UPPSALA UNIVERSITET

Machine learning, big data and artificial intelligence – Block 4

Måns Magnusson
Department of Statistics, Uppsala University

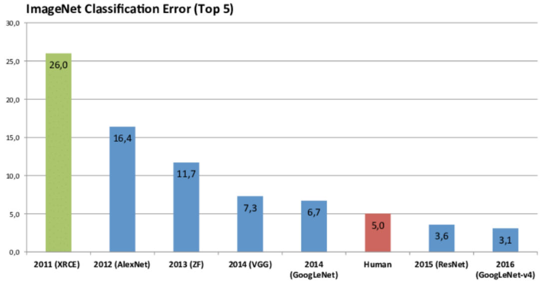
November 2020

This week's lecture

- Feed-Forward Neural Networks

The Hype: Computer Vision

Figure: ImageNet performance (Roessler, 2019)



The Hype: Speech Recognition

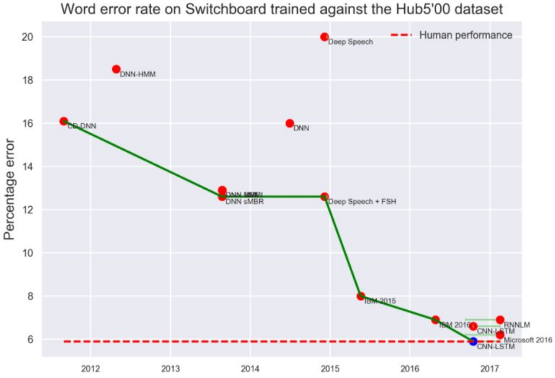


Figure: Speech recognition performance (source: `https://eff.org/ai/metrics`)

The Hype: Natural Language Processing

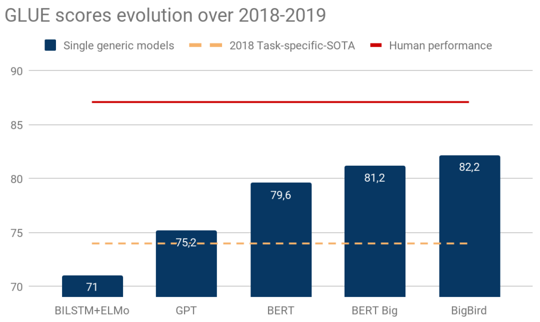


Figure: General Language Understanding (source: `https://www.programmersought.com/article/4251948498/`)

Work is very much ongoing:

`https://gluebenchmark.com/leaderboard`

The Hype

- Although - Neural Networks is not a silver bullet

The Hype

- Although - Neural Networks is not a silver bullet
- Remember the Bayes error

The Hype

- Although - Neural Networks is not a silver bullet
- Remember the Bayes error
- Some times a linear regression (or Random Forest) is enough

UPPSALA UNIVERSITET

The Feed-Forward Network

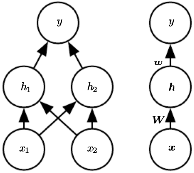


Figure 6.2: An example of a feedforward network, drawn in two different styles. Specifically, this is the feedforward network we use to solve the XOR example. It has a single hidden layer containing two units. *(Left)*In this style, we draw every unit as a node in the graph. This style is very explicit and unambiguous but for networks larger than this example it can consume too much space. *(Right)*In this style, we draw a node in the graph for each entire vector representing a layer's activations. This style is much more compact.

Figure: A simple feed-forward network (Goodfellow et al, 2017)

Important concepts:

Layers, neurons, input, output, weights, bias, architecture

UPPSALA UNIVERSITET

Different Architectures for Different Purposes

- Different networks for different purposes
  - Convolutional Neural Networks: Computer Vision

Different Architectures for Different Purposes

- Different networks for different purposes
  - Convolutional Neural Networks: Computer Vision
  - Recurrent Neural Networks: Speech Audio (?)

UPPSALA UNIVERSITET

Different Architectures for Different Purposes

- Different networks for different purposes
  - Convolutional Neural Networks: Computer Vision
  - Recurrent Neural Networks: Speech Audio (?)
  - Transformers/Attention: Textual data
- The Neural Network Zoo: `https://www.asimovinstitute.org/neural-network-zoo/`

Areas of Use: All fields

- Supervised learning
- Unsupervised learning
- Reinforcement learning

Why and when neural nets?

- Learning feature representations
- Needs a lot of data to learn complex representations
- Good for sensor data (high-dimensional)

Why and when neural nets?

- Learning feature representations
- Needs a lot of data to learn complex representations
- Good for sensor data (high-dimensional)
- When should we not use neural networks?

Learning Representations

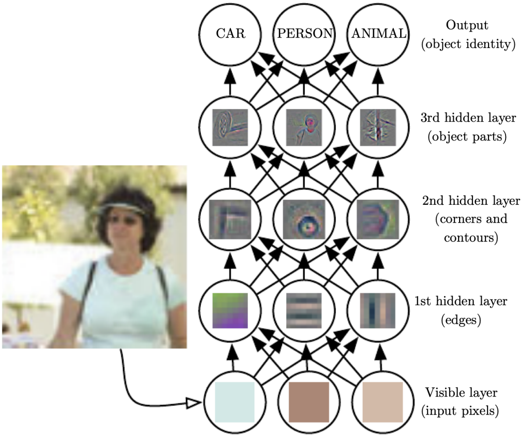


Figure: Learning representations can be crucial (Goodfellow et al, 2017, Fig. 1.2)

The Feed-Forward Network

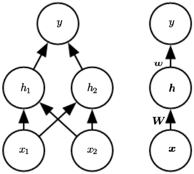


Figure 6.2: An example of a feedforward network, drawn in two different styles. Specifically, this is the feedforward network we use to solve the XOR example. It has a single hidden layer containing two units. *(Left)*In this style, we draw every unit as a node in the graph. This style is very explicit and unambiguous but for networks larger than this example it can consume too much space. *(Right)*In this style, we draw a node in the graph for each entire vector representing a layer's activations. This style is much more compact.

Figure: A simple feed-forward network (Goodfellow et al, 2017, Fig. 6.2)

In mathematical notation:

$$\mathrm{w}^T g(\mathrm{W}^T \mathrm{x} + \mathrm{b}_1) + \mathrm{b}_2$$

UPPSALA UNIVERSITET

- Introduction to Neural Networks
  - Feed-Forward Neural Networks
  - Hyper-parameters
- Optimization
- Regularization

The Feed-Forward Network

A feed-forward network for one observation (x_i).

$$\underbrace{\mathrm{h}_1}_{1\times k_1} = g_1(\underbrace{\mathrm{x}^T}_{1\times p}\underbrace{\mathrm{W}_1}_{p\times k_1} + \underbrace{\mathrm{b}_1}_{1\times k_1})$$

$$\vdots$$

$$\underbrace{\mathrm{h}_l}_{1\times k_l} = g_l(\underbrace{\mathrm{h}_{l-1}^T}_{1\times k_{l-1}}\underbrace{\mathrm{W}_l}_{k_{l-1}\times k_l} + \underbrace{\mathrm{b}_l}_{1\times k_l})$$

$$\vdots$$

$$\underbrace{\hat{\mathrm{y}}}_{1\times m} = g_L(\underbrace{\mathrm{h}_{L-1}^T}_{1\times k_{l-1}}\underbrace{\mathrm{W}_L}_{k_{l-1}\times m} + \underbrace{\mathrm{b}_L}_{1\times m})$$

$$\hat{y} = f_L(f_{L-1}(\ldots f_1(x)\ldots))$$

Activation functions (g_l)

- Sometimes use notation σ as in $\sigma(Wh + b)$

Activation functions (g_l)

- Sometimes use notation σ as in $\sigma(Wh + b)$
- Historically $g(z)$ has been the sigmoid or or hyperbolic tangent (tanh)

Activation functions (g_l)

- Sometimes use notation σ as in $\sigma(Wh + b)$
- Historically $g(z)$ has been the sigmoid or or hyperbolic tangent (tanh)
- Now, usually variants of Rectified linear unit (ReLU)
  - $g(z) = \max 0, z$
  - Easier to estimate with SGD
  - Easier for deep models
- Last activation is the output function g_L, usually a softmax (if classification)

$$f(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{J} e^{z_j}}$$

Activation functions (g_l)

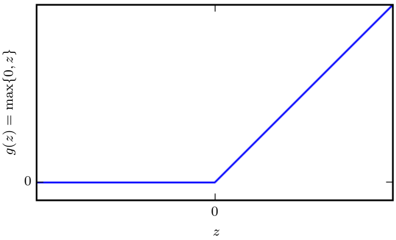


Figure: Rectified Linear Unit (Goodfellow et al, 2017, Fig. 6.3)

Universal Approximation Theorem

"A feed-forward neural network with a linear output layer and at least one hidden layer with any 'squashing' activation function can approximate any Borel measurable function from one finite-dimensional space to another with any desired non-zero amount of error, provided that the network is given enough hidden units." (Goodfellow et al. 2017, p. 198)

- Also holds for ReLU
- No garantuee we can learn the network
- No garantuee that it will generalize
- No indication of how large the network need to be

Hyper-parameters in feed-forward networks

- The number of layers
- The number of neurons
- Activation functions
- The type of layers (CNN, MaxPooling, Multi-head attention)

How to choose parameters

- Trial and error on validation sets
- Art rather than science
- Specialized approaches (Bayesian Optimization)

UPPSALA UNIVERSITET

How to choose parameters

- Trial and error on validation sets
- Art rather than science
- Specialized approaches (Bayesian Optimization)
- Grid search (combinatorical explosion)
  - Really bad with many parameters with less effects
  - If we have 5 irrelevant parameters we try 3 values for: 125 training per relevant run
  - Instead use...

How to choose parameters

- Trial and error on validation sets
- Art rather than science
- Specialized approaches (Bayesian Optimization)
- Grid search (combinatorical explosion)
  - Really bad with many parameters with less effects
  - If we have 5 irrelevant parameters we try 3 values for: 125 training per relevant run
  - Instead use...
- Random search

Grid search vs. Random Search

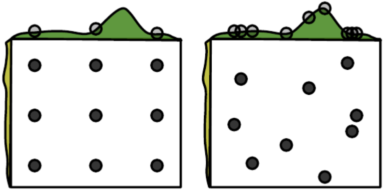


Figure: Grid search and random search (Goodfellow et al, 2017, Fig. 11.2)

Optimization of Neural Networks

- Difficult problem
- Many local minima (weight space symmetry)
- Platueas and sadel points
  - Gradient is small - but not a minimum or maximum
  - Sadel points increases with the number of dimensions (?)
  - Large areas with small change in cost function

Optimization of Neural Networks II

- A lot of parameters (W and b)
- Usually a lot of data

Optimization of Neural Networks II

- A lot of parameters (W and b)
- Usually a lot of data
- Stochastic Gradient Descent, commonly
  - Adam

Optimization of Neural Networks II

- A lot of parameters (W and b)
- Usually a lot of data
- Stochastic Gradient Descent, commonly
  - Adam
- To compute gradients: backpropagation
  - Chain-rule for derivatives

UPPSALA UNIVERSITET

Initial values

- We need to have starting values for SGD - non-trivial
- Bad initial values might
  - Bad convergence (local optimum)
  - Numerical problems
- We want to break symmetry between layers
- Initialization can be seen as a hyperparameter
- Good practice
  - Initialize values randomly close to zero (uniform or normal)

Neural Networks in Practice: TensorFlow and Keras

- Tensorflow
  - Framework for large-scale machine learning and Neural Networks
  - Developed by Google
  - Computational graphs
  - Handles:
    - Computing gradients for Neural Networks
    - Enable simple use of graphical processing units (GPU) and Tensor processing Units (TPU)
    - Used in both research and production
- Keras
  - Syntax for 'building' Neural Networks
  - Platform independent (ish)

Regularization of Neural Networks

- Reduce traing error but improve test/validation error

Regularization of Neural Networks

- Reduce traing error but improve test/validation error
- Neural Networks are extremely flexible / high model capacity

Regularization of Neural Networks

- Reduce traing error but improve test/validation error
- Neural Networks are extremely flexible / high model capacity
- Regularization is crucial for good generalizability of NN

Regularization of Neural Networks

- Reduce traing error but improve test/validation error
- Neural Networks are extremely flexible / high model capacity
- Regularization is crucial for good generalizability of NN

Weight decay / Norm penalty

- Let
$$\tilde{J}(W, b) = J(W, b) + \alpha\Omega(W),$$
where $J(W, b)$ is the cost function and $\alpha\Omega(W)$ is the penalty for the weight matrices.
- α is the strength of the penalty.

UPPSALA UNIVERSITET

Weight decay / Norm penalty

- Let

$$\Omega_1(W) = \sum_i \sum_j |w|_{i,j}\,,$$

and

$$\Omega_2(W) = \sum_i \sum_j w_{i,j}^2\,,$$

be the L_1 and L_2 regularization respectively.

- We can then get the cost function

$$\tilde{J}(W, b) = J(W, b) + \sum_l \alpha_l \Omega_2(W_l)\,,$$

UPPSALA UNIVERSITET

Weight decay / Norm penalty

- Lets define the cost function as

$$\begin{aligned}\tilde{J}(w) &= J(w) + \alpha\Omega_2(w) \\ &= J(w) + \alpha w^T w\end{aligned}$$

- Then the gradient update becomes

$$\nabla_w \tilde{J}(w) = \nabla_w J(w) + 2\alpha w$$

- To update our weights with gradient descent

$$\begin{aligned}w &\leftarrow w - \epsilon(\nabla_w J(w) + 2\alpha w) \\ w &\leftarrow (1 - 2\alpha\epsilon)w - \epsilon\nabla_w J(w)\end{aligned}$$

Weight decay / Norm penalty

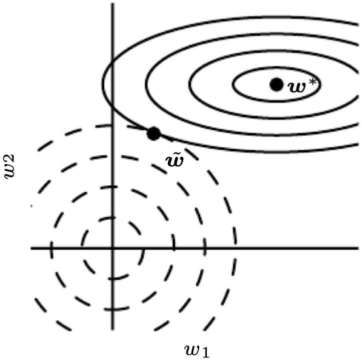


Figure: L_2 regularization (Goodfellow et al, 2017, Fig. 7.1)

Early Stopping

- Stop optimization early based on validation error
- Rerun to that number of epochs (hyperparameter)
- Can be shown to be quivalent (under strict assumptions) to L_2 regularization

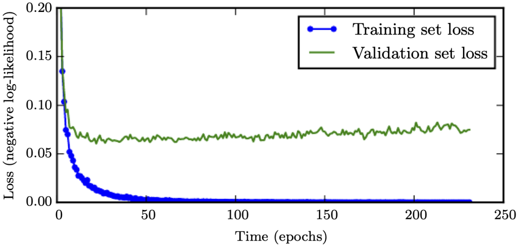


Figure: Early Stopping (Goodfellow et al, 2017, Fig. 7.3)

Early Stopping

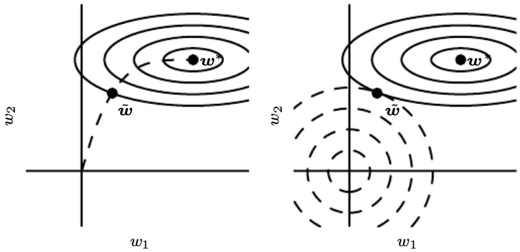


Figure: Early Stopping (Goodfellow et al, 2017, Fig. 7.4)

UPPSALA UNIVERSITET

Dropout

- In each iteration:
  - Sample an indicator I_i for each node i
  - Set the value h_i to 0 with probability p
- The dropout probability is typically 0.8 for input nodes and 0.5 for hidden nodes
- Forces the network to
  - not rely on individual nodes
  - spread out the weights over more nodes
- Can be seen as an ensamble method

Dropout

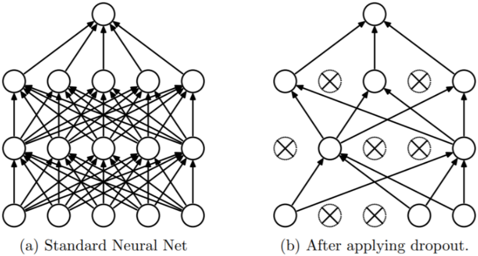


Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Figure: Dropout (Srivastava et al, 2014)

Other regularization techniques

- In CNN: Dataset augmentation
- Get more data...