Lecture 11: Neural networks

STATS 202: Statistical Learning and Data Science

Linh Tran
stat202@gmail.com

Department of Statistics
Stanford University

July 30, 2025

Announcements

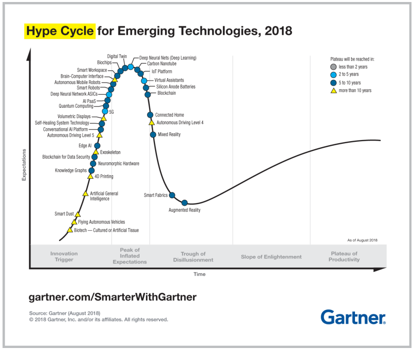
- Kaggle predictions due in 11 days
- Homework 4 is due in 7 days
- Final exam covers all material up to today
- No review session this Friday.

Outline

Neural networks

Currently, the most popular algorithm amongst ML practitioners.

- Many times, used within the context of Artificial Intelligence.
- Simply a general function estimation algorithm.
- Though is often hyped up the media.



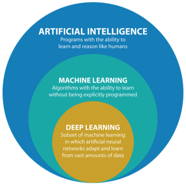
Gartner's hype cycle for 2018.

Neural networks

Lots of buzz words, but what do they mean?

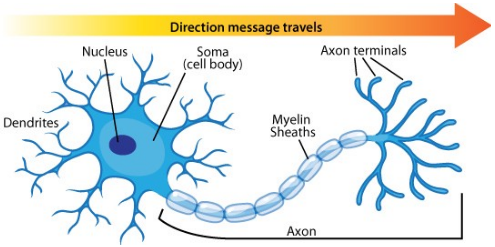
Definitions:

- *AI*: human-like machines or programs.
- *ML*: Algorithms that learn from data.
- *DL*: A type of ML algorithm, using neural networks (typically with many layers).


ARTIFICIAL INTELLIGENCE
Programs with the ability to learn and reason like humans
MACHINE LEARNING
Algorithms with the ability to learn without being explicitly programmed
DEEP LEARNING
Subset of machine learning in which artificial neural networks adapt and learn from vast amounts of data

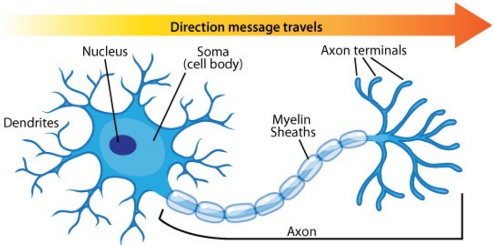
Neural networks

But: what exactly are neural networks?



Neural networks

But: what exactly are neural networks?

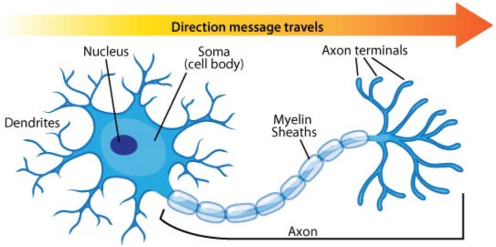


Some potential answers

- A *universal function approximator*.

Neural networks

But: what exactly are neural networks?

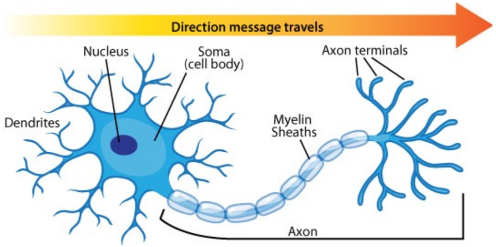


Some potential answers

- A *universal function approximator*.
- A *feature extractor*.

Neural networks

But: what exactly are neural networks?



Some potential answers

- A *universal function approximator*.
- A *feature extractor*.
- A *model generalizer*.

Logistic regression

Recall: logistic regression is a linear model with a logit link function, i.e.

$$\mathbb{P}(Y = 1|\mathbf{X}) = \sigma(\beta_0 + \beta_1 X_1 + ... + \beta_p X_p) : \sigma(z) = \frac{1}{1 + exp(-z)} \tag{1}$$

Let's rephrase this by:

1. Using b to denote β_0 (aka the bias)
2. Using **W** to denote $(\beta_0, ..., \beta_p)$ (aka the weights)
3. Using matrix notation

$$\mathbb{P}(Y = 1|\mathbf{X}) = \underbrace{\sigma}_{non-linearity} (\mathbf{X}\boldsymbol{W} + b) \tag{2}$$

Logistic regression

When the function is non-linear, our prior option was to do feature transformations, e.g.

- Expand predictor set (e.g. non-linear transformations, interactions, etc.).
- Define a kernel (e.g. find a function $f(\cdot, \cdot)$ that is positive definite).

Logistic regression

When the function is non-linear, our prior option was to do feature transformations, e.g.

- Expand predictor set (e.g. non-linear transformations, interactions, etc.).
- Define a kernel (e.g. find a function $f(\cdot, \cdot)$ that is positive definite).

Another option: build the non-linearity into the model specification, e.g.

$$\mathbb{P}(Y = 1|\mathbf{X}) = \sigma(\sigma(\mathbf{X}\boldsymbol{W}_1 + b_1)\boldsymbol{W}_2 + b_2) \tag{3}$$

Logistic regression

When the function is non-linear, our prior option was to do feature transformations, e.g.

- Expand predictor set (e.g. non-linear transformations, interactions, etc.).
- Define a kernel (e.g. find a function $f(\cdot, \cdot)$ that is positive definite).

Another option: build the non-linearity into the model specification, e.g.

$$\mathbb{P}(Y = 1|\mathbf{X}) = \sigma(\sigma(\mathbf{X}\boldsymbol{W}_1 + b_1)\boldsymbol{W}_2 + b_2) \tag{3}$$

This is a neural network (with 1 hidden layer)!

Hidden nodes

For logistic regression:

$$\mathbb{P}(Y = 1|\mathbf{X}) = \sigma(\mathbf{X}\underbrace{\boldsymbol{W}}_{p\times 1} + \underbrace{b}_{1\times 1}) \tag{4}$$

Hidden nodes

For logistic regression:

$$\mathbb{P}(Y = 1|\mathbf{X}) = \sigma(\mathbf{X} \underbrace{\boldsymbol{W}}_{p \times 1} + \underbrace{b}_{1 \times 1}) \tag{4}$$

For neural networks:

$$\mathbb{P}(Y = 1|\mathbf{X}) = \sigma(\sigma(\mathbf{X} \underbrace{\boldsymbol{W}_1}_{p \times M} + \underbrace{b_1}_{1 \times M}) \underbrace{\boldsymbol{W}_2}_{M \times 1} + \underbrace{b_2}_{1 \times 1}) \tag{5}$$

M specifies how many hidden nodes we have

- Called ‘*hidden*’ since it’s not directly observed by us.
- Also referred to as ‘*embeddings*’.

Hidden nodes

$$\mathbb{P}(Y=1|\mathbf{X}) = \sigma(\sigma(\mathbf{X}\underbrace{\boldsymbol{W}_1}_{p\times M} + \underbrace{b_1}_{1\times M})\underbrace{\boldsymbol{W}_2}_{M\times 1} + \underbrace{b_2}_{1\times 1}) \tag{6}$$

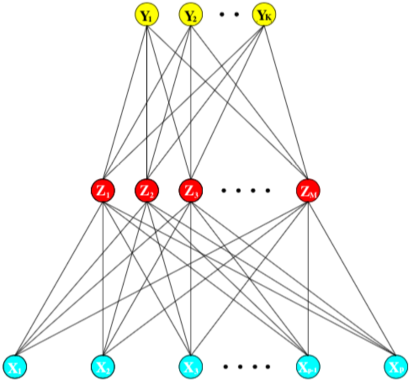


FIGURE 11.2. *Schematic of a single hidden layer, feed-forward neural network.*

Hidden layers

We can iteratively apply our non-linear operations, e.g.

$$\mathbb{P}(Y = 1|\mathbf{X}) = \sigma(\cdots\sigma(\sigma(\mathbf{X}\boldsymbol{W}_1 + b_1)\boldsymbol{W}_2 + b_2)\cdots\boldsymbol{W}_B + b_B) \quad (7)$$

Where B is the number of iterations (i.e. *hidden layers*).

Hidden layers

We can iteratively apply our non-linear operations, e.g.

$$\mathbb{P}(Y = 1|\mathbf{X}) = \sigma(\cdots\sigma(\sigma(\mathbf{X}\boldsymbol{W}_1 + b_1)\boldsymbol{W}_2 + b_2)\cdots\boldsymbol{W}_B + b_B) \quad (7)$$

Where B is the number of iterations (i.e. *hidden layers*).

n.b. Consequently, we have three hyper-parameters:

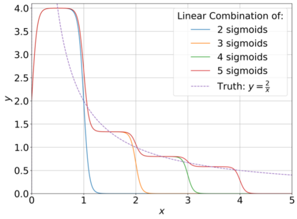
- The number of hidden layers
- The number of nodes within each layer
- The activation function

Deeper vs wider network

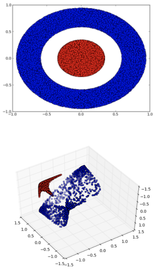
Considerations

1. Increasing both requires more parameters (and can overfit)
2. Deeper networks capture more “complex” functions; wider networks capture more “diverse” features
3. Wider networks result in more “localization”
4. Deeper networks are harder to train (vanishing gradient)
5. In practice, typically start with random sets of hyperparameters and fine tune from there.

Wider networks

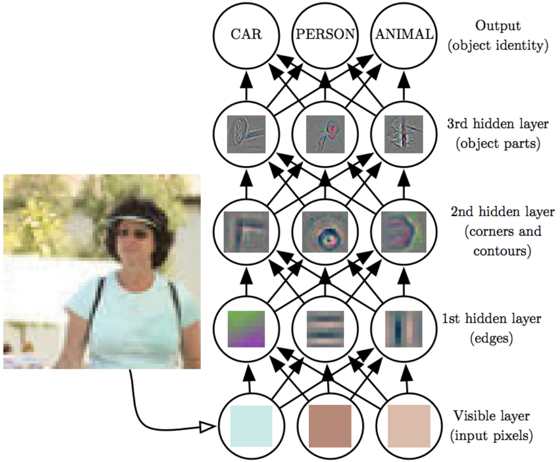


Example 1



Example 2

Deeper networks



Multinomial logistic regression

Our examples have been for binary outcomes so far.
Question: What about multinomial outcomes

- e.g. Which of digits 0 through 9 is this photo?

Multinomial logistic regression

Our examples have been for binary outcomes so far.
Question: What about multinomial outcomes

- e.g. Which of digits 0 through 9 is this photo?

Recall: for logistic regression, we're modeling

$$\begin{aligned}\log\left[\frac{\mathbb{P}(Y=1|\mathbf{X})}{1-\mathbb{P}(Y=1|\mathbf{X})}\right] &= \beta_0+\beta_1X_1+\ldots+\beta_pX_p \quad (8)\\ &= \mathbf{XW} \quad (9)\end{aligned}$$

where $\mathbf{X}$ is our $n \times p$ design matrix and $\mathbf{W}$ is our $p \times 1$ parameter vector.

Multinomial logistic regression

For multinomial regression, let $Y \in \{1, \ldots, K\}$. We can model

$$\log\left[\frac{\mathbb{P}(Y=1|\mathbf{X})}{\mathbb{P}(Y=K|\mathbf{X})}\right] = \mathbf{XW_1} \tag{10}$$

$$\log\left[\frac{\mathbb{P}(Y=2|\mathbf{X})}{\mathbb{P}(Y=K|\mathbf{X})}\right] = \mathbf{XW_2} \tag{11}$$

$$\cdots = \cdots \tag{12}$$

$$\log\left[\frac{\mathbb{P}(Y=K-1|\mathbf{X})}{\mathbb{P}(Y=K|\mathbf{X})}\right] = \mathbf{XW_{K-1}} \tag{13}$$

where each $\mathbf{W_k}$ is a $p \times 1$ parameter vector.

Exponentiating both sides and solving for $\mathbb{P}(Y=K|\mathbf{X})$ (using the fact that the probabilities have to sum to 1) gives us

$$\mathbb{P}(Y=K|\mathbf{X}) = \frac{1}{1+\sum_{k=1}^{K-1} e^{\mathbf{XW_k}}} \tag{14}$$

Multinomial logistic regression

Equivalently, can represent the multinomial logistic model as

$$\begin{aligned}
\log \mathbb{P}(Y = 1|\mathbf{X}) &= \mathbf{X}\mathbf{W_1} - \log(Z) \quad (15)\\
\log \mathbb{P}(Y = 2|\mathbf{X}) &= \mathbf{X}\mathbf{W_2} - \log(Z) \quad (16)\\
\cdots &= \cdots \quad (17)\\
\log \mathbb{P}(Y = K|\mathbf{X}) &= \mathbf{X}\mathbf{W_K} - \log(Z) \quad (18)
\end{aligned}$$

resulting in the following probabilities

$$\begin{aligned}
\mathbb{P}(Y = 1|\mathbf{X}) &= \frac{\exp(\mathbf{X}\mathbf{W_1})}{\sum_{k=1}^{K} \exp(\mathbf{X}\mathbf{W_k})} \quad (19)\\
\mathbb{P}(Y = 2|\mathbf{X}) &= \frac{\exp(\mathbf{X}\mathbf{W_2})}{\sum_{k=1}^{K} \exp(\mathbf{X}\mathbf{W_k})} \quad (20)\\
\cdots &= \cdots \quad (21)\\
\mathbb{P}(Y = K|\mathbf{X}) &= \frac{\exp(\mathbf{X}\mathbf{W_K})}{\sum_{k=1}^{K} \exp(\mathbf{X}\mathbf{W_k})} \quad (22)
\end{aligned}$$

Multinomial logistic regression

This leads us to the softmax function, i.e.

$$\text{softmax}(\mathbf{XW_1}, \ldots, \mathbf{XW_K})_k = \frac{e^{\mathbf{XW_k}}}{\sum_{l=1}^{K} e^{\mathbf{XW_l}}} \tag{23}$$

Or, more succintly, we have

$$\text{softmax}(\mathbf{XW^K})_k = \frac{\exp((\mathbf{XW^K})_{k\cdot})}{\sum_{k=1}^{K} \exp((\mathbf{XW^K})_{k\cdot})} \tag{24}$$

where the $p \times K$ matrix $\mathbf{W^K}$ is simply the (concatenated) matrix of $\mathbf{W_1}, \ldots, \mathbf{W_K}$.

This is what multiclass neural networks are modeling!

The chain rule

Recall: In logistic regression we try to maximize the likelihood

- Equivalent to minimizing the cross-entropy

$$
\begin{aligned}
L(y_i, f(\mathbf{X}_i)) &= -y_i \log(p_i) - (1 - y_i)\log(1 - p_i), \text{ where} && (25)\\
p_i &= \frac{1}{1 + exp(-Z_i)} && (26)\\
Z_i &= \mathbf{X}_i \boldsymbol{W} && (27)
\end{aligned}
$$

The chain rule

Recall: In logistic regression we try to maximize the likelihood

- Equivalent to minimizing the cross-entropy

$$\begin{aligned} L(y_i, f(\mathbf{X}_i)) &= -y_i \log(p_i) - (1-y_i)\log(1-p_i), \text{ where} & (25) \\ p_i &= \frac{1}{1+exp(-Z_i)} & (26) \\ Z_i &= \mathbf{X}_i \boldsymbol{W} & (27) \end{aligned}$$

Can apply *the derivative chain rule* to get our gradient, i.e.

$$\frac{\partial L(y_i, f(\mathbf{X}_i))}{\partial \boldsymbol{W}} = \frac{\partial L(y_i, f(\mathbf{X}_i))}{\partial p_i} \times \frac{\partial p_i}{\partial Z_i} \times \frac{\partial Z_i}{\partial \boldsymbol{W}} \qquad (28)$$

The chain rule

Recall: In logistic regression we try to maximize the likelihood

- Equivalent to minimizing the cross-entropy

$$\begin{aligned} L(y_i, f(\mathbf{X}_i)) &= -y_i \log(p_i) - (1 - y_i) \log(1 - p_i), \text{ where} && (25) \\ p_i &= \frac{1}{1 + exp(-Z_i)} && (26) \\ Z_i &= \mathbf{X}_i \boldsymbol{W} && (27) \end{aligned}$$

Can apply *the derivative chain rule* to get our gradient, i.e.

$$\frac{\partial L(y_i, f(\mathbf{X}_i))}{\partial \boldsymbol{W}} = \frac{\partial L(y_i, f(\mathbf{X}_i))}{\partial p_i} \times \frac{\partial p_i}{\partial Z_i} \times \frac{\partial Z_i}{\partial \boldsymbol{W}} \tag{28}$$

Which gives us

$$\frac{\partial L(y_i, f(\mathbf{X}_i))}{\partial \boldsymbol{W}} = \mathbf{X}_i (y_i - p_i) \tag{29}$$

Backpropagation

Neural networks are simply a generalization of the logistic regression case, e.g. for

$$\mathbb{P}(Y = 1 | \mathbf{X}) = \sigma(\sigma(\mathbf{X} \boldsymbol{W}_1) \boldsymbol{W}_2) \tag{30}$$

Backpropagation

Neural networks are simply a generalization of the logistic regression case, e.g. for

$$\mathbb{P}(Y = 1|\mathbf{X}) = \sigma(\sigma(\mathbf{X}\boldsymbol{W}_1)\boldsymbol{W}_2) \tag{30}$$

Our loss is

$$\begin{aligned} L(y_i, f(\mathbf{X}_i)) &= -y_i \log(p_i) - (1 - y_i)\log(1 - p_i), \text{ where} && (31) \\ p_i &= \frac{1}{1 + exp(-Z_{2,i})} && (32) \\ Z_{2,i} &= h_i \boldsymbol{W}_2 && (33) \\ h_i &= \frac{1}{1 + exp(-Z_{1,i})} && (34) \\ Z_{1,i} &= \mathbf{X}\boldsymbol{W}_1 && (35) \end{aligned}$$

Backpropagation

Our loss is

$$\begin{aligned}
L(y_i, f(\mathbf{X}_i)) &= -y_i \log(p_i) - (1 - y_i)\log(1 - p_i), \text{ where} && (36)\\
p_i &= \frac{1}{1 + exp(-Z_{2,i})} && (37)\\
Z_{2,i} &= h_i \boldsymbol{W}_2 && (38)\\
h_i &= \frac{1}{1 + exp(-Z_{1,i})} && (39)\\
Z_{1,i} &= \mathbf{X}\boldsymbol{W}_1 && (40)
\end{aligned}$$

Applying *the derivative chain rule*:

$$\begin{aligned}
\frac{\partial L(y_i, f(\mathbf{X}_i))}{\partial \boldsymbol{W}_2} &= \frac{\partial L(y_i, f(\mathbf{X}_i))}{\partial p_i} \times \frac{\partial p_i}{\partial Z_{2,i}} \times \frac{\partial Z_{2,i}}{\partial \boldsymbol{W}_2}\\
\frac{\partial L(y_i, f(\mathbf{X}))}{\partial \boldsymbol{W}_1} &= \frac{\partial L(y_i, f(\mathbf{X}))}{\partial p_i} \times \frac{\partial p_i}{\partial Z_{2,i}} \times \frac{\partial Z_{2,i}}{\partial h_i} \times \frac{\partial h_i}{\partial Z_{1,i}} \times \frac{\partial Z_{1,i}}{\partial \boldsymbol{W}_1}
\end{aligned}$$

Gradient descent

Our gradient is estimated using our data, i.e. $(y_i, \mathbf{X}_i) : i = 1, 2, \ldots, n$.

Gradient descent

Our gradient is estimated using our data, i.e. $(y_i, \mathbf{X}_i) : i = 1, 2, \ldots, n$.

We can estimate it using, e.g.

- **Stochastic gradient descent**: estimating our (full) gradient using just one observation.
- **Gradient descent**: estimating our (full) gradient using all observations.
- **Mini-batch gradient descent**: using a (random) subsample of our observations.

Each will trade off between variance for the gradient and memory size.

Gradient descent

Our gradient is estimated using our data, i.e. $(y_i, \mathbf{X}_i) : i = 1, 2, \ldots, n$.

We can estimate it using, e.g.

- **Stochastic gradient descent**: estimating our (full) gradient using just one observation.
- **Gradient descent**: estimating our (full) gradient using all observations.
- **Mini-batch gradient descent**: using a (random) subsample of our observations.

Each will trade off between variance for the gradient and memory size.

When done iteratively, we'll typically specify a stopping point (e.g. by using a dev set).

Gradient descent

The use of non-linearities results in multiple minima, tendency to overfit, and can be unstable.
Some considerations to make:

- Set initial weight values near zero.
- Over parameterize and regularize heavily.
- Standardize input features.
- Use a dev set and stop training earlier.
- Try out different weight randomizations and take the one with the lowest (validated) error.
    - Or average the predictions (or apply bagging).

Estimation summary

Estimating neural network parameters simply requires '*propagating back*' errors.

- We're just applying (matrix) multiplications
  - GPU's can be very good for this
- Matrix multiplications can get pretty big (for large networks)
  - Commonly not worth it to use the Hessian
- Should be careful with large values going into sigmoid activations
  - Results in saturated gradients

Universal approximation theorem

Hornik's theorem

Whenever the activation function is continuous, bounded, and non-constant, then, for arbitrary compact subsets $X \subseteq R^k$, standard multilayer feedforward networks can approximate any continuous function on X arbitrarily well with respect to uniform distance, provided that sufficiently many hidden units are avaiable.

Universal approximation theorem

Hornik's theorem

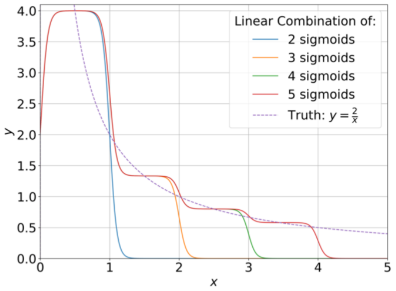
Whenever the activation function is continuous, bounded, and non-constant, then, for arbitrary compact subsets $X \subseteq R^k$, standard multilayer feedforward networks can approximate any continuous function on X arbitrarily well with respect to uniform distance, provided that sufficiently many hidden units are avaiable.

In words: A 2-layer neural network with enough hidden nodes can closely approximate any continuous function $f(x)$.

References:
Cybenko (1989) "Approximations by superpostions of sigmoidal function"
Hornik (1991) "Approximation Capabilities of Multilayer Feedforward Networks"
Leshno and Schocken (1993) "Multilayer Feedforward Networks with Non-Polynomial Activation Functions Can Approximate Any Function"

Universal approximation theorem

Given enough hidden nodes, we can approximate any function.



Check out this *visual example* of this.

Universal approximation theorem

Some caveats to the theorem

- We're approximating the function within some bound, i.e. $|\hat{f}_n(x) - f(x)| < \epsilon$.
- Result is meant for *continuous* functions on *compact* subsets of $\mathbb{R}$.
- Nothing is guaranteed on the how quickly we can learn the function's parameters.
- Other function estimators also do a good job approximating!

Feature engineering

For non-linear functions,

Logistic regression: expand our feature set via transformations

Neural network: define the model non-linearly

Feature engineering

For non-linear functions,

Logistic regression: expand our feature set via transformations

- We have to specify the feature transformations

Neural network: define the model non-linearly

- The model learns the feature transformations

Feature engineering

For non-linear functions,

Logistic regression: expand our feature set via transformations

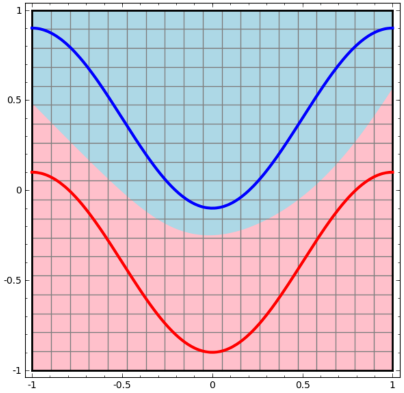
- We have to specify the feature transformations

Neural network: define the model non-linearly

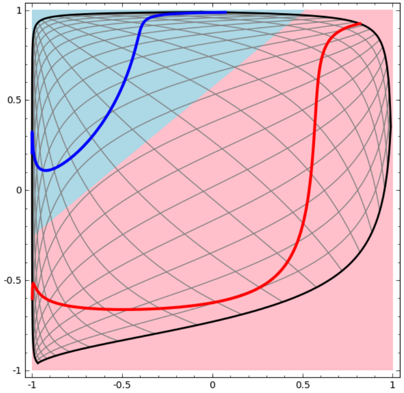
- The model learns the feature transformations
- *This helps us greatly when dealing with abstract or high dimensional problems (e.g. images & text)!*

Feature representation

How do the feature transformations get learned?



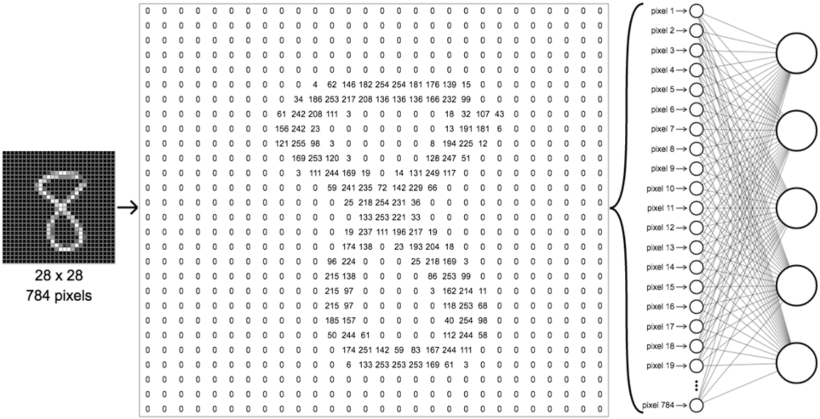
Original representation of curves



Hidden layer representation of curves

Well demonstrated by *Chris Olah's blog*.

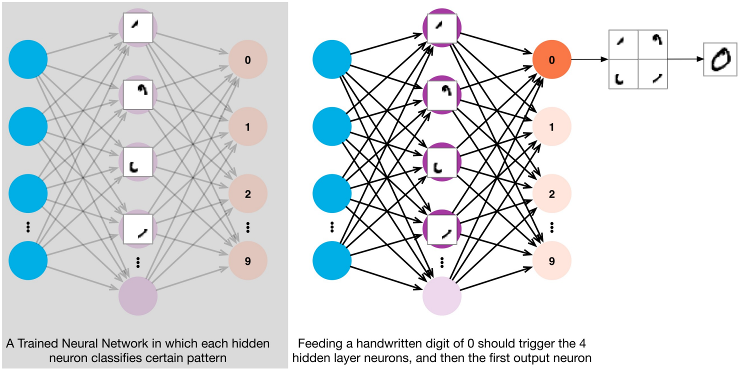
Example: The MNIST data



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	4	62	146	182	254	254	181	176	139	15	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	34	186	253	217	208	136	136	136	166	232	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	61	242	208	111	3	0	0	0	0	0	18	32	107	43	0	0	0	0	0	0
0	0	0	0	0	0	0	0	156	242	23	0	0	0	0	0	0	0	13	191	181	6	0	0	0	0	0	0
0	0	0	0	0	0	0	0	121	255	98	3	0	0	0	0	0	8	194	225	12	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	169	253	120	3	0	0	0	0	128	247	51	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3	111	244	169	19	0	14	131	249	117	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	59	241	235	72	142	229	66	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	25	218	254	231	36	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	133	253	221	33	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	19	237	111	196	217	19	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	138	0	23	193	204	18	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	96	224	0	0	0	25	218	169	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	138	0	0	0	0	86	253	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	3	162	214	11	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	0	118	253	68	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	185	157	0	0	0	0	0	40	254	98	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	50	244	61	0	0	0	0	112	244	58	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	251	142	59	83	167	244	111	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	6	133	253	253	253	169	61	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

A linear model (e.g. for multinomial logistic regression)

Example: The MNIST data



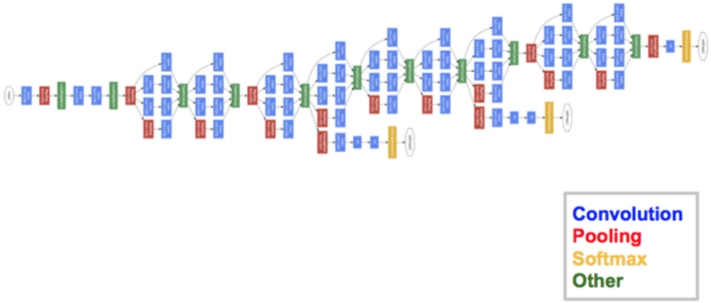
A Trained Neural Network in which each hidden neuron classifies certain pattern

Feeding a handwritten digit of 0 should trigger the 4 hidden layer neurons, and then the first output neuron

Neural network's learned (kernel) features.

Model generalizations

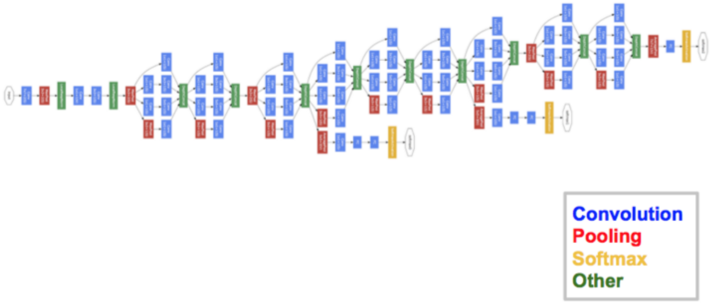
Knowing that we can simply back propagate errors via multiplication opens many doors for us, e.g.



Google's InceptionNet architecture

Model generalizations

Knowing that we can simply back propagate errors via multiplication opens many doors for us, e.g.



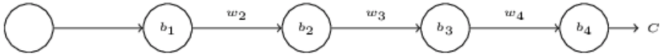
Google's InceptionNet architecture

Problem: *large networks are vulnerable to vanishing/exploding gradients.*

The vanishing gradient problem

Recall: Our gradient is simply a product of partial derivatives.

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$

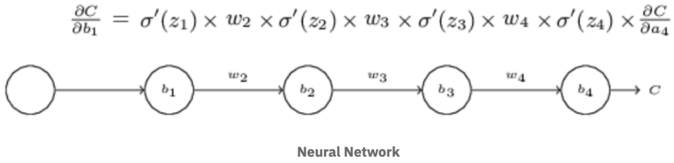


Neural Network

Example neural network and gradient.

The vanishing gradient problem

Recall: Our gradient is simply a product of partial derivatives.



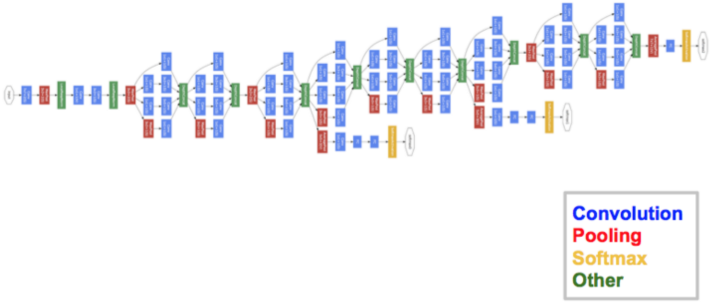
$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$
b_1
w_2
b_2
w_3
b_3
w_4
b_4
C
Neural Network

Example neural network and gradient.

Question: What is the derivative of the sigmoid function?

The vanishing gradient problem

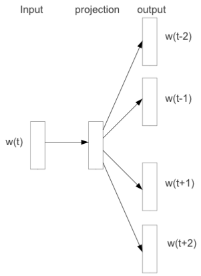
Google's InceptionNet address this via strategically placed loss functions.



Google's InceptionNet architecture

Model generalizations

Embeddings: The skip gram model

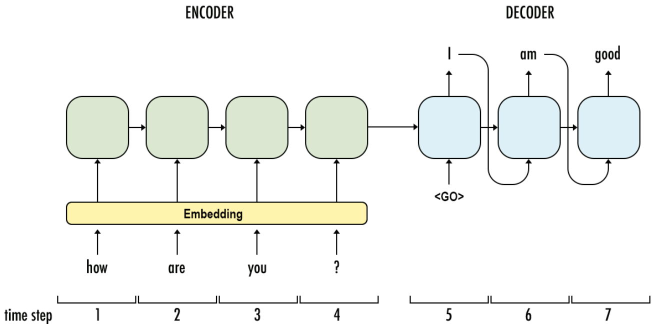


$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$$

$$p(w_O|w_I) = \frac{\exp\left({v'_{w_O}}^\top v_{w_I}\right)}{\sum_{w=1}^{W}\exp\left({v'_w}^\top v_{w_I}\right)}$$

Model generalizations

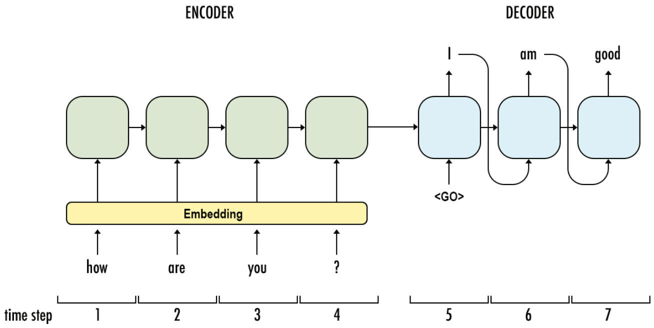
We can also train models end to end, e.g.



Encoder-decoder architecture

Model generalizations

We can also train models end to end, e.g.

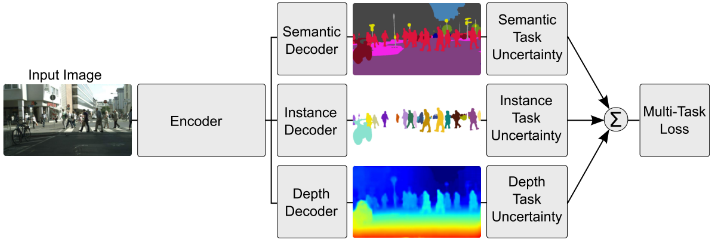


Encoder-decoder architecture

Or: in a modular fashion, e.g. pre-training.

Model generalizations

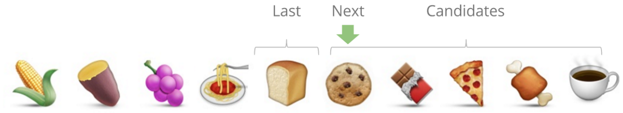
Or over multiple tasks (i.e. multi-task learning), e.g.



Kendall et al. 2017's multi-task model

Model generalizations

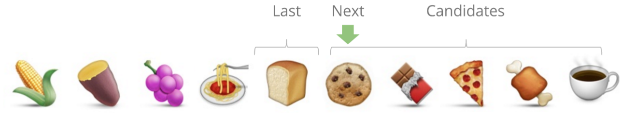
Many researchers will create unique architectures for specific problems, e.g. *Instacart*



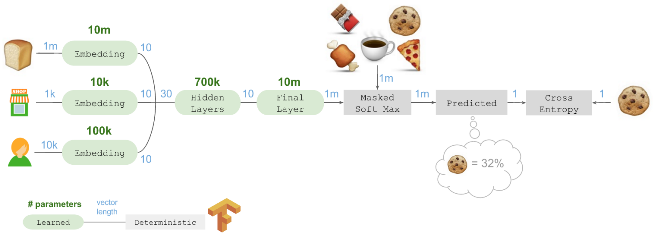
The prediction problem

Model generalizations

Many researchers will create unique architectures for specific problems, e.g. *Instacart*



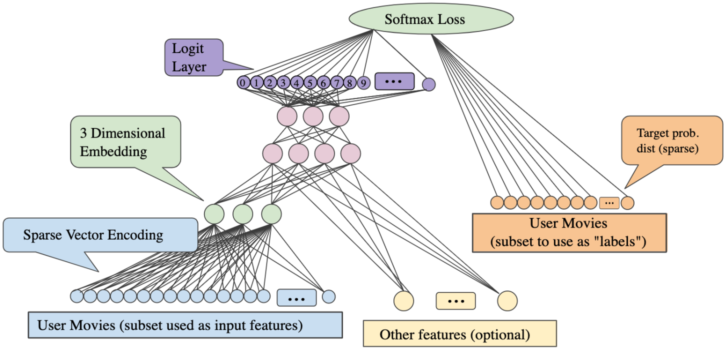
The prediction problem



The intial solution

Model generalizations

Another example using the Netflix data.

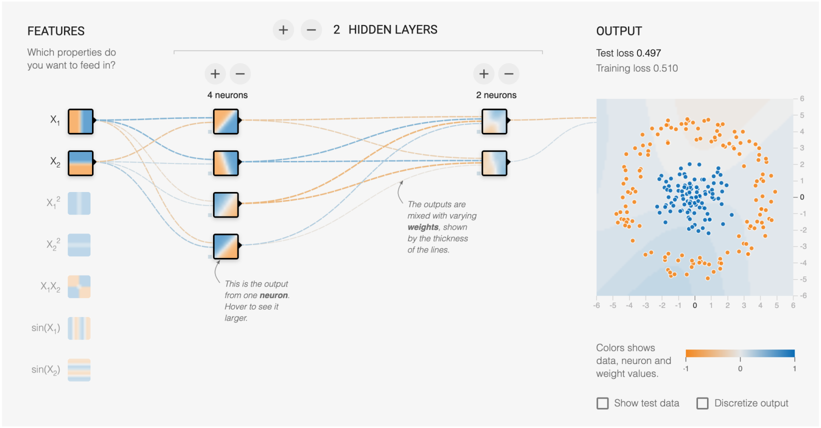


Softmax Loss
Logit Layer
0 1 2 3 4 5 6 7 8 9 ...
3 Dimensional Embedding
Target prob. dist (sparse)
User Movies (subset to use as "labels")
Sparse Vector Encoding
User Movies (subset used as input features)
Other features (optional)

Figure 5. A sample DNN architecture for learning movie embeddings from collaborative filtering data.

The tensorflow playground

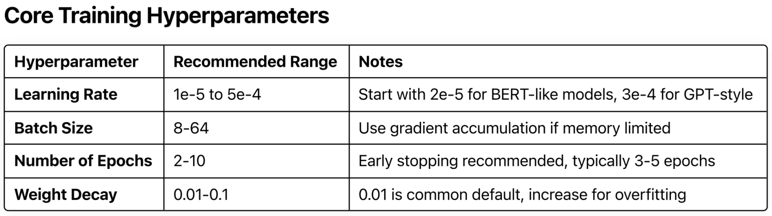
An *interactive demo* that allows you to play with a neural network.



FEATURES
Which properties do you want to feed in?
X_1
X_2
X_1^2
X_2^2
X_1X_2
$\sin(X_1)$
$\sin(X_2)$
+ − 2 HIDDEN LAYERS
4 neurons
2 neurons
This is the output from one **neuron**. Hover to see it larger.
The outputs are mixed with varying **weights**, shown by the thickness of the lines.
OUTPUT
Test loss 0.497
Training loss 0.510
Colors shows data, neuron and weight values.
-1
0
1
Show test data
Discretize output

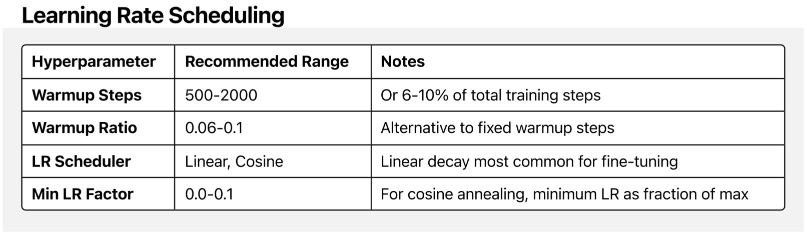
Hyperparameters

Some proposed settings



Core Training Hyperparameters

Hyperparameter	Recommended Range	Notes
Learning Rate	1e-5 to 5e-4	Start with 2e-5 for BERT-like models, 3e-4 for GPT-style
Batch Size	8-64	Use gradient accumulation if memory limited
Number of Epochs	2-10	Early stopping recommended, typically 3-5 epochs
Weight Decay	0.01-0.1	0.01 is common default, increase for overfitting



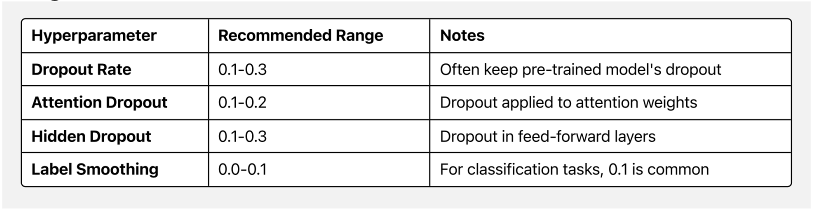
Learning Rate Scheduling

Hyperparameter	Recommended Range	Notes
Warmup Steps	500-2000	Or 6-10% of total training steps
Warmup Ratio	0.06-0.1	Alternative to fixed warmup steps
LR Scheduler	Linear, Cosine	Linear decay most common for fine-tuning
Min LR Factor	0.0-0.1	For cosine annealing, minimum LR as fraction of max

Hyperparameters

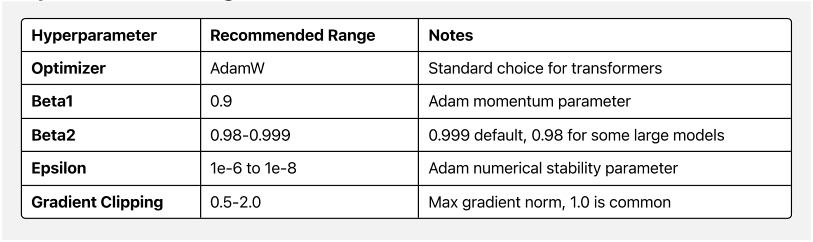
Some proposed settings

Regularization



Hyperparameter	Recommended Range	Notes
Dropout Rate	0.1-0.3	Often keep pre-trained model's dropout
Attention Dropout	0.1-0.2	Dropout applied to attention weights
Hidden Dropout	0.1-0.3	Dropout in feed-forward layers
Label Smoothing	0.0-0.1	For classification tasks, 0.1 is common

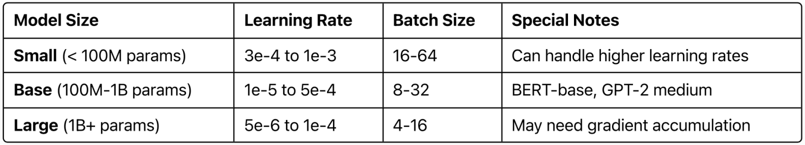
Optimization Settings



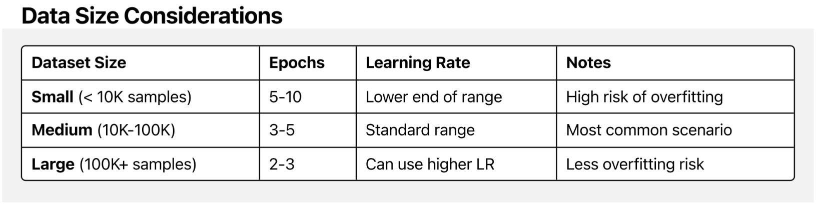
Hyperparameter	Recommended Range	Notes
Optimizer	AdamW	Standard choice for transformers
Beta1	0.9	Adam momentum parameter
Beta2	0.98-0.999	0.999 default, 0.98 for some large models
Epsilon	1e-6 to 1e-8	Adam numerical stability parameter
Gradient Clipping	0.5-2.0	Max gradient norm, 1.0 is common

Hyperparameters

Some proposed settings



Model Size	Learning Rate	Batch Size	Special Notes
Small (< 100M params)	3e-4 to 1e-3	16-64	Can handle higher learning rates
Base (100M-1B params)	1e-5 to 5e-4	8-32	BERT-base, GPT-2 medium
Large (1B+ params)	5e-6 to 1e-4	4-16	May need gradient accumulation



Data Size Considerations

Dataset Size	Epochs	Learning Rate	Notes
Small (< 10K samples)	5-10	Lower end of range	High risk of overfitting
Medium (10K-100K)	3-5	Standard range	Most common scenario
Large (100K+ samples)	2-3	Can use higher LR	Less overfitting risk

Additional topics

Neural net related core topics:

- Weight initializations
- Activation functions
- Optimization functions
- Loss functions
- Normalization
- Regularization / dropout
- Model architectures
- Hyperparameter optimization
- Bayesian neural networks
- Computation graphs
- Software / platforms
- Encoding / adding outside knowledge
- Hardware accelerators

Additional topics

Neural net applied topics:

- Computer vision
- Natural language processing
- Signal processing
- Generative models
- Unsupervised learning
- Reinforcement learning
- One/Zero shot learning
- Transfer learning
- Auto-ML
- Memory Augmented Neural Networks

References

[1] ESL. Chapter 11

[2] Pancha N, Zhai A, Leskovec J, Rosenberg C. PinnerFormer: Sequence Modeling for User Representation at Pinterest. arXiv 2022.