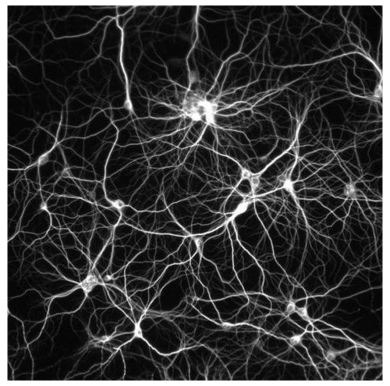


Neural Network (Basic) Cheat Sheet

(source: Deep Learning - Goodfellow, Ian)

V2021.01.01

(Dr Yan Xu)

Core Assumption

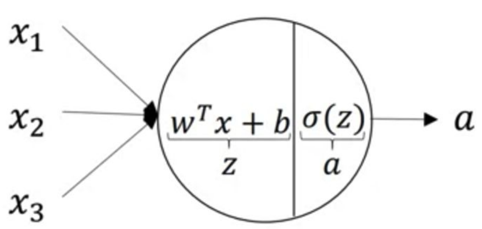
The data (or the target problem) was generated by the composition of factors (or features), potentially at multiple levels in a hierarchy.

Modelling Mindset

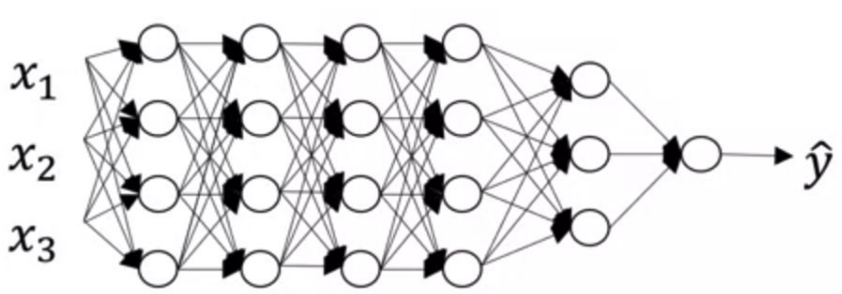
- ➔ Models with high representation capability
- ➔ Appropriate regularization
- ➔ Big training data

Basic Concepts

- ➔ Neural:



- ➔ Activation Function:
  - Sigmoid
  - Tanh
  - ReLU (most popular)
  - Leaky ReLU
- ➔ Neural Network:



- ➔ Output Units:
  - Linear Units (Gaussian Distribution)
  - Sigmoid Units (Bernoulli Distribution)
  - Softmax Units (Multinoulli Distribution)

Forward and Backward Propagation

➔ Forward Propagation:

Require: Network depth, l
Require: $\boldsymbol{W}^{(i)}, i \in \{1, \ldots, l\}$, the weight matrices of the model
Require: $\boldsymbol{b}^{(i)}, i \in \{1, \ldots, l\}$, the bias parameters of the model
Require: $\boldsymbol{x}$, the input to process
Require: $\boldsymbol{y}$, the target output
$\boldsymbol{h}^{(0)} = \boldsymbol{x}$
for $k = 1, \ldots, l$ **do**
$\boldsymbol{a}^{(k)} = \boldsymbol{b}^{(k)} + \boldsymbol{W}^{(k)} \boldsymbol{h}^{(k-1)}$
$\boldsymbol{h}^{(k)} = f(\boldsymbol{a}^{(k)})$
end for
$\hat{\boldsymbol{y}} = \boldsymbol{h}^{(l)}$
$J = L(\hat{\boldsymbol{y}}, \boldsymbol{y}) + \lambda \Omega(\theta)$

(note: the symbols (***a, h***) are not consistent with other figures)

➔ Backward Propagation:

After the forward computation, compute the gradient on the output layer:
$\boldsymbol{g} \leftarrow \nabla_{\hat{\boldsymbol{y}}} J = \nabla_{\hat{\boldsymbol{y}}} L(\hat{\boldsymbol{y}}, y)$
for $k = l, l-1, \ldots, 1$ **do**
Convert the gradient on the layer's output into a gradient into the pre-nonlinearity activation (element-wise multiplication if f is element-wise):
$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{a}^{(k)}} J = \boldsymbol{g} \odot f'(\boldsymbol{a}^{(k)})$
Compute gradients on weights and biases (including the regularization term, where needed):
$\nabla_{\boldsymbol{b}^{(k)}} J = \boldsymbol{g} + \lambda \nabla_{\boldsymbol{b}^{(k)}} \Omega(\theta)$
$\nabla_{\boldsymbol{W}^{(k)}} J = \boldsymbol{g}\, \boldsymbol{h}^{(k-1)\top} + \lambda \nabla_{\boldsymbol{W}^{(k)}} \Omega(\theta)$
Propagate the gradients w.r.t. the next lower-level hidden layer's activations:
$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{h}^{(k-1)}} J = \boldsymbol{W}^{(k)\top} \boldsymbol{g}$
end for

(source: Deep Learning - Goodfellow, Ian)

Regularization

➔ Parameter Norm
- L1 and L2 norm
- Typically, biases are not regularized

➔ Dataset Augmentation
- random rotation, shift, reflection
- random noise

➔ EarlyStopping
- must have validation set

➔ Dropout (How it works?)
- Motivation: "***Bagging + Parameter Sharing***"
- used after the activation function
- *Training*: random selection & weight adjustment
- *Prediction*: standard forward propagation

➔ Dropout (some notes)
- could be used to estimate prediction uncertainty
- could be used in the input layer and hidden layers
- typically performs better than Norm
- typically used on fully connected layers (e.g. p = 0.5)

Optimisation

➔ Local Minima
- the cost function of neural networks is non-convex
- however, local minima seems not a major problem

➔ Second-order (or Newton Method)
- remain difficult to scale to large neural networks
- gradient-based method is still the mainstream

➔ Mini-batch Optimisation
- typical values: 32 - 256

➔ Stochastic Gradient Descent

➔ Batch Normalization
- Training: mini-batch normalization in hidden layers
- Prediction: using learned mean & variance

➔ Note: Batch Normalization is not a regularization method

➔ Optimizer: RMSProp

➔ Optimizer: Adam (most popular)

➔ Learning Rate Decay

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
Initialize 1st and 2nd moment variables $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{0}$
Initialize time step $t = 0$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
$t \leftarrow t + 1$
Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1) \boldsymbol{g}$
Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2) \boldsymbol{g} \odot \boldsymbol{g}$
Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$
Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$
Compute update: $\Delta \boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
end while

Neural Network Tuning

- ➔ Major Hyperparameters:
  - Optimizer (***learning rate***, momentum, decay, others)
  - Network layers and ***hidden units***
  - Regularization (***L2, Dropout***)
  - Mini-batch Size
- ➔ Need to code with hyperparameters as inputs
- ➔ Cross Validation could be very slow
- ➔ Grid Search vs Random Sampling (*preferred*)
- ➔ Batch Normalization (not working for all networks)
- ➔ [Code Example]