

BST 261: Data Science II

Lecture 3

**Feedforward networks in
Python with Keras, Regularization**

Heather Mattie
Harvard T.H. Chan School of Public Health
Spring 2 2020

Recipe of the day

Black Forest Cake



Fancy Black Forest Cake



Paper Presentations

ImageNet: A Large-Scale Hierarchical Image Database (2009)

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li and Li Fei-Fei Dept. of Computer Science,
Princeton University, USA

Presented by Daniel Waranch

Importance

- Image classification is a foundational tool for deep learning
- This paper presents a few cool techniques that are easy to understand but very powerful

Ideas of the paper

- Nouns can be classified by synonyms, which we call a synset. For example, person and human can be thought of as one synset (one category for classification.)
- If we had 500-1000 high resolution, clean images for each synset, in theory we could fully determine all nouns in the English language via images.
- This information could provide powerful training and benchmarking data for researchers and algorithms.
- ImageNet seeks to create this database of images

Structure of ImageNet

- ImageNet is named after WordNet which has 80,000 sysnets that ImageNet hopes to populate. At the time of the paper they had 5247 sysnets consisting of 3.2 million images. Today it's 14 million images documenting 21841 sysnets.
- Hierarchical, “Is-A” relationships. “tree based” structure.



- Candidate images are found via Internet search (10% accuracy) and then labelled by humans with Mechanical Turk. Generally speaking, the lower on the hierarchy tree, human accuracy is lower (dog. Vs. husky). Statistical methods are used to achieve high level of confidence for human labelling, leading to 99.7% precision.

Comparison with other image databases

- The authors compare this database to other images, and claim that ImageNet is superior due to the following properties:
- Disambiguated Labels: (can tell the difference between a river bank and a financial bank. Largely due to tree structure.)
- Clean Annotations
- Dense Hierarchies (much more subtrees of categories than other databases.)
- Full res. Images
- Publicly available
- ImageNet does not have segmented labelling (can label a tree and a dog in a single image.) There are databases that do this, but they are much smaller.

3 Applications/tests for ImageNet

- Tree based classification: This technique serves to show the importance of the tree based structure of the database.
- Demonstrative example.

(1.) I ask the algorithm
“Is this picture a
mammal?” I’m 60%
sure it is a Mammal.

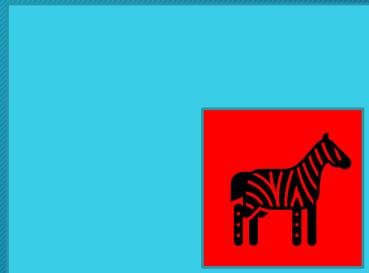
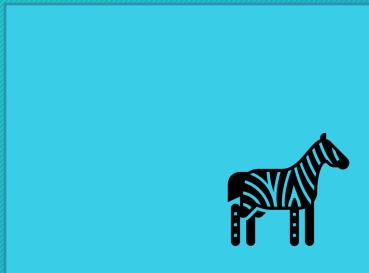
(2.) Ask the algorithm “Is
the **same** picture a dog?”
I am 80% sure it is a dog.

(3.) Because this is a
hierarchical tree
database, we actually
know that true answer to
(1.) is actually 80%.

- Because the algorithm is dense and tree based, we can improve our classification quite easily via checking all the sub-nodes of the tree. If a sub-node provides better performance than we can just use that instead!

3 Applications/tests for ImageNet

- Object Localization: Technique to get a more “obvious” image. For example, consider a 1000x1000 pixel image that has a zebra in it. Perhaps the algorithm give it a 95% chance of having a zebra. We can consider many possible “sub-rectangles” of the 1000x1000 image. We feed the sub-rectangles into the algorithm and find the one that gives the highest percent chance to be a zebra, lets say 99%. This “localizes” onto the zebra.



3 Applications/tests for ImageNet

- Object recognition: This is the most “obvious” application. The authors perform 4 tests. 2 with an algorithm with noisy vs. clean data to show the advantage of clean data. And 2 with 100 images per category of training vs. the full category for training to show the importance of lots of images. Results are as we would expect, with the clean data performing better than noisy data, and the full category performing better than the partial category. Furthermore they extract features from the high resolution data to show that high res data performs better than low res data. Performance is plotted by ROC curves.

Conclusion

- ImageNet is a great resource from training and testing image classification algorithms.
- It has many properties that make it useful.
- There really aren't *that* many nouns in the world, so classifying them with images feels like an approachable task.
- We have seen two cool methods that is allowed with the structure of ImageNet.

ImageNet Classification with Deep Convolutional Neural Networks

AUTHORS: ALEX KRIZHEVSKY, ILYA SUTSKEVER, GEOFFREY E. HINTON
(2012)

PRESENTATION: GENEVIEVE LYONS



Goal: To improve performance of object-recognition ML (address common barriers)

❖ Collect larger datasets

❖ More powerful models

❖ Preventing overfitting

❖ Execution time

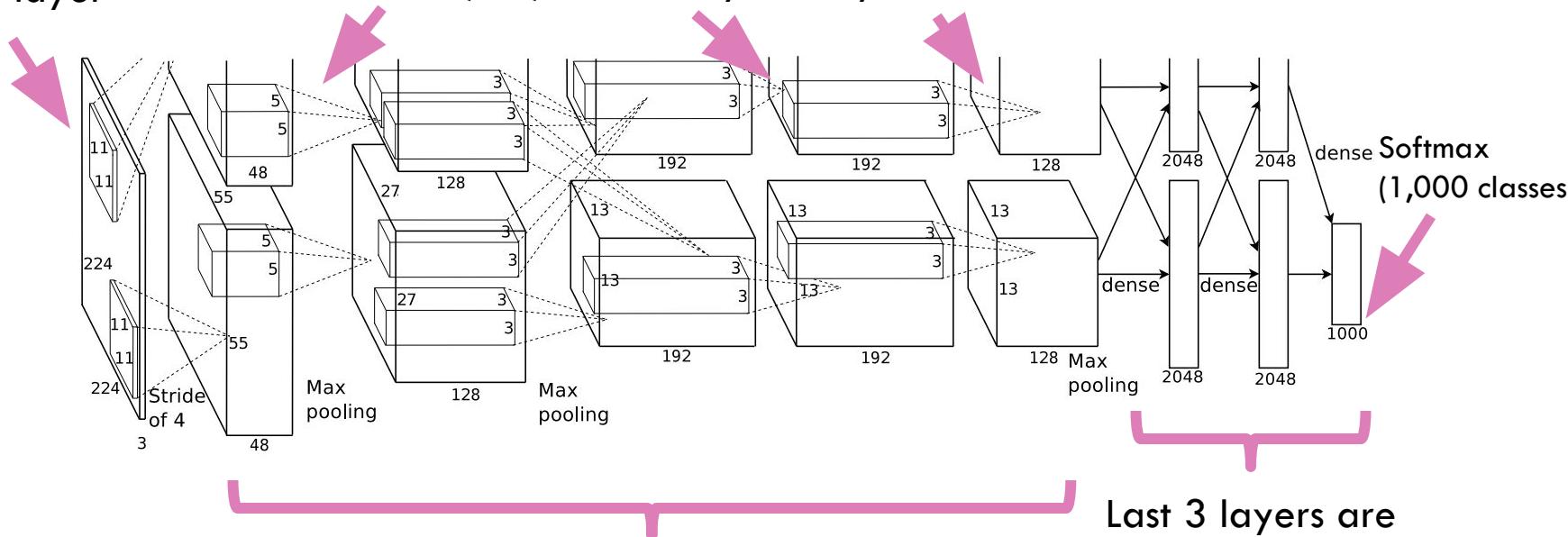
Goal: To improve performance of object-recognition ML

- ❖ Collect larger datasets
 - ❖ ImageNet: > 15M labeled high-resolution images in > 22,000 categories
- ❖ More powerful models
 - ❖ CNNs: Appropriate for images (locality of pixels), fewer connections, fewer parameters
- ❖ Preventing overfitting
 - ❖ Augmenting Data, Dropout layers
- ❖ Execution time
 - ❖ Highly optimized GPU implementation of 2D convolution and other operations (across 2 GPUs)

Results: One of the largest CNNs to date on ImageNet, with the best results ever reported on these datasets
(AlexNet)

Input layer

Kernels of 2nd, 4th, and 5th layers only connected to same GPU



First 5 layers are convolutional

Last 3 layers are
fully connected

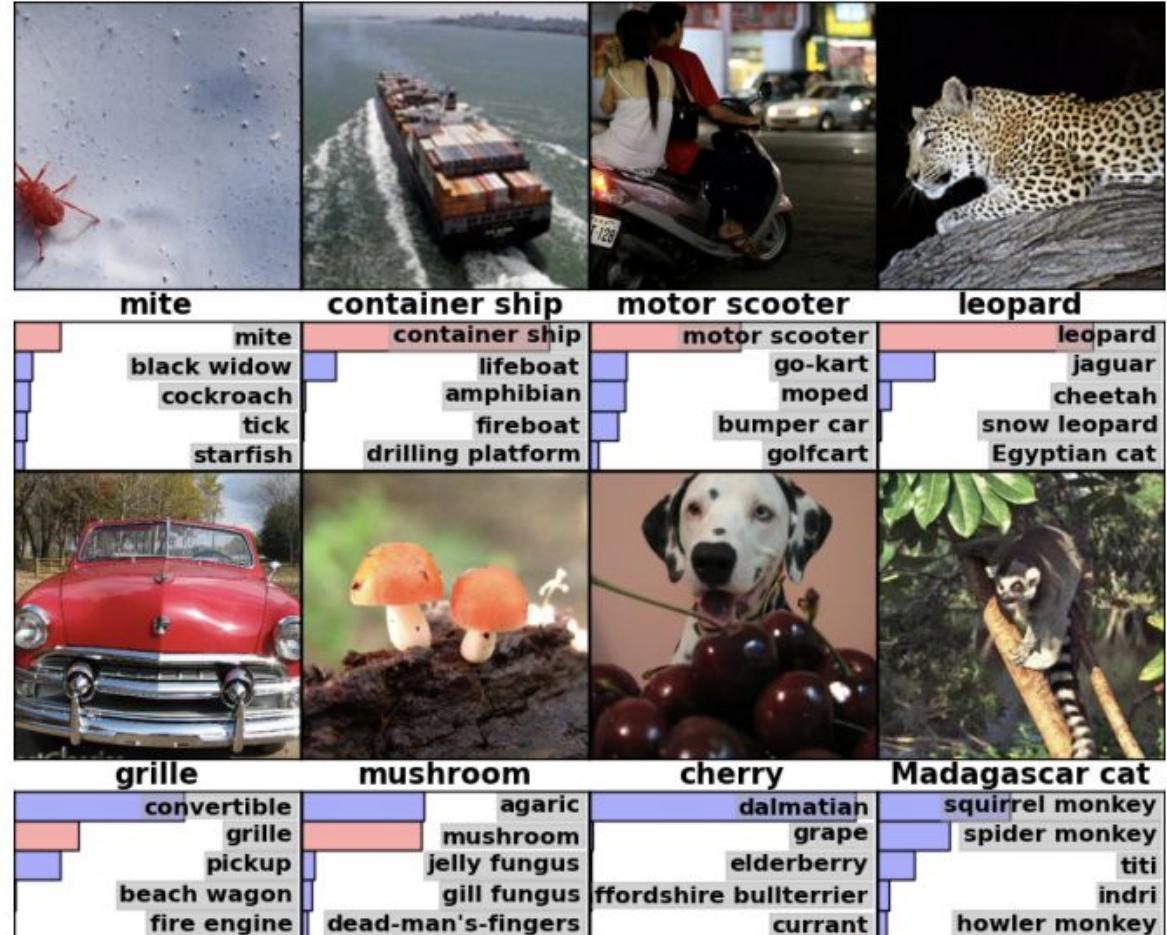
60 million parameters!

Reducing Overfitting:

- ❖ Data Augmentation
 - ❖ Train network on randomly selected patches of images (increases training data size!)
 - ❖ Alter intensity of RGB channels through PCA
- ❖ Dropout
 - ❖ Randomly exclude neurons ($p = 0.5$) for each new input

Results

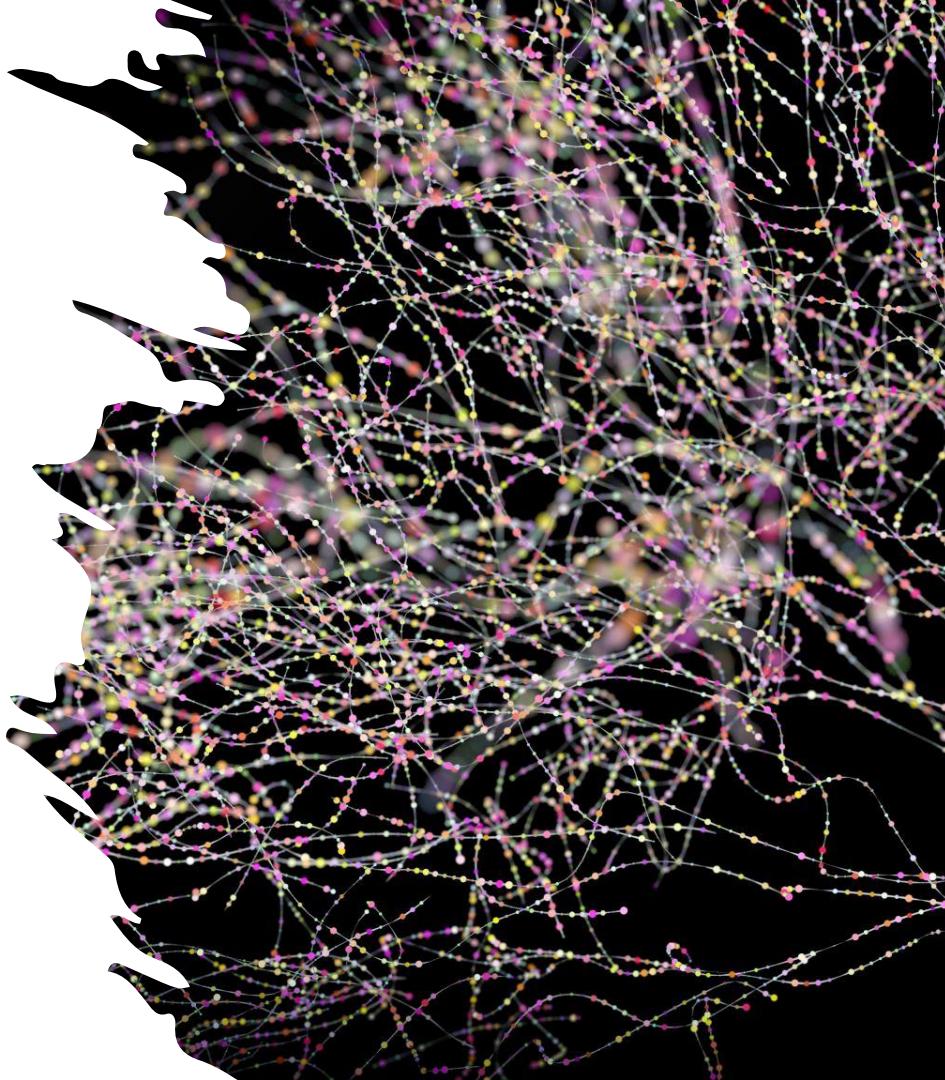
Model	Top-1	Top-5
<i>Sparse coding [2]</i>	47.1%	28.2%
<i>SIFT + FVs [24]</i>	45.7%	25.7%
CNN	37.5%	17.0%



Going Deeper with Convolutions
Szegedy et al.

PRESENTATION BY:

REBECCA YOUNGERMAN



Inception – A New Deep CNN

- Classification and Detection
 - ILSVRC 2014
- Width and Depth
- Computational Budget Constant
- Sparse Substructure

Min Lin^{1,2}, Qiang Chen², Shuiceng Yan²
¹Graduate School for Integrative Sciences and Engineering
²Department of Electronic & Computer Engineering
National University of Singapore, Singapore
{linmin,chenqiang,eleyans}@nus.edu.sg

Abstract

We propose a novel deep network structure called “Network In Network”(NIN) to enhance model discriminability for local patches within the receptive field. The conventional convolutional layer uses linear filters followed by a nonlinear activation function to scan the input. Instead, we build micro neural networks with more complex structures to abstract the data within the receptive field. We instantiate the micro neural network with a multilayer perceptron, which is a potent function approximator. The feature maps are obtained by sliding the micro networks over the input in a similar manner as CNN; they are then fed into the next layer. Deep NIN can be implemented by stacking multiple of the above described structure. With enhanced local modeling via the micro network, we are able to utilize global average pooling over feature maps in the classification layer, which is easier to interpret and less prone to overfitting than traditional fully connected layers. We demonstrated the state-of-the-art classification performances with NIN on CIFAR-10 and CIFAR-100, and reasonable performances on SVHN and MNIST datasets.

+



Save the computation – dimension reduction

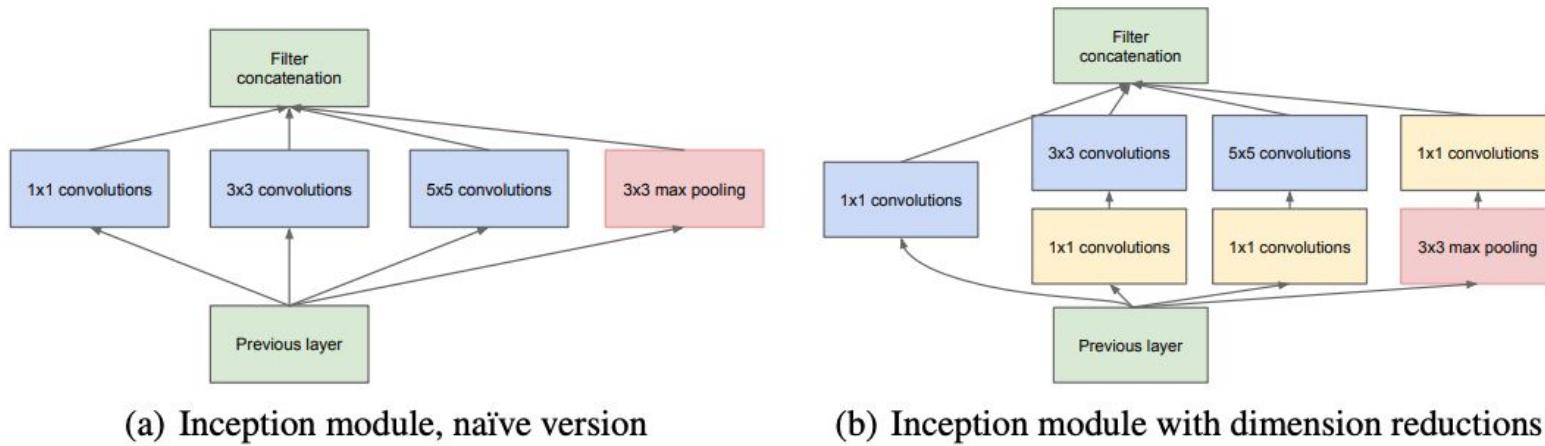


Figure 2: Inception module

GoogLeNet – Better than everyone else's LeNet

- ReLU!
- Computational efficiency and practicality
- Auxiliary networks for backpropagation concerns

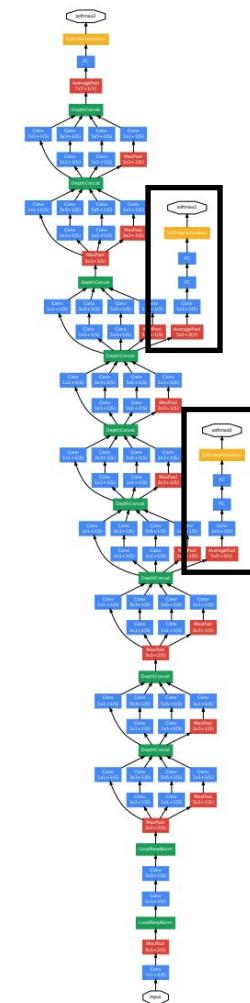
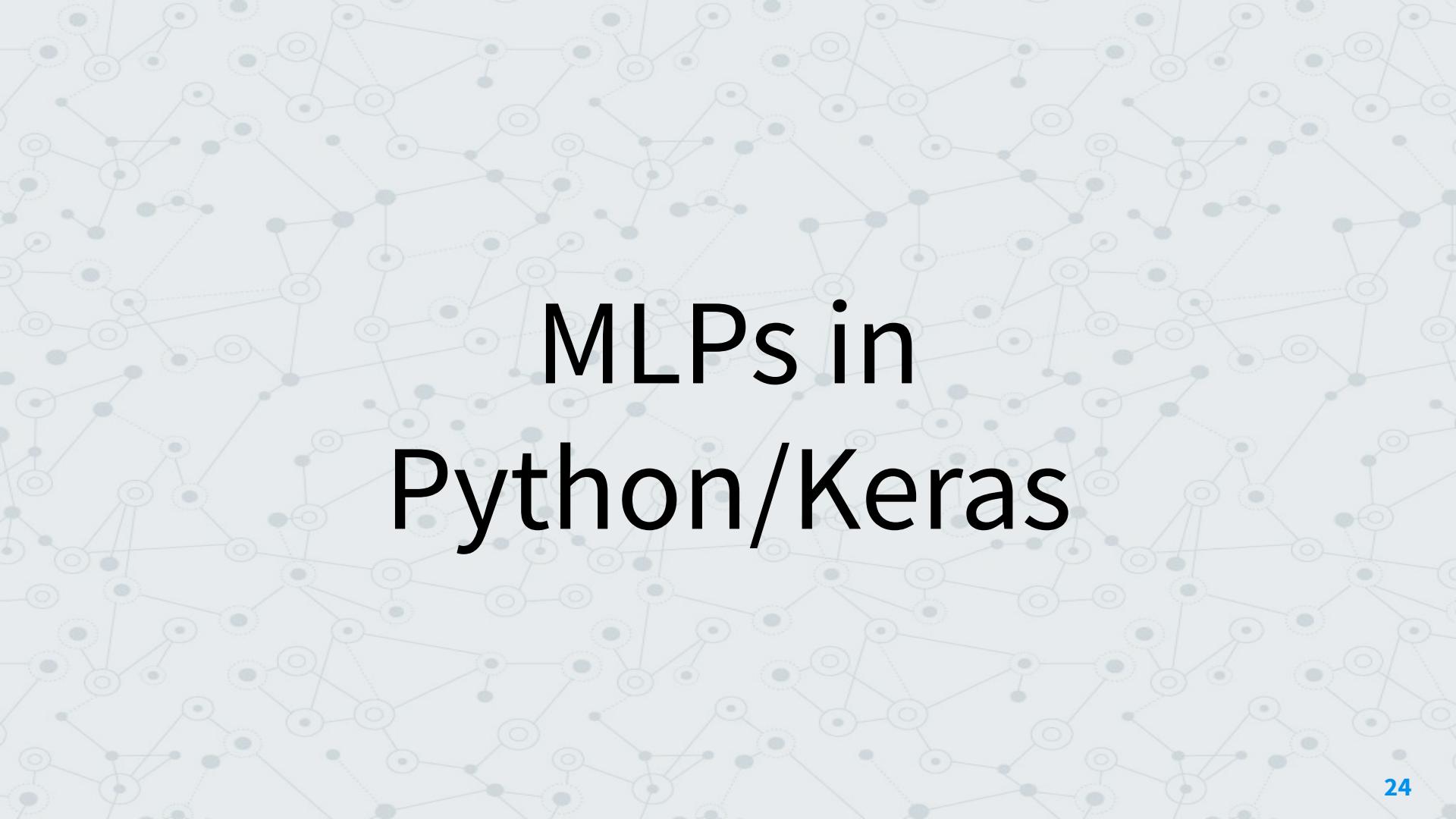


Figure 3: GoogLeNet network with all the bells and whistles

References

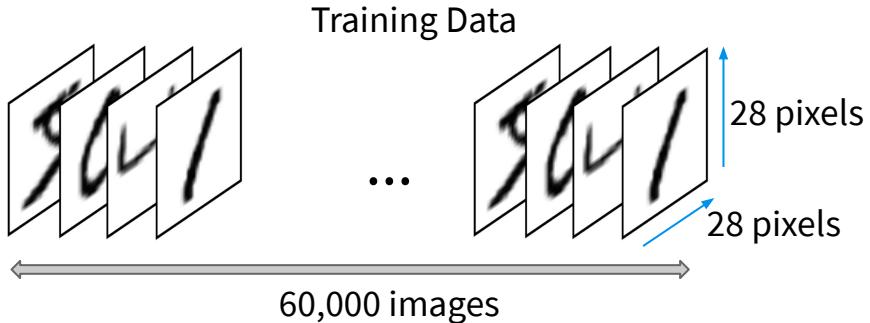
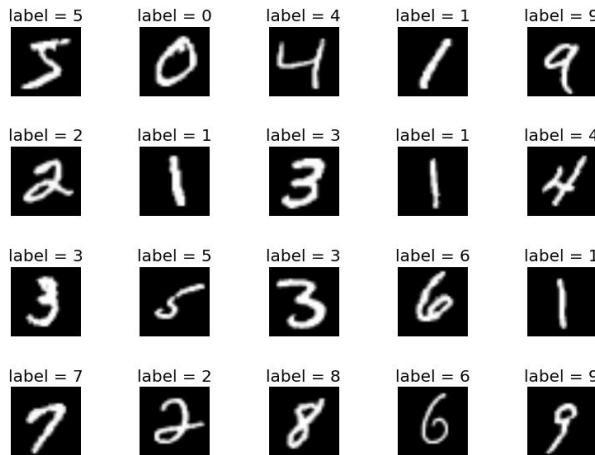
- Going Deeper with Convolutions:
<https://arxiv.org/pdf/1409.4842.pdf>
- Inception Meme:
<http://knowyourmeme.com/memes/we-need-to-go-deeper>

A light gray background featuring a complex, dense network graph composed of numerous small, semi-transparent nodes and connecting lines.

MLPs in Python/Keras

MNIST Data Example

- The [MNIST data set](#) includes handwritten digits with corresponding labels
- Training set: 60,000 images of handwritten digits and corresponding labels
 - Each digit is represented as a 28×28 matrix of grayscale values 0 - 255
 - The entire training set is stored in a 3D tensor of shape $(60000, 28, 28)$
 - The corresponding image values are stored as a 1D tensor of values 0 - 9
- Testing set: 10,000 images with the same set up as the training set



MNIST Data Example

Data wrangling

- ◎ We'll get into RGB images later, but for grayscale images, we need to first transform the matrix of values into a vector of values, and then normalize them to be between 0 and 1. It is not strictly necessary to normalize your inputs, but smaller numbers help speed up training and avoid getting stuck in local minima. This also ensures the gradients don't "explode" or "vanish"
 - Reshape each image from a 28×28 matrix of grayscale values 0 - 255 to a vector of length $28 \times 28 = 784$ of values 0 - 1 (divide each by 255)
- ◎ We now have 10 classes (categories; the digits 0-9)
 - We need to have multiclass labels that tell the network which digit the example is
 - Reshape each corresponding image label to a vector of length 10 of values 0 or 1
 - Example: the digit 3 would be represented as [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
 - You can think of this as "dummy coding" the labels

Activation and Loss Function Choices

Task	Last-layer activation	Loss function
Binary classification	sigmoid	Binary cross-entropy
Multiclass, single-label classification	softmax	Categorical cross-entropy
Multiclass, multilabel classification	sigmoid	Binary cross-entropy
Regression to arbitrary values	None	Mean square error (MSE)
Regression to values between 0 and 1	sigmoid	MSE or binary cross-entropy

Softmax function

$$\text{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

- ◎ Softmax units are used as outputs when predicting a discrete variable y with k possible values
- ◎ In this setting, which can be seen as a generalization of the Bernoulli distribution, we need to produce a vector $\hat{\mathbf{y}}$ with $\hat{y}_i = P(y = i|x)$
- ◎ We require that each \hat{y}_i lie in the $[0, 1]$ interval and that the entire vector sums to 1
- ◎ We first compute $z = w^T x + b$ as usual
- ◎ Here, $z_i = \log[\tilde{P}(y = i|x)]$ represents an unnormalized log probability for class i

The softmax function then exponentiates and normalizes z to obtain $\hat{\mathbf{y}}$

Softmax function

- ◎ In this case we want to maximize

$$\log[P(y = i; z)] = \log[\text{softmax}(z)_i] = z_i - \log \sum_j \exp(z_j)$$

- ◎ The first term shows that the input always has a direct contribution to the loss function
- ◎ Because $\log \sum_j \exp(z_j) \approx \max_j z_j$, the negative log-likelihood loss function always strongly penalizes the most active incorrect prediction

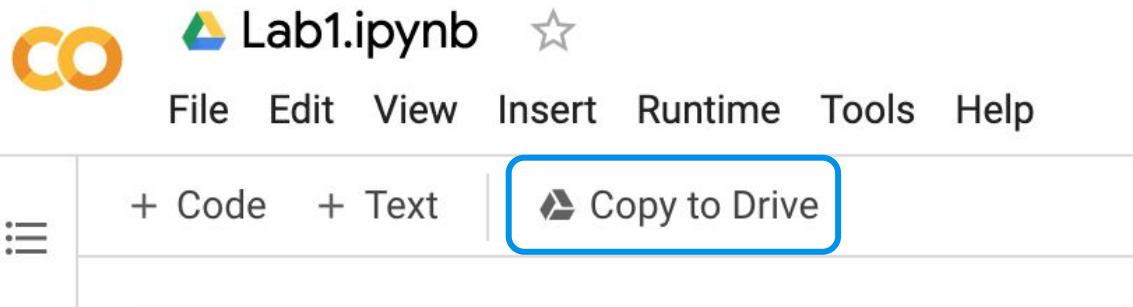
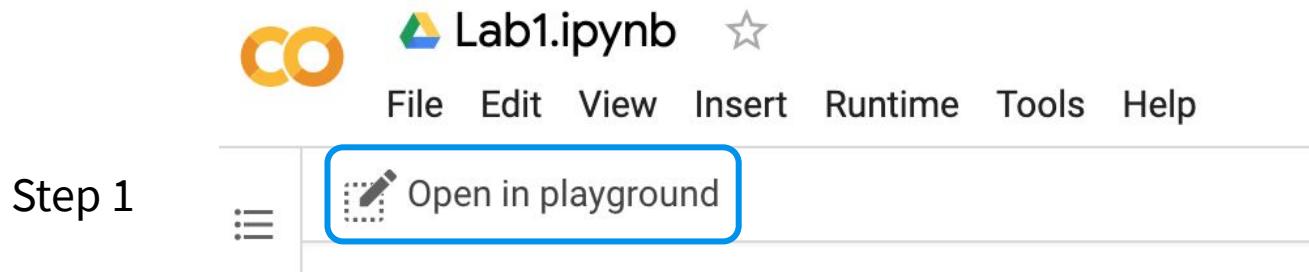
MNIST Data Example

Network Architecture

- ◎ Let's start with 2 layers:
 - Hidden layer will have 512 hidden units and the **relu activation function**
 - Output layer with 10 units (one for each possible digit) and the **softmax activation function** (this produces a vector of length 10, where each element is a probability between 0 and 1 of the image being classified as that digit)
 - Example: [0, 0.3, 0, 0, 0, 0, 0, 0.7, 0, 0] - the highest probability corresponds to a label of 7, so the network would classify this image as a 7
 - **rmsprop optimization algorithm**
categorical_crossentropy loss function
accuracy performance measure (the proportion of times the correct class is chosen)

MNIST Data Example

[Colab link](#)



IMDb Data Example



The [IMDb data set](#) is a set of movie reviews that have been labeled as either positive or negative, based on the text content of the reviews

- ◎ Training set: 25,000 either positive or negative movie reviews that have each been turned into a vector of integers
 - We'll see how to actually do this later in the course
 - Each review can be of any length
 - Only the top 10,000 most frequently occurring words are kept i.e. rare words are discarded
 - Each review includes a label: 0 = negative review and 1 = positive review

- ◎ Testing set: 25,000 either positive or negative movie reviews, similar to the training set

IMDb Data Example

Data Wrangling

- ◎ Each review is of a varying length and is a list of integers - we need to turn this into a tensor with a common length for each review
- ◎ Create a 2D tensor of shape 25,000 x 10,000
 - 25,000 reviews and 10,000 possible words
- ◎ Use the **vectorize_sequences** function to turn a movie review list of integers into a vector of length 10,000 with 1s for each word that appears in the review and 0s for words that do not
- ◎ The labels are already 0s and 1s, so the only thing we need to do is make them float numbers

Activation and Loss Function Choices

Task	Last-layer activation	Loss function
Binary classification	sigmoid	Binary cross-entropy
Multiclass, single-label classification	softmax	Categorical cross-entropy
Multiclass, multilabel classification	sigmoid	Binary cross-entropy
Regression to arbitrary values	None	Mean square error (MSE)
Regression to values between 0 and 1	sigmoid	MSE or binary cross-entropy

IMDb Data Example

Network Architecture

- 3 layers
 - 2 hidden layers and 1 output layer
 - Hidden layers have 16 hidden units each and a **relu activation function**
 - Output layer has 1 unit (the probability a review is positive)
- **Sigmoid activation function**
- **rmsprop optimization algorithm**
- **binary_crossentropy loss function**
- **accuracy performance measure** (proportion of times the correct class is chosen)

IMDb Data Example

[Colab link](#)

Regularization

Regularization

- One of the biggest problems with neural networks is overfitting.
- Regularization schemes combat overfitting in a variety of different ways

A perceptron represents the following optimization problem:

$$\operatorname{argmin}_W l(y, f(X)) \text{ where } f(X) = \frac{1}{1+\exp(-\phi(XW))}$$

Regularization

One way to regularize is to introduce penalties and change

$$\operatorname{argmin}_W l(y, f(X))$$

to

$$\operatorname{argmin}_W l(y, f(X)) + \lambda R(W)$$

where $R(W)$ is often the L1 or L2 norm of W . These are the well-known ridge and LASSO penalties, and referred to as **weight decay** by the neural net community.

L2 Regularization

We can limit the size of the L2 norm of the weight vector:

$$\operatorname{argmin}_W l(y, f(X)) + \lambda \|W\|_2$$

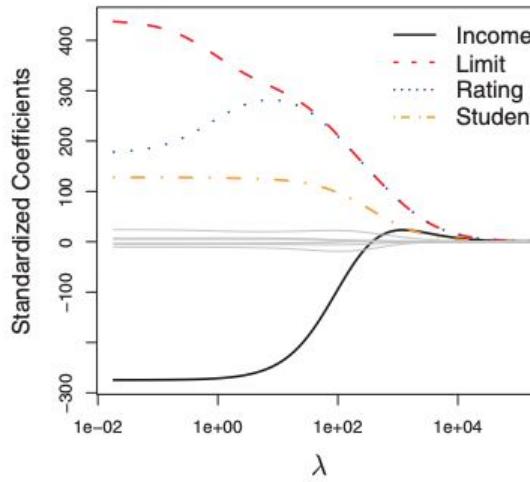
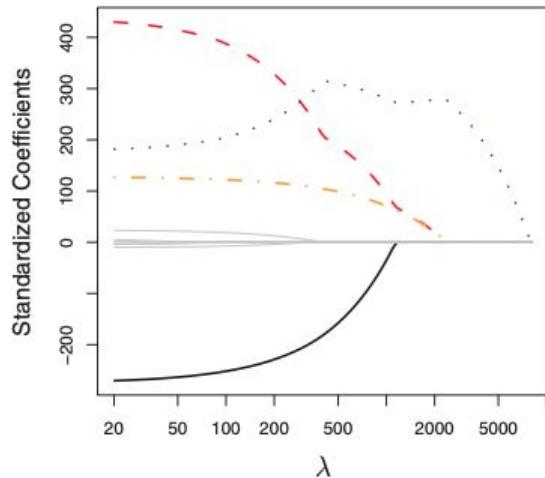
where

$$\|W\|_2 = \sum_{j=1}^p w_j^2$$

We can do the same for the L1 norm. What do the penalties do?

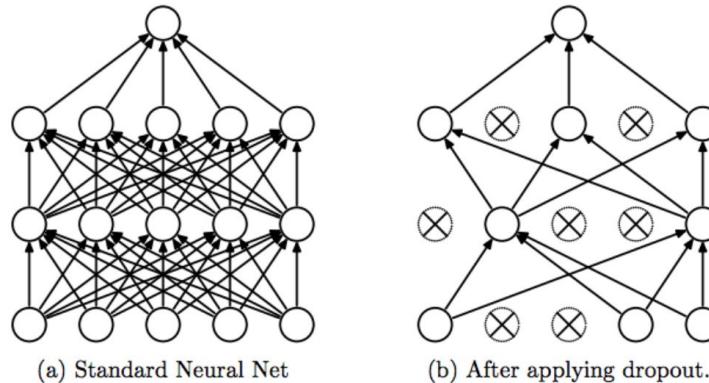
Shrinkage

The L1 and L2 penalties shrink the weights to or towards 0.



Stochastic Regularization

- Why is this a good idea?
- One of the most popular ways to do this is **dropout**
- Given a hidden layer, we are going to set each element of the hidden layer to 0 with probability p each SGD update.



Stochastic Regularization

- One way to think of this is the network is trained by bagged versions of the network.
- Bagging** reduces variance.
- Others have argued this is an approximate Bayesian model

Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning

Yarin Gal
Zoubin Ghahramani
University of Cambridge

YG279@CAM.AC.UK
ZG201@CAM.AC.UK

Abstract

Deep learning tools have gained tremendous attention in applied machine learning. However such tools for regression and classification do not capture model uncertainty. In comparison, Bayesian models offer a mathematically grounded framework to reason about model uncertainty, but usually come with a prohibitive computational cost. In this paper we develop a new theoretical framework casting dropout training in deep neural networks (NNs) as approximate Bayesian inference in deep Gaussian processes. A direct result of this theory gives us tools to model uncertainty with dropout NNs – extracting information from existing models that has been thrown away so far. This mitigates the problem of representing uncertainty in deep

With the recent shift in many of these fields towards the use of Bayesian uncertainty (Herzog & Ostwald, 2013; Trafimow & Marks, 2015; Nuzzo, 2014), new needs arise from deep learning tools.

Standard deep learning tools for regression and classification do not capture model uncertainty. In classification, predictive probabilities obtained at the end of the pipeline (the softmax output) are often erroneously interpreted as model confidence. A model can be uncertain in its predictions even with a high softmax output (fig. 1). Passing a point estimate of a function (solid line 1a) through a softmax (solid line 1b) results in extrapolations with unjustified high confidence for points far from the training data. x^* for example would be classified as class 1 with probability 1. However, passing the distribution (shaded area 1a) through a softmax (shaded area 1b) better reflects classification uncertainty far from the training data.

Stochastic Regularization

- Many have argued that SGD itself provides regularization

Journal of Machine Learning Research 18 (2017) 1-35

Submitted 4/17; Revised 10/17; Published 12/17

Stochastic Gradient Descent as Approximate Bayesian Inference

Stephan Mandt
Data Science Institute
Department of Computer Science
Columbia University
New York, NY 10025, USA

STEPHAN.MANDT@GMAIL.COM

SelectorGadget
Has access to this site

Matthew D. Hoffman
Adobe Research
Adobe Systems Incorporated
601 Townsend Street
San Francisco, CA 94103, USA

MATHOFFM@ADOBECOM

David M. Blei
Department of Statistics
Department of Computer Science
Columbia University
New York, NY 10025, USA

DAVID.BLEI@COLUMBIA.EDU

Editor: Manfred Opper

Abstract

Stochastic Gradient Descent with a constant learning rate (constant SGD) simulates a Markov chain with a stationary distribution. With this perspective, we derive several new results. (1) We show that constant SGD can be used as an approximate Bayesian posterior inference algorithm. Specifically, we show how to adjust the tuning parameters of constant SGD to best match the stationary distribution to a posterior, minimizing the Kullback-Leibler divergence between these two distri-

Initialization Regularization

- ◎ The weights in a neural network are given random values initially.
- ◎ There is an entire literature on the best way to do this initialization
 - Normal
 - Truncated Normal
 - Uniform
 - Orthogonal
 - Scaled by number of connections
 - Etc.
- ◎ Try to “bias” the model into initial configurations that are easier to train

Initialization Regularization

- ◎ A popular way is to do **transfer learning**

Train the model on auxiliary task
where lots of data is available



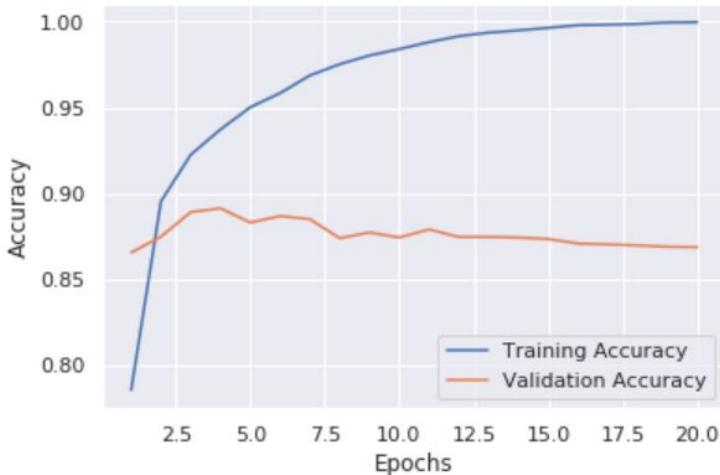
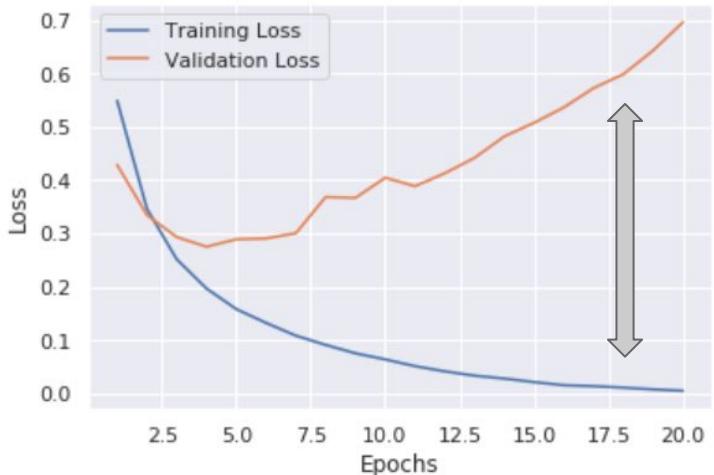
Use final weight values from previous
task as initial values and “fine tune”
on primary task

Structural Initialization

- ◎ The key advantage of neural nets is the ability to easily include properties of the data directly into the model through the network's structure
- ◎ Convolutional neural networks (CNNs) are a prime example of this

IMDb Example

We saw overfitting in the IMDb example:



How do we make this model better?

Regularization

1. Reduce network size
2. Weight regularization
3. Dropout

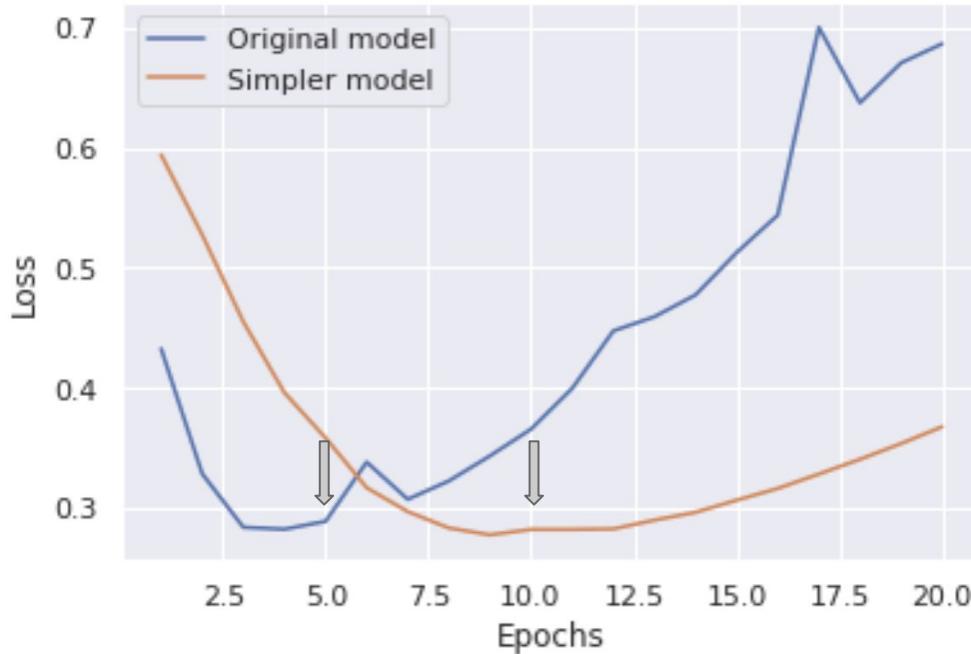
[Back to the IMDb colab](#)

Regularization: reducing network size

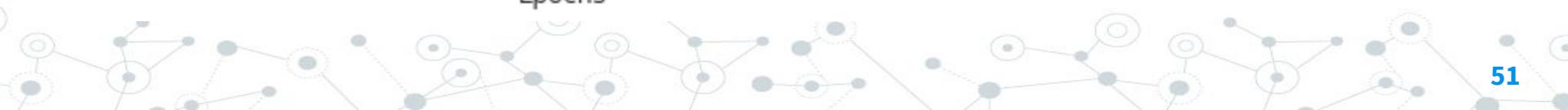
When we are battling overfitting, one option is to simplify the model. Let's compare the performance we get from a simpler model. Here we have simplified the model by reducing the number of hidden units in each hidden layer.

```
1 # Original model
2 model = tf.keras.models.Sequential([
3     tf.keras.layers.Dense(16, activation='relu'),
4     tf.keras.layers.Dense(16, activation='relu'),
5     tf.keras.layers.Dense(1, activation='sigmoid')
6 ])
7
8 # Reduced model
9 model2 = tf.keras.models.Sequential([
10    tf.keras.layers.Dense(4, activation='relu'),
11    tf.keras.layers.Dense(4, activation='relu'),
12    tf.keras.layers.Dense(1, activation='sigmoid')
13 ])
```

Regularization: reducing network size



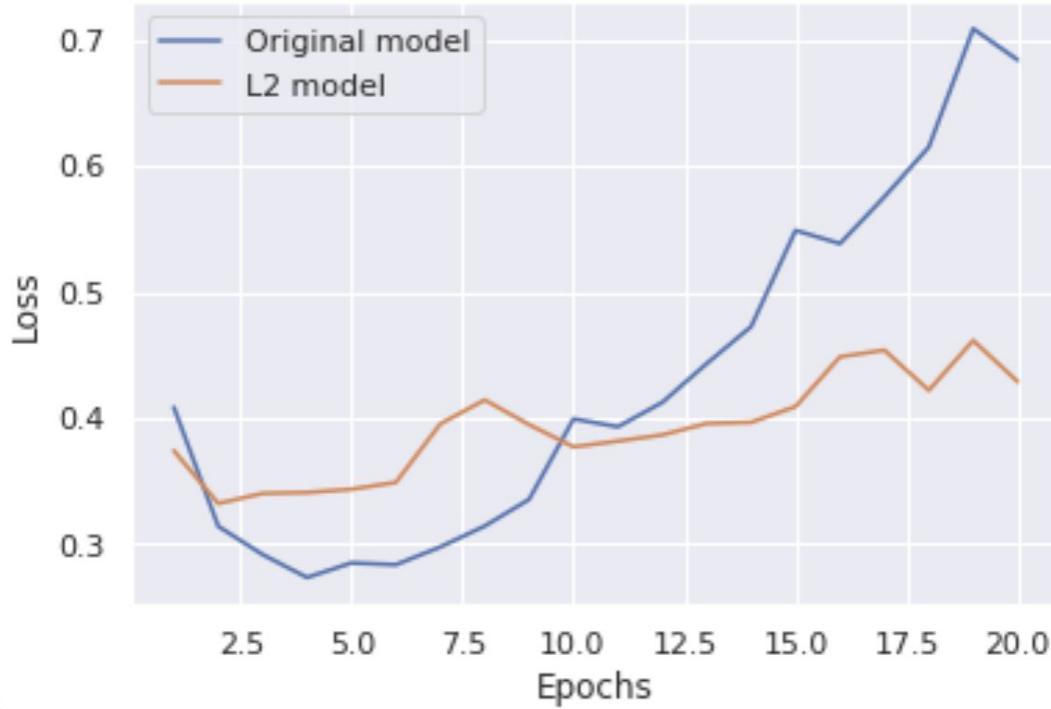
The smaller network performs better than the original model - it starts to overfit at epoch 10 rather than epoch 6. These values are when the validation loss starts to increase.



Regularization: weight regularization

```
1 # L2 model
2 l2_model = tf.keras.models.Sequential([
3     # Layer 1 (Hidden layer)
4     tf.keras.layers.Dense(16, activation='relu',
5                           kernel_regularizer = tf.keras.regularizers.l2(0.001)), ←
6     # Layer 2 (Hidden layer)
7     tf.keras.layers.Dense(16, activation='relu',
8                           kernel_regularizer = tf.keras.regularizers.l2(0.001)), ←
9     # Layer 3 (Output layer)
10    tf.keras.layers.Dense(1, activation='sigmoid')
11 ])
12
13 # Define how to execute training
14 l2_model.compile(optimizer='rmsprop',
15                     loss='binary_crossentropy',
16                     metrics=['accuracy'])
```

Regularization: weight regularization



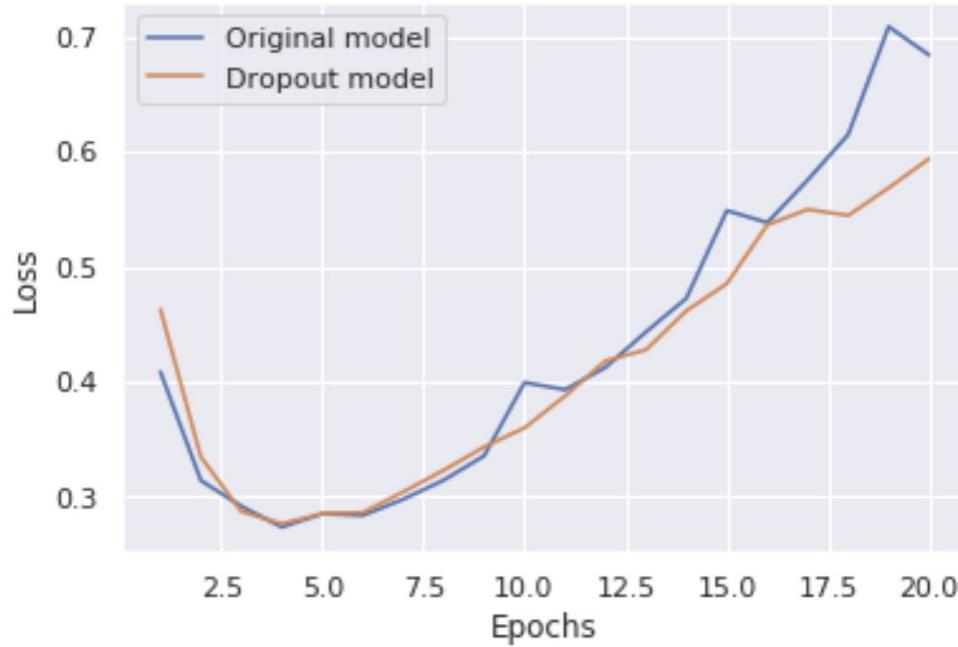
The L2-regularized model is much more resistant to overfitting
- the validation loss starts to increase at a much slower rate

Regularization: adding dropout

```
1 # Dropout model
2 dmodel = tf.keras.models.Sequential([
3     # Layer 1 (Hidden layer)
4     tf.keras.layers.Dense(16, activation='relu'),
5     # Dropout layer
6     tf.keras.layers.Dropout(0.5), ←
7     # Layer 2 (Hidden layer)
8     tf.keras.layers.Dense(16, activation='relu'),
9     # Dropout layer
10    tf.keras.layers.Dropout(0.5), ←
11    # Layer 3 (Output layer)
12    tf.keras.layers.Dense(1, activation='sigmoid')
13 ])
14
15 # Define how to execute training
16 dmodel.compile(optimizer='rmsprop',
17                  loss='binary_crossentropy',
18                  metrics=[ 'accuracy' ])
```

The 0.5 indicates a 50% probability of dropping out a unit. Typically, 20% is used in practice but you can try different values and see what performs best.

Regularization: adding dropout



The dropout model is slightly better than the original model but does not control for overfitting as well as the L2 network

