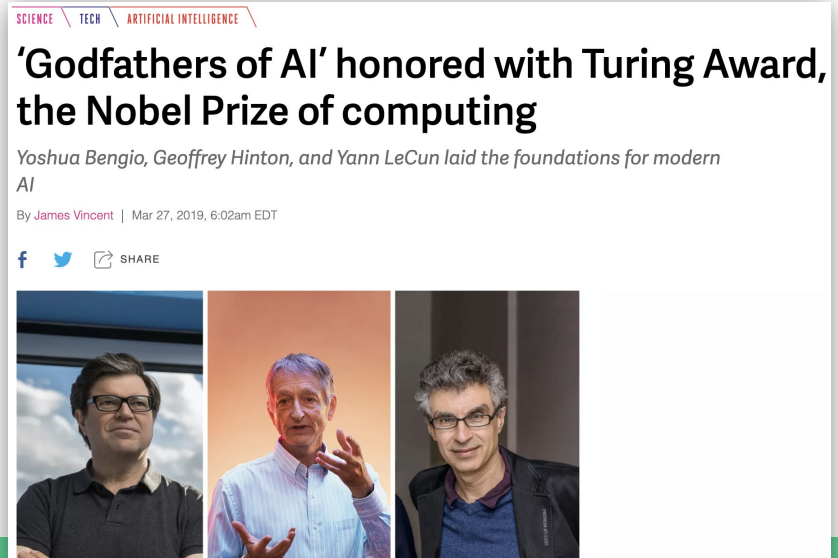
Going neural

ML at the time of Deep Learning

- There is no doubt that the recent wave of interest in AI has been driven by the so-called “**deep learning revolution**”.

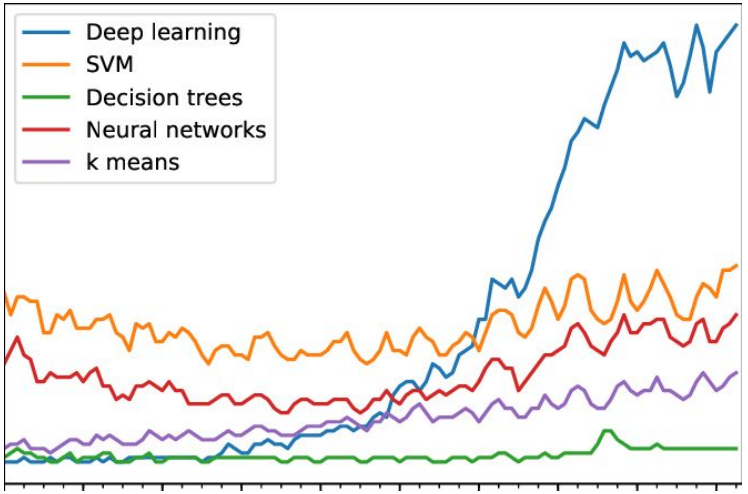

SCIENCE | TECH | ARTIFICIAL INTELLIGENCE

‘Godfathers of AI’ honored with Turing Award, the Nobel Prize of computing

Yoshua Bengio, Geoffrey Hinton, and Yann LeCun laid the foundations for modern AI

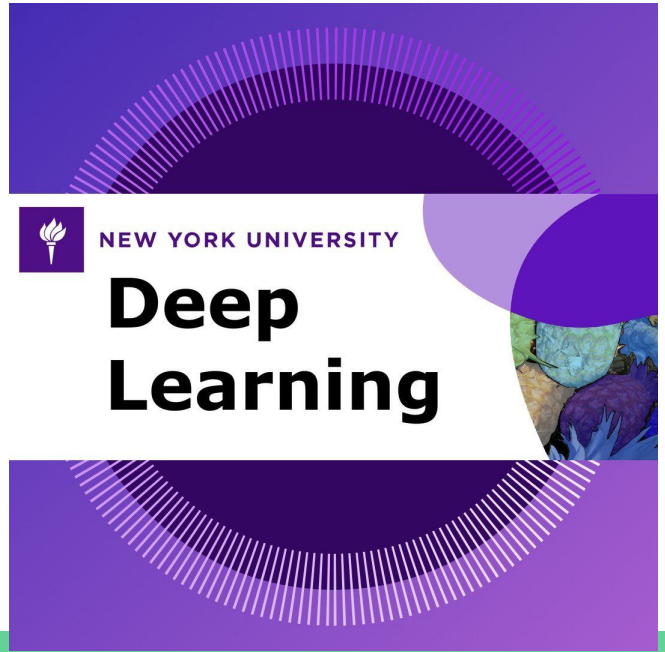
By James Vincent | Mar 27, 2019, 6:02am EDT

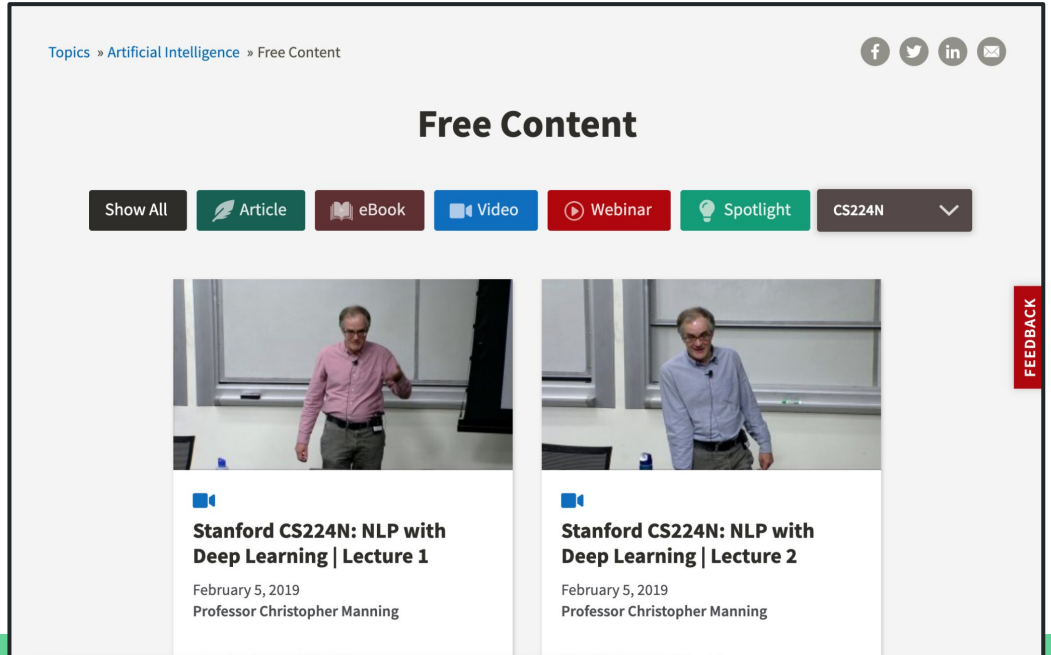
SHARE



Bonus: Internet is an amazing place!

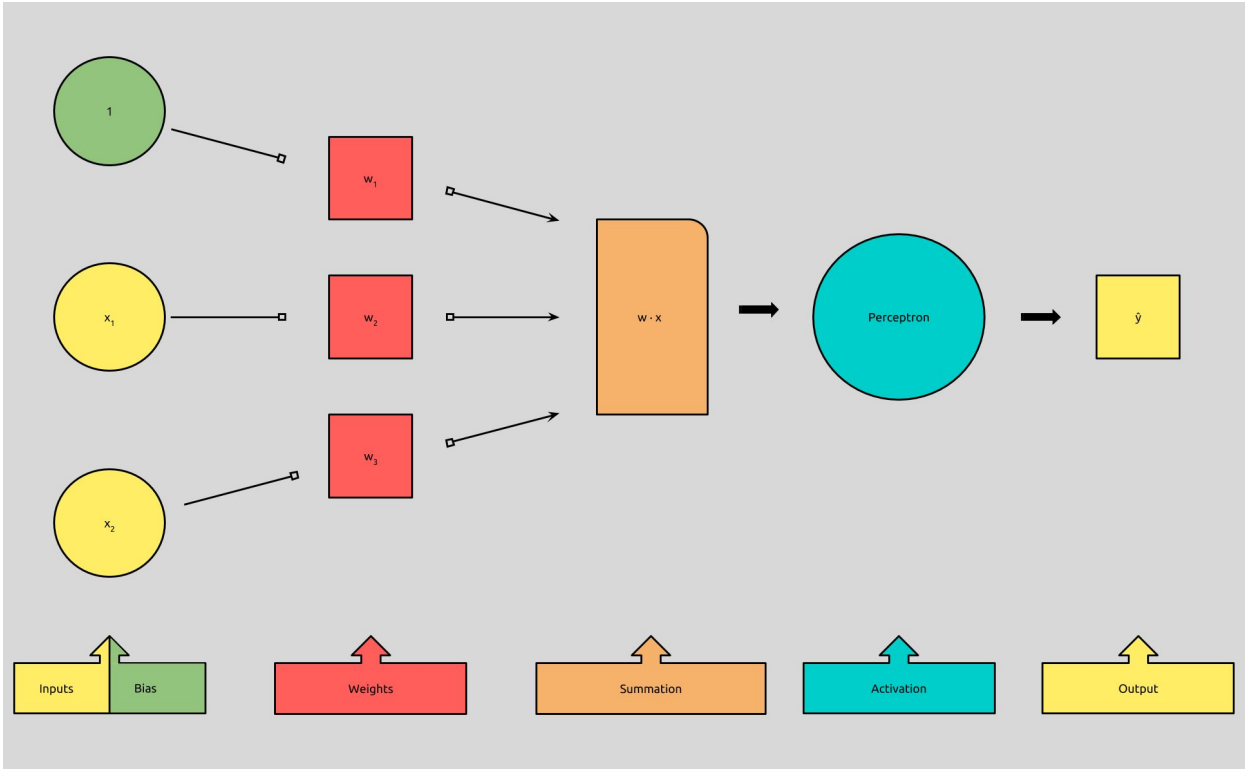
- NYU DL course
- Stanford NLP course





Blast from the past (1957): the perceptron

- Inputs, x_1, x_2
- Bias, **1**
- Weights, w_1, w_2, w_3
- Activation
- Result



Blast from the past (1957): the perceptron

- **Q1**: how do we make a decision (“forward pass”), given inputs and weights?

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0, \\ 0 & \text{otherwise} \end{cases}$$

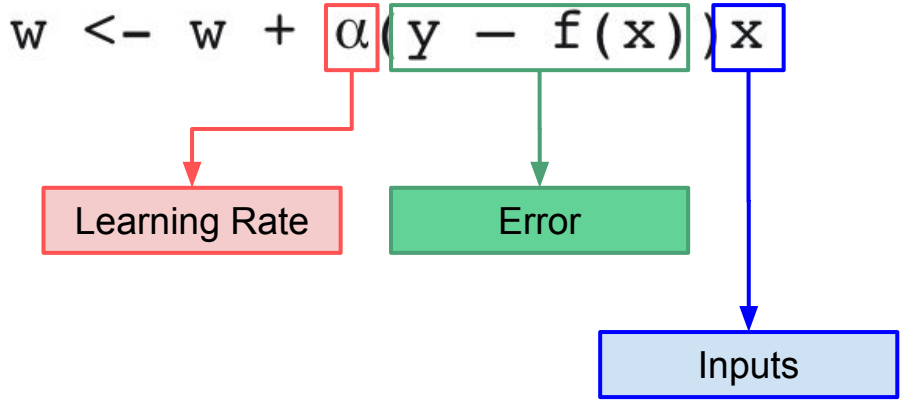
A threshold function: if the dot product of weights and inputs + bias > 0, the output is 1 (0 otherwise).

Example:

- Inputs = 1, 1, Weights: 0,0
- Bias = 1
- dot(input, weights) = 0
- Dot + bias = 1

Blast from the past (1957): the perceptron

- **Q2**: how do we learn the weights, based on a training set (“backward”)?



We calculate the prediction error, multiply by a learning rate and by the inputs, and update the weights.

Example:

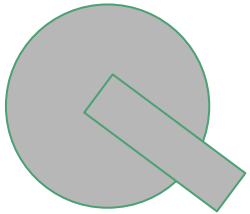
- Inputs = 0, 0, Weights: 0,0, Bias = 1
- Prediction = 1, Target = 0
- 0 = 0 + alpha(0 - 1) * 0

Blast from the past (1957): the perceptron

- Perceptrons can only learn linearly separable functions (Q: can they learn XOR?)
- Deep neural networks (that is networks in which several layers of neurons are stacked together, between the input layer and the output) are the distant cousins of the original perceptron, and have been proven to be able to learn extremely complex non-linear boundaries.
- While NNs are significantly more complex, they share the same basic building properties:
  a. We need an activation function, which may be a bit more complex (e.g. ReLU), but still determines the output of neurons based on inputs and weights;
  b. We need a learning procedure, which may be a bit more complex (backprop), but still determines how to iteratively adjust weights based on how “off” is the prediction with respect to the true label.

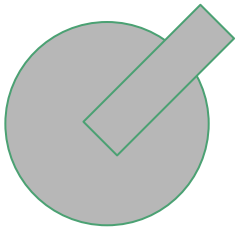
Back-propagation in the shower

1



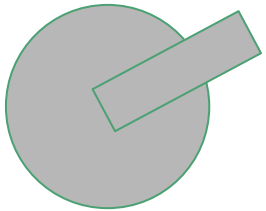
A bit too cold!

2



A bit too hot!

3



Great!

Back to NLP: from words to vectors (remember?)

- **Q**: We are used to feed “scikit models” with numbers for, say, regression, but how do we feed them *words*?
- **A**: We feed words by converting them to numbers!

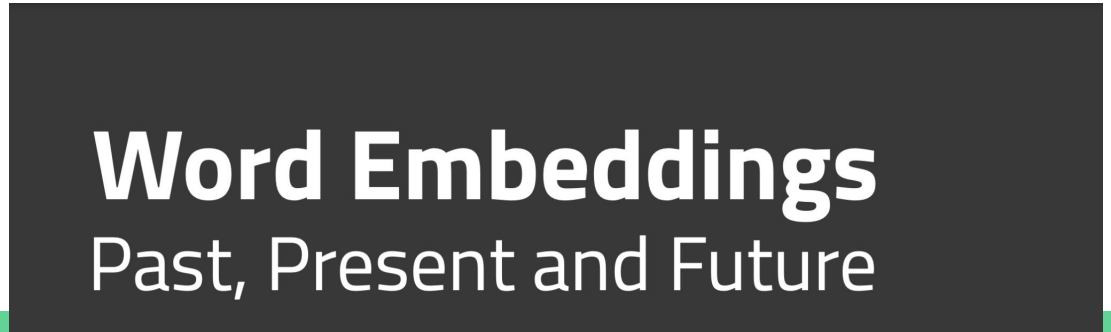
Option #1: count vectorizer (big and sparse!)

Option #2: TF IDF vectorizer (big and sparse!)

Option #3: word2vec (small and dense!)

Word vectors in a *prediction* task

- (Philosophical) distributional hypothesis: “words that appear in similar contexts have similar meanings”
- Computational hypothesis: learn a classifier that given a target word (e.g. cat) tell me how likely is that a context word appear next to it (Germany, furry), *and use the learned weights as the word vector.* Since the weights will adjust during training to make sure similar words have high probability, their vectors will be close in the resulting embedding space.



Word vectors in a *prediction* task

- CORPUS: “The furry cat is on the mat”
- WINDOW LENGTH: 2
- TARGET: “cat”
- INPUT PREPARATION, positive and negative samples

Target	Context	Label
cat	furry	1
cat	the	1
cat	is	1
cat	on	1

Target	Context	Label
cat	Berlin	0
cat	Jacopo	0
cat	ciao	0
cat	table	0

Word vectors in a *prediction* task

- CORPUS: “The furry cat is on the mat”
- WINDOW LENGTH: 2
- TARGET: “cat”
- INPUT PREPARATION, positive and negative samples (a=0.75)

$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_{w'} count(w')^\alpha}$$

Target	Context	Label
cat	furry	1
cat	the	1
cat	is	1
cat	on	1

Target	Context	Label
cat	Berlin	0
cat	Jacopo	0
cat	ciao	0
cat	table	0

Word vectors in a *prediction* task

- Now that we have input examples (positive and negative), let's define our learning objective:
  - We want to maximize the similarity of (t,c) drawn from the positive examples
  - We want to minimize the similarity of (t,c) drawn from the negative examples

$$L(\theta) = \sum_{(t,c)\in+} \log P(+|t,c) + \sum_{(t,c)\in-} \log P(-|t,c)$$

Word vectors in a *prediction* task

- Now that we have input examples (positive and negative), let's define our learning objective:
  - We want to maximize the similarity of (t,c) drawn from the positive examples
  - We want to minimize the similarity of (t,c) drawn from the negative examples

Dot product

Sigmoid

$$= \log \sigma(c \cdot t) + \sum_{i=1}^{k} \log \sigma(-n_i \cdot t)$$

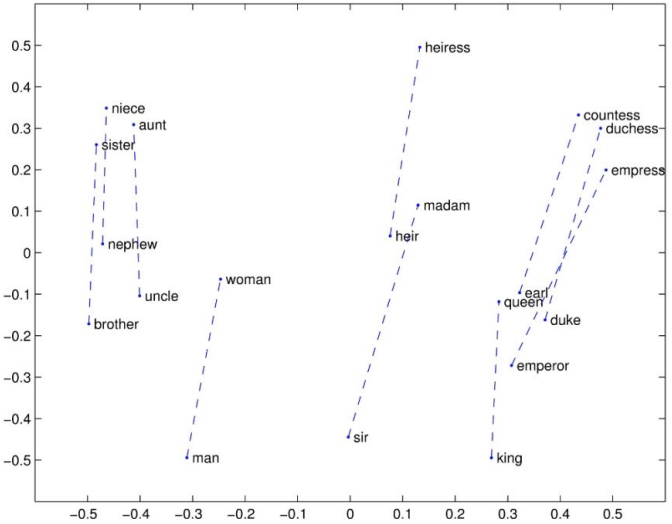
$$= \log \frac{1}{1+e^{-c \cdot t}} + \sum_{i=1}^{k} \log \frac{1}{1+e^{n_i \cdot t}}$$

“Talk is cheap, show me the code”

- Some pretty cool features of word2vec:
  - Vectors are good for downstream models: if we use 100-dim embeddings as features, a classifier can just learn 100 weights to represent a function.
  - Learning is “self-supervised”, as any text we can find on the internet can be turned into positive/negative pairs *without manual intervention*.
  - The vector space encodes automatically similarities and analogies.
  - Vectors are “*portable*”: they can be learned on Wikipedia and applied to finance news.
- The window size influences which “similarity” we care about:
  - Shorter windows leads to representations that are syntactically and semantically similar: “cat” will be very similar to “dog” for example - same topic (pet), same part of speech (noun)
  - Long context windows leads to representations that are topically related but not necessarily syntactically equivalent, such as “cat” and “furry” for example.
- We are going to drive these points home by looking at a practical implementation using the fantastic Python library Gensim - check the notebook code!

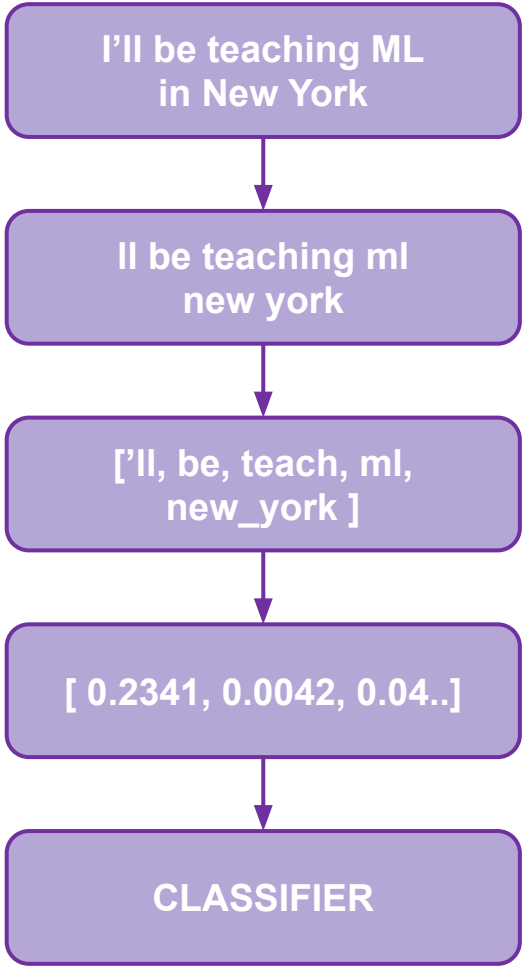
Doing algebra with words

- word2vec tend to capture well similarity between words and some analogical relations.
- In particular, once you have a well-trained embedding space, the offsets between vector embeddings can be used to solve analogies such as: “man : king = women : ?” (*queen*)
- This is possible since the result of vector(‘king’) - vector(‘man’) + vector(‘woman’) is a vector close to vector(‘queen’).



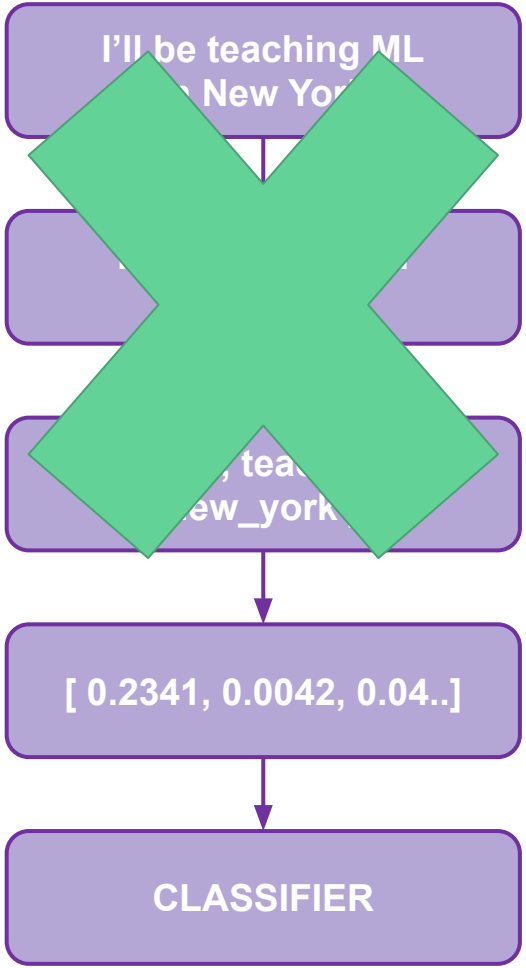
Text classification flow (again)

- Pre-processing: casing (?), punctuation (?), stop words (?).
- Tokenization: split text in tokens { stemming (?), lemmatization (?), phrases (?) }.
- Vectorization: apply a vectorization technique -> **word2vec**
- Modelling (training and testing): train a classifier model over text vectors - this is equivalent to your previous experience with scikit-like APIs.



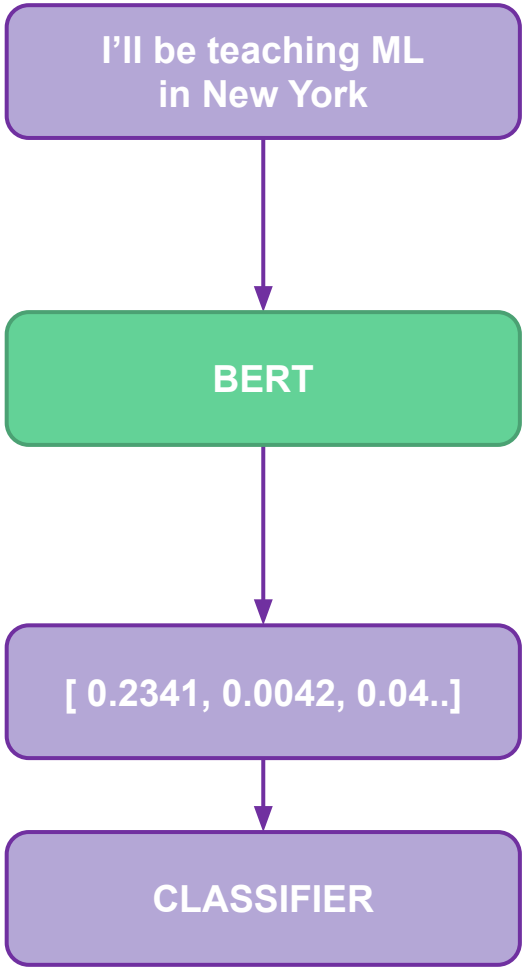
Text classification with BERT (teaser)

- Download a large pre-trained model, such as BERT.
- Vectorization: transform text into dense vectors.
- Feed vectors to a neural architecture for classification.



Text classification with BERT (teaser)

- Download a large pre-trained model, such as BERT.
- Vectorization: transform text into dense vectors.
- Feed vectors to a neural architecture for classification.



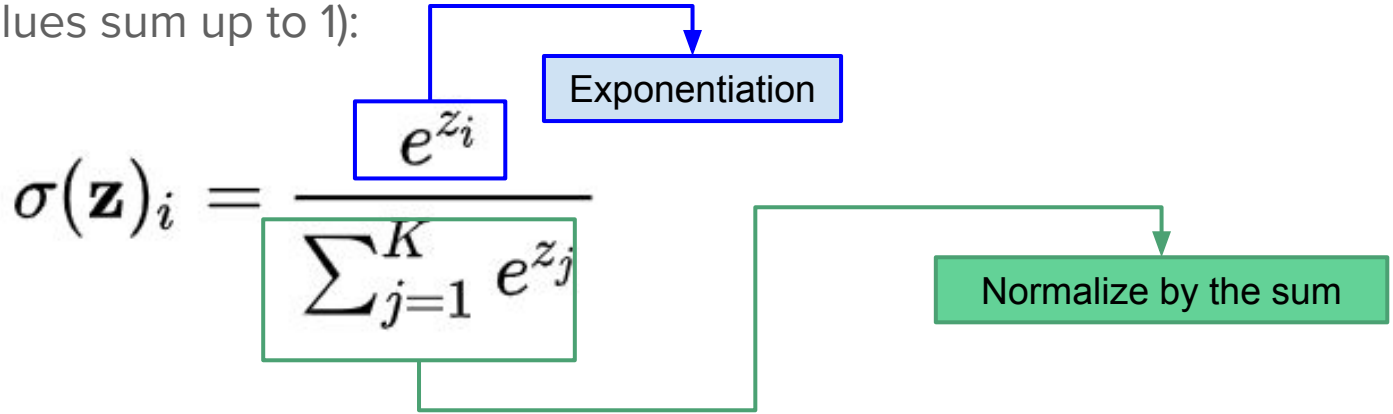
The softmax function (teaser)

- How do we go from a set of values (floats) outputted by neurons to a classification decision?
- The last activation function for classification is typically the *softmax* function, which takes a vector of floats and converts it into a vector of probabilities (i.e. values sum up to 1):

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

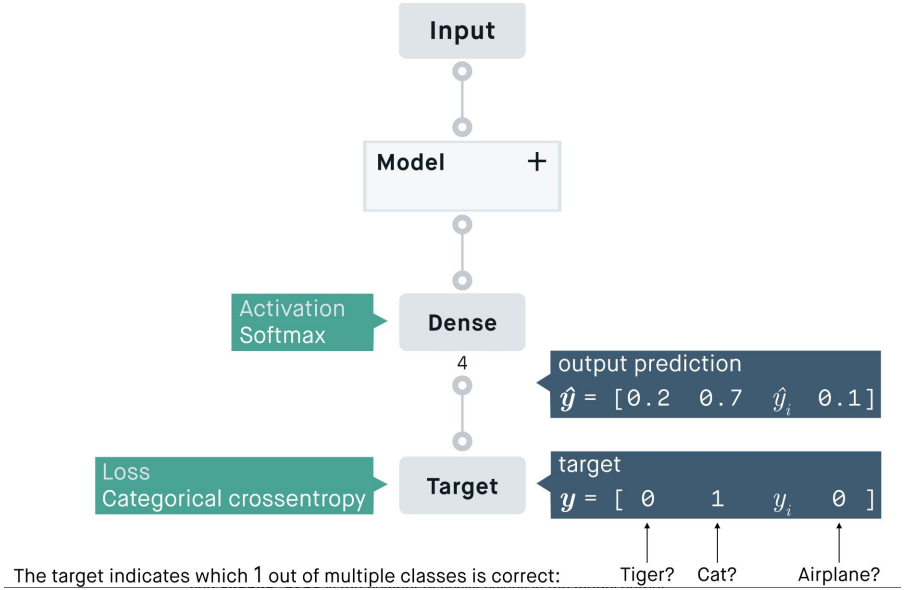
The softmax function (teaser)

- How do we go from a set of values (floats) outputted by neurons to a classification decision?
- The last activation function for classification is typically the *softmax* function, which takes a vector of floats and converts it into a vector of probabilities (i.e. values sum up to 1):



Cross-entropy loss (teaser)

- Given the output of a softmax and a true label for a classification problem, how do we define our loss?
- The typical choice is what is known as categorical cross-entropy:
  - "*The cross-entropy error function instead of the sum-of-squares for a classification problem leads to faster training as well as improved generalization*"



$$\text{Loss} = -\sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i$$

More readings

Word embeddings

- Skip-gram with negative sampling - the algorithm we explained - is one among many ways of getting word embeddings: another popular choice is GloVe.
- Piero Molino's fantastic slides (and lecture) on word2vec, and, more generally, the link between ideas in linguistics and philosophy and modern distributional methods in NLP.

Deep Neural Networks

- Our explanations of NN has been comically short, and mostly useful to introduce basic intuition. There are many resources for a proper deep learning introduction: in particular, the fantastic book by Keras creator, the deep learning book, and Fast AI course are excellent places to start.