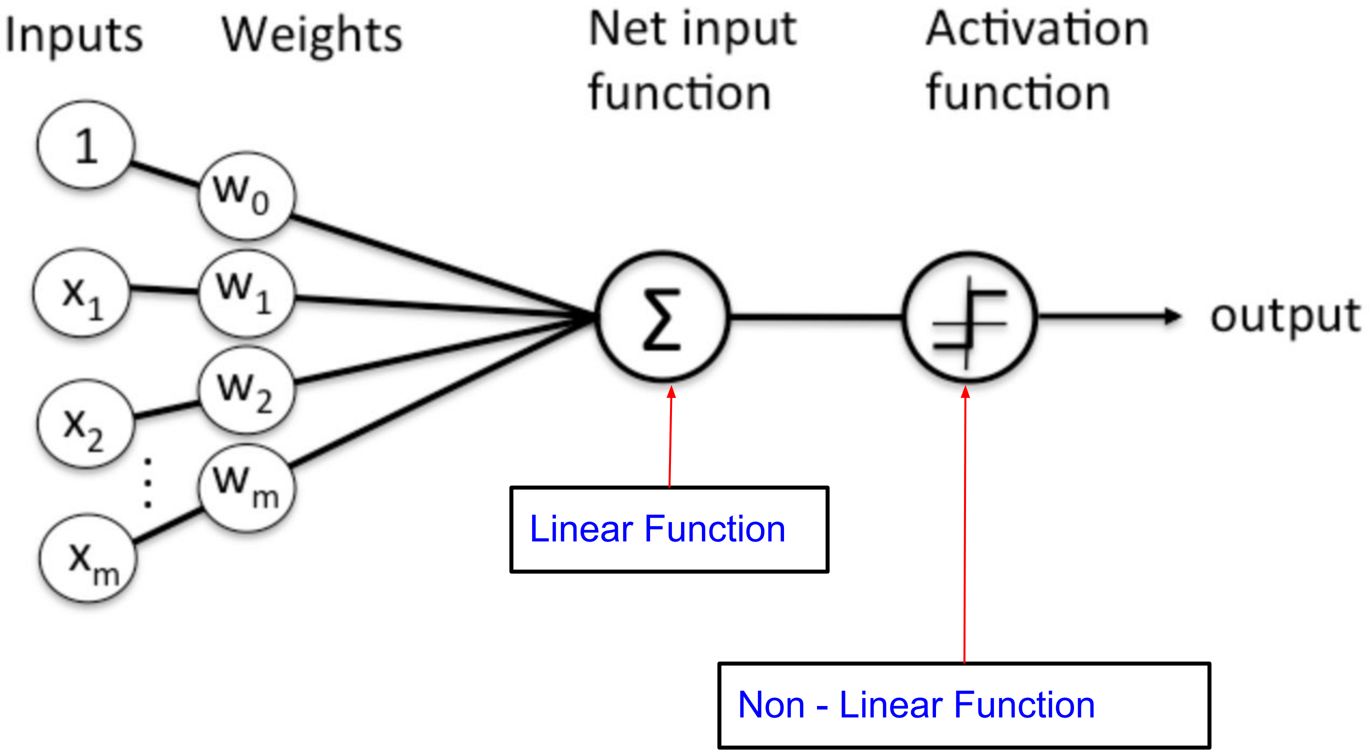
Recommender Systems

E0 259: Data Analytics
Lecture 3

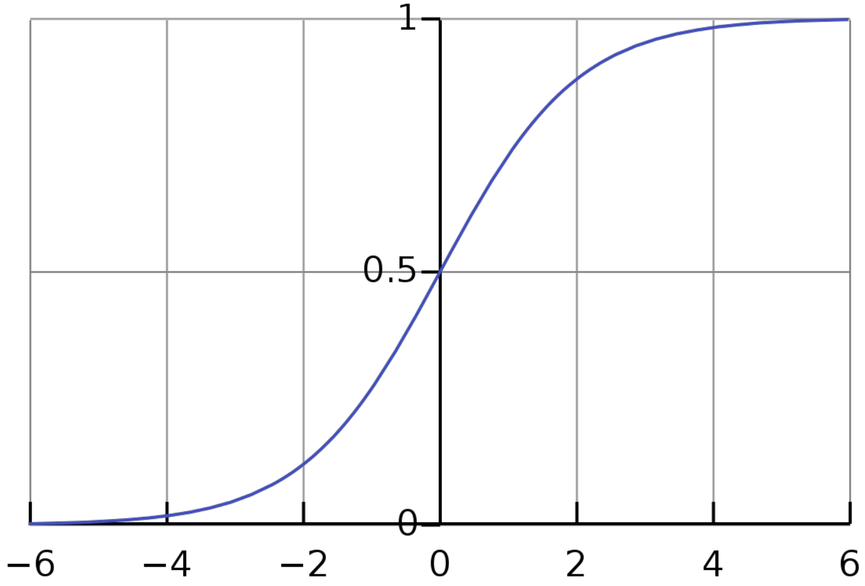
Deep Learning 101



https://wiki.pathmind.com/neural-network

Types of Activation Functions - Sigmoid

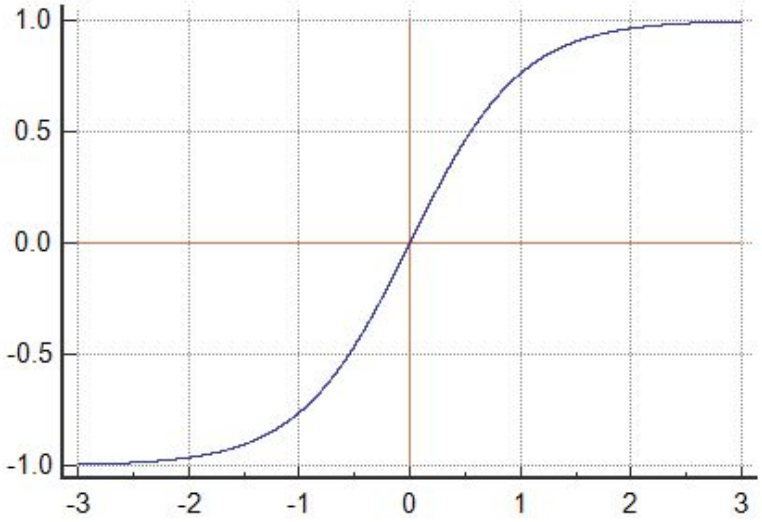
- Goes from 0 to 1.
- Increase weights to make it steeper
- Since it flattens out - when doing gradient descent, if a value becomes 0 or 1, neurons don’t learn.



$$\sigma(z) \equiv 1/(1 + e^{-z})$$

Types of Activation Functions - tanh

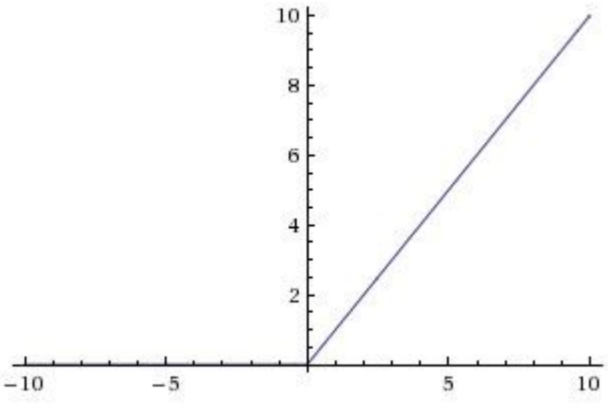
- Goes from -1 to 1.
- Centered around 0, rather 0.5
- Convergence faster



$$\frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

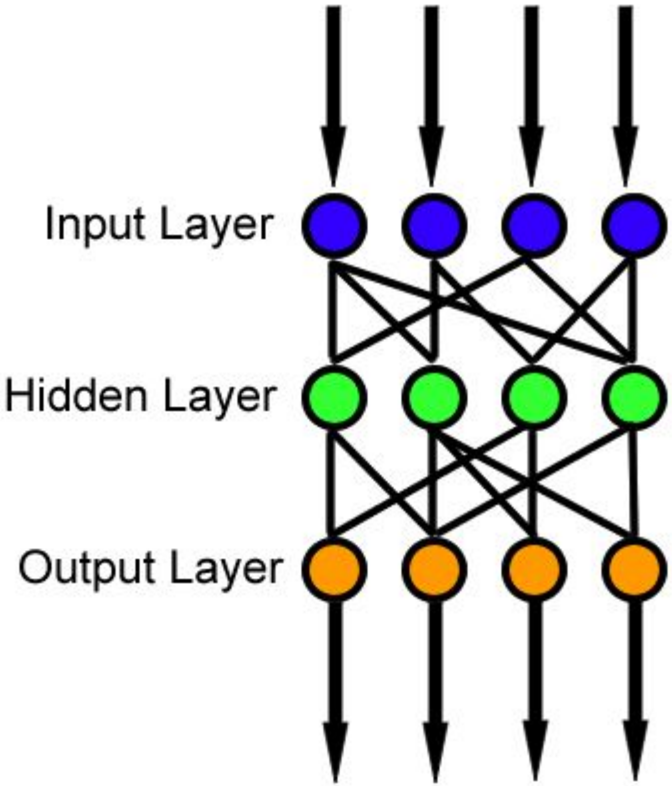
Types of Activation Functions - ReLU

- Goes from 0 to infinity
- In positive side gradient is never vanishes or explodes
- For negative part, can fix with something called leaky ReLU



max(0,x)

Multi Layer Perceptron

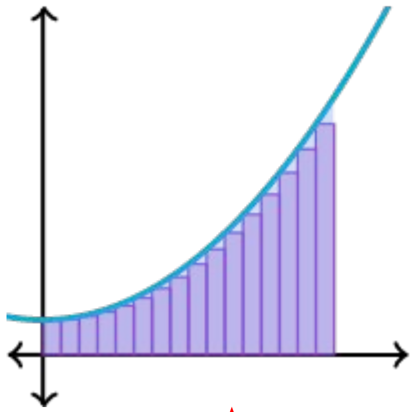


- Stack together multiple neurons
- Output of each circle is linear sum of inputs followed by non linear activation!

https://wiki.pathmind.com/neural-network

Universal Approximation Theorem

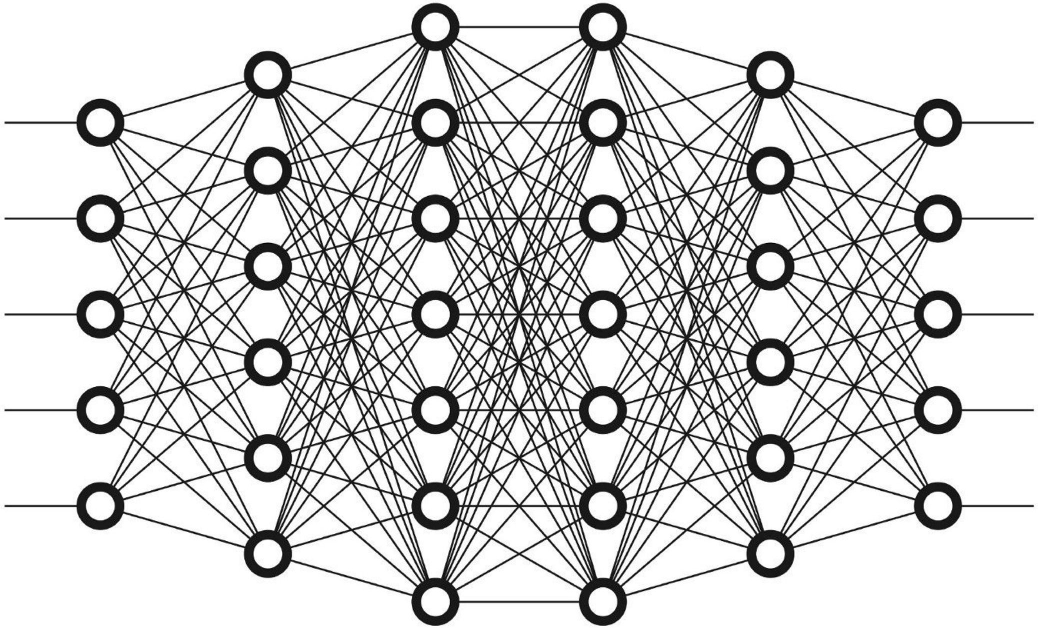
For any arbitrary continuous function f(x), given any ε, there exists a single hidden layer and weights such that the output function of the perceptron g(x) is such that $|f(x) - g(x)| < \varepsilon$, for all x.

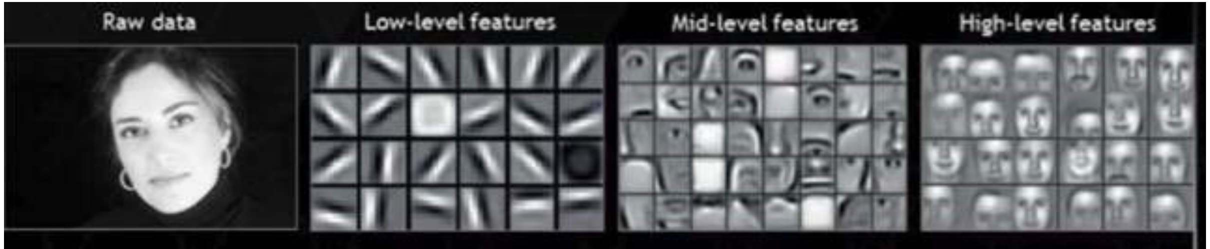


Each rectangle represents 2 hidden neurons.

What is Deep Learning

- Stack lots of hidden layers!
- Can have several billion parameters!
- Why stack, when 1 hidden layer is enough by Universal Approximation Theorem?
- Represent knowledge or feature hierarchies.

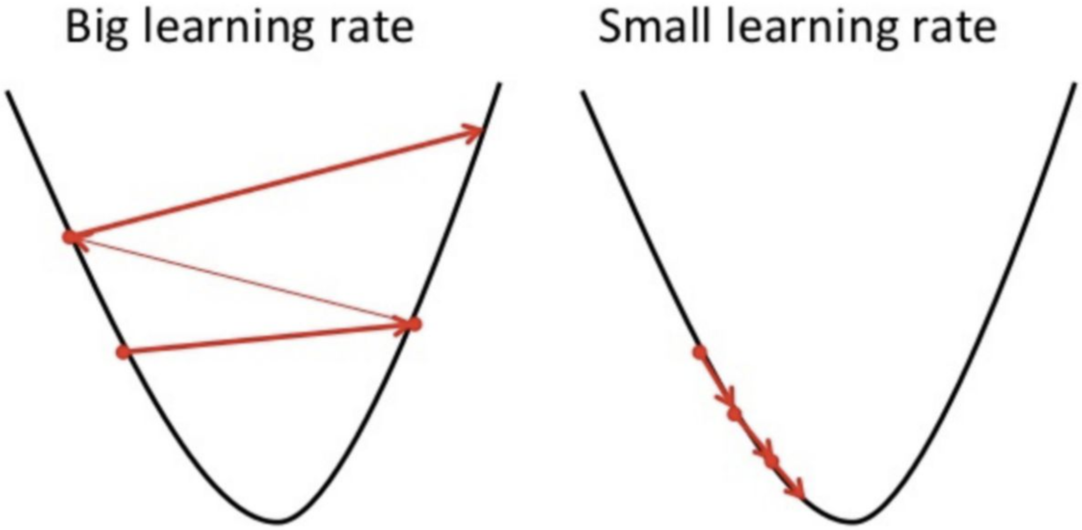




From fast.ai

Learning Rate and Optimizers

- Big steps/learning rate: never converge
- Small steps/learning rate: converge very slowly



Loss Functions

- Cross Entropy for classification tasks
- Mean square error
- Log likelihood
- Hinge loss etc.
- Use gradient descent to solve.
- Batch, stochastic and mini-batch methods

Types of Optimizers

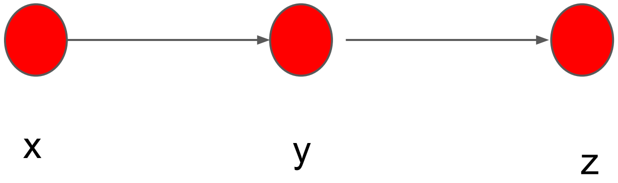
- **Momentum:**
  - to escape local optima.
  - Use some factored weight from previous step - to keep the momentum
- **Nesterov accelerated gradient:**
  - Use momentum term plus correction fact based on where you are going to be.
  - Correction step accounts for very large steps and oscillations

- **Adagrad:**
  - Small learning rate for frequent features
  - Large learning rate for infrequent features
- **RMSprop:**
  - Normalization factor that slows diminishing learning rate of Adagrad
- **Adam:**
  - Trade of between momentum and RMSProp

What Do We Need to Learn and How?

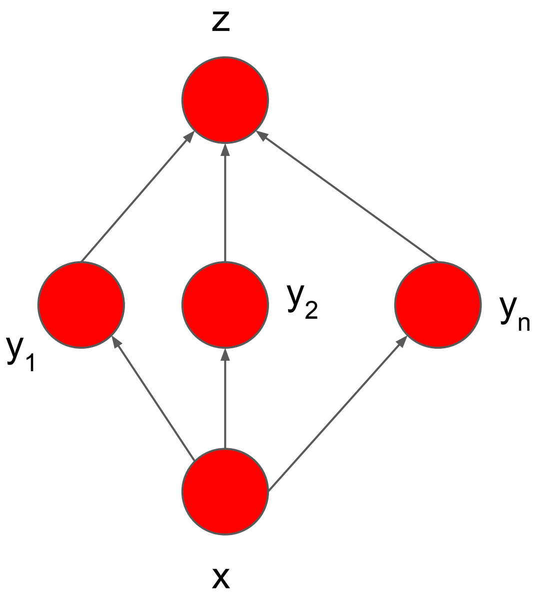
- The Weight matrices for the linear transformation at each layer.
- How Do We Learn This?
- By Stochastic Gradient Descent (several details and nuances here)
- If we have billions of parameters in these matrices, and tons of data, isn’t learning these parameters computationally hard?
- Stochastic Gradient Descent with backpropagation to the rescue!

Chain Rule of Calculus



$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

For Multiple Paths



$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

Forward Propagation

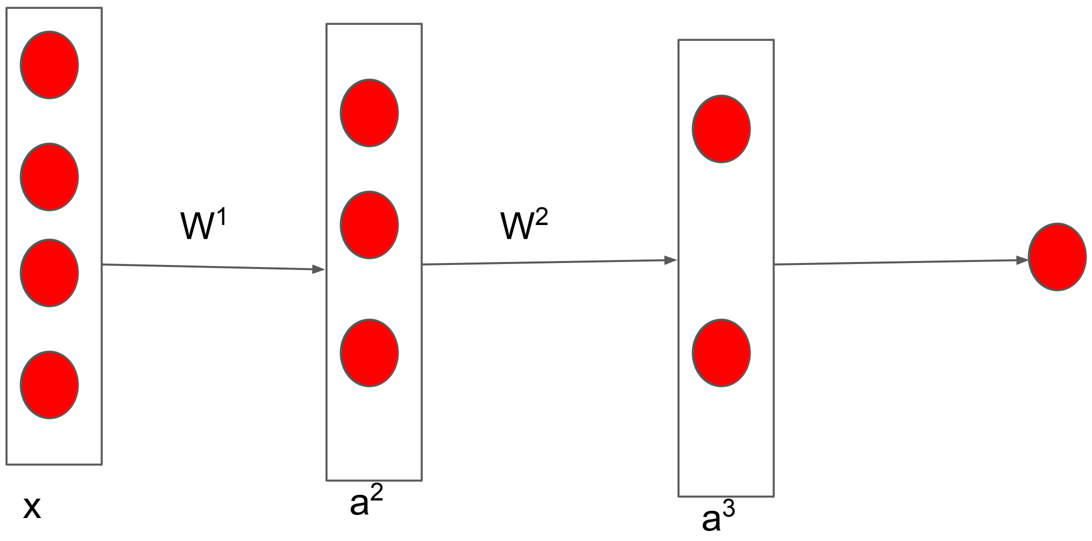
- Let x_t be the t^{th} training sample.
- Let current set of weights at layer l be W_t^l
- Let $\mathbf{a_t^l}$ be values of activation function at l^{th} layer.

Forward Prop (contd)

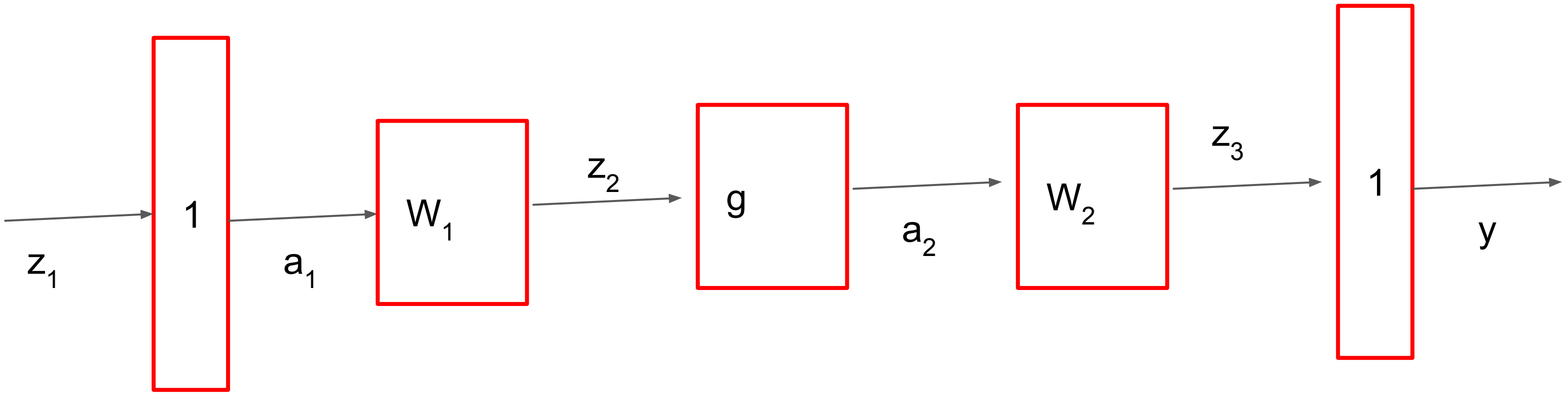
- Let $\mathbf{z_t^l} = \mathbf{W_t^{l-1} a_t^{l-1}}$ be linear transformation at l^{th} layer
- Let $\mathbf{a_t^l} = \mathbf{g(z_t^l)}$, g is non linear activation function
- Let L be the final loss function
- e.g. for regression, $L = \sum_{i=1}^{N} (y_i - a^L)^2$

Backpropagation

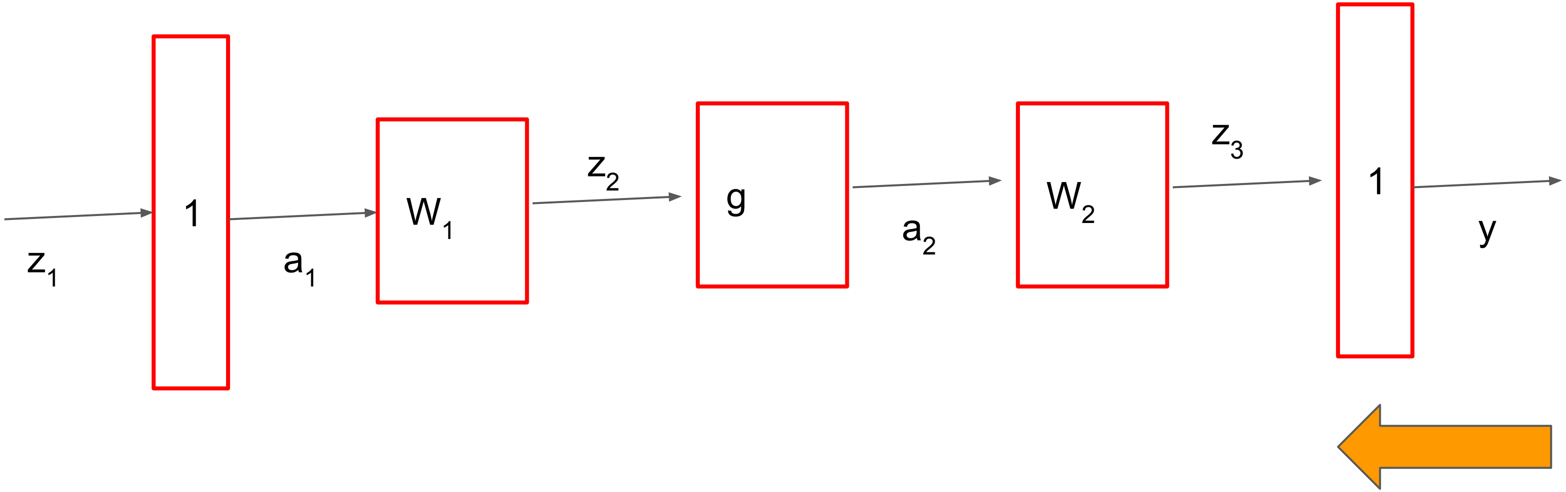
- From this final loss value for the x_t, we want to figure out how to adjust the weights.
- Consider simple 2 layer network



Flow Graph

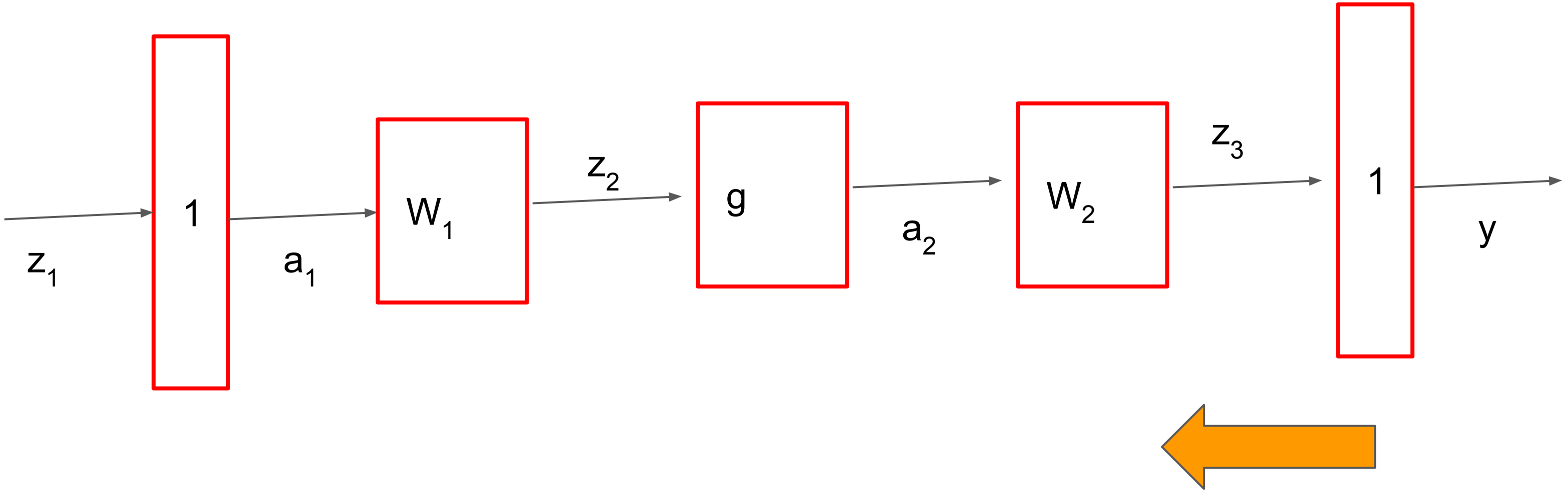


Backprop Step 1



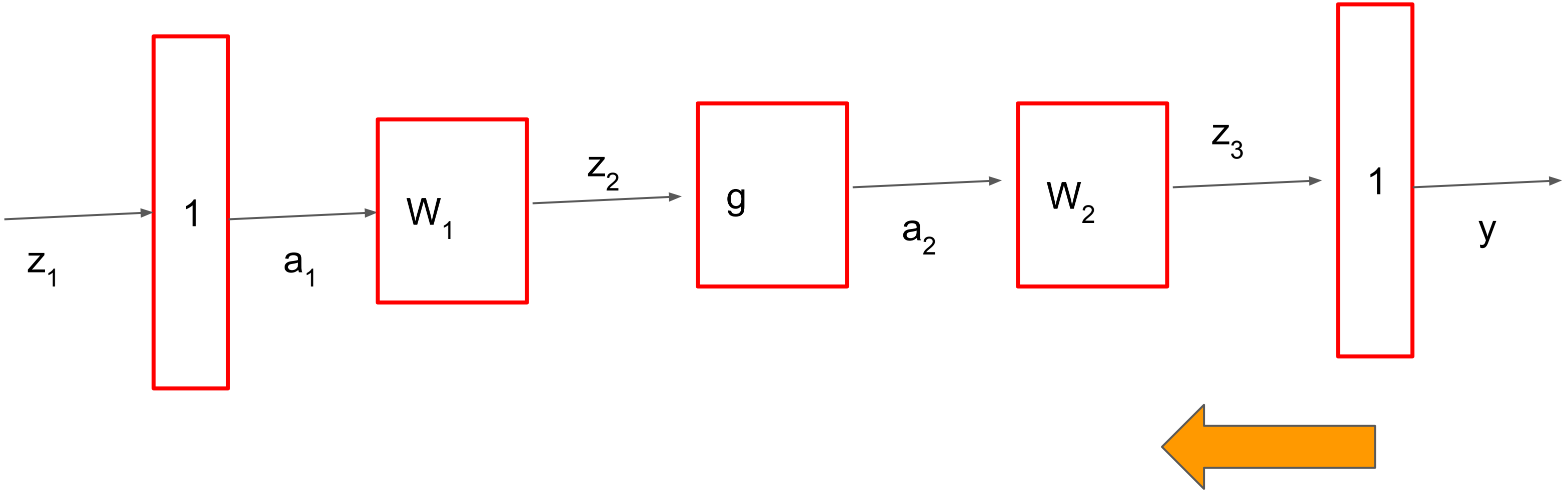
- Assume mean square loss.
- Error $= \delta_3 = 2(\hat{y} - y)$

Backprop Step 2 - Calculate Gradient W.R.T. W_2



$$\text{Gradient w.r.t. } W_2 = \delta_3 a_2^T$$

Backprop Step 2 - Calculate Gradient W.R.T. a_2



Gradient w.r.t. a_2 is $\delta_2 = g'(z_2) \circ \delta_3 a_2^T$

In General

- $\frac{\nabla \mathbf{L}}{\nabla \mathbf{x}} = \frac{\nabla \mathbf{L}}{\nabla \mathbf{a^L}} \frac{\nabla \mathbf{a^L}}{\nabla \mathbf{z^L}} \frac{\nabla \mathbf{z^L}}{\nabla \mathbf{a^{L-1}}} \cdots \frac{\nabla \mathbf{z^1}}{\nabla \mathbf{a^1}} \frac{\nabla \mathbf{a^1}}{\nabla \mathbf{x}}$

- $\frac{\nabla \mathbf{a^l}}{\nabla \mathbf{z^l}} = \mathbf{g'(z^l)}$

- $\frac{\nabla \mathbf{z^l}}{\nabla \mathbf{a^{l-1}}} = \mathbf{W^{l-1}}$

- $\frac{\nabla \mathbf{L}}{\nabla \mathbf{x}} = \frac{\nabla \mathbf{L}}{\nabla \mathbf{a^L}} \mathbf{g'(z^L)}.\mathbf{W^{L-1}} \ldots \mathbf{W^1 g'(z^1)}$

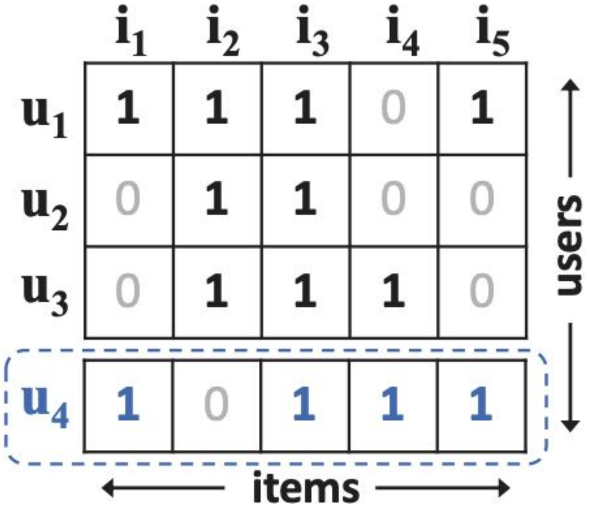
Backpropagation

- Let error at level l be $\delta^{\mathbf{l}}$
- $\delta^{\mathbf{L}} = \frac{\nabla \mathbf{L}}{\nabla \mathbf{a}^{\mathbf{L}}}$
- $\delta^{\mathbf{L-1}} = \delta^{\mathbf{L}} \mathbf{g}'(\mathbf{z}^{\mathbf{L}}) \mathbf{W}^{\mathbf{L-1}}$
- $\delta^{\mathbf{l}} = \delta^{\mathbf{l+1}} \mathbf{g}'(\mathbf{z}^{\mathbf{l+1}}) \mathbf{W}^{\mathbf{l}}$

Neural Collaborative Filtering

- Most often we do not have explicit ratings.
- We only have implicit feedback based on user item interactions (user watches a movie, buys a book etc.)
- This is a 0-1 signal.
- Matrix factorization approach models only simple linear interactions over this

Example User Item Matrix (WWW 2017 - He et. al)



(a) user–item matrix

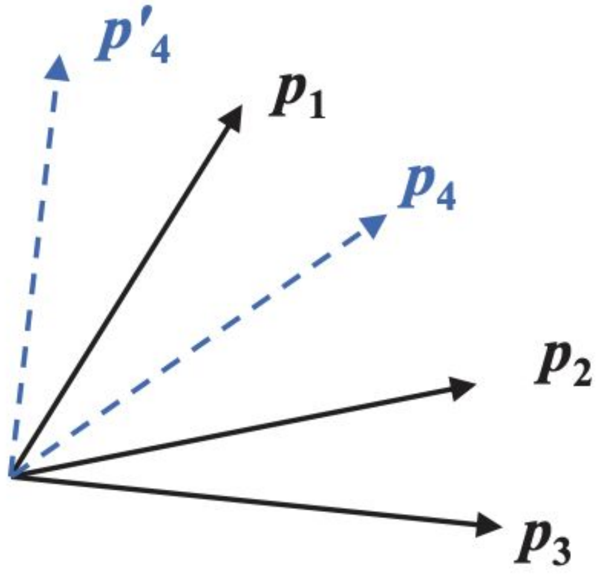
- With implicit ratings, Jaccard similarity good proxy for cosine similarity

- sim(1, 2) = $\frac{1}{2}$, , sim(1,3) = $\frac{2}{5}$, sim(2,3) = $\frac{2}{3}$

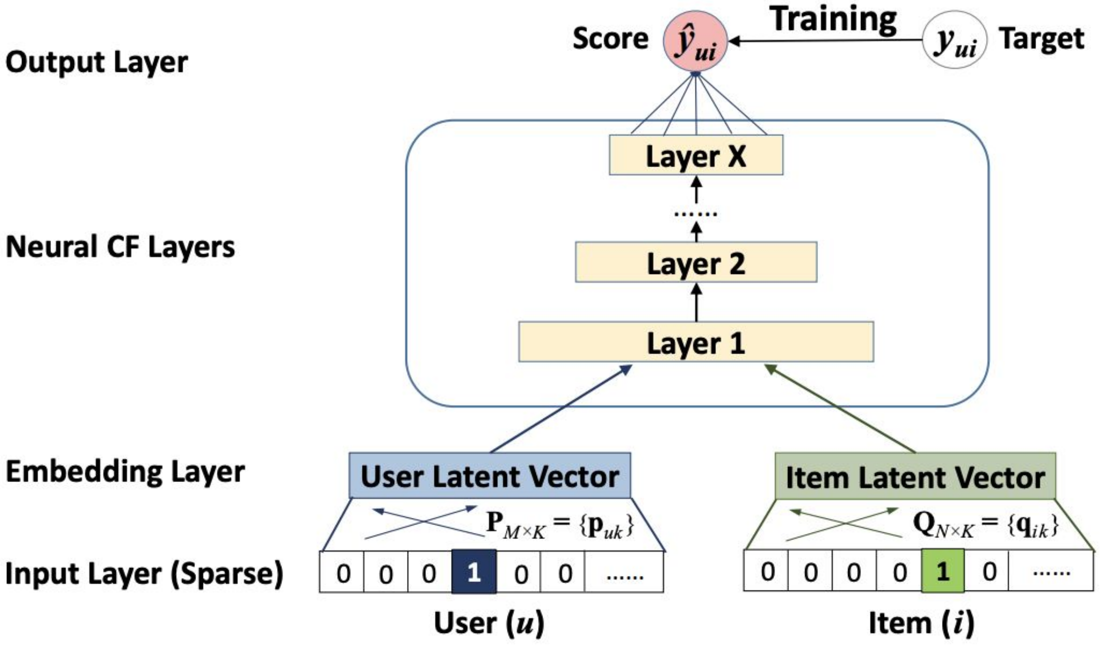
- Consider new user 4, sim(1,4) = 3/5 > sim(3,4) = $\frac{2}{5}$ > sim(2,4) = 1/5

Matrix Factorization

- Linear Transformation
- Suppose we factorize to 2 factors.
- No matter where we place user 4, she is closer to 2 than 3!!!:(



Neural Collaborative Filtering



NCF Model

$$\hat{y}_{ui} = f(\mathbf{P}^T \mathbf{v}_u^U, \mathbf{Q}^T \mathbf{v}_i^I | \mathbf{P}, \mathbf{Q}, \Theta_f),$$

$$\mathbf{P} \in \mathbb{R}^{M \times K}$$

M users, K factors

$$\Theta_f$$

Parameters of Neural Layers

$$\mathbf{Q} \in \mathbb{R}^{N \times K}$$

N items, K factors

Loss Function for NCF Model

- Let **Y** be set of all possible interactions between users and items.
- $\mathbf{Y}^{+}$ be set of observed interactions between users and items
- $\mathbf{Y}^{-}$ be set of non interactions between users and items
- Since interactions are binary, we want the predicted value to take a value between 0 and 1.
- Can try to model the probability distribution of interactions

Loss Function for NCF Model

$$p(\mathcal{Y}, \mathcal{Y}^- | \mathbf{P}, \mathbf{Q}, \Theta_f) = \prod_{(u,i)\in\mathcal{Y}} \hat{y}_{ui} \prod_{(u,j)\in\mathcal{Y}^-} (1 - \hat{y}_{uj}).$$

- Use log likelihood

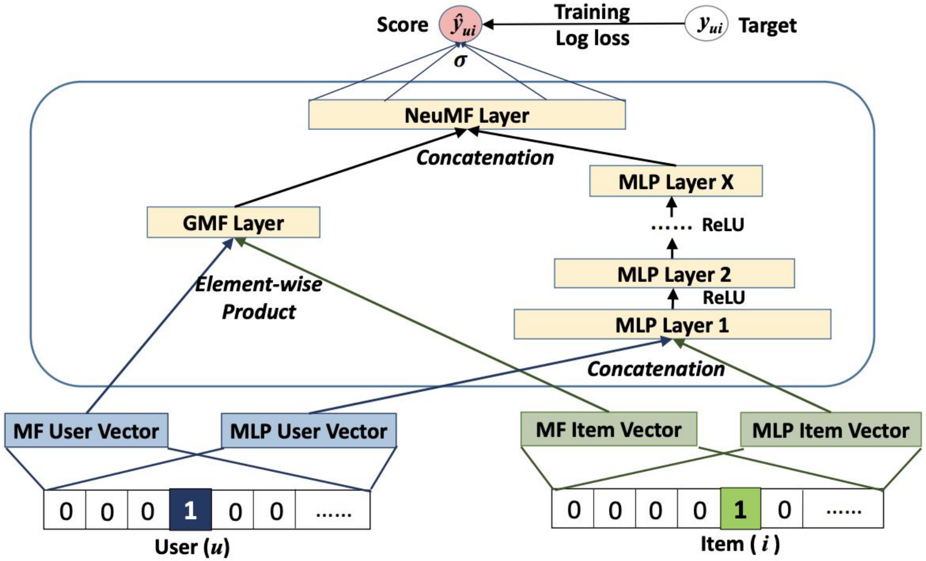
$$\begin{aligned} L &= -\sum_{(u,i)\in\mathcal{Y}} \log \hat{y}_{ui} - \sum_{(u,j)\in\mathcal{Y}^-} \log(1 - \hat{y}_{uj}) \\ &= -\sum_{(u,i)\in\mathcal{Y}\cup\mathcal{Y}^-} y_{ui} \log \hat{y}_{ui} + (1 - y_{ui}) \log(1 - \hat{y}_{ui}). \end{aligned}$$

NCF Generalizes Matrix Factorization

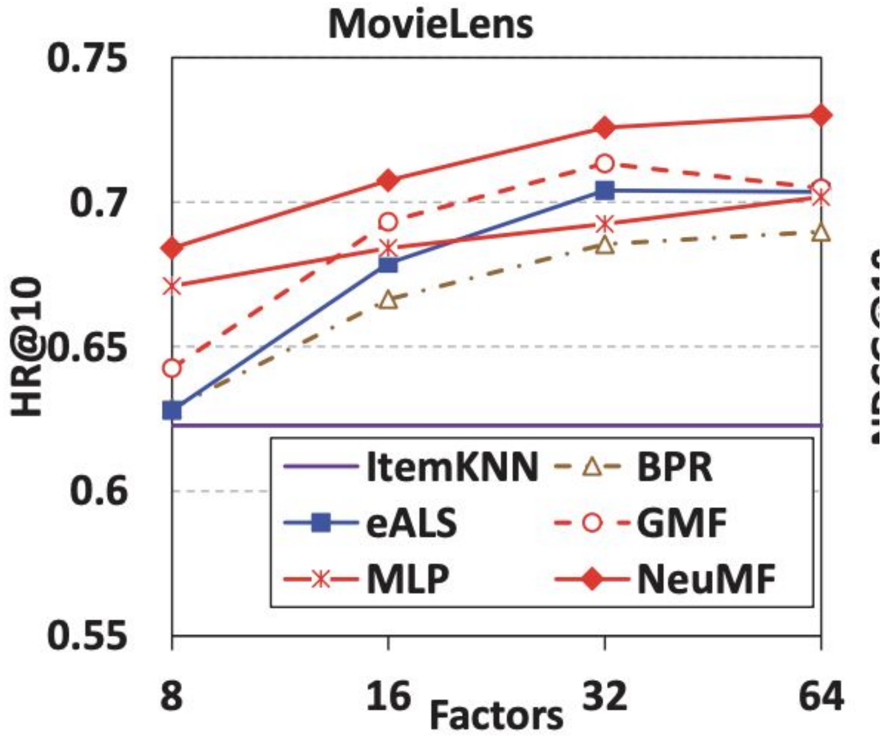
- Make the Neural Layer and identity layer. Then same as MF.

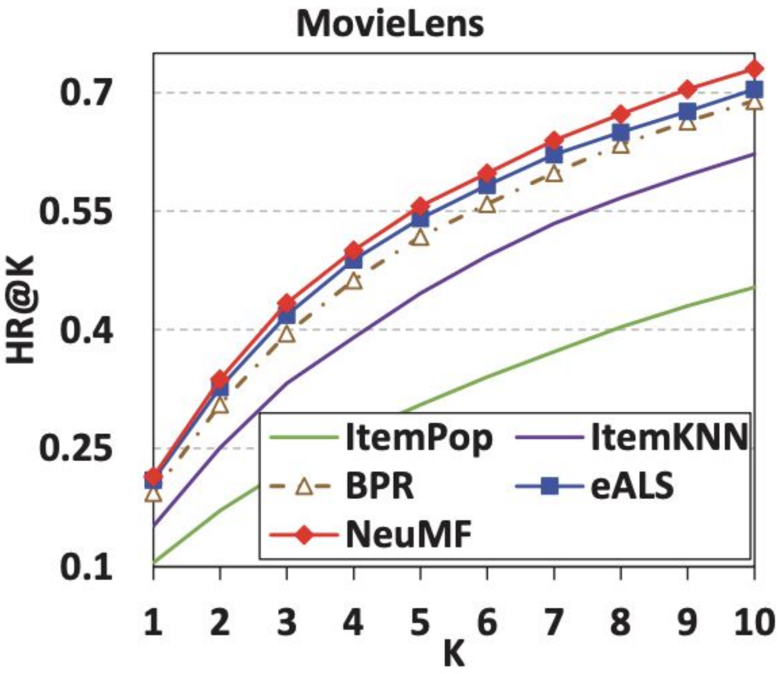
Flexible Model - Fusion of Matrix Factorization and and MLP

- Shared embeddings limits flexibility
- Doesn't allow for each embedding dimension to be different



Performance of Neural Collaborative Filtering





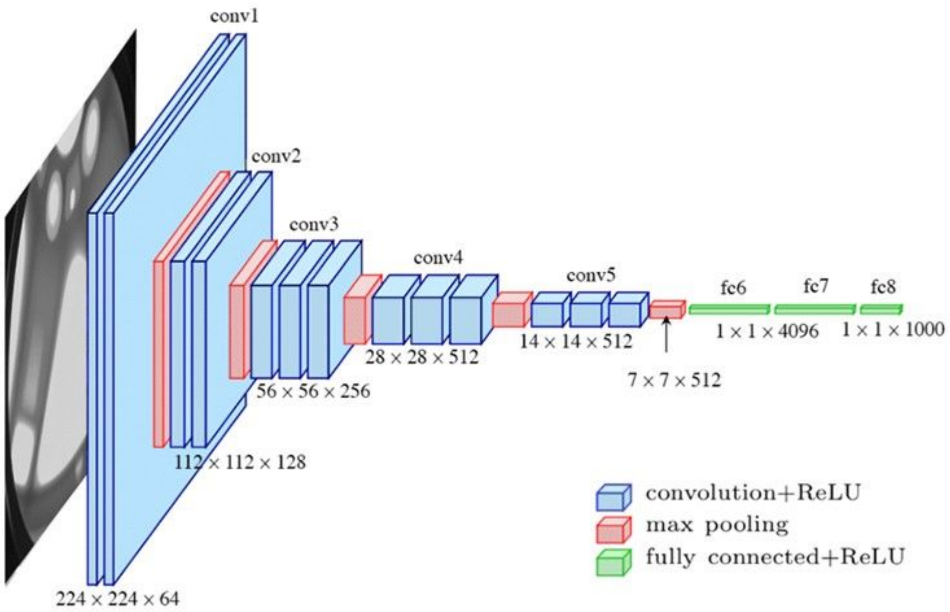
How Does/Did Spotify Solve Cold Start? - CNNs

1	0	1	2	3
5	6	3	4	2
6	3	6	1	2
4	6	3	1	3
4	2	6	4	9

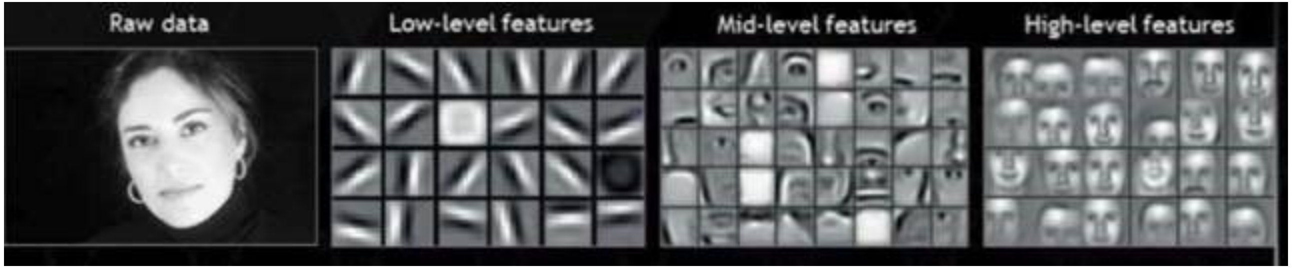
1	2
3	4

Kernel

- Linear Transform: Slide Kernel across image.
- Multiply and add original matrix values and kernel values. E.g.: 1*1 + 0*2 + 5*3 + 6*4 = 40
- Apply non linearity - on pixel value - also consider neighboring values for smoothing (pooling)
- Have different kernels and multiple layers

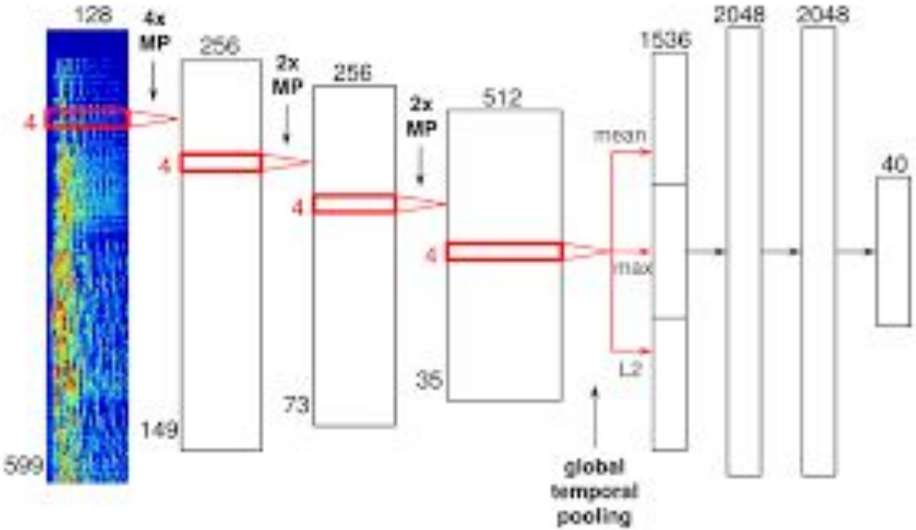


VGG Net



From fast.ai

Spotify Recommendation System



- Take audio gram
- Apply CNN on top of it
- Do vector similarity search