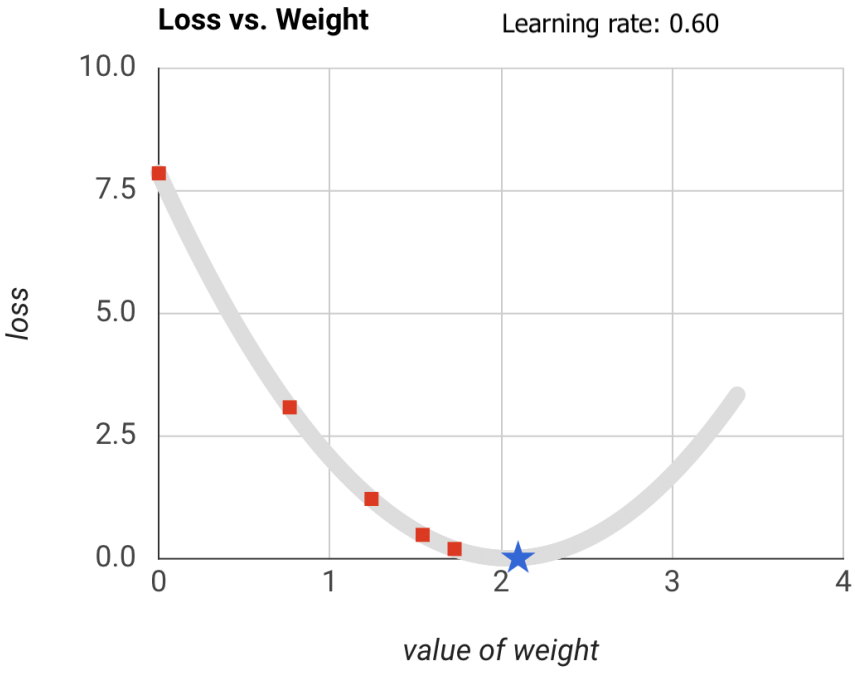
Functions with multiple inputs require partial derivatives. The **gradient** of f is a vector containing all the partial derivatives denoted $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$.

1: Set initial parameters $\theta_1^0, \ldots, \theta_k^0$
2: ϵ = learning rate (step size)
3: **while** not converged calculate ∇f **do**
4: $\quad \theta_1 = \theta_1 - \epsilon \frac{\partial f}{\partial \theta_1}$.
5: $\quad$ ...
6: $\quad \theta_k = \theta_k - \epsilon \frac{\partial f}{\partial \theta_k}$
7: **end while**
8: small enough ϵ ensures that $f(\theta_1^i, \ldots, \theta_k^i) \leq f(\theta_1^{i-1}, \ldots, \theta_k^{i-1})$

Learning Rate

Also called step size which is more intuitive:





0.06 = 138 steps to reach the minimum; 0.60 = 12 steps needed

1.60 = 1 step; 1.70 = 2 steps

Image produced by `https://developers.google.com/machine-learning/crash-course/fitter/graph`

Overview: Gradient Descent

Definition Gradient Descent in NN

Method to (step-wise) **converge** to a local minimum of a cost function by iteratively calculating the gradient and adapting the weights of the neural network accordingly in order to minimize the loss.

- **(Batch) gradient descent**: update weights after having passed through the whole dataset (use all examples at once); that is $\sum \frac{\partial J}{\partial W}$
- **Stochastic gradient descent**: update weights incrementally with each training input $\frac{\partial J}{\partial W}$ (one example at a time)
- **Mini-batch gradient descent**: 10-1000 training examples in a batch; loss and gradient are averaged over the batch (equivalent to one step)

SGD

- needs more steps than GD but each step is cheaper (computation)
- there is not always a decrease for each step
- **minibatch**:
  - common way to uniformly draw examples from the training data
  - usually very small between one and a few hundred examples
  - batch size is held fixed even with an increasing training set size
- SGD outside of deep learning: e.g. main way to train linear models on large datasets

Architecture Design

- number of units: dependent on presumed number of characteristics for each feature (e.g. how many different loops in MNIST task?) that layer should represent; deeper networks (more layers) = potentially fewer units
- number of layers: dependent on the presumed different types of features (e.g. contours, edges, etc. in image classification); start out with simplest solution and then slowly increase
- interconnections: fully connected means each input is connected to every output; reducing the number of connections reduces the number of parameters that have to be computed (more later)

Feedforward Neural Networks

Feedforward Neural Networks

- approximate a function $f*$
- defines a mapping $y = f(\boldsymbol{x}, \theta)$ and learns the value of θ that represents the best function approximation
- feedforward means information flows from function being evaluated from $\boldsymbol{x}$ through intermediate computation to define f to output $\hat{y}$
- no feedback connections from output that are fed back - no cycles, no loops
- with feedback connections = recurrent neural network
- special kind of feedforward: Convolutional Neural Networks (CNN)

Review of Lecture 8

- How does deep learning differ from statistical machine learning?
- What are some typical application scenarios of deep learning?
- Why is it called a network? And why neural?
- Which optimizer do you know and how does it work?
- What other design decisions are central in deep learning?
- How do you choose the correct number of layers?