

nn-5Ws

Neural Networks: What, Why, How, Where, and When? [browser playground](#)

What does it do?

It learns best weights (parameters, variables) of a mathematical function.

- what is that function?

Why does it learn weights?

So we can use the weights with that function to get (predict) the solution to any input variables.

How

Where do the weights come from?

From the network (model) design.

When does the learning take place?

During (feed) input to the network.

gradient \equiv slope

What slope

Simple algorithm with description

```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
# src: cs231n.github.io
```

TODO

☐ Complete [Michael Nielsen guide](#)

References:

- T. Rashid, [A gentle intro to NN](#)
- Conceptual intro: [Grokking Deep Learning](#) and NN in 13 lines [iamtrask](#)
- nn in [9 lines](#)
- [Gradient Descent](#)

Practice:

- [simple-neural-network.ipynb](#)
- [machine-learning-review](#)

ml-review

Also see: [ML cheatsheets](#)

Prepared prior to Startup.ML fellowship interview on 9.31.2016

Machine learning

- ☒ *learning types*: [supervised](#) vs. [unsupervised](#) vs. [reinforcement](#)
- ☒ What models learn: [generative vs. discriminative algorithms](#)
- ☒ *[problem types][r2]*: [regression](#) vs. [classification](#) vs. [clustering](#)
- ☒ *[training techniques][r1]* data split (train/evaluate/test), k-fold.
- ☒ details implementations on a specific algorithm e.g. linear regression, [k-nearest](#), naive bayes, perceptron, decision trees
- ☒ overfitting: reduce features or regularization
- ☒ Relation between [statistics and machine learning](#) and normal distribution.
- ☒ [ensemble method](#)
- ☒ [Cost functions](#): [MSE and cross-entropy](#)

Adv. ML (deep learning)

- ☒ neural network and their applications
- ☒ [logistic regression vs. neural network](#)
- ☐ networks architecture: RNN/CNN/LSTM
- ☒ backpropagation algorithm / Gradient descent

Advanced (NLP and text learning):

- ☒ Representation: BoW, TF-IDF, word embedding and algorithms behind it.

Software

- ☒ Summary on most used packages and their usages; SE and programming paradigms

Experience

- ☒ summary (with detail descriptions) on what I have done in the past, which kaggle competitions: problem/solutions/methods
- ☒ explain about my ability/enjoyment in data munging prior to the task e.g. [exploratory, visualization, and transforming](#)
- ☒ summary on my current school research and its relevant technology
- ☐ most of my ML work was involving data wrangling, transforming, preprocessing

[r1]: file:///Users/Aziz/resources/python/pym/Python%20Machine%20Learning.pdf "Python Machine Learning p.12"

[r2]: <https://www.quora.com/What-is-the-main-difference-between-classification-problems-and-regression-problems-in-machine-learning> "difference: regression vs. classification"

[r3]: <http://webpage.pace.edu/aa10212w/course/CS855/3-to-2-class-problem-with-classifier.html> "Dichotomy Model 3-to-2 classes problem"

[r4]: <http://webpage.pace.edu/aa10212w/course/CS855/bayes-classifier.html> "Bayes Density Classifier"

SUMMARY

What is ML?

The ability of computers to learn on their own without being explicitly programmed or told.
i.e. using various learning algorithms.

Generative vs. Discriminative models

- A generative algorithm models how the data was generated in order to categorize a signal. It asks the question: based on my generation assumptions, which category is most likely to generate this signal?
- A discriminative algorithm does not care about how the data was generated, it simply categorizes a given signal.

"Generative classifiers learn a model of the joint probability, $p(x, y)$, of the inputs x and the label y , and make their predictions by using Bayes rules to calculate $p(y|x)$, and then picking the most likely label y .

Discriminative classifiers model the posterior $p(y|x)$ directly, or learn a direct map from inputs x to the class labels." (Ng et al. 2002)

see: [cs229 notes](#)

Generative	Discriminative
models how data was generated	categorizes "classifies" data
$p(x, y)$	$p(y, x)$
learns the joint probability distribution	learns the conditional "posterior" probability distribution
models the distribution of individual classes	learns the boundary between classes
e.g. NB, RBM, LDA, HMM	e.g. NNs, logit, linReg, SVM, RandFor,
learn a model of the joint prob $p(x, y)$ and make predictions by using Bayes rules to calculate $p(x y)$, then pick most likely class	model posterior prob directly, or learn a direct map from x to the class label

Supervised models

the general theme of the learning algorithm:

- set a hypothesis (a function with params)
- initialize the model params with random values
- using training samples: compare our model's predicted value with the actual value; keeping the difference "residual" as the error
- use the error function (sum squared errors) as a guide for updating "tuning" the model's params toward predicting closer values to the actual ones
 - the updating method varies depending on the problem and the used model (e.g gradient descent, SGD, backprop .. etc)

regression analysis

- statistically estimates the relationship between variables (i.e. independent 'input' and dependent 'output')

e.g. **linear regression** (univariate: $h(x) = \theta_0 + \theta_1 x$ or multivariate: $h(x) = \theta^T x$) and **non-linear regression** (polynomial regression / curve line)

linear regression: assumption: relationship between input variable and the outcome variable is linear.

classification

- autoamtically identify the category (class) of a new observation based on previous observations (statistically).

- e.g. **logit** $h(x) = \frac{1}{1+e^{-\theta^T x}}$, **perceptron**, **Naive Bayes** $p(c|x) = \frac{p(x|c)p(c)}{p(x)}$
-

logistic regression (for classification):

- def:
 - logit vs. linear regression: in logit we **use sigmoid** to restrict the output $0 \leq h_{\theta}(x) \leq 1$; thereafter map $0.5 < 0 \rightarrow 0$ and $0.5 \geq 1 \rightarrow 1$
 - its hypothesis is the same as linear regression but we need to restrict output to the interval $(0, 1)$, thus we plug our hypothesis in the sigmoid function.
 - output is $\{0, 1\}$ (in case of binary) instead of the continuous \hat{y}
 - logistic from its 'logistic' sigmoid function i.e. $g(z) = \frac{1}{1+e^{-z}}$ and z is real number i.e. $z = \theta^T x$
 - so: $h_{\theta}(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$
 - $h_{\theta}(x) = p(y = 1|x; \theta) = 1 - p(y = 0|x; \theta)$
 - e.g. $h_{\theta}(x) = 0.35$ means 35% probability the output is 1, and 65% the output is 0
 - in short: logistic regression = linear regression + sigmoid function
 - decision boundary:
 - $\theta^T x \geq 0 \equiv h_{\theta}(x) \geq 0.5 \rightarrow y = 1$
 - $\theta^T x < 0 \equiv h_{\theta}(x) < 0.5 \rightarrow y = 0$
-

Overfitting problem:

- overfitting occurs while minimizing the error functions e.g. $\sum_{i=1}^N (y_i - (Wx_i + b))^2$

Two main **options** to address overfitting:

- Reduce features
- Regularization** (reduce the parameters θ_j)

To prevent fitting the training data very well (i.e. $J(\theta) \approx 0$), we add a regularization term (to penalize the loss function) e.g. $\sum_{i=1}^N (y_i - (Wx_i + b)) + \lambda(W^2)$

- cross-validation is one way to tune λ term.

Cross-validation:

- a partioning technique used during training to assess how well the model will generalize.

bias and variance:

- high variance** means **overfitting**, while **high bias** means **underfitting**.
- Bias in nn is analogous to the intercept in a regression model.

Ensemble methods:

- Two types:
 - **boosting**: combining multi (weak) classifiers into one strong classifier.
 - **bagging**: to generate new data samples from existing ones, if not enough.
- e.g. random forest

Neural network *more details*:

- def: NNs
- NNs are used to estimate/approximate a function (more precisely the parameter of our model).
- e.g 1) a three layer network $f(x) = W_3 \max(0, W_2 \max(0, W_1 x))$
- e.g 2) 2 input 3 hidden 1 output:
 - $\text{sigmoid}(\text{sigmoid}(XW^{(1)})W^{(2)})$
 - in details:

$$z^{(2)} = X \cdot W^{(1)}$$

$$a^{(2)} = \text{sigmoid}(z^{(2)})$$

$$z^{(3)} = a^{(2)} W^{(2)}$$

$$a^{(3)} = \text{sigmoid}(z^{(3)}) = \hat{y}$$

why use neural networks over logit ?

To tackle the non-linearity problems; where logit uses lots of features which leads to a model that is complex and prone to overfitting.

metaphor: essentially a NN is like a logit that perform multiple processing (computations) of the the input through the hidden units/layers.

A.K.A think of logit as a one layer NN

A.K.A logit \approx perceptron *see*

> Gradient Descent <https://quiver-note-url/9C7B40AC-87EE-4047-ACDD-F4792EA406DC>
 {more\;details}}\$ > - gradient \equiv derivative - def: move against the gradient to find the minimum of a function.

Naive Bayes

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)} \rightarrow \text{Posterior} = \frac{\text{likelihood} \cdot \text{Prior}}{\text{Evidence} \text{ "margin"}}$$

> **Stats and Prob** > - pmf (Prob. Mass function) \rightarrow prob. distribution for discrete random variables. - pdf (Prob. Density function) \rightarrow prob. distribution for continuous random variables. (aka gaussian 'normal' distribution) - a density function gives a rough picture of the distribution from which a random variable is drawn. - univariate: \rightarrow one random variable - multivariate: \rightarrow more than one random variable

definitions:

- **mean**: a measure of the central tendency
- **variance**: take differences from the mean, square each, and then average them. $\sigma^2 = \frac{1}{N} \sum (X - \mu)^2$
- **standard deviation**: the average of the squared differences from the Mean. $\sigma = \sqrt{\sigma^2}$

> **highlights** > When dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a Gaussian distribution.

Q/A

How machine learning algorithms are different ? [see answer](#)

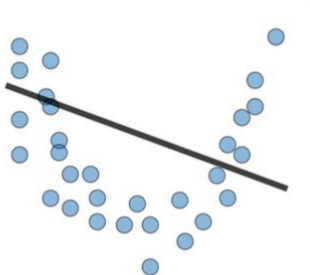
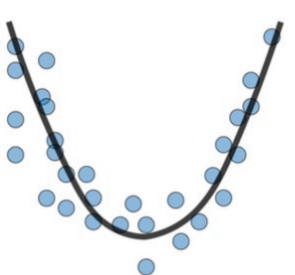
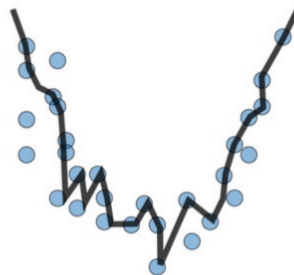
What is (the difference between) linear vs. non-linear ?

What is variance vs. bias ?

□ **Bias** – The bias of a model is the difference between the expected prediction and the correct model that we try to predict for given data points.

□ **Variance** – The variance of a model is the variability of the model prediction for given data points.

□ **Bias/variance tradeoff** – The simpler the model, the higher the bias, and the more complex the model, the higher the variance.

	Underfitting	Just right	Overfitting
Symptoms	<ul style="list-style-type: none">- High training error- Training error close to test error- High bias	<ul style="list-style-type: none">- Training error slightly lower than test error	<ul style="list-style-type: none">- Low training error- Training error much lower than test error- High variance
Regression			

What is linear regression ?

What is the difference between linear regression and logistic regression?

Why we need intercept/bias in linReg, logit, and neural networks ?

What is the relation between statistics and machine learning ? [see answer](#)

What is Maximum Likelihood Estimation (MLE) ?

▮ a technique for estimating a param of a distribution.

What is the rule of regularization ?

▮ reduce model complexity → solve overfitting problem.

ml-metrics

- **Precision** is the **correct fraction** of the **retrieved docs** $P = \frac{TP}{TP+FP} = \frac{\text{relevant} \cap \text{retrieved}}{\text{retrieved}}$ i.e. how many retrieved docs is correct out of all **retrieved**?
- **Recall** is the **correct fraction** of the **true docs** $R = \frac{TP}{TP+FN} = \frac{\text{relevant} \cap \text{retrieved}}{\text{relevant}}$ i.e. how many retrieved docs is correct out of all **true**?
- **F-measure** (also F-score, F1) is the **harmonic mean** of **P** and **R** which is $F = 2 \frac{P \cdot R}{P+R}$
- **P@n** is **Precision** considering only the **topmost n** retrieved docs instead of all retrieved.
- **R@n** is **Recall** considering only the **topmost n** retrieved docs instead of all retrieved.

- Mean reciprocal rank

$$MMR = \frac{1}{|Q|} \sum_{i=1}^Q \frac{1}{rank_i} .$$

ml-math

machine learning math

Algebra

1. Norms (regularization)

General formula:

$$\|x\|_p = (\sum |x_i|^p)^{\frac{1}{p}}$$

popular norms L1 (Lasso) and L2 (Ridge)

$$L1 = \|x\|_1 \Rightarrow \sum |y_i - \hat{y}| \Rightarrow \text{least absolute difference}$$

$$L2 = \|x\|^2 \Rightarrow \sum (y_i - \hat{y})^2 \Rightarrow \text{least square difference}$$

```
# in Python
np.linalg.norm(x, ord=1) # L1
np.linalg.norm(x)        # L2
```

Vectors

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\| \cdot \|B\|} \text{ also: } A \cdot B = \|A\| \cdot \|B\| \cos \theta$$

Matrices

symmetric, if $A^T == A$

matrix factorization (decomposition)

Theorem: Let A be a m -by- n matrix of rank r .

Then there is a m -by- r matrix X and r -by- n matrix Y such that

$$A = XY$$

This is how low rank matrices can be compressed.

Instead of storing A we store X and Y

$$m \cdot n > mr + nr \\ r(m+n)$$

Why Matrix Decomposition?

to reduce the size (number) of stored values (items) in memory.

e.g. if $A = [200\text{-by-}100]$ and $\text{rank}(A) = 4$ then (instead of storing \Rightarrow **20,000** number):

we can store only $X: [200\text{-by-}4] + Y: [4\text{-by-}100] \Rightarrow$ **1200** number

To find the rank: `np.linalg.matrix_rank(A)`

Dimensionality Reduction?

What? reduce the number of dimensions in the representational vectors.

Why? Computational time

Probability

Distributions

Univariate: $N(x; \mu, \sigma^2) =$

$$\sqrt{\frac{1}{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2} (x - \mu)^2\right)$$

Multivariate: $N(x; \mu, \Sigma) =$

$$\sqrt{\frac{1}{(2\pi)^2 \det(\Sigma)}} \exp\left(-\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu)\right)$$

calculus

chain rule

$$y = f(u) \textbf{ and } u = g(x)$$

$$\therefore y = f(g(x))$$

$$\frac{dy}{dx} = \frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \cdot \frac{\partial u}{\partial x}$$

Example:

$$y = f(u) = 5u^4 \textbf{ and } u = g(x) = x^3 + 7$$

$$\frac{\partial y}{\partial x} = \frac{\partial}{\partial x} 5(\underbrace{x^3 + 7}_u)^4$$

$$\frac{\partial y}{\partial u} = 20u^3 \textbf{ and } \frac{\partial u}{\partial x} = 3x^2$$

$$\frac{\partial y}{\partial x} = 20(x^3 + 7)^3 \cdot 3x^2$$

ml-concepts

cost function

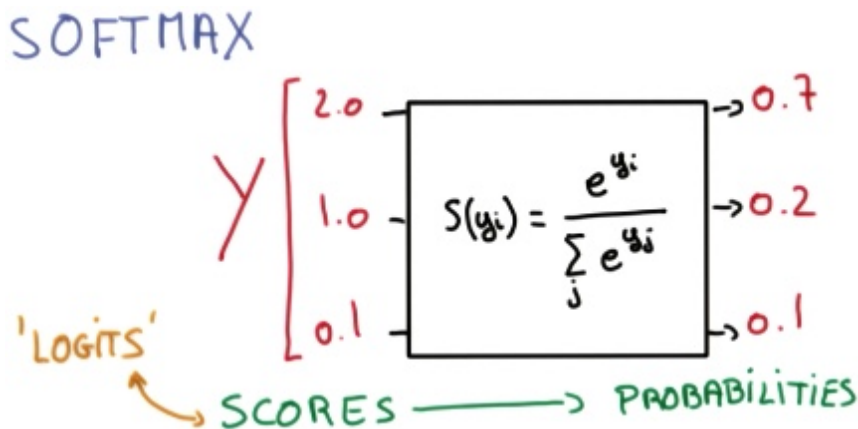
Any cost function is expected to possess two properties. **First**, it should be a non-negative always, regardless of the predicted output. **Second**, it should be close to zero when the model's output is close to the desired output for all training inputs. (M. Nielsen, 2015 ch3)

e.g. MSE, cross-entropy.

softmax classifier:

softmax is a variant of (or the same as) cross-entropy.

- softmax job is to change predicted scores to probabilities:



see [Karpathy's class](#)

gradient descent:

What is GD? An algorithm that iteratively finds the minimum (or maximum "gradient ascent") of a function.

How? Guided by the slope (derivative of the function) of a current point "slope of the tangent line".

Why the derivative? it defines the relationship between each parameter (weight) and how much we missed (error). (in other words, it tells us how much changing this param contributes to the error).

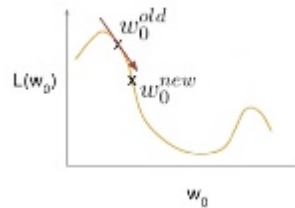
What's the error? it's *loss (cost) function*, which measures our prediction's (hypothesis function) accuracy. prediction - real

non-convex cost function (i.e. its curve does not always go in the same direction) can cause gd to get stuck in a local minimum.

- To choose good weights, iteratively update each weight in a way that decreases L

$$w_0^{new} = w_0^{old} - \alpha \frac{\partial}{\partial w_0} L(w_0, w_1, \dots)$$

learning rate



$$\mathcal{E}(i) = \frac{1}{2} \sum_{m=1}^{K_i} (e_m(i))^2 = \frac{1}{2} \sum_{m=1}^{K_i} (\hat{y}_m(i) - y_m(i))^2$$

- Gradient descent starts with an initial guess at the weights over all layers of the network.
- We then use these weights to compute the network output $\hat{y}(i)$ for each input vector $\mathbf{x}(i)$ in the training data.
- This allows us to calculate the error $\mathcal{E}(i)$ for each of these inputs.
- Then, in order to minimize this error, we incrementally update the weights in the negative gradient direction:

$$\mathbf{w}_j^r(\text{new}) = \mathbf{w}_j^r(\text{old}) - \mu \frac{\partial \mathcal{J}}{\partial \mathbf{w}_j^r} = \mathbf{w}_j^r(\text{old}) - \mu \sum_{i=1}^N \frac{\partial \mathcal{E}(i)}{\partial \mathbf{w}_j^r}$$



see [blog](#)

Stochastic Gradient Descent [see video](#)

SGD (i.e. use one training example at a time (instead of full dataset) to compute/update params gradient) is a solution to find the global minimum regardless of the function type (i.e. even with non-convex ones).

Algorithm 1 Stochastic Gradient descent.

Given a starting point $\theta \in \text{dom} g$

Given a step size $\epsilon \in \mathbb{R}^+$

repeat

1. Sample a minibatch of m examples $\{(x_1, y_1), \dots, (x_m, y_m)\}$ from training data
2. Estimate the gradient $\nabla_{\theta} g(\theta) \approx \nabla_{\theta} \left[\frac{1}{m} \sum_{i=1}^m L(f_{\theta}(x_i), y_i) + R(f_{\theta}) \right]$ with backpropagation
3. Compute the update direction: $\Delta \theta := -\epsilon \nabla_{\theta} g(\theta)$
4. Perform a parameter update: $\theta := \theta + \Delta \theta$

until convergence.

Algorithm 2.1 Online stochastic gradient descent training.*Input:*

- Function $f(\mathbf{x}; \Theta)$ parameterized with parameters Θ .
- Training set of inputs $\mathbf{x}_1, \dots, \mathbf{x}_n$ and desired outputs y_1, \dots, y_n .
- Loss function L .

```

1: while stopping criteria not met do
2:   Sample a training example  $\mathbf{x}_i, y_i$ 
3:   Compute the loss  $L(f(\mathbf{x}_i; \Theta), y_i)$ 
4:    $\hat{\mathbf{g}} \leftarrow$  gradients of  $L(f(\mathbf{x}_i; \Theta), y_i)$  w.r.t  $\Theta$ 
5:    $\Theta \leftarrow \Theta - \eta_t \hat{\mathbf{g}}$ 
6: return  $\Theta$ 

```

Algorithm 2.2 Minibatch stochastic gradient descent training.*Input:*

- Function $f(\mathbf{x}; \Theta)$ parameterized with parameters Θ .
- Training set of inputs $\mathbf{x}_1, \dots, \mathbf{x}_n$ and desired outputs y_1, \dots, y_n .
- Loss function L .

```

1: while stopping criteria not met do
2:   Sample a minibatch of  $m$  examples  $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)\}$ 
3:    $\hat{\mathbf{g}} \leftarrow 0$ 
4:   for  $i = 1$  to  $m$  do
5:     Compute the loss  $L(f(\mathbf{x}_i; \Theta), y_i)$ 
6:      $\hat{\mathbf{g}} \leftarrow \hat{\mathbf{g}} +$  gradients of  $\frac{1}{m}L(f(\mathbf{x}_i; \Theta), y_i)$  w.r.t  $\Theta$ 
7:    $\Theta \leftarrow \Theta - \eta_t \hat{\mathbf{g}}$ 
8: return  $\Theta$ 

```

([Goldberg, 2017](#)) book

Backpropagation

What is backprop?

- It is a method to calculate the error contributed from each gate (e.g. neuron, step, operation, or layer ...) in the network.
So, that an optimization algorithm can use it to update the params (i.e. weights) of each gate in the network (with respect to its corresponding error).
- It is the **output_error** (L_{error}) *multiplied* by the **derivative of the function** (∇L) of that output.

In other words, in two steps:

1. Calculate **backprop value**:
... **multiply** the *error of the output* (L_{error}) by the *derivative of the function* of that output ($\frac{dL}{dW}$) i.e.

$$backprop = L_{error} \cdot \nabla L$$

2. **Update** the input's weight (syn_i) with respect to the calculated value:
... **multiply** the input (x_i) by the value of *backprop*. i.e.

$$syn_i = x_i \cdot backprop$$

this process is called "backpropagation".

"At the heart of backpropagation is an expression for the partial derivative $\frac{\partial C}{\partial w}$ of the cost function C with respect to any weight w (or bias b) in the network. The expression tells us how quickly the cost changes when we change the weights and biases." (Nielsen, 2015)

references:

[iamtrask 13 lines](#)

<http://cs231n.github.io/optimization-2/>

<http://neuralnetworksanddeeplearning.com/chap2.html>

<http://colah.github.io/posts/2015-08-Backprop/>

<http://vinhkhuc.github.io/2015/03/29/backpropagation.html>

Repeat until convergence

1. **Initialization**
 - ▣ Initialize all weights with small random values
2. **Forward Pass**
 - ▣ For each input vector, run the network in the forward direction, calculating:
$$v_j^r(i) = (w_j^r)^T y^{r-1}(i); \quad y_j^r(i) = F(v_j^r(i))$$
and finally
$$\epsilon(i) = \frac{1}{2} \sum_{n=1}^{k_i} (e_n(i))^2 = \frac{1}{2} \sum_{n=1}^{k_i} (\hat{y}_n(i) - y_n(i))^2$$
3. **Backward Pass**
 - ▣ Starting with the output layer, use our inductive formula to compute the $\delta_j^{r-1}(i)$:
 - ▣ Output Layer (Base Case): $\delta_j^L(i) = e_j^L(i) F'(v_j^L(i))$
 - ▣ Hidden Layers (Inductive Case): $\delta_j^{r-1}(i) = F'(v_j^{r-1}(i)) \sum_{k=1}^{k_r} \delta_k^r(i) w_{kj}^r$
4. **Update Weights**

$$w_j^r(\text{new}) = w_j^r(\text{old}) - \mu \sum_{i=1}^N \frac{\partial \epsilon(i)}{\partial w_j^r} \quad \text{where} \quad \frac{\partial \epsilon(i)}{\partial w_j^r} = \delta_j^r(i) y^{r-1}(i)$$

[source](#)

The Back-Propagation Algorithm

To train a neural network to perform some task, we must adjust the weights of each unit in such a way that the error between the desired output and the actual output is reduced. This process requires that the neural network compute the error derivative of the weights (EW). In other words, it must calculate how the error changes as each weight is increased or decreased slightly. The back-propagation algorithm is the most widely used method for determining the EW.

To implement the back-propagation algorithm, we must first describe a neural network in mathematical terms. Assume that unit j is a typical unit in the output layer and unit i is a typical unit in the previous layer. A unit in the output layer determines its activity by following a two-step procedure. First, it computes the total weighted input x_j , using the formula

$$x_j = \sum_i y_i w_{ij},$$

where y_i is the activity level of the i th unit in the previous layer and w_{ij} is the weight of the connection between the i th and j th unit.

Next, the unit calculates the activity y_j using some function of the total weighted input. Typically, we use the sigmoid function:

$$y_j = \frac{1}{1 + e^{-x_j}}.$$

Once the activities of all the output units have been determined, the network computes the error \mathcal{E} , which is defined by the expression

$$\mathcal{E} = \frac{1}{2} \sum_j (y_j - d_j)^2,$$

where y_j is the activity level of the j th unit in the top layer and d_j is the desired output of the j th unit.

The back-propagation algorithm consists of four steps:

1. Compute how fast the error changes as the activity of an output unit is changed. This error derivative (EA) is

the difference between the actual and the desired activity.

$$EA_j = \frac{\partial \mathcal{E}}{\partial y_j} = y_j - d_j$$

2. Compute how fast the error changes as the total input received by an output unit is changed. This quantity (EI) is the answer from step 1 multiplied by the rate at which the output of a unit changes as its total input is changed.

$$EI_j = \frac{\partial \mathcal{E}}{\partial x_j} = \frac{\partial \mathcal{E}}{\partial y_j} \frac{dy_j}{dx_j} = EA_j y_j (1 - y_j)$$

3. Compute how fast the error changes as a weight on the connection into an output unit is changed. This quantity (EW) is the answer from step 2 multiplied by the activity level of the unit from which the connection emanates.

$$EW_{ij} = \frac{\partial \mathcal{E}}{\partial w_{ij}} = \frac{\partial \mathcal{E}}{\partial x_j} \frac{\partial x_j}{\partial w_{ij}} = EI_j y_i$$

4. Compute how fast the error changes as the activity of a unit in the previous layer is changed. This crucial step allows back propagation to be applied to multilayer networks. When the activity of a unit in the previous layer changes, it affects the activities of all the output units to which it is connected. So to compute the overall effect on the error, we add together all these separate effects on output units. But each effect is simple to calculate. It is the answer in step 2 multiplied by the weight on the connection to that output unit.

$$EA_i = \frac{\partial \mathcal{E}}{\partial y_i} = \sum_j \frac{\partial \mathcal{E}}{\partial x_j} \frac{\partial x_j}{\partial y_i} = \sum_j EI_j w_{ij}$$

By using steps 2 and 4, we can convert the EAs of one layer of units into EAs for the previous layer. This procedure can be repeated to get the EAs for as many previous layers as desired. Once we know the EA of a unit, we can use steps 2 and 3 to compute the EWs on its incoming connections.

Cross entropy (Information theory view)

The *cross-entropy* between a 'true' distribution \mathbf{p} and an estimated distribution \mathbf{q} is defined as:

$$H(p, q) = - \sum p(x) \log q(x)$$

In machine learning, we use the equivalent form:

$$CE(y, \hat{y}) = - \sum y_i \log(\hat{y}_i)$$

where y is the one-hot label vector, and \hat{y} is the predicted probability vector for all classes.

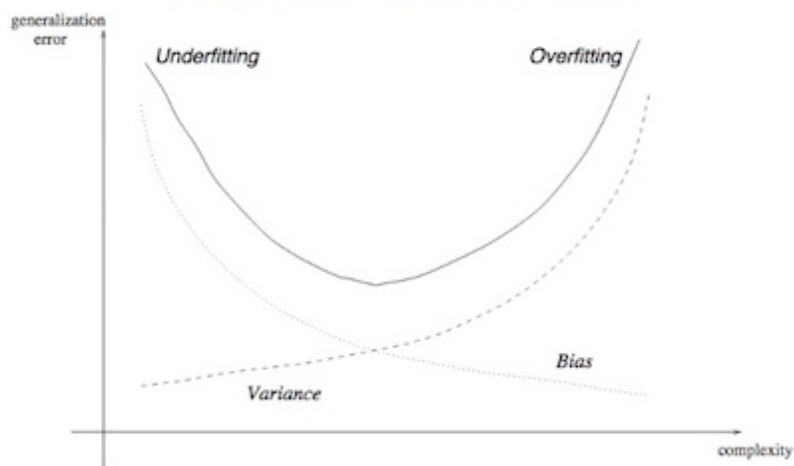
[cross entropy error vs. MSE explained](#)

Why use cross entropy, in neural network, instead of MSE?

- One reason is to address the issue of [learning slowdown](#) (introduced when a quadratic cost is used e.g. MSE). '... saying "learning is slow" is really the same as saying that those partial derivatives are small.'
- "Very often when we are trying to learn a probability from some true probability, we look to information theory to give us a measure of the distance between two distributions." [cs224d, notes1.pdf]

Bias vs. Variance trade-off [src](#)

Bias/variance trade-off



References:

[many terms for only two concepts in machine learning](#)

embedding-math

objective function: $\sum_{t=1}^T \sum_{c \in C_t} \log p(w_c | w_t)$

Softmax: $p(w_c | w_t) = \frac{\exp(w_t^T \cdot w_c)}{\sum_{w_i \in V} \exp(w_t^T w_i)}$

Vector comparison

Word similarity

Cosine similarity between two vectors (words) \vec{A} and \vec{B} :

$$\cos(\theta) = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| \cdot \|\vec{B}\|} = \frac{\text{dot}(A, B)}{\text{length}(A) \cdot \text{length}(B)} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

- θ is the angle between A and B
- $\|A\|$ is the magnitude/length of A
- \hat{A} is the unit vector (normalized vector) of A, defined as $\frac{\vec{A}}{\|A\|}$

cosine distance

cosine distance \equiv L2 (length) normalization *then* \Rightarrow Euclidean Distance [see vector comparison](#)

$$\text{Eulid. Dist.} = \sqrt{\sum (u_i - v_i)^2}$$

See [embedding-explained.ipynb](#)

TODO

- ☐ solve HWs of [cs224D course](#) local [folder](#)

embedding-explained

Embedding Explained

Idea and hypothesis

Based on an old assumption:

- **Distributional Similarity Hypothesis**, "You shall know a word by the company it keeps." (Firth, 1957)
- "a word is represented by means of its neighbors."
- "Similar words tend to occur in similar contexts".
- See description: CS224d [ppt](#), [vid](#), and [udacity](#)

Learning model (General):

- Word vectors are learned by predicting the correct word/context in a sequence of words (window).
- See [example](#)
- **Objective function** (General "Skip-gram"):

Given a large training corpus represented as a sequence of words w_1, \dots, w_T , the objective is to maximize the log-likelihood:

$$\sum_{t=1}^T \sum_{c \in C_t} \log p(w_c | w_t)$$

where C_t is the set of indices of words surrounding w_t "the window of w_t ".

The probability of observing a context word w_c given w_t is parametrized using the word vectors.

Given a scoring functions, which maps pairs of (word, context) to scores in \mathbb{R} , a possible choice to define the probability of a context word is the softmax

([Bojanowski, Mikolov et al., 2016](#))

Where the probability of a surrounding context word w_c given the center word w_t is calculated using **softmax** function as follows:

$$p(w_c | w_t) = \frac{\exp(w_t^T \cdot w_c)}{\sum_{w_i \in V} \exp(w_t^T w_i)}$$

- V is the vocabulary in our corpus.
- w is the vector representation of a word.

NOTE: in the above softmax function it is computationally expensive to evaluate all $w_i \in V$ "i.e. compute the conditional probabilities of all words" (cost is proportional to V), therefore the **hierarchical softmax** is used as an alternative to the full softmax. HS uses a binary tree representation, thus it cuts the evaluation to $\log(V)$. Similarly,

Negative Sampling (or Noise Contrastive Estimation NCE) is proposed as an alternative to hierarchical softmax.

Example calculation

If the input corpus is $w_1 w_2 w_3 w_4$ and the window size is 1, then our objective is to maximize the following sum:

t = 2

$$\log p(w_1 | w_2) + \log p(w_3 | w_2) \\ +$$

t = 3

$$\log p(w_2 | w_3) + \log p(w_4 | w_3)$$

and with the (full) softmax function, **the above** becomes

t = 2

$$\log \frac{\exp(w_2^T w_1)}{\text{denum}_{w_2}} + \log \frac{\exp(w_2^T w_3)}{\text{denum}_{w_2}} \\ +$$

t = 3

$$\log \frac{\exp(w_3^T w_2)}{\text{denum}_{w_3}} + \log \frac{\exp(w_3^T w_4)}{\text{denum}_{w_3}}$$

Where

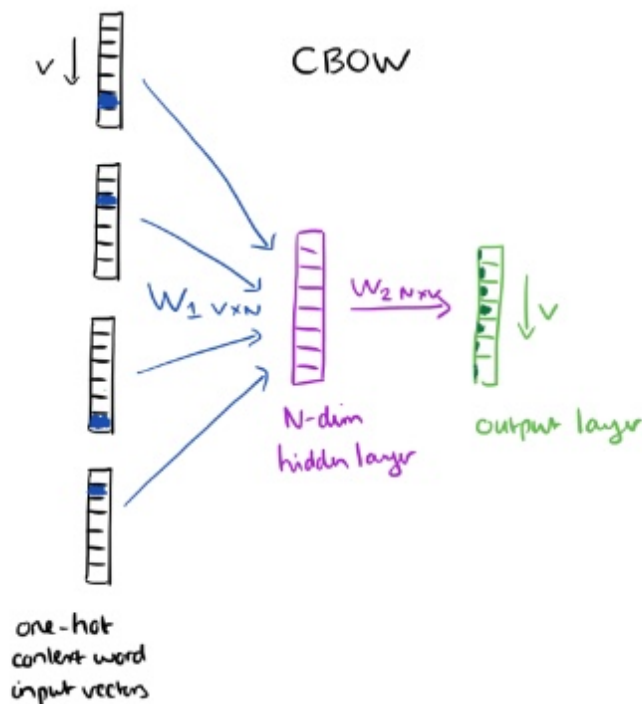
$$\text{denum}_{w_2} = \sum_{w_i \in V} \exp(w_2^T w_i) = \exp(w_2^T w_1) + \exp(w_2^T w_3) + \exp(w_2^T w_4)$$

$$\text{denum}_{w_3} = \sum_{w_i \in V} \exp(w_3^T w_i) = \exp(w_3^T w_1) + \exp(w_3^T w_2) + \exp(w_3^T w_4)$$

Powerfull gain: - Capture **semantic** and **syntactic** charactersitics (features) of text.

see notebook [embedding-explained.ipynb](#)

The context words form the input layer. Each word is encoded in one-hot form, so if the vocabulary size is V these will be V -dimensional vectors with just one of the elements set to one, and the rest all zeros. There is a single hidden layer and an output layer.



Hierarchical softmax [see](#), [keras's](#), and [Hugo's vid](#)

negative sampling

Concepts:

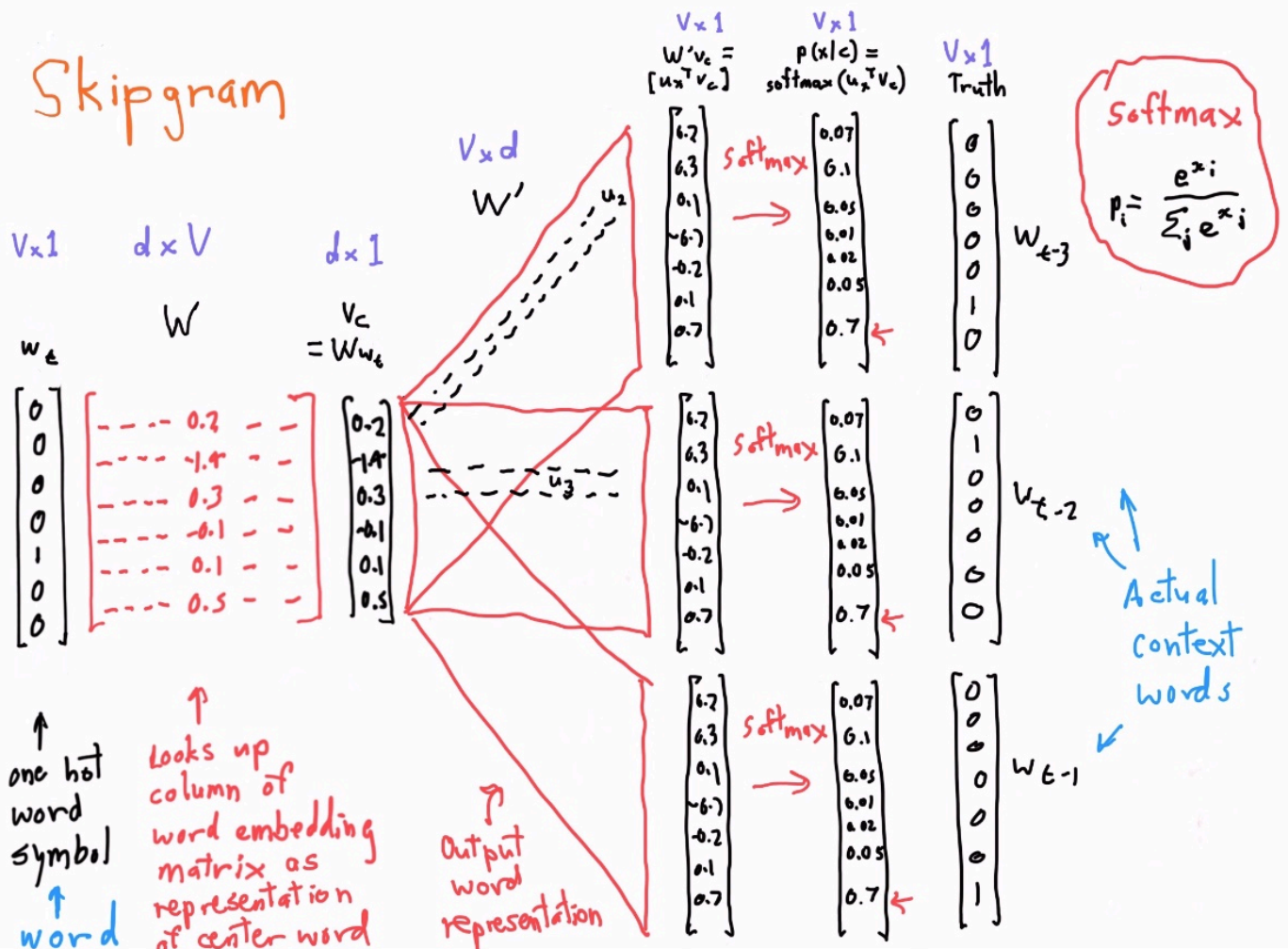
The powerful idea behind **word2vec speed** (thus its feasibility) is not about deep learning, it is because of hierarchical softmax and/or negative sampling which reduce the computation of softmax from $O(n)$ to $O(\log(n))$ and/or $O(k)$ respectively.

Q/A on Embedding:

[Why do low-dimensional embeddings capture huge statistical information?](#)

skip-gram model

Skipgram



source [cs224n Lec2](https://github.com/AskNowQA/QA-Tutorial)

<https://github.com/AskNowQA/QA-Tutorial>

Word Embeddings

Predictive model:

Given a word, what is the probable context.

$$p(w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2} | w_t)$$

or

Given a context, what is the probable word.

$$p(w_t | w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2})$$

TODO:

- ☐ Understand hierarchical softmax and negative sampling

References

1. [\(Mikolov et al., 2013\)](#) NIPS
2. [Word Embeddings for fun and profit in Gensim](#)
3. [Ilya Sutskever: continuous vector rep.](#)
4. [On word embeddings - Part 1](#) Sebastian Ruder blog
5. [TensorFlow tutorial: Vector Representations of Words](#)
6. [CS224D video lectures](#)
7. [Word Embeddings, Sense Embeddings and their Application to Word Sense Induction](#)
8. [Word Embeddings: Explaining their properties](#) "Off the Convext Path"
9. [Deep Learnign, Udacity vidoes](#)
10. Blog: [Word Embedding for the digital humanities](#)