# Agenda

- 5:30 - Networking
- 6:00 - Talk
- 6:30 - Discussion and Networking

WiFi

- Name: RokkinCat Guest
- Password: makingstuff

Bathroom

- Take a bathroom key and go down a floor and the bathrooms are at the end of the hall

# Learning to use Keras to Analyze and Classify Newspaper Articles

## Milwaukee Machine Learning Meetup

Mitchell Henke / @MitchellHenke

# About Me/This Meetup

- Software Architect
- Specialize in data, databases, APIs
- Self-taught R, Python/Keras, Machine Learning
  - Jeremy Howard - http://course.fast.ai
  - Watch talks, read papers
  - Experiment

You Can Learn Machine Learning
(Yes, You)

# NEURAL NETS IN 60 SECONDS

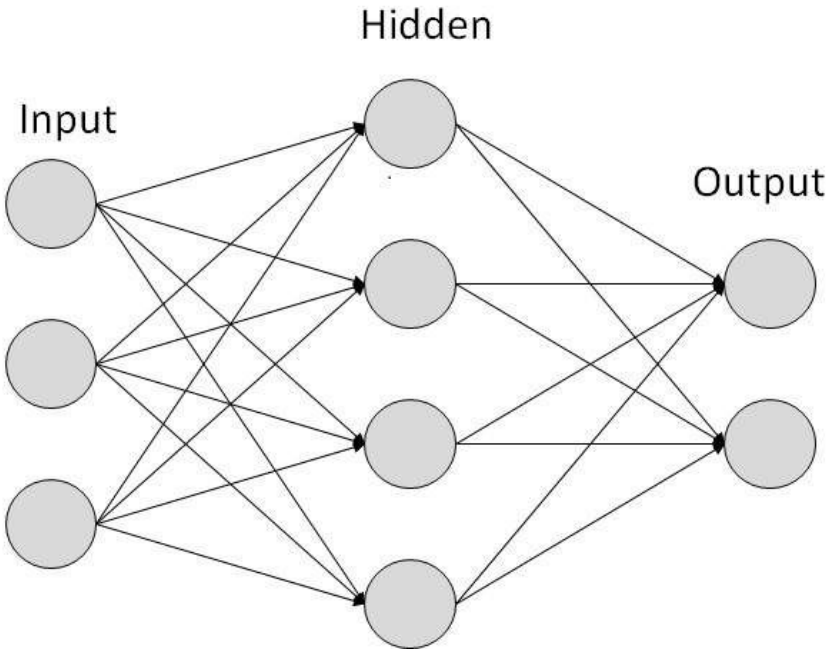
# History

- Initial ideas in 1940s
- Improvements in 1950s-1990s
- 2000s, 2010s a time of huge and fast improvements in deep learning

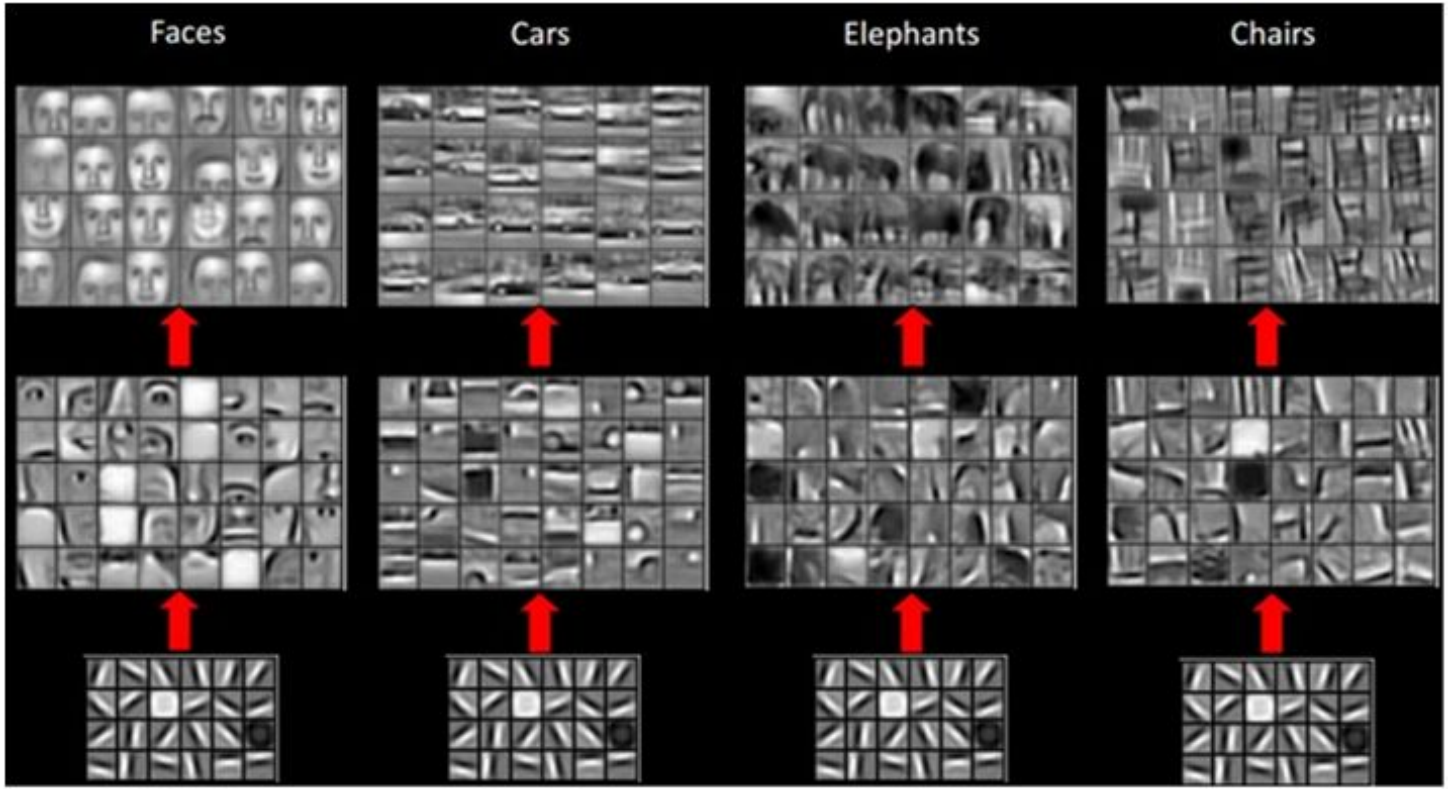
Geoffrey Hinton

# How It Works

- Compare model output to expected output
- Automatically adjust internal parameters
- Repeat



# Convolutional Neural Networks



# Text Classification with Keras

# Keras

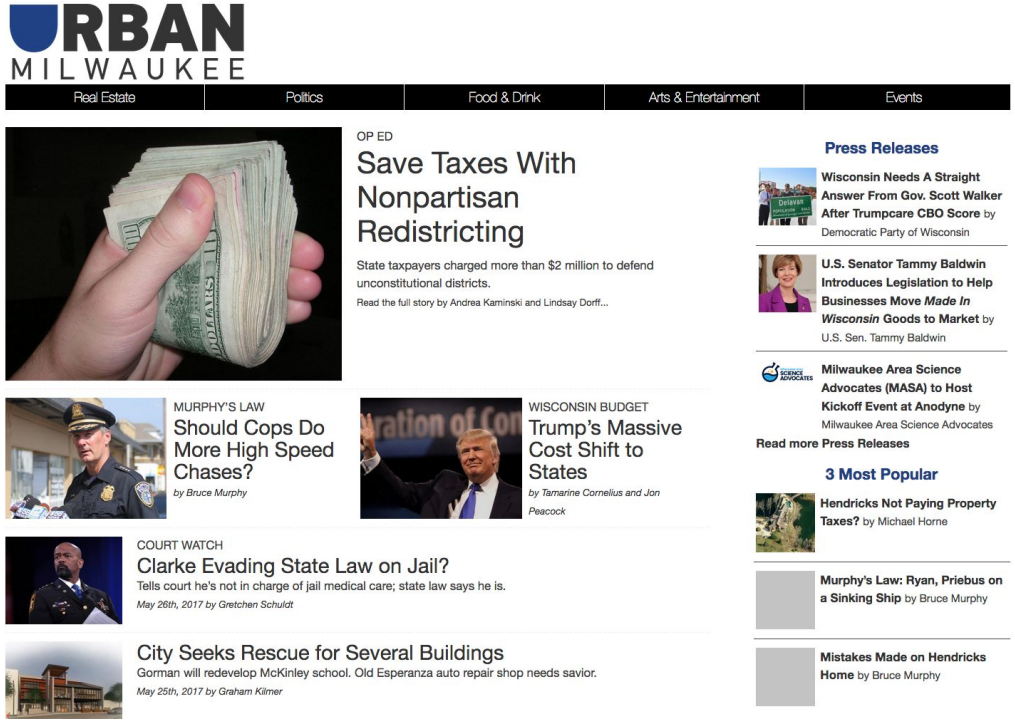
- Python
- Higher level abstraction over Theano/TensorFlow
- CPU or GPU
- Everything is a layer
- Deploy to mobile phones!

# Goal and Process

- Goal: Classify text by author
  - Metric: Correct author classifications
- Process
  - Prepare Data
  - Build Model
  - Test and Evaluate
  - Modify
  - Repeat

# Data

- Over 2,500 articles
- ~150-200 words in the first paragraph
- 4 Authors
  - Dave Reid - 826
  - Jeramey Jannene - 677
  - Michael Horne - 535
  - Bruce Murphy - 524



# Data Processing and Preparation

- Standardize format
  - Tokenize
  - Drop punctuation
- Turn input paragraph string into a list of numbers
  - Pad with zeros

```
import pandas as pd
import numpy as np
data =
pd.read_csv(‘data/urbanmilwaukee/data.csv’)

tokenizer = Tokenizer(nb_words=25000)
tokenizer.fit_on_texts(data.p1)

# turn “What is this?” into [“what”, “is”, this”]
# then [“what”, “is”, “this”] into [2, 23, 70]

sequences = tokenizer.texts_to_sequences(data.p1)
word_index = tokenizer.word_index
padded_data = pad_sequences(sequences,
maxlen=233)
```

# Data Processing and Preparation

- Make output labels into a list of numbers
  - Jeramey Jannene -> [0 0 0 1]
  - Bruce Murphy -> [1 0 0 0]
- “This week in Milwaukee… ”, “Bruce Murphy”
- [[0, 0, 0, 0, 24, 37, 3540, …], [1, 0, 0, 0]]
- Using numbers instead of letters makes the math a lot easier

```
labels_index = dict(((val, idx) for idx,
val in enumerate(set(data.author))))

labels_numbers = [labels_index[x] for x in
data.author]
labels =
to_categorical(np.asarray(labels_numbers))
```

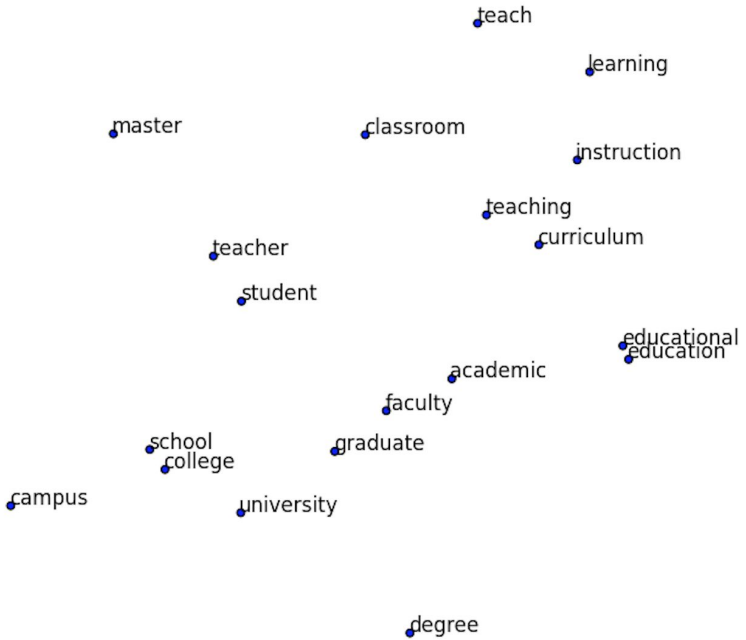
# Data Processing and Preparation

- Shuffle and split data up
- Separate inputs and outputs
- Train on 80%, validate on other 20%
- Data is now ready to be used in a neural net (sort of!)

```
indices = np.arange(data.shape[0])
np.random.shuffle(indices)
padded_data = padded_data[indices]
labels = labels[indices]

nb_validation_samples = int(0.2 * padded_data.shape[0])

x_train = padded_data[:-nb_validation_samples]
y_train = labels[:-nb_validation_samples]
x_val = padded_data[-nb_validation_samples:]
y_val = labels[-nb_validation_samples:]
```

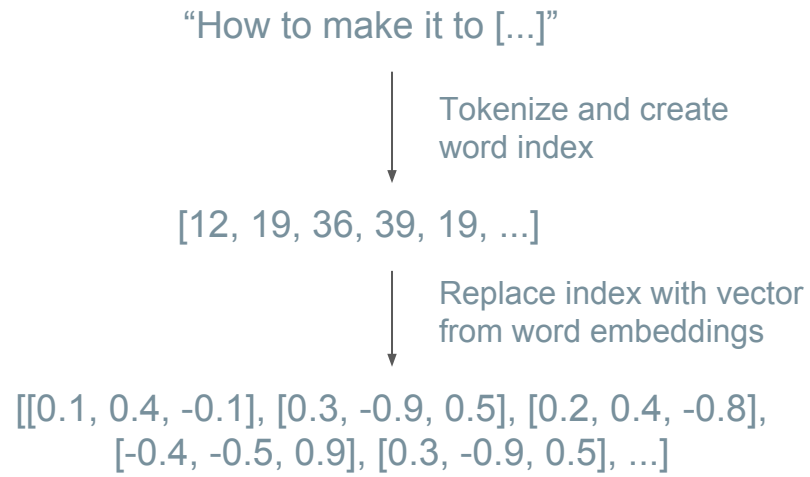
# Word Embeddings

- Express words as high dimensional, numeric vectors
- GloVe (Stanford), Word2vec (Google)
- Similar words near each other in vector space
- “King” - “Man” + “Woman” ≈ “Queen”



# Word Embeddings

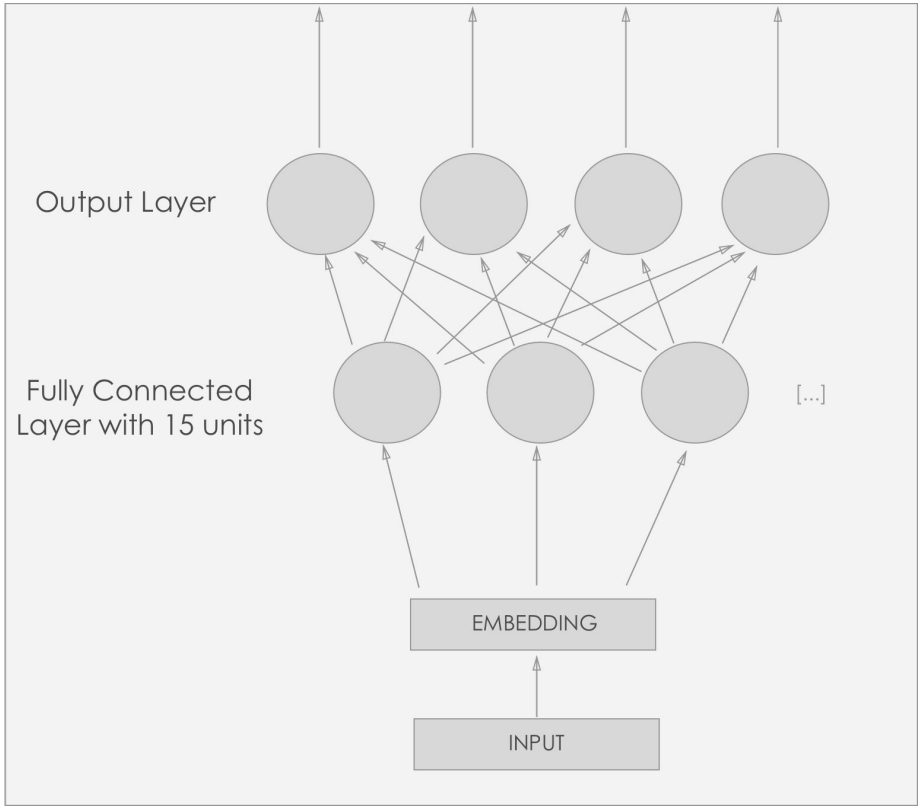
```
EMBEDDING_DIM = 100

embeddings_index = {}
f = open('glove.6B.100d.txt')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

embedding_matrix = np.zeros((len(word_index) + 1,
EMBEDDING_DIM))

for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # set words not in embedding index to zeros
        embedding_matrix[i] = embedding_vector
```



# The First Neural Net

```
model = Sequential()
model.add(Embedding(len(word_index) + 1,
                            EMBEDDING_DIM,
                            input_length=SEQUENCE_LENGTH,
                           weights=[embedding_matrix],
                            trainable=True))
model.add(Flatten())
model.add(Dense(15, activation='relu'))
model.add(Dense(4, activation='softmax'))

model.compile(loss='categorical_crossentropy',
           optimizer=Adam(),
           metrics=['acc'])

model.fit(x_train, y_train, validation_data=(x_val, y_val), nb_epoch=20, batch_size=128)
```

# The First Neural Net



# The First Neural Net

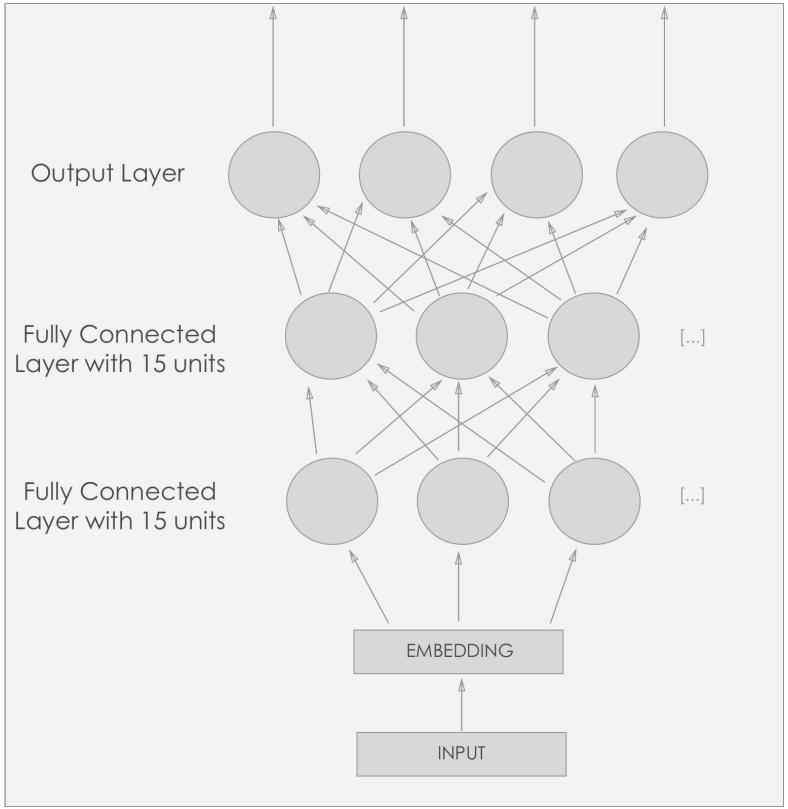
```
Trainable params: 1,738,979

Train on 2050 samples, validate on 512 samples
Epoch 1/20
2050/2050 [====================] - 0s - loss: 1.2801 - acc: 0.3937 - val_loss: 1.1943 - val_acc: 0.4062
Epoch 2/20
2050/2050 [====================] - 0s - loss: 0.9904 - acc: 0.5459 - val_loss: 1.1395 - val_acc: 0.4883
...
Epoch 17/20
2050/2050 [====================] - 1s - loss: 0.0099 - acc: 1.0000 - val_loss: 0.8048 - val_acc: 0.6660
...
```

# Deeper Neural Net

```
model = Sequential()
model.add(Embedding(len(word_index) + 1,
                            EMBEDDING_DIM,
                            input_length=SEQUENCE_LENGTH,
                           weights=[embedding_matrix],
                            trainable=True))
model.add(Flatten())
model.add(Dense(15, activation='relu'))
# Second hidden layer
model.add(Dense(15, activation='relu'))

model.add(Dense(4, activation='softmax'))

model.compile(loss='categorical_crossentropy',
           optimizer=Adam(),
           metrics=['acc'])

model.fit(x_train, y_train, validation_data=(x_val, y_val), nb_epoch=20, batch_size=128)
```

# Deeper Neural Net

New Layer



# Deeper Neural Net

```
Trainable params: 1,739,219

Train on 2050 samples, validate on 512 samples
Epoch 1/20
2050/2050 [==================] - 0s - loss: 1.2485 - acc: 0.3966 - val_loss: 1.1696 - val_acc: 0.4707
Epoch 2/20
2050/2050 [==================] - 0s - loss: 0.9668 - acc: 0.5771 - val_loss: 1.0887 - val_acc: 0.5215
...
Epoch 11/20
2050/2050 [==================] - 0s - loss: 0.0574 - acc: 1.0000 - val_loss: 1.0820 - val_acc: 0.5859
Epoch 12/20
2050/2050 [==================] - 0s - loss: 0.0411 - acc: 1.0000 - val_loss: 1.0929 - val_acc: 0.5918
...
```

# OVERFITTING

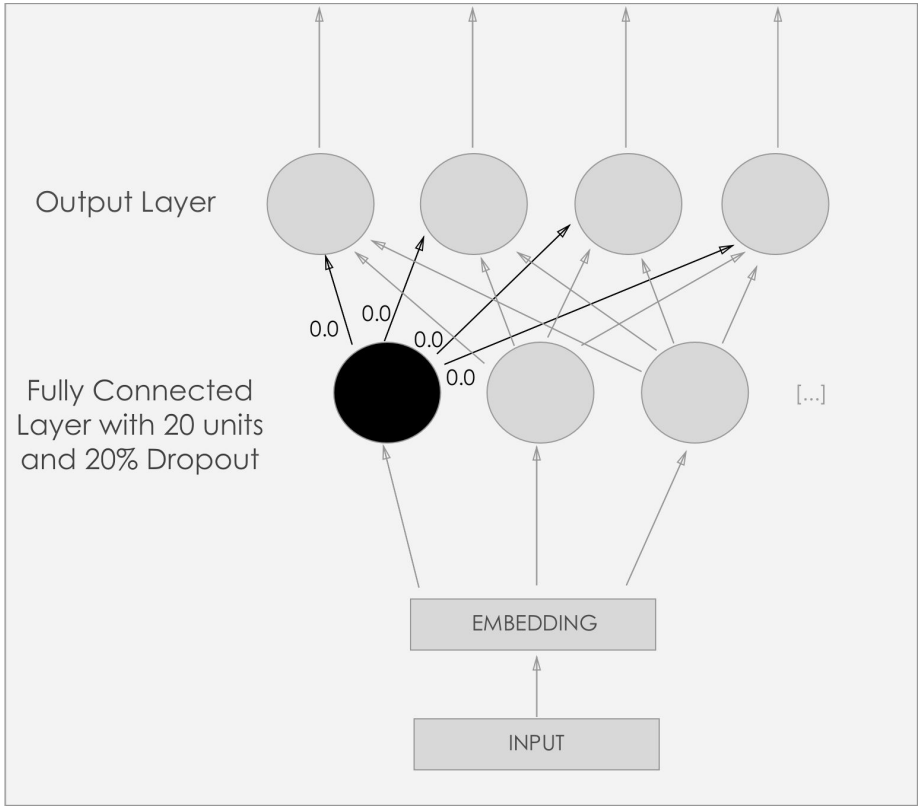
Solutions:

- Get more data
- Simplify architecture
- Regularization

# Neural Net with Regularization

```
model = Sequential()
model.add(Embedding(len(word_index) + 1,
                            EMBEDDING_DIM,
                            weights=[embedding_matrix],
                            input_length=SEQUENCE_LENGTH,
                            trainable=True))
model.add(Flatten())
model.add(Dense(20, activation='relu'))
# Add dropout
model.add(Dropout(0.2))
model.add(Dense(4, activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(),
              metrics=['acc'])

model.fit(x_train, y_train, validation_data=(x_val, y_val),
nb_epoch=20, batch_size=128)
```

# Neural Net with Regularization



# Neural Net with Regularization

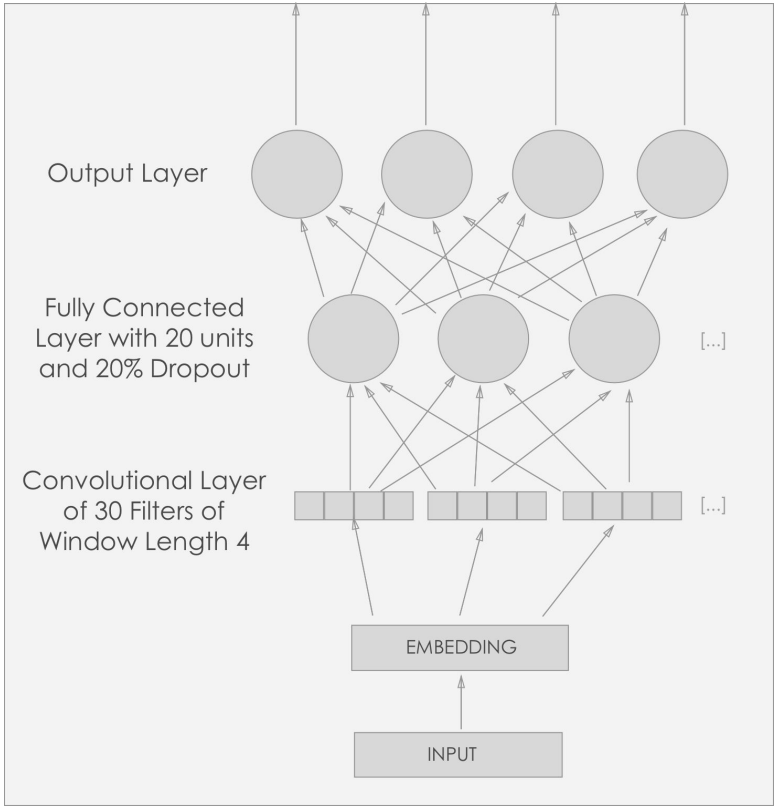
```
Trainable params: 1,738,979

Train on 2050 samples, validate on 512 samples
Epoch 1/25
2050/2050 [==============================] - 1s - loss: 1.3681 - acc: 0.3122 - val_loss: 1.3328 - val_acc: 0.3672
Epoch 2/25
2050/2050 [==============================] - 1s - loss: 1.2986 - acc: 0.3532 - val_loss: 1.2801 - val_acc: 0.3535
...
Epoch 12/25
2050/2050 [==============================] - 1s - loss: 0.0404 - acc: 0.9980 - val_loss: 0.8323 - val_acc: 0.6562

...
Epoch 23/25
2050/2050 [==============================] - 1s - loss: 0.0112 - acc: 0.9990 - val_loss: 0.8286 - val_acc: 0.6914
Epoch 24/25
2050/2050 [==============================] - 1s - loss: 0.0103 - acc: 0.9990 - val_loss: 0.8483 - val_acc: 0.6816
Epoch 25/25
2050/2050 [==============================] - 1s - loss: 0.0087 - acc: 1.0000 - val_loss: 0.8681 - val_acc: 0.6777
```

# Convolutional Neural Net

```
model = Sequential()
model.add(Embedding(len(word_index) + 1,
                    EMBEDDING_DIM,
                    weights=[embedding_matrix],
                    input_length=SEQUENCE_LENGTH,
                    trainable=True))
model.add(Conv1D(30,4, activation='relu'))
model.add(Flatten())
model.add(Dense(20, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(4, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['acc'])

model.fit(x_train, y_train, validation_data=(x_val, y_val),
nb_epoch=25, batch_size=128)
```

# Convolutional Neural Network

New Layer



# Convolutional Neural Net

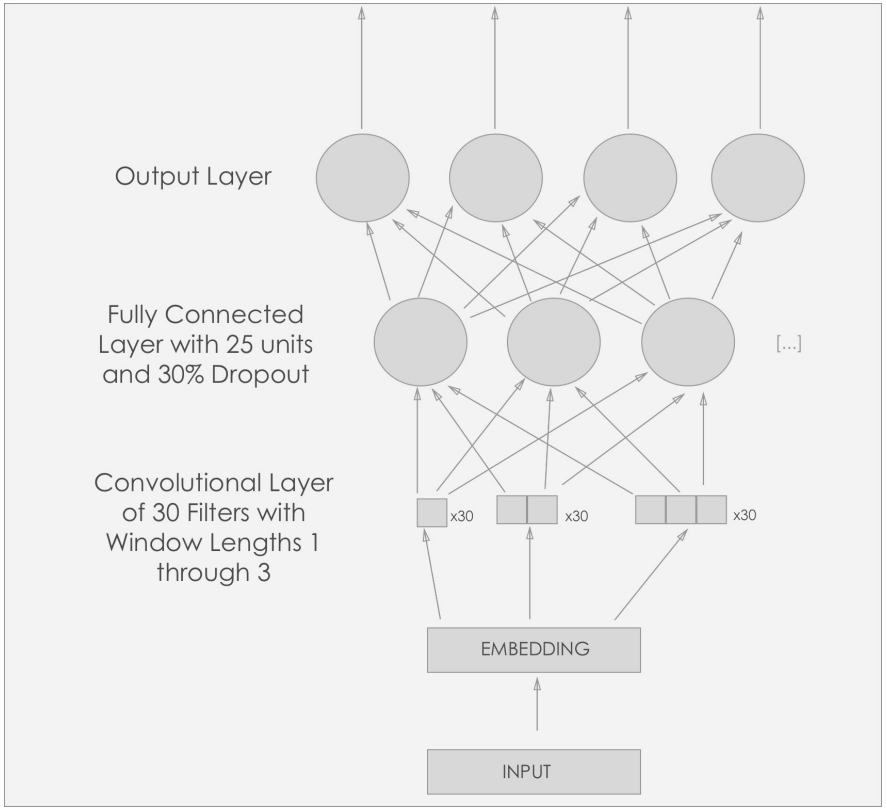
```
Trainable params: 1,539,534

Train on 2050 samples, validate on 512 samples
Epoch 1/25
2050/2050 [==================] - 4s - loss: 1.3299 - acc: 0.3376 - val_loss: 1.2476 - val_acc: 0.5137
...
Epoch 19/25
2050/2050 [==================] - 4s - loss: 0.1081 - acc: 0.9683 - val_loss: 0.9736 - val_acc: 0.6895
Epoch 20/25
2050/2050 [==================] - 4s - loss: 0.1069 - acc: 0.9629 - val_loss: 0.9258 - val_acc: 0.7148
...
```

# Complex Convolutional Neural Net

```
graph_in = Input(shape=(233, EMBEDDING_DIM))
convs = []
for fsz in range(1, 4):
      conv = Convolution1D(nb_filter=30, filter_length=fsz, border_mode='valid', activation='relu')(graph_in)
      conv = MaxPooling1D(pool_length=2)(conv)
      flatten = Flatten()(conv)
      convs.append(flatten)

out = Merge(mode='concat')(convs)
graph = Model(input=graph_in, output=out)
model = Sequential()
model.add(Embedding(len(word_index) + 1,EMBEDDING_DIM, weights=[embedding_matrix],
input_length=SEQUENCE_LENGTH,
                            trainable=True))
model.add(graph)
model.add(Dense(25, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(4, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])
model.fit(x_train, y_train, validation_data=(x_val, y_val), nb_epoch=20, batch_size=128)
```

# Complex Convolutional Neural Network

New Layers



# Convolutional Neural Net

```
Trainable params: 1,667,869

Train on 2050 samples, validate on 512 samples
...
Epoch 17/20
2050/2050 [====================] - 9s - loss: 0.0673 - acc: 0.9815 - val_loss: 1.0135 - val_acc: 0.7266
Epoch 18/20
2050/2050 [====================] - 9s - loss: 0.0937 - acc: 0.9717 - val_loss: 0.9287 - val_acc: 0.7344
Epoch 19/20
2050/2050 [====================] - 9s - loss: 0.0724 - acc: 0.9751 - val_loss: 1.1462 - val_acc: 0.7285
...
```

# Recap

- Set a goal
- Start simple
- Incremental complexity
- Model improvement
  - 66% -> 59% -> 68% -> 72%

# Next Steps

- Different architectures
  - Different algorithms
  - Recurrent NN
  - Character Level CNN
- Data
  - Pseudo-labelling
  - New articles
  - Alternate author texts (tweets?)

# Next Steps

Apply to different domains

- Product classification
- Sentiment Analysis
- Article tagging
- Images, Image recognition

# Thanks!

References:


- http://course.fast.ai
- https://quid.com/feed/how-quid-uses-deep-learning-with-small-data
- Convolutional Neural Networks for Sentence Classification, Yoon Kim (2014)